Sound Numeric Computations in Abstract Acceleration

Dario Cattaruzza Alessandro Abate Peter Schrammel
Daniel Kroening

Department of Computer Science, University of Oxford

School of Engineering and Informatics, University of Sussex, UK

July 22, 2017

Linear Time Invariant (LTI) Systems.

We verify safety for Linear Loops in Real Domains in the form:

$$\textit{while}\ (k \leq \overline{k})\ \ \boldsymbol{x} = \boldsymbol{A}\boldsymbol{x}$$

- $\boldsymbol{x} \in \mathbb{R}^n$ are state variables
- $\boldsymbol{A} \in R^{n \times n}$ are the dynamics of the system.
- verification is performed using floating point computations.

Linear Time Invariant (LTI) Systems.

We verify safety for Linear Loops in Real Domains in the form:

$$\textit{while}\ (k \leq \overline{k})\ \ \boldsymbol{x} = \boldsymbol{A}\boldsymbol{x}$$

- $\boldsymbol{x} \in \mathbb{R}^n$ are state variables
- $\boldsymbol{A} \in R^{n \times n}$ are the dynamics of the system.
- verification is performed using floating point computations.
- This work applies to General Linear loops with Linear Guards and Inputs. For simplicity we will refer to the case above only.

Using floating points to evaluate reachability

$$\tilde{\boldsymbol{x}}_0 = \boldsymbol{x}_0$$

$$\textit{while } (k \leq \overline{k}) \;\; \tilde{\boldsymbol{x}} = \boldsymbol{A}\tilde{\boldsymbol{x}} + \boldsymbol{e}$$

$$\|\boldsymbol{e}\| \leq n\|\overline{\boldsymbol{\delta}}\| : \overline{\boldsymbol{\delta}} = [\overline{\delta_1} \;\; \cdots \overline{\delta_n}]^T$$

$$\overline{\delta_i} = \sup_j \left(|\delta(\boldsymbol{A}_{ij}\tilde{\boldsymbol{x}}_j : \tilde{\boldsymbol{x}} \in X)|\right)$$

Using eigenvalues to reduce the number of operations

Using the Eigendecomposition $\boldsymbol{A} = \boldsymbol{SJS}^{-1}$, where $\boldsymbol{J}$ is a bidiagonal matrix, we may replace the algorithm for:

Algorithm 1 Reals

1 : $\boldsymbol{x}'_0 = \boldsymbol{S}^{-1}\boldsymbol{x}_0$
2 : *while* $(k \leq \overline{k})$
3 : $\boldsymbol{x}'_k = \boldsymbol{J}\boldsymbol{x}'_{k-1}$
4 : $\boldsymbol{x}_{\overline{k}} = \boldsymbol{S}\boldsymbol{x}'_{\overline{k}}$

Algorithm 2 Floating Point

1 : $\tilde{\boldsymbol{x}}'_0 = \boldsymbol{S}^{-1}\boldsymbol{x}_0 + \boldsymbol{e}_{s1}$
2 : *while* $(k \leq \overline{k})$
3 : $\tilde{\boldsymbol{x}}'_k = \boldsymbol{J}\tilde{\boldsymbol{x}}'_{k-1} + \delta'$
4 : $\tilde{\boldsymbol{x}}_{\overline{k}} = \boldsymbol{S}\tilde{\boldsymbol{x}}'_{\overline{k}} + \boldsymbol{e}_{s2}$

This algorithm is approximately n times faster and has an error of $\frac{1}{n}$ if the eigendecomposition is precise.

Symbolic vs Numerical Eigendecomposition

- Symbolic Eigendecomposition ($n < 10$). $\overline{\boldsymbol{e}_s} \propto n\overline{\delta}$
- Numeric Eigendecomposition ($n < 10000$).
  - Narrow Bounds - Slower($1x$). $\overline{\boldsymbol{e}_s} \propto n^2\overline{\delta}$
  - Wide Bounds - Faster ($100x$). $\overline{\boldsymbol{e}_s} \propto n^n\overline{\delta}$
  - Using higher precision makes for a better trade-off.

Method	Dimension	Precision	Speed	Error Scale
Narrow	$n = 20$	128 bit	$2x$	10^{-37}
Wide	$n = 20$	128 bit	$1.01x$	10^{-15}
Narrow	$n = 20$	256 bit	$4x$	10^{-77}
Wide	$n = 20$	256 bit	$2.02x$	10^{-55}

Using Abstract Acceleration to reduce the number of operations

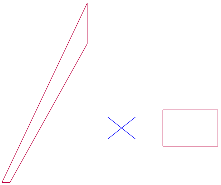
The set of points evaluated at each iteration is:

$$X'_k = \boldsymbol{J} X_{k-1} = \boldsymbol{J}^k X'_0$$

$$X' = \bigcup_{k \le \overline{k}} X'_k$$

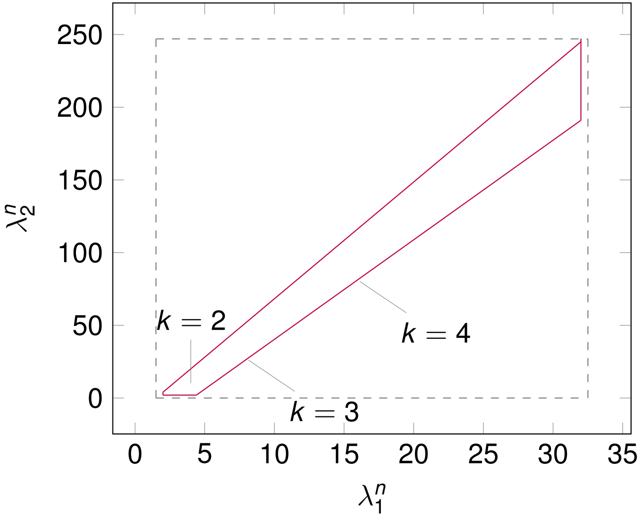
Which can be overapproximated by a single multiplication using an Abstract Matrix

$$X' \subseteq X^\sharp = \mathcal{J} X'_0, \text{ such that } \bigcup_{k \le \overline{k}} \boldsymbol{J}^k \subseteq \mathcal{J},$$

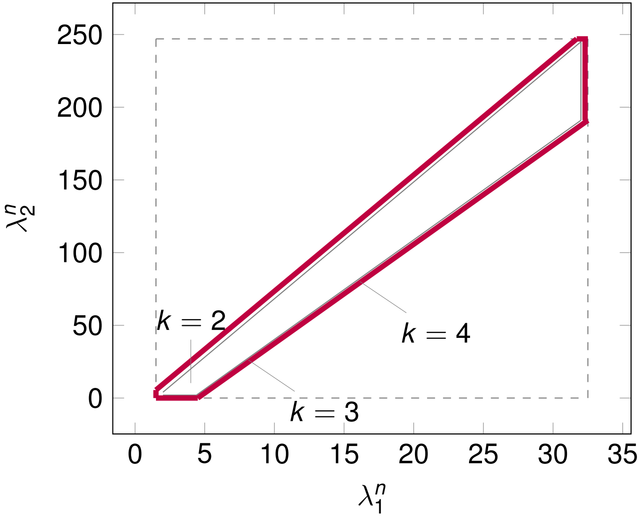


$\mathcal{J}$ X'_0

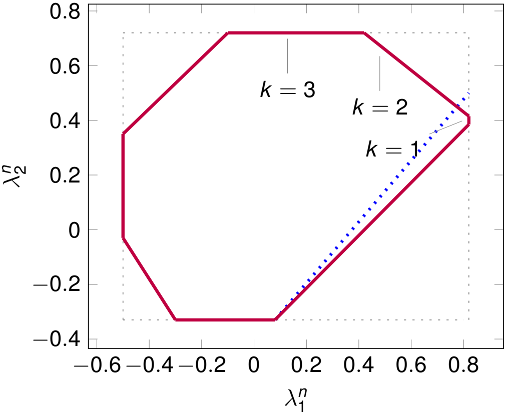
Abstract Matrices for positive real eigenvalues



Abstract Matrices for positive real eigenvalues

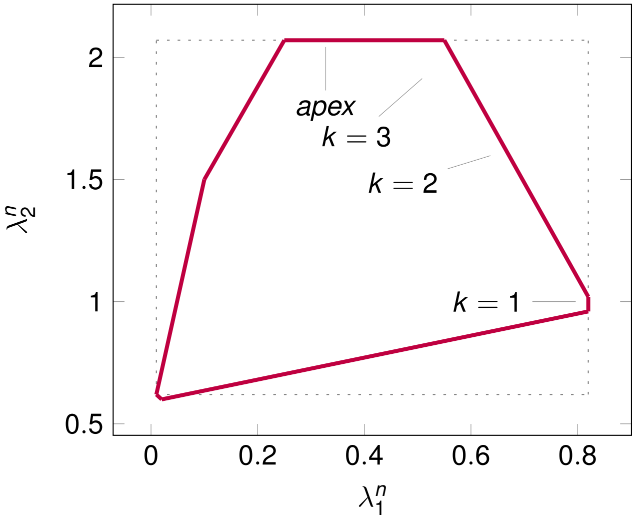


Abstract Matrices for Complex Eigenvalues



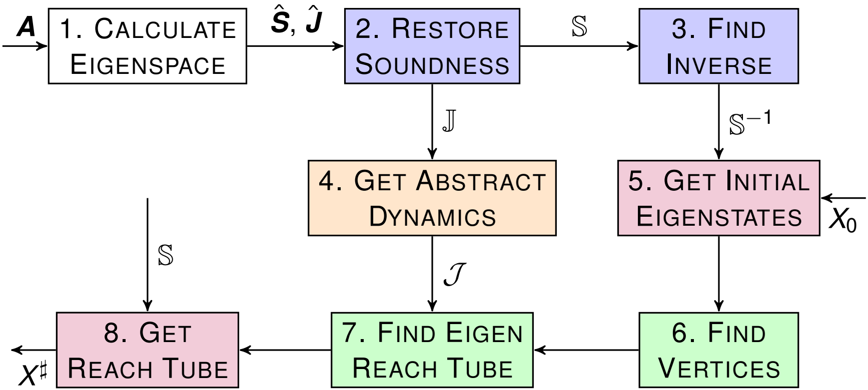
Polyhedral faces from an $\mathbb{R}^2$ complex conjugate subspace .

Abstract Matrices for Jordan blocks

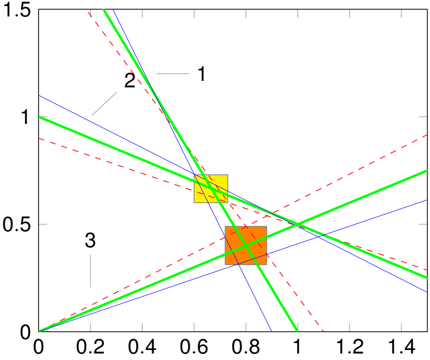


Polyhedral faces from an $\mathbb{R}^2$ Jordan block subspace .

Calculation of Abstract Reach Tube Using Numeric Abstract Acceleration



Pivot operations



Three interval half-planes with negative (dashed red), zero (thick green) and positive (thin blue) angular error representations. The yellow and orange areas (hypercubes) over-approximate all possible vertices of the resulting polyhedron at the given location. If these hypercubes partially intersect, the abstract vertex ι^k must necessarily contain all intersecting hypercubes.

Performance results

Benchmark	Dimension	Unsound		Sound	
		long double	mp	mpi	exact
Building	48	18.10s	185.03s	558.15s	$t.o.$
issr10	10	2.02s	23.46s	41.23s	$t.o.$
Convoy Car 3	6	0.30s	1.31s	3.60s	24.60s
Convoy Car 2	3	0.013s	0.033s	0.07s	5.46s
Parabola	4	0.012s	0.012s	0.05s	2.50s

Table: Axelerator[1] time performance on various benchmarks.
mp is the required precision for the algorithm using non-interval arithmetic
mpi is the sound algorithm

[1] www.cprover.org/LTI

Conclusions

- We have shown a numerical method for performing abstract acceleration using interval analysis
- It significantly improves the speed of the algorithm, allowing for user defined compromises that ensure scalability and soundness.
- The use of eigendecomposition and interval simplex can be applied to a number of approaches in order to achieve fast sound results.

The tool can be found at www.cprover.org/LTI