Developer's Notes

Siconos Development Team

September 16, 2011

Contents

Chapter 1

OneStepNSProblem formalisation for several interactions

author	F. Pérignon
date	May 16, 2006
version	?

1.1 LinearDS - Linear Time Invariant Relations

1.1.1 General notations

We consider n dynamical systems of the form:

$$\dot{x}_i = A_i x_i + R_i \tag{1.1}$$

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:

$$y_\alpha = C_\alpha X_\alpha + D_\alpha \lambda_\alpha \tag{1.2}$$

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.

$$C_\alpha = \begin{bmatrix} C_\alpha^i & C_\alpha^j & \dots \end{bmatrix} \tag{1.3}$$

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

$$\begin{bmatrix} R_\alpha^i \\ R_\alpha^j \\ \dots \end{bmatrix} = B_\alpha \lambda_\alpha = \begin{bmatrix} B_\alpha^i \\ B_\alpha^j \\ \dots \end{bmatrix} \lambda_\alpha \tag{1.4}$$

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

$$R_i = \sum_{\beta \in \mathcal{I}_i} R_\beta^i = \sum_{\beta \in \mathcal{I}_i} B_\beta^i \lambda_\beta \tag{1.5}$$

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

$$x_i^{k+1} - x_i^k = hA_i x_i^{k+1} + hR_i^{k+1} \tag{1.6}$$

$$y_\alpha^{k+1} = C_\alpha X_\alpha^{k+1} + D_\alpha \lambda_\alpha^{k+1} \tag{1.7}$$

$$R_i^{k+1} = \sum_{\beta \in \mathcal{I}_i} B_\beta^i \lambda_\beta^{k+1} \tag{1.8}$$

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

$$x_i^{k+1} = W_i x_i^k + hW_i R_i^{k+1} \tag{1.9}$$

$$y_\alpha^{k+1} = C_\alpha W_\alpha X_\alpha^k + C_\alpha h W_\alpha \sum_{\beta \in \mathcal{I}_i} B_\beta^i \lambda_\beta^{k+1} + D_\alpha \lambda_\alpha^{k+1} \tag{1.10}$$

$$= C_\alpha W_\alpha X_\alpha^k + (C_\alpha h W_\alpha B_\alpha + D_\alpha)\lambda_\alpha^{k+1} + \sum_{\beta \neq \alpha} \Big(\sum_{i \in \mathcal{DS}_\alpha \cap \in \mathcal{DS}_\beta} h C_\alpha^i W_i B_\beta^i \lambda_\beta^{k+1} \Big) \tag{1.11}$$

with

$$W_\alpha = \begin{bmatrix} W_i & 0 & \dots \\ 0 & W_j & \dots \\ 0 & \dots & \dots \end{bmatrix} \tag{1.12}$$ {Walpha}

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:

$$Y_{k+1} = q_{OSNSP} + M_{OSNSP} \Lambda_{k+1} \tag{1.13}$$

or,

$$Y_{k+1} = \begin{bmatrix} y_1 \\ \dots \\ y_m \end{bmatrix}_{k+1} = \begin{bmatrix} C_1^1 & \dots & C_1^n \\ \vdots & \dots & \vdots \\ C_m^1 & \dots & C_m^n \end{bmatrix} \begin{bmatrix} W_1 & 0 & \dots & 0 \\ 0 & W_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ & & 0 & W_n \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ \vdots \\ x_n \end{bmatrix}_k \tag{1.14}$$

$$+ \begin{bmatrix} D_1 + h\sum_{j \in \mathcal{DS}_1} C_1^j W_j B_1^j & h \sum_{j \in \mathcal{DS}_1 \cap \mathcal{DS}_2} C_1^j W_j B_2^j & \dots \\ \vdots & \ddots & \\ & h \sum_{j \in \mathcal{DS}_m} C_m^j W_j B_{m-1}^j & D_m + h \sum_{j \in \mathcal{DS}_m \cap \mathcal{DS}_{m-1}} C_m^j W_j B_m^j \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_m \end{bmatrix}_{k+1}$$

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

$$D_\alpha + h \sum_{j \in \mathcal{DS}_\alpha} C_\alpha^j W_j B_\alpha^j \tag{1.15}$$

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

$$h \sum_{j \in \mathcal{DS}_\alpha \cap \mathcal{DS}_\beta} C_\alpha^j W_j B_\beta^j \tag{1.16}$$

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:

$$D_{\alpha,l} + h \sum_{j \in \mathcal{DS}_\alpha} C^j_{\alpha,l} W_j B^j_\alpha \tag{1.17}$$

for the diagonal, and

$$h \sum_{j \in \mathcal{DS}_\alpha \cap \mathcal{DS}_\beta} C^j_{\alpha,l} W_j B^j_\beta \tag{1.18}$$

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".

1.1.2 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:

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

$$y_1 = \begin{bmatrix} C^1_1 & C^3_1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_3 \end{bmatrix} + D_1 \lambda_1 \tag{1.19}$$

$$y_2 = \begin{bmatrix} C^2_2 & C^3_2 \end{bmatrix} \begin{bmatrix} x_2 \\ x_3 \end{bmatrix} + D_2 \lambda_2 \tag{1.20}$$

$$\begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} B^1_1 \lambda_1 \\ B^2_2 \lambda_2 \\ B^3_1 \lambda_1 + B^3_2 \lambda_2 \end{bmatrix} \tag{1.21}$$

$$M_{OSNSP} = \begin{bmatrix} D_1 + hC^1_1 W_1 B^1_1 + hC^3_1 W_3 B^3_1 & hC^3_1 W_3 B^3_2 \\ hC^3_2 W_3 B^3_1 & D_2 + hC^2_2 W_2 B^2_2 + hC^3_2 W_3 B^3_2 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix}_{k+1} \tag{1.22}$$

1.1.3 relative degree

Let us consider the global vector

$$Y = \begin{bmatrix} y_1 \\ ... \\ y_M \end{bmatrix} = CX + D\Lambda \tag{1.23}$$

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

$$y_j = \sum_{i=1}^{n} C_j^i x_i + D_{j,j}\lambda_j + \sum_{i\neq j, i=1}^{m} D_{j,i}\lambda_i \tag{1.24}$$

$D_{j,i}$ 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:

$$R_i = \sum_{k=1}^{M} B_k^i \lambda_j \tag{1.25}$$

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

$$\begin{aligned} \dot{y}_j &= \sum_{i=1}^{n} C_j^i (A_i x_i + \sum_{k=1}^{M} B_k^i \lambda_k) + f(\lambda_k)_{k\neq j} && (1.26) \\ &= \sum_{i=1}^{n} C_j^i (A_i x_i + B_j^i \lambda_j + \sum_{k=1, k\neq j}^{M} B_k^i \lambda_k) + \dots && (1.27) \end{aligned}$$

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:

$$coeff_j = \sum_{i=1}^{n} C_j^i (A_i)^{r-1} B_j^i \tag{1.28}$$

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

1.2 LagrangianDS - Lagrangian Linear Relations

1.2.1 General notations

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

$$M_i(q_i)\ddot{q}_i + N_i(\dot{q}_i, q_i) = F_{Int,i}(\dot{q}_i, q_i, t) + F_{Ext,i}(t) + p_i \tag{1.29}$$

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:

$$y_\alpha = H_\alpha Q_\alpha + b_\alpha \tag{1.30}$$

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.

$$H_\alpha = \left[\begin{array}{cccc} H_\alpha^i & H_\alpha^j & \dots \end{array} \right] \tag{1.31}$$

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

$$\left[\begin{array}{c} R_\alpha^i \\ R_\alpha^j \\ \dots \end{array} \right] = {}^tH_\alpha \lambda_\alpha = \left[\begin{array}{c} {}^tH_\alpha^i \\ {}^tH_\alpha^j \\ \dots \end{array} \right] \lambda_\alpha \tag{1.32}$$

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:

$$R_i = \sum_{\beta \in \mathcal{I}_i} R_\beta^i = \sum_{\beta \in \mathcal{I}_i} H_\beta^i \lambda_\beta \tag{1.33}$$

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

$$\begin{aligned} \dot{q}_i^{k+1} &= \dot{q}_{free,i} + W_i R_i^{k+1} \\ \dot{y}_\alpha^{k+1} &= H_\alpha \dot{Q}_\alpha^{k+1} \end{aligned} \tag{1.34}$$

$$R_i^{k+1} = \sum_{\beta \in \mathcal{I}_i} H_\beta^i \lambda_\beta^{k+1} \tag{1.35}$$

ie,

$$y_\alpha^{k+1} = H_\alpha Q_\alpha^{free} + H_\alpha W_\alpha {}^tH_\alpha \lambda_\alpha + \sum_{i \in \mathcal{DS}_\alpha} \sum_{\beta \in \mathcal{I}_i, \alpha \neq \beta} H_\alpha^i W_i H_\beta^j \lambda_\beta \tag{1.36}$$

with W_α given by (1.12).

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

$$Y_{k+1} = q_{OSNSP} + M_{OSNSP} \Lambda_{k+1} \tag{1.37}$$

with:

$$q_{OSNSP}^\alpha = H_\alpha Q_\alpha^{free} \tag{1.38}$$

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

$$\sum_{j \in \mathcal{DS}_\alpha} H_\alpha^j W_j {}^tH_\alpha^j \tag{1.39}$$

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

$$\sum_{j \in \mathcal{DS}_\alpha \cap \mathcal{DS}_\beta} H_\alpha^j W_j {}^tH_\beta^j \tag{1.40}$$

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:

$$\sum_{j \in \mathcal{DS}_\alpha} H^j_{\alpha,l} {W_j}^t H^j_\alpha \tag{1.41}$$

for the diagonal, and

$$\sum_{j \in \mathcal{DS}_\alpha \cap \mathcal{DS}_\beta} H^j_{\alpha,l} {W_j}^t H^j_\beta \tag{1.42}$$

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.

1.3 Block matrix approach

The built of the OSNSProblem matrix could be computed using block matrix structure. This section describe these matrices. It is not implemented in Siconos. Using previous notations, n is the number of DS. m the number of interations.

1.3.1 Block matrix of DS

$$\boldsymbol{M}\dot{\boldsymbol{X}} = \boldsymbol{A}\boldsymbol{X} + \boldsymbol{R}$$

where $\boldsymbol{M} = diag(M_1, ...M_n)$ and $\boldsymbol{A} = diag(A_1, .., A_n)$.

$$\boldsymbol{R} = \boldsymbol{B}\boldsymbol{\lambda}$$

$$\boldsymbol{B} = \begin{pmatrix} B^1_1 ... B^1_m \\ . \\ . \\ B^n_1 ... B^n_m \end{pmatrix}$$

Some of B^i_j doesn't exist.

1.3.2 Block matrix of interaction

$$\boldsymbol{Y} = \boldsymbol{C}\boldsymbol{X} + \boldsymbol{D}\boldsymbol{\lambda}$$

with $\boldsymbol{D} = diag(D_1..D_m)$ and

$$\boldsymbol{C} = \begin{pmatrix} C^1_1 .. C^n_1 \\ . \\ . \\ C^1_m ... C^n_m \end{pmatrix}$$

Some of C^i_j doesn't exist.

1.3.3 OSNSProblem using block matrices

The Matrix of the OSNS Problem could be assambled using the following block-product-matrices $\boldsymbol{CWB}$.

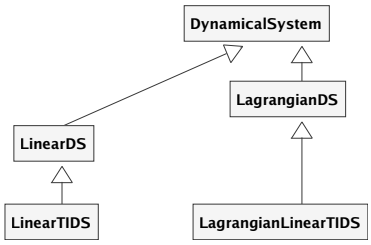
Chapter 2

Dynamical Systems formulations in Siconos.

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

2.1 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:



{DSDiagram}

2.2 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:

$$\dot{x} = f(x,t) + T(x)u(x,\dot{x},t) + r \quad (2.1)$$

$$x(t_0) = x_0 \quad (2.2)$$

- 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.

2.3 First order linear dynamical systems $\rightarrow$ class *LinearDS*

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

$$\dot{x} = A(t)x(t) + Tu(t) + b(t) + r \tag{2.3}$$
$$x(t_0) = x_0 \tag{2.4}$$

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:

$$f(x,t) = A(t)x(t) + b(t) \tag{2.5}$$
$$jacobianX = \nabla_x f(x,t) = A(t) \tag{2.6}$$

2.4 Second order non linear Lagrangian dynamical systems $\rightarrow$ class *LagrangianDS*

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

$$M(q)\ddot{q} + NNL(\dot{q}, q) + F_{Int}(\dot{q}, q, t) = F_{Ext}(t) + p \tag{2.7}$$
$$q(t_0) = q0 \tag{2.8}$$
$$\dot{q}(t_0) = velocity0 \tag{2.9}$$

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:

$$n = 2nDof \tag{2.10}$$

$$x = \begin{bmatrix} q \\ \dot{q} \end{bmatrix} \tag{2.11}$$

$$f(x,t) = \begin{bmatrix} \dot{q} \\ M^{-1}(F_{Ext} - F_{Int} - NNL) \end{bmatrix} \tag{2.12}$$

$$\tag{2.13}$$

$$\nabla_x f(x,t) = \begin{bmatrix} 0_{nDof \times nDof} & I_{nDof \times nDof} \\ \nabla_q(M^{-1})(F_{Ext} - F_{Int} - NNL) - M^{-1}\nabla_q(F_{Int} + NNL) & -M^{-1}\nabla_{\dot{q}}(F_{Int} + NNL) \end{bmatrix} \tag{2.14}$$

$$r = \begin{bmatrix} 0_{nDof} \\ p \end{bmatrix} \tag{2.15}$$

$$u(x, \dot{x}, t) = u_L(\dot{q}, q, t) \text{ (not yet implemented)} \tag{2.16}$$

$$T(x) = \begin{bmatrix} 0_{nDof} \\ T_L(q) \end{bmatrix} \text{ (not yet implemented)} \tag{2.17}$$

$$\tag{2.18}$$

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.

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

{Sec:LagrangianL

$$M\ddot{q} + C\dot{q} + Kq = F_{Ext}(t) + p \tag{2.19}$$

With:

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

And:

$$F_{Int} = C\dot{q} + Kq \tag{2.20}$$

$$NNL = 0_{nDof} \tag{2.21}$$

Chapter 3

Dynamical Systems implementation in Siconos.

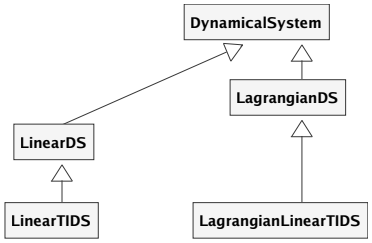
author	F. Pérignon
date	November 7, 2006
version	Kernel 1.3.0

3.1 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.

3.2 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:



{DSDiagram}

3.3 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 ...).

3.3.1 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.

3.3.2 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?

3.4 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 ...)

3.5 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 4

Interactions

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 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.

4.2 Class Diagram

4.3 Description

4.3.1 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 5

Notes on the Non Smooth Dynamical System construction

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

5.1 Introduction

5.2 Class Diagram

5.3 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).

5.4 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 6

OneStepIntegrator and derived classes.

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

6.1 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.

6.2 Class Diagram

6.3 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
- ...

6.4 Construction

Each constructor must:

-

6.4.1 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.

6.4.2 Lsodar

Required data:

Optional:

Always allocated in constructor:

Chapter 7

First Order Nonlinear Relation

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

Chapter 8

Computation of the number of Derivatives, index Set, levelMin and levelMax

author	V. Acary
date	Septembre 16, 2011
version	Kernel 3.3.0

In this chapter, we give some hints on the automatic computation of the number of index sets, the number of derivatives in the `Interaction` and the levelMin and LevelMax.

8.1 Why is the relative degree not relevant ?

In this section, we give a very brief overview of the notion of relative degree.

8.1.1 First order Linear complementary systems

A Linear Complementarity System (LCS) is defined by

$$\begin{cases} \dot{x} = Ax + B\lambda \\ y = Cx + D\lambda \\ 0 \leq y \perp \lambda \geq 0 \end{cases} \tag{8.1}$$ {eq:LCS-bis}

Definition 1 (Relative degree in the SISO case) *Let us consider a linear system in state representation given by the quadruplet* $(A, B, C, D) \in \mathbb{R}^{n\times n} \times \mathbb{R}^{n\times m} \times \mathbb{R}^{m\times n} \times \mathbb{R}^{m\times m}$*:*

$$\begin{cases} \dot{x} = Ax + B\lambda \\ y = Cx + D\lambda \end{cases} \tag{8.2}$$ {eq:LS}

- *In the Single Input/ Single Output (SISO) case* ($m = 1$)*, the relative degree is defined by the first non zero Markov parameters :*

$$D, CB, CAB, CA^2B, \ldots, CA^{r-1}B, \ldots \tag{8.3}$$ {eq:Markov-Parame

- *In the multiple input/multiple output (MIMO) case* ($m > 1$)*, an uniform relative degree is defined as follows. If* D *is non singular, the relative degree is equal to* 0*. Otherwise, it is assumed to be the first positive integer* r *such that*

$$CA^iB = 0, \quad i = 0 \ldots q-2 \tag{8.4}$$ {eq:mimo-r}

while

$$CA^{r-1}B \text{ is non singular.} \tag{8.5}$$ {eq:mimo-r2}

The Markov parameters arise naturally when we derive with respect to time the output y,

$$\begin{aligned} y &= Cx + D\lambda \\ \dot{y} &= CAx + CB\lambda, \text{ if } D = 0 \\ \ddot{y} &= CA^2x + CAB\lambda, \text{ if } D = 0, CB = 0 \\ &\dots \\ y^{(r)} &= CA^rx + CA^{r-1}B\lambda, \text{ if } D = 0, CB = 0, CA^{r-2}B = 0, r = 1 \dots r-2 \\ &\dots \end{aligned}$$

and the first non zero Markov parameter allows us to define the output y directly in terms of the input λ.

In continuous time, the nature of solutions depends strongly on the relative degree. When we want to perform the time–integration of such systems, we need also to reduce the relative degree or to known it to correctly operate.

We can observe that the relative degree 0 is well defined only by the relation (D nonsingular) and by the nonsmooth law. Indeed, let us imagine that the nonsmooth law is defined by $0 \leq \dot{y} \perp \lambda \geq 0$. We can easily see that the relative degree is reduced.

In the MIMO, the computation of non uniform relative degree is hard task. This is also the case for nonlinear systems.

8.1.2 Second order Lagrangian systems

Let us consider a second order linear and time-invariant Lagrangian dynamical system (see § 2.5)

$$\begin{cases} M\dot{v} + Cv + Kq = F_{Ext}(t) + p \\ \dot{q} = v \end{cases} \tag{8.6}$$ {eq:rd1}

together with a Lagrangian linear relation

$$y = Cq + e + D\lambda + Fz, \tag{8.7}$$ {eq:rd2}

$$p = C^t\lambda \tag{8.8}$$ {eq:rd3}

and a simple nonsmooth law,

$$0 \leq y \perp \lambda \geq 0 \tag{8.9}$$ {eq:rd4}

If $D > 0$, the relative degree is uniformly zero and the system can be solved without deriving the output (8.7). Indeed, we known that the solution of the LCP

$$0 \leq Cq + e + D\lambda + Fz, \perp \lambda \geq 0 \tag{8.10}$$ {eq:rd5}

is unique and Lipschitz with respect to q. It can be denoted as $\lambda(q) = \text{SOL}(D, Cq + e + Fz)$. Therefore, the differential equation (8.6) reduces to a standard ODE with a Lipschitz RHS

$$\begin{cases} M\dot{v} + Cv + Kq = F_{Ext}(t) + C^t\lambda(q) \\ \dot{q} = v \end{cases} \tag{8.11}$$ {eq:rd6}

In the case that we deal with unilateral contact, we usually have $D = 0$ and the relative degree of the system is 2. In this case, the output has to be differentiated as

$$\dot{y} = C\dot{q}, \tag{8.12}$$ {eq:rd7}

and an impact law has to added, for instance the newton's impact law

$$\text{if } y = 0, \text{ when } \dot{y}^+ = -e y^- \tag{8.13}$$ {eq:rd8}

In the same vein, the equations of motion (8.6) is not sufficient since the velocity may encounter jumps. The dynamics is usually replaced by a measure differential equation of the form

$$\begin{cases} M dv + Cv^+(t)dt + Kq(t)dt = F_{Ext}(t)dt + di \\ \dot{q} = v \end{cases} \tag{8.14}$$ {eq:rd10}

where di is the measure that can be related to p thanks to

$$di = pdt + \sigma \delta_{t^*} \tag{8.15}$$ {eq:rd11}

is only one jump is expected at t^*.

8.1.3 Conclusion for the implementation

From the continuous time mathematical analysis, the relative degree is very important to know if we have to compute the derivatives of the output $y^{(n)}$ and to consider various levels for the input $p, \sigma, ...$

However in the numerical practice, the time –discretization makes an assumption on the relative degree and treats the nonsmooth law at different levels. The resulting time discretized system posseses more or less variables.

Consider for instance (8.6) in the case of the Moreau scheme

$$\begin{cases} M(v_{k+1} - v_k) + h(Kq_{k+\theta} + Cv_{k+\theta}) = p_{k+1} = G(q_{k+1})\lambda_{k+1}, & \text{(8.16a)} \\ q_{k+1} = q_k + h v_{k+\theta}, & \text{(8.16b)} \\ \dot{y}_{k+1} = G^\top(q_{k+1})\, v_{k+1} & \text{(8.16c)} \\ \text{if} \quad \bar{y}^\alpha_{k+1} \le 0 \text{ then } 0 \le \dot{y}^\alpha_{k+1} + e\dot{y}^\alpha_k \perp \lambda^\alpha_{k+1} \ge 0, \\ \text{otherwise } \lambda^\alpha_{k+1} = 0. \end{cases} \quad , \alpha \in \mathcal{I} \tag{8.16d}$$ {eq:MoreauTS} {eq:MoreauTSd}

and the Schatzman–Paoli scheme

$$\begin{cases} M(q_{k+1} - 2q_k + q_{k-1}) + h^2(Kq_{k+\theta} + Cv_{k+\theta}) = p_{k+1}, & \text{(8.17a)} \\ v_{k+1} = \dfrac{q_{k+1} - q_{k-1}}{2h}, & \text{(8.17b)} \\ y_{k+1} = h\left(\dfrac{q_{k+1} + eq_{k-1}}{1+e}\right) & \text{(8.17c)} \\ p_{k+1} = G\left(\dfrac{q_{k+1} + eq_{k-1}}{1+e}\right)\lambda_{k+1} & \text{(8.17d)} \\ 0 \le y_{k+1} \perp \lambda_{k+1} \ge 0. & \text{(8.17e)} \end{cases}$$

We can see easily that the number of derivatives (or levels) that we store for y and λ is independent of the relative degree but is chosen by the `OneStepIntegrator` with respect to the type of systems.

8.2 How to define and compute the various levels and the number of indexSets

8.2.1 y related variables

The size of the vector `y` in the `Interaction` depends on

- the OneStepIntegrator type.
  - see the difference between the Moreau and Schatzman Paoli scheme,
  - plan the time–discontinuous Galerkin scheme
  - plan the Higher Order Moreau sweeping process (HOSP)
- the Simulation type.
  - In Timestepping or Event-driven we do not need the same number of stored y
- the NonSmoothLaw type.
  - If we consider some cases with or without friction in Timestepping or Event-driven, we need to adapt the number of stored y

Since the various levels of y are used to build the index sets we will need from 0 to a computed size that depends on the previous criteria. Only a part will be used in the OneStepNSProblem.

8.2.2 λ related variables

The size of the vector lambda in the Interaction depends on the same criteria than in the previous section. Only, the number of lambda is not the same as y since a multiplier lambda[i] is not necessarily related to y[i]

8.3 Rules for implementation

We can define new members in Interaction:

- _levelMinForY, this value is to 0 by default
- _levelMaxForY, this value must be computed at initialization with respect to the previous criteria
- _levelMinForlambda, this value must be computed at initialization with respect to the previous criteria
- _levelMaxForlambda, this value must be computed at initialization with respect to the previous criteria

8.3.1 Redaction note V. ACARY
Where can we centralize this computation of level ?

- the values y[i] must be initialized from _levelMinForY to _levelMaxForY.
- the values lamdba[i] must be initialized from _levelMinForlambda to _levelMinForlambda.
- the values y[i] in Interaction must be used in priority to store the i-th derivative of y. When it is needed, higher index i can be used for other triggering variables. For instance, for an Event–Driven scheme with a Lagrangian systems with friction, sliding velocity must be stored.
- the values of lamdba[i] must stored the various multiplier for the nonsmooth law. We affect the same index i as for the level of y[i] present in the corresponding nonsmooth law.
- The number of IndexSets should follows _levelMaxForY.
- The number of levels for _r and _p in the DS should follow _levelMinForlambda to _levelMinForlambda.

- A new variable should be added in the LagrangianDS to store the multiplier at the position level (`_tau` ?) to avoid the use of `_p[0]`. Indeed, we will continue to assume that `_p` is the input in the equation of motion. For `lambda` we can use `lambda[0]`

TODO LIST

- Remove pseudo computation of relative degree ans this notion form the Kernel
- Implement the centralized computation of levelMin and levelMax...

Chapter 9

Newton's linearization

author	O.Bonnefon, V. Acary
date last update	Sept, 07, 2007 Feb, 2011
version	

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 $t_0 = 0$, $t_{k+1} = t_k + h$, $k \geq 0$, $t_N = T$ 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 $h > 0$.

9.1 Various first order dynamical systems with input/output relations

Fully nonlinear case Let us introduce the following system,

$$\begin{aligned} M\dot{x}(t) &= f(x(t), t) + r(t) \\ y(t) &= h(t, x(t), \lambda(t)) \\ r(t) &= g(t, x(t), \lambda(t)) \end{aligned} \tag{9.1}$$ {first-DS}

where $\lambda(t) \in \mathbb{R}^m$ and $y(t) \in \mathbb{R}^m$ are complementary variables related through a multi-valued mapping. According to the class of systems, 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 **??**. This fully nonlinear case is not implemented in Siconos yet. This fully general case is not yet implemented in Siconos.

FirstOrderType2R Let us introduce a new notation,

$$\begin{aligned} M\dot{x}(t) &= f(x(t), t) + r(t) \\ y(t) &= h(t, x(t), \lambda(t)) \\ r(t) &= g(t, \lambda(t)) \end{aligned} \tag{9.2}$$ {first-DS2}

This case is implemented in Siconos with the relation FirstOrderType2R.

Linear case Let us introduce anew notation,

$$\begin{aligned} M\dot{x}(t) &= Ax(t) + r(t) + b(t) \\ y(t) &= h(x(t), \lambda(t), z) = Cx + Fz + D\lambda \\ r(t) &= g(t, \lambda(t)) = B\lambda \end{aligned} \tag{9.3}$$ {fisrt-DS3}

9.2 Time–discretizations

9.2.1 Standard $\theta - \gamma$ scheme.

Let us now proceed with the time discretization of (9.1) by a fully implicit scheme :

$$\begin{aligned} Mx_{k+1} &= Mx_k + h\theta f(x_{k+1}, t_{k+1}) + h(1-\theta) f(x_k, t_k) + h\gamma r(t_{k+1}) + h(1-\gamma) r(t_k) \\ y_{k+1} &= h(t_{k+1}, x_{k+1}, \lambda_{k+1}) \\ r_{k+1} &= g(x_{k+1}, \lambda_{k+1}, t_{k+1}) \end{aligned} \tag{9.4}$$ {eq:toto1}

where $\theta = [0, 1]$ and $\gamma \in [0, 1]$. As in **?**), we call the problem (9.4) 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.

9.2.2 Full $\theta - \gamma$ scheme

$$\begin{aligned} Mx_{k+1} &= Mx_k + hf(x_{k+\theta}, t_{k+1}) + hr(t_{k+\gamma}) \\ y_{k+\gamma} &= h(t_{k+\gamma}, x_{k+\gamma}, \lambda_{k+\gamma}) \\ r_{k+\gamma} &= g(x_{k+\gamma}, \lambda_{k+\gamma}, t_{k+\gamma}) \\ &\text{NsLaw}(y_{k+\gamma}, \lambda_{k+\gamma}) \end{aligned} \tag{9.5}$$ {eq:toto1-ter}

9.3 Newton's linearization of (9.4)

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" (9.4) by performing an external Newton linearization.

Newton's linearization of the first line of (9.4) The first line of the problem (9.4) can be written under the form of a residue $\mathcal{R}$ depending only on x_{k+1} and r_{k+1} such that

$$\mathcal{R}(x_{k+1}, r_{k+1}) = 0 \tag{9.6}$$ {eq:NL3}

with $\mathcal{R}(x,r) = M(x - x_k) - h\theta f(x, t_{k+1}) - h(1-\theta)f(x_k, t_k) - h\gamma r - h(1-\gamma)r_k$. 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

$$\begin{cases} x^0_{k+1} = x_k \\ \\ \mathcal{R}_L(x^{\alpha+1}_{k+1}, r^{\alpha+1}_{k+1}) = \mathcal{R}(x^\alpha_{k+1}, r^\alpha_{k+1}) + \left[\nabla_x \mathcal{R}(x^\alpha_{k+1}, r^\alpha_{k+1})\right](x^{\alpha+1}_{k+1} - x^\alpha_{k+1}) + \left[\nabla_r \mathcal{R}(x^\alpha_{k+1}, r^\alpha_{k+1})\right](r^{\alpha+1}_{k+1} - r^\alpha_{k+1}) = 0 \end{cases} \tag{9.7}$$ {eq:NL7}

9.3.1 Redaction note V. ACARY
What about r^0_{k+1} ?

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

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

which yields

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

$$\mathcal{R}(x^\alpha_{k+1}, r^\alpha_{k+1}) = \boxed{\mathcal{R}^\alpha_{k+1} \stackrel{\Delta}{=} \mathcal{R}_{free}(x^\alpha_{k+1}, r^\alpha_{k+1}) - h\gamma r^\alpha_{k+1} - h(1-\gamma)r_k} \tag{9.8}$$ {eq:rfree-1}

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

The computation of the Jacobian of $\mathcal{R}$ with respect to x, denoted by W^α_{k+1} leads to

$$W^\alpha_{k+1} \stackrel{\Delta}{=} \nabla_x \mathcal{R}(x^\alpha_{k+1}, r^\alpha_{k+1}) = M - h\theta \nabla_x f(x^\alpha_{k+1}, t_{k+1}). \tag{9.9}$$ {eq:NL9}

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

$$\mathcal{R}^\alpha_{k+1} + W^\alpha_{k+1}(x^{\alpha+1}_{k+1} - x^\alpha_{k+1}) - h\gamma(r^{\alpha+1}_{k+1} - r^\alpha_{k+1}) = 0, \tag{9.10}$$ {eq:NL10}

By using (9.8), we get

$$\mathcal{R}_{free}(x^\alpha_{k+1}, r^\alpha_{k+1}) - h\gamma r^{\alpha+1}_{k+1} - h(1-\gamma)r_k + W^\alpha_{k+1}(x^{\alpha+1}_{k+1} - x^\alpha_{k+1}) = 0 \tag{9.11}$$ {eq:rfree-2}

$$\boxed{x^{\alpha+1}_{k+1} = h(W^\alpha_{k+1})^{-1} r^{\alpha+1}_{k+1} + x^\alpha_{free}} \tag{9.12}$$

with :

$$\boxed{x^\alpha_{free} \stackrel{\Delta}{=} x^\alpha_{k+1} - (W^\alpha_{k+1})^{-1}(R^\alpha_{freek+1} - h(1-\gamma)r_k)} \tag{9.13}$$ {eq:rfree-12}

The matrix W is clearly non singular for small h.

Newton's linearization of the second line of (9.4) The same operation is performed with the second equation of (9.4)

$$\mathcal{R}_y(x, y, \lambda) = y - h(t_{k+1}, x, \lambda) = 0 \tag{9.14}$$

which is linearized as

$$\begin{aligned}\mathcal{R}_{Ly}(x^{\alpha+1}_{k+1}, y^{\alpha+1}_{k+1}, \lambda^{\alpha+1}_{k+1}) = \mathcal{R}_y(x^\alpha_{k+1}, y^\alpha_{k+1}, \lambda^\alpha_{k+1}) + (y^{\alpha+1}_{k+1} - y^\alpha_{k+1}) - \\ C^\alpha_{k+1}(x^{\alpha+1}_{k+1} - x^\alpha_{k+1}) - D^\alpha_{k+1}(\lambda^{\alpha+1}_{k+1} - \lambda^\alpha_{k+1}) = 0\end{aligned} \tag{9.15}$$ {eq:NL9}

This leads to the following linear equation

$$\boxed{y^{\alpha+1}_{k+1} = y^\alpha_{k+1} - \mathcal{R}^\alpha_{yk+1} + C^\alpha_{k+1}(x^{\alpha+1}_{k+1} - x^\alpha_{k+1}) + D^\alpha_{k+1}(\lambda^{\alpha+1}_{k+1} - \lambda^\alpha_{k+1})}. \tag{9.16}$$ {eq:NL11y}

with,

$$\begin{aligned}C^\alpha_{k+1} &= \nabla_x h(t_{k+1}, x^\alpha_{k+1}, \lambda^\alpha_{k+1}) \\ D^\alpha_{k+1} &= \nabla_\lambda h(t_{k+1}, x^\alpha_{k+1}, \lambda^\alpha_{k+1})\end{aligned} \tag{9.17}$$

and

$$\boxed{\mathcal{R}^\alpha_{yk+1} \stackrel{\Delta}{=} y^\alpha_{k+1} - h(x^\alpha_{k+1}, \lambda^\alpha_{k+1})} \tag{9.18}$$

Newton's linearization of the third line of (9.4) The same operation is performed with the third equation of (9.4)

$$\mathcal{R}_r(r, x, \lambda) = r - g(x, \lambda, t_{k+1}) = 0 \tag{9.19}$$

which is linearized as

$$\mathcal{R}_{L\lambda}(r^{\alpha+1}_{k+1}, x^{\alpha+1}_{k+1}, \lambda^{\alpha+1}_{k+1}) = \mathcal{R}^\alpha_{rk+1} + (r^{\alpha+1}_{k+1} - r^\alpha_{k+1}) - K^\alpha_{k+1}(x^{\alpha+1}_{k+1} - x^\alpha_{k+1}) - B^\alpha_{k+1}(\lambda^{\alpha+1}_{k+1} - \lambda^\alpha_{k+1}) = 0 \tag{9.20}$$ {eq:NL9}

$$\boxed{r^{\alpha+1}_{k+1} = g(x^\alpha_{k+1}, \lambda^\alpha_{k+1}, t_{k+1}) - B^\alpha_{k+1}\lambda^\alpha_{k+1} + B^\alpha_{k+1}\lambda^{\alpha+1}_{k+1} - K^\alpha_{k+1}x^\alpha_{k+1} + K^\alpha_{k+1}x^{\alpha+1}_{k+1}} \tag{9.21}$$ {eq:rrL}

with,

$$\begin{aligned}K^\alpha_{k+1} &= \nabla_x g(x^\alpha_{k+1}, \lambda^\alpha_{k+1}, t_{k+1}) \\ B^\alpha_{k+1} &= \nabla_\lambda g(x^\alpha_{k+1}, \lambda^\alpha_{k+1}, t_{k+1})\end{aligned} \tag{9.22}$$

and the residue for r:

$$\boxed{\mathcal{R}^\alpha_{rk+1} = r^\alpha_{k+1} - g(x^\alpha_{k+1}, \lambda^\alpha_{k+1}, t_{k+1})} \tag{9.23}$$

Reduction to a linear relation between $x_{k+1}^{\alpha+1}$ and $\lambda_{k+1}^{\alpha+1}$ Inserting (9.21) into (9.13), we get the following linear relation between $x_{k+1}^{\alpha+1}$ and $\lambda_{k+1}^{\alpha+1}$,

$$x_{k+1}^{\alpha+1} = h\gamma(W_{k+1}^{\alpha})^{-1}\left[g(x_{k+1}^{\alpha},\lambda_{k+1}^{\alpha},t_{k+1}) + B_{k+1}^{\alpha}(\lambda_{k+1}^{\alpha+1}-\lambda_{k+1}^{\alpha}) + K_{k+1}^{\alpha}(x_{k+1}^{\alpha+1}-x_{k+1}^{\alpha})\right] + x_{free}^{\alpha} \tag{9.24}$$

that is

$$(I - h\gamma(W_{k+1}^{\alpha})^{-1}K_{k+1}^{\alpha})x_{k+1}^{\alpha+1} = x_p + h\gamma(W_{k+1}^{\alpha})^{-1}B_{k+1}^{\alpha}\lambda_{k+1}^{\alpha+1} \tag{9.25}$$

with

$$\boxed{x_p \stackrel{\Delta}{=} h\gamma(W_{k+1}^{\alpha})^{-1}\left[g(x_{k+1}^{\alpha},\lambda_{k+1}^{\alpha},t_{k+1}) + -B_{k+1}^{\alpha}(\lambda_{k+1}^{\alpha}) - K_{k+1}^{\alpha}(x_{k+1}^{\alpha})\right] + x_{free}^{\alpha}} \tag{9.26}$$

Let us define the new matrix

$$\hat{K}_{k+1}^{\alpha} = (I - h\gamma(W_{k+1}^{\alpha})K_{k+1}^{\alpha}). \tag{9.27}$$ {eq:hatW}

We get the linear relation

$$\boxed{x_{k+1}^{\alpha+1} \stackrel{\Delta}{=} \hat{K}_{k+1}^{\alpha,-1}x_p + \hat{K}_{k+1}^{\alpha,-1}\left[h\gamma(W_{k+1}^{\alpha})^{-1}B_{k+1}^{\alpha}\lambda_{k+1}^{\alpha+1}\right]} \tag{9.28}$$ {eq:rfree-13}

Reduction to a linear relation between $y_{k+1}^{\alpha+1}$ and $\lambda_{k+1}^{\alpha+1}$ Inserting (9.28) into (9.16), we get the following linear relation between $y_{k+1}^{\alpha+1}$ and $\lambda_{k+1}^{\alpha+1}$,

$$y_{k+1}^{\alpha+1} = y_p + \left[hC_{k+1}^{\alpha}(\tilde{K}_{k+1}^{\alpha})^{-1}(W_{k+1}^{\alpha})^{-1}B_{k+1}^{\alpha} + D_{k+1}^{\alpha}\right]\lambda_{k+1}^{\alpha+1} \tag{9.29}$$

with

$$\boxed{y_p = y_{k+1}^{\alpha} - \mathcal{R}_{yk+1}^{\alpha} + C_{k+1}^{\alpha}(x_q) - D_{k+1}^{\alpha}\lambda_{k+1}^{\alpha}} \tag{9.30}$$

$$\boxed{x_q = (\tilde{K}_{k+1}^{\alpha})^{-1}x_p - x_{k+1}^{\alpha}} \tag{9.31}$$ {eq:xqq}

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

$$\begin{cases} y_{k+1}^{\alpha+1} = W_{mlcpk+1}^{\alpha}\lambda_{k+1}^{\alpha+1} + b_{k+1}^{\alpha} \\ \\ -y_{k+1}^{\alpha+1} \in N_{[l,u]}(\lambda_{k+1}^{\alpha+1}). \end{cases} \tag{9.32}$$ {eq:NL14}

with $W_{mlcpk+1} \in \mathbb{R}^{m\times m}$ and $b \in \mathbb{R}^{m}$ defined by

$$\begin{array}{l} W_{mlcpk+1}^{\alpha} = hC_{k+1}^{\alpha}(\tilde{K}_{k+1}^{\alpha})^{-1}(W_{k+1}^{\alpha})^{-1}B_{k+1}^{\alpha} + D_{k+1}^{\alpha} \\ b_{k+1}^{\alpha} = y_p \end{array} \tag{9.33}$$ {eq:NL15}

The problem (9.32) 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,

$$
\begin{aligned}
& y_{k+1}^{\alpha+1} = -W_{k+1}^{\alpha} \lambda_{k+1}^{\alpha+1} + b_{k+1}^{\alpha} \\
& (y_{k+1}^{\alpha+1})_i = 0 \qquad \text{for } i \in \{1..n\} \\
& 0 \leq (\lambda_{k+1}^{\alpha+1})_i \perp (y_{k+1}^{\alpha+1})_i \geq 0 \qquad \text{for } i \in \{n..n+m\}
\end{aligned}
\tag{9.34}
$$

{eq:MLCP1}

9.4 Newton's linearization of (9.2)

Let us now proceed with the time discretization of (9.2) by a fully implicit scheme :

$$\begin{aligned} Mx_{k+1} &= Mx_k + h\theta f(x_{k+1},t_{k+1}) + h(1-\theta)f(x_k,t_k) + h\gamma r(t_{k+1}) + h(1-\gamma)r(t_k) \\ y_{k+1} &= h(t_{k+1},x_{k+1},\lambda_{k+1}) \\ r_{k+1} &= g(\lambda_{k+1},t_{k+1}) \end{aligned} \tag{9.35}$$ {eq:mlcp2-toto1-

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

$$\mathcal{R}(x_{k+1},r_{k+1}) = 0 \tag{9.36}$$ {eq:mlcp2-NL3}

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

$$\begin{cases} x_{k+1}^{0} = x_k \\ \\ \mathcal{R}_L(x_{k+1}^{\alpha+1},r_{k+1}^{\alpha+1}) = \mathcal{R}(x_{k+1}^{\alpha},r_{k+1}^{\alpha}) + \left[\nabla_x\mathcal{R}(x_{k+1}^{\alpha},r_{k+1}^{\alpha})\right](x_{k+1}^{\alpha+1}-x_{k+1}^{\alpha}) + \left[\nabla_r\mathcal{R}(x_{k+1}^{\alpha},r_{k+1}^{\alpha})\right](r_{k+1}^{\alpha+1}-r_{k+1}^{\alpha}) = 0 \end{cases} \tag{9.37}$$ {eq:mlcp2-NL7}

9.4.1 Redaction note V. ACARY
What about r_{k+1}^{0} ?

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

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

which yields

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

$$\mathcal{R}(x_{k+1}^{\alpha},r_{k+1}^{\alpha}) = \boxed{\mathcal{R}_{k+1}^{\alpha} \stackrel{\Delta}{=} \mathcal{R}_{free}(x_{k+1}^{\alpha},r_{k+1}^{\alpha}) - h\gamma r_{k+1}^{\alpha} - h(1-\gamma)r_k} \tag{9.38}$$ {eq:mlcp2-rfree-

$$\mathcal{R}_{free}(x_{k+1}^{\alpha},r_{k+1}^{\alpha}) = \boxed{\mathcal{R}_{freek+1}^{\alpha} \stackrel{\Delta}{=} M(x_{k+1}^{\alpha}-x_k) - h\theta f(x_{k+1}^{\alpha},t_{k+1}) - h(1-\theta)f(x_k,t_k)}$$

The computation of the Jacobian of $\mathcal{R}$ with respect to x, denoted by W_{k+1}^{α} leads to

$$W_{k+1}^{\alpha} \stackrel{\Delta}{=} \nabla_x\mathcal{R}(x_{k+1}^{\alpha},r_{k+1}^{\alpha}) = M - h\theta\nabla_x f(x_{k+1}^{\alpha},t_{k+1}). \tag{9.39}$$ {eq:mlcp2-NL9}

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

$$\mathcal{R}_{k+1}^{\alpha} + W_{k+1}^{\alpha}(x_{k+1}^{\alpha+1}-x_{k+1}^{\alpha}) - h\gamma(r_{k+1}^{\alpha+1}-r_{k+1}^{\alpha}) = 0, \tag{9.40}$$ {eq:mlcp2-NL10}

By using (9.38), we get

$$\mathcal{R}_{free}(x_{k+1}^{\alpha},r_{k+1}^{\alpha}) - h\gamma r_{k+1}^{\alpha+1} - h(1-\gamma)r_k + W_{k+1}^{\alpha}(x_{k+1}^{\alpha+1}-x_{k+1}^{\alpha}) = 0 \tag{9.41}$$ {eq:mlcp2-rfree-

$$\boxed{x_{k+1}^{\alpha+1} = h(W_{k+1}^{\alpha})^{-1} r_{k+1}^{\alpha+1} + x_{free}^{\alpha}} \tag{9.42}$$

with :

$$\boxed{x_{free}^{\alpha} \stackrel{\Delta}{=} x_{k+1}^{\alpha} - (W_{k+1}^{\alpha})^{-1}(R_{freek+1}^{\alpha} - h(1-\gamma)r_k)} \tag{9.43}$$ {eq:mlcp2-rfree-

The matrix W is clearly non singular for small h.

Newton's linearization of the second line of (9.35) The same operation is performed with the second equation of (9.35)

$$\mathcal{R}_y(x, y, \lambda) = y - h(t_{k+1}, x, \lambda) = 0 \tag{9.44}$$

which is linearized as

$$\begin{aligned} \mathcal{R}_{Ly}(x_{k+1}^{\alpha+1}, y_{k+1}^{\alpha+1}, \lambda_{k+1}^{\alpha+1}) = \mathcal{R}_y(x_{k+1}^{\alpha}, y_{k+1}^{\alpha}, \lambda_{k+1}^{\alpha}) + (y_{k+1}^{\alpha+1} - y_{k+1}^{\alpha}) - \\ C_{k+1}^{\alpha}(x_{k+1}^{\alpha+1} - x_{k+1}^{\alpha}) - D_{k+1}^{\alpha}(\lambda_{k+1}^{\alpha+1} - \lambda_{k+1}^{\alpha+1}) = 0 \end{aligned} \tag{9.45}$$ {eq:mlcp2-NL9}

This leads to the following linear equation

$$\boxed{y_{k+1}^{\alpha+1} = y_{k+1}^{\alpha} - \mathcal{R}_{yk+1}^{\alpha} + C_{k+1}^{\alpha}(x_{k+1}^{\alpha+1} - x_{k+1}^{\alpha}) + D_{k+1}^{\alpha}(\lambda_{k+1}^{\alpha+1} - \lambda_{k+1}^{\alpha})}. \tag{9.46}$$ {eq:mlcp2-NL11y}

with,

$$\begin{aligned} C_{k+1}^{\alpha} &= \nabla_x h(t_{k+1}, x_{k+1}^{\alpha}, \lambda_{k+1}^{\alpha}) \\ D_{k+1}^{\alpha} &= \nabla_\lambda h(t_{k+1}, x_{k+1}^{\alpha}, \lambda_{k+1}^{\alpha}) \end{aligned} \tag{9.47}$$

and

$$\boxed{\mathcal{R}_{yk+1}^{\alpha} \stackrel{\Delta}{=} y_{k+1}^{\alpha} - h(x_{k+1}^{\alpha}, \lambda_{k+1}^{\alpha})} \tag{9.48}$$

Newton's linearization of the third line of (9.35) The same operation is performed with the third equation of (9.35)

$$\mathcal{R}_r(r, x, \lambda) = r - g(\lambda, t_{k+1}) = 0 \tag{9.49}$$

which is linearized as

$$\mathcal{R}_{L\lambda}(r_{k+1}^{\alpha+1}, x_{k+1}^{\alpha+1}, \lambda_{k+1}^{\alpha+1}) = \mathcal{R}_{rk+1}^{\alpha} + (r_{k+1}^{\alpha+1} - r_{k+1}^{\alpha}) - B_{k+1}^{\alpha}(\lambda_{k+1}^{\alpha+1} - \lambda_{k+1}^{\alpha}) = 0 \tag{9.50}$$ {eq:mlcp2-NL9}

$$\boxed{r_{k+1}^{\alpha+1} = g(x_{k+1}^{\alpha}, \lambda_{k+1}^{\alpha}, t_{k+1}) - B_{k+1}^{\alpha}\lambda_{k+1}^{\alpha} + B_{k+1}^{\alpha}\lambda^{\alpha+1}} \tag{9.51}$$ {eq:mlcp2-rrL}

with,

$$B_{k+1}^{\alpha} = \nabla_\lambda g(x_{k+1}^{\alpha}, \lambda_{k+1}^{\alpha}, t_{k+1}) \tag{9.52}$$

and the residue for r:

$$\boxed{\mathcal{R}_{rk+1}^{\alpha} = r_{k+1}^{\alpha} - g(\lambda_{k+1}^{\alpha}, t_{k+1})} \tag{9.53}$$

Reduction to a linear relation between $x_{k+1}^{\alpha+1}$ and $\lambda_{k+1}^{\alpha+1}$ Inserting (9.51) into (9.43), we get the following linear relation between $x_{k+1}^{\alpha+1}$ and $\lambda_{k+1}^{\alpha+1}$,

$$x_{k+1}^{\alpha+1} = h\gamma(W_{k+1}^{\alpha})^{-1}\left[g(x_{k+1}^{\alpha}, \lambda_{k+1}^{\alpha}, t_{k+1}) + B_{k+1}^{\alpha}(\lambda_{k+1}^{\alpha+1} - \lambda_{k+1}^{\alpha})\right] + x_{free}^{\alpha} \tag{9.54}$$

that is

$$x_{k+1}^{\alpha+1} = x_p + h\gamma(W_{k+1}^{\alpha})^{-1}B_{k+1}^{\alpha}\lambda_{k+1}^{\alpha+1} \tag{9.55}$$

with

$$\boxed{x_p \stackrel{\Delta}{=} h\gamma(W_{k+1}^{\alpha})^{-1}\left[g(x_{k+1}^{\alpha}, \lambda_{k+1}^{\alpha}, t_{k+1}) + -B_{k+1}^{\alpha}(\lambda_{k+1}^{\alpha})\right] + x_{free}^{\alpha}} \tag{9.56}$$

We get the linear relation

$$\boxed{x_{k+1}^{\alpha+1} \stackrel{\Delta}{=} x_p + \left[h\gamma(W_{k+1}^{\alpha})^{-1}B_{k+1}^{\alpha}\lambda_{k+1}^{\alpha+1}\right]} \tag{9.57}$$ {eq:mlcp2-rfree-

Reduction to a linear relation between $y_{k+1}^{\alpha+1}$ and $\lambda_{k+1}^{\alpha+1}$ Inserting (9.57) into (9.46), we get the following linear relation between $y_{k+1}^{\alpha+1}$ and $\lambda_{k+1}^{\alpha+1}$,

$$y_{k+1}^{\alpha+1} = y_p + \left[hC_{k+1}^{\alpha}(W_{k+1}^{\alpha})^{-1}B_{k+1}^{\alpha} + D_{k+1}^{\alpha}\right]\lambda_{k+1}^{\alpha+1} \tag{9.58}$$

with

$$\boxed{y_p = y_{k+1}^{\alpha} - \mathcal{R}_{yk+1}^{\alpha} + C_{k+1}^{\alpha}(x_q) - D_{k+1}^{\alpha}\lambda_{k+1}^{\alpha}} \tag{9.59}$$

$$\boxed{x_q = x_p - x_{k+1}^{\alpha}} \tag{9.60}$$ {eq:mlcp2-xqq}

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

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

$$\begin{aligned} Mx_{k+1}^{\alpha+1} &= Mx_k + h\theta Ax_{k+1}^{\alpha+1} + h(1-\theta)Ax_k + h\gamma r_{k+1}^{\alpha+1} + h(1-\gamma)r(t_k) + hb \\ y_{k+1}^{\alpha+1} &= Cx_{k+1}^{\alpha+1} + D\lambda_{k+1}^{\alpha+1} + Fz + e \\ r_{k+1}^{\alpha+1} &= B\lambda_{k+1}^{\alpha+1} \end{aligned} \tag{9.61}$$ {eq:toto1-DS3}

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

9.4.2 Resulting Newton step (only one step)

suppose:$\gamma = 1$

$$\begin{aligned} (M - h\theta A)x_{k+1}^{\alpha+1} &= Mx_k + h(1-\theta)Ax_k + hr_{k+1}^{\alpha+1} + hb \\ y_{k+1}^{\alpha+1} &= Cx_{k+1}^{\alpha+1} + D\lambda_{k+1}^{\alpha+1} + Fz + e \\ r_{k+1}^{\alpha+1} &= B\lambda_{k+1}^{\alpha+1} \end{aligned} \tag{9.62}$$

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

$$\begin{aligned} x_{k+1}^{\alpha+1} &= W^{-1}(Mx_k + h(1-\theta)Ax_k + r_{k+1}^{\alpha+1} + hb) = xfree + W^{-1}(r_{k+1}^{\alpha+1}) \\ y_{k+1}^{\alpha+1} &= (D + hCW^{-1}B)\lambda_{k+1}^{\alpha+1} + Fz + CW^{-1}(Mx_k + h(1-\theta)Ax_k + hb) + e \end{aligned} \tag{9.63}$$

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

$$\begin{aligned} y_{k+1}^{\alpha+1} &= (D + hCW^{-1}B)\lambda_{k+1}^{\alpha+1} + Fz + Cx_{free} + e \\ r_{k+1}^{\alpha+1} &= B\lambda_{k+1}^{\alpha+1} \end{aligned} \tag{9.64}$$

9.4.3 coherence with previous formulation

$$y_p = y_{k+1}^{\alpha} - \mathcal{R}_{yk+1}^{\alpha} + C_{k+1}^{\alpha}(x_p - x_{k+1}^{\alpha}) - D_{k+1}^{\alpha}\lambda_{k+1}^{\alpha}$$
$$y_p = Cx_k + D\lambda_k + C(\tilde{x}_{free}) - D\lambda_k + Fz + e$$
$$y_p = Cx_k + C(\tilde{x}_{free}) + Fz + e$$
$$y_p = Cx_k + C(\tilde{x}_{free}) + Fz + e$$
$$y_p = C(x_{free}) + Fz + e$$

9.5 Newton's linearization of (9.2) with (9.5)

$$\begin{aligned} Mx_{k+1} &= Mx_k + h\theta f(x_{k+1}, t_{k+1}) + h(1-\theta) f(x_k, t_k) + hr_{k+\gamma} \\ y_{k+\gamma} &= h(t_{k+\gamma}, x_{k+\gamma}, \lambda_{k+\gamma}) \\ r_{k+\gamma} &= g(\lambda_{k+\gamma}, t_{k+\gamma}) \end{aligned} \tag{9.65}$$ {eq:full-toto1-t

Newton's linearization of the first line of (9.65) The first line of the problem (**??**) can be written under the form of a residue $\mathcal{R}$ depending only on x_{k+1} and $r_{k+\gamma}$ such that

$$\mathcal{R}(x_{k+1}, r_{k+\gamma}) = 0 \tag{9.66}$$ {eq:full-NL3}

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

$$\begin{cases} x^0_{k+1} = x_k \\ \mathcal{R}_L(x^{\alpha+1}_{k+1}, r^{\alpha+1}_{k+\gamma}) = \mathcal{R}(x^\alpha_{k+1}, r^\alpha_{k+\gamma}) + \left[\nabla_x \mathcal{R}(x^\alpha_{k+1}, r^\alpha_{k+\gamma})\right](x^{\alpha+1}_{k+1} - x^\alpha_{k+1}) + \\ \qquad \left[\nabla_r \mathcal{R}(x^\alpha_{k+1}, r^\alpha_{k+\gamma})\right](r^{\alpha+1}_{k+\gamma} - r^\alpha_{k+\gamma}) = 0 \end{cases} \tag{9.67}$$ {eq:full-NL7}

9.5.1 Redaction note V. ACARY
What about $r^0_{k+\gamma}$?

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

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

which yields

$$\mathcal{R}(x, r) = \mathcal{R}_{free}(x) - hr.$$

$$\mathcal{R}(x^\alpha_{k+1}, r^\alpha_{k+\gamma}) = \boxed{\mathcal{R}^\alpha_{k+1} \stackrel{\Delta}{=} \mathcal{R}_{free}(x^\alpha_{k+1}) - hr^\alpha_{k+\gamma}} \tag{9.68}$$ {eq:full-rfree-1

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

The computation of the Jacobian of $\mathcal{R}$ with respect to x, denoted by W^α_{k+1} leads to

$$W^\alpha_{k+1} \stackrel{\Delta}{=} \nabla_x \mathcal{R}(x^\alpha_{k+1}) = M - h\theta \nabla_x f(x^\alpha_{k+1}, t_{k+1}). \tag{9.69}$$ {eq:full-NL9}

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

$$\mathcal{R}^\alpha_{k+1} + W^\alpha_{k+1}(x^{\alpha+1}_{k+1} - x^\alpha_{k+1}) - h(r^{\alpha+1}_{k+\gamma} - r^\alpha_{k+\gamma}) = 0, \tag{9.70}$$ {eq:full-NL10}

By using (9.68), we get

$$\mathcal{R}_{free}(x^\alpha_{k+1}) - hr^{\alpha+1}_{k+\gamma} + W^\alpha_{k+1}(x^{\alpha+1}_{k+1} - x^\alpha_{k+1}) = 0 \tag{9.71}$$ {eq:full-rfree-2

$$\boxed{x^{\alpha+1}_{k+1} = h(W^\alpha_{k+1})^{-1} r^{\alpha+1}_{k+1} + x^\alpha_{free}} \tag{9.72}$$

with :

$$\boxed{x^\alpha_{free} \stackrel{\Delta}{=} x^\alpha_{k+1} - (W^\alpha_{k+1})^{-1}\mathcal{R}^\alpha_{freek+1}} \tag{9.73}$$ {eq:full-rfree-1

The matrix W is clearly non singular for small h.

Newton's linearization of the second line of (??) The same operation is performed with the second equation of (**??**)

$$\mathcal{R}_y(x, y, \lambda) = y - h(t_{k+\gamma}, \gamma x + (1-\gamma)x_k, \lambda) = 0 \tag{9.74}$$

which is linearized as

$$\begin{aligned}\mathcal{R}_{Ly}(x^{\alpha+1}_{k+1}, y^{\alpha+1}_{k+\gamma}, \lambda^{\alpha+1}_{k+\gamma}) = \mathcal{R}_y(x^\alpha_{k+1}, y^\alpha_{k+\gamma}, \lambda^\alpha_{k+\gamma}) + (y^{\alpha+1}_{k+\gamma} - y^\alpha_{k+\gamma}) - \\ \gamma C^\alpha_{k+1}(x^{\alpha+1}_{k+1} - x^\alpha_{k+1}) - D^\alpha_{k+\gamma}(\lambda^{\alpha+1}_{k+\gamma} - \lambda^\alpha_{k+\gamma}) = 0\end{aligned} \tag{9.75}$$ {eq:full-NL9}

This leads to the following linear equation

$$\boxed{y^{\alpha+1}_{k+\gamma} = y^\alpha_{k+\gamma} - \mathcal{R}^\alpha_{y,k+1} + \gamma C^\alpha_{k+1}(x^{\alpha+1}_{k+1} - x^\alpha_{k+1}) + D^\alpha_{k+\gamma}(\lambda^{\alpha+1}_{k+\gamma} - \lambda^\alpha_{k+\gamma})}. \tag{9.76}$$ {eq:full-NL11y}

with,

$$\begin{aligned}C^\alpha_{k+\gamma} &= \nabla_x h(t_{k+1}, x^\alpha_{k+\gamma'}, \lambda^\alpha_{k+\gamma}) \\ D^\alpha_{k+\gamma} &= \nabla_\lambda h(t_{k+1}, x^\alpha_{k+\gamma'}, \lambda^\alpha_{k+\gamma})\end{aligned} \tag{9.77}$$

and

$$\boxed{\mathcal{R}^\alpha_{yk+1} \stackrel{\Delta}{=} y^\alpha_{k+\gamma} - h(x^\alpha_{k+\gamma'}, \lambda^\alpha_{k+\gamma})} \tag{9.78}$$

Newton's linearization of the third line of (??) The same operation is performed with the third equation of (**??**)

$$\mathcal{R}_r(r, \lambda) = r - g(\lambda, t_{k+1}) = 0 \tag{9.79}$$

which is linearized as

$$\mathcal{R}_{L\lambda}(r^{\alpha+1}_{k+\gamma}, \lambda^{\alpha+1}_{k+\gamma}) = \mathcal{R}^\alpha_{r,k+\gamma} + (r^{\alpha+1}_{k+\gamma} - r^\alpha_{k+\gamma}) - B^\alpha_{k+\gamma}(\lambda^{\alpha+1}_{k+\gamma} - \lambda^\alpha_{k+\gamma}) = 0 \tag{9.80}$$ {eq:full-NL9}

$$\boxed{r^{\alpha+1}_{k+\gamma} = g(\lambda^\alpha_{k+\gamma}, t_{k+\gamma}) - B^\alpha_{k+\gamma}\lambda^\alpha_{k+\gamma} + B^\alpha_{k+\gamma}\lambda^{\alpha+1}_{k+\gamma}} \tag{9.81}$$ {eq:full-rrL}

with,

$$B^\alpha_{k+\gamma} = \nabla_\lambda g(\lambda^\alpha_{k+\gamma}, t_{k+\gamma}) \tag{9.82}$$

and the residue for r:

$$\boxed{\mathcal{R}^\alpha_{rk+\gamma} = r^\alpha_{k+\gamma} - g(\lambda^\alpha_{k+\gamma}, t_{k+\gamma})} \tag{9.83}$$

Reduction to a linear relation between $x_{k+1}^{\alpha+1}$ and $\lambda_{k+\gamma}^{\alpha+1}$ Inserting (9.81) into (9.73), we get the following linear relation between $x_{k+1}^{\alpha+1}$ and $\lambda_{k+1}^{\alpha+1}$,

$$x_{k+1}^{\alpha+1} = h(W_{k+1}^{\alpha})^{-1}\left[g(\lambda_{k+\gamma}^{\alpha}, t_{k+\gamma}) + B_{k+\gamma}^{\alpha}(\lambda_{k+\gamma}^{\alpha+1} - \lambda_{k+\gamma}^{\alpha})\right] + x_{free}^{\alpha} \tag{9.84}$$

that is

$$\boxed{x_{k+1}^{\alpha+1} = x_p + h(W_{k+1}^{\alpha})^{-1} B_{k+\gamma}^{\alpha} \lambda_{k+\gamma}^{\alpha+1}} \tag{9.85}$$ {eq:full-rfree-1

with

$$\boxed{x_p \stackrel{\Delta}{=} h(W_{k+1}^{\alpha})^{-1}\left[g(\lambda_{k+\gamma}^{\alpha}, t_{k+\gamma}) - B_{k+\gamma}^{\alpha}(\lambda_{k+\gamma}^{\alpha})\right] + x_{free}^{\alpha}} \tag{9.86}$$

Reduction to a linear relation between $y_{k+\gamma}^{\alpha+1}$ and $\lambda_{k+\gamma}^{\alpha+1}$ Inserting (9.85) into (9.76), we get the following linear relation between $y_{k+1}^{\alpha+1}$ and $\lambda_{k+1}^{\alpha+1}$,

$$y_{k+1}^{\alpha+1} = y_p + \left[h\gamma C_{k+\gamma}^{\alpha}(W_{k+1}^{\alpha})^{-1} B_{k+1}^{\alpha} + D_{k+1}^{\alpha}\right] \lambda_{k+1}^{\alpha+1} \tag{9.87}$$

with

$$y_p = y_{k+1}^{\alpha} - \mathcal{R}_{yk+1}^{\alpha} + \gamma C_{k+1}^{\alpha}(x_q) - D_{k+1}^{\alpha}\lambda_{k+1}^{\alpha} \tag{9.88}$$

that is

$$\boxed{y_p = h(x_{k+\gamma}^{\alpha}, \lambda_{k+\gamma}^{\alpha}) + \gamma C_{k+1}^{\alpha}(x_q) - D_{k+1}^{\alpha}\lambda_{k+1}^{\alpha}} \tag{9.89}$$

$$\boxed{x_q = (x_p - x_{k+1}^{\alpha})} \tag{9.90}$$ {eq:full-xqq}

The linear case

$$\begin{aligned} y_p &= h(x_{k+\gamma}^{\alpha}, \lambda_{k+\gamma}^{\alpha}) + \gamma C_{k+1}^{\alpha}(x_q) - D_{k+1}^{\alpha}\lambda_{k+1}^{\alpha} \\ &= C_{k+1}^{\alpha}x_{k+\gamma}^{\alpha} + D_{k+1}^{\alpha}\lambda_{k+\gamma}^{\alpha} + \gamma C_{k+1}^{\alpha}(x_q) - D_{k+1}^{\alpha}\lambda_{k+1}^{\alpha} \\ &= C_{k+1}^{\alpha}(x_{k+\gamma}^{\alpha} + \gamma x_p - \gamma x_{k+1}^{\alpha}) \\ &= C_{k+1}^{\alpha}((1-\gamma)x_k + \gamma x_{free}) \text{since } x_p = x_{free} \end{aligned} \tag{9.91}$$

Implementation details For the moment (Feb. 2011), we set $x_q = (1-\gamma)x_k + \gamma x_{free}$ in the linear case The nonlinear case is not yet implemented since we need to change the management of `H_alpha}`in Relation to be able to compute the mid–point values. things that remain to do

- implement the function `BlockVector computeg(t,lambda)` and `SimpleVector computeh(t,x,lambda)` which takes into account the values of the argument and return and vector
- remove temporary computation in Relation of `Xq,g_qlphaand H_alpha`. This should be stored somewhere else. (in the node of the graph)

Chapter 10

NewtonEuler Dynamical Systems

Author	O. Bonnefon	2010
Revision	section 9.1 to 9.3 V. Acary	05/09/2011

10.1 The Equations of motion

The equations of motion in the Newton-Euler(**??**) formalism can be stated as

$$\left\{\begin{array}{rcl} M\dot{V} & = & F_{ext}(X,V,\Omega,R), \\ I\dot{\Omega}+\Omega\wedge I\Omega & = & M_{ext}(X,V,\Omega,R), \\ \dot{X} & = & V, \\ \dot{R} & = & R\tilde{\Omega}, \quad R^{-1}=R^{T}, \quad \det(R)=1. \end{array}\right. \tag{10.1}$$ {eq:NewtonEuler}

with

- x_G, v_G position and velocity of the center of mass expressed in a spatial frame
- Ω angular velocity vector expressed in the inertial frame (frame attached to the object)
- R rotation matrix form the spatial frame to the inertia frame [1] R
- $M = m\, I_{3\times 3}$ diagonal mass matrix
- $m \in I\!R$ mass
- I constant inertia matrix
- F_{ext} and M_{ext} are the external applied forces and torques

10.2 The relation

Let us define by q by the position and the orientation, we don't focus on the representation of the orientation.

$$\begin{array}{l} Y = H(q) \\ R = G(q,\lambda) \end{array} \tag{10.2}$$ {Relation}

The first equation is derived:

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

Let us assume that it exists an operator T such that :

$$T : \left(\begin{array}{c} V \\ \Omega \end{array}\right) \rightarrow \dot{q}$$

[1] $R^{-1} = R^T, \det(R) = 1$, *i.e* $R \in SO^+(3)$

Using this operator, (10.2) leads to :

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

10.3 Time discretization $t_k \to t_{k+1}$, and implementation in Siconos

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

10.3.1 The unknowns

The unknowns stored in the `NewtonEulerDS` class are the velocity

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

and the parameter $\_q_k$ that locates the system. Usually it may be the coordinate of the center of mass and a representation of the orientation (quaternion, Euler Angles, ...).

10.3.2 Explicit case

It consists in evaluating $\Omega \wedge I\Omega$ in an explicit way. We note that it can cause trouble (numerical instabilities) for object having an important condition number of the inertial matrix. The dynamical system (12.2) results in to the system:

$$\begin{pmatrix} m & 0 \\ 0 & I \end{pmatrix} (\_v_{k+1} - \_v_k) = h \left[\_Fl_k + (CT)^\top \lambda_{k+1} \right] \tag{10.3}$$

with the total forces applied to the system

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

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

10.3.3 θ method case

It consists in evaluating $\Omega \wedge I\Omega$ as $\Omega_{k+\theta} \wedge I\Omega_{k+\theta}$.

$$\begin{pmatrix} m & 0 \\ 0 & I \end{pmatrix} (\_v_{k+1} - \_v_k) = h(1-\theta)\_Fl_k + h\theta Fl_{k+1} + (CT)^\top h\lambda_{k+1} \tag{10.4}$$ {eq:NE-thetaSche

Using the linearization

$$Fl_{k+1} = \_Fl_k + \nabla_v Fl(\_v_{k+1} - \_v_k)$$

system (10.4) leads to:

$$\left(\begin{pmatrix} m & 0 \\ 0 & I \end{pmatrix} - h\theta\nabla_v Fl \right) (\_v_{k+1} - \_v_k) = h\_Fl_k + (CT)^\top h\lambda_{k+1} \tag{10.5}$$

and we set $W = \left(\begin{pmatrix} m & 0 \\ 0 & I \end{pmatrix} - h\theta\nabla_v Fl \right)^{-1}$

10.3.1 Redaction note V. ACARY

From this point, I do not understand

Normally, F_{ext} et M_{ext} depends on V, X, Ω, R. Where are the Jacobian ? Where is the substitution of the nonlinear equation in q ?

$$(\Omega+\epsilon)\wedge I(\Omega+\epsilon) = \Omega\wedge I\Omega + \epsilon\wedge I\Omega + \Omega\wedge I\epsilon + O(\epsilon^2)$$

case $\epsilon = h * e_i$ leads to:

$$\frac{\partial(\Omega\wedge I\Omega)}{\partial e_i} = e_i\wedge I\Omega + \Omega\wedge Ie_i \tag{10.6}$$ {eq:NE_nablaFL1}

$$\nabla_v Fl = \begin{pmatrix} 0_{3x3} & 0_{3x3} \\ 0_{3x3} & \left(\frac{\partial(\Omega\wedge I\Omega)}{\partial e_i}\right)_{i=1,2,3} \end{pmatrix}$$

10.3.4 Building of the OSNSP

$$\boxed{\_v_{k+1} = W(h\_Fl_k) + W(CT)^\top h\lambda_{k+1} + \_v_k} \tag{10.7}$$ {NE_dis_explicit

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

$$\_ResiduFree_k = -h\_Fl_k$$

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

The relation 10.2 leads to the system:

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

Substitute $\_v_{k+1}$ using 10.7 leads:

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

Ones gets:

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

Solving the one step problem gives $h\lambda_{k+1}$, and from 10.7 we get $\_v_{k+1}$. At least, $\_v_{k+1}$ is used to compute $\dot{\_}q_{k+1}$, provided $\_q_{k+1}$.

10.4 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.

10.5 The Newton linearization applied to NewtonEuler formalisme

Let us define the residu:

$$\mathcal{R}_k(v, \lambda) = W(v - v_k) - hF_{ext} - (CT)^\top \lambda \tag{10.8}$$ {eq:newton_NE1_r

The linearized residu is:

$$\mathcal{R}_{L_k}(v, \lambda) = \mathcal{R}_k(v_k, \lambda_k) + W(v - v_k) - (CT)^\top (\lambda - \lambda_k) \tag{10.9}$$ {eq:newton_NE1_r

Let us define v_k^p and λ_k^p the current Newton iteration, initialized with v_k and λ_k. We are looking for v_k^{p+1} and λ_k^{p+1} such that $R_{L_k}(v_k^{p+1}, \lambda_k^{p+1}) = 0$. That is:

$$0 = \mathcal{R}_k(v_k^p, \lambda_k^p) + W(v_k^{p+1} - v_k^p) - (CT)^\top (\lambda_k^{p+1} - \lambda_k^p) \tag{10.10}$$ {eq:newton_NE1_e

That leads to:

$$v_k^{p+1} = v_k^p + W^{-1}[-\mathcal{R}_k(v_k^p, \lambda_k^p) + (CT)^\top (\lambda_k^{p+1} - \lambda_k^p)] \tag{10.11}$$ {eq:newton_NE1_e

The NSLAW is:

$$\dot{y}_k^{p+1} = CTv_k^{p+1} \tag{10.12}$$ {eq:newton_NE1_n

that leads to the OSNSP:

$$\dot{y}_k^{p+1} = (CT)W^{-1} (CT)^\top \lambda_k^{p+1} + (CT)[v_k^p - (CT)(CT)^\top \lambda_k^p - W^{-1}\mathcal{R}_k(v_k^p, \lambda_k^p)] \tag{10.13}$$ {eq:newton_NE1_o

10.5.1 Siconos implementation

The expression: $W(v_k^p - v_k) - hF_{ext}$ is saved in DS->residiFree.Moreau->computeResidu.
The expression: $\mathcal{R}_k(v_k^p, \lambda_k^p) = W(v_k^p - v_k) - -hF_{ext} + (CT)^\top (\lambda_k^p)$ is saved in DS->workFree.
The expression: $vfree = \dot{y}_k^p - W^{-1} residufree$ is saved in DS->workFree. Moreau->computeFreeState.
The computation: $y_k^{p+1} = vfree + W^{-1}\lambda_k^{p+1}$ is done in OSI::updateState.
The OSNSP is :

$$\dot{y}_k^{p+1} = (CT)W^{-1} (CT)^\top \lambda_k^{p+1} + (CT)vfree + nslaweffect \tag{10.14}$$

Chapter 11

NewtonEulerR: computation of $\nabla_q H$

11.0.2 Gradient computation, case of NewtonEuler with quaternion

In the section, q is the quaternion of the dynamical system.

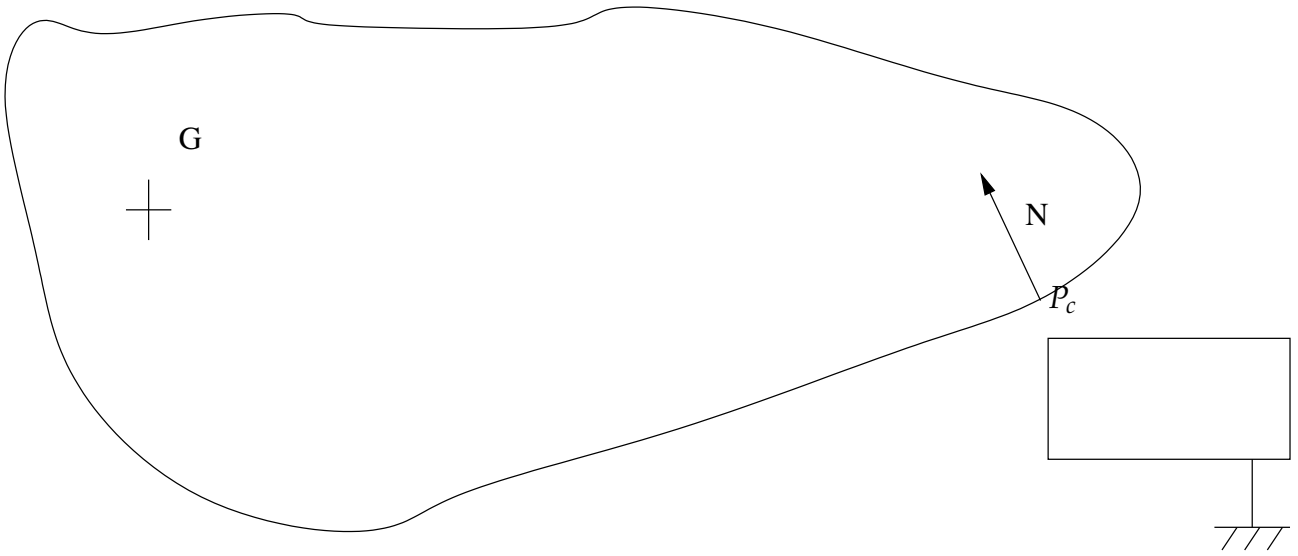


Figure 11.1: Impact of one DS.

{figCase}

The normal vector N is view as a constant.

$$\tilde{h}(q) = P_c(\frac{q}{\|q\|})$$

$$^t\nabla h(q)(\delta q) = \lim_{e\to 0} \frac{(\tilde{h}(q+e\delta q) - \tilde{h}(q)).N}{e}$$

$\nabla_q h$ consist in computing $P_c(\frac{q+\delta q}{\|q+\delta q\|}) - P_c(q)$.

$$GP(q) = qG_0P_0\ ^c q$$

$$GP(\frac{q+\delta q}{\|q+\delta q\|}) = (q+\delta q)G_0P_0\ ^c(q+\delta q)\frac{1}{\|q+\delta q\|^2}$$

$$= (q+\delta q)\ ^c qGP(q)q\ ^c(q+\delta q)\frac{1}{\|q+\delta q\|^2}$$

$$= ((1,0,0,0) + \delta q\ {}^c q) GP(q) ((1,0,0,0) + q\ {}^c \delta q) \frac{1}{\|q+\delta q\|^2}$$

$$= GP(q) + \delta q\ {}^c q GP(q) + GP(q) q\ {}^c \delta q + 0(\delta q)^2 \frac{1}{\|q+\delta q\|^2}$$

So, because G is independant of q:

$$P(\frac{q+\delta q}{\|q+\delta q\|}) - P(q) = qGP(\frac{q+\delta q}{\|q+\delta q\|}) - GP(q) = \delta q\ {}^c q GP(q) + GP(q) q\ {}^c \delta q + 0(\delta q)^2 + GP(q) \frac{1}{\|q+\delta q\|^2}$$

For the directional derivation, we chose $\delta q = \epsilon * (1,0,0,0)$. using a equivalent to $\frac{1}{1+\epsilon}$

$$\lim_{\epsilon \to 0} \frac{P(\frac{q+\delta q}{\|q+\delta q\|}) - P(q)}{\epsilon} = {}^c q GP(q) + GP(q) q - 2 q_i GP(q)$$

For the directional derivation, we chose $\delta q = \epsilon * (0,1,0,0) = \epsilon * e_i$

$$\lim_{\epsilon \to 0} \frac{P(\frac{q+\delta q}{\|q+\delta q\|}) - P(q)}{\epsilon} = e_i\ {}^c q GP(q) - GP(q) q e_i - 2 q_i GP(q)$$

Application to the NewtonEulerRImpact:

$$H : \mathbb{R}^7 \to \mathbb{R}$$

$$\nabla_q H \in \mathcal{M}^{1,7}$$

$$\nabla_q H = \begin{pmatrix} N_x \\ N_y \\ N_z \\ ({}^c q GP(q) + GP(q) q - 2 q_0 GP(q)).N \\ (e_2\ {}^c q GP(q) - GP(q) q e_2 - 2 q_1 GP(q)).N \\ (e_3\ {}^c q GP(q) - GP(q) q e_3 - 2 q_2 GP(q)).N \\ (e_4\ {}^c q GP(q) - GP(q) q e_4 - 2 q_3 GP(q)).N \end{pmatrix}$$

11.0.3 Ball case

It is the case where $GP = -N$: for $e2$:

$$(0,1,0,0).(q_0, -\underline{p}).(0,-N) =$$

$$\left(\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} .\underline{p}, \begin{pmatrix} q_0 \\ 0 \\ 0 \end{pmatrix} - \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} * \underline{p} \right) .(0,-N) =$$

$$\left(?, -\underline{p}_x\ N - \left(\begin{pmatrix} q_0 \\ 0 \\ 0 \end{pmatrix} - \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} * \underline{p} \right) * N \right) =$$

and:

$$(0,-N).(q_0, \underline{p}).(0,1,0,0) =$$

$$(N.\underline{p}, -q_0 N - N * \underline{p}).(0,1,0,0) =$$

$$\left(?, (N.\underline{p}) \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} * (q_0 N + N * \underline{p}) \right) =$$

$$\left(?, (N.\underline{p}) \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + q_0 \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} * N + \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} * (N * \underline{p}) \right)$$

sub then and get the resulting vector.N:

$$\left[-\underline{p}_x \, N - N.\underline{p} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + () * N - \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} * (N * \underline{p}) \right] .N =$$

$$-\underline{p}_x - N_x N.\underline{p} + 0 - (\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} * (N * \underline{p})).N =$$

using $a * (b * c) = b(a.c) - c(a.b)$ leads to

$$-q_1 - N_x N.\underline{p} - (q_1 \, N - N_x \, \underline{p}).N =$$

$$-q_1 - N_x N.\underline{p} - q_1 + N_x N.\underline{p} = -2q_1$$

for $e1 = (1, 0, 0, 0)$:

$$(q_0, -\underline{p}).(0, -N) = (?, -q_0 N + \underline{p} * N)$$

$$(0, -N).(q_0, \underline{p}) = (?, -q_0 N - \underline{p} * N)$$

So

$$\nabla_q H = \begin{pmatrix} N_x \\ N_y \\ N_z \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

11.0.4 Case FC3D: using the local frame and momentum

$$\begin{pmatrix} m\dot{V} \\ I\dot{\Omega} + \Omega I \Omega \end{pmatrix} = \begin{pmatrix} Fect + R \\ Mext_{R_{obj}} + (R * PG)_{R_{obj}} \end{pmatrix}$$

with * vectoriel product, R reaction in the globla frame. P the point of contact. r is the reaction in the local frame. $M_{R_{obj} to R_{abs}} = M^t_{R_{abs} to R_{obj}} r = R$ with:

$$M_{R_C to R_{abs}} = \begin{pmatrix} nx & t_1 x & t_2 x \\ ny & t_1 y & t_2 y \\ nz & t_1 z & t_2 z \end{pmatrix}$$

we have :

$$\begin{pmatrix} R \\ (R * PG)_{R_{obj}} \end{pmatrix} = \begin{pmatrix} I_3 \\ M_{R_{abs} to R_{obj}} N_{PG} \end{pmatrix} .R$$

$$= \begin{pmatrix} I_3 \\ M_{R_{abs} to R_{obj}} N_{PG} \end{pmatrix} .M_{R_{obj} to R_{abs}} r$$

$$N_{PG} = \begin{pmatrix} 0 & PG_z & -PG_y \\ -PG_z & 0 & PG_x \\ PG_y & -PG_X & 0 \end{pmatrix}$$

that is:

$$\begin{pmatrix} m\dot{V} \\ I\dot{\Omega} + \Omega I \Omega \end{pmatrix} = \begin{pmatrix} M_{R_C to R_{abs}} \\ M_{R_{abs} to R_{obj}} N_{PG} M_{R_C to R_{abs}} \end{pmatrix} r$$

So $jachqt = MN$

11.0.5 Case FC3D: using the local frame local velocities

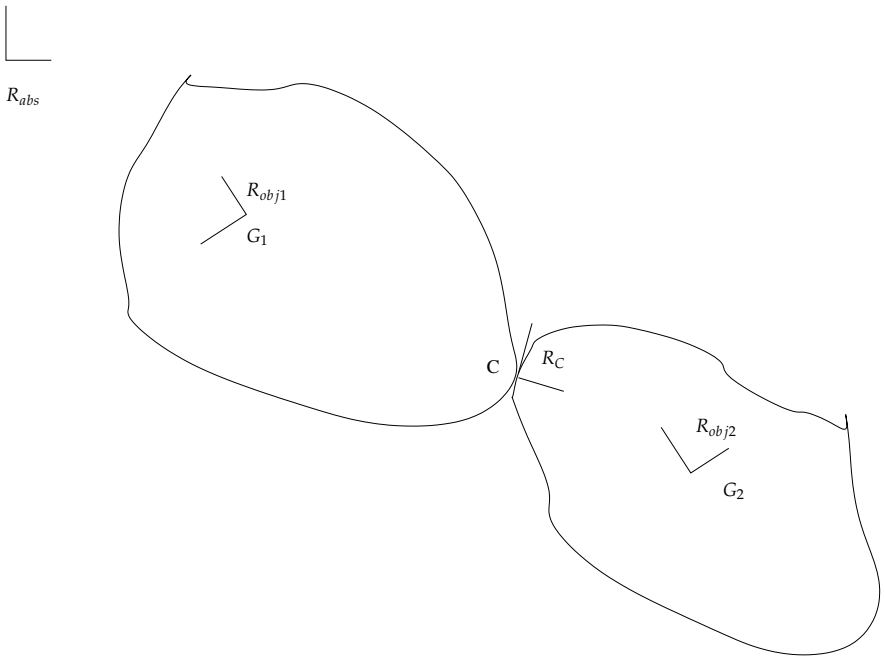


Figure 11.2: Two objects colliding. {figCase}

We are looking for an operator named CT such that:

$$V_C = \begin{pmatrix} V_N \\ V_T \\ V_S \end{pmatrix}_{R_C} = CT \begin{pmatrix} V_{G1\ R_{abs}} \\ \Omega_{1\ R_{obj1}} \\ V_{G2\ R_{abs}} \\ \Omega_{2\ R_{obj2}} \end{pmatrix}$$

$$V_c = V_{G1\ R_{abs}} + w_1 * G_1P_{\ R_{abs}} - (V_{G2\ R_{abs}} + w_2 * G_1P_{\ R_{abs}})$$

where w_1 and w_2 are given in R_{abs}. We note $M_{R_{obj1}toR_{abs}}$ the matrice converting the object 1 coordinate to the absolute coordinate. We note N_{GP} the matrice such that $w_1 * G_1P_{\ R_{abs}} = N_{GC}w_1$. Endly, we note $M_{R_{abs}toR_C}$ converting the absolute coordinate to the R_C frame. we get:

$$CT = M_{R_{abs}toR_C} \begin{pmatrix} I_3 & N_{G_1C}M_{R_{obj1}toR_{abs}} & -I_3 & -N_{G_2C}M_{R_{obj2}toR_{abs}} \end{pmatrix}$$

11.0.5.a Expression of $M_{R_{obj1}toR_{abs}}$

Using quaternion, we get :

$$M_{R_{obj1}toR_{abs}} = \begin{pmatrix} q\begin{pmatrix}1\\0\\0\end{pmatrix}{}^cq & q\begin{pmatrix}0\\1\\0\end{pmatrix}{}^cq & q\begin{pmatrix}0\\0\\1\end{pmatrix}{}^cq \end{pmatrix} \quad (11.1)$$ {eq:newton_Mobjt

11.0.5.b Expression of N_1

$$N_{GC} = \begin{pmatrix} 0 & G_1C_z & -G_1C_y \\ -G_1C_z & 0 & G_1C_x \\ G_1C_y & -G_1C_X & 0 \end{pmatrix}$$

Chapter 12

Projection On constraints

12.0.6 Velocity formulation

The first step consists in doing a velocity formulation of the system:

$$\begin{array}{l} M\dot{v} = F_{ext} + B\lambda \\ \dot{q} = Tv \\ y = h(q) \\ NSLAW(y, \lambda, ...) \end{array} \tag{12.1}$$ {NE_Dyn1}

The constraint $\dot{q} = Tv$ is suffisiant to keep a normal quaternion. Because of the speed formulation, $h(q)$ could violate the NSLAW. A solution coul be to add a formulation in position. We must underline that the constraints $\mid Q \mid = 1$ is implicit in this system. Endeed, the direction $\dot{q} = Tv$ is tangential to the sphere.

12.0.7 Posion formulation

It consists in writting a position formulation on the system:

$$h(q) = \begin{pmatrix} HI(q) \\ HE(q) \end{pmatrix} \tag{12.2}$$ {NE_Dyn1}

12.0.7.a Approach using q

We are looking for q_1 from q_0:

$$q_1 = q_0 + \nabla HI\Lambda_I + \nabla HE\Lambda_E \tag{12.3}$$

Assume that $h(q_0)$ doesn't satisfy the constraints, ie $HI(q_0) \ngeq 0$ or $HE(q_0) \neq 0$). Linearize h leads to:

$$0 \leq HI(q_0) + \nabla^t HI(\nabla HI\Lambda_I + \nabla HE\Lambda_E) \perp \Lambda_I \geq 0 \tag{12.4}$$

$$0 = HE(q_0) + \nabla^t HE(\nabla HI\Lambda_I + \nabla HE\Lambda_E) \tag{12.5}$$

The getting system could be written has a MLCP:

$$C \ni h(q_0) + \nabla^t h(\nabla h\Lambda), \Lambda \in C^* \tag{12.6}$$

In the case of a quaternion Q for the rotation representation, it is noteworthy that this system doesn't deal with the constraints $\mid Q \mid = 1$. Thus, the direction (q_1, q_0) can be normal to this constraint, in that case this approach doesn't work. (It happens in practice) The solution that consists in normaliaed q after this formulation is not convenient because, it could be incompatible with $\mid Q \mid = 1$. A better approach is to add this constraint.

The constraint $\mid Q \mid = 1$ in the system HE:

$$\tilde{HE}(q) = \begin{pmatrix} HE(q) \\ \mid Q \mid -1 \end{pmatrix} \tag{12.7}$$

The formulation described above can be done.

12.0.7.b Approach using V

It consists in building the OSNSP using CT instead of C.

$$h(q_1) = h(q_0) + \nabla^t H \delta q \tag{12.8}$$ {NE_projV}

ie:

$$h(q_1) = h(q_0) + \nabla^t HTV \tag{12.9}$$ {NE_projV}

We are looking for q_1 such that:

$$q_1 - q_0 = \nabla H \Lambda \tag{12.10}$$

We have

$$\delta q = TV, \qquad {}^tT\delta q = {}^t TTV, \qquad ({}^tTT)^{-1}\,{}^tT\delta q = V$$

ie

$$h(q_1) = h(q_0) + {}^t \nabla_q hT({}^tTT)^{-1}\,{}^tT\nabla_q h\Lambda \tag{12.11}$$

With $C = {}^t \nabla_q h$ leading to the prolem:

$$K \ni h(q_0) + CT\,({}^tTT)^{-1}\,{}^t(CT)\Lambda \in K^* \tag{12.12}$$

Chapter 13

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

$$\mu\frac{d^2u(t)}{dt^2}+\zeta\frac{du(t)}{dt}+\kappa u(t)=f_v(t), \qquad \text{if} \quad u(t)>c(t). \tag{13.1}$$ {eq:sols}

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

$$v(t^+)=\frac{dc}{dt}-r\left(v(t^-)-\frac{dc}{dt}\right)=(1+r)\frac{dc}{dt}-rv(t^-), \quad \text{if} \quad u(t)=c(t). \tag{13.2}$$ {eq:il}

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 13.1 is presented the schematic diagram of the physical cam-follower system. In Figure 13.1.a for $t=0$, 13.1.b for $t=\beta$, and 13.1.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.

13.0.8 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 (13.1) and (13.2) can be expressed as (the argument t will not be explicitly written)

$$\mu\frac{d^2\hat{u}}{dt^2}+\zeta\frac{d\hat{u}}{dt}+\kappa\hat{u}=f_v-\left(\mu\frac{d^2c}{dt^2}+\zeta\frac{dc}{dt}+\kappa c\right) \equiv \hat{f}, \qquad \text{if} \quad \hat{u}>0. \tag{13.3}$$ {eq:trans}

$$\hat{v}^+ = -r\hat{v}^-, \quad \text{if} \quad \hat{u}=0. \tag{13.4}$$

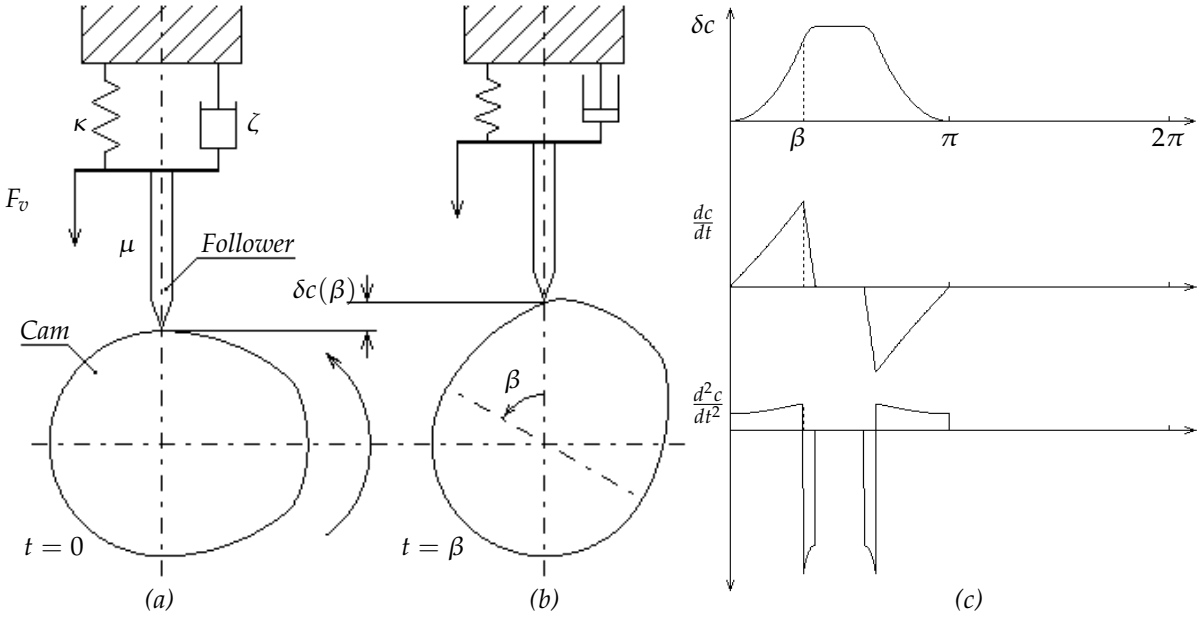


Figure 13.1: Cam-Shaft's schematics. *(a)* t=0. *(b)* t=β. *(c)* Constraint position $\delta c(t)$, velocity $\frac{dc}{dt}(t)$ and acceleration $\frac{d^2c}{dt}(t^2)$. {Fig:cam-shaft}

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

$$M(q)\ddot{q} + Q(q,\dot{q}) + F(\dot{q},q,t) = F_{ext}(t) + R \tag{13.5}$$ {eq:lag1}

From the (13.3) we can derive all of the terms which define a Lagrangian NSDS. In our case the model is completely linear:

$$\begin{aligned} q &= \begin{bmatrix} \hat{u} \end{bmatrix} \\ M(q) &= \begin{bmatrix} \mu \end{bmatrix} \\ Q(q,\dot{q}) &= \begin{bmatrix} 0 \end{bmatrix} \\ F(q,\dot{q}) &= \begin{bmatrix} \zeta \end{bmatrix}\dot{q} + \begin{bmatrix} \kappa \end{bmatrix} q \\ F_{ext} &= \begin{bmatrix} \hat{f} \end{bmatrix} \end{aligned} \tag{13.6}$$ {eq:lag2}

The unilateral constraint requires that:

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

so we can obtain

$$\begin{aligned} y &= H^T q + b \\ H^T &= \begin{bmatrix} 1 \end{bmatrix} \\ b &= 0 \end{aligned} \tag{13.7}$$ {eq:constr}

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

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

The unilataral contact law may be formulated as follow:

$$0 \leq y \perp \lambda \geq 0 \tag{13.8}$$ {eq:17}

and the Newton's impact law:

$$\text{If } y = 0, \dot{y}^+ = -r\dot{y}^- \tag{13.9}$$ {eq:17}

13.0.9 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;

}
```

13.0.10 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. 13.2 and 13.3 show the time simulations for different values of the cam rotational speed and Fig. 13.4 show the chaotic attractor at $rpm = 660$ for impact and stroboscopic Poincarè sections.

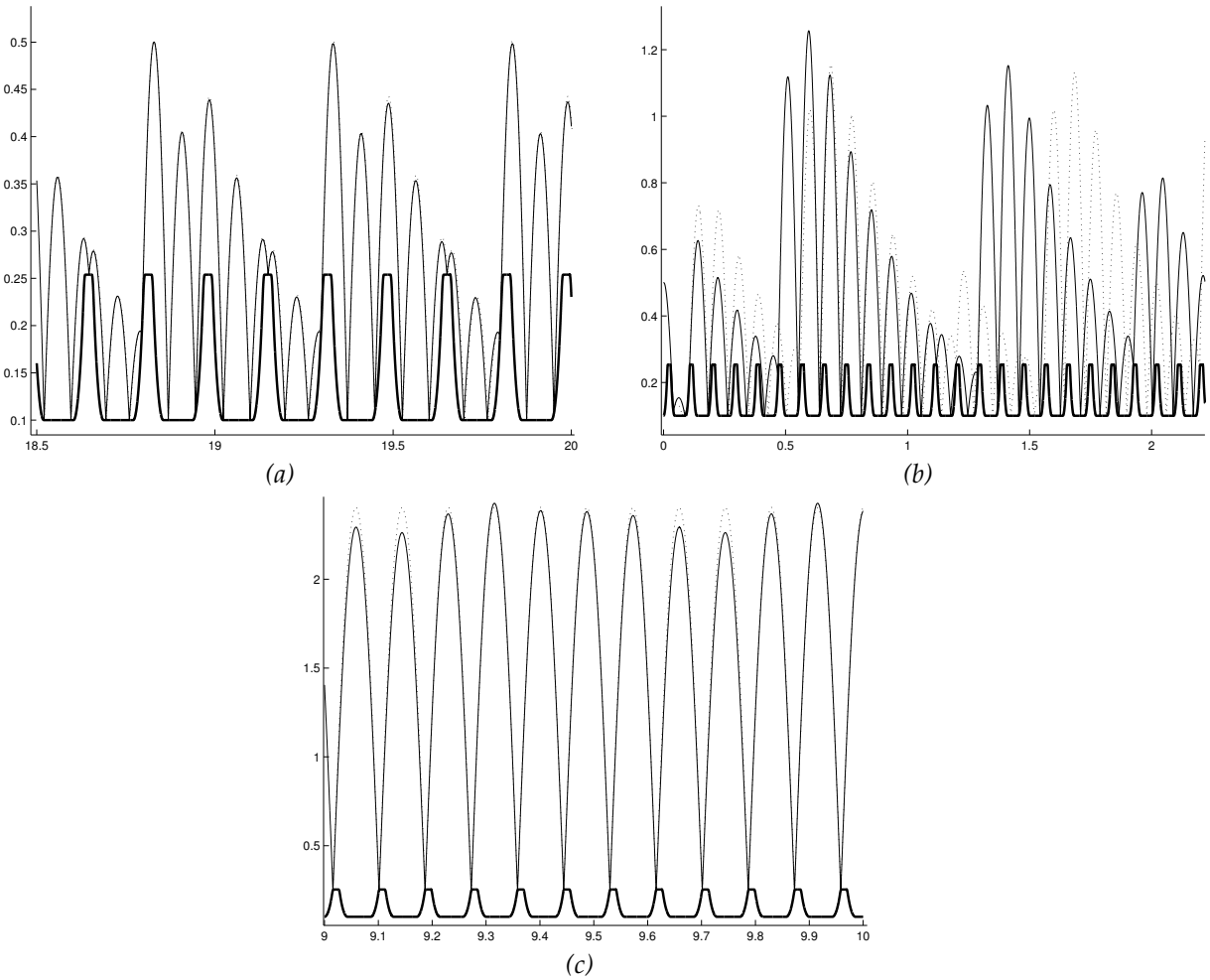


Figure 13.2: Time series using SICONOS platform. Time-stepping scheme (continuous line). Event-driven scheme (dashed line) *(a)* rpm=358. *(b)* rpm=660. *(c)* rpm=700. {Fig:time_compari

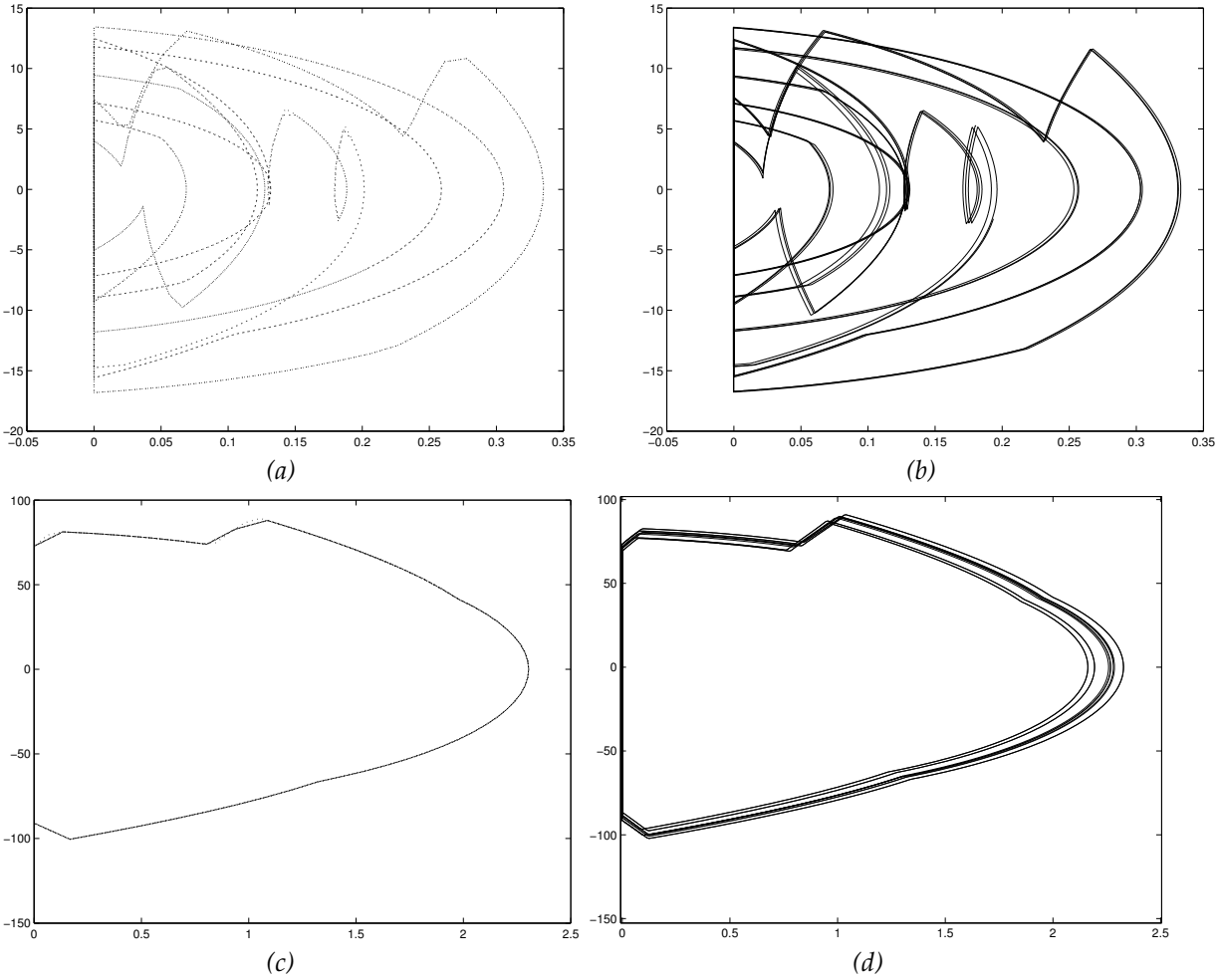


Figure 13.3: State space comparison using SICONOS platform. *(a)* rpm=358. Event Driven *(b)* rpm=358. Time Stepping ($h = 1e^{-4}$)*(c)* rpm=700. Event Driven *(d)* rpm=700. Time Stepping ($h = 1e^{-4}$) {Fig:state_compa

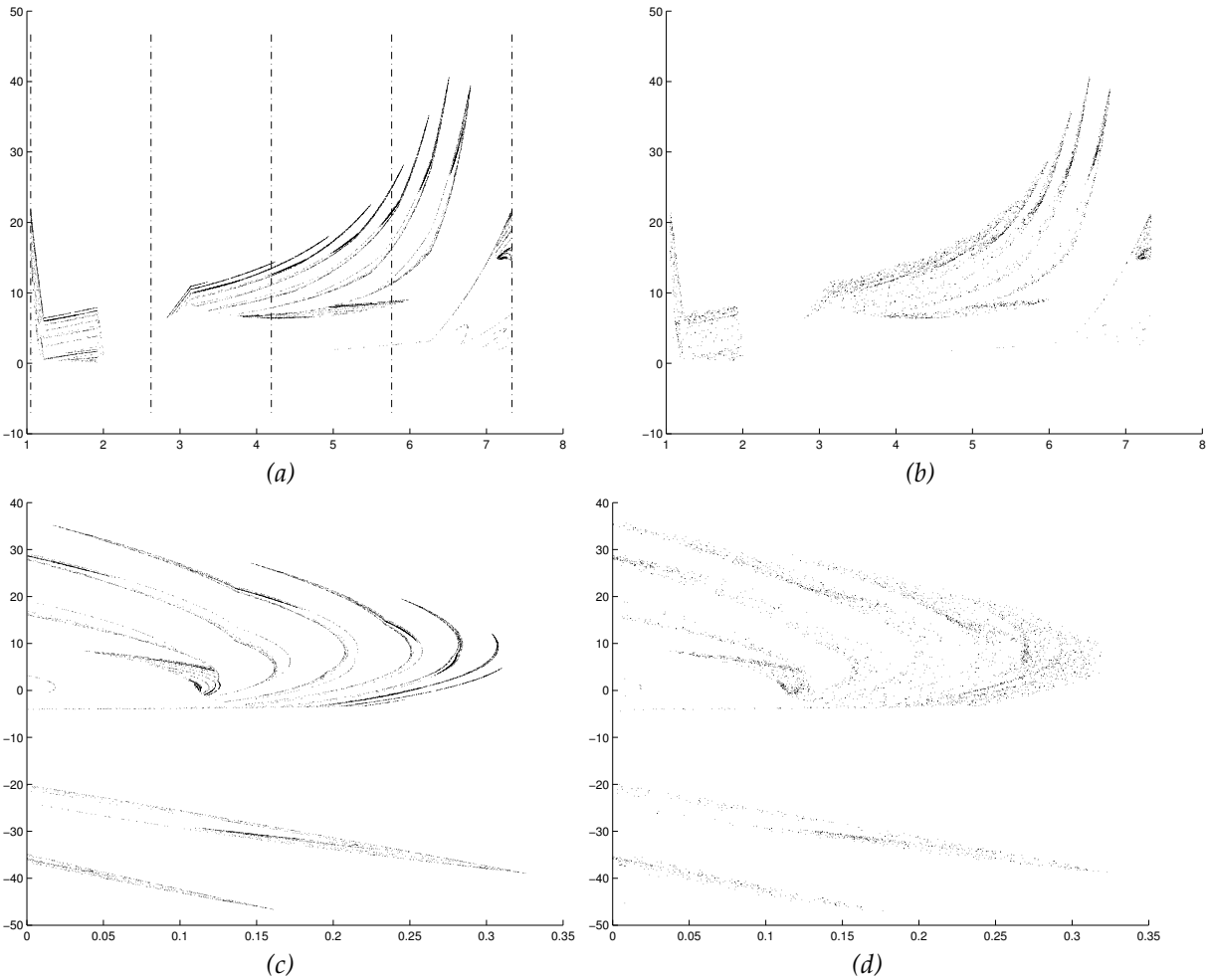


Figure 13.4: Attractors comparison using SICONOS platform at rpm=660. *(a)* Impact map. (Event Driven) *(b)* Impact Map. Time Stepping ($h = 1e^{-4}$)*(a)* Stroboscopic map. (Event Driven) *(b)* Stroboscopic Map. Time Stepping ($h = 1e^{-4}$)

{Fig:attractor_c

Chapter 14

Quartic Formulation

14.0.11 Slidding ?

It consists in finding $\alpha > 0$ and $R \in \partial K_\mu$ such that $-\alpha \begin{pmatrix} 0 \\ R_T \end{pmatrix} = MR + q$. That is :

$$\left[M + \begin{pmatrix} 0 & 0 & 0 \\ 0 & \alpha & 0 \\ 0 & 0 & \alpha \end{pmatrix} \right] R + q = 0 \tag{14.1}$$ {eq_quartic1}

14.0.11.a R_T is on a conic

The first line of the system 14.1 and the $R \in \partial K_\mu$ is the intersection between a plan and a cone in $\mathbb{R}^3$, endeed:

$$\begin{array}{l} \mu R_N = \| R_T \| \\ \frac{M_{11}}{\mu} \| R_T \| = -q_1 - M_{12}R_{T1} - M_{13}R_{T2} \end{array} \tag{14.2}$$ {eq_quartic2}

That is:

$$\begin{array}{l} \mu^2 R_N^2 = (R_{T1}^2 + R_{T1}^2) \\ \frac{M_{11}^2}{\mu^2}(R_{T1}^2 + R_{T1}^2) = (-q_1 - M_{12}R_{T1} - M_{13}R_{T2})^2 \end{array} \tag{14.3}$$ {eq_quartic2}

That means that R_T is contained in a conic, focus and directrice are:

$$\begin{array}{l} \mathcal{D} : q_1 + M_{12}R_{T1} + M_{13}R_{T2} = 0 \\ focus : \mathcal{O} \\ \frac{M_{11}^2}{\mu^2} Dist(\mathcal{O}, R_T)^2 = Dist(\mathcal{D}, R_T)^2 (M_{12}^2 + M_{13}^2) \\ \frac{Dist(\mathcal{O}, R_T)}{Dist(\mathcal{D}, R_T)} = \frac{\mu\sqrt{(M_{12}^2 + M_{13}^2)}}{M_{11}} = e \end{array} \tag{14.4}$$ {eq_quartic3}

The parametric equation is:

$$\begin{array}{l} R_{T1} = rcos(\theta) \\ R_{T2} = rsin(\theta) \\ r = \frac{p}{1 + ecos(\theta - \phi)} \end{array} \tag{14.5}$$ {eq_quartic4}

With p an simple expression of M_{11}, M_{12}, M_{13}, and ϕ a constant angle between $\mathcal{D}$ and (O, R_{T1})

14.0.11.b The two last line of the system 14.1

$$\frac{\| R_T \|}{\mu} \tilde{M}_{1.} + \left(\tilde{M} + \begin{pmatrix} \alpha & 0 \\ 0 & \alpha \end{pmatrix} \right) R_T + \tilde{q} = 0 \tag{14.6}$$ {eq_quartic5}

$\tilde{M}$ is symetric, so it exists a unitary matrix V such that $V\tilde{M}V^T = \begin{pmatrix} d_1 & 0 \\ 0 & d_2 \end{pmatrix}$. One can get:

$$\frac{\| R_T \|}{\mu} V\tilde{M}_{1.} + V\left(\tilde{M} + \begin{pmatrix} \alpha & 0 \\ 0 & \alpha \end{pmatrix}\right) V^T V R_T + V\tilde{q} = 0 \tag{14.7}$$ {eq_quartic6}

Rename:

$$\frac{\| \bar{R}_T \|}{\mu} \bar{M}_{1.} + \begin{pmatrix} d_1+\alpha & 0 \\ 0 & d_2+\alpha \end{pmatrix} \overline{R}_T + \bar{q} = 0 \tag{14.8}$$ {eq_quartic7}

In the plan, either V is a rotation or a symetrie. So $\bar{R}_T = VR_T$ is a conic with the same focus and a rotated directrice, it means that it exists ϕ_1 such that :

$$\begin{array}{l} \bar{R}_{T1} = rcos(\theta) \\ \bar{R}_{T2} = rsin(\theta) \\ r = \frac{p}{1+ecos(\theta-\phi_1)} \end{array} \tag{14.9}$$ {eq_quartic8}

The equation 14.8 is :

$$\begin{array}{l} (d_1+\alpha)\bar{R}_{T1} = -\bar{q}_1 + a_1 \| R_T \| \\ (d_2+\alpha)\bar{R}_{T2} = -\bar{q}_2 + a_2 \| R_T \| \end{array} \tag{14.10}$$ {eq_quartic9}

The case ($\bar{R}_{T1} = 0$ or $\bar{R}_{T2} = 0$) has to be examine. We try to eliminate $alpha$:

$$\begin{array}{l} d_1\bar{R}_{T1}\bar{R}_{T2} + \alpha\bar{R}_{T1}\bar{R}_{T2} = -\bar{q}_1\bar{R}_{T2} + a_1\bar{R}_{T2} \| R_T \| \\ d_2\bar{R}_{T1}\bar{R}_{T2} + \alpha\bar{R}_{T1}\bar{R}_{T2} = -\bar{q}_2\bar{R}_{T1} + a_2\bar{R}_{T1} \| R_T \| \end{array} \tag{14.11}$$ {eq_quartic10}

that leads to:

$$(d_1-d_2)\bar{R}_{T1}\bar{R}_{T2} = -\bar{q}_1\bar{R}_{T2} + \bar{q}_2\bar{R}_{T1} + (a_1\bar{R}_{T2} - a_2\bar{R}_{T1}) \| R_T \| \tag{14.12}$$ {eq_quartic10}

The parametric expression of $\bar{R}_T$ leads to:

$$\begin{array}{l} (d_1-d_2)r^2cos(\theta)sin(\theta) = -\bar{q}_1 rsin(\theta) + \bar{q}_2 rcos(\theta) + r(a_1 rsin(\theta) - a_2 rcos(\theta)) \\ \text{ie:}(d_1-d_2)rcos(\theta)sin(\theta) = -\bar{q}_1 sin(\theta) + \bar{q}_2 cos(\theta) + r(a_1 sin(\theta) - a_2 cos(\theta)) \end{array} \tag{14.13}$$ {eq_quartic11}

with the expression of r:

$$\begin{array}{l} (d_1-d_2)\frac{p}{1+ecos(\theta-\phi_1)}cos(\theta)sin(\theta) = \\ -\bar{q}_1 sin(\theta) + \bar{q}_2 cos(\theta) + \frac{p}{1+ecos(\theta-\phi_1)}(a_1 sin(\theta) - a_2 cos(\theta)) \\ \\ \text{ie:}(d_1-d_2)pcos(\theta)sin(\theta) = \\ (1+ecos(\theta-\phi_1))(-\bar{q}_1 sin(\theta) + \bar{q}_2 cos(\theta)) + p(a_1 sin(\theta) - a_2 cos(\theta)) \\ \\ \text{ie:}(d_1-d_2)pcos(\theta)sin(\theta) = \\ (1+e(cos(\theta)cos(\phi_1) + sin(\theta)sin(\phi_1)))(-\bar{q}_1 sin(\theta) + \bar{q}_2 cos(\theta)) + p(a_1 sin(\theta) - a_2 cos(\theta)) \\ \\ \text{ie:}(d_1-d_2)pcos(\theta)sin(\theta) + \\ (1+ecos(\theta)cos(\phi_1) + esin(\theta)sin(\phi_1))(\bar{q}_1 sin(\theta) - \bar{q}_2 cos(\theta)) + p(-a_1 sin(\theta) + a_2 cos(\theta)) = 0 \end{array} \tag{14.14}$$ {eq_quartic12}

rename :

$$Acos(\theta)^2 + Bsin(\theta)^2 + Csin(\theta)cos(\theta) + Dsin(\theta) + Ecos(\theta) = 0 \tag{14.15}$$ {eq_quartic13}

with

$$\begin{array}{l} A = -e\bar{q}_2 cos(\phi_1) \\ B = e\bar{q}_1 sin(\phi_1) \\ C = (d_1-d_2)p + ecos(\phi_1)\bar{q}_1 - esin(\phi_1)\bar{q}_2 \\ D = \bar{q}_1 - pa_1 \\ E = -\bar{q}_2 + pa_2 \end{array} \tag{14.16}$$ {eq_quartic12}

rename : Using the following set of unknown :

$$\begin{array}{l} t = tan(\theta/2) \\ sin(\theta) = \frac{2t}{1+t^2} \\ cos(\theta) = \frac{1-t^2}{1+t^2} \end{array} \tag{14.17}$$ {eq_quartic14}

leads to:

$$\begin{array}{l} A\frac{(1-t^2)^2}{1+t^2} + B\frac{4t^2}{1+t^2} + C\frac{2t(1-t^2)}{1+t^2} + D2t + E(1-t^2) = 0 \\ \text{ie:} A(1-t^2)^2 + 4Bt^2 + C2t(1-t^2) + 2Dt(1+t^2) + E(1-t^2)(1+t^2) = 0 \end{array} \tag{14.18}$$ {eq_quartic13}

ie:$P_4 = A - E \qquad P_3 = -2C + 2D \qquad P_2 = 4B - 2A \qquad P_1 = 2C + 2D \qquad P_0 = A + E$

Finally, we get 4 possible values for R_T, checking the sign of α and R_N selects the solutions.

14.0.11.c case $R_{T12} = 0$

From 14.10, R_{T1} leads to:

$$\| R_T \| = |\bar{R}_{T2}| = \frac{\bar{q}_1}{a_1}$$

$$\bar{R}_T = \begin{pmatrix} 0 \\ \pm\frac{\bar{q}_1}{a_1} \end{pmatrix} \tag{14.19}$$ {eq_quartic14}

From 14.10, R_{T2} leads to:

$$\| R_T \| = |\bar{R}_{T1}| = \frac{\bar{q}_2}{a_2}$$

$$\bar{R}_T = \begin{pmatrix} \pm\frac{\bar{q}_2}{a_2} \\ 0 \end{pmatrix} \tag{14.20}$$ {eq_quartic14}

From $\bar{R}_T$, we have to check the coherence with the equation 14.9. If it is on the conic, we compute R, and the sign condition of the equation 14.1 must be check.