Taylor-based reachability analysis of continuous and hybrid systems

Émilie THOMÉ

Software verification and introduction to hybrid systems analysis
Master IP Paris – Cyber-Physical Systems

December, 1^{st} 2020

Outlines

Introduction

Project goal

The goal of this project is to implement a Taylor-based reachability analysis for continuous systems and to extend it to hybrid systems.

Taylor-Lagrange expansion

$$x(t+h) = x(t) + \sum_{i=1}^{n-1} \frac{h^i}{!i} \frac{d^i x}{dt^i}(t) + O(h^n)$$

Tools

In order to implement the analysis in C++ with interval as suggested, I used some existing tools:

FADBAD++ Forward, backward and Taylor methods implementation

FILIB++ `filib++` Interval library

Matplotlib for C++ C++ wrapper for Python's matplotlib (MPL) plotting library

Tools

In order to implement the analysis in C++ with interval as suggested, I used some existing tools:

FADBAD++ Forward, backward and Taylor methods implementation

FILIB++ `filib++` Interval library

Matplotlib for C++ C++ wrapper for Python's matplotlib (MPL) plotting library

Tools

In order to implement the analysis in C++ with interval as suggested, I used some existing tools:

FADBAD++ Forward, backward and Taylor methods implementation

FILIB++ `filib++` Interval library

Matplotlib for C++ C++ wrapper for Python's matplotlib (MPL) plotting library

Program description

Project global method

The method used to compute an enclosure of $x(t_i + h)$ from an enclosure X_i of $x(t_i)$ was to compute a polynomial enclosure:

Polynomial enclosure of $x(t_i + t)$ for $t \in [0, h]$:

$$P_i(t) = X_i + \sum_{k=1}^{n-1} \frac{t^k}{!k} L_f^k(x)(X_i) + \frac{t^n}{!n} L_f^n(x)(B)$$

With, for $\dot{x} = f(x)$:

$$\begin{cases} L_f^1(x)(X_i) = L_f(x)(X_i) = \{f(x_i), x_i \in X_i\} \\ L_f^k(x)(X_i) = L_f(x)(L^{i-1}(x)(X_i)) \end{cases}$$

Examples

I tested the project over different examples:

- **ODE:** Linear
- **ODE:** Exponential
- **ODE:** Cosinus
- **ODE:** Brusselator
- **Hybrid system:** Bouncing Ball

Examples - Linear

```
/* ODE definitions */
vector<T<interval>> Linear(vector<T<interval>> x){
        vector<T<interval>> f = vector<T<interval>>(DIM);
        f[0] = 1;
        return f;
}
```

Examples - Exponential

```
/* ODE definitions */
vector<T<interval>> Exponential(vector<T<interval>> x){
        vector<T<interval>> f = vector<T<interval>>(DIM);
        f[0] = -x[0];
        return f;
}
```

Examples - Cosinus

```
/* ODE definitions */
vector<T<interval>> Cos(vector<T<interval>> x){
        vector<T<interval>> f = vector<T<interval>>(DIM);
        f[0] = x[1];
        f[1] = -interval(39.478417, 39.478418)*x[0];
        return f;
}
```

Examples - Brusselator

```
/* ODE definitions */
vector<T<interval>> Brusselator(vector<T<interval>> x){
        vector<T<interval>> f = vector<T<interval>>(DIM);
        f[0] = interval(1.) + x[0]*x[0]*x[1] - 2.5*x[0];
        f[1] = 1.5*x[0] - x[0]*x[0]*x[1];
        return f;
}
```

Examples - Bouncing Ball

```
/* ODE definitions */
vector<T<interval>> BouncingBall(vector<T<interval>> x){
        vector<T<interval>> f = vector<T<interval>>(DIM);
        f[0] = x[1];
        f[1] = interval(-g);
        return f;
}

/* Modifications definitions */
v_interval Bump(v_interval x){
        v_interval new_x = v_interval(DIM);
        new_x[0] = x[0];
        new_x[1] = -interval(amortissement)*x[1];
        return new_x;
}
```

Results

Results - Linear

There is no limit in the reachability of a linear system but the result printed on the terminal is approximated by $[1e+03, 1e+03]$ for a $x_0 \in [0.9999, 1.0001]$.

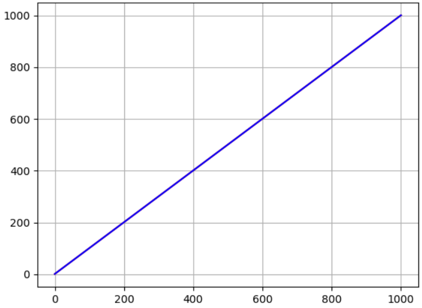

Figure: No Limit for Linear

Results - Linear

But we can see when zooming on the graph that it is just what is printed that is approximated.

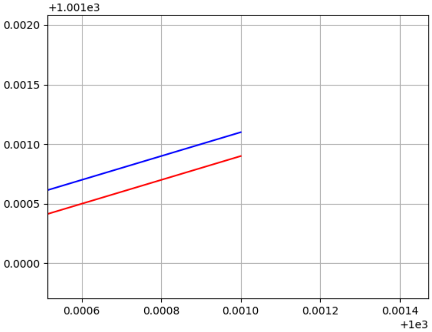

Figure: No Limit for Linear

Results - Exponential

For $t_{end} = 10$ we have the enclosure:

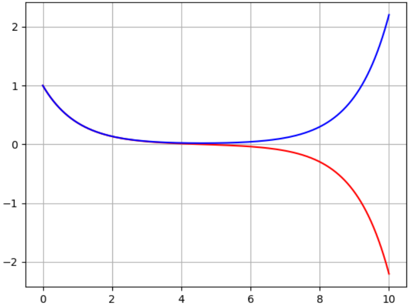

Figure: Exponential Limit

For $t_{end} = 5$ we have the enclosure:

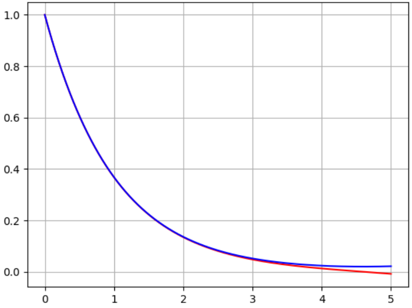

Figure: Exponential

Results - Cosinus

For $t_{end} = 2.5$ the program reach a limit:

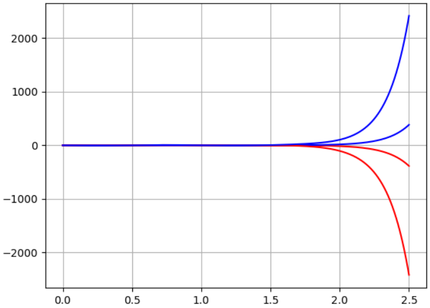

Figure: Cosinus Limit

For $t_{end} = 1$ we have the enclosure:

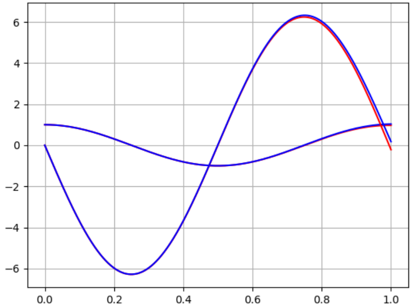

Figure: Cosinus

Results - Brusselator

For $t_{end} = 2.6$ the program reach a limit:

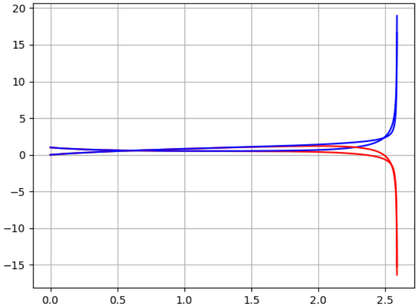

Figure: Brusselator Limit

For $t_{end} = 1$ we have the enclosure:

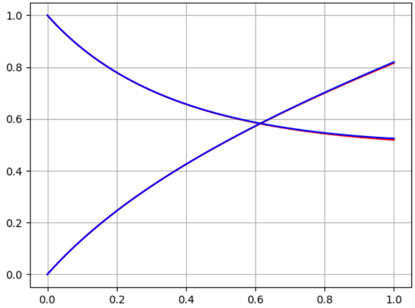

Figure: Brusselator

Results - Bouncing Ball

For $t_{end} = 2.2$, surprising enclosure:

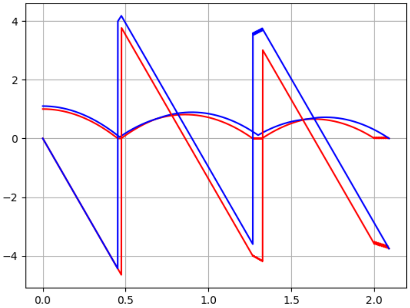

Figure: Bouncing Ball Limit

For $t_{end} = 1$ we have the enclosure:

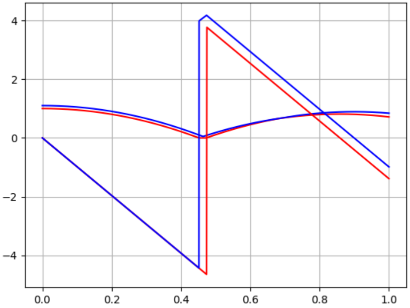

Figure: Bouncing Ball

Comparison with VNODE

Comparison - Linear

VNODE:

```
// set initial condition and endpoint:
const int n= 1;
interval t= 0.0, tend= 1000;
iVector x(n);
x[0] = interval(0.9999, 1.0001);
```

```
  $ Solution enclosure at t =
      9.9999999999999999[9,10]E+2
  $ 1.000999[8999999997977,
              1100000002023]E+3
```

Project:

```
  $ x0(1000) = [1e+03, 1e+03]
```

Comparison - Exponential

VNODE:

```
// set initial condition and endpoint:
const int n= 1;
interval t= 0.0, tend= 5.;
iVector x(n);
x[0] = interval(0.9999, 1.0001);
```

```
$ Solution enclosure at t = [5,5]
$ 0.00673[72732043855,86207937854]
```

Project:

```
$ x0(5.0005) = [-0.00811, 0.0216]
```

Comparison - Cosinus

VNODE:

```
// set initial condition and endpoint:
const int n= 2;
interval t= 0.0, tend= 1.;
iVector x(n);
x[1] = interval(-0.0001, 0.0001);
x[0] = interval(0.9999, 1.0001);
```

```
  $ Solution enclosure at t = [1,1]
  $ 0.999[8999999999952,1100000000049]
  $ [-0.000100000000287,
      0.000100000000314]
```

Project:

```
  $ x0(1.0005) = [0.969, 1.03]
  $ x1(1.0005) = [-0.215, 0.176]
```

Comparison - Brusselator

VNODE:

```
// set initial condition and endpoint:
const int n= 2;
interval t= 0.0, tend= 1.0005;
iVector x(n);
x[0] = interval(0.9999, 1.0001);
x[1] = interval(-0.0001, 0.0001);
```

```
$ Solution enclosure at t =
    1.00049999999999[9,10]
$ 0.521[5917083997330,6585373290130]
$ 0.817[5579798431930,7759852494221]
```

Project:

```
$ x0(1.0005) = [0.519, 0.524]
$ x1(1.0005) = [0.816, 0.82]
```

Reliability for Bouncing Ball

Theory of the Bouncing Ball

$$\begin{cases} x = -\sqrt{2g}(t - \sqrt{\frac{2}{g}}) - \frac{g}{2}(t - \sqrt{\frac{2}{g}})^2 \\ v = 0.9 * \sqrt{2g} - g(t - \sqrt{\frac{2}{g}}) \end{cases}$$

So for $t = 1.0005$ and $g = 9.80665$:

$$\begin{cases} x = 0.71048975284 \notin [0.713, 0.841] \\ v = -1.39704127735 \notin [-1.39, -0.989] \end{cases}$$

Project:

```
$ x0(1.0005) = [0.713, 0.841]
$ x1(1.0005) = [-1.39, -0.989]
```

Conclusion

Conclusion

Far from being a success, this project was still a challenging exercise that I am proud to show you today.
I pinpointed some mistakes that I have done and improvements.

Improvements:

1. Forgetting some interval functions (pi(), pow(...), etc)
2. Interval computation seems to be unsafe when printed
3. Use of interval in VNODE time definition

Thank you for your attention