Computing the (almost) exact value of $\mu$ with the SMART library of the SMAC toolbox

C. Roos, J-M. Biannic

http://w3.onera.fr/smac
BRISTOL, UK
September 18th, 2017

International workshop on robust modeling, design and analysis: tools, methods and aerospace applications
Content

1. Brief introduction to robustness analysis

2. Focus on $\mu$-analysis

3. Presentation of the SMART library

4. Numerical results
Context and objective

Control laws design is usually based on a (linear) mathematical model, which significantly simplifies/alters reality.

Such a model is not a perfect representation of the real behavior of a physical system because of:

- high-frequency uncertainties (neglected dynamics),
- uncertainties on the parameters which characterize the system (e.g. mass, inertia, aerodynamic coefficients),
- time-varying parameters:
  - fast variations (e.g. mass of a launcher during atmospheric flight),
  - slow variations (e.g. mass, velocity, altitude of a transport aircraft),
- nonlinear phenomena:
  - aerodynamic phenomena at high angles,
  - actuators saturations,
  - transmission delays...

Robustness with respect to these phenomena must be ensured!
Classical industrial approach

Monte-Carlo simulations:

1. choose many random samples, each of them being composed of random operating points (e.g. mass configurations, CoG position...) and random inputs (e.g. pilot inputs, wind...),
2. perform a closed-loop simulation for each sample,
3. perform statistical analysis on the resulting output samples to get the probabilities that some stability, performance, loads, comfort... criteria are satisfied.

Advantage:
- easy to implement

Drawbacks:
- exponential-time approach ⇒ high computational complexity
- statistical approach ⇒ worst cases can be missed
New trend

Develop some **inexpensive tools** to determine quickly the most critical parametric configurations **without performing extensive simulations**.

Many robustness analysis problems can be formulated as **optimization problems**, where it must be checked that a set of criteria lie within certain limits for all admissible system configurations.

Some techniques such as $\mu$, IQC-based or Lyapunov-based analysis can be **efficient alternatives** to Monte-Carlo simulations.
Different analysis techniques

**Preliminary task:** transform the uncertain/time-varying/nonlinear model of the system into a Linear Fractional Representation, e.g. using the GSS & APRICOT libraries of the SMAC Toolbox.

Several system components can be isolated in the $\Delta$ block, such as:

- parametric / dynamic uncertainties
- time-varying / time-invariant parameters
- non-linearities (saturations, deadzones, sector non-linearities...
Different analysis techniques

The selected analysis technique depends on the elements in \( \Delta \):

- **time-invariant uncertainties:** \( \mu \)-analysis
  - \( \rightarrow \) SMART library (Skew Mu Analysis based Robustness Tools)

- **uncertainties, time-varying parameters & non-linearities:** IQC-based analysis
  - \( \rightarrow \) IQC library (Integral Quadratic Constraints)
  - \( \rightarrow \) SeDuMi-based IQC solver

- **saturations, deadzones and sector non-linearities:** Lyapunov-based analysis and nonsmooth multi-objective \( H_\infty \) optimization
  - \( \rightarrow \) SAW library (Saturated systems analysis and Anti-Windup design)
  - \( \rightarrow \) OISTER library (Out to In Saturation Transformation Extensions for Robustness)

All these tools are available on the SMAC website: [http://w3.onera.fr/smac](http://w3.onera.fr/smac)
Content

1. Brief introduction to robustness analysis

2. Focus on $\mu$-analysis
   - Problem statement
   - Standard computational approach

3. Presentation of the SMART library

4. Numerical results
Problem statement

Stability/performance of LTI systems with time-invariant uncertainties.

\[ M(s) \text{ is a stable and proper real-rational transfer function } \Rightarrow \text{ nominal system.} \]

\[ \Delta(s) = \text{diag}(\Delta_1(s), \ldots, \Delta_N(s)) \text{ is a block-diagonal operator } \Rightarrow \text{ model uncertainties.} \]

\[ \Delta_i(s) \text{ can be:} \]

\[ \begin{align*}
\text{a time-invariant diagonal matrix } & \Delta_i(s) = \delta_i I_{n_i}, \text{ where } \\
\text{a stable and proper real-rational unstructured transfer function representing neglected dynamics.}
\end{align*} \]

\[ \Delta \text{ is the set of all matrices with the same block-diagonal structure and the same nature (real or complex) as } \Delta(j\omega) \Rightarrow \text{ admissible uncertainties.} \]

\[ k\mathcal{B}_\Delta = \{ \Delta \in \Delta : \sigma(\Delta) < k \} \Rightarrow \text{ maximum size of the uncertainties.} \]
Robust stability margin

Definition of the structured singular value

$$\mu_{\Delta}(M(j\omega)) = \left[ \min_{\Delta \in \Delta} \{ \sigma(\Delta), \det(I - M(j\omega)\Delta) = 0 \} \right]^{-1}$$

Small gain theorem for structured uncertainties

The interconnexion $M(s) - \Delta(s)$ is stable $\forall \Delta(s) \in k\mathcal{B}_\Delta$ if and only if:

$$\sup_{\omega \in \mathbb{R}_+} \mu_{\Delta}(M(j\omega)) \leq \frac{1}{k}$$

Definition of the robust stability margin

$$k_r = \left[ \sup_{\omega \in \mathbb{R}_+} \mu_{\Delta}(M(j\omega)) \right]^{-1}$$

- the interconnexion is stable $\forall \Delta(s)$ of size strictly less than $k_r$
- $\exists \Delta(s)$ of size $k_r$ for which the interconnexion is unstable
Standard computational approach

The exact computation of the robust stability margin $k_r$ is NP hard in the general case, so both lower and upper bounds are computed instead:

- a lower bound (i.e. a $\mu$ upper bound $\overline{\mu}_\Delta$) provides a guaranteed but conservative value of $k_r$,
- an upper bound (i.e. a $\mu$ lower bound $\underline{\mu}_\Delta$), usually associated to a worst-case parametric configuration, allows to quantity the conservatism of the lower bound.

But even computing these bounds is a challenging problem with an infinite number of frequency-domain constraints.

It is usually solved on a finite frequency grid $\{\omega_i\}_{i \in [1,M]}$ and an estimate of the robust stability margin is then obtained as:

$$
\frac{1}{\max_{i \in [1,M]} (\overline{\mu}_\Delta(M(j\omega_i)))} \leq k_r \leq \frac{1}{\max_{i \in [1,M]} (\underline{\mu}_\Delta(M(j\omega_i)))}
$$
Limitations of the standard approach

Problem

The grid must contain the **most critical frequency point** for which the maximal value of $\mu_\Delta$ is reached.

If not:

- The upper bound on $k_r$ can be **very poor**, notably for flexible systems, for which the function $\mu_\Delta(M(j\omega))$ often exhibits very high and narrow peaks.

- Even worse, the lower bound on $k_r$ can be **over-evaluated**, i.e. be larger than the real value of $k_r$.

Unfortunately, the aforementioned critical frequency is usually unknown!

In this context, the considered frequency grid must be **sufficiently dense**, which can lead to a **prohibitive computational cost**.

But even so, **it is still possible to miss a critical frequency**...
Content

1. Brief introduction to robustness analysis

2. Focus on $\mu$-analysis

3. Presentation of the SMART library

4. Numerical results
Objectives

Main purposes of the **SMART library**

- **implement state-of-the-art algorithms** to compute the (almost) **exact value of $\mu$** in (almost) all cases with a low computational time, even for large size problems addressed by control engineers.

- **ensure that reliable results are obtained**, *i.e.* that no critical frequency is missed and that the computed margins are guaranteed on the whole frequency range.

- **solve a wide class of problems**, including (skewed) robust stability margin, worst-case $H_\infty$ performance level and worst-case gain/phase/modulus/delay margins.

- **propose a user-friendly Matlab toolbox**, which can be used both by researchers and by control engineers.

⇒ Go beyond the **Robust Control Toolbox**, which can prove useful but:

- only addresses a **limited number of issues**,
- does not propose any solution when the gap between the bounds on $\mu$ is large,
- usually considers a **finite frequency grid** instead of the whole frequency range.
An alternative approach

To achieve these goals, and compute both tight and reliable bounds on $k_r$, alternative methods are implemented in the SMART library [MSC 2011, MSC 2013]:

- **An $\mu$ upper bound** is first determined at some frequency. A **hamiltonian-based technique** is then applied to determine all frequency intervals on which this bound remains valid. This strategy is repeated and a **guaranteed lower bound on $k_r$** is finally obtained when the union of all intervals covers the whole frequency range.

- **A poles migration technique** is applied to compute the smallest possible perturbation $\hat{\Delta}(s) \in \Delta$ such that the interconnection between $M(s)$ and $\hat{\Delta}(s)$ is unstable. A $\mu$ lower bound, and thus a **guaranteed upper bound on $k_r$**, is obtained. Frequency is an optimization parameter, which allows to detect critical frequencies and usually leads to very accurate bounds.

- **A branch & bound algorithm** is implemented, which **tightens the gap between the bounds on $k_r$** to the desired accuracy at a reasonable computational cost.
Addressed issues

Different kinds of problems can be solved with the SMART library:

1. **robust stability margin** - Compute the robust stability margin $k_r$ for a given block structure $\Delta$.

2. **skewed robust stability margin** - Compute the skewed robust stability margin for a given block structure $\Delta$.

3. **worst-case $H_\infty$ performance level** - Compute the highest value of the $H_\infty$ norm of the transfer matrix from $e$ to $y$ when $\Delta(s)$ takes all possible values in $B_\Delta$ [MSC 2010].

4. **worst-case input-output margins** - Compute the worst-case gain, modulus, phase and delay margins, *i.e.* the highest value of the (real or complex) gain, the phase shift or the time delay that can be inserted between $y$ and $e$ without destabilizing the system when $\Delta(s)$ takes all possible values in $B_\Delta$ [CDC 2011].
About the Matlab code

The SMART library has been tested on systems with up to 100 states and 500 uncertainties (counting repetitions). It usually gives very accurate results, only requires a moderate computational effort, and has proved successful in many real-world applications.

- It can be used on all platforms (Linux, Mac and Windows).
- All routines run on Matlab R2010b or higher. They should run on older releases, but no validation has been performed.
- It is fully compatible with gss, lfr and uss objects.
- Each routine is documented and a complete on-line documentation is available.

The full version is freely available on the SMAC website
http://w3.onera.fr/smac/smart
Content

1. Brief introduction to robustness analysis

2. Focus on $\mu$-analysis

3. Presentation of the SMART library

4. Numerical results
   - $\mu$ lower bound computation
   - $\mu$ upper bound computation
   - Branch & bound to master conservatism
   - How to reduce computational time
   - Approaching the exact value of $\mu$
List of benchmarks

A large set of **36 challenging benchmarks** is considered:

- a few academic systems and many real-world applications
- purely real, mixed real/complex or purely complex uncertainties
- **large number of states** (from 2 up to 70)
- **large number of uncertainties** (from 1 up to 28), repeated or not, even highly repeated in some cases (size of $\Delta$ up to $507 \times 507$)
- **both rigid & highly flexible models** (aircraft, telescope mockup, satellite...)
- **several fields of application** (civilian & fighter aircraft, launcher, re-entry vehicle, satellite, telescope, helicopter, spacecraft, missile, hard disk drive, biochemical network, car, hydraulic servo system, spark ignition engine...)

32 benchmarks are available in the control literature. The other 4 ones have been developed by ONERA in cooperation with industrial partners.
$k_r$ upper bound computation ($\Leftrightarrow \mu$ lower bound)

All existing polynomial-time $\mu$ lower bound algorithms are applied to compute upper bounds on $k_r$ for each benchmark [CEP 2015].

<table>
<thead>
<tr>
<th>Description</th>
<th>Uncertainties</th>
<th>Matlab code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Power algorithm</td>
<td>all</td>
<td>Robust Control Toolbox</td>
</tr>
<tr>
<td>2. Gain-based algorithm</td>
<td>all but complex</td>
<td>Robust Control Toolbox</td>
</tr>
<tr>
<td>3. Poles migration technique</td>
<td>all</td>
<td>Carsten Döll</td>
</tr>
<tr>
<td>4. Poles migration technique</td>
<td>real</td>
<td>SMART library</td>
</tr>
<tr>
<td>5. Poles migration technique</td>
<td>all</td>
<td>Mark Halton</td>
</tr>
<tr>
<td>6. Direct nonlinear optimization</td>
<td>all</td>
<td>Mark Halton</td>
</tr>
<tr>
<td>7. Direct nonsmooth optimization</td>
<td>all</td>
<td>Alberto Simoes</td>
</tr>
<tr>
<td>8. Geometrical approach</td>
<td>real</td>
<td>Jongrae Kim</td>
</tr>
</tbody>
</table>
## Results for purely real problems (1-29)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Number of times the gap w.r.t. the best $\mu$ lower bound is</th>
<th>Mean gap w.r.t. the best $\mu$ lower bound</th>
<th>Mean CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\leq 0%$</td>
<td>$\leq 5%$</td>
<td>$\leq 25%$</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>9</td>
<td>13</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>19</td>
<td>24</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>16</td>
<td>22</td>
</tr>
<tr>
<td>4</td>
<td>26</td>
<td>27</td>
<td>29</td>
</tr>
<tr>
<td>5</td>
<td>22</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>6 (g)</td>
<td>5</td>
<td>18</td>
<td>24</td>
</tr>
<tr>
<td>6 (i)</td>
<td>9</td>
<td>20</td>
<td>23</td>
</tr>
<tr>
<td>7 (g)</td>
<td>8</td>
<td>17</td>
<td>23</td>
</tr>
<tr>
<td>7 (i)</td>
<td>24</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>8</td>
<td>17</td>
</tr>
</tbody>
</table>

The most relevant algorithm is the poles migration technique of Ferreres & Biannic [CEP 2001] with both the highest accuracy and the lowest CPU time.
Improvement for purely real problems

Improving the poles migration technique is not a trivial issue.

Idea: combine several algorithms

1. algorithm 4 is executed first (most efficient technique in almost all cases),
2. algorithm 2 is then executed only for a few selected frequencies with previous results as initialization,
3. particle swarm optimization is finally applied with previous results as initialization.

This strategy is implemented in the **SMART library**.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Algorithm 4</th>
<th>Other algorithms</th>
<th>Combination</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>value</td>
<td>time</td>
<td>best value</td>
</tr>
<tr>
<td>20</td>
<td>0.9380</td>
<td>0.5 s</td>
<td>0.9947</td>
</tr>
<tr>
<td>26</td>
<td>0.9881</td>
<td>2.2 s</td>
<td>1.2134</td>
</tr>
<tr>
<td>29</td>
<td>724.15</td>
<td>46.9 s</td>
<td>733.86</td>
</tr>
</tbody>
</table>

The best lower bound is obtained for all benchmarks with this strategy.
Results for purely complex and mixed problems (30-36)

Only algorithms 1-3-5-6-7 can be applied.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Number of times the gap w.r.t. the best $\mu$ lower bound is</th>
<th>Mean gap w.r.t. the best $\mu$ lower bound</th>
<th>Mean CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4 $=0%$, 7 $\leq 5%$, 7 $\leq 25%$</td>
<td>0.10$%$</td>
<td>1.1 s</td>
</tr>
<tr>
<td>3</td>
<td>0 $=0%$, 4 $\leq 5%$, 6 $\leq 25%$</td>
<td>16.60$%$</td>
<td>0.9 s</td>
</tr>
<tr>
<td>5</td>
<td>2 $=0%$, 2 $\leq 5%$, 3 $\leq 25%$</td>
<td>57.12$%$</td>
<td>140.8 s</td>
</tr>
<tr>
<td>6 (g)</td>
<td>1 $=0%$, 7 $\leq 5%$, 7 $\leq 25%$</td>
<td>0.34$%$</td>
<td>2648.8 s</td>
</tr>
<tr>
<td>6 (i)</td>
<td>0 $=0%$, 5 $\leq 5%$, 7 $\leq 25%$</td>
<td>4.26$%$</td>
<td>874.2 s</td>
</tr>
<tr>
<td>7 (g)</td>
<td>4 $=0%$, 4 $\leq 5%$, 6 $\leq 25%$</td>
<td>10.63$%$</td>
<td>3972.6 s</td>
</tr>
<tr>
<td>7 (i)</td>
<td>2 $=0%$, 5 $\leq 5%$, 6 $\leq 25%$</td>
<td>7.91$%$</td>
<td>249.5 s</td>
</tr>
</tbody>
</table>

The most relevant algorithm is the **power algorithm of Young & Doyle [TAC 1997]** with the highest accuracy and almost the lowest CPU time.
Improvement for purely complex and mixed problems

Idea: better exploit the power algorithm

1. Algorithm 1 is applied on a rough frequency grid (e.g. 20 frequency points)
2. The grid is gradually tightened around the peak frequencies until improvement in the $\mu$ lower bound becomes marginal

At each frequency, algorithm 1 is not only initialized with the best result obtained at the previous frequency, but also with one or more random values.

This strategy is implemented in the SMART library.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Algorithm 1</th>
<th>Other algorithms</th>
<th>Improved algo 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>value</td>
<td>time</td>
<td>best value</td>
</tr>
<tr>
<td>33</td>
<td>0.4346</td>
<td>0.7 s</td>
<td>0.4362</td>
</tr>
<tr>
<td>34</td>
<td>0.9587</td>
<td>1.1 s</td>
<td>0.9604</td>
</tr>
<tr>
<td>35</td>
<td>0.9910</td>
<td>1.7 s</td>
<td>0.9927</td>
</tr>
</tbody>
</table>

The best lower bound is obtained for all benchmarks with this strategy.
What about **conservatism**, *i.e.* the gap w.r.t. the exact value of $k_r$?

A lower bound on $k_r$ is computed for each of the 36 benchmarks using the **hamiltonian-based technique of Biannic et al** [IFAC 2005] and Roos & Biannic [ACC 2010] implemented in the **SMART library**.

The mean gap between the best $k_r$ lower and upper bounds is:

- **12.71%** for purely real problems
- **0.39%** for purely complex and mixed real/complex problems

Is it the lower or the upper bound which is responsible for this gap?

**Claim**

The $k_r$ upper bound almost always equals the exact value of $k_r$.

No proof, but **true in many practical cases**!
Branch & bound to reduce conservatism...

**Purely complex and mixed problems:**
- For 6 benchmarks out of 7, the gap is less than 0.01%.
- For the last one, a standard **branch & bound algorithm** allows to increase the lower bound on $k_r$ until a gap less than 0.01% is obtained.

The exact value of $k_r$ is obtained in all cases.

**Purely real problems:**
- For 19 benchmarks out of 29, the gap is less than 0.01%.
- For the 10 others, a standard **branch & bound algorithm** allows to increase the lower bound on $k_r$, leading to a gap less than 0.01% in 7 cases, and of 0.12%, 1.99% and 10.00% in the last 3 cases.

The mean value of the gap over all 29 benchmarks is now only **0.42%**.

The exact value of $k_r$ is (almost) obtained in all except one case.
... at the price of an increase in computational time

**Standard branch & bound algorithm**

Cut the uncertainty domain in more and more subsets until the gap between
- the highest lower bound computed on all subsets and
- the highest upper bound computed on all subsets

becomes lower than a user-defined threshold $\eta$.

Branch & bound allows to **reduce conservatism to an arbitrarily small value**
for systems with only real parametric uncertainties.

**Problem**

*Computational complexity grows exponentially* as a function of the number of uncertainties. Computing tight $k_r$ lower bounds can be extremely long.
Towards a reduction of computational time

Improved branch & bound algorithm [MSC 2011]

Assume that for a given subset $\mathcal{D}_i$, stability can only be proved for frequencies $\omega \in \Omega_V \subset \mathbb{R}_+$:

→ $\mathcal{D}_i$ is partitioned,

→ for each subset of $\mathcal{D}_i$, the analysis is restricted to the frequency domain $\Omega_I = \mathbb{R}_+ \setminus \Omega_V$.

After a few steps, each analysis is restricted to narrow frequency intervals.

⇒ drastic reduction of computational cost, but still not sufficient...
Use of the $\mu$-sensitivities

The $\mu$-sensitivities aim at determining how much the $\mu$ upper bound is affected by a small variation of a single uncertainty in $\Delta$.

They can be used to identify which uncertainties have the largest influence on the $\mu$ upper bound.

Idea

Cut the uncertainties with the highest $\mu$-sensitivities only, so as to further improve the branch & bound algorithm.

The number of selected uncertainties allows to handle the tradeoff between accuracy and computational complexity [ROCOND 2015].

The same accuracy is obtained with a much lower computational time.

This branch & bound algorithm is implemented in the SMART library.
Approaching the exact value of $\mu$

For each benchmark, the computational time required to obtain a gap of either 1% or 0.1% between the $\mu$ upper and lower bounds (using the improved branch-and-bound algorithm if needed) is displayed.
Approaching the exact value of $\mu$

For 26 benchmarks out of 36, the computational time is negligible:

- less than 5 s for a gap of 0.1%

For 8 benchmarks, the computational time is low:

- between 8 s and 80 s for a gap of 1%
- between 10 s and 158 s for a gap of 0.1%

Only 2 benchmarks require a larger computational effort:

- launcher (30 states and $45 \times 45$ matrix): 9 min 30 s for a gap of 1%, much longer for a gap of 0.1%
- biochemical network (7 states and $507 \times 507$ matrix): 22 min 30 s for a gap of 10%, 3 h 04 min for a gap of 5%, 14 h 54 min for a gap of 1%, much longer for a gap of 0.1%

With few exceptions, computing the exact value of $\mu$ is no longer an issue.
Conclusion

The **SMART library** of the SMAC Toolbox implements the most efficient $\mu$-analysis based algorithms developed at ONERA during the past 15 years.

It allows to compute:

- the (skewed) structured singular value,
- the (skewed) robust stability margin,
- the worst-case $H_\infty$ performance level,
- the worst-case gain, modulus, phase and delay margins.

It can be applied to high-order systems with numerous uncertainties.

It has been validated on many realistic benchmarks, and the exact value of $\mu$ is obtained in almost all cases with a low computational time.

The **full version** is available on the SMAC website: [http://w3.onera.fr/smac/smart](http://w3.onera.fr/smac/smart)
References

\(\mu\) lower bound


\(\mu\) upper bound


References

Worst-case $H_\infty$ performance


Worst-case delay margin


SMART library
