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

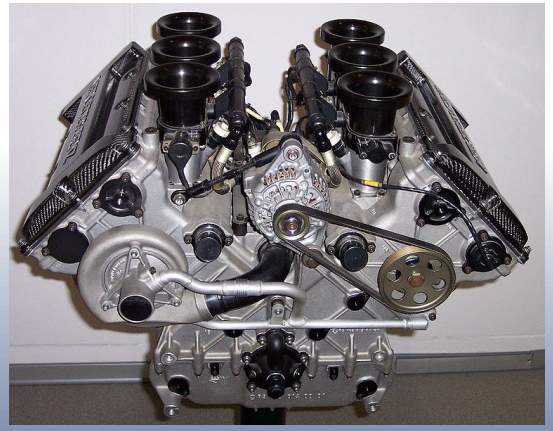
[1]ONERA Toulouse, France

[2]Georgia Tech Atlanta, Georgia, USA

Control-Command Critical Systems

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] = (A[0])*St[0] + (A[1])*St[1] + (A[2])*St[2] + (A[3])*St[3];
    xnew[0] += (B[0])*INPUT[0] + (B[1])*INPUT[1];
    xnew[1] = (A[4])*St[0] + (A[5])*St[1] + (A[6])*St[2] + (A[7])*St[3];
    xnew[1] += (B[2])*INPUT[0] + (B[3])*INPUT[1];
    xnew[2] = (A[8])*St[0] + (A[9])*St[1] + (A[10])*St[2] + (A[11])*St[3];
    xnew[2] += (B[4])*INPUT[0] + (B[5])*INPUT[1];
    xnew[3] = (A[12])* St[0] + (A[13])*St[1] + (A[14])*St[2] + (A[15])*St[3];
    xnew[3] += (B[6])*INPUT[0] + (B[7])*INPUT[1];
    (void) memcpy(St, xnew, sizeof(real_T)*4);
}
```

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] = (A[0])*St[0] + (A[1])*St[1] + (A[2])*St[2] + (A[3])*St[3];
    xnew[0] += (B[0])*INPUT[0] + (B[1])*INPUT[1];
    xnew[1] = (A[4])*St[0] + (A[5])*St[1] + (A[6])*St[2] + (A[7])*St[3];
    xnew[1] += (B[2])*INPUT[0] + (B[3])*INPUT[1];
    xnew[2] = (A[8])*St[0] + (A[9])*St[1] + (A[10])*St[2] + (A[11])*St[3];
    xnew[2] += (B[4])*INPUT[0] + (B[5])*INPUT[1];
    xnew[3] = (A[12])* St[0] + (A[13])*St[1] + (A[14])*St[2] + (A[15])*St[3];
    xnew[3] += (B[6])*INPUT[0] + (B[7])*INPUT[1];
    (void) memcpy(St, xnew, sizeof(real_T)*4);
}
```

Result:

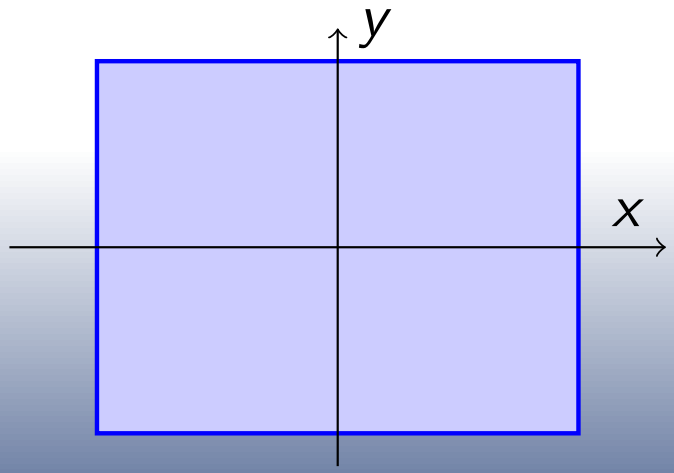
- if INPUT[0..1] remains in $[-1, 1]$
- then St[0..3] remains in [-5, 5]

Quadratic invariants

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

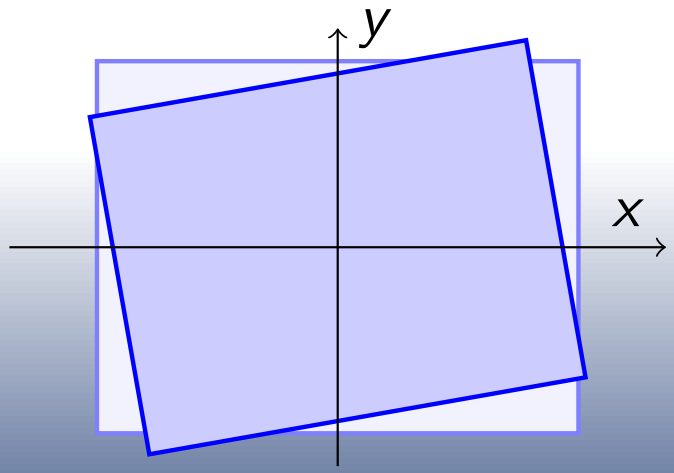
Quadratic invariants

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



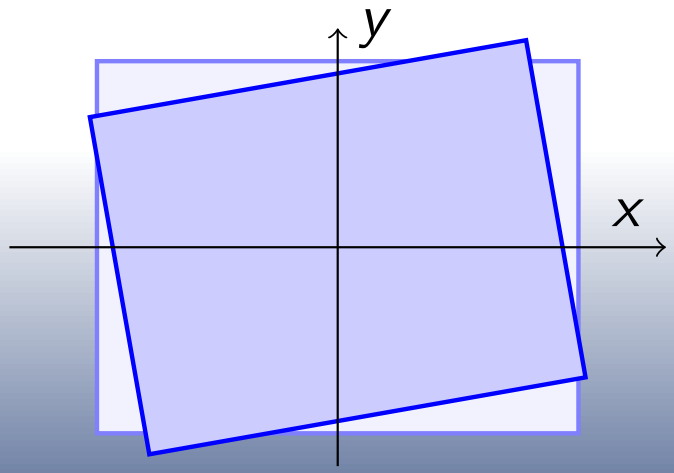
Quadratic invariants

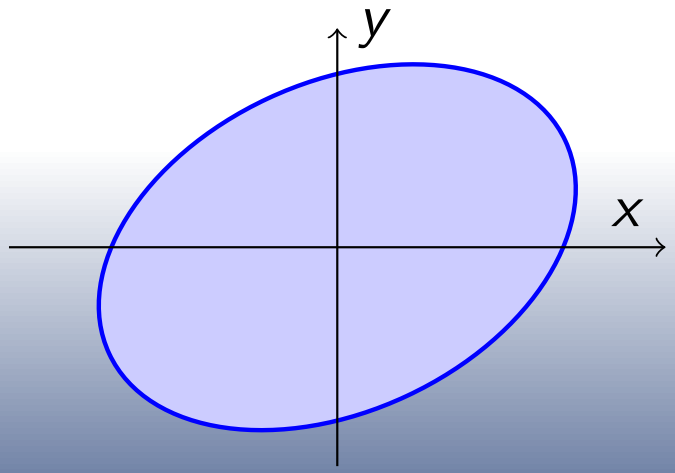
- **Linear invariants** commonly used in static analysis are not well suited:
  - at best costly;
  - at worst no result.



Quadratic invariants

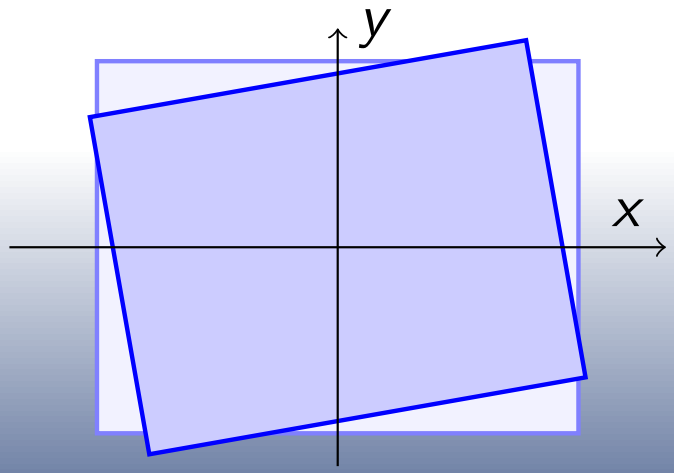
- **Linear invariants** commonly used in static analysis are not well suited:
  - at best costly;
  - at worst no result.
- Control theorists know for long that **quadratic invariants** are a good fit for linear systems.

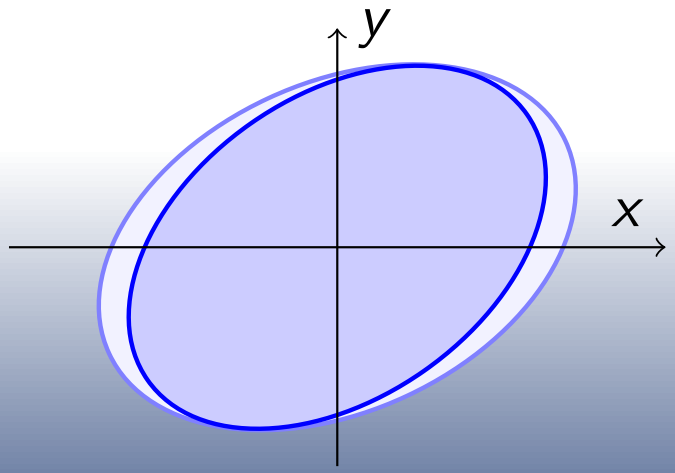




Quadratic invariants

- **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.





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 $x \in \mathbb{R}^n$: $x \neq 0 \Rightarrow x^T P x < 0$.

Lyapunov Stability, Invariant

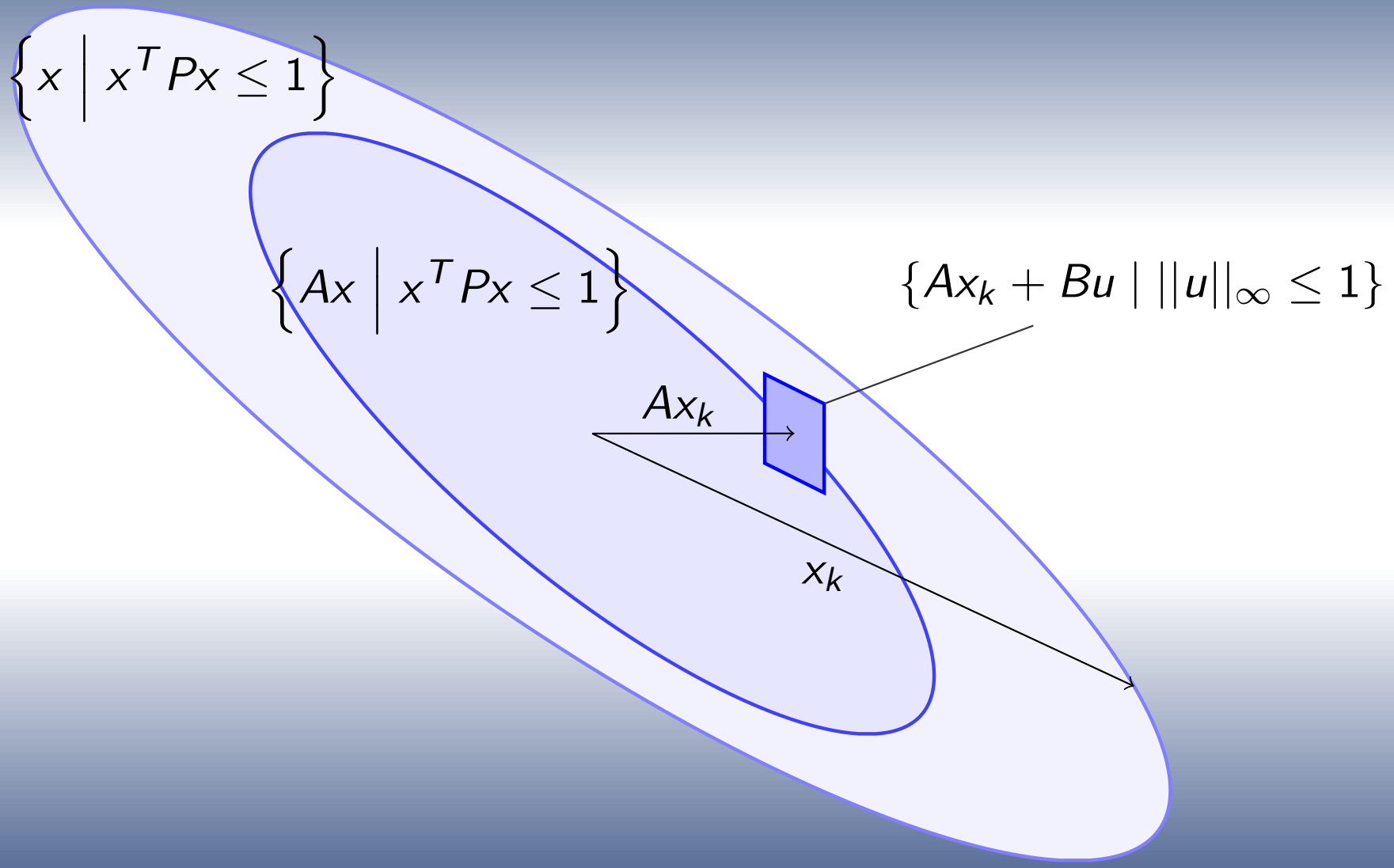
Invariant ellipsoid

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

In Computer Science Language

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

Lyapunov Stability, Illustration



Method

Overall Method

1. First determine the **shape** of the ellipsoid by choosing a matrix P;
2. then find the smallest possible **ratio** λ such that $x^T P x \leq \lambda$ is an invariant.

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.

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.

What are we looking for ?

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

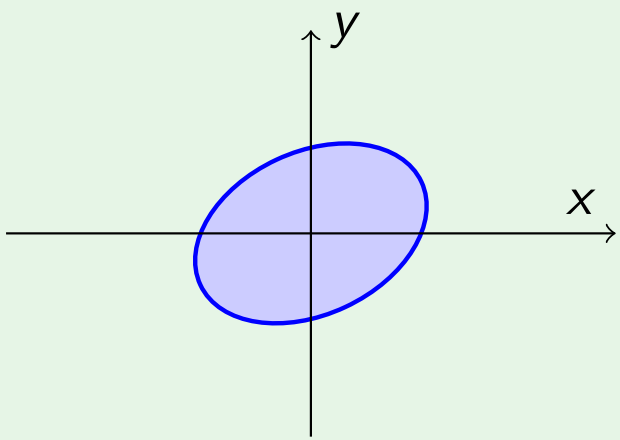
What are we looking for ?

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

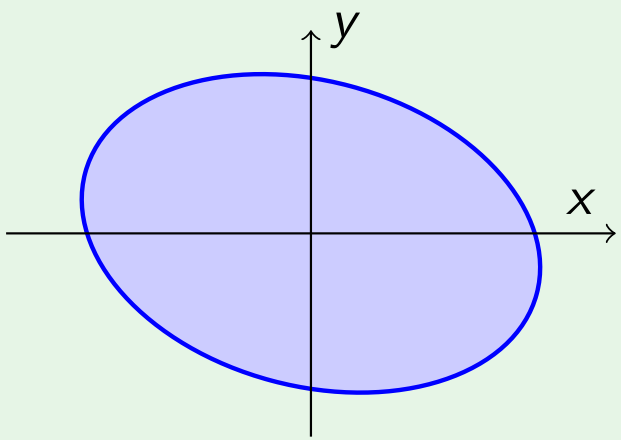
What are we looking for ?

- 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 $\left\{x \mid x^T P x \leq \lambda\right\}$ should be small enough.

Example



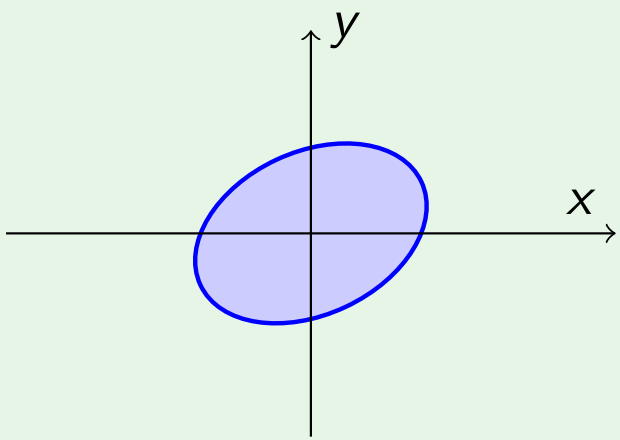
is better than



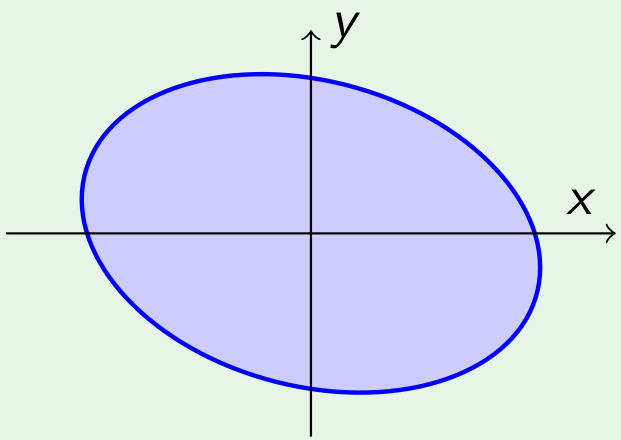
What are we looking for ?

- 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 $\left\{x \mid x^T P x \leq \lambda\right\}$ should be small enough.

Example



is better than



→ Tested three heuristics.

Rounding Errors

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

Example

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

Rounding Errors

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

Example

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

Gives 100958!

Rounding Errors

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

Example

```
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;

Rounding Errors

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

Example

```
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.

Our Prototype

- OCaml front-end to Scilab (lmisolver function).
- Only 2.5 kloc.
- Available under GPL at http://cavale.enseeiht.fr/.

Experimental Results

	Shape	Bounds		Valid.
Ex. 1 n=2, 1 input	0.07	[140.4; 189.9]	0.40	0.01
	0.16	[22.2; 26.5]	0.28	0.01
	0.23	[16.2; 17.6]	0.20	0.01
Ex. 2 n=4, 1 input	0.09	[18.1; 25.2; 24.3; 33.7]	0.40	0.01
	0.27	[6.3; 7.7; 2.2; 3.4]	0.27	0.02
	0.40	[1.7; 2.0; 2.2; 2.5]	0.21	0.01
Ex. 3 lead-lag controller n=2, 1 input	0.07	$\perp$	$\perp$	$\perp$
	0.17	[36.2; 36.1]	0.33	0.01
	0.20	[38.8; 20.3]	0.20	0.01
Ex. 4 LQG regulator n=3, 1 input	0.09	[1.2; 0.9; 0.5]	0.32	0.02
	0.19	[0.9; 0.9; 0.9]	0.26	0.01
	0.24	[0.7; 0.4; 0.3]	0.22	0.02

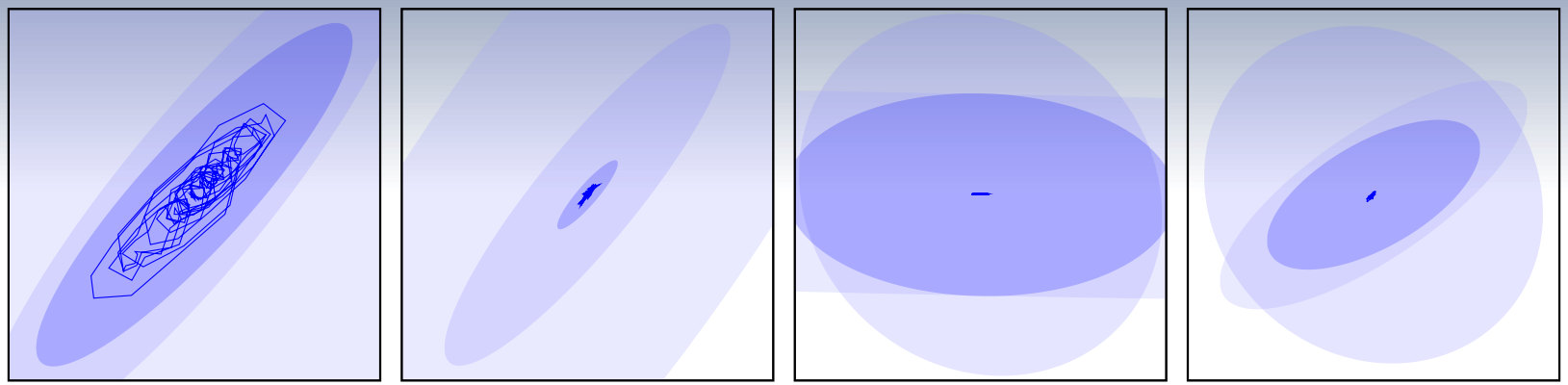
Analysis times (in s) and bounds compared for the three heuristics.

Experimental Results, continued

	Shape	Bounds		Valid.
Ex. 5 coupled	0.09	$[9.8; 8.9; 11.0; 16.8]$	0.43	0.03
mass system	0.24	$[5.7; 5.6; 6.4; 10.1]$	0.33	0.03
n=4, 2 inputs	0.48	$[5.0; 4.9; 4.8; 4.7]$	0.22	0.03
Ex. 6 Butterworth	0.10	$[7.5; 8.7; 6.1; 7.0; 6.5]$	0.38	0.03
low-pass filter	0.32	$[3.6; 5.0; 4.7; 8.1; 8.9]$	0.29	0.02
n=5, 1 input	0.78	$[2.3; 1.1; 1.9; 2.0; 2.9]$	0.24	0.03
Ex. 7 Dampened	0.07	$[1.7; 2.1]$	0.23	0.01
oscillator	0.15	$[2.0; 2.0]$	0.20	$\perp$
n=2, no input	0.27	$[1.5; 1.5]$	0.16	0.01
Ex. 8 Harmonic	0.08	$[1.5; 1.5]$	0.23	0.01
oscillator	0.24	$[1.5; 1.5]$	0.20	$\perp$
n=2, no input	0.15	$[1.5; 1.5]$	0.16	0.01

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

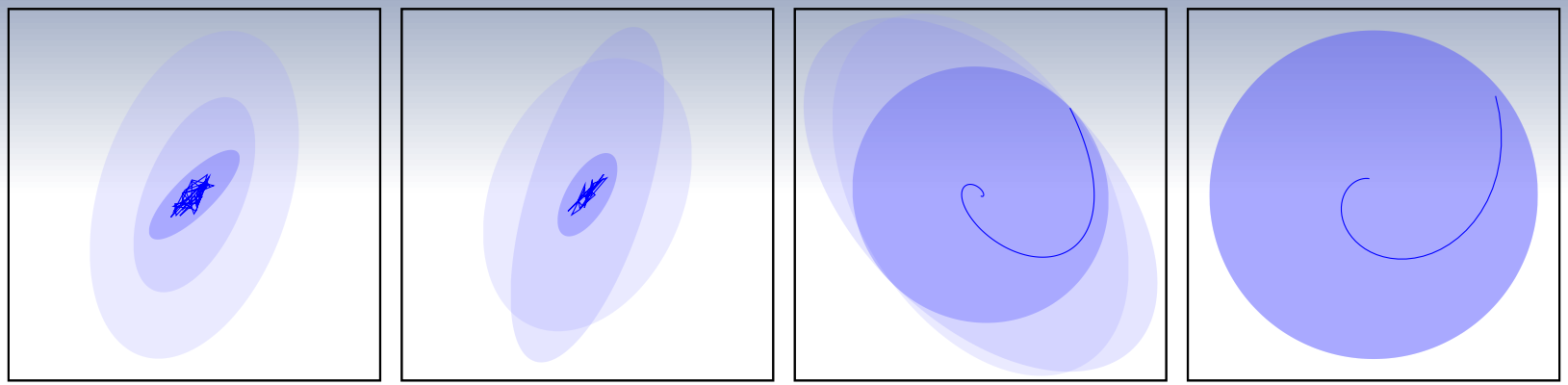
Experimental Results, continued



(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).

Experimental Results, end



(a) Ex. 5 (b) Ex. 6 (c) Ex. 7 (d) Ex. 8

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).

Related Work

- [Feret04]
  - − Adresses 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.
  - + Adresses a broader class of systems.
  - + Can handle actual code.

Perspectives

- Adapt policy iteration.
- Include this new domain in our Lustre static analyzer.

Questions

Thank you for your attention!

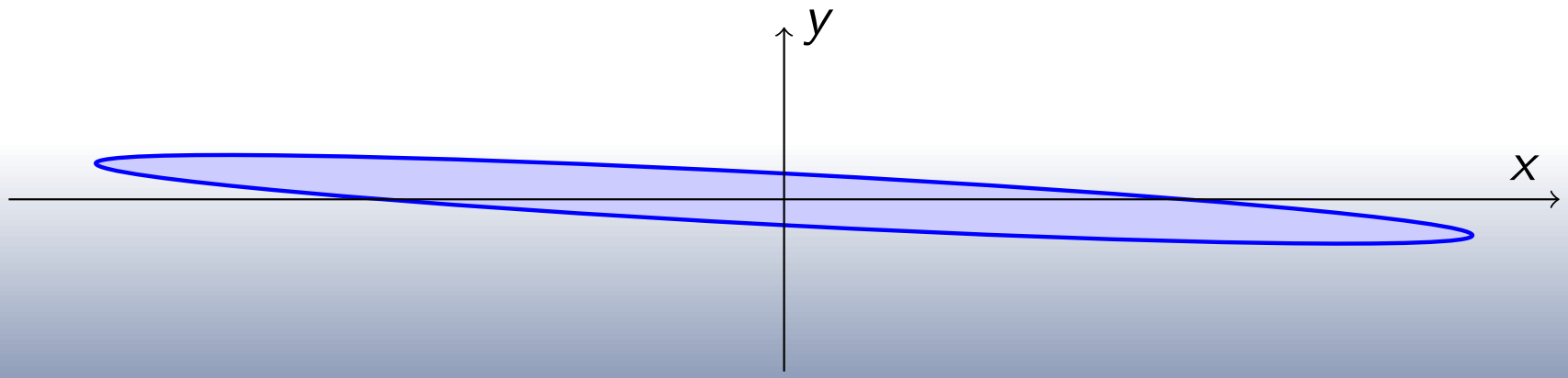
?

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.

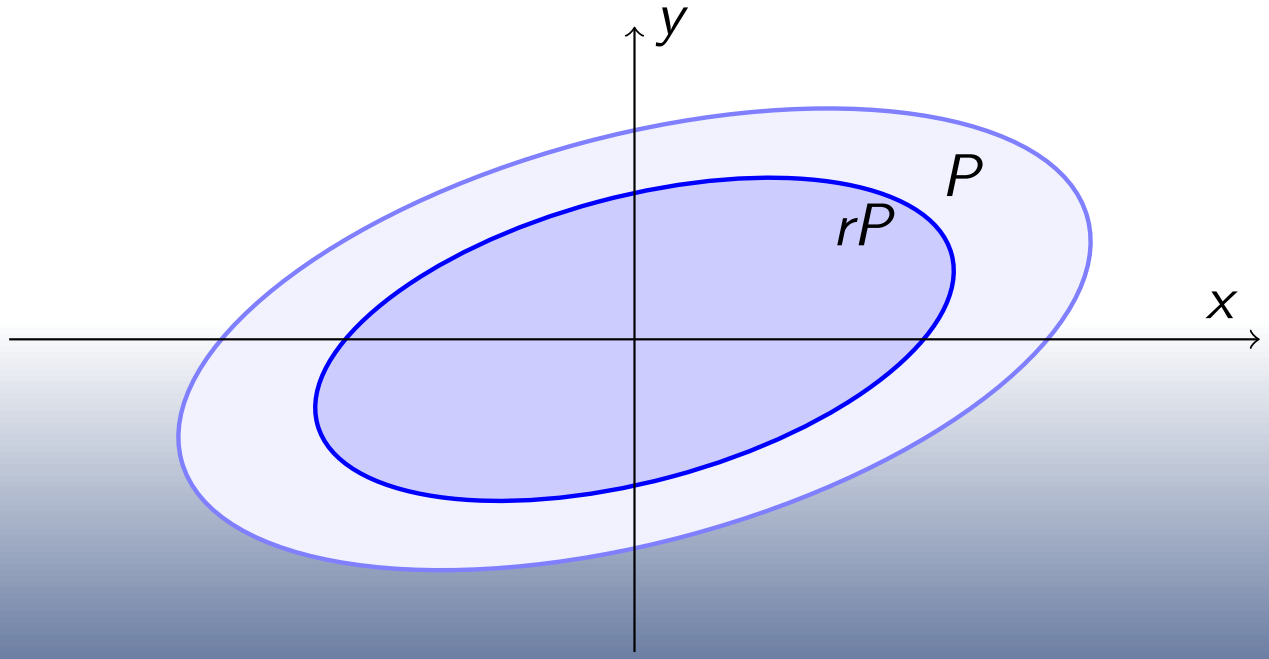


Preserving the Shape

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 $x_{k+1} = Ax_k$.



All in One

- The two previous heuristics did not take B into account.

All in One

- 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.

All in One

- 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.

+ 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.

All in One

- 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.

+ 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

$$\mathrm{fl}(Ax + Bu)^T P \, \mathrm{fl}(Ax + Bu) \leq a^2\lambda + 2ab\sqrt{\lambda} + b^2$$

with $\mathrm{fl}(e)$ 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 $\mathrm{O}(n^3)$ for an $n \times n$ matrix).