Interval-based Robustness of Linear Parametrized Filters

A. Chapoutot, T. Hilaire, P. Chevrel

September 25th, 2012
Context – 1

Filters/Controllers Algorithms

Finite precision implementation (fixed-point arithmetic)

Linear Time Invariant systems

hardware (FPGA, ASIC) or software (DSP, µC)

Evaluate the robustness of the implemented filter

Propose a methodology for the implementation of embedded controllers with finite precision considerations
Context – 1

Filters/Controllers
Algorithms

Targets

Finite precision implementation (fixed-point arithmetic)

Evaluate the robustness of the implemented filter

Propose a methodology for the implementation of embedded controllers with finite precision considerations
Context – 1

Filters/Controllers
Algorithms

Implementation

Targets

Finite precision implementation (fixed-point arithmetic)

Linear Time Invariant systems

hardware (FPGA, ASIC) or software (DSP, μC)

Evaluate the robustness of the implemented filter

Propose a methodology for the implementation of embedded controllers with finite precision considerations
Finite precision implementation (fixed-point arithmetic)
Linear Time Invariant systems
hardware (FPGA, ASIC) or software (DSP, μC)
Finite precision implementation (fixed-point arithmetic)
- Linear Time Invariant systems
- hardware (FPGA, ASIC) or software (DSP, µC)

Evaluate the robustness of the implemented filter

Propose a methodology for the implementation of embedded controllers with finite precision considerations
We are considering **parametrized** linear filters/controllers, *i.e.* filters where the coefficients depend on extra parameters $\theta$.

- These parameters are fixed, but **unknown** at implementation and compile-time;
- We only know intervals $[\theta]$ they belong to;
- The coefficients are computed once *in-situ* at initialization-time;
We are considering parametrized linear filters/controllers, i.e. filters where the coefficients depend on extra parameters $\theta$.

- These parameters are fixed, but unknown at implementation and compile-time;
- We only know intervals $[\theta]$ they belong to;
- The coefficients are computed once in-situ at initialization-time;

this is widely used by car manufacturers in order to calibrate controllers much more later in the development lifecycle.
Parametrized filters

Initialization and execution
Outline

Running example

Quantification Error Formalization

Finding Maximal Quantification Error
Running example
Second order linear filter

We consider a continuous-time second order Butterworth filter. Its transfer function is:

\[ H(s) = \frac{g}{s^2 + 2\xi\omega_c s + \omega_c^2} . \]

defined from 3 parameters:

- \( g \) the static gain;
- \( \xi \) the quality factor;
- \( \omega_c \) the cutoff pulsation.

Object of the study: a discrete version of this filter.
The equivalent discrete-time filter is

\[ H(z) = \frac{b_0 z^2 + b_1 z + b_2}{a_0 z^2 + a_1 z + a_2} \]

with
- \( b_0 = gT^2 \), \( b_1 = 2gT^2 \), \( b_2 = gT^2 \)
- \( a_0 = 4\xi \omega_c T + \omega_c^2 T^2 + 4 \), \( a_1 = 2\omega_c^2 T^2 - 8 \), \( a_2 = \omega_c^2 T^2 - 4\xi \omega_c T + 4 \)
Discrete-time algorithm – 1

The equivalent discrete-time filter is

\[ H(z) = \frac{b_0 z^2 + b_1 z + b_2}{a_0 z^2 + a_1 z + a_2} \]

with

- \( b_0 = g T^2 \), \( b_1 = 2 g T^2 \), \( b_2 = g T^2 \)
- \( a_0 = 4 \xi \omega_c T + \omega_c^2 T^2 + 4 \), \( a_1 = 2 \omega_c^2 T^2 - 8 \), \( a_2 = \omega_c^2 T^2 - 4 \xi \omega_c T + 4 \)

To implement it, one can use Direct Form

\[
\begin{align*}
\begin{cases}
x(k+1) & = \begin{pmatrix} -\frac{a_1}{a_0} & -\frac{a_2}{a_0} \\ 1 & 0 \end{pmatrix} x(k) + \begin{pmatrix} 1 \\ 0 \end{pmatrix} u(k) \\
y(k) & = \begin{pmatrix} \frac{b_1 a_0-b_0 a_1}{a_0^2} & \frac{b_2 a_0-b_0 a_2}{a_0^2} \end{pmatrix} x(k) + \frac{b_0}{a_0} u(k)
\end{cases}
\end{align*}
\]
But it is also possible to use various other algorithms

- Or any other state-space form
- $\rho$ Direct Form II transposed
- cascade decomposition, parallel, lattice, LGC or LCW forms, etc.
Discrete-time algorithm – 2

But it is also possible to use various other algorithms

▶ Or any other state-space form
▶ $\rho$ Direct Form II transposed
▶ cascade decomposition, parallel, lattice, LGC or LCW forms, etc.

Remark
All these implementations are only equivalent in infinite precision arithmetic

Our goal
We seek the implementation which is the closest in finite precision than the infinite precision implementation.
Setting simulation parameters

List of parameter values and uncertainties:

- $\pi$
- $f_c$ (cutoff frequency): 10.0 ± 20%
- $f_e$ (sampling frequency): 200.0 ± 1%
- $\xi$ (quality factor): 0.5 ± 10%

Questions:

- What will be the impact of the quantization of these parameters?
- What set of parameters will give us the worst degradation?
Setting simulation parameters

List of parameter values and uncertainties:

- $\pi$
- $f_c$ (cutoff frequency): $10.0 \pm 20\%$
- $f_e$ (sampling frequency): $200.0 \pm 1\%$
- $\xi$ (quality factor): $0.5 \pm 10\%$

Questions:

- What will be the impact of the quantization of these parameters?
- What set of parameters will give us the worst degradation?
Quantification Error
Formalization
Formulation of the problem – 1

Notations:

- \( Z(\theta) \) the matrix containing all the coefficients used by the realization
- \( h_{Z(\theta)} \) the associated transfer function
- \( \theta^\dagger \) the quantized version of \( \theta \)
- \( Z^\dagger(\theta^\dagger) \) is then the set of the quantized coefficients, i.e. the quantization of coefficients \( Z(\theta^\dagger) \) computed from the quantized parameters \( \theta^\dagger \)
- The corresponding transfer function is denoted \( h_{Z^\dagger(\theta^\dagger)} \).
Formulation of the problem – 2

For a given $\theta$, the measure of the degradation of the finite precision implementation is given by:

$$\| h_{Z}(\theta) - h_{Z}^{\dagger}(\theta^{\dagger}) \|_{\diamond}, \quad \text{with} \quad \diamond \in \{2, \infty\}$$

such that, for $g : \mathbb{C} \rightarrow \mathbb{C}$, we have:

- **2-Norm:**
  $$\| g \|_{2} \triangleq \sqrt{\frac{1}{2\pi} \int_{0}^{2\pi} |g(e^{j\omega})|^{2} \, d\omega}$$

- **Max Norm:**
  $$\| g \|_{\infty} \triangleq \max_{\omega \in [0,2\pi]} |g(e^{j\omega})|$$

**Problem**

We look for the worst-case parameters $\theta_{0}$ such that:

$$\arg \max_{\theta \in \Theta} \| h_{Z}(\theta) - h_{Z}^{\dagger}(\theta^{\dagger}) \|_{\diamond}$$
Finding Maximal Quantification Error
Interval global optimization approach

New formulation of the problem

Maximize \( \| [h]_{Z^\dagger(\theta^\dagger)} - [h]_{Z(\theta)} \| \) subject to \( \theta \in [\theta] \).

such that \([h]\) is a transfer function with interval coefficients.

To apply for example Hansen’s algorithm, we need:

▶ a sharp inclusion function for \( \| [h]_{Z^\dagger(\theta^\dagger)} - [h]_{Z(\theta)} \| \)
Inclusion functions – 1

Recall, for $g : \mathbb{C} \to \mathbb{C}$, we have:

- 2 Norm: $\| g \|_2 \triangleq \sqrt{\frac{1}{2\pi}} \int_0^{2\pi} |g(e^{j\omega})|^2 \, d\omega$
- Max Norm: $\| g \|_\infty \triangleq \max_{\omega \in [0,2\pi]} |g(e^{j\omega})|$

In both cases, the first step is to compute $|g(e^{j\omega})|$:  
- either by using complex interval arithmetic;
- or real interval arithmetic after proper symbolic manipulations.
Inclusion functions – 2

We tried different approaches:

- Direct evaluation of \([g](e^{j\omega})\) with Cartesian complex interval form;
- Rewriting \(|[g](e^{j\omega})|\) with symmetric and antisymmetric decomposition (cos and sin), and evaluation with real intervals; (development with sin and cos doesn’t help)
- Direct evaluation of \([g](e^{j\omega})\) with polar complex interval form; addition polar complex form very difficult – to be explored more deeply, see J. Flores Complex Fans.

A+B
(Real)
(Computed)
Inclusion functions – 2

We tried different approaches:

- Direct evaluation of \([g](e^{j\omega})\) with Cartesian complex interval form; 😞 (too loosy)

- Rewriting \(|g|(e^{j\omega})\) with symmetric and antisymmetric decomposition (cos and sin), and evaluation with real intervals; (development with sin and cos doesn't help)

- Direct evaluation of \([g](e^{j\omega})\) with polar complex interval form; addition polar complex form very difficult – to be explored more deeply, see J. Flores Complex Fans
Inclusion functions – 2

We tried different approaches:

- Direct evaluation of $[g](e^{j\omega})$ with Cartesian complex interval form; 😞 (too loosy)
- Rewriting $|[g](e^{j\omega})|$ with symmetric and antisymmetric decomposition (cos and sin), and evaluation with real intervals;
Inclusion functions – 2

We tried different approaches:

- Direct evaluation of \([g](e^{j\omega})\) with Cartesian complex interval form; 😞 (too loose)
- Rewriting \(|[g](e^{j\omega})|\) with symmetric and antisymmetric decomposition (cos and sin), and evaluation with real intervals; 😞 (development with sin and cos doesn’t help)
- Direct evaluation of \([g](e^{j\omega})\) with polar complex interval form;

\[A + B\]

\[
\begin{array}{c}
A \\
B
\end{array}
\]

\[
\begin{array}{c}
\text{(Real)} \\
\text{(Computed)}
\end{array}
\]
Inclusion functions – 2

We tried different approaches:

- Direct evaluation of \([g](e^{j\omega})\) with Cartesian complex interval form; ☹ (too loose)
- Rewriting \(|[g](e^{j\omega})|\) with symmetric and antisymmetric decomposition (cos and sin), and evaluation with real intervals; ☹ (development with sin and cos doesn’t help)
- Direct evaluation of \([g](e^{j\omega})\) with polar complex interval form; ☹ addition polar complex form very difficult – to be explored more deeply, see J. Flores *Complex Fans*
Inclusion functions – 3

But also:

- Symbolic computation of $[g](e^{j\omega}) \cdot [g](e^{j\omega})^*$ and evaluation with real interval arithmetic;

- For 2-norm, we can also use Lyapunov equation:

\[
g(z) \text{ is put in form } g(z) = c(zI - A)^{-1}b + d\]

\[
\|g\|_2 = \sqrt{\text{tr}(cWc^\top + d^2)} \quad \text{with} \quad W = AW A^\top + bb^\top
\]

Here $A$ and $W$ are interval matrices.

Software Versoft (Rohn), based on Intlab (Rump) can deal with Lyapunov equation to solve. But results too loose, due to dependency between coefficients.
Inclusion functions – 3

But also:

- Symbolic computation of $[g](e^{j\omega}) \cdot [g](e^{j\omega})^*$ and evaluation with real interval arithmetic;
- Generation of very long formula which increases the problem of dependency (loosey results), more work could be done to simplify this expression (e.g. factorizing the common sub-expressions)
But also:

- Symbolic computation of \([g](e^{j\omega}) \cdot [g](e^{j\omega})^*\) and evaluation with real interval arithmetic; 😊 Generation of very long formula which increases the problem of dependency (loosy results), more work could be done to simplify this expression (e.g. factorizing the common sub-expressions)

- For 2-norm, we can also use Lyapunov equation:

  \(g(z)\) is put in form \(g(z) = c(zI - A)^{-1}b + d\) and

  \[
  \|g\|_2 = \sqrt{\text{tr}(cWc^T + d^2)} \quad \text{with} \quad W = AWA^T + bb^T
  \]
Inclusion functions – 3

But also:

- Symbolic computation of \([g](e^{j\omega}) \cdot [g](e^{j\omega})^*\) and evaluation with real interval arithmetic; ☺️ Generation of very long formula which increases the problem of dependency (loosy results), more work could be done to simplify this expression (e.g. factorizing the common sub-expressions)

- For 2-norm, we can also use Lyapunov equation: \(g(z)\) is put in form \(g(z) = c(zI - A)^{-1}b + d\) and

\[
\|g\|_2 = \sqrt{\text{tr}(cWc^T + d^2)} \quad \text{with} \quad W = AWA^T + bb^T
\]

Here \(A\) and \(W\) are interval matrices.
Software *Versoft* (Rohn), based on *Intlab* (Rump) can deal with Lyapunov equation to solve. 😞 But results too lossy, due to dependency between coefs in \(A\) and \(b\)
Global solver

Once the problem of the inclusion function solved, we will apply Hansen’s algorithm\(^1\) whose main steps are:

**Input:** \([f]\) inclusion function, \(X\) initial box, \(\epsilon\) tolerance  
**Output:** \(Y\) the sub-box associated to the minimal value of \(f\).

Set \(Y := X\)  
Compute \([f](Y), \tilde{f} := \text{ub}([f](\text{mid}(Y))), y := \text{lb}([f](Y))\)  
Initialize list \(L := \{(Y, y)\}\)  
(*) Choose a coordinate direction \(k\) parallel to which \(Y\) has an edge of maxim length  
Bisect \(Y\) following \(k\) to get \(V_1, V_2\) such that \(Y = V_1 \cup V_2\)  
Remove \((Y, y)\) from \(L\)  
Compute \([f](V_1), [f](V_2)\) and \(v_i = \text{lb}([f](V_i))\) for \(i = 1, 2\)  
Enter pairs \((V_1, v_1)\) and \((V_2, v_2)\) at the end of \(L\)  
Choose a pair \((\tilde{Y}, \tilde{y})\) in \(L\) such that \(\tilde{y} \leq z\) for all \((Z, z)\)  
Remove all \((Z, z)\) such that \(\tilde{f} \leq z\)  
If width(\(Y\)) \(\leq \epsilon\) end algorithm  
Denote the first pair of \(L\) as \((Y, y), \tilde{f} = \min(\tilde{f}, \text{ub}([f](\text{mid}(Y))))\) go to (*).  

\(^{1}\)“New computer method for global optimization” H. Ratschek and J. Rokne
Current state of our work
We did not find a satisfactory solution for the inclusion function! In all cases, evaluation usually produces large and useless intervals for $\| [h]^{\dagger}_{Z^{\dagger}(\theta^{\dagger})} - [h]Z(\theta) \|_{\diamond}$, event with small width interval parameter values.

Moreover, a Monte-Carlo-like evaluation of $\| [h]^{\dagger}_{Z^{\dagger}(\theta^{\dagger})} - [h]Z(\theta) \|_{\diamond}$ with desired interval parameters showed us the inclusion function should be more accurate.
Conclusion:

- the inclusion function is the most difficult problem to solve to apply interval global optimization method

Perspective:

- use of affine arithmetic or Taylor arithmetic to define inclusion function to avoid dependence problem.
- develop polar form interval arithmetic package
- or dedicated interval transfer-function evaluation techniques