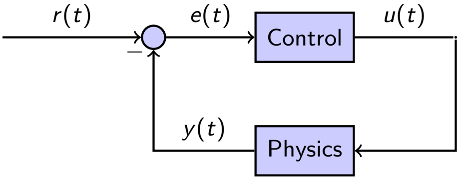
DynIBEX: une boîte à outils pour la vérification des systèmes cyber-physiques

Alexandre Chapoutot

joint work with Julien Alexandre dit Sandretto and Olivier Mullier
U2IS, ENSTA ParisTech, Palaiseau, France

Journées AFADL
7 juin 2016

A small cyber-physical system: closed-loop control



- **Control** may be a continuous-time PI algorithm

$$e(t) = r(t) - y(t) \ , \qquad u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau$$

- **Physics** is usually defined by non-linear differential equations (with parameters)

$$\dot{\mathbf{x}} = f(\mathbf{x}(t), u(t), \mathbf{p}) \ , \qquad y(t) = g(\mathbf{x}(t))$$

What is designing a controller?

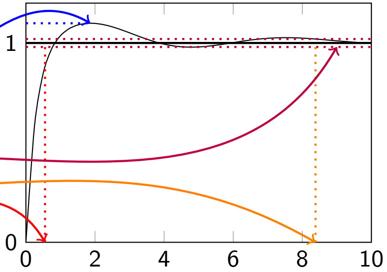
Find values for K_p and K_i such that a **given specification** is satisfied.

Specification of PID Controllers

PID controller: requirements based on closed-loop response

We observe the output of the plant

- Overshoot: Less than 10%
- Steady-state error: Less than 2%
- Settling time: Less than 10s
- Rise time: Less than 2s



Note: such properties come from the **asymptotic behavior** (well defined for linear case) of the system.

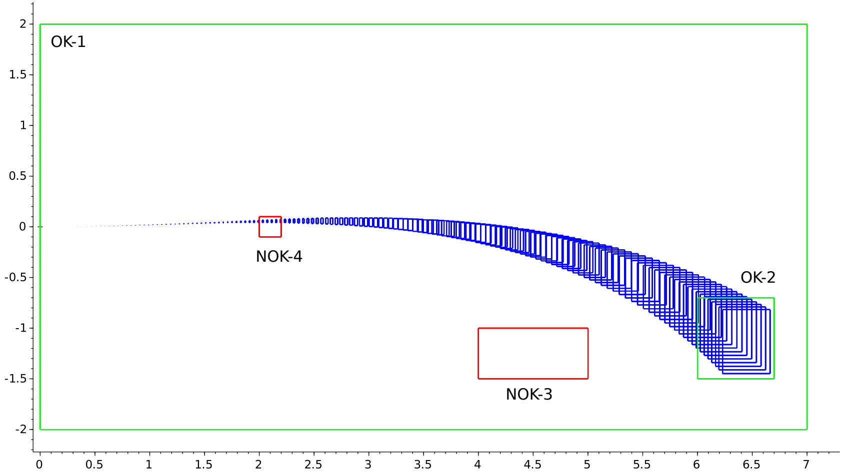
Classical method to study/verify closed-loop systems

Numerical simulations but

- do not take into account that models are only an approximation;
- produce approximate results.

and not adapted to deal with uncertainties

More properties on a simulation



Example of properties

- system stays in safe zone ($\forall t$) or finishes in goal zone ($\exists t$)
- system avoids obstacle ($\exists t$) or not in invalid space at a given time ($\exists t$)

for **different quantification's** of initial state-space ($\forall \mathbf{x}$ or $\exists \mathbf{x}$), parameters, etc.

Set-based simulation

Definition

numerical simulation methods implemented interval analysis methods

Goals

takes into account various uncertainties (bounded) or approximations to produce rigorous results

Example

A simple dynamics of a car

$$\dot{y} = \frac{-50.0y - 0.4y^2}{m} \qquad \text{with} \quad m \in [990, 1010]$$

One Implementation **DynIBEX**: a combination of **CSP solver** (IBEX[1]) with **validated numerical integration methods**

`http://perso.ensta-paristech.fr/~chapoutot/dynibex/`

[1] Gilles Chabert (EMN) et al. `http://www.ibex-lib.org`

IBEX in one slide

- Easy definition of functions
- Numerical constraints
- Pruning methods
- Interval evaluation of functions

```
#include "ibex.h"

using namespace std;
using namespace ibex;

int main() {

  Variable x;
  Function f (x, x*exp(x));

  NumConstraint c1(x, f(x) <= 3.0);

  CtcFwdBwd contractor(c1);

  IntervalVector box(1);
  box[0]=Interval(1,10);

  cout << "f" << box << " = " << f.eval(box) << endl;
  contractor.contract(box);
  cout << "after contraction box = " << box << endl;
}
```

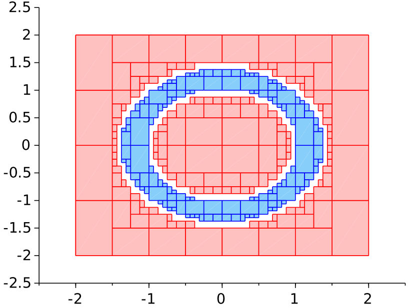
IBEX is also a parametric solver of constraints, an optimizer, etc.

Paving

Methods used to represent complex sets $\mathcal{S}$ with

- inner boxes i.e. set of boxes included in $\mathcal{S}$
- outer boxes i.e. set of boxes that does not belong to $\mathcal{S}$
- the frontier i.e. set of boxes we do not know

Example, a ring $\mathcal{S} = \{(x, y) \mid x^2 + y^2 \in [1, 2]\}$ over $[-2, 2] \times [-2, 2]$



Remark: involving bisection algorithm and so complexity is exponential in the size of the state space (`contractor programming` to overcome this).

Initial Value Problem of Ordinary Differential Equations

Consider an IVP for ODE, over the time interval $[0, T]$

$$\dot{\mathbf{y}} = f(\mathbf{y}) \quad \text{with} \quad \mathbf{y}(0) = \mathbf{y}_0$$

IVP has a unique solution $\mathbf{y}(t; \mathbf{y}_0)$ if $f : \mathbb{R}^n \to \mathbb{R}^n$ is Lipschitz in $\mathbf{y}$
but for our purpose we suppose f smooth enough i.e., of class C^k

Goal of numerical integration

- Compute a sequence of time instants: $t_0 = 0 < t_1 < \cdots < t_n = T$
- Compute a sequence of values: $\mathbf{y}_0, \mathbf{y}_1, \ldots, \mathbf{y}_n$ such that

$$\forall i \in [0, n], \quad \mathbf{y}_i \approx \mathbf{y}(t_i; \mathbf{y}_0) \ .$$

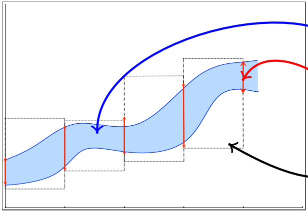
Validated solution of IVP for ODE

Goal of validated numerical integration

- Compute a sequence of time instants: $t_0 = 0 < t_1 < \cdots < t_n = T$
- Compute a sequence of values: $[\mathbf{y}_0], [\mathbf{y}_1], \ldots, [\mathbf{y}_n]$ such that

$$\forall i \in [0, n], \quad [\mathbf{y}_i] \ni \mathbf{y}(t_i; \mathbf{y}_0) \ .$$

A two-step approach



- Exact solution of $\dot{\mathbf{y}} = f(\mathbf{y}(t))$ with $\mathbf{y}(0) \in \mathcal{Y}_0$
- Safe approximation at discrete time instants
- Safe approximation between time instants

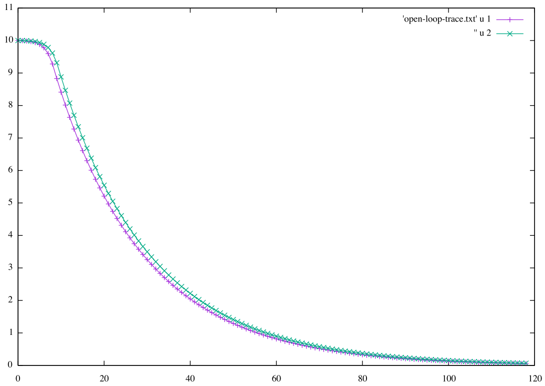
Note A contractor approach can be used on the guaranteed solution of ODE [SWIM16]

Simulation of an open loop system

A simple dynamics of a car

$$\dot{y} = \frac{-50.0y - 0.4y^2}{m} \qquad \text{with} \quad m \in [990, 1010]$$

Simulation for 100 seconds with $y(0) = 10$



The last step is $y(100) = [0.0591842, 0.0656237]$

Simulation of an open loop system

- ODE definition
- IVP definition
- Parametric simulation engine

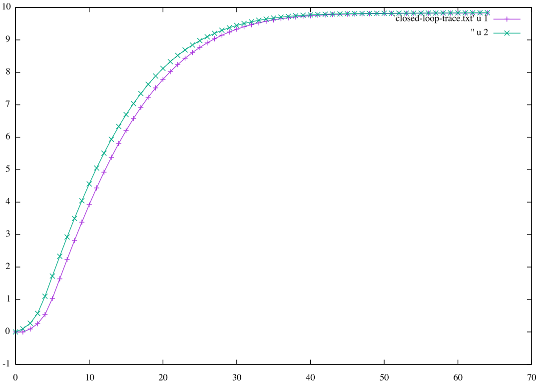
```
int main(){

  const int n = 1;
  Variable y(n);

  IntervalVector state(n);
  state[0] = 10.0;

  // Dynamique d'une voiture avec incertitude sur sa
masse
  Function ydot(y, ( -50.0 * y[0] - 0.4 * y[0] * y[0])
                 / Interval (990, 1010));
  ivp_ode vdp = ivp_ode(ydot, 0.0, state);

  // Integration numerique ensembliste
  simulation simu = simulation(&vdp, 100, RK4, 1e-5);
  simu.run_simulation();

  //For an export in order to plot
  simu.export1d_yn("export-open-loop.txt", 0);

  return 0;
}
```

Simulation of a closed-loop system

A simple dynamics of a car with a PI controller

$$\begin{pmatrix} \dot{y} \\ \dot{w} \end{pmatrix} = \begin{pmatrix} \frac{k_p(10.0-y)+k_i w - 50.0y - 0.4y^2}{m} \\ 10.0 - y \end{pmatrix} \quad \text{with} \quad m \in [990, 1010], k_p = 1440, k_i = 35$$

Simulation for 10 seconds with $y(0) = w(0) = 0$



The last step is $y(10) = [9.83413, 9.83715]$

Simulation of a closed-loop system

```
#include "ibex.h"

using namespace ibex;

int main(){

  const int n = 2;
  Variable y(n);

  IntervalVector state(n);
  state[0] = 0.0;
  state[1] = 0.0;

  // Dynamique d'une voiture avec incertitude sur sa masse + PI
  Function ydot(y, Return ((1440.0 * (10.0 - y[0]) + 35.0 * y[1]  - y[0] * (50.0  + 0.4 * y[0]))
                          / Interval (990, 1010),
                          10.0 - y[0]));
  ivp_ode vdp = ivp_ode(ydot, 0.0, state);

  // Integration numerique ensembliste
  simulation simu = simulation(&vdp, 10.0, RK4, 1e-7);
  simu.run_simulation();

  simu.export1d_yn("export-closed-loop.txt", 0);

  return 0;
}
```

Simulation of a closed-loop system with safety

A simple dynamics of a car with a PI controller

$$\begin{pmatrix} \dot{y} \\ \dot{w} \end{pmatrix} = \begin{pmatrix} \frac{k_p(10.0-y)+k_i w-50.0y-0.4y^2}{m} \\ 10.0-y \end{pmatrix} \quad \text{with} \quad m \in [990, 1010], k_p = 1440, k_i = 35$$

and a safety propriety

$$\forall t, y(t) \in [0, 11]$$

Failure

$$y([0, 0.0066443]) \in [-0.00143723, 0.0966555]$$

Simulation of a closed-loop system with safety property

```
#include "ibex.h"

using namespace ibex;

int main(){
  const int n = 2;
  Variable y(n);

  IntervalVector state(n);
  state[0] = 0.0; state[1] = 0.0;

  // Dynamique d'une voiture avec incertitude sur sa masse + PI
  Function ydot(y, Return ((1440.0 * (10.0 - y[0]) + 35.0 * y[1]  - y[0] * (50.0  + 0.4 * y[0]))
                         / Interval (990, 1010),
                         10.0 - y[0]));
  ivp_ode vdp = ivp_ode(ydot, 0.0, state);

  simulation simu = simulation(&vdp, 10.0, RK4, 1e-6);
  simu.run_simulation();

  // verification de surete
  IntervalVector safe(n);
  safe[0] = Interval(0.0, 11.0);
  bool flag = simu.stayed_in (safe);
  if (!flag) {
    std::cerr << "error safety violation" << std::endl;
  }

  return 0;
}
```

Properties in DynIBEX

- `safe`(X) i.e.

$$\forall t \leqslant t_{\text{end}}, y(t) \in X$$

- `finished_in`(X) i.e.

$$y(t_{\text{end}}) \in X$$

- `go_out`(X) i.e.,

$$\exists t \leqslant t_{\text{end}}, y(t) \cap X = \emptyset$$

- `has_crossed`(X) i.e.,

$$\exists t \leqslant t_{\text{end}}, y(t) \cap X \neq \emptyset$$

- `has_reached`(X) i.e.,

$$y(t_{\text{end}}) \cap X \neq \emptyset$$

- `one_in`(X_1, X_2, ..., X_n) i.e.,

$$\exists t \leqslant t_{\text{end}}, \exists i, y(t) \in X_i$$

- `stayed_in`(X) i.e.,

$$\forall t \leqslant t_{\text{end}}, y(t) \in X$$

- `stayed_in_till`(X, T) i.e.,

$$\forall t \leqslant T \leqslant t_{\text{end}}, y(t) \in X$$

Simulation of an hybrid closed-loop system

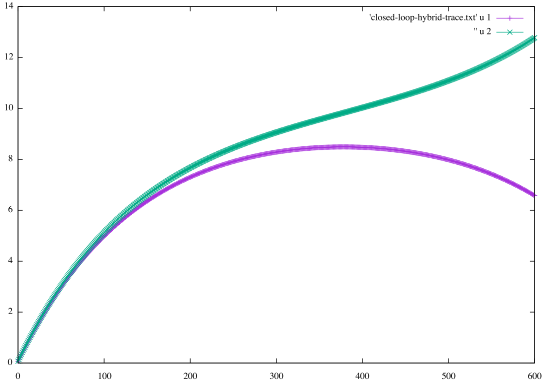
A simple dynamics of a car with a discrete PI controller

$$\dot{y} = \frac{u(k) - 50.0y - 0.4y^2}{m} \quad \text{with} \quad m \in [990, 1010]$$

$$i(t_k) = i(t_{k-1}) + h(c - y(t_k)) \quad \text{with} \quad h = 0.005$$

$$u(t_k) = k_p(c - y(t_k)) + k_i i(t_k) \quad \text{with} \quad k_p = 1400, k_i = 35$$

Simulation for 3 seconds with $y(0) = 0$ and $c = 10$



Simulation of an hybrid closed-loop system

```
int main(){
  const int n = 1;
  Variable y(n);

  IntervalVector state(n); state[0] = 0.0;

  double t = 0;
  const double sampling = 0.005;
  Affine2 integral(0.0);

  while (t < 3.0) {
    Affine2 goal(10.0);
    Affine2 error = goal - state[0];

    // Controleur PI discret
    integral = integral + sampling * error;
    Affine2 u = 1400.0 * error + 35.0 * integral;

    // Dynamique d'une voiture avec incertitude sur sa masse
    Function ydot(y, (u.itv() - 50.0 * y[0] - 0.4 * y[0] * y[0])
                    / Interval (990, 1010));
    ivp_ode vdp = ivp_ode(ydot, 0.0, state);

    // Integration numerique ensembliste
    simulation simu = simulation(&vdp, sampling, RK4, 1e-6);
    simu.run_simulation();

    // Mise a jour du temps et des etats
    state = simu.get_last();
    t += sampling;
  }

  return 0;
}
```

- Two representations of sets: box, zonotopes **Interaction must be improved**
- Manual handling of discrete-time evolution

Conclusion

DynIBEX is one **ingredient** of verification tools for cyber-physical systems.
It can **handle uncertainties**, can **reason on sets of trajectories**.

Already applied on

- Parameter synthesis of PI controllers [SYNCOP'15]
- Controller synthesis of sampled switched systems [SNR'16]
- Parameter tuning in the design of mobile robots [MORSE'16]

Future work (a lot)

- Pursue and improve cooperation with IBEX language
- Extend the set of properties
- SMT modulo ODE
- Improve algorithm of validated numerical integration
- Simulation of hybrid systems