A Generic Ellipsoid Abstract Domain
for Linear Time Invariant Systems

Pierre Roux ¹  Romain Jobredeaux ²
Pierre-Loïc Garoche ¹  Éric Féron ²

April 18, 2012

¹ONERA
Toulouse, France

²Georgia Tech
Atlanta, Georgia, USA
Critical systems rely on a control command computation core:

- flight commands of an aircraft;
- attitude control of a satellite;
- control of a car engine;
- ...
Stability of Linear Systems

Those are reactive systems
Stability of Linear Systems

Those are reactive systems periodically computing output as a combination of inputs and internal state variables.
Stability of Linear Systems

Those are reactive systems periodically computing output as a combination of inputs and internal state variables.

Goal

Assuming a bound on inputs, we need to bound the outputs.
Stability of Linear Systems

Those are reactive systems periodically computing output as a combination of inputs and internal state variables.

Goal

Assuming a bound on inputs, we need to bound the outputs. (for control theorists: open loop stability)
Stability of Linear Systems

Those are reactive systems periodically computing output as a combination of inputs and internal state variables.

Goal

Assuming a bound on inputs, we need to bound the outputs. (for control theorists: open loop stability)

We will focus on the (very common) case of linear systems.
Stability of linear systems, example

// DiscreteStateSpace_A: A
real_T A[16] =
    { 0.6227, 0.3871, -0.113, 0.0102,
    -0.3407, 0.9103, -0.3388, 0.0649,
    0.0918, -0.0265, -0.7319, 0.2669,
    0.2643, -0.1298, -0.9903, 0.3331 }; 

// DiscreteStateSpace_A: B
real_T B[8] =
    { 0.3064, 0.1826,
    -0.0054, 0.6731,
    -0.0494, 1.6138,
    -0.0531, 0.4012 }; 

static void MIMO_update(int_T tid) {
    static real_T xnew[4];
    xnew[0] += (B[0])*INPUT[0] + (B[1])*INPUT[1];
    xnew[1] += (B[2])*INPUT[0] + (B[3])*INPUT[1];
    xnew[2] += (B[4])*INPUT[0] + (B[5])*INPUT[1];
    xnew[3] += (B[6])*INPUT[0] + (B[7])*INPUT[1];
    (void) memcpy(St, xnew, sizeof(real_T)*4);
}

Result:
if INPUT[0..1] remains in [−1, 1] then St[0..3] remains in [−5, 5]
Stability of linear systems, example

// DiscreteStateSpace_A: A
real_T A[16] =
{ 0.6227, 0.3871, -0.113, 0.0102,
-0.3407, 0.9103, -0.3388, 0.0649,
0.0918, -0.0265, -0.7319, 0.2669,
0.2643, -0.1298, -0.9903, 0.3331
};

// DiscreteStateSpace_A: B
real_T B[8] =
{ 0.3064, 0.1826,
-0.0054, 0.6731,
-0.0494, 1.6138,
-0.0531, 0.4012
};

static void MIMO_update(int_T tid) {
    static real_T xnew[4];
    xnew[0] += (B[0])*INPUT[0] + (B[1])*INPUT[1];
    xnew[1] += (B[2])*INPUT[0] + (B[3])*INPUT[1];
    xnew[2] += (B[4])*INPUT[0] + (B[5])*INPUT[1];
    xnew[3] += (B[6])*INPUT[0] + (B[7])*INPUT[1];
    (void) memcpy(St, xnew, sizeof(real_T)*4);
}

Result:

- if INPUT[0..1] remains in [−1, 1]
- then St[0..3] remains in [−5, 5]
Quadratic invariants

Linear invariants commonly used in static analysis are not well suited:

- at best costly;
- at worst no result.

Control theorists know for long that quadratic invariants are a good fit for linear systems.
**Quadratic invariants**

- **Linear invariants** commonly used in static analysis are not well suited:
  - at best costly;
  - at worst no result.

A Generic Ellipsoid Abstract Domain for Linear Time Invariant Systems
Quadratic invariants

**Linear invariants** commonly used in static analysis are not well suited:

- at best costly;
- at worst no result.
Quadratic invariants

- **Linear invariants** commonly used in static analysis are not well suited:
  - at best costly;
  - at worst no result.

- Control theorists know for long that **quadratic invariants** are a good fit for linear systems.
Quadratic invariants

- **Linear invariants** commonly used in static analysis are not well suited:
  - at best costly;
  - at worst no result.

- Control theorists know for long that **quadratic invariants** are a good fit for linear systems.
1. Overall Method

2. Shape of the Ellipsoid

3. Floating Point Issues

4. Experimental Results

5. Related Work and Perspectives
1 Overall Method

2 Shape of the Ellipsoid

3 Floating Point Issues

4 Experimental Results

5 Related Work and Perspectives
Overall Method

Lyapunov Stability

Theorem

For $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{n \times p}$, the sequence

$$\begin{cases} 
    x_0 \in \mathbb{R}^n \\
    x_{k+1} = Ax_k + Bu_k
\end{cases}$$

is bounded for all $u \in (\mathbb{R}^p)^\mathbb{N}$ such that for all $k \in \mathbb{N}$, $\|u_k\|_\infty \leq 1$ if and only if there exist $P \in \mathbb{R}^{n \times n}$ positive definite such that

$$A^T P A - P \prec 0$$

where "$P \prec 0$" means that for all $x \in \mathbb{R}^n$: $x \neq 0 \Rightarrow x^T P x < 0$. 
Overall Method

Lyapunov Stability, Invariant

Invariant ellipsoid

Moreover, there exist a $\lambda > 0$ such that $x$ remains in the ellipsoid $\{ x \in \mathbb{R}^n \mid x^T P x \leq \lambda \}$.

In Computer Science Language

The property $"x^T P x \leq \lambda"$ is a loop invariant.
Overall Method

Lyapunov Stability, Illustration

\{ x \mid x^T P x \leq 1 \}

\{ A x \mid x^T P x \leq 1 \}

\{ A x_k + B u \mid \| u \|_{\infty} \leq 1 \}

A Generic Ellipsoid Abstract Domain for Linear Time Invariant Systems
Overall Method

Method

1. First determine the shape of the ellipsoid by choosing a matrix $P$;

2. then find the smallest possible ratio $\lambda$ such that $x^TPx \leq \lambda$ is an invariant.
Definition (Semidefinite Programming)

Minimize a linear objective function of variables $y_i$ under constraint

$$A_0 + \sum_{i=1}^{k} y_i A_i \succeq 0$$

where the $A_i$ are known matrices and “$P \succeq 0$” means $x^T P x \geq 0$ for all vector $x$. 

Remark

Efficient tools do exist.
Overall Method

Tools

**Definition (Semidefinite Programming)**

Minimize a linear objective function of variables $y_i$ under constraint

$$A_0 + \sum_{i=1}^{k} y_i A_i \succeq 0$$

where the $A_i$ are known matrices and “$P \succeq 0$” means $x^T P x \geq 0$ for all vector $x$.

**Remark**

Efficient tools do exist.
1 Overall Method

2 Shape of the Ellipsoid

3 Floating Point Issues

4 Experimental Results

5 Related Work and Perspectives
What are we looking for?

- We look for a positive definite matrix $P$ such that $A^T P A - P ≺ 0$. 

...but we will then look for a $\lambda > 0$ such that $x^T P x ≤ \lambda$ is an invariant and the ellipsoid $\{x | x^T P x ≤ \lambda\}$ should be small enough.
Shape of the Ellipsoid

What are we looking for?

- We look for a positive definite matrix $P$ such that $A^T P A - P \prec 0$.
- There are lot of very different solutions...
What are we looking for?

- We look for a positive definite matrix $P$ such that $A^T P A - P \prec 0$.
- There are **lot of** very different **solutions**...
- ...but we will then look for a $\lambda > 0$ such that $x^T P x \leq \lambda$ is an invariant and the ellipsoid $\{x \mid x^T P x \leq \lambda\}$ should be small enough.

**Example**

```
\begin{align*}
\text{is better than} & \quad \text{is better than}
\end{align*}
```
What are we looking for?

- We look for a positive definite matrix $P$ such that $A^T P A - P \prec 0$.
- There are **lot of** very different **solutions**...
- ...but we will then look for a $\lambda > 0$ such that $x^T P x \leq \lambda$ is an invariant and the ellipsoid $\{ x \mid x^T P x \leq \lambda \}$ should be small enough.

**Example**

![Ellipsoids](image)

→ Tested three heuristics.
1 Overall Method

2 Shape of the Ellipsoid

3 Floating Point Issues

4 Experimental Results

5 Related Work and Perspectives
Floating Point Issues

Rounding Errors

Actual computations are carried out with floating point numbers leading to rounding errors.

Example

```c
int i = 0;
float x = 0;
while (i < 1000000) {
    x += 0.1;
    ++i;
}
printf("%.0f\n", x);
```
Floating Point Issues

Rounding Errors

Actual computations are carried out with floating point numbers leading to rounding errors.

Example

```c
int i = 0;
float x = 0;
while (i < 1000000) {
    x += 0.1;
    ++i;
}
printf("%.0f\n", x);
```

Gives 100958!
Floating Point Issues

**Rounding Errors**

Actual computations are carried out with floating point numbers leading to *rounding errors*.

**Example**

```c
int i = 0;
float x = 0;
while (i < 1000000) {
    x += 0.1;
    ++i;
}
printf("%.0f\n", x);
```

Gives 100958!

We have to distinguish two problems:

- rounding errors *in the analyzed program*;
- and rounding errors *in the analyzer* itself;
Floating Point Issues

Rounding Errors

Actual computations are carried out with floating point numbers leading to **rounding errors**.

**Example**

```c
int i = 0;
float x = 0;
while (i < 1000000) {
  x += 0.1;
  ++i;
}
printf("%.0f\n", x);
```

Gives 100958!

We have to distinguish two problems:

- rounding errors **in the analyzed program**;
- and rounding errors **in the analyzer** itself;

→ both are addressed efficiently.
1 Overall Method

2 Shape of the Ellipsoid

3 Floating Point Issues

4 Experimental Results

5 Related Work and Perspectives
Experimental Results

Our Prototype

- OCaml front-end to Scilab (lmisolver function).
- Only 2.5 kloc.
### Experimental Results

<table>
<thead>
<tr>
<th>Shape</th>
<th>Bounds</th>
<th>Valid.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ex. 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>n=2, 1 input</td>
<td>0.07</td>
<td>[140.4; 189.9]</td>
</tr>
<tr>
<td></td>
<td>0.16</td>
<td>[22.2; 26.5]</td>
</tr>
<tr>
<td></td>
<td>0.23</td>
<td>[16.2; 17.6]</td>
</tr>
<tr>
<td>Ex. 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>n=4, 1 input</td>
<td>0.09</td>
<td>[18.1; 25.2; 24.3; 33.7]</td>
</tr>
<tr>
<td></td>
<td>0.27</td>
<td>[6.3; 7.7; 2.2; 3.4]</td>
</tr>
<tr>
<td></td>
<td>0.40</td>
<td>[1.7; 2.0; 2.2; 2.5]</td>
</tr>
<tr>
<td>Ex. 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lead-lag controller</td>
<td>0.07</td>
<td>⊥</td>
</tr>
<tr>
<td>n=2, 1 input</td>
<td>0.17</td>
<td>[36.2; 36.1]</td>
</tr>
<tr>
<td></td>
<td>0.20</td>
<td>[38.8; 20.3]</td>
</tr>
<tr>
<td>Ex. 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LQG regulator</td>
<td>0.09</td>
<td>[1.2; 0.9; 0.5]</td>
</tr>
<tr>
<td>n=3, 1 input</td>
<td>0.19</td>
<td>[0.9; 0.9; 0.9]</td>
</tr>
<tr>
<td></td>
<td>0.24</td>
<td>[0.7; 0.4; 0.3]</td>
</tr>
</tbody>
</table>

Analysis times (in s) and bounds compared for the three heuristics.
## Experimental Results

### Experimental Results, continued

<table>
<thead>
<tr>
<th>Example</th>
<th>Shape</th>
<th>Bounds</th>
<th>Valid.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ex. 5 coupled mass system</td>
<td>0.09</td>
<td>[9.8; 8.9; 11.0; 16.8]</td>
<td>0.43</td>
</tr>
<tr>
<td>n=4, 2 inputs</td>
<td>0.24</td>
<td>[5.7; 5.6; 6.4; 10.1]</td>
<td>0.33</td>
</tr>
<tr>
<td></td>
<td>0.48</td>
<td>[5.0; 4.9; 4.8; 4.7]</td>
<td>0.22</td>
</tr>
<tr>
<td>Ex. 6 Butterworth low-pass filter</td>
<td>0.10</td>
<td>[7.5; 8.7; 6.1; 7.0; 6.5]</td>
<td>0.38</td>
</tr>
<tr>
<td>n=5, 1 input</td>
<td>0.32</td>
<td>[3.6; 5.0; 4.7; 8.1; 8.9]</td>
<td>0.29</td>
</tr>
<tr>
<td></td>
<td>0.78</td>
<td>[2.3; 1.1; 1.9; 2.0; 2.9]</td>
<td>0.24</td>
</tr>
<tr>
<td>Ex. 7 Dampened oscillator</td>
<td>0.07</td>
<td>[1.7; 2.1]</td>
<td>0.23</td>
</tr>
<tr>
<td>n=2, no input</td>
<td>0.15</td>
<td>[2.0; 2.0]</td>
<td>0.20</td>
</tr>
<tr>
<td></td>
<td>0.27</td>
<td>[1.5; 1.5]</td>
<td>0.16</td>
</tr>
<tr>
<td>Ex. 8 Harmonic oscillator</td>
<td>0.08</td>
<td>[1.5; 1.5]</td>
<td>0.23</td>
</tr>
<tr>
<td>n=2, no input</td>
<td>0.24</td>
<td>[1.5; 1.5]</td>
<td>0.20</td>
</tr>
<tr>
<td></td>
<td>0.15</td>
<td>[1.5; 1.5]</td>
<td>0.16</td>
</tr>
</tbody>
</table>

Analysis times (in s) and bounds compared for the three heuristics.
Experimental Results

Experimental Results, continued

(a) Ex. 1  (b) Ex. 2  (c) Ex. 3  (d) Ex. 4

Figure: Comparison of obtained ellipsoids by the three methods from lighter to darker, plus a random simulation trace ((b) and (d), being of dimension greater than 2, are cuts along planes containing the origin and two vectors of the canonical base, to show how the three different templates compare together).
Figure: Comparison of obtained ellipsoids by the three methods from lighter to darker, plus a random simulation trace (a) and (b), being of dimension greater than 2, are cuts along planes containing the origin and two vectors of the canonical base, to show how the three different templates compare together.)
1 Overall Method

2 Shape of the Ellipsoid

3 Floating Point Issues

4 Experimental Results

5 Related Work and Perspectives
Related Work and Perspectives

Related Work

- [Feret04]
  - Addresses a more restricted class of systems.
  + Extremely precise thanks to formal expansion (unrolling).
  + Handles actual C code.
  + Proved industrially useful.

- [AdjéGaubertGoubault10] and [GawlitzaSeidl10] (policy iteration)
  - Need to be given template ellipsoids.
  - Does not address floating point issues.
  + Addresses a broader class of systems.
  + Can handle actual code.
Perspectives

- Adapt policy iteration.
- Include this new domain in our Lustre static analyzer.
Thank you for your attention!
Related Work and Perspectives

Minimizing Condition Number

We can look for the roundest possible ellipsoid by minimizing a new variable $r$ in the extra constraint

$$I \preceq P \preceq rI.$$  

Rationale: “flat” ellipsoids can yield a very bad bound on one of the variables.
Another heuristic is to minimize $r \in (0, 1)$ in

$$A^T P A - rP \preceq 0.$$  

Intuitively, the shape of ellipsoid that gets “preserved” the best through the update $x_{k+1} = Ax_k$.
The two previous heuristics did not take $B$ into account.
The two previous heuristics did not take $B$ into account.

We could look for $P$ maximizing $r$ in $P \succeq rI$ such that

$$
\begin{pmatrix}
-A^T PA & -A^T PBe_i \\
-e_i^T B^T PA & 1 - e_i^T B^T PBe_i
\end{pmatrix} - \tau_i \begin{pmatrix} -P & 0 \\ 0 & 1 \end{pmatrix} \succeq 0
$$

for all the vertices $e_i$ of the hypercube of dimension $p$. 
The two previous heuristics did not take $B$ into account.

We could look for $P$ maximizing $r$ in $P \succeq rl$ such that

$$
- \begin{pmatrix}
  -A^T PA & -A^T PBe_i \\
  -e_i^T B^T PA & 1 - e_i^T B^T PBe_i
\end{pmatrix}
- \tau_i \begin{pmatrix}
  -P \\
  0
\end{pmatrix} \succeq 0
$$

for all the vertices $e_i$ of the hypercube of dimension $p$.

Previous equations amounts to the smallest ellipsoid such that

$$
\forall x, \forall u, \|u\|_\infty \leq 1 \Rightarrow x^T P x \leq 1 \Rightarrow (Ax + Bu)^T P (Ax + Bu) \leq 1,
$$

i.e. theoretically the best solution.
The two previous heuristics did not take $B$ into account.

We could look for $P$ maximizing $r$ in $P \succeq rl$ such that

$$\begin{pmatrix}
-A^T PA & -A^T PBe_i \\
-e_i^T B^T PA & 1 - e_i^T B^T PBe_i
\end{pmatrix} - \tau_i \begin{pmatrix}
-P & 0 \\
0 & 1
\end{pmatrix} \succeq 0$$

for all the vertices $e_i$ of the hypercube of dimension $p$.

Previous equations amounts to the smallest ellipsoid such that

$$\forall x, \forall u, \|u\|_\infty \leq 1 \Rightarrow x^T P x \leq 1 \Rightarrow (Ax + Bu)^T P (Ax + Bu) \leq 1,$$

i.e. theoretically the best solution.

But in practice, it’s hard to find a good solution.
Rounding Errors in the Program

Knowing the precision of the floating point system used and the dimensions of matrices \( A \) and \( B \) of the analyzed system, we can compute two reals \( a \) and \( b \) such that if

\[
(Ax + Bu)^T P (Ax + Bu) \leq \lambda
\]

then

\[
\text{fl}(Ax + Bu)^T P \text{fl}(Ax + Bu) \leq a^2 \lambda + 2ab\sqrt{\lambda} + b^2
\]

with \( \text{fl}(e) \) the computation of \( e \) in any order and with any IEEE754 rounding mode (in practice \( a \) is near from 1 and \( b \) from 0).
Soundness of the result

- Checking the soundness of the result basically amounts to checking positive definiteness of a matrix.
- This is done by carefully bounding the rounding errors in a Cholesky decomposition.
- Hence an efficient soundness check (in $O(n^3)$ for an $n \times n$ matrix).