# Developer's Notes

Siconos Development Team

June 30, 2009

# Chapter 1

# Dynamical Systems implementation in Siconos.

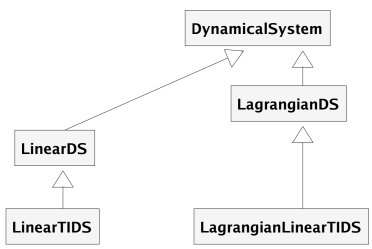
| | |
|---|---|
| author | F. Pérignon |
| date | November 7, 2006 |
| version | Kernel 1.3.0 |

## 1.1 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.

## 1.2 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:




## 1.3 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 ...).

### 1.3.1 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.**

### 1.3.2 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?**

## 1.4 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 ...)**

## 1.5 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 2

# Interactions

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

## 2.1 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.**

## 2.2 Class Diagram

## 2.3 Description

```
2.3.1 Redaction note F. PERIGNON
      review of interactions (for EventDriven implementation) 17th May 2006.
```

- **variable *nInter* renamed in *interactionSize*: represents the size of $y$ and $\lambda$. 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 3

# Notes on the Non Smooth Dynamical System construction

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

## 3.1 Introduction

## 3.2 Class Diagram

## 3.3 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).**

## 3.4 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 4

# OneStepIntegrator and derived classes.

| | |
|---|---|
| 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 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.

## 4.2 Class Diagram

## 4.3 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
- ...

## 4.4 Construction

Each constructor must:

- 

### 4.4.1 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.

### 4.4.2 Lsodar

Required data:

Optional:

Always allocated in constructor:

# Chapter 5

# 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

$$\mu \frac{d^2 u(t)}{dt^2} + \zeta \frac{du(t)}{dt} + \kappa u(t) = f_v(t), \qquad \textbf{if} \quad u(t) > c(t). \tag{5.1}$$

where $\mu$, $\zeta$ and $\kappa$ 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

$$v(t^+) = \frac{dc}{dt} - r\left(v(t^-) - \frac{dc}{dt}\right) = (1+r)\frac{dc}{dt} - rv(t^-), \ \textbf{if} \quad u(t) = c(t). \tag{5.2}$$

where $v(t^+)$ and $v(t^-)$ are the post and pre impact velocities respectively, $\frac{dc}{dt}$ is the velocity vector of the cam at the contact point with the follower, and $r \in [0,1]$ is the restitution coefficient to model from plastic to elastic impacts. In Figure 5.1 is presented the schematic diagram of the physical cam-follower system. In Figure 5.1.a for $t = 0$, 5.1.b for $t = \beta$, and 5.1.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.

### 5.0.3 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 (5.1) and (5.2) can be expressed as (the argument $t$ will not be explicitly written)

$$\mu \frac{d^2\hat{u}}{dt^2} + \zeta \frac{d\hat{u}}{dt} + \kappa \hat{u} = f_v - \left(\mu \frac{d^2 c}{dt^2} + \zeta \frac{dc}{dt} + \kappa c\right) \quad \equiv \quad \hat{f}, \qquad \textbf{if} \quad \hat{u} > 0. \tag{5.3}$$

$$\hat{v}^+ \quad = \quad -r\hat{v}^-, \ \textbf{if} \quad \hat{u} = 0. \tag{5.4}$$

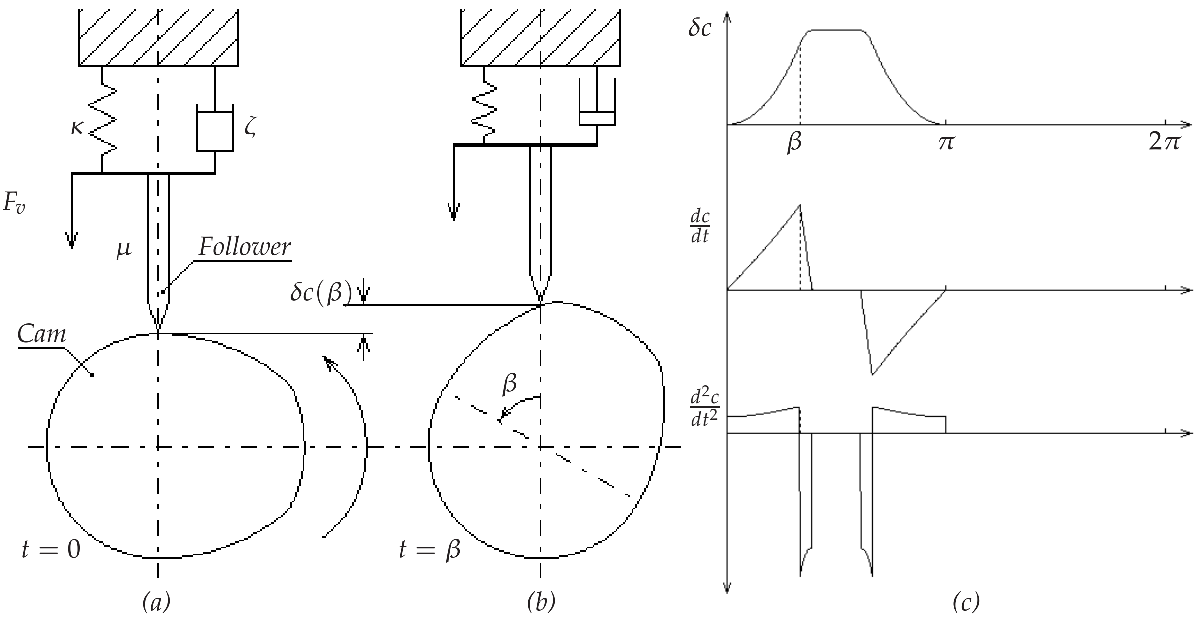



Figure 5.1: Cam-Shaft's schematics. *(a)* t=0. *(b)* t=$\beta$. *(c)* Constraint position $\delta c(t)$, velocity $\frac{dc}{dt}(t)$ and acceleration $\frac{d^2c}{dt}(t^2)$.

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

$$M(q)\ddot{q} + Q(q,\dot{q}) + F(\dot{q},q,t) = F_{ext}(t) + R \tag{5.5}$$

**From the (5.3) we can derive all of the terms which define a Lagrangian NSDS. In our case the model is completely linear:**

$$\begin{array}{rcl} q & = & \left[\ \hat{u}\ \right] \\ M(q) & = & \left[\ \mu\ \right] \\ Q(q,\dot{q}) & = & \left[\ 0\ \right] \\ F(q,\dot{q}) & = & \left[\ \zeta\ \right]\dot{q} + \left[\ \kappa\ \right] q \\ F_{ext} & = & \left[\ \hat{f}\ \right] \end{array} \tag{5.6}$$

**The unilateral constraint requires that:**

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

**so we can obtain**

$$\begin{array}{rcl} y & = & H^T q + b \\ H^T & = & \left[\ 1\ \right] \\ b & = & 0 \end{array} \tag{5.7}$$

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

$$R = H\lambda, \qquad \textbf{with } H = \left[\ 1\ \right]$$

**The unilataral contact law may be formulated as follow:**

$$0 \leq y \perp \lambda \geq 0 \tag{5.8}$$

**and the Newton's impact law:**

$$\textbf{If } y = 0, \dot{y}^+ = -r\dot{y}^- \tag{5.9}$$

### 5.0.4 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;**

}

### 5.0.5 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. 5.2 and 5.3 show the time simulations for different values of the cam rotational speed and Fig. 5.4 show the chaotic attractor at $rpm = 660$ for impact and stroboscopic Poincarè sections.**

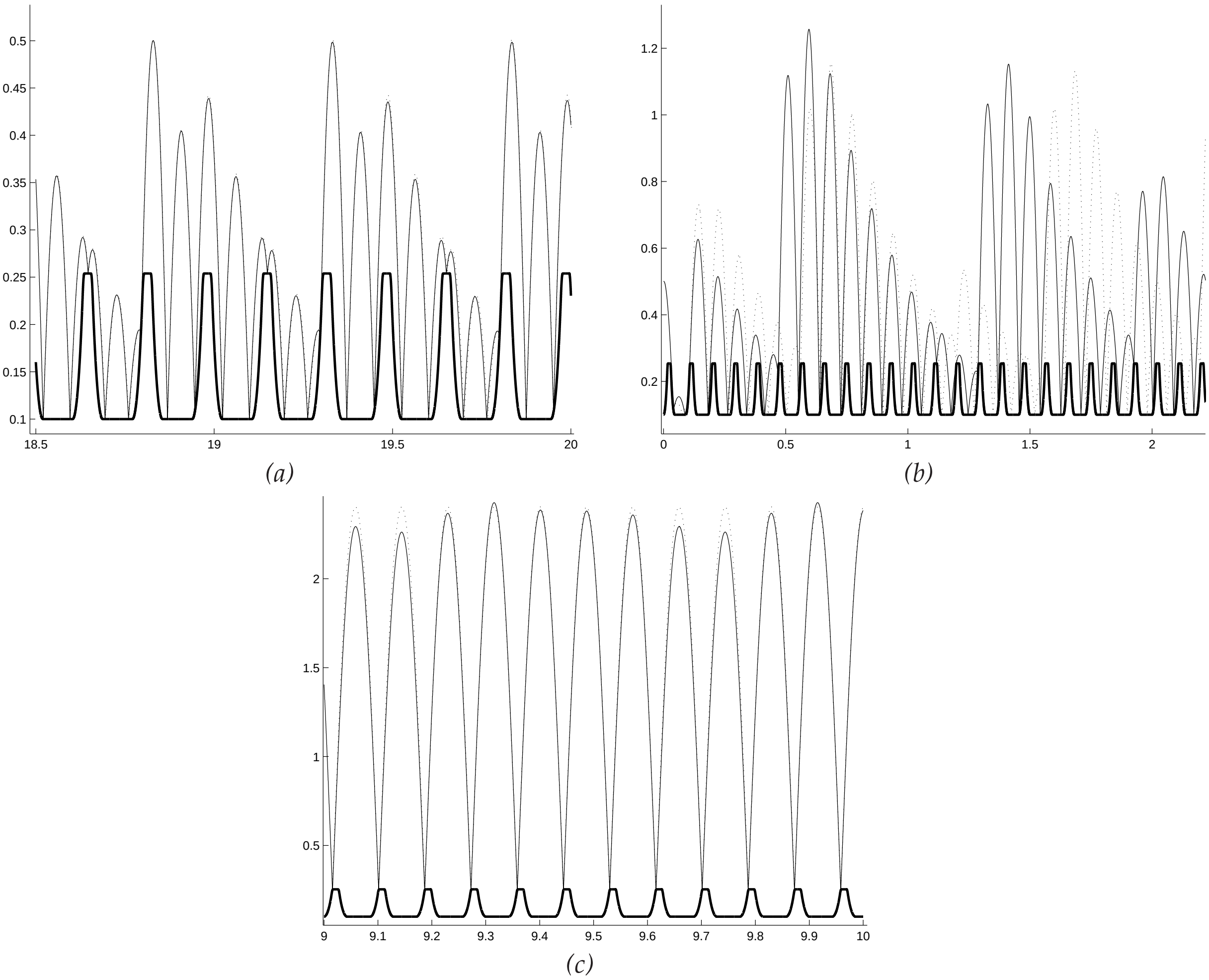



Figure 5.2: Time series using SICONOS platform. Time-stepping scheme (continuous line). Event-driven scheme (dashed line) *(a)* rpm=358. *(b)* rpm=660. *(c)* rpm=700.

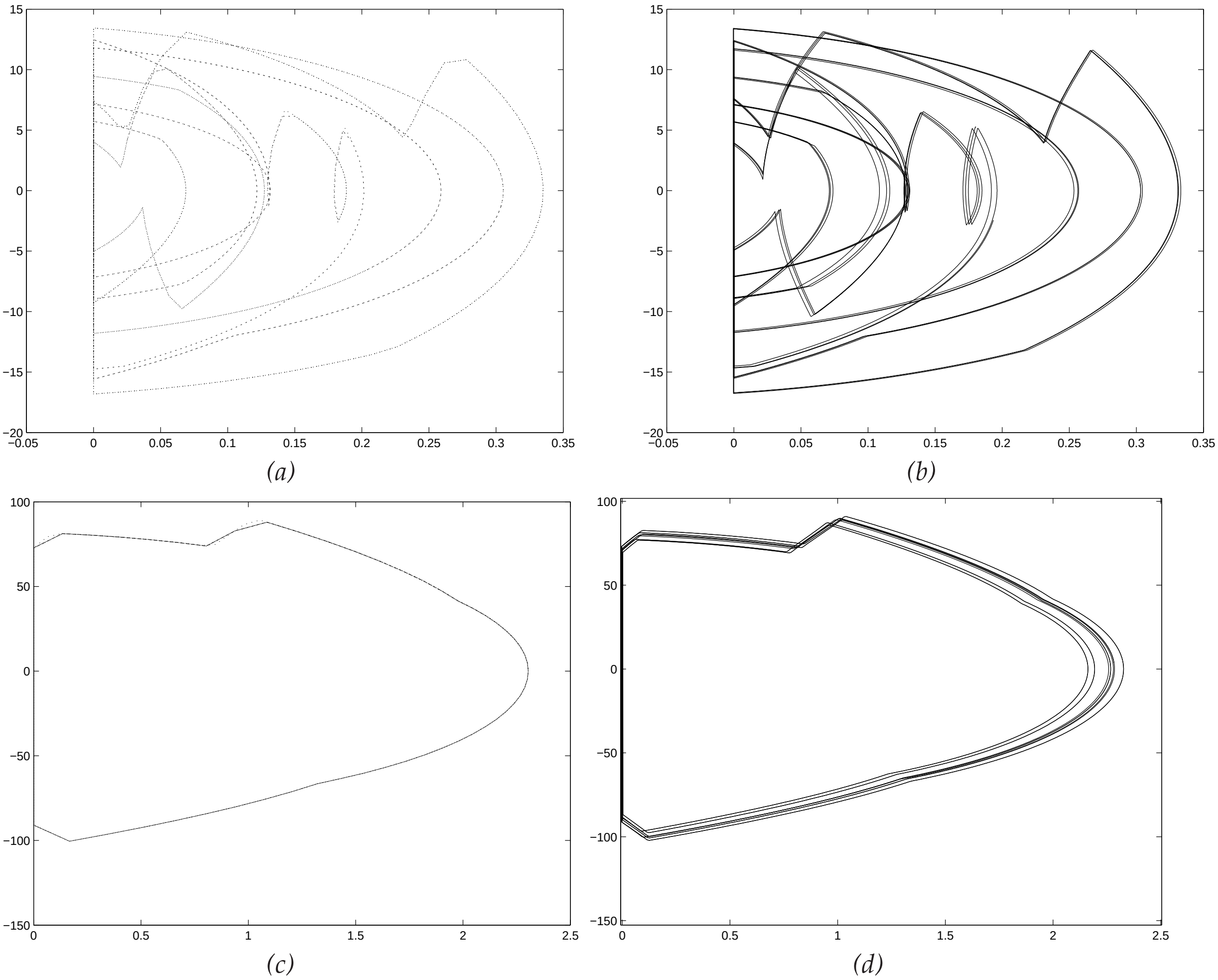



Figure 5.3: State space comparison using SICONOS platform. *(a)* rpm=358. Event Driven *(b)* rpm=358. Time Stepping ($h = 1e^{-4}$)*(c)* rpm=700. Event Driven *(d)* rpm=700. Time Stepping ($h = 1e^{-4}$)

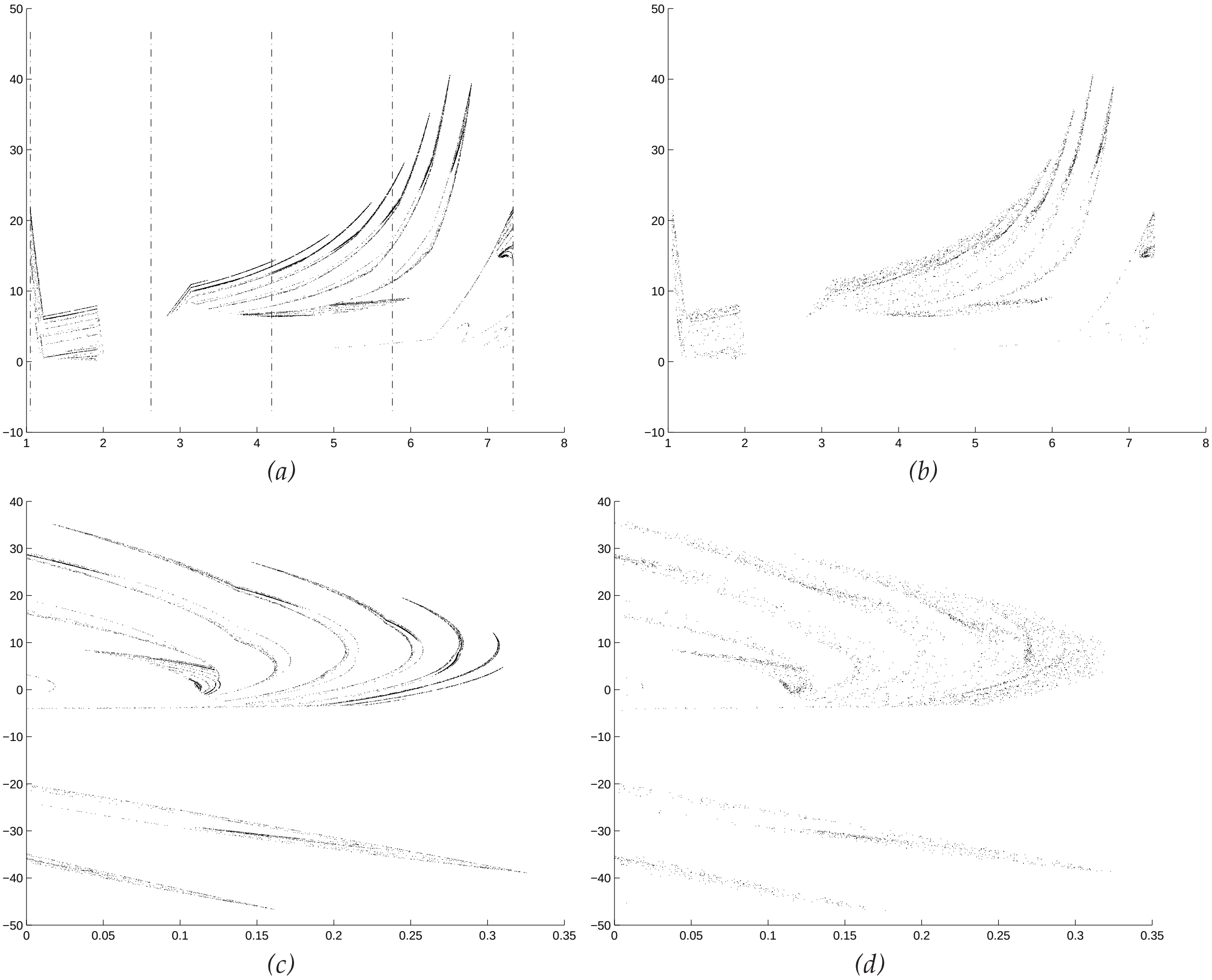


Figure 5.4: Attractors comparison using SICONOS platform at rpm=660. *(a)* Impact map. (Event Driven) *(b)* Impact Map. Time Stepping ($h = 1e^{-4}$)*(a)* Stroboscopic map. (Event Driven) *(b)* Stroboscopic Map. Time Stepping ($h = 1e^{-4}$)