常微分方程式の精度保証付き数値解法

柏木 雅英
kashi@waseda.jp
http://verifiedby.me/

早稲田大学 基幹理工学部 応用数理学科

2018 年 12 月 26 日

概要

- 常微分方程式の精度保証付き数値計算
- ベキ級数演算 (PSA)
- Lohner 法
- Affine Arithmeric と QR 分解法
- kv ライブラリの紹介
- 既存の実装の紹介
- AWA, Intlab, kv の比較

常微分方程式の精度保証

大きく分けて 2 種類の方法がある。

Taylor 展開を利用する方法

初期値問題を Taylor 展開をもとに精度保証付きで解く方法。境界値問題等は射撃法と非線形方程式の精度保証法を組み合わせて解く。

関数解析的方法

境界条件を満たす適当な関数空間を設定し、解全体を関数方程式と見て何らかの不動点定理を用いて解の存在保証と誤差評価を行う。

Taylor 展開ベースの方法について解説する。

初期値問題の精度保証

一階連立常微分方程式

$$\frac{dx}{dt} = f(x,t), \quad x \in \mathbb{R}^l, t \in \mathbb{R}$$
$$x(t_0) = x_0$$

両辺を積分して Picard 型の不動点形式に

$$x(t) = x_0 + \int_{t_0}^{t} f(x(t),t)dt$$

Picard-Lindelöf の定理 (1)

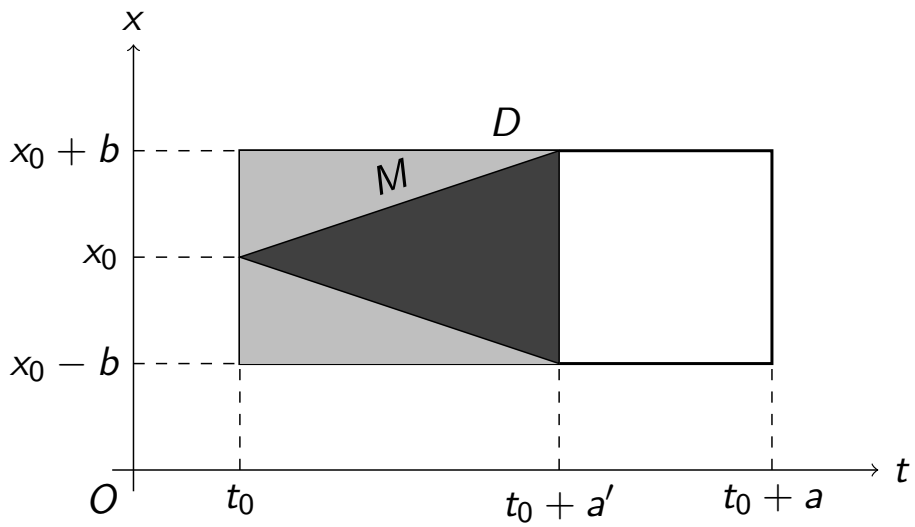
Picard-Lindelöf の定理

$$\frac{dx}{dt} = f(x,t), \quad x \in \mathbb{R}^l, t \in \mathbb{R}$$
$$x(t_0) = x_0$$

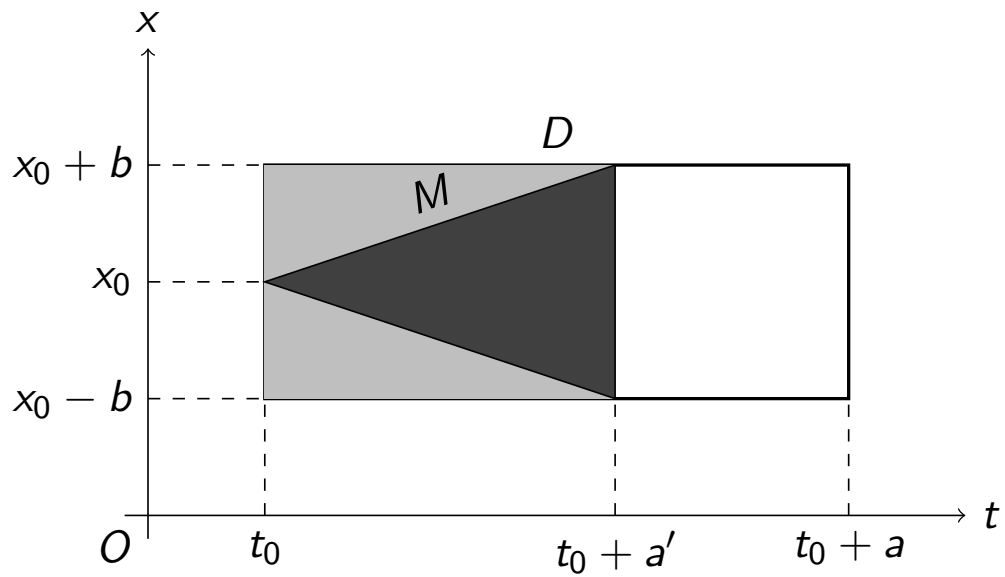
について、f がある領域 $D = \{ (x,t) \mid t_0 \le t \le t_0 + a, \|x - x_0\| \le b \}$ で Lipschitz 条件

$$\|f(x_1,t) - f(x_2,t)\| \le L\|x_1 - x_2\|$$

を満たすとする。このとき、$M = \sup_{(x,t)\in D} \|f(x,t)\|$、$a' = \min(a, \frac{b}{M})$ とすると、$[t_0, t_0 + a']$ でただ一つの解を持つ。



Picard-Lindelöf の定理 (2)



Picard-Lindelöf の定理の証明の概略

$$P : x(t) \mapsto x_0 + \int_{t_0}^{t} f(x(t), t)dt$$

とし、$D' = \{ (x, t) \mid t_0 \leq t \leq t_0 + a', \|x - x_0\| \leq b \}$ とすると、

- P は D' 内の全ての連続関数を D' 内に移す。
- P は重み付き最大値ノルム $\|x(t)\| = \sup_{t_0 \leq t \leq t_0 + a'} e^{-2L(t-t_0)} \|x(t)\|$ を入れると縮小写像となる。
- よって、P の不動点は D' に唯一存在する。

Picard 型の不動点形式による近似解の生成

例えば、次の問題を Picard 型に直して、

$$\frac{dx}{dt} = -x^2$$
$$x(0) = 1, \quad t \in [0, 0.1]$$

$$P : x(t) \mapsto 1 + \int_0^t -x(t)^2 dt$$

初期値を定数関数 $x_0(t) = 1$ として $x_{i+1}(t) = P(x_i(t))$ のように反復すると、

$$x_0(t) = 1$$
$$x_1(t) = 1 - t$$
$$x_2(t) = 1 - t + t^2 - \frac{1}{3}t^3$$
$$x_3(t) = 1 - t + t^2 - t^3 + \frac{2}{3}t^4 - \frac{1}{3}t^5 + \frac{1}{9}t^6 - \frac{1}{63}t^7$$
$$x_4(t) = \cdots$$

解に近づいていく。

Picard 型の不動点形式による近似解の生成

例えば、次の問題を Picard 型に直して、

$$\frac{dx}{dt} = -x^2$$
$$x(0) = 1, \quad t \in [0, 0.1]$$

$$P : x(t) \mapsto 1 + \int_0^t -x(t)^2 dt$$

初期値を定数関数 $x_0(t) = 1$ として $x_{i+1}(t) = P(x_i(t))$ のように反復すると、

$$x_0(t) = 1$$
$$x_1(t) = 1 - t$$
$$x_2(t) = 1 - t + t^2 - \frac{1}{3}t^3$$
$$x_3(t) = 1 - t + t^2 - t^3 + \frac{2}{3}t^4 - \frac{1}{3}t^5 + \frac{1}{9}t^6 - \frac{1}{63}t^7$$
$$x_4(t) = \cdots$$

解に近づいていく。

Picard 型の不動点形式による近似解の生成

例えば、次の問題を Picard 型に直して、

$$\frac{dx}{dt} = -x^2$$
$$x(0) = 1, \quad t \in [0, 0.1]$$

$$P : x(t) \mapsto 1 + \int_0^t -x(t)^2 dt$$

初期値を定数関数 $x_0(t) = 1$ として $x_{i+1}(t) = P(x_i(t))$ のように反復すると、

$$x_0(t) = 1$$
$$x_1(t) = 1 - t$$
$$x_2(t) = 1 - t + t^2 - \frac{1}{3}t^3$$
$$x_3(t) = 1 - t + t^2 - t^3 + \frac{2}{3}t^4 - \frac{1}{3}t^5 + \frac{1}{9}t^6 - \frac{1}{63}t^7$$
$$x_4(t) = \cdots$$

解に近づいていく。

Picard 型の不動点形式による近似解の生成

例えば、次の問題を Picard 型に直して、

$$\frac{dx}{dt} = -x^2$$
$$x(0) = 1, \quad t \in [0, 0.1]$$

$$P : x(t) \mapsto 1 + \int_0^t -x(t)^2 dt$$

初期値を定数関数 $x_0(t) = 1$ として $x_{i+1}(t) = P(x_i(t))$ のように反復すると、

$$x_0(t) = 1$$
$$x_1(t) = 1 - t$$
$$x_2(t) = 1 - t + t^2 - \frac{1}{3}t^3$$
$$x_3(t) = 1 - t + t^2 - t^3 + \frac{2}{3}t^4 - \frac{1}{3}t^5 + \frac{1}{9}t^6 - \frac{1}{63}t^7$$
$$x_4(t) = \cdots$$

解に近づいていく。

Picard 型の不動点形式による近似解の生成

例えば、次の問題を Picard 型に直して、

$$\frac{dx}{dt} = -x^2$$
$$x(0) = 1, \quad t \in [0, 0.1]$$

$$P : x(t) \mapsto 1 + \int_0^t -x(t)^2 dt$$

初期値を定数関数 $x_0(t) = 1$ として $x_{i+1}(t) = P(x_i(t))$ のように反復すると、

$$x_0(t) = 1$$
$$x_1(t) = 1 - t$$
$$x_2(t) = 1 - t + t^2 - \frac{1}{3}t^3$$
$$x_3(t) = 1 - t + t^2 - t^3 + \frac{2}{3}t^4 - \frac{1}{3}t^5 + \frac{1}{9}t^6 - \frac{1}{63}t^7$$
$$x_4(t) = \cdots$$

解に近づいていく。

Picard 型の不動点形式による近似解の生成

例えば、次の問題を Picard 型に直して、

$$\frac{dx}{dt} = -x^2$$
$$x(0) = 1, \quad t \in [0, 0.1]$$

$$P : x(t) \mapsto 1 + \int_0^t -x(t)^2 dt$$

初期値を定数関数 $x_0(t) = 1$ として $x_{i+1}(t) = P(x_i(t))$ のように反復すると、

$$x_0(t) = 1$$
$$x_1(t) = 1 - t$$
$$x_2(t) = 1 - t + t^2 - \frac{1}{3}t^3$$
$$x_3(t) = 1 - t + t^2 - t^3 + \frac{2}{3}t^4 - \frac{1}{3}t^5 + \frac{1}{9}t^6 - \frac{1}{63}t^7$$
$$x_4(t) = \cdots$$

解に近づいていく。解の Taylor 展開が 1 次ずつ得られる。

Picard 型の不動点形式による解の包含

2 次の Taylor 展開

$$X_2 = 1 - t + t^2$$

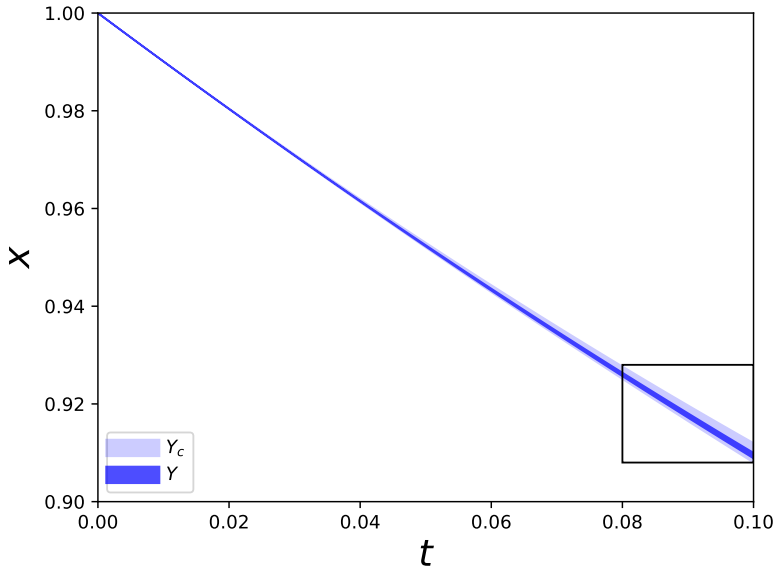
の最終項の係数を膨らませた区間関数

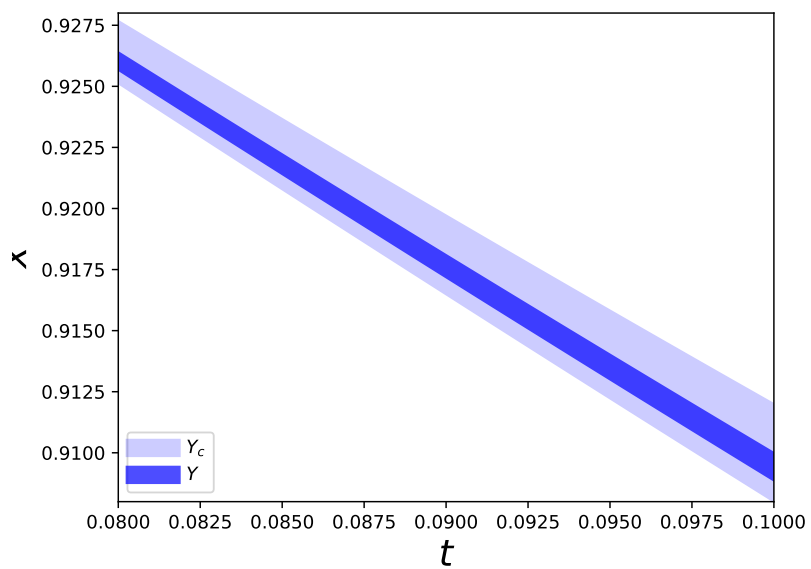
$$Y_c = 1 - t + [0.8, 1.2]t^2$$

に対して、

$$P(Y_c) \subset Y = 1 - t + [0.886, 1]t^2$$

となる。最終項の係数の比較で $Y \subset Y_c$ なので、Y に解が存在する。





ベキ級数演算の導入

PSA = Power Series Arithmeric

Type-I PSA

$$
\begin{aligned}
x_0(t) &= 1 \\
x_1(t) &= 1 - t \\
x_2(t) &= 1 - t + t^2 - \frac{1}{3}t^3 \\
x_3(t) &= 1 - t + t^2 - t^3 + \frac{2}{3}t^4 - \frac{1}{3}t^5 + \frac{1}{9}t^6 - \frac{1}{63}t^7 \\
x_4(t) &= \cdots
\end{aligned}
$$

の、Taylor 展開の級数を決めるのに不要な部分を計算したくない!

Type-II PSA

$$Y_c = 1 - t + [0.8, 1.2]t^2$$

に対する包含

$$P(Y_c) \subset Y = 1 - t + [0.886, 1]t^2$$

を計算したい!

ベキ級数演算 (psa)

- 常微分方程式の初期値問題や数値積分の精度保証に使う。高階微分の計算にも使える。
- Type-I と Type-II の 2 種類がある。n を整数として、

  Type-I PSA　単に n 次より高次の項を捨てる。

  Type-II PSA　n 次より高次の項の影響を最高次の項 t^n の区間係数に含めさせる。

PSA の例 (積)

$1+2t-3t^2$ と $1-t+t^2$ の積	
Type-I PSA	Type-II PSA
定義域は決めなくてよい	定義域 $=[0,0.1]$
$1+t-4t^2$	$1+t+[-4,-3.5]t^2$

$$(1+2t-3t^2)(1-t+t^2) = 1+t-4t^2+5t^3-3t^4$$
$$= 1+t+(-4+5t-3t^2)t^2 \in 1+t+[-4,-3.5]t^2$$

Type-I PSA

ベキ級数型

$$x_0 + x_1 t + x_2 t^2 + \cdots + x_n t^n$$

- ベキ級数型同士の加減乗除。
- ベキ級数型に対する exp, log, $\int$ などの数学関数。
- 演算結果の n 次までの項を残し、$n+1$ 次以降は切り捨てる。
- 以下とほとんど同じ:
  - Mathemarica の ‘`Series`’。
  - Intlab の ‘`taylor`’。
  - 1 変数関数の高階微分を計算する自動微分法。

Type-II PSA

ベキ級数型

$$x_0 + x_1 t + x_2 t^2 + \cdots + x_n t^n$$

- 固定された有限閉区間 $D = [t_1, t_2]$ 上で定義される。
- 演算結果は n 次までしか保持しないが、$n+1$ 次以降の項の影響は n 次の項の係数を区間にすることで吸収する。
- 係数 $x_0, \cdots, x_n$ は区間。
- ただし多くの場合、$x_0, \cdots, x_{n-1}$ は幅の狭い区間、x_n は幅の広い区間。

Type-II PSA の演算規則 (1/5)

$$x(t) = x_0 + x_1 t + x_2 t^2 + \cdots + x_n t^n$$
$$y(t) = y_0 + y_1 t + y_2 t^2 + \cdots + y_n t^n$$

加減算

$$x(t) \pm y(t) = (x_0 \pm y_0) + (x_1 \pm y_1)t + \cdots (x_n \pm y_n)t^n$$

加算の例

$$x(t) = 1 + 2t - 3t^2$$
$$y(t) = 1 - t + t^2$$

$$x(t) + y(t) = 2 + t - 2t^2$$

Type-II PSA の演算規則 (2/5)

乗算

1. まず、打ち切り無しで乗算を行う。

$$x(t) \times y(t) = z_0 + z_1 t + \cdots + z_{2n} t^{2n}$$

$$z_k = \sum_{i=\max(0,k-n)}^{\min(k,n)} x_i y_{k-i}$$

2. $2n$ 次から n 次に減次する。

m 次から n 次への減次

$$x_0 + x_1 t + x_2 t^2 + \cdots + x_m t^m \Longrightarrow z_0 + z_1 t + \cdots + z_n t^n$$

$$z_i = x_i \quad (0 \le i \le n-1)$$

$$z_n = \left\{ \sum_{i=n}^{m} x_i t^{i-n} \mid t \in D \right\}$$

Type-II PSA の演算規則 (3/5)

乗算の例

定義域を $D = [0, 0.1]$ とする。

$$x(t) = 1 + 2t - 3t^2$$
$$y(t) = 1 - t + t^2$$

$$\begin{aligned} x(t) \times y(t) &= 1 + t - 4t^2 + 5t^3 - 3t^4 \\ &= 1 + t + (-4 + 5t - 3t^2)t^2 \\ &\in 1 + t + \left\{-4 + 5t - 3t^2 \mid t \in [0, 0.1]\right\} t^2 \\ &= 1 + t + [-4, -3.5]t^2 \end{aligned}$$

Type-II PSA の演算規則 (4/5)

sin などの数学関数

その関数を g として、

$$
\begin{aligned}
&g(x_0 + x_1 t + \cdots + x_n t^n) \\
&= g(x_0) + \sum_{i=1}^{n-1} \frac{1}{i!} g^{(i)}(x_0)(x_1 t + \cdots + x_n t^n)^i \\
&+ \frac{1}{n!} g^{(n)} \left(\mathrm{hull} \left(x_0, \left\{ \sum_{i=0}^{n} x_i t^i \;\middle|\; t \in D \right\} \right) \right) (x_1 t + \cdots + x_n t^n)^n
\end{aligned}
$$

のように g の点 x_0 での剰余項付きの Taylor 展開に代入することによって得る。この計算中に現れる加算や乗算は Type-II PSA で行う。

Type-II PSA の演算規則 (5/5)

除算

$x \div y = x \times (1/y)$ と乗算と逆数関数に分解

不定積分

$$\int_0^t x(t)dt = x_0 t + \frac{x_1}{2} t^2 + \cdots \frac{x_n}{n+1} t^{n+1}$$

常微分方程式の初期値問題

一階連立常微分方程式

$$\frac{dx}{dt} = f(x,t), \quad x \in \mathbb{R}^l, t \in \mathbb{R}$$
$$x(t_0) = x_0$$

初期値問題の精度保証アルゴリズム

$t_0 < t_1 < t_2 < \ldots$ に対して、

- ベキ級数演算 (PSA) を用いた、$x(t_i)$ を元に $x(t_{i+1})$ を精度保証付きで計算する方法 (短い区間での精度保証)
- Affine Arithmetic を用いて区間幅の膨らみを抑制しながら、短い区間での精度保証で得られた解を長い区間に渡って接続する方法

短い区間での精度保証 (1/3)

$v = x(t_s)$ の値を初期値として、$x(t_e)$ の値を計算する。

平行移動 & 両辺を積分で不動点形式に

$$x(t) = v + \int_0^t f(x(t), t + t_s)dt$$
$$(v = x(t_i), \quad t \in [0, t_e - t_s])$$

解の Taylor 展開の生成

Type-I PSA 型の変数 $X_0 = v$, $T = t$ を用いて、$k = 0$ とし、

1. 次数 k の Type-I PSA で以下を計算

$$X_{k+1} = v + \int_0^t f(X_k, T + t_s)dt$$

2. 次数 $k = k + 1$ とする。

を n 回繰り返すと、X_n として解の n 次の Taylor 展開が得られる。

短い区間での精度保証 (2/3)

解の存在保証

Type-II PSA の定義域を $D = [0, t_e - t_s]$ と設定し、Type-I PSA の反復で得られた n 次の Taylor 近似

$$X_n = x_0 + x_1 t + x_2 t^2 + \cdots + x_n t^n$$

と $T = t$ を用いて、

1. X_n の最終項の係数を膨らませた候補者集合

$$Y_c = x_0 + x_1 t + x_2 t^2 + \cdots + V_c t^n$$

を作成する。

2. $v + \int_0^t f(Y_c, T + t_s)dt$ を次数 n の Type-II PSA で計算し、$n+1$ 次から n 次に減次したものを

$$Y = x_0 + x_1 t + x_2 t^2 + \cdots + V t^n$$

とする。$n-1$ 次までの係数は X_n と全く同じになることに注意。

3. $V \subset V_c$ なら Y 内に解の存在が保証される。

短い区間での精度保証 (3/3)

解を含みそうな候補者集合の作成は、例えば次のように行なうことが出来る。

候補者集合の作成

1. $v + \int_0^t f(X_n, T + t_s)dt$ を次数 n の Type-II PSA で計算し、$n+1$ 次から n 次に減次したものを $Y_0 = x_0 + x_1 t + \cdots + V_0 t^n$ とする。
2. $r = ||V_0 - x_n||$ とし、

$$V_c = x_n + 2r\left([-1, 1], \ldots, [-1, 1]\right)^T$$

とする。(半径を Newton 法の修正量の 2 倍にする。)

短い区間での精度保証の例

$$\frac{dx}{dt} = -x^2$$
$$x(0) = 1, \quad t \in [0, 0.1]$$

展開の次数: $n = 2$
10 進 3 桁演算。

(Type-I PSA による Taylor 展開の生成)

$$X_0 = \boxed{1}$$
$$X_1 = 1 + \int_0^t (-X_0^2)dt$$
$$= 1 + \int_0^t (-1)dt$$
$$= \boxed{1 - t}$$
$$X_2 = 1 + \int_0^t (-X_1^2)dt$$
$$= 1 + \int_0^t (-(1-t)^2)dt$$
$$= 1 + \int_0^t (-(1-2t))dt$$
$$= \boxed{1 - t + t^2}$$

(候補者集合の生成)

$$1 + \int_0^t (-X_2^2)dt$$
$$= 1 + \int_0^t (-(1 - t + t^2)^2)dt$$
$$= 1 + \int_0^t (-(1 - 2t + [2.8, 3]t^2))dt$$
$$= 1 - t + t^2 + [-1, -0.933]t^3$$

2 次に減次して、

$$Y_0 = 1 - t + [0.9, 1]t^2$$

$r = ||[0.9, 1] - 1|| = 0.1$ なので、

$$Y_c = \boxed{1 - t + [0.8, 1.2]t^2}$$

(Type-II PSA による精度保証)

$$1 + \int_0^t (-Y_c^2)dt$$
$$= 1 - t + t^2 + [-1.14, -0.786]t^3$$

2 次に減次して、

$$Y = \boxed{1 - t + [0.886, 1]t^2}$$

$[0.886, 1] \subset [0.8, 1.2]$ なので、Y 内に真の解が存在する。

Lohner 法 (1/3)

(以下は未定係数法など、任意の別の方法でもよい。)

平行移動 & 両辺を積分で不動点形式に

$$x(t) = v + \int_0^t f(x(t), t + t_i)dt$$
$$(v = x(t_s), \quad t \in [0, t_e - t_s])$$

解の Taylor 展開の生成

Type-I PSA 型の変数 $X_0 = v$, $T = t$ を用いて、$k = 0$ とし、

1. 次数 k の Type-I PSA で以下を計算

$$X_{k+1} = v + \int_0^t f(X_k, T + t_s)dt$$

2. 次数 $k = k + 1$ とする。

を n 回繰り返すと、X_n として解の n 次の Taylor 展開が得られる。求まった X_n の t^i の係数を α_i とする。

$$X_n = v + \alpha_1 t + \alpha_2 t^2 + \cdots + \alpha_n t^n$$

Lohner 法 (2/3)

大雑把な解の包含 V

$[0, t_e - t_s]$ における解 $x(t)$ を包含する候補者区間 $V_c \subset \mathbb{IR}^l$ を考える。

$$\begin{aligned} P(V_c) &\subset v + \int_0^t f(V_c, [0, t_e - t_s] + t_i)dt \\ &\subset v + f(V_c, [t_s, t_e])t \\ &\subset v + f(V_c, [t_s, t_e])[0, t_e - t_s] \end{aligned}$$

により、$V = v + f(V_c, [t_e, t_s])[0, t_e - t_s] \subset V_c$ が成立すれば V 内に $x(t)$ が包含される。

反復改良

$V_1 = V, k = 1$ とし、

$$V_{k+1} = V_k \cap (v + f(V_k, [t_e, t_s])[0, t_e - t_s])$$

で更に精度を上げることも出来る。

Lohner 法 (3/3)

区間初期値による解の Taylor 展開の生成

初期値を v の代わりに V、初期時刻を t_s の代わりに $[t_s, t_e]$ として、再度 Taylor 展開の生成を行う。すなわち、Type-I PSA 型の変数 $X_0 = V$, $T = t$ を用いて、$k = 0$ とし、

1. 次数 k の Type-I PSA で以下を計算

$$X_{k+1} = V + \int_0^t f(X_k, T + [t_s, t_e])dt$$

2. 次数 $k = k + 1$ とする。

を n 回繰り返すと、X_n として解の n 次の Taylor 展開が得られる。求まった X_n の t^i の係数を β_i とする。

$$X_n = V + \beta_1 t + \beta_2 t^2 + \cdots + \beta_n t^n$$

精度保証解

このとき、

$$v + \alpha_1 t + \alpha_2 t^2 + \cdots + \alpha_{n-1} t^{n-1} + \beta_n t^n$$

に真の解が存在する。

短い区間での Lohner 法の例

$$\frac{dx}{dt} = -x^2$$
$$x(0) = 1, \quad t \in [0, 0.1]$$

展開の次数: $n = 2$
10 進 3 桁演算。

(Type-I PSA による Taylor 展開の生成)

$$X_0 = \boxed{1}$$
$$X_1 = 1 + \int_0^t (-X_0^2)dt$$
$$= 1 + \int_0^t (-1)dt$$
$$= \boxed{1 - t}$$
$$X_2 = 1 + \int_0^t (-X_1^2)dt$$
$$= 1 + \int_0^t (-(1-t)^2)dt$$
$$= 1 + \int_0^t (-(1-2t))dt$$
$$= \boxed{1 - t + t^2}$$

(大雑把な解の包含の生成)

$$r = ||\mathrm{mag}((-1^2)[0, 0.1])|| = 0.1$$
$$V_c = 1 + 2 \times 0.1 \times [-1, 1] = \boxed{[0.8, 1.2]}$$
$$1 + (-[0.8, 1.2]^2)[0, 0.1] = \boxed{[0.856, 1]}$$

$[0.856, 1] \subset [0.8, 1.2]$ なので $[0.856, 1]$ 内に真の解が存在する。

(解の包含の生成) 初期値を $[0.856, 1]$ として Taylor 展開を行う。

$$X_0 = [0.856, 1]$$
$$X_1 = [0.856, 1] + [-1, -0.732]t$$
$$X_2 = \boxed{[0.856, 1] + [-1, -0.732]t + [0.627, 1]t^2}$$

初期値を 1 とした Taylor 展開と合成した、

$$\boxed{1 - t + [0.627, 1]t^2}$$

内に、真の解が存在する。

長い区間に渡って接続する (1/2)

flow map

常微分方程式において、値 $x(t_s)$ ($t = t_s$ における初期値) に対して値 $x(t_e)$ (時刻 $t = t_e$ における解の値) を対応させる写像

$$\phi_{t_s,t_e} : \mathbb{R}^l \to \mathbb{R}^l, \quad \phi_{t_s,t_e} : x(t_s) \mapsto x(t_e)$$

を flow map と呼ぶ。

初期値に関する変分方程式

$x^*(t)$ を v を初期値とした常微分方程式の解とすると、

$$\frac{d}{dt}y(t) = f_x(x^*(t), t)y(t), \quad y \in \mathbb{R}^{l\times l}$$
$$y(t_s) = I, \quad t \in [t_s, t_e]$$

を解くことによって、flow map の微分 (ヤコビ行列) を $\phi'_{t_s,t_e}(v) = y(t_e)$ で得られる。

長い区間に渡って接続する (2/2)

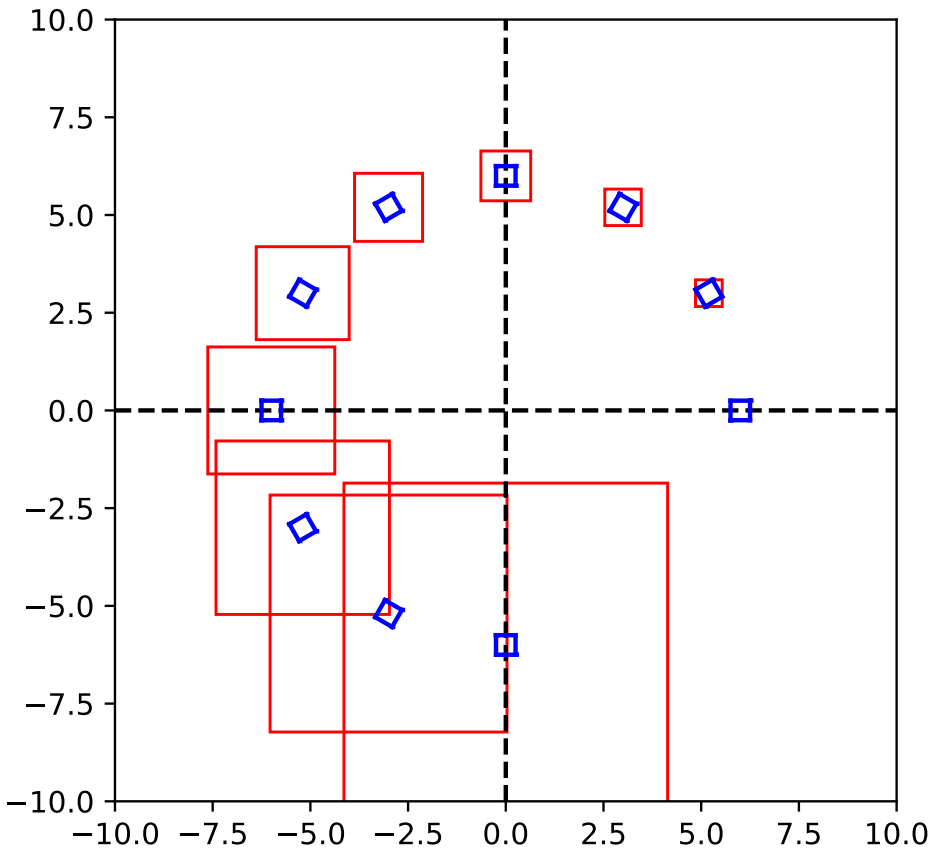
時刻 $t_0 < t_1 < t_2 < \cdots$ における解の包含を J_i, $c_i = \mathrm{mid}(J_i)$ とする。

平均値形式

$$\phi_{t_i,t_{i+1}}(x) \in \phi_{t_i,t_{i+1}}(c_i) + \phi'_{t_i,t_{i+1}}(J_i)(x - c_i)$$

- 単純にこのまま計算すると、wrapping effect によって区間幅が激しく増大する。
- wrapping effect を抑えるため、affine arithmetic を使って接続する

Wrapping Effect



成分が区間であるような「区間ベクトル」が $\mathbb{R}^n$ の超直方体領域しか表現できないため、真の像が直方体で包含されることにより区間幅が増大する現象。

区間 $I_0 = \begin{pmatrix} [5.75, 6.25] \\ [-0.25, 0.25] \end{pmatrix}$ に対して $I_{k+1} = \begin{pmatrix} \sqrt{3}/2 & -0.5 \\ 0.5 & \sqrt{3}/2 \end{pmatrix} I_k$ のように行列を作用させた例。

青が真の像、赤が区間演算の結果。

Affine Arithmetic

Affine Arithmetic とは

- 区間演算の過大評価を抑制できる。その代わり計算時間がかかる。
- 全ての変数について、入力変数またはノイズに関する依存性を保持するため、区間幅の爆発を防げる。
- 全ての数値は

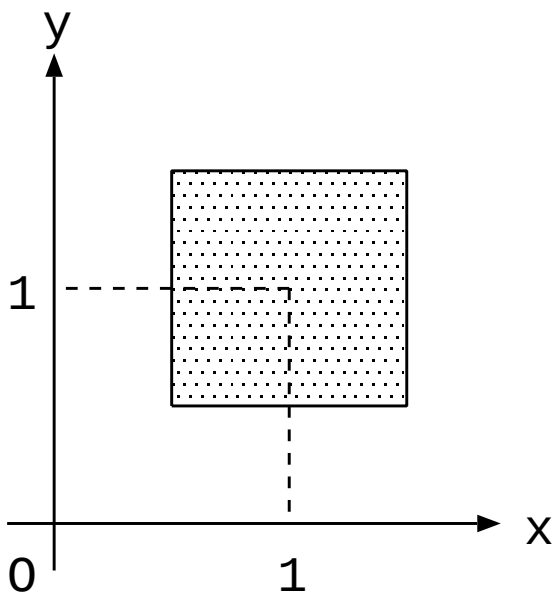
$$x_0 + x_1\varepsilon_1 + x_2\varepsilon_2 + \cdots + x_n\varepsilon_n$$

のような Affine 形式で表現される。ε_i は $-1 \leq \varepsilon_i \leq 1$ を動くダミー変数であり、その係数により依存性を表現する。

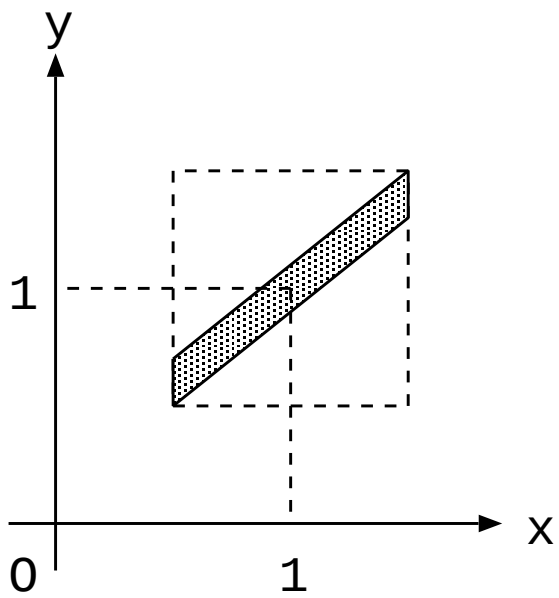
- 乗除算や数学関数などの非線形演算が出現する度にダミー変数の数が増え、計算が遅くなる。

ダミー変数 ε は異なる変量間の相関性を表現する

$$x = 1 + 0.5\varepsilon_1$$
$$y = 1 + 0.5\varepsilon_2$$



$$x = 1 + 0.5\varepsilon_1$$
$$y = 1 + 0.4\varepsilon_1 + 0.1\varepsilon_2$$



x, y それぞれの変域は同じだが、 "joint range" は異なる。

区間との相互変換

区間 $\to$ affine

$$\begin{pmatrix} [\underline{x_1}, \overline{x_1}] \\ [\underline{x_2}, \overline{x_2}] \\ \vdots \\ [\underline{x_n}, \overline{x_n}] \end{pmatrix} \Longrightarrow \begin{pmatrix} \frac{\overline{x_1}+\underline{x_1}}{2} + \frac{\overline{x_1}-\underline{x_1}}{2}\varepsilon_1 \\ \frac{\overline{x_2}+\underline{x_2}}{2} + \frac{\overline{x_2}-\underline{x_2}}{2}\varepsilon_2 \\ \vdots \\ \frac{\overline{x_n}+\underline{x_n}}{2} + \frac{\overline{x_n}-\underline{x_n}}{2}\varepsilon_n \end{pmatrix}$$

affine $\to$ 区間

$$x = a_0 + a_1\varepsilon_1 + \cdots + a_n\varepsilon_n$$

$$\Downarrow$$

$$[a_0 - \delta, a_0 + \delta] \qquad , \quad (\delta = \sum_{i=1}^{n} |a_i|)$$

線形計算は簡単

$$
\begin{aligned}
x &= x_0 + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n \\
y &= y_0 + y_1\varepsilon_1 + \cdots + y_n\varepsilon_n
\end{aligned}
$$

加算、減算

$$
\begin{aligned}
x \pm y &= (x_0 \pm y_0) + (x_1 \pm y_1)\varepsilon_1 + \cdots + (x_n \pm y_n)\varepsilon_n \\
x \pm \alpha &= (x_0 \pm \alpha) + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n \quad .
\end{aligned}
$$

定数の乗算

$$
\alpha x = (\alpha x_0) + (\alpha x_1)\varepsilon_1 + \cdots + (\alpha x_n)\varepsilon_n \quad .
$$

非線形単項演算 (数学関数)

f: $\exp, \log, \cdots$ などの単項演算
affine 変数 x に対して、$z = f(x)$ を計算することを考える。

$$x = x_0 + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n$$

1. x の変動区間 I を次のように計算する:

$$I = [x_0 - \delta, x_0 + \delta], \quad \delta = \sum_{i=1}^{n} |x_i| \quad ,$$

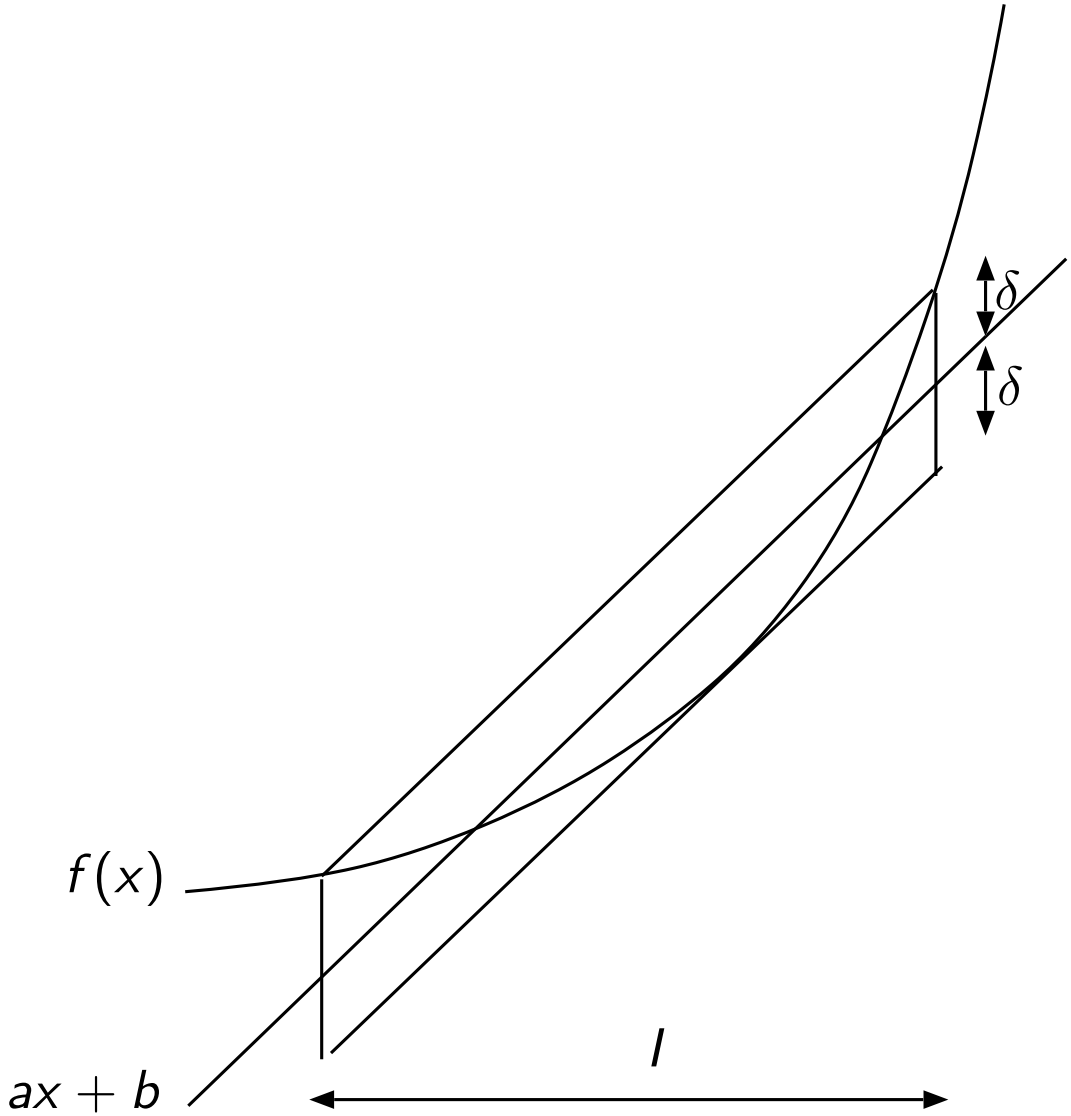
2. f の I 上における線形近似 $ax + b$ と最大誤差 δ を計算する:

$$\delta = \max_{x \in I} |f(x) - (ax + b)|$$

3. 計算結果 z は次の式で計算する:

$$a(x_0 + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n) + b + \delta\varepsilon_{n+1}$$

線形近似 $ax + b$ と誤差 δ



非線形二項演算

二項演算子 $g(x, y)$ に対して、線形近似 $ax + by + c$ を考える。(単項演算の場合とほぼ同様。)

乗算

$$\begin{aligned} z &= y_0 x + x_0 y - x_0 y_0 + \delta_x \delta_y \varepsilon_{n+1} \\ &= x_0 y_0 + \sum_{i=1}^{n} (y_0 x_i + x_0 y_i) \varepsilon_i \\ &\quad + (\sum_{i=1}^{n} |x_i|)(\sum_{i=1}^{n} |y_i|) \varepsilon_{n+1} \end{aligned}$$

QR 分解を用いた解の接続

J_i, $c_i = \mathrm{mid}(J_i)$ に対する平均値形式

$$\phi_{t_i,t_{i+1}}(x) \in \phi_{t_i,t_{i+1}}(c_i) + \phi'_{t_i,t_{i+1}}(J_i)(x - c_i)$$

について、$A_i = \phi'_{t_i,t_{i+1}}(J_i)$, $B_i = \phi_{t_i,t_{i+1}}(c_i)$ と書く。
J_0 を初期値、$c_0 = \mathrm{mid}(J_0)$, $K_0 = J_0 - c_0$, $Q_0 = I$, $i = 0$ とし、

1. J_i を元に A_i, B_i を計算する。
2. $c_{i+1} = \mathrm{mid}(B_i)$
3. A_iQ_i の中心を
$$\mathrm{mid}(A_iQ_i) \simeq QR$$
のように (近似)QR 分解し、得られた Q を Q_{i+1} とする。
4. $K_{i+1} = (Q_{i+1}^{-1}A_iQ_i)K_i + Q_{i+1}^{-1}(B_i - c_i)$
5. $J_{i+1} = Q_{i+1}K_{i+1} + c_{i+1}$
6. $i = i + 1$

を繰り返す。ただし、Q_{i+1}^{-1} は Q_{i+1} の真の逆行列またはそれを含む区間行列でなければならない。

区間演算と Affine Artihmetic と QR 法の比較 (1)

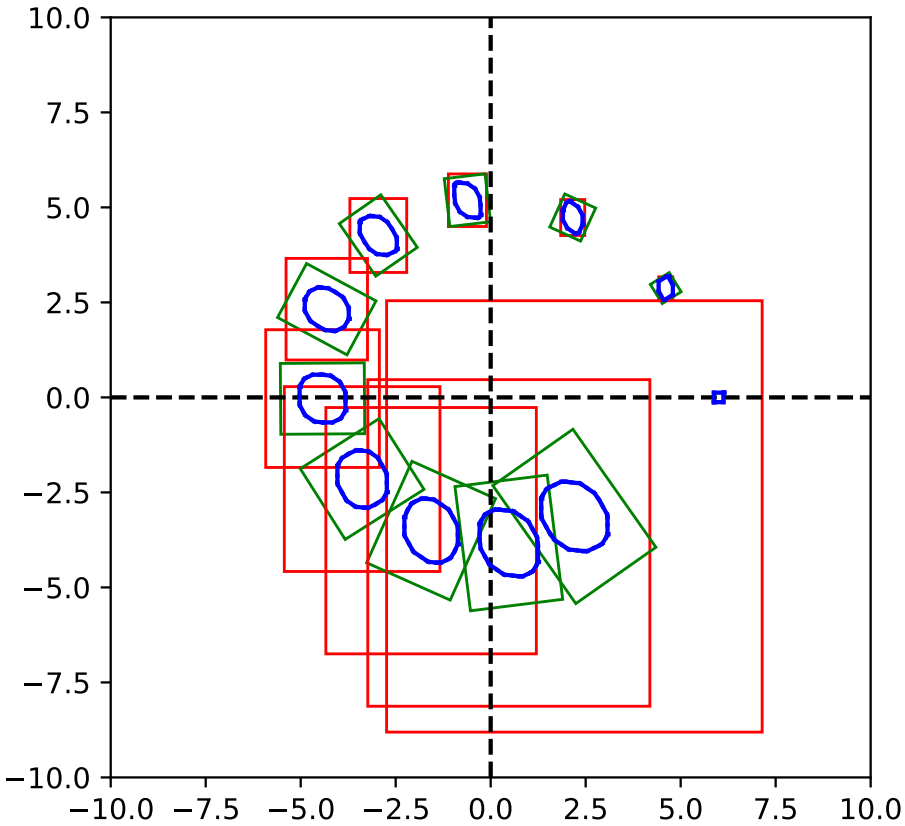
$$I_0 = \begin{pmatrix} [6 - 2^{-3}, 6 + 2^{-3}] \\ [-2^{-3}, 2^{-3}] \end{pmatrix}$$

を初期区間とし、

$$I_{k+1} = \begin{pmatrix} \frac{7}{8}\sin(0.5) & -\cos(0.5) \\ \cos(0.5) & \sin(0.5) \end{pmatrix} I_k + \begin{pmatrix} [-2^{-5}, 2^{-5}] \\ [-2^{-3}, 2^{-3}] \end{pmatrix}$$

を計算させてみた。
線形変換 + ノイズ成分というこの形は、常微分方程式の初期値問題において各ステップ毎のノイズ (= 離散化誤差) の混入が避けられないことをモデル化している。

区間演算と Affine Artihmetic と QR 法の比較 (2)



青が Affine Arithmeric、緑が QR 分解法、赤が区間演算の結果。

ステップ幅 Δt の決定

ε_0 を 1 ステップで混入する誤差の目標値とする。例えば machine epsilon。

1. Type-I PSA を用いて Taylor 展開を計算し、その係数を見て適切なステップ幅 Δt_0 を推定する。Type-I PSA で計算された Taylor 展開を

$$x_0 + x_1 t + x_2 t^2 + \cdots + x_{n-1} t^{n-1} + x_n t^n$$

として、

$$\Delta t_0 = \frac{\varepsilon_0^{\frac{1}{n}}}{\max(|x_{n-1}|^{\frac{1}{n-1}}, |x_n|^{\frac{1}{n}})}$$

とする。
2. ステップ幅 Δt_0 を用いて Type-II PSA を使って候補者集合を作成する。
3. その候補者集合を使ったとき新たに混入する誤差を ε として、新しいステップ幅を

$$\Delta t_1 = \Delta t_0 \left(\frac{\varepsilon_0}{\varepsilon}\right)^{\frac{1}{n}}$$

で推定する。ただし、n は Taylor 展開の次数。
4. ステップ幅 Δt_1 を用いて Type-II PSA を使って再度候補者集合を作成し、解の存在検証を行う。検証に失敗したら例えばステップ幅を半分にする。

kv ライブラリ

- http://verifiedby.me/kv/ で公開中。
- 作成開始は 2007 年秋頃。公開開始は 2013 年 9 月 18 日。最新版は version 0.4.47。
- 言語は C++。boost C++ Libraries (`http://www.boost.org/`) も必要。
- 全てヘッダファイルで記述されており、インストールはヘッダファイルをどこかに配置するだけ。
- オープンソースである。精度保証付き数値計算の結果が「証明」であると主張するならば、計算に使われたプログラムは必ず公開されているべき。
- 計算に使う数値の型は double に制限されていない。C++のテンプレート機能を用いて容易に変更することが出来る。

kv ライブラリで扱える数値型

kv ライブラリで扱える数値型

- double
- 区間演算 (多数の精度保証された数学関数含む)
- 4 倍精度 (double-double) 演算
- MPFR ラッパー
- 複素数演算
- 自動微分
- affine arithmeric
- ベキ級数演算
- と、それらの型の組み合わせ

数値型の組み合わせ

例えば、「autodif<interval<dd>>」は、「自動微分型の内部型として区間型を用い、区間型の内部型は double-double」を意味する。

kv ライブラリで実装されたアプリケーション

kv ライブラリで実装されたアプリケーション

- Krawczyk 法による非線形方程式の精度保証
- 非線形方程式の全解探索
- 常微分方程式の初期値問題の精度保証
- 常微分方程式の境界値問題の精度保証
- 精度保証付き数値積分 (1, 2 次元)
- 端点特異性を持つ関数の精度保証付き数値積分
- ガンマ関数、ベッセル関数などの精度保証付き特殊関数
- KKT 方程式を用いた非線形最適化問題の精度保証
- その他

区間演算 (interval)

- 上端下端型の区間演算を行う。
- exp, log, sin, cos, tan, sinh, cosh, tanh, asin, acos, atan, asinh, acosh, atanh, expm1, log1p, abs, pow などの精度保証付きの数学関数を持つ。
- 10 進文字列と double との丸め方向指定付き相互変換を持ち、正しく入出力が出来る。
- 上端と下端に用いる数値型はテンプレートになっており、double 以外の型も使える。例えば double-double 型や MPFR を使える。ただし、上向き下向き双方の丸めに対応した加減乗除、平方根、文字列との相互変換の方法を定義する必要がある。
- サポートする環境は、C99 準拠の fesetround が使えること。
  - X86 CPU の SSE2
  - 最近点丸めのみを用いた方向付き丸めのエミュレーション
  - 最新の Intel CPU の AVX-512
  - FMA 命令

  を使うオプションもある。

区間演算プログラムの例

```
#include <kv/interval.hpp> // 区間演算
#include <kv/rdouble.hpp> // double の方向付き丸めを定義

int main()
{
    kv::interval<double> s, x;

    std::cout.precision(17);

    s = 0;
    for (int i=1; i<=1000; i++) {
        x = i;
        s += 1/x;
    }

    std::cout << s << "\n";
}
```

```
[7.4854708605 49956,7.4854708605 508238]
```

使い方

解凍

```
$ ls
kv-0.4.47.tar.gz
$ tar xfz kv-0.4.47.tar.gz
$ ls
kv-0.4.47/  kv-0.4.47.tar.gz
$ cd kv-0.4.47
$ ls
LICENSE.txt  README.txt  example  kv  test
```

インストール

必要なのは kv ディレクトリ以下。適当な場所 (例えば `/usr/local/include/`) に配置する。

compile & run

```
$ ls
interval.cc  kv/
$ c++ -I. -O3 interval.cc
$ ./a.out
[7.485470860549956,7.4854708605508238]
```

区間演算プログラム (double-double)

```
#include <kv/interval.hpp> // 区間演算
#include <kv/dd.hpp> // double-double
#include <kv/rdd.hpp> // dd の方向付き丸めを定義

int main()
{
    kv::interval<kv::dd> s, x;

    std::cout.precision(34);

    s = 0;
    for (int i=1; i<=1000; i++) {
        x = i;
        s += 1/x;
    }

    std::cout << s << "\n";
}
```

```
[7.4854708605503449126565182043 08257,7.4854708605503449126565182043 60964]
```

double-double (dd)

- twosum: 2 つの数の和を 2 つの仮数部の重なりのない数の和に変換するアルゴリズム。(例: $1234 + 5.432 \to 1239 + 0.432$)
- twoproduct: 2 つの数の積を 2 つの仮数部の重なりのない数の和に変換するアルゴリズム。(例: $1234 \times 5.432 \to 6703 + 0.088$)
- twosum と twoproduct を組み合わせると、2 つの倍精度数を用いた擬似的な 4 倍精度計算を実現できる。一般的な多倍長演算よりかなり高速。
- 単体 (`dd.hpp`) で使った時は近似計算。
- `dd.hpp` と `rdd.hpp`(方向付き丸めでの `dd` 型の演算を定義) を併用し、`interval` 型の内部型として `dd` 型を使うと、端点に `dd` 型を持つ 4 倍精度区間演算が可能。

mpfr

- 高精度浮動小数点計算が行える有名な MPFR ライブラリの簡単な wrapper。
- 単体 (`mpfr.hpp`) で使った時は近似計算。
- `kv::mpfr<106>`のようにパラメータとして仮数部長を指定して使う。
- `mpfr.hpp` と `rmpfr.hpp` を併用し、`interval` 型の内部型として `mpfr` 型を使うと、端点に `mpfr` 型を持つ区間演算が可能。但し、MPFR の機能を使うのは加減乗除と平方根のみであり、せっかく MPFR が持っている優秀な数学関数群は一切用いられない。

自動微分 (autodif)

- Bottom-Up 型の自動微分を実装している。一階微分のみ。(一変数関数であれば、ベキ級数型 (psa) で高階微分を行える。)

```
#include <kv/autodif.hpp>
namespace ub = boost::numeric::ublas;
// 関数の定義
template <class T> ub::vector<T> func(const
      ub::vector<T>& x) {
    ub::vector<T> y(2);
    y(0) = 2. * x(0) * x(0) * x(1) - 1.;
    y(1) = x(0) + 0.5 * x(1) * x(1) - 2.;
    return y;
}
int main()
{
    ub::vector<double> v1, v2;
    ub::vector< kv::autodif<double> > va1,
        va2;
    ub::matrix<double> m;
    v1.resize(2);
    v1(0) = 5.; v1(1) = 6.;
    // 自動微分型の初期化
    va1 = kv::autodif<double>::init(v1);
    // 関数呼び出し
    va2 = func(va1);
    // 自動微分型を分解
    kv::autodif<double>::split(va2, v2, m);
    // f(5, 6)
    std::cout << v2 << "\n";
    // Jacobian matrix at (5, 6)
    std::cout << m << "\n";
}
```

```
[2](299,21)
[2,2]((120,50),(1,6))
```

常微分方程式の初期値問題の例 (1/4)

van del Pol 方程式

$$\frac{d^2x}{dt^2} - \mu(1-x^2)\frac{dx}{dt} + x = 0$$

一階に直す

$$\frac{dx}{dt} = y$$
$$\frac{dy}{dt} = \mu(1-x^2)y - x$$

常微分方程式の初期値問題の例 (2/4)

```
#include <kv/ode-maffine.hpp>
namespace ub = boost::numeric::ublas;
typedef kv::interval<double> itv;
class VDP { // 解きたい問題の右辺を関数オブジェクトで記述
    public:
    template <class T> ub::vector<T> operator() (const ub::vector<T>& x, T t){
        ub::vector<T> y(2);
        y(0) = x(1);
        y(1) = 1. * (1. - x(0)*x(0)) * x(1) - x(0);
        return y;
    }
};

int main()
{
    ub::vector<itv> x;
    itv end;
    std::cout.precision(17);
    x.resize(2);
    x(0) = 1.; // 初期値
    x(1) = 1;
    end = 100.; // 終了時刻
    kv::odelong_maffine(VDP(), x, itv(0.), end); // 初期値問題を解く(0-end)
    std::cout << x << "\n";
}
```

```
[2]([2.007790480952114,2.007790480952139],[-0.056051438751153989,-0.056051438750559116])
```

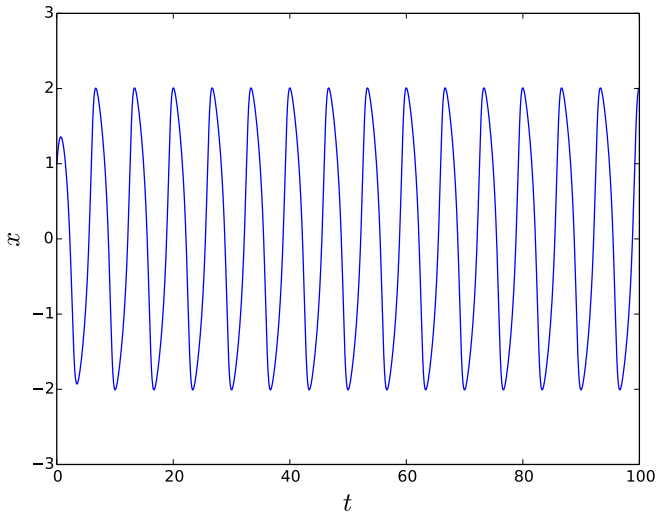
常微分方程式の初期値問題の例 (3/4) (途中経過の表示)

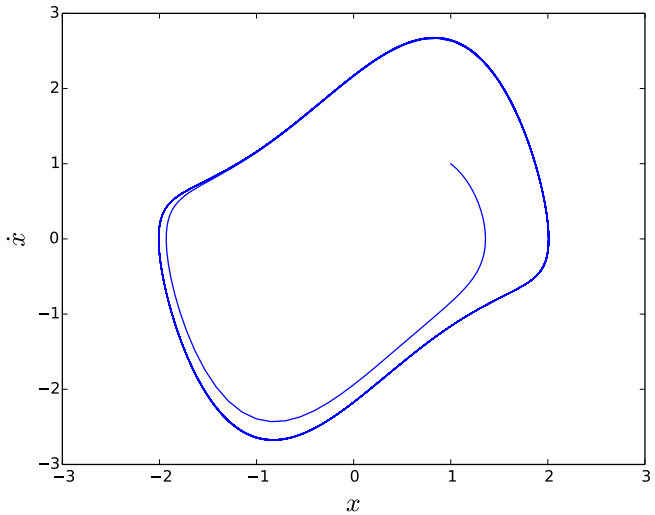
2^{-4} 間隔で密出力させた例

```
#include <kv/ode-maffine.hpp>
#include <kv/ode-callback.hpp>
namespace ub = boost::numeric::ublas;
typedef kv::interval<double> itv;
class VDP {
    public:
    template <class T> ub::vector<T> operator() (const ub::vector<T>& x, T t){
        ub::vector<T> y(2);
        y(0) = x(1);
        y(1) = 1. * (1. - x(0)*x(0)) * x(1) - x(0);
        return y;
    }
};

int main()
{
    ub::vector<itv> x;
    itv end;
    std::cout.precision(17);
    x.resize(2);
    x(0) = 1.;
    x(1) = 1;
    end = 100.;
    kv::odelong_maffine(VDP(), x, itv(0.), end, kv::ode_param<double>(), kv::
        ode_callback_dense_print<double>(itv(0.), itv(pow(2., -4))));
}
```

常微分方程式の初期値問題の例 (4/4) (途中経過の表示)

```
t: [-0,0]
[2]([1,1],[1,1])
t: [0.0625,0.0625]
[2]([1.0604282381493324,1.0604282381493327],[0.93186430539999509,0.93186430539999521])
t: [0.125,0.125]
[2]([1.1162696582692208,1.1162696582692211],[0.8534995323034189,0.85349953230341902])
t: [0.1875,0.1875]
[2]([1.1669406889500346,1.1669406889500352],[0.76674349796008578,0.7667434979600859])
t: [0.25,0.25]
[2]([1.2119811457513768,1.2119811457513768],[0.67368071112755956,0.6736807111275599])
----------omitted----------
t: [99.9375,99.9375]
[2]([2.0074651477352984,2.0074651477354078],[0.07045240241356479,0.07045240241453340 5])
t: [100,100]
[2]([2.0077904809520377,2.007790480952215],[-0.0560514387514576,-0.05605143875025545])
```





kv 以外の常微分方程式の精度保証パッケージ

- Lohner による AWA
- Intlab 11 の AWA toolbox
- Intlab 11 の Taylor model toolbox

を紹介する。

Lohner の AWA (1)

- Rudolf J. Lohner 氏による、初期値問題の精度保証を行う、非常に先駆的で優秀なアルゴリズム。

R. J. Lohner : "Enclosing the Solutions of Ordinary Initial and Boundary Value Problems", In E. Kaucher, U. Kulisch and Ch. Ullrich (eds.) : "Computer Arithmetic, Scientific Computation and Programming Languages", B. G. Teubner, Stuttgart, pp.255–286 (1987).

に載っているらしいが、見たことはない。

- 非常に早い時期に高速な実装を行い、後の研究者の一つの到達目標になった。
- AWA = ドイツ語の anfangswertaufgabe (日本語で初期値問題)。
- 古いソフトウェアだが、今でも動く。

Lohner の AWA (2)

- AWA は Pascal-XSC という精度保証付き数値計算専用に開発された Pascal 言語で記述されている。コンパイル済みの AWA は MS-DOS 用しかなく、さすがに現在での利用は現実的ではない。
- 幸いなことに、Pascal-XSC は今でも、
  - XSC Languages (http://www2.math.uni-wuppertal.de/ xsc/)

  で公開されており、また幸いなことに Pascal-XSC のソースコードは内部に AWA 一式を含んでいる。
- 上のページからリンクを辿って p-xsc-3.6.2.tar.gz (released 2005-12-19) をダウンロードし make すれば、Pascal-XSC のコンパイラとともにコンパイル済みの awa の実行ファイルが手に入る。
- 詳しくは、Lohner の AWA で遊ぼう に書いた。

Intlab 11 の AWA toolbox と Taylor model toolbox

- INTLAB は説明するまでもないだろう。Rump 先生が作られた MATLAB 上で動く精度保証付き数値計算ソフトウェア。
- 2018 年 9 月のアップデート (Intlab 11) で、Florian Bünger 先生が開発した、常微分方程式の初期値問題を精度保証付きで解く AWA toolbox と、Taylor model toolbox が追加された。
- AWA toolbox は、Lohner の AWA を Intlab で再現したもの。
- Taylor model toolbox は、Berz, Makino の Taylor Model を用いた精度保証付き初期値問題ソルバー COSY INFINITY を Intlab で再現したもの。
- Taylor Model は、多変数関数の Taylor 展開を扱うもの。例えば常微分方程式 $\dot{x} = f(x,t)$, $f(t_0) = x_0$ において、解を t について高次に展開するのみならず初期値 x_0 についても高次に展開することによって、非常に高性能な精度保証付きアルゴリズムが得られることが知られている。
- 詳しくは、Intlab 11 の精度保証付き ODE Solver に書いた。

実際のプログラム (intlab-awa)

vdp_awa.m

```
init = [1; 1];
tic; [T,X] = awa(@vdp, @vdp_J,  [0, 20], init, awaset('order', 24)); toc
format long e
awa_disp(T,X);

function f = vdp(t, x)
    mu = 1;
    f =     [x(2);
        mu*(1-x(1)^2)*x(2)-x(1)];
end

function J = vdp_J(t, x)
    mu = 1;
    J =     [typeadjust(0, x), typeadjust(1, x);
        -2*mu*x(1)*x(2) - 1, mu*(1-x(1)^2)];
end
```

ODE の右辺の Jacobi 行列を手で書く必要があるのが残念。Bünger 先生によると、高速化のためらしい。

実際のプログラム (intlab-taylor)

vdp_taylor.m

```
init = [1; 1];
tic; [T,X,Xr] = verifyode(@vdp, [0, 20], init, verifyodeset('order', 18)); toc;
format long e
verifyode_disp(T,X,[],init);

function f = vdp(t, x, i)
    mu = 1;
    if nargin == 2 || isempty(i)
        f =     [x(2);
            mu*(1-x(1)^2)*x(2)-x(1)];
    else
        switch i
        case 1
            f =     x(2);
        case 2
            f = mu*(1-x(1)^2)*x(2)-x(1);
        end
    end
end
```

こちらは Jacobi 行列は要らないが、int の引数で番号を指定されたら n 本の ODE の右辺のうちの 1 本だけを返すように書く必要がある。これも高速化のためらしい。

実際のプログラム (awa)

vdp.in

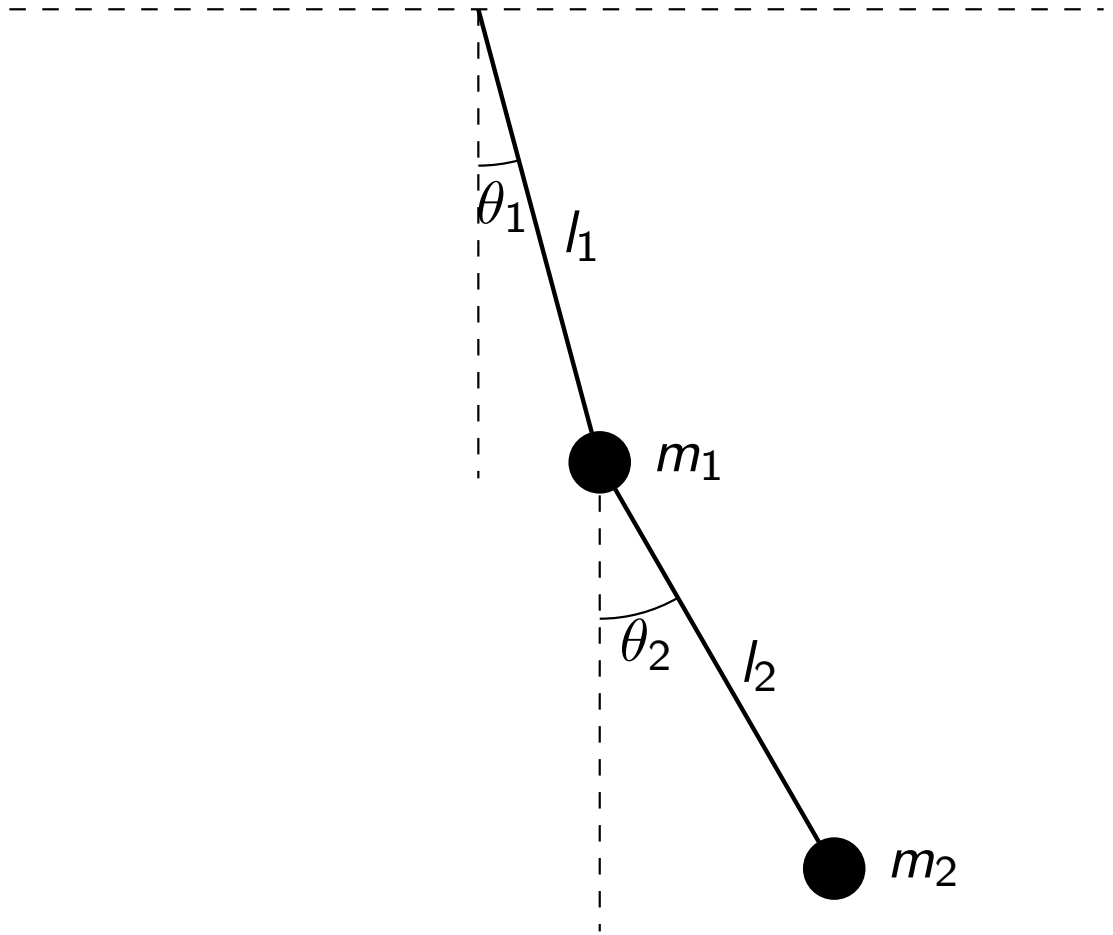
```
2
vdp.dat
vdp.out
```

vdp.dat

```
F1 = U2
F2 = 1 * (1-U1*U1) * U2 - U1
;;

1 0 20 0 24

4 0

1 1

1e-16 1e-16
```

```
./awa < vdp.in
```

二重振り子の精度保証付き数値計算 (1/5)



(ブログ記事) 二重振り子の精度保証付き数値計算

二重振り子の精度保証付き数値計算 (2/5)

使った方程式

$$(m_1+m_2)l_1\ddot{\theta_1}+m_2l_2\ddot{\theta_2}\cos(\theta_1-\theta_2)+m_2l_2\dot{\theta_2}^2\sin(\theta_1-\theta_2)+(m_1+m_2)g\sin\theta_1=0$$

$$l_1l_2\ddot{\theta_1}\cos(\theta_1-\theta_2)+l_2^2\ddot{\theta_2}-l_1l_2\dot{\theta_1}^2\sin(\theta_1-\theta_2)+gl_2\sin\theta_2=0$$

$\ddot{\theta_1},\ddot{\theta_2}$ に関する連立方程式と見て解き、$\dot{\theta_1},\dot{\theta_2}$ に別名を付けて 1 階 4 変数とする。

パラメータ

$$m_1=m_2=1$$
$$l_1=l_2=1$$
$$g=9.8$$

初期値

$$\theta_1(0)=\frac{3}{4}\pi$$
$$\theta_2(0)=\frac{3}{4}\pi$$
$$\dot{\theta_1}(0)=0$$
$$\dot{\theta_2}(0)=0$$

二重振り子の精度保証付き数値計算 (3/5)

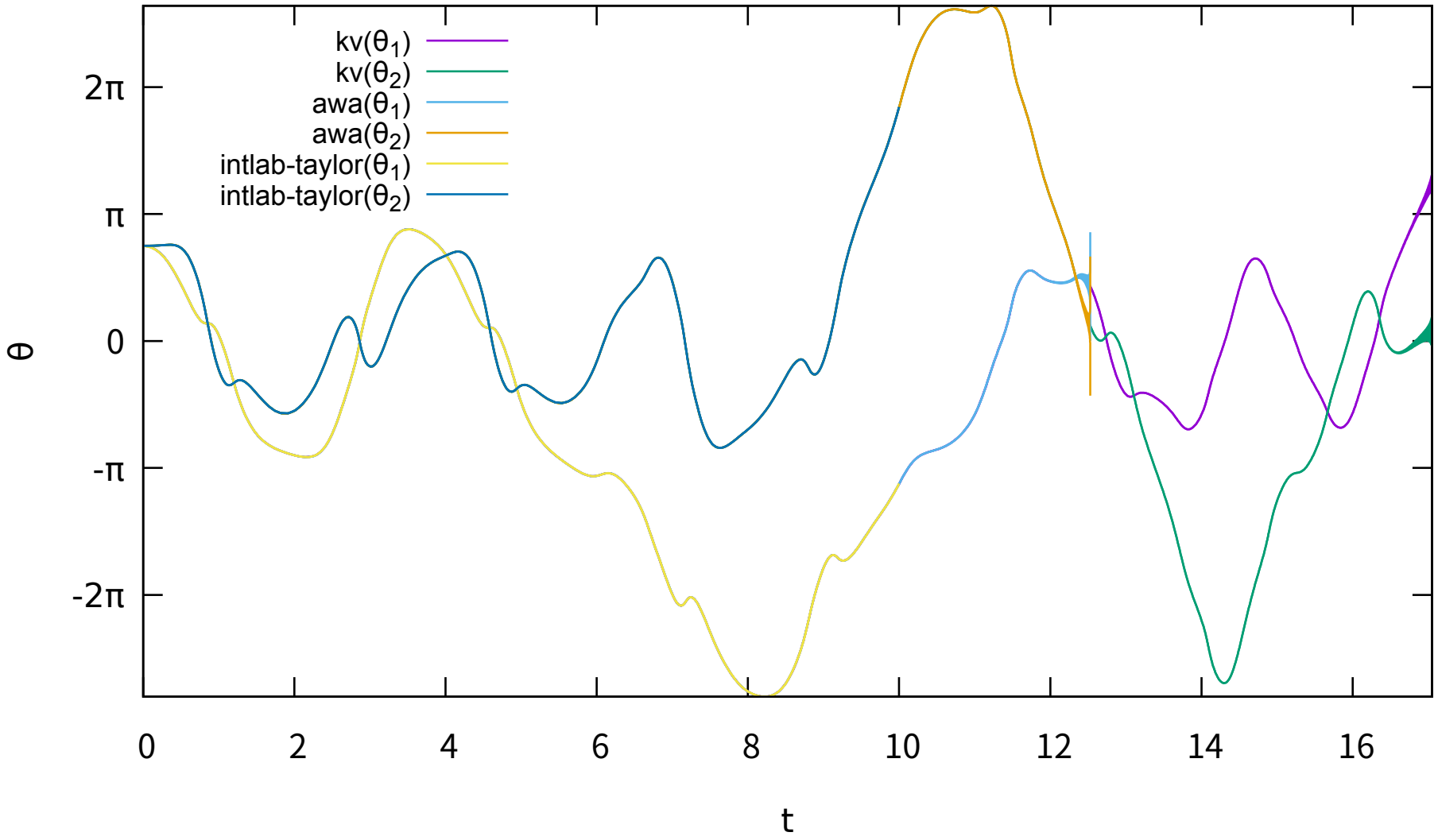
solver のパラメータ

- (kv) order=24, localerr = 2^{-52}
- (AWA) order=24, localerr = 10^{-16}
- (Taylor model toolbox) order=24, localerr=10^{-10}, 'shrinkwrap'=0, 'precondition'=1, 'blunting'=0

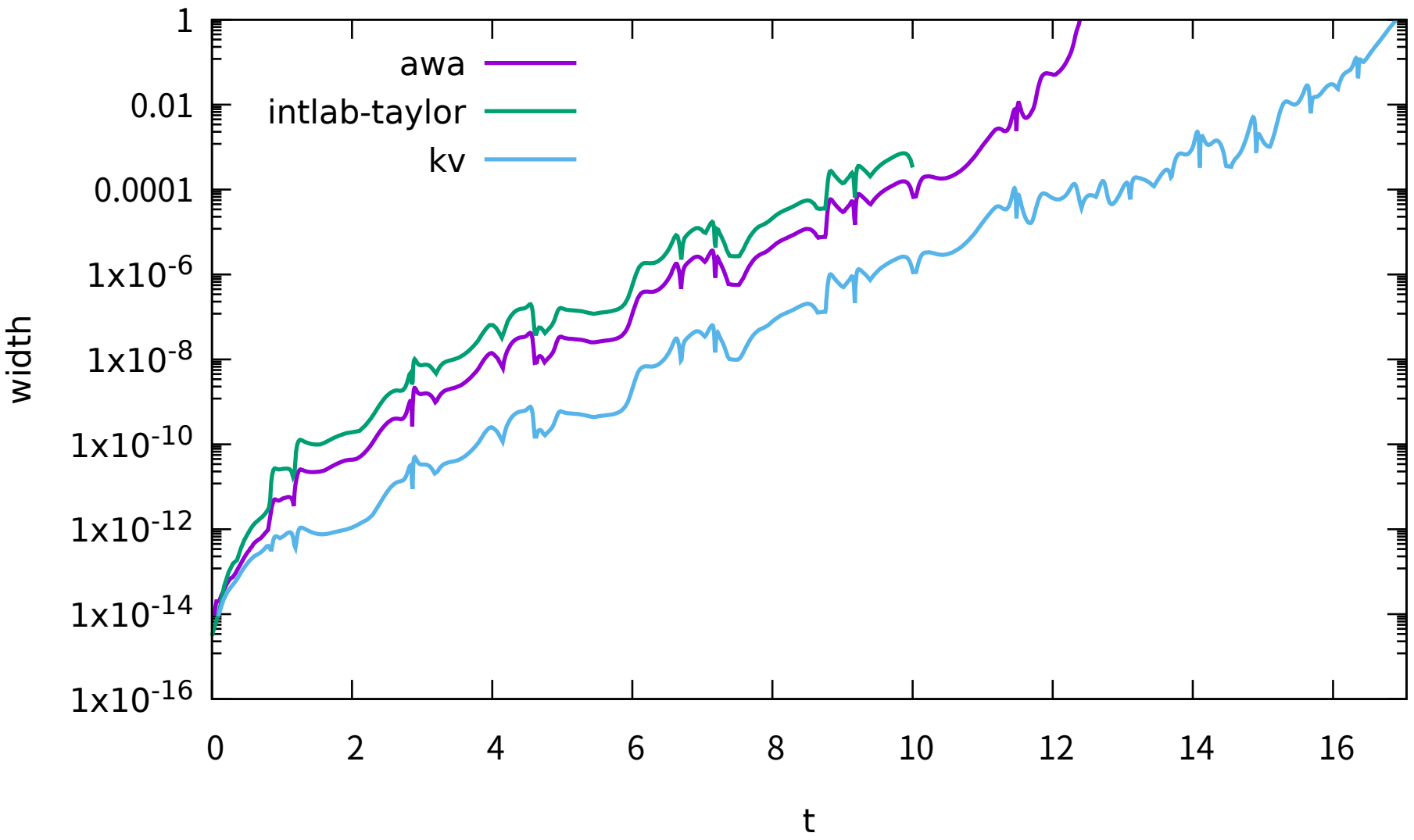
計算時間

- (kv) 22.9sec
- (AWA) 17.9sec
- (Taylor model toolbox) 5h20min17sec

二重振り子の精度保証付き数値計算 (4/5)



二重振り子の精度保証付き数値計算 (5/5)



まとめ

- 常微分方程式の精度保証付き数値計算
- ベキ級数演算 (PSA)
- Lohner 法
- Affine Arithmeric と QR 分解法
- kv ライブラリの紹介
- 既存の実装の紹介
- AWA, Intlab, kv の比較

精度保証付き数値計算の理論もソフトウェアも、大勢の人に利用されることで鍛えられます。皆様のご利用をお待ちしております!