Interval Methods for Reliable Modeling, Identification and Control of Dynamic Systems

Pre-Conference Workshop at ECC2015
14th European Control Conference

Linz, Austria, July 14th, 2015

Andreas Rauh*, Luise Senkel*, Ekaterina Auer**

*Chair of Mechatronics, University of Rostock, Germany
**Faculty of Engineering, University of Applied Sciences Wismar, Germany
Contents

Part 1
- Fundamentals of Interval Arithmetics: Concept and Software Demonstration
- Verified Simulation of Dynamic Systems
- Kinds of Uncertainty and their Treatment during Modeling and Simulation
- Possibilities for Simulating Uncertain Non-Smooth Dynamic Systems

Part 2
- Control-Oriented Applications: Identification and Optimization
- Interval-Based Sliding Mode Approaches: Control and Estimation
- Application to Fuel Cell Systems for Identification and Control
Part 1

1. Fundamentals of Interval Arithmetic

Presentation of the Fundamental Mathematical Concept of Interval Arithmetic for Set-Valued Computations
Presentation of the Fundamental Mathematical Concept of Interval Arithmetic for Set-Valued Computations

Contents

- Motivation
- Definition of Real Intervals, Interval Vectors, Interval Matrices
- Definition of Complex Intervals
- Calculating with Real Intervals
- Static and Dynamic System Descriptions
- Overestimation: Dependency Problem and Wrapping Effect
  - Monotonicity for Numerical Integration of Systems with Intervals
  - Taylor Expansion and Mid-point Rule
  - Affine System Representation
Motivation: Uncertainty

- Manufacturing Tolerances
- Measurement Noise
- Disturbances
- Input Range and Rate Constraints
- Non-Measurable States
- Non-representable effects
- Simplifications (continuous, time-discrete)
- Reliable Simulation
- Identification
- Optimization
- Real-Time Control
- State Observation

Model Accuracy

Real System

Static or Dynamic Mathematical Model

Implementation Effort

Computation Time

Numerical Problems (Rounding and Discretization Errors,...)
Motivation: Uncertainty

Model Accuracy

Computation Time

Implementation Effort

Manufacturing Tolerances
Measurement Noise
Disturbances
Input Range and Rate Constraints
Non-Measurable States
Non-representable effects
Simplifications
Reliable Simulation
Identification
Optimization
Real-Time Control
State Observation

Real System

\[ \dot{x}(t) = f(x(t), p(t), u(t)) \]

\[ u = u(t) \quad \text{open-loop dynamic system} \]

\[ u = u(x(t)) \quad \text{closed-loop dynamic system} \]
Motivation: Uncertainty

Manufacturing Tolerances  Measurement Noise  Disturbances  Input Range and Rate Constraints  Non-Measurable States

Model Accuracy  Simplifications  Non-representable effects

Real System

\[ x(t_{k+1}) = f(x(t_k), p(t_k), u(t_k)) \]

\[ u_k = u(t_k) \quad \text{open-loop dynamic system} \]

\[ u_k = u(x(t_k)) \quad \text{closed-loop dynamic system} \]

Reliable Simulation
Identification
Optimization
Real-Time Control
State Observation

Implementation Effort  Computation Time  Numerical Problems (Rounding and Discretization Errors,...)
Motivation: Uncertainty

- Manufacturing Tolerances
- Measurement Noise
- Disturbances
- Input Range and Rate Constraints
- Non-Measurable States
- Non-representable effects
- Simplifications

Real System

\[ 0 = f(x(t_k), u(t_k)) \]
Static case: algebraic (nonlinear) system of equations

- Model Accuracy
- Implementation Effort
- Computation Time
- Numerical Problems (Rounding and Discretization Errors, ...)

Reliable Simulation
Identification
Optimization
Real-time Control
State Observation
Motivation: Uncertainty

Intervals
- Manufacturing Tolerances
- Measurement Noise *
- Disturbances *
- Input Range and Rate Constraints
- Non-Measurable States

Real System

Model Accuracy

Static or Dynamic Mathematical Model

Intervals

Implemention Effort

Computation Time

Numerical Problems (Rounding and Discretization Errors,...)

* - more details about stochastic approaches in Part 2
Definition of Real Intervals, Interval Vectors, Interval Matrices

Scalar Real Interval

\[ [a] = [a; \bar{a}] = [\inf([a]); \sup([a))], \ a \leq \bar{a}, \ \{x \in \mathbb{R} | a \leq x \leq \bar{a}\} \]

Interval Vector

\[
[a] = \begin{bmatrix}
[a_1; \bar{a}_1] \\
[a_2; \bar{a}_2] \\
\vdots \\
[a_n; \bar{a}_n]
\end{bmatrix}
\]

Interval Matrix

\[
[A] = \begin{bmatrix}
[a_{11}; \bar{a}_{11}] & [a_{12}; \bar{a}_{12}] & \cdots & [a_{1n}; \bar{a}_{1n}] \\
[a_{21}; \bar{a}_{21}] & [a_{22}; \bar{a}_{22}] & \cdots & [a_{2n}; \bar{a}_{2n}] \\
\vdots & \vdots & \ddots & \vdots \\
[a_{n1}; \bar{a}_{n1}] & [a_{n2}; \bar{a}_{n2}] & \cdots & [a_{nn}; \bar{a}_{nn}]
\end{bmatrix}
\]
Definition of Complex Intervals

\[ [a] = [a_1; \bar{a}_1] + j[a_2; \bar{a}_2] \]

⇒ Useful for dynamic systems with oscillatory behavior
Calculating with Real Intervals - Natural Interval Evaluation

**Addition**

\[
[p; \overline{p}] + [q; \overline{q}] = [p + q; \overline{p} + \overline{q}]
\]

\[
[1; 2] + [−2; 2] = [1 + (−2); 2 + 2] = [−1; 4]
\]

\[
\begin{bmatrix}
[-2; -1] \\
[0; 4]
\end{bmatrix} + \begin{bmatrix}
[-10; -3] \\
[5; 8]
\end{bmatrix} = \begin{bmatrix}
[-12; -4] \\
[5; 12]
\end{bmatrix}
\]

\[
\begin{bmatrix}
[2; 3] & [-4; -3] \\
[7; 9] & [10; 15]
\end{bmatrix} + \begin{bmatrix}
[12; 13] & [-14; -13] \\
[17; 19] & [20; 25]
\end{bmatrix} = \begin{bmatrix}
[14; 16] & [-18; -16] \\
[24; 28] & [30; 40]
\end{bmatrix}
\]
Calculating with Real Intervals - Natural Interval Evaluation

Subtraction

\[ [p; \overline{p}] - [q; \overline{q}] = [p - \overline{q}; \overline{p} - \overline{q}] \]

\[ [1; 2] - [2; 3] = [1 - 3; 2 - 2] = [-2; 0] \]

\[
\begin{bmatrix}
[2; 3] & [-4; -3] \\
[7; 9] & [10; 15]
\end{bmatrix} - \begin{bmatrix}
[12; 13] & [-14; -13] \\
[17; 19] & [20; 25]
\end{bmatrix} = \begin{bmatrix}
[-11; -9] & [9; 11] \\
[-12; -8] & [-15; -5]
\end{bmatrix}
\]
Calculating with Real Intervals - Natural Interval Evaluation

**Multiplication**

\[
[p; \overline{p}] \cdot [q; \overline{q}] = \left[ \min \{ pq, \overline{p} \overline{q}, \overline{p} q, \overline{p} \overline{q} \} ; \max \{ pq, \overline{p} \overline{q}, \overline{p} q, \overline{p} \overline{q} \} \right]
\]

\[
[1; 2] \cdot [2; 3] = \left[ \min \{1 \cdot 2, 1 \cdot 3, 2 \cdot 2, 2 \cdot 3 \} ; \max \{1 \cdot 2, 1 \cdot 3, 2 \cdot 2, 2 \cdot 3 \} \right] = [2; 6]
\]
Calculating with Real Intervals - Natural Interval Evaluation

**Division**

\[
\frac{[p]}{[q]} = [p] \cdot \left[ \frac{1}{q}; \frac{1}{q} \right] \quad \text{if} \quad 0 \not\in [q]
\]

\[
\frac{[1; 2]}{[2; 3]} = [1; 2] \cdot \left[ \frac{1}{3}; \frac{1}{2} \right] = \left[ \frac{1}{3}; 1 \right]
\]
## Calculating with Real Intervals - Natural Interval Evaluation

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Radius of a Real Interval</strong></td>
<td>( r([a]) = \frac{1}{2}(\bar{a} - a) )</td>
</tr>
<tr>
<td><strong>Width of an Interval</strong></td>
<td>( w([a]) = \bar{a} - a = 2 \cdot r([a]) )</td>
</tr>
<tr>
<td><strong>Mid-point of an Interval</strong></td>
<td>( m([a]) = \frac{1}{2}(a + \bar{a}) )</td>
</tr>
</tbody>
</table>

⇒ For real interval vectors and matrices, these characteristics hold component-wise
Continuous- and Discrete-Time Systems — Dynamic Case

Continuous-Time System \( \dot{x}(t) = f(x(t), p, u(t)) \)

- \( x(t) \) State Vector
- \( p \) Vector of Uncertain Parameters: \( p_i \in [\underline{p}_i ; \overline{p}_i] \), \( i = 1, ..., n_p \)
- \( u(t) \) Input Vector: \( u_j \in [\underline{u}_j ; \overline{u}_j] \), \( j = 1, ..., n_u \)
Continuous- and Discrete-Time Systems — Dynamic Case

**Continuous-Time System** \( \dot{x}(t) = f(x(t), p, u(t)) \)

- \( x(t) \) State Vector
- \( p \) Vector of Uncertain Parameters: \( p_i \in [\underline{p}_i; \bar{p}_i], i = 1, ..., n_p \)
- \( u(t) \) Input Vector: \( u_j(t) \in [\underline{u}_j; \bar{u}_j], j = 1, ..., n_u \)

**Discrete-Time System** \( x(t_{k+1}) = f(x(t_k), p(t_k), u(t_k)) \)

- \( x(t_k) \) State Vector
- \( p \) Vector of Uncertain Parameters: \( p_i \in [\underline{p}_i; \bar{p}_i] \)

\( \implies \) **Range Bounds / Tolerances**

- \( u(t_k) \) Input Vector: \( u_j(t_k) \in [\underline{u}_j(t_k); \bar{u}_j(t_k)] \)

\( \implies \) **Input Range Constraints**

\( \Rightarrow \) Calculate all reachable states
Continuous- and Discrete-Time Systems — Static Case

Continuous-Time System \( \dot{x}(t) = 0 = f(x(t), p, u(t)) \)

- \( x(t) \) State Vector
- \( p \) Vector of Uncertain Parameters: \( p_i \in [\underline{p}_i; \overline{p}_i], i = 1, \ldots, n_p \)
- \( u(t) \) Input Vector: \( u_j \in [\underline{u}_j; \overline{u}_j], j = 1, \ldots, n_u \)
## Continuous- and Discrete-Time Systems — Static Case

**Continuous-Time System** \( \dot{x}(t) = 0 = f(x(t), p, u(t)) \)

- **\( x(t) \)**: State Vector
- **\( p \)**: Vector of Uncertain Parameters: \( p_i \in [\underline{p}_i; \bar{p}_i], i = 1, \ldots, n_p \)
- **\( u(t) \)**: Input Vector: \( u_j \in [\underline{u}_j; \bar{u}_j], j = 1, \ldots, n_u \)

**Discrete-Time System** \( x(t_{k+1}) = x(t_k) = f(x(t_k), p(t_k), u(t_k)) \)

- **\( x(t_k) \)**: State Vector
- **\( p \)**: Vector of Uncertain Parameters: \( p_i \in [\underline{p}_i; \bar{p}_i] \)  
  \( \Rightarrow \) **Range Bounds / Tolerances**
- **\( u(t_k) \)**: Input Vector: \( u_j(t_k) \in [\underline{u}_j(t_k); \bar{u}_j(t_k)] \)  
  \( \Rightarrow \) **Input Range Constraints**

\( \Rightarrow \) Solve for state vector \( x(t) \) or \( x_k \) resp. for 1 time step
Overestimation: Dependency Problem

Problem: Multiple occurrence of an interval in one equation

Necessary: Factorizations, simplifications, reformulations as far as possible
⇒ Reduction of overestimation and computation time

Example

\[ f([x]) = 2 \cdot [x] - [x] \cdot [x] \text{ and } [x] = [-1 ; 2], \text{ Results provided by Intlab} \]

1. \[ f([x]) = 2 \cdot [x] - [x] \cdot [x] = [-6 ; 6] \]
2. \[ f([x]) = 2 \cdot [x] - [x]^2 = [-6 ; 4] \]
3. \[ f([x]) = -([x] - 1) \cdot ([x] - 1) + 1 = [-3 ; 3] \]
4. \[ f([x]) = -([x] - 1)^2 + 1 = [-3 ; 1] \text{ (Exact evaluation)} \]
Overestimation: Dependency Problem

Problem: Multiple occurrence of an interval in one equation

Necessary: Factorizations, simplifications, reformulations as far as possible
⇒ Reduction of overestimation and computation time

Example

\[ f([x]) = 2 \cdot [x] - [x] \cdot [x] \] and \([x] = [-1 ; 2]\), Results provided by Intlab

Higher-order Interval Evaluation for Polynomials: Taylor Expansion

\((x_m = \text{mid}([x]))\) according to

\[
T(f) = f(x_m) + \left( \sum_{i=1}^{n-1} \frac{\partial^i f(x)}{\partial x^i} \bigg|_{x_m} \cdot \frac{([x] - x_m)^i}{i!} \right) + \frac{\partial^n f(x)}{\partial x^n} \bigg|_{[x]} \cdot \frac{([x] - x_m)^n}{n!}
\]
Overestimation: Dependency Problem

Problem: Multiple occurrence of an interval in one equation

Necessary: Factorizations, simplifications, reformulations as far as possible
⇒ Reduction of overestimation and computation time

Example

\[ f([x]) = 2 \cdot [x] - [x] \cdot [x] \text{ and } [x] = [-1 ; 2], \text{ Results provided by Intlab} \]

5 Taylor Expansion with \( x_m = \text{mid}([x]) = 0.5 \)

\[
[f](x) = f(x_m) + \frac{\partial f}{\partial x} \bigg|_{x_m} \cdot (x - x_m) + \frac{\partial^2 f}{\partial x^2} \bigg|_{x_m} \cdot \frac{(x - x_m)^2}{2!} \\
f(x_m) = 2 \cdot 0.5 - 0.5^2 = 0.75 \]

\[
\frac{\partial f}{\partial x} \bigg|_{x_m} \cdot (x - x_m) = (2 - 2 \cdot x) \bigg|_{x_m} \cdot (x - x_m) = [-1.5 ; 1.5] 
\]
Overestimation: Dependency Problem

Example

\[ f([x]) = 2 \cdot [x] - [x] \cdot [x] \text{ and } [x] = [-1 ; 2], \text{ Results provided by Intlab} \]

5 Taylor Expansion with \( x_m = \text{mid}([x]) = 0.5 \)

\[ f([x]) = 0.75 + [-1.5 ; 1.5] + \left. \frac{\partial^2 f}{\partial x^2} \right|_{[x]} \cdot \frac{([x]-x_m)^2}{2!} = \]

\[ \left. \frac{\partial^2 f}{\partial x^2} \right|_{[x]} \cdot \frac{([x]-x_m)\cdot([x]-x_m)}{2!} = -2 \cdot \frac{([x]-x_m)\cdot([x]-x_m)}{2!} = \]

\[ -1 \cdot [-1.5 ; 1.5] \cdot [-1.5 ; 1.5] = [-2.25 ; 2.25] \]

⇒ Taylor expansion will be demonstrated later with two software libraries
Overestimation: Dependency Problem

Example

\[ f([x]) = 2 \cdot [x] - [x] \cdot [x] \text{ and } [x] = [-1 ; 2], \text{ Results provided by Intlab} \]

Taylor Expansion with \( x_m = \text{mid}([x]) = 0.5 \)

\[ f([x]) = 0.75 + [-1.5 ; 1.5] + \left. \frac{\partial^2 f}{\partial x^2} \right|_{[x]} \cdot \frac{([x]-x_m)^2}{2!} = \]

\[ \left. \frac{\partial^2 f}{\partial x^2} \right|_{[x]} \cdot \frac{([x]-x_m) \cdot ([x]-x_m)}{2!} = -2 \cdot \frac{([x]-x_m) \cdot ([x]-x_m)}{2!} = \]

\[ -1 \cdot [-1.5 ; 1.5] \cdot [-1.5 ; 1.5] = [-2.25 ; 2.25] \]

\[ [x]^2 = \begin{cases} \min(a\bar{a}, a\bar{a}) ; \max(a\bar{a}, a\bar{a}) & \text{if } 0 \notin [x] \\ [0 ; \max(a\bar{a}, a\bar{a})] & \text{if } 0 \in [x] \end{cases} \]

\[ \left. \frac{\partial^2 f}{\partial x^2} \right|_{[x]} \cdot \frac{([x]-x_m)^2}{2!} = -2 \cdot \frac{([x]-x_m)^2}{2!} = -1 \cdot [-1.5 ; 1.5]^2 = [-2.25 ; 0] \]

\( \Rightarrow \) Taylor expansion will be demonstrated later with two software libraries
Overestimation: Dependency Problem

Example

\[ f([x]) = 2 \cdot [x] - [x] \cdot [x] \] and \( [x] = [-1 ; 2] \), Results provided by Intlab

Taylor Expansion with \( x_m = \text{mid}([x]) = 0.5 \)

\[ f([x]) = f(x_m) + \frac{\partial f}{\partial x} \bigg|_{x_m} \cdot ([x] - x_m) + \frac{\partial^2 f}{\partial x^2} \bigg|_{[x]} \cdot \frac{([x] - x_m)^2}{2!} = \]

\[ f(x_m) = 2 \cdot 0.5 - 0.5^2 = 0.75 \]

\[ \frac{\partial f}{\partial x} \bigg|_{x_m} \cdot ([x] - x_m) = (2 - 2 \cdot x) \bigg|_{x_m} \cdot ([x] - x_m) = [-1.5 ; 1.5] \]

\[ \frac{\partial^2 f}{\partial x^2} \bigg|_{[x]} \cdot \frac{([x] - x_m)^2}{2!} = -2 \cdot \frac{([x] - x_m)^2}{2!} = -1 \cdot [-1.5 ; 1.5]^2 = [-2.25 ; 0] \]

\[ \Rightarrow f([x]) = 0.75 + [-1.5 ; 1.5] + [-2.25 ; 0] = [-3 ; 2.25] \]

⇒ Taylor expansion will be demonstrated later with two software libraries
Overestimation: Dependency Problem

\[ f(x) = - (x-1)^2 + 1 \]
Overestimation: Dependency Problem

\[ f(x) = 2x - x^2 \]

\[ f(x) = -(x-1)^2 + 1 \]
Overestimation: Dependency Problem

Motivation

Arithmetics

System Formulations

Problem: Overestimation and How to Reduce

\[ f(x) = 2x - x^2 \]

\[ f(x) = -(x-1)^2 + 1 \]
Motivation

Arithmetics

System Formulations

Problem: Overestimation and How to Reduce

Overestimation: Dependency Problem

\[
[f](x) = 2x - x^2
\]

\[
[f](x) = -([x] - 1) ([x] - 1) + 1
\]
Overestimation: Dependency Problem

\[ f(x) = 2x - x^2 \]

\[ f'(x) = 2 - 2x \]

\[ f''(x) = -2 \]

**Taylor Expansion**

\[ f(x) = -(x-1)(x-1) + 1 \]
Special Case of Taylor Expansion: Mid-point Rule

\[ f(x) \subseteq f_m([x]) = f(x_m) + \frac{\partial f}{\partial x} \bigg|_{[x]} ([x] - x_m) \]
Special Case of Taylor Expansion: Mid-point Rule

\[ f(x) \subseteq f_m([x]) = f(x_m) + \left. \frac{\partial f}{\partial x} \right|_{[x]} ([x] - x_m) \]
Special Case of Taylor Expansion: Mid-point Rule

$$f(x) \subseteq f_m([x]) = f(x_m) + \frac{\partial f}{\partial x} \bigg|_{[x]} ([x] - x_m)$$
**Special Case of Taylor Expansion: Mid-point Rule**

\[
  f(x) \subseteq f_m([x]) = f(x_m) + \frac{\partial f}{\partial x} \bigg|_{[x]} ([x] - x_m)
\]

- **Exact enclosure** \([f_{ex}([x])]\)
- **Natural evaluated enclosure** \([f_{nat}([x])]\)
- **Enclosure evaluated by mid-point rule** \([f_{mp}([x])]\)
- **Constructed range of function value** using mid-point rule and tangent on \(f(x_m)\)
- **Approximation of the solution of the function** \(f(x)\)
Special Case of Taylor Expansion: Mid-point Rule

\[ f(x) \subseteq f_m([x]) = f(x_m) + \frac{\partial f}{\partial x} \bigg|_{[x]} ([x] - x_m) \]

Approximation of the solution of the function \( f(x) \) by using the smallest and largest slope depicted by the triangles.
Monotonicity

Consider: Interval-Valued Function given by $F = x + x \cdot x$

- Two intervals $[x_1] = [-2; 4]$ and $[x_2] = [-1; 4]$ with $[x_1] \subset [x_2]$
- $F([x_1]) = [-2; 4] + [-2; 4] \cdot [-2; 4] = [-2; 4] + [-8; 16] = [-10; 20]$
- $F([x_2]) = [-1; 4] + [-1; 4] \cdot [-1; 4] = [-1; 4] + [-4; 16] = [-5; 20]$
- Consequence $F([x_2]) \subset F([x_1]) \Rightarrow F$ is an inclusion monotonic function
- 4 basic arithmetic operators are also inclusion monotonic

Consequence for Calculating with Intervals

- Splitting of large intervals
- Hull of all evaluations with the subintervals
- Tighter range bounds than with original interval
Monotonicity

Consider: Interval-Valued Function given by $F = x + x \cdot x$

- Two intervals $[x_1] = [-2; 4]$ and $[x_2] = [-1; 4]$ with $[x_1] \subset [x_2]$
- $F([x_1]) = [-2; 4] + [-2; 4] \cdot [-2; 4] = [-2; 4] + [-8; 16] = [-10; 20]$
- $F([x_2]) = [-1; 4] + [-1; 4] \cdot [-1; 4] = [-1; 4] + [-4; 16] = [-5; 20]$
- Consequence $F([x_2]) \subset F([x_1]) \Rightarrow F$ is an inclusion monotonic function
- 4 basic arithmetic operators are also inclusion monotonic

Monotonicity of a Function Using Derivatives

$$\frac{\partial F}{\partial x} \bigg|_{x \in [x]} < 0 \quad \Rightarrow \quad F \in [F(x); F(\overline{x})]$$

$$\frac{\partial F}{\partial x} \bigg|_{x \in [x]} > 0 \quad \Rightarrow \quad F \in [F(\overline{x}); F(x)]$$
Overestimation: Wrapping Effect — Example

Discrete System Model

\[ [x](t_{k+1}) = A \cdot [x](t_k) \quad \text{with} \quad [x](t_0) = \begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix} \quad \text{and} \quad A = \frac{1}{2} \sqrt{2} \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \]

Aim

Evaluation of interval enclosure \([x](t_{k+1})\)

Problem in Engineering Tasks

Uncertainty in parameters, significantly larger than representation errors of floating-point values (rounding errors)
Overestimation: Wrapping Effect — Example

\[
[x](t_{k+1}) = A \cdot [x](t_k) \quad \text{with} \quad [x](t_0) = \begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix} \quad \text{and} \quad A = \frac{1}{2}\sqrt{2} \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}
\]

- Exact recursive evaluation

⇒ Rotation of 45° due to structure of system matrix A
Overestimation: Wrapping Effect — Example

\[
\mathbf{x}(t_{k+1}) = \mathbf{A} \cdot \mathbf{x}(t_k), \quad \mathbf{x}(t_0) = \begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix}, \quad \mathbf{A} = \mathbf{A}_k = \frac{1}{2} \sqrt{2} \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}
\]

- Traditional recursive interval evaluation (using e.g. Intlab)

\[
\begin{align*}
\mathbf{x}(t_1) &= \mathbf{A} \mathbf{x}(t_0) \\
\mathbf{x}(t_2) &= \mathbf{A} \mathbf{x}(t_1) \\
& \quad \cdots \\
\mathbf{x}(t_{k+1}) &= \mathbf{A} \mathbf{x}(t_k)
\end{align*}
\]

⇒ Exponential growth of the enclosing interval boxes
Overestimation: Wrapping Effect — Example

\[ [\mathbf{x}](t_{k+1}) = \mathbf{A} \cdot [\mathbf{x}](t_k), \quad [\mathbf{x}](t_0) = \begin{bmatrix} -1 ; 1 \\ -1 ; 1 \end{bmatrix}, \quad \mathbf{A} = \mathbf{A}_k = \frac{1}{2} \sqrt{2} \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \]

- Intelligent recursive evaluation (affine): Modified system matrix
  \[ \tilde{\mathbf{A}}_k = \mathbf{A} \tilde{\mathbf{A}}_{k-1} \]
  \[ [\mathbf{x}](t_1) = \mathbf{A} [\mathbf{x}](t_0) = \tilde{\mathbf{A}}_0 [\mathbf{x}](t_0) \]
  \[ [\mathbf{x}](t_2) = \mathbf{A} \tilde{\mathbf{A}}_0 [\mathbf{x}](t_0) = \tilde{\mathbf{A}}_1 [\mathbf{x}](t_0) \]
  \[ \vdots \]
  \[ [\mathbf{x}](t_{k+1}) = \mathbf{A} \tilde{\mathbf{A}}_{k-1} [\mathbf{x}](t_0) = \tilde{\mathbf{A}}_k [\mathbf{x}](t_0) \]

⇒ Significant reduction of the wrapping effect for linear systems
Affine System Representation for Discrete Systems

Advantages
- Directly mapping of interval variables to their initial intervals in each time step
- No dependencies between intervals $\Rightarrow$ no interval box rotations

Discretization depends on
- Additive interval or multiplicatively coupled parameter interval
- Input variable constant or changing
- Explicit or implicit Euler method
Affine System Representation for a SISO System

Case 1: Additive interval uncertainty \([a], u(t_k) \neq \text{const}, \text{step size } T = 1\)

\[
f(y(t_k), [u(t_k)]) = 2 \cdot [y](t_k) + 1 \cdot u(t_k) + 3 \cdot [a]
\]

\[
[x](t_{k+1}) = \begin{bmatrix} [y](t_{k+1}) \\ [a](t_{k+1}) \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} [y](t_k) \\ [a](t_k) \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \cdot u(t_k)
\]

E.g. Explicit Euler Discretization (time discretization error neglected)

\[
[x](t_{k+1}) = M(t_{k+1}) \cdot [x](t_0) + \gamma(t_{k+1}) \quad \text{with}
\]

\[
M(t_k) = M \quad \Rightarrow \quad M(t_{k+1}) = M \cdot M(t_k)
\]

\[
\gamma(t_{k+1}) = M(t_k) \cdot \gamma(t_k) + T \cdot \rho(t_k)
\]

initial conditions \([x](t_0) = \begin{bmatrix} [y](t_0) \\ [a](t_0) \end{bmatrix}, \quad M(t_0) = I_{2 \times 2}, \quad \gamma(t_0) = 0\)
Affine System Representation for a SISO System

Case 2: Additive interval uncertainty \([a]\), \(u=\text{const}\), step size \(T = 1\)

\[
f(y(t), [u(t)]) = 2 \cdot [y](t) + 1 \cdot u(t) + 3 \cdot [a]
\]

\[
[x](t_{k+1}) = \begin{bmatrix} [y](t_{k+1}) \\ [a](t_{k+1}) \\ u(t_{k+1}) \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} [y](t_k) \\ [a](t_k) \\ u(t_k) \end{bmatrix}
\]

Extended state vector \(x(t_{k+1})\)

Explicit Euler Discetization (time discretization error neglected)

\[
[x](t_{k+1}) = \mathbf{M}(t_{k+1}) \cdot [x](t_0) \quad \text{with}
\]

\[
\mathbf{M}(t_k) = \mathbf{M} \quad \Rightarrow \quad \mathbf{M}(t_{k+1}) = \mathbf{M} \cdot \mathbf{M}(t_k)
\]

\[
[x](t_0) = \begin{bmatrix} [y](t_0) \\ [a](t_0) \\ u(t_0) \end{bmatrix}
\]
Affine System Representation — Comparison

Implicit Euler Method \((u(t_{k+1}) = u(t_k) = \text{const})\)

\[
\dot{x}(t) = f(x(t), u(t)) = Ax(t) + bu(t)
\]

\[
\Rightarrow f(x(t_{k+1}), u(t_{k+1})) \approx \frac{x(t_{k+1}) - x(t_k)}{T}
\]

\[
x(t_{k+1}) = (I - T \cdot A)^{-1} \cdot (x(t_k) + T \cdot b \cdot u(t_{k+1}))
\]

\[
\begin{bmatrix}
\hat{x}(t_{k+1}) \\
u(t_{k+1})
\end{bmatrix} = (I - T \cdot A)^{-1} \cdot (I - T \cdot A)^{-1} \cdot T \cdot b \\
\begin{bmatrix}
\hat{x}(t_k) \\
u(t_k)
\end{bmatrix}.
\]

\[
\begin{bmatrix}
\hat{x}(t_{k+1}) \\
u(t_{k+1})
\end{bmatrix} = \tilde{A} \cdot \\
\begin{bmatrix}
\hat{x}(t_k) \\
u(t_k)
\end{bmatrix}.
\]
Affine System Representation — Comparison

Explicit Euler Method \( u(t_{k+1}) = u(t_k) = \text{const} \)

\[
\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t), u(t)) = A\mathbf{x}(t) + b u(t)
\]

\[
\Rightarrow \mathbf{f}(\mathbf{x}(t_k), u(t_k)) \approx \frac{\mathbf{x}(t_{k+1}) - \mathbf{x}(t_k)}{T}
\]

\[
\mathbf{x}(t_{k+1}) = (I + T \cdot A) \cdot \mathbf{x}(t_k) + T \cdot b \cdot u(t_k)
\]

\[
\begin{bmatrix}
\mathbf{x}(t_{k+1}) \\
u(t_{k+1})
\end{bmatrix}
= \begin{bmatrix}
(I + T \cdot A) & T \cdot b \\
0^T & 1
\end{bmatrix} \cdot \begin{bmatrix}
\mathbf{x}(t_k) \\
u(t_k)
\end{bmatrix}
\]

\[
\tilde{\mathbf{x}}(t_{k+1}) = \tilde{\mathbf{A}} \cdot \begin{bmatrix}
\mathbf{x}(t_k) \\
u(t_k)
\end{bmatrix}
\]
Affine System Representation for a One-Mass Oscillator

\[
m \cdot \ddot{x}(t) + c \cdot x(t) + k \cdot x(t) = F(t)
\]

\[
\begin{bmatrix}
\dot{x}(t) \\
\ddot{x}(t)
\end{bmatrix}
= \begin{bmatrix}
0 & 1 \\
-k/m & -c/m
\end{bmatrix}
\begin{bmatrix}
x(t) \\
\dot{x}(t)
\end{bmatrix}
+ \begin{bmatrix}
0 \\
1/m
\end{bmatrix} F(t)
\]
Affine System Representation for a One-Mass Oscillator

\[ m \cdot \ddot{x}(t) + c \cdot x(t) + k \cdot x(t) = F(t) \]

\[
\begin{bmatrix}
\dot{x}(t) \\
\ddot{x}(t)
\end{bmatrix} = 
\begin{bmatrix}
0 & -\frac{k}{m} \\
-\frac{c}{m} & 0
\end{bmatrix}
\begin{bmatrix}
x(t) \\
\dot{x}(t)
\end{bmatrix} + 
\begin{bmatrix}
0 \\
\frac{1}{m}
\end{bmatrix} F(t)
\]

Euler Discretization with constant input variable

\[ \tilde{x}(t_{k+1}) = \tilde{A} \cdot \tilde{x}(t_k) \]

\[ \tilde{x}(t_1) = \tilde{A} \cdot \tilde{x}(t_0) \]
\[ \tilde{x}(t_2) = \tilde{A} \cdot \tilde{x}(t_1) = \tilde{A} \cdot (\tilde{A} \cdot \tilde{x}(t_0)) = \tilde{A}^2 \cdot \tilde{x}(t_0) \]
\[ \vdots \]
\[ \tilde{x}(t_{k+1}) = \tilde{A}^k \cdot \tilde{x}(t_0) \]
Affine System Representation for a One-Mass Oscillator

\[ m \cdot \ddot{x}(t) + c \cdot x(t) + k \cdot x(t) = F(t) \]

\[
\begin{bmatrix}
\dot{x}(t) \\
\ddot{x}(t)
\end{bmatrix} =
\begin{bmatrix}
0 & 0 \\
-\frac{k}{m} & -\frac{c}{m}
\end{bmatrix}
\begin{bmatrix}
x(t) \\
\dot{x}(t)
\end{bmatrix} +
\begin{bmatrix}
0 \\
\frac{1}{m}
\end{bmatrix} F(t)
\]

Euler Discretization with constant input variable

\[ \tilde{x}(t_{k+1}) = \tilde{A} \cdot \tilde{x}(t_k) \]

\[ \tilde{x}(t_1) = \tilde{A} \cdot \tilde{x}(t_0) \]

\[ \tilde{x}(t_2) = \tilde{A} \cdot \tilde{x}(t_1) = \tilde{A} \cdot (\tilde{A} \cdot \tilde{x}(t_0)) = \tilde{A}^2 \cdot \tilde{x}(t_0) \]

\[ \vdots \]

\[ \tilde{x}(t_{k+1}) = \tilde{A}^k \cdot \tilde{x}(t_0) \]

Example in MATLAB
Part 1

1. Fundamentals of Interval Arithmetic

Software Demonstration
Software Demonstration of Interval Arithmetics

- **INTLAB**: INTerval LABoratory — Matlab toolbox for Reliable Computing
- **C-XSC** — C++ Class Library

Importance of Verified Computing

- Floating-point arithmetics on today’s computer is always affected by a maximum accuracy
  \[ \Rightarrow \text{rounded results differ at most by 1 unit in the last place from the exact result} \]
- After further calculations, the result may be wrong because of rounding
  \[ \Rightarrow \text{Results have to be verified} \]
INTerval LABoratory

Development by Prof. Dr. Siegfried M. Rump, Hamburg University of Technology
http://www.ti3.tu-harburg.de/rump/intlab/

Standard interval arithmetic
- Arithmetic operators $+, -, \cdot, /$
- Real and complex intervals

Automatic Differentiation
- Forward mode: forward substitution to find the derivatives
- Compute derivatives using the chain rule for composite functions
- Calculate an enclosure of the true derivative of an interval function
INTErval LABoratory

http://www.ti3.tu-harburg.de/rump/intlab/

**Verified Functions for Linear Systems of Equations**

- Solution of linear systems of equations in a verified way
- Computation of an enclosure of the solution hull
- Aim: produce a tight bound on the true solution

**Rounding Mode**

- Function `setround(y)`: changes the rounding mode of the processor to the nearest (0), round down (-1), round up (1)
- Function `getround` outputs the current rounding mode
INTerval LABoratory
http://www.ti3.tu-harburg.de/rump/intlab/

Verified Functions for Linear Systems of Equations
- Solution of linear systems of equations in a verified way
- Computation of an enclosure of the solution hull
- Aim: produce a tight bound on the true solution

Rounding Mode
- Function `setround(y)`: changes the rounding mode of the processor to the nearest (0), round down (-1), round up (1)
- Function `getround` outputs the current rounding mode

Matlab Example: `intlab_fundamentals.m` and `taylorExpansion.m`
C-XSC

http://www2.math.uni-wuppertal.de/~xsc/xsc/cxsc.html

Information

- C++ Class Library for Extended Scientific Computing
- Compatible to Windows, Linux, Mac Os

Data Types

- real, interval, complex, cinterval (complex interval)
- rvector, ivector, cvector, civector (complex interval vector)
- rmatrix, imatrix, cmatrix, cimatrix (complex interval matrix)
C-XSC

http://www2.math.uni-wuppertal.de/~xsc/xsc/cxsc.html

Data Types

- real, interval, complex, cinterval (complex interval)
- rvector, ivector, cvector, civector (complex interval vector)
- rmatrix, imatrix, cmatrix, cimatrix (complex interval matrix)

Rounding Mode

- by-default: all operations are only one rounding away from the exact result
- Modes: long fix-point accumulator for dot product computations (default), pure floating point operations, DotK algorithm (based on so-called error free transformations)
C-XSC

http://www2.math.uni-wuppertal.de/~xsc/xsc/cxsc.html

Data Types fundamentals.cpp

- real, interval, complex, cinterval (complex interval)
- rvector, ivector, cvector, civector (complex interval vector)
- rmatrix, imatrix, cmatrix, cimatrix (complex interval matrix)

Rounding Mode

- by-default: all operations are only one rounding away from the exact result
- Modes: long fix-point accumulator for dot product computations (default), pure floating point operations, DotK algorithm (based on so-called error free transformations)
FADBAD++

http://www.fadbad.com/fadbad.html#General_introduction

**General**

- Flexible Automatic Differentiation using templates and operator overloading in C++
- Implementing the forward, backward and Taylor methods utilizing C++ templates and operator overloading
- Differentiate a C++ function by replacing all occurrences of the original arithmetic type with the AD-template version
- Possible to generate high-order derivatives
FADBAD++

http://www.fadbad.com/fadbad.html#General_introduction

General taylorExpansion_FF.cpp and taylorExpansion_T.cpp

- Flexible Automatic Differentiation using templates and operator overloading in C++
- Implementing the forward, backward and Taylor methods utilizing C++ templates and operator overloading
- Differentiate a C++ function by replacing all occurrences of the original arithmetic type with the AD-template version
- Possible to generate high-order derivatives

Advantage of Using C++ instead of Intlab

Interface to rapid control prototyping environments is possible
Thank you for your attention!

All presentations, examples and selected publications will be available at
http://www.com.uni-rostock.de/ecc15/
in the 1st week of August

User: ECC15
Password: intervals-are-fun