平成11年度

卒業論文

ベキ級数演算を用いた常微分方程式の長時間積分

Long Time Integration for Ordinary Differential Equations
Using Power Series Arithmetic

平成12年2月4日

指導教授： 柏木 雅英 助教授

早稲田大学理工学部情報学科

学籍番号 ： G96P088-9

長友 泰崇

目 次

図 目 次

表 目 次

第 1 章

序論

1.1 背景

理工学の分野において、非線形現象の解析は古くから研究の対象とされてきた。一般に、非線形現象の解析は、実際の現象を微分方程式や有次元非線形方程式といった数学的なモデルに置き換え、それを解くことによって行われる。しかし、この微分方程式や非線形方程式の解が、解析的に厳密に求められることは稀である。通常は計算機による数値計算を行なうことによって解が求められる。つまり、数値計算の技術は非線形現象の解析には必要不可欠である。実際、近年の計算機能力の大幅の向上に伴って、数値計算の技術も発展し、大規模、高速、高精度な計算が行えるようになり、より複雑な問題に対して多くの有益な結果が得られるようになっている。

しかし、数値計算を利用して何らかの結果を得ようとした場合、その計算過程において様々な誤差が生じる。解析する非線形現象を微分方程式等の数学的なモデルに置き換えるときに生じる『モデル化誤差』、つぎに得られた式、例えば微分方程式を解く際、有限次元の連立方程式に変形した場合に『離散化誤差』や『打ちきり誤差』が生じ、さらに有限次元方程式を実際に計算機で解く際には『丸め誤差』が生じる。このように、得られた計算結果に対しての信頼性については何の保証もされていないことは以前から指摘されてきた。そこで、数値計算を近似計算のみにとどめておくのではなく、計算結果の信頼性を与えるという意味で、数学的に正しいことが保証された結論を導くための理論と技術が進められてきた。このような数値計算は” 精度保証付き数値計算” とよばれており、今後の数値計算のあるべき姿として注目を集めている。

” 精度保証付き数値計算” の手法は、現在、既に線形連立方程式や、常微分方程式 (と一部の偏微分方程式) に対して確立されている。中でも微分方程式を解く場合、有限次元の連立方程式に変形する段階で離散化誤差が生じる分だけ、線形連立方程式と比べて精度保証付きで解くことが困難となる。

上に述べたうち、『丸め誤差』の厳密な評価を行う代表的な手法が区間演算である。詳しくは第 2 章で述べるが、この区間演算は全ての数値を点としてではなく、その値を含む下限と上限を持つような区間として表現し、区間同士の四則演算や、引数として区間を取る

ような初等関数のアルゴリズムにより、計算結果として得られる区間が厳密な計算結果の真の値を含むようにすることができる。

この区間演算を用いることにより、決定的なプロセスによって求められるような数値計算の結果が精度保証される。線形連立方程式の場合にはGaussの消去法などの決定的なプロセスにより解くことができるので、その解を精度保証付きで求めることができる。しかし、非線形方程式の場合、一般に決定的なプロセスが存在しないので、これらの解はNewton法等の反復法による近似解法が広く用いられている。しかし、この計算過程に区間演算を適用しても反復後の近似解を含む区間が得られるだけで、真の解との誤差は評価できない。

非線形方程式の場合にはそもそも解が存在するかどうかすらわからないので、まずは解の存在を保証する必要がある。即ち、解が存在するための十分条件を与える必要がある。Newton法が収束するための十分条件 (Newton法の収束定理) としてはKantorovichの定理や占部の定理が知られており、それらはいずれも関数解析の分野のいわゆる不動点定理によって証明される。即ち、与えられた方程式をこれと同値な不動点問題に置き換え、区間演算を用いて不動点定理の成立条件を数値的に評価することによって解の存在の保証と誤差評価を行なうことができる。このように不動点定理をうまく利用することによって、非線形方程式の解の精度保証を行なうことが可能となる。

1.2 本論文の目的

本論文は、前節で述べた背景を踏まえて、次のような正規形の連立一階常微分方程式の初期値問題に対する精度保証付き数値計算についての研究をまとめたものである。

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = f(x(t), t), \quad t \in [a, b] \tag{1.1}$$

ここで、$x(t)$ は n 次元ベクトル値関数である。

ところで、常微分方程式を解くとはどういうことであるかというと、解析的には式(1.1)を区間 $t \in [a, b]$ において恒等的に満たす $x(t) = (t\text{の式})$ を求めることである。しかし、一般にはそのようなことができないため、数値計算の立場から常微分方程式を解くとは、解

を求めることが要求されている区間に等間隔の基点をとり、式

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = f(x(t), t) \tag{1.2}$$

を満たす関数 $x(t)$ のこれらの基点における値を求めることである。

本論文では、区間を $[a, b]$ とし、基点を $(a =)t_1 < t_2 < \cdots < t_m (= b)$ とする。よって定義域を $a = t_1 < t_2 < \cdots < t_m = b$ のように分割・離散化したときの、$x(t_2), x(t_3), \cdots, x(t_m)$ の値を求めることが本論文の目的となる。

長時間積分とは式 (1.1) においてとくに $a \ll b$ のときに $x(t_2), x(t_3), \cdots, x(t_m)$ のようにそれぞれ t における常微分方程式の数値解を逐次求めていくことである。ここで『積分』という言葉を使うのは、『微分方程式を解く』ことを『積分する』と言うからにほかならない。

本論文の基礎となる文献 [1] の方法を用いると、$x(t_i)$ が既知のときに $x(t_{i+1})$ の値を精度保証することが可能となる。文献 [1] の方法を、第 2 章で述べる平均値形式と組み合わせることによって、区間幅の広がりを抑えながら $x(b)$ の値を精度保証付きで求めたのが本論文の手法である。

また、後述する Wrapping Effect を考慮することによってさらに区間幅が抑えられることも検証していく。

1.3 本論文の構成

本論文の構成を述べる。

第 2 章では、『準備』と題してまず本論文の基礎となる区間解析について説明する。次に文献 [1] の方法で用いる 2 種類のベキ級数演算の定義を紹介する。さらに、多項式の評価を行う方法としてホーナー法を説明する。

第 3 章では、ベキ級数演算を常微分方程式の初期値問題の長時間積分に適用するための準備として、文献 [1] の方法について述べる。

第 4 章では、実際に長時間積分を行う手法を述べる。

第 5 章では、例題に対して実際に計算を行い、本手法の有効性について検証する。

第 6 章では、「まとめ」として、数値実験の結果について考察する。

第 7 章では、謝辞を述べる。

第 2 章

準備

2.1 区間解析

2.1.1 はじめに

数値計算において、計算を行うと同時にその結果の誤差評価をも同時に計算するような方法を総称して精度保証付き数値計算と呼び、近年急速な進歩を遂げている。精度保証付き数値計算の実現において最も基本的かつ重要な技法に、区間演算が挙げられる。区間演算とは、実数値 [下限, 上限] という 2 つの浮動小数点数で挟まれた区間で表現し、その区間同士の加減乗除等の演算を「演算結果としてあり得る集合を包含するように」定義することにより行われるものである。

2.1.2 計算機における数値の表現

数値計算で用いられる数値 (C 言語でいうところの double や float) は、いわゆる**浮動小数点形式**で表されている。かつては CPU 毎に様々な方式が乱立していたが、IEEE 754Std. の提案以降、この方式に統一されつつある。ここでは、この方式における数値の表現方法を示しておく。

実数 x を 2 進数で表現したとき、0 である場合を除いて、次のように正規化することができる。

$$x = 1.x_1x_2x_3\cdots \times 2^n$$

倍精度浮動小数点数 (C 言語における double) は、64bit で表され、その内訳は

符号 s (1bit)	指数部 e (11bit)	仮数部 m (52bit)

のとおりである。

s は符号 bit で、0 なら正、1 なら負である。

e は符号無し整数とみて 0 から 2047 までの値をとるが、通常の数 (正規化数) では両端の値を除いた $1 \leq e \leq 2046$ であり、

$$x = \pm 1.m \times 2^{e-1023}$$

のように実数と対応する。このように、正規化された数は必ず先頭は 1 なのでそれはメモリには格納せず、52bit の領域で 53bit の精度を達成する。

この方法で表現できる絶対値最大の数は

$$1.111\cdots\times 2^{2046-1023}=2^{1024}-2^{971}\simeq 10^{308.25}$$

である。

また、この方法で表現できる最大値最小の数は

$$1.000\cdots\times 2^{1-1023}=2^{-1022}\simeq 10^{-307.65}$$

であるが、それより絶対値の小さい数は「非正規化数」として格納する。このときは $e=0$ とし、

$$x=\pm 0.m\times 2^{-1022}$$

のように実数と対応する。これによって、絶対値が $2^{-1074}\simeq 10^{-323.3}$ から $2^{-1022}-2^{-1074}$ の間の数が表現できるが、非正規化数の場合は、精度は 53bit より少ないことに注意する必要がある。

0 は $e=m=0$ で表現する。

$e=2047$ は、$\pm\infty(m=0$ のとき) や NaN などの特殊な数を表現するのに使用される。

倍精度浮動小数点数の精度は、1 より大きな最小の浮動小数点数と 1 との差 (いわゆるマシンイプシロン) で見ると、$2^{-52}\simeq 10^{-15.65}$ であり、およそ 15 桁の精度があると考えられる。ただしこれは数値が正規化数として表されている場合に限られる。

単精度浮動小数点数 (C 言語における float) の場合は、全体の長さが 64bit$\rightarrow$ 32bit、指数部が 11bit$\rightarrow$ 7bit、仮数部が 52bit $\rightarrow$ 24bit, 指数部のバイアス値が $1023\rightarrow 127$ と変わっているだけで、基本的には同様である。

プロセッサによっては、倍精度浮動小数点数よりも高精度の拡張倍精度浮動小数点数を備える場合もあるが、これの表現形式は規定されておらず、プロセッサによって異なる。

倍精度浮動小数点数を用いた場合、実数のうちで計算機で正確に表現できる数は有限の高々2^{64} 個に過ぎない (実際はもう少し少ない)。よって、例えば浮動小数点を 2 つ加算した

結果は、浮動小数点数で表せるとは限らず、それに近い浮動小数点数で近似することになる。これによって発生する誤差を、**丸め誤差**という。通常は最も近い浮動小数点数に丸めるが、IEEE 754 Std. に従う CPU では丸めの方法は変更可能である。

言うまでもないことだが、倍精度浮動小数点数の精度が 10 進数で 15 桁あるからと言って、計算結果が 15 桁正しいとは限らない。計算式と数値によってはわずか数ステップの計算で有効数字が完全に失われることもめずらしくない。

2.1.3 区間演算

区間演算は、浮動小数点演算における丸め誤差の把握を行なうための基本的な手法である。区間演算では、実数値を [下限, 上限] という 2 つの浮動小数点数で挟まれた区間で表現し、その区間同士の加減乗除の演算を「演算結果としてあり得る集合を包含するように」定義することにより行われる。例えば、区間 $X = [a, b], Y = [c, d]$ に対する減算 $X - Y$ は明らかにとり得る最小値は $a - d$、最大値は $b - c$ であるから、

$$X - Y = [a - d, b - c]$$

のように行う。いくつかの演算に対する区間演算をまとめておく。

$$\begin{aligned}
X + Y &= [a + c, b + d] \\
X - Y &= [a - d, b - c] \\
X \times Y &= [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)] \\
X/Y &= X \times [\frac{1}{d}, \frac{1}{c}] \quad (0 \notin [c, d]) \\
\exp(X) &= [\exp(a), \exp(b)]
\end{aligned}$$

丸め誤差を伴う浮動小数点演算だけしか利用できないという仮定の下で数値計算の厳密な誤差評価を行う場合、ある計算の誤差の計算式が仮にあったとしても、その式を計算するときの誤差は誰が計算するのかという問題に直面することになる。

区間演算においても、両端の数を浮動小数点演算で計算せざるを得ない以上、この問題が生じる。区間演算では、区間の両端を計算する際に丸めの向きを「外向き」にしておくことによって丸め誤差の影響分を区間内に収めるという方法でこの問題を解決している。すなわち、区間の下限を計算するときには「切り捨て」、区間の上限を計算するときには「切り上げ」で計算を行う。こうすれば、厳密計算に比べて多少区間幅が大きくなるものの、区間内に真値を包含しながら計算するという区間演算の性質は損なわれない。

例 3.1 簡単な例として、$(1/3) \times 3$ を 10 進数で 3 桁の有効数字をもつシステムで計算することを考えよう。普通に計算すると、

$$\begin{aligned} 1/3 &= 0.333 \\ 0.333 \times 3 &= 0.999 \end{aligned}$$

となる。区間演算の場合は、最初に 1 と 3 を $[1,1]$、$[3,3]$ のように幅 0 の区間としておき、計算を行う。

$$[1,1]/[3,3] = [0.333, 0.334]$$

$$[0.333, 0.334] \times [3,3] = [0.999, 1.01]$$

このように計算が進行すると、外向きの丸めのため数値は真値を含む微小区間となる。

このように、浮動小数点演算を用いた区間演算を実現するためには、確実に切り捨て (または切り上げ) で計算を行う必要がある。IEEE 754 Std. に従って設計された CPU では、加減乗除及び開閉おける丸めの発生の仕方を制御することができる。普段は真値に最も近い浮動小数点数に丸めるモードになっているが、これを「常に切り上げ」や「常に切り捨て」を行うように変更することができる。これを用いて、

1. 丸めモードを切り捨てに変更
2. 区間の下限を計算
3. 丸めモードを切り上げに変更
4. 区間の上限を計算

5.(必要なら丸めモードを通常の状態に戻す。)

という手順で計算を行えばよい。ただし、丸めモードの変更が有効なのは加減乗除及び開閉だけであり、exp などの超越関数の計算においては、

- Taylor 展開などを用いて自力で計算し、厳密に誤差評価する。
- 使用 CPU の超越関数の誤差限界が分かる場合は、計算後に誤差分を補正する (加減の計算時は引き、上限の計算時は足す)。

などの方法を採らなければならない。

例 3.1 のように、幅 0 の区間から出発して区間演算を行うと丸め誤差の把握が可能である。一方、はじめから幅を持った区間から出発して区間演算を行えば、それは関数の値域を評価することになるのは区間演算の定義から自明であろう。すなわち、関数

$$f(x), \quad f : \mathbf{R^n} \to \mathbf{R^m}$$

に対して、$I \in \mathbf{IR^n}$($\mathbf{IR}$ は実数区間の集合、$\mathbf{IR} = \{[a,b] | a \leq b, a, b \in \mathbf{R}\}$) から出発して区間演算を行った結果を $F(I) \in \mathbf{IR^m}$ とすると、

$$F(I) \supset \{f(x) | x \in I\}$$

を満たす。例えば、$f(x) = x^2 + 2x$ に対して、区間 $I = [0,1]$ として区間演算を行うと $[0,3]$ を得るが、$[0,3] \supset \{f(x) | x \in [0,1]\}$ である。

一般に区間を受け取り、区間を返すような関数を**区間写像**という。また、f に対して上の性質を満たすように作られた**区間写像** $F : \mathbf{IR^n} \to \mathbf{IR^m}$ を、f の**区間包囲**と言う。すなわち、関数の計算式が与えられている場合にその計算手順を区間演算に置き換えることは、関数の区間包囲を得る一つの手段である。

このように、区間演算には、

- 丸め誤差の評価

- 関数の値域の評価

という 2 つの役割がある。前者は無論、精度保証付き数値計算の基礎となるものだが、方程式の解の精度保証などにおいてはむしろ後者が重要な役割を果たす。

2.1.4 区間演算の問題点

区間演算の問題点の一つとして、確かに計算された区間は真の値を含むものの、区間幅が極端に広くなってしまうことが多いて点が挙げられる。これは特に関数の値域を評価するなど、比較的幅の広い区間を扱った場合に顕在化する。例として、関数

$$f(x) = x^2 - 2x$$

の値域を、区間 $[0.9, 1.1]$ で評価する問題を考えよう。区間演算の定義に従って計算すると、

$$\begin{aligned} & [0.9, 1.1]^2 - 2 \times [0.9, 1.1] \\ = \ & [0.81, 1.21] - [1.8, 2.2] \\ = \ & [-1.39, -0.59] \end{aligned}$$

のようになるが、真の像は、

$$\{f(x) | x \in [0.9, 1.1]\} = [-1, -0.99]$$

であり、実に 80 倍もの区間幅の開きがあることが分かる。この現象は、二つの関数 x^2 と $2x$ が同じ x の関数で互いに相関があるにもかかわらず、区間演算における減算ではそれを無視して独立な値として計算を行ってしまうことに原因がある。

なお、上の式は、$x^2 - 2x = x(x-2)$ と変型して評価すれば、$[-1.21, -0.81]$ といくらか改善されるし、$x^2 - 2x = (x-1)^2 - 1$ とすれば、真の像が得られる。この例で分かるように、区間演算においては数学的に同値な式でも計算の順序で大きく結果が異なる場合があり、注意が必要である。一般に、どのような式を変型すれば良い結果が得られるかは極めて難しい問題であり、問題を多項式に限っても最適な方法は知られれていない。

また、同様に変数間の相関を表現できないことに起因する問題に、**Wrapping Effect** と呼ばれる現象が知られている。n 次元ユークリッド空間 $\mathbf{R^n}(n > 1)$ の部分集合を区間で表す場合、区間ベクトルと呼ばれる成分が区間であるようなもので表現するが、これでは $\mathbf{R^n}$ の超直方体領域しか表現することができない。別の言葉で言えば、成分間の相関を表現できない。これによって、区間幅の増大が発生してしまうことがある。例えば、

$$A = \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}, B = \frac{1}{2}\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}, I = \begin{pmatrix} [\text{-}1,1] \\ [\text{-}1,1] \end{pmatrix}$$

のとき、$(AB)I$ と $A(BI)$ を区間演算で計算すると、

$$(AB)I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} I = \begin{pmatrix} [\text{-}1,1] \\ [\text{-}1,1] \end{pmatrix}$$

$$A(BI) = A\begin{pmatrix} [\text{-}1,1] \\ [\text{-}1,1] \end{pmatrix} = \begin{pmatrix} [\text{-}2,2] \\ [\text{-}2,2] \end{pmatrix}$$

のように、結果が大きく異なる。これは、I に B を掛けた結果の真の像は超直方体領域でないので区間演算の結果としてはそれを包む (wrap する) 超直方体となってしまい、領域の増大が発生し、またそれに A を掛けたときにも同様の現象が発生していることによる (図 2.1 参照)。

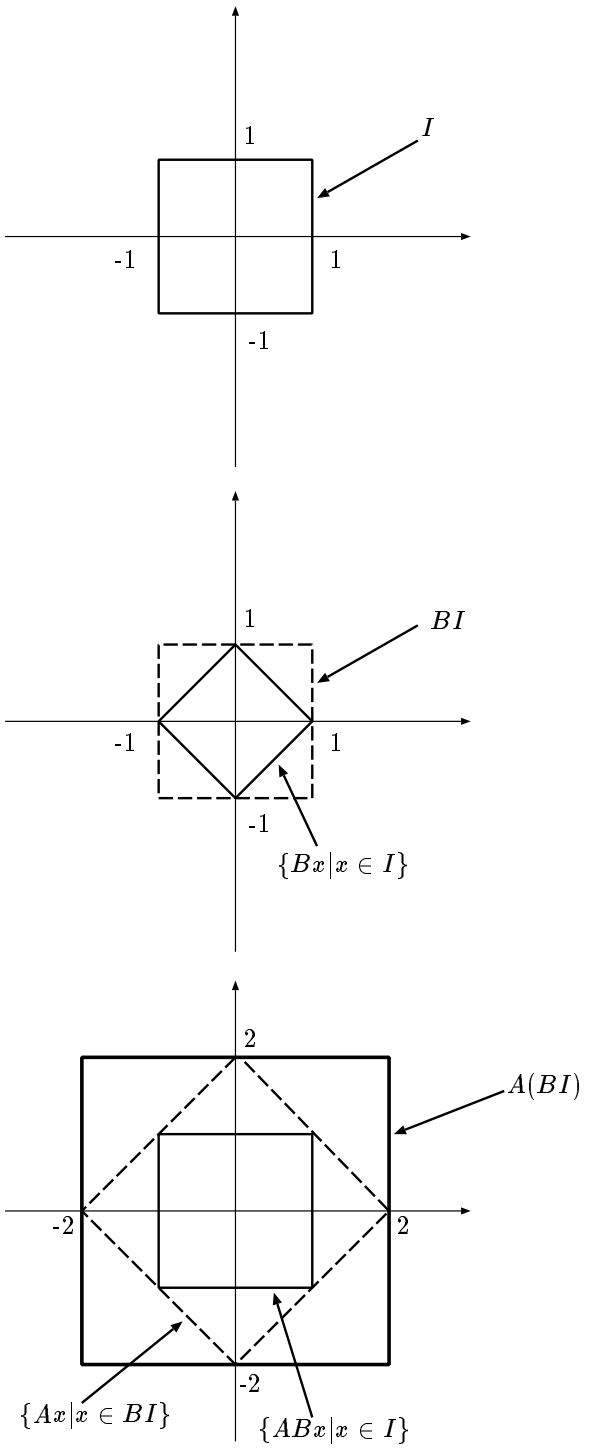

図 2.1: Wrapping Effect の様子

2.1.5 平均値形式

前で述べた区間演算における過大評価の問題を本質的に解決するための方法として変数間の相関性を考慮しつつ計算を行う方法が考えられる。**平均値形式** (Mean Value Form) は、そのような方法で代表的なものの一つである。

関数

$$f(x), \quad f : \mathbf{R^n} \to \mathbf{R^m}$$

に対して、$I \in \mathbf{IR^n}$ の像 $f(I) = \{f(x) | x \in I\}$ を評価することを考える。$x, c \in I$ に対して、平均値の定理

$$f(x) - f(c) = \int_0^1 f'(c + t(x - c))\mathrm{d}t(x - c)$$

が成立する。ここで、

$$\int_0^1 f'(c + t(x - c))\mathrm{d}t \in co\{f'(x) | x \in I\}$$

(co は凸包) が成立することを利用すると、$f(I) = \{f(x) | x \in I\}$ は

$$f(c) + F'(I)(I - c)$$

のように評価できる。ここで F' は f の導関数 f' の区間包囲である。区間行列 $F'(I)$ は導関数 f' に区間 I を代入して区間演算を行なうことによって計算することができる。区間行列は凸集合であるから、凸包を考える必要は無い。c は入力区間 I に含まれる任意の点だが、普通は I の中心に取る。前述の例をこの方法で計算すると、

$$\begin{aligned} f(c) + F'(I)(I - c) &= f(1) + F'([0.9, 1.1])([0.9, 1.1] - 1) \\ &= -1 + [-0.2, 0.2][-0.1, 0.1] \\ &= [-1.02, -0.98] \end{aligned}$$

とかなり改善されることが分かる。

で述べたように x^2 と $2x$ の間の強い相関を無視したことが通常の区間演算での過大評価の原因であったが、ここでは

$$F'([0.9, 1, 1]) = 2[0.9, 1.1] - 2 = [-0.2, 0.2]$$

のようにきちんと傾きがキャンセルされていることが分かる。

また、変数の数が多いときは実用的ではないが、一変数関数に関してはより高次の Taylor 展開を考えるのにも役立つ。Taylor 展開

$$f(x) = \sum_{k=0}^{n} \frac{1}{k!} f^{(k)}(c)(x-c)^k + R_{n+1}$$

に対して、Bernoulli の剰余項

$$R_{n+1} = \frac{1}{n!} \int_0^1 (1-t)^n f^{(n+1)}(c + t(x-c)) \mathrm{d}t (x-c)^{n+1}$$

を考える。($f^{(k)}$ は f の k 階導関数)。ここで、$s = (1-t)^{n+1}$ とおくと、

$$R_{n+1} = \frac{1}{(n+1)!} \int_0^1 f^{(n+1)}(c + (1 -_{n+1} \sqrt{s}(x-c)) \mathrm{d}s (x-c)^{n+1}$$

と変型できるので、平均値形式と同様に考えて、

$$\sum_{k=0}^{n} \frac{1}{k!} f^{(k)}(c)(I-c)^k + \frac{1}{(n+1)!} f^{(n+1)}(I)(I-c)^{n+1}$$

のような区間演算を行うことで $f(I)$ の包含で得られる。平均値形式は $n = 0$ の場合に相当する。

2.2 ベキ級数演算

以下に、文献 [1] の方法で用いる 2 種類のベキ級数演算 (Power Series Arithmetic、以下 PSA) を定義を説明する。

2.2.1 Type-I PSA

Type-I PSA では、order$-n$ のベキ級数を扱い、それより高次の項は切り捨てる。$[a_0, a_1, a_2, \cdots, a_n]$ という表記で、ベキ級数

$$a_0 + a_1 t + a_2 t^2 + \cdots + a_n t^n \quad (+O(t^{n+1})) \tag{2.1}$$

を表すことにする。

加算、減算および乗算は次のように行われる。

$$[a_0, \cdots, a_n] \pm [b_0, \cdots, b_n] = [a_0 \pm b_0, \cdots, a_n \pm b_n], \tag{2.2}$$

$$[a_0, \cdots, a_n] \times [b_0, \cdots, b_n] = [c_0, \cdots, c_n] \tag{2.3}$$

$$c_k = \sum_{i=0}^{k} a_i b_{k-i} \tag{2.4}$$

関数の適用は次のように行われる。

$$f([a_0, \cdots, a_n]) = f(a_0) + \sum_{i=1}^{n} \frac{1}{i!} f^{(i)}(a_0)[0, a_1, \cdots, a_n]^i \tag{2.5}$$

上式中に現れる加算および乗算は、式 (2.2), 式 (2.3) によって行われる。

除算は逆関数と乗算との組み合わせによって行われる。$(x/y = x \times (1/y))$.

Type-I PSA は、全く打ち切りのない演算を行った場合の最初の $(n+1)$ 項を保存するように定義された演算である。

2.2.2 Type-II PSA

Type-II PSA も Type-I と同様に order$-n$ のべき級数を扱う。Type-II PSA では、取り扱う級数の定義域（t の変域）を、$[0, d]$ のように定める必要がある。$[a_0, \cdots, a_n]$ という表記

は同じく

$$a_0 + a_1 t + a_2 t^2 + \cdots + a_n t^n \tag{2.6}$$

を表すが、最後の項の係数 a_n は一般的に区間となり、$[a_0, \cdots, a_n]$ は $f(t) \in a_0 + \cdots + a_n t^n$ in all t を満たすような $[0, d]$ 上で定義された連続関数の集合を表す。

加算および減算は Type-I PSA と同様である。乗算は次のように行われる。

(1) $[a_0, \cdots, a_n]$ と $[b_0, \cdots, b_n]$ を打ち切りなしに乗ずる。計算結果 $C = [c_0, \cdots, c_n]$ は order$-2n$ となる。

(2) C の次数を n に減らす。

減次は次のように行う。

定義　2.1(減次 = 丸め)

$A = [a_0, \cdots, a_m]$ をベキ級数、n を $n < m$ を満たす自然数とする。このとき A の次数 n への減次は次の $B = [b_0, \cdots, b_n]$ で定義される。

$$b_i = a_i \quad (i \leq 0 \leq n-1) \tag{2.7}$$

$$b_n = \left\{ \sum_{i=n}^{m} a_i t^{i-n} \mid t \in [0, d] \right\} \tag{2.8}$$

これは、項 $a_i t^i$ を $a_i t^{i-n} t^n$ と変形し、t^{i-n} を $[0, d]^{i-n}$ で置き換えることによって、高次の剰余を t^n の係数に加えている。よって、乗算の結果 C は丸めなしの乗算によって得られる可能性のある結果をすべて含んでいる。

関数は次のように適用される。

$$f([a_0, \cdots, a_n]) = f(a_0) + \sum_{i=1}^{n-1} \frac{1}{i!} f^{(i)}(a_0)[0, a_1, \cdots, a_n]^i] \tag{2.9}$$

$$+ \quad \frac{1}{n!} f^{(n)} \left\{ \sum_{i=0}^{n} a_i t^i \mid t \in [0, d] \right\} [0, a_1, \cdots, a_n]^n \tag{2.10}$$

上のアルゴリズム中に現れる加算および乗算はType-II PSAによって行われる。この方法ではLagrangeの剰余項を用いている。

除算はType-I PSAと同様に行われる。

ここで、Type-IPSAとType-IIPSAの違いを$(1-4t-t^2)\times(2+t-3t^2), t\in[0,1]$を例にして示す。

(1)Type-I PSAの場合

$$(1-4t-t^2)(2+t-3t^2)$$

$$=2-7t-9t^2+11t^3+3t^4$$

$$\Downarrow \text{高次の項の切り捨て}$$

$$\to 2-7t-9t^2.$$

(2)Type-II PSAの場合

$$(1-4t-t^2)(2+t-3t^2)$$

$$=2-7t-9t^2+11t^3+3t^4$$

$$=2-7t+(-9+11t+3t^2)t^2$$

$$\Downarrow \text{多項式の評価}$$

$$\to 2-7t-[-9,5]t^2. \quad \leftarrow 4\text{次から}2\text{次への減次}$$

(2)のようにType-II PSAでは、丸めなしの乗算によって得られる可能性のある結果を全て含ませるために、最後の項に対して多項式の評価を行っている。本論文においては、後述するホーナー法によって多項式の評価を行っている。

2.3 ホーナー法

本論文において、多項式の評価の方法として取り上げているホーナー法について説明する。

多項式 $p : \mathbf{R} \to \mathbf{R}$

$$p(t) = \sum_{i=0}^{n} p_i t^i \tag{2.11}$$

を考える。ここで、$p_i, t \in \mathbf{R}$ である。ここで $p_n \neq 0$ とし、多項式 p の値を点 $t \in \mathbf{R}$ で計算する問題を考える。通常、多項式の値の評価にはこのホーナー法が用いられる。これは、次の形に変形して計算する方法である。

$$p(t) = ((\cdots((p_n t + r_{n-1})t + p_{n-2})t + \cdots p_2)t + p_1)t + p_0 \tag{2.12}$$

すなわち、ホーナー法は

$$\begin{cases} x_n = p_n, \\ x_i = x_{i+1}t + p_i \quad (i = n-1, \cdots, 0) \end{cases} \tag{2.13}$$

と計算したとき、$p(t) = x_0$ と計算する方法である。

式 (2.12) から

$$x_{n-1} = x_n t + p_{n-1} = p_n t + p_{n-1}, x_{n-2} = (p_n t + p_{n-1})t + p_{n-2}, \cdots \tag{2.14}$$

と順次求めていくと、最後に $x_0 = ((2.12)\text{ の右辺 }) = p(t)$ が得られる。

ホーナー法では乗算と加算がそれぞれ n 回で多項式の値が計算できる。これは 1 点 t での多項式の値を計算する乗除算加減算の回数の最小値であることが知られている。

また、計算誤差の点でも優れた方法である。

第 3 章

ベキ級数演算を用いた解の包み込み

(文献[1])

3.1 はじめに

本節では、ベキ級数演算を常微分方程式の初期値問題の長時間積分に適用するための準備として、参考文献 [1] の方法について述べる。

[1] の方法は、式 (3.1) のような正規形の連立一階常微分方程式の初期値問題に対する精度保証付きの数値解法を提示したものである。

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = f(x(t), t), \quad t \in [a, b] \tag{3.1}$$

このような微分方程式を数値的に解くためには、定義域を $a = t_1 < t_2 < \cdots < t_m = b$ のように分割して離散化し、$x(t_i)$ の近似 x_i を未知数とする有限次元方程式に帰着させることがよく行われる。この過程でいわゆる離散化誤差が生じ、それを厳密に見積もることはきわめて難しい。しかし、もし隣同士の値 x_i と x_{i+1} との関係を正確に記述することができれば、真の値（微分方程式の軌道上の値）$x(t_i)$ に一致する解 x_i を持つような有限次元方程式が得られるはずである。すなわち、条件 $x(t_s) = v$ の下で正確な $x(t_e)$ の値を与えるような関数 $\phi(v, t_s, t_e)$ を計算できればその正確な解をもつ有限次元方程式を、

$$\begin{aligned} x_2 &= \phi(x_1, t_1, t_2) \\ x_3 &= \phi(x_2, t_2, t_3) \\ &\vdots \\ x_m &= \phi(x_{m-1}, t_{m-1}, t_m) \end{aligned} \tag{3.2}$$

のように書くことができる。この方程式は $n \times m$ 個の未知数と $n \times (m-1)$ 個の方程式を持つ。従って、n 個の境界条件を付加することによって、解けることが期待できる。

次節に関数 $\phi(v, t_s, t_e)$ の計算方法を示す。

3.2 $\phi(v, t_s, t_e)$の計算方法

$\phi : \mathbf{R^n} \times \mathbf{R} \times \mathbf{R} \to \mathbf{R^n}$ は次の初期値問題の保証付き解を計算することによって計算できる。

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = f(x(t), t) \tag{3.3}$$

$$x(t_s) = v \tag{3.4}$$

$$t \in [t_s, t_e] \tag{3.5}$$

$x(t)$が正確に計算できれば、$\phi(x, t_s, t_e)$は$x(t_e)$で得られる。

ここで、Picardの反復法に基づき、$x(t)$の包み込みを得るアルゴリズムを与える。式(3.3)を同値な不動点形式

$$x(t) = v + \int_{t_s}^{t} f(x(s), s)ds \tag{3.6}$$

に変換し、定数関数vを初期値とする反復によって近似解を得たあと、真の解の存在をSchauderの不動点定理によって示す。

アルゴリズムを示す。独立変数tが$([t_s, t_e] \to [0, t_e - t_s])$と変換していることに注意する。

アルゴリズム 3.1 $t_e > t_s, \triangle = t_e - t_s$とし、Type-IIPSAの定義域を$[0, \triangle]$とする。

Step-1 べき級数ベクトルXを

$$X = \begin{pmatrix} [v_1] \\ \vdots \\ [v_n] \end{pmatrix} \tag{3.7}$$

のように初期化し、$m = 0$とする。

Step-2(近似解の生成) Step-2(1)- Step-2(3)を適当な回数繰り返す。

Step-2(1) べき級数 $T(t_s+t)$ を m 次で打ち切ったものとする。

$$T=\begin{cases}[t_s] & (m=0)\\ [t_s,1] & (m=1)\\ [t_s,1,0,\cdots,0] & (m<2)\end{cases} \tag{3.8}$$

Step-2(2) $X=f(X,T)$ を Type-I PSA で計算する。$X=v+\int_0^t X\,dt$ を計算する。これらによって、X の次数は m は $m+1$ となる。

Step-2(3) $m=m+1$

Step-3 $T=[t_s,1,0,\cdots,0](\text{order } m)$ とする。

Step-4 $Y=f(X,T)$ を Type-II PSA で計算し、$Y=v+\int_0^t Y\,dt$ を計算し、定義 2.1 によって Y の次数を $m+1$ から m に減次する。

Step-5

$$r=\max_{1\le i\le n}|Y_i^{(m)}-X_i^{(m)}| \tag{3.9}$$

を計算する。ここで添字 (k) は t^k の係数を表す。

Step-6 $Let\ x_i^{(m)}=x_i^{(m)}+[-2r,2r]for 1\le i\le n.$

Step-7 $Y=f(X,T)$ を Type-II PSA で計算し、$Y=v+\int_0^t Y\,dt$ を計算し、定義 2.1 によって Y の次数を $m+1$ から m に減次する。

Step-8(存在検証) 全ての i について

$$Y_i^{(m)}\subset X_i^{(m)} \tag{3.10}$$

が成立すれば、Y に式 (3.3) の解が存在することが Schauder の不動点定理により検証される。

Step-9(解の改良) Step-9(1)-Step-9(2) を $Y_i^{(m)}$ が十分小さくなるまで繰り返す。

Step-9(1) $X = f(X,T)$ を Type-II PSA で計算し、$X = v + \int_0^t X dt$ を計算し、定義 2.1 によって X の次数を $m+1$ から m に減次する。

Step-9(2) 全ての i について $Y_i^{(m)} = Y_i^{(m)} \cap X_i^{(m)}$ とする。

Step10 $\phi(v, t_s, t_e)$ は

$$\phi = \begin{pmatrix} \sum_{i=0}^{m} Y_1^{(i)} \triangle^i \\ \vdots \\ \sum_{i=0}^{m} Y_n^{(i)} \triangle^i \end{pmatrix} \tag{3.11}$$

で得られる。

Step-8 において、$0 \leq k \leq m-1$ で $Y_i^{(k)} = X_i^{(k)}$ が常に成立し、よって最後の項だけで比較を行えばよい。

第 4 章

長時間積分

4.1 長時間積分の手法

本節では、前節に示した文献 [1] の方法を用いて、$x(t_2), x(t_3), \cdots, x(t_m)$ の包み込みを順に求めることを考える。

まず、$\phi(x_1, t_1, t_2)$ を計算することで、$x(t_2)$ の値を包み込む区間が得られる。このとき、得られた区間を初期値とする $\phi(x_2, t_2, t_3)$ を計算すると、$x(t_3)$ の値を包み込む区間が得られる。これは区間演算の性質から、当然 $x(t_2)$ の包み込みより大きな幅の区間となる。

もし、このあと同様に $x(t_4), x(t_5), \cdots$ の包み込みを計算していくと区間幅はどんどん増大し、$x(t_m)$ の包み込みは下の図 4.1 に示すようにかなり大きな幅の区間となることが予想される。

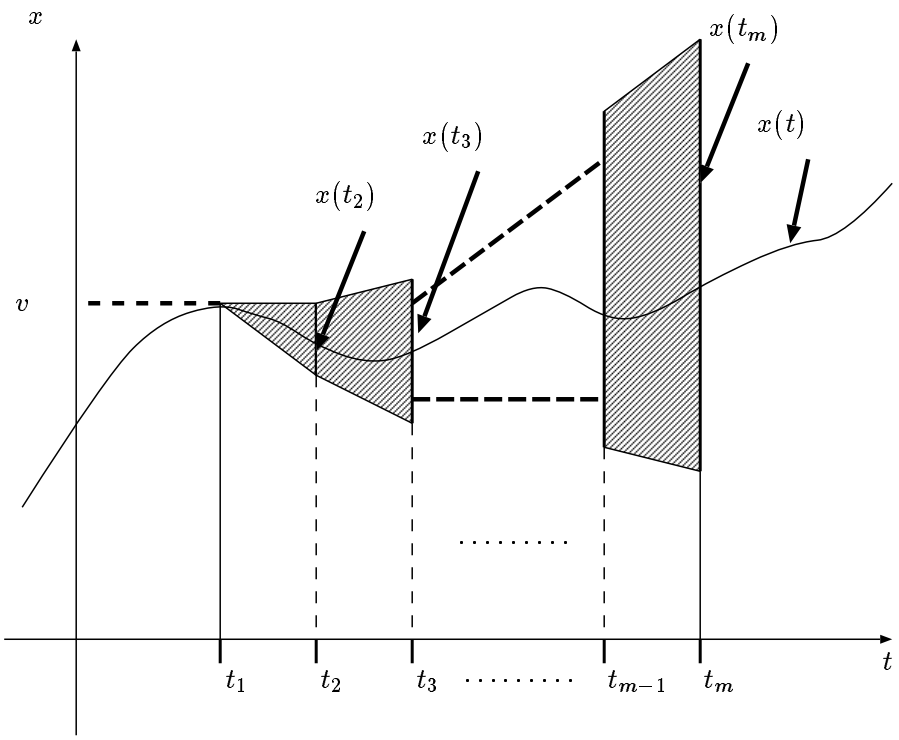

図 4.1: 区間を初期値として長時間積分を行った場合

そこで、ベキ級数演算を常微分方程式の初期値問題の長時間積分に適用する方法として、次のような平均値形式を用いる (但し、Mid(I) は区間 I の中心)。

$$F(I) = f(c) + F'(I)(I - c) \ , \ c = \mathrm{Mid}(I) \tag{4.1}$$

ここで、$F'(I)$ は $\phi_v(v, t_s, t_e)$ の区間包囲である。

但し、$\phi_v(v, t_s, t_e)$ は $\phi(v, t_s, t_e)$ の初期値 v に関して偏微分したものである。

この平均値形式を用いることによって、上に述べたような区間幅の増大を次の図 4.2 に示すように抑えられることが期待できる。

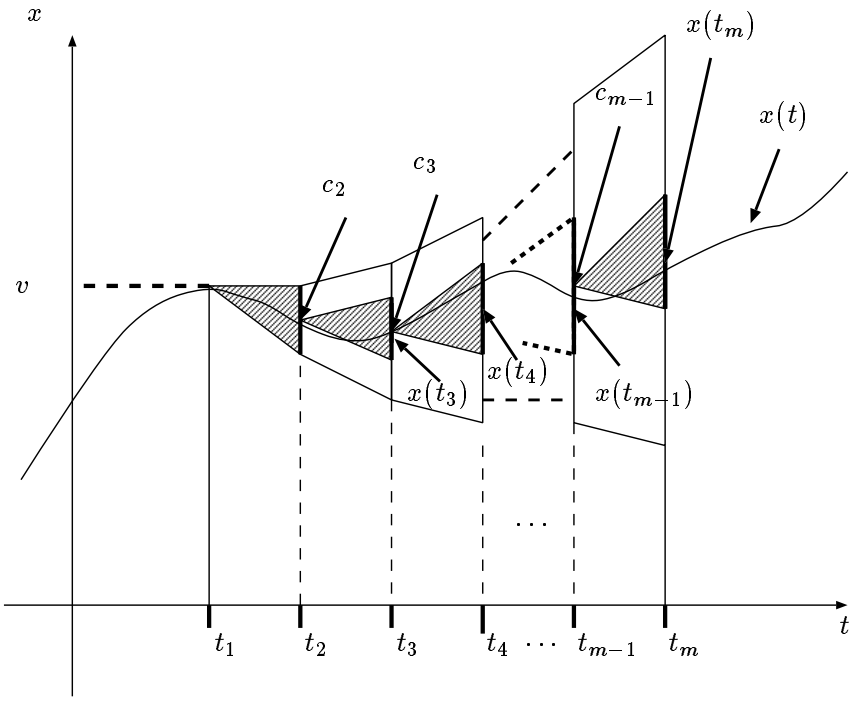

図 4.2: 平均値形式を用いて長時間積分を行った場合

4.2 $\phi_v(v, t_s, t_e)$の計算方法

$\phi_v(x, t_s, t_e)$を次のようにして得ることができる。: 連立初期値問題

$$\begin{aligned}
\frac{\mathrm{d}x(t)}{\mathrm{d}t} &= f(x(t), t) && (4.2)\\
\frac{\mathrm{d}y(t)}{\mathrm{d}t} &= f_x(x(t), t)y(t) && (4.3)\\
x(t_s) &= v && (4.4)\\
y(t_s) &= I \ (n \times n \text{ identity matrix}) && (4.5)\\
t &\in [t_s, t_e] && (4.6)
\end{aligned}$$

を考える。これを第3章2節の方法で解けば、$\phi_v(x, t_s, t_e)$は$y(t_e)$で得ることができる。

4.3 Wrapping Effectの抑止方法

平均値形式

$$F(I) = f(c) + F'(I)(I - c) \ , \ c = \mathrm{Mid}(I) \tag{4.7}$$

を用いる場合、$F'(I)$は区間行列で、Iが区間ベクトルであるため、$F'(I) \times I$を計算する際、第2章1節の4で述べたWrapping Effectの影響で、区間幅の増大が生じる。そこで、そのWrapping Effectの影響を抑止する方法について説明する。

まず、Wrapping Effectの影響について考慮しない場合は、次のように計算されている。

ここで、$F'(I_n) = A_n$とする。

$$\begin{aligned}
I_1 &= f(c_0)\\
I_2 &= f(c_1) + A_1(I_1 - c_1)\\
I_3 &= f(c_2) + A_2(I_2 - c_2)\\
&\vdots\\
I_n &= f(c_{n-1}) + A_{n-1}(I_{n-1} - c_{n-1}) && (4.8)
\end{aligned}$$

この場合は、ステップ毎に $A_{n-1} \times I_{n-1}$ を計算するので、求められる区間 I はステップ毎に Wrapping Effect の影響を受け、かなりの区間幅の増大が生じる。

そこで、以下のように式 (4.8) を変形し、Wrapping Effect の影響を抑止する。

$$
\begin{aligned}
I_1 &= f(c_0) \\
I_2 &= f(c_1) + A_1(I_1 - c_1) \\
&= f(c_1) + A_1(f(c_0) - c_1) \\
I_3 &= f(c_2) + A_2(I_2 - c_2) \\
&= f(c_2) + A_2(f(c_1) + A_1(f(c_0) - c_1) - c_2) \\
&= f(c_2) + A_2(f(c_1) - c_2) + A_2A_1(f(c_0) - c_1)
\end{aligned}
$$

同様に

$$
\begin{aligned}
I_4 &= f(c_3) + A_3(f(c_2) - c_3) + A_3A_2(f(c_1) - c_2) + A_3A_2A_1(f(c_0) - c_1) \\
&\vdots \\
I_n &= f(c_{n-1}) + \sum_{i=1}^{n-1}[(\prod_{j=i}^{n-1} A_{n-j})(f(c_{i-1}) - c_i)] \qquad (4.9)
\end{aligned}
$$

上のように式変形をすると、ステップ毎に区間行列と区間の乗算を行うことがない。最後に求める区間 I において、まとめて行列を掛け合わせた上で、区間との乗算を行うので、最後の計算のときだけしか Wrapping Effect の影響を受けなくてすむ。

しかしながら、現実にこの方法で進めていくと、例えば I_{100} を求めると、

$$
\begin{aligned}
I_{100} = f(c_{99}) &+ A_{99}(f(c_{98}) - c_{99}) \\
&+ A_{99}A_{98}(f(c_{97}) - c_{98}) \\
&+ A_{99}A_{98}A_{97}(f(c_{96}) - c_{97}) \\
&+ \cdots \\
&\vdots \\
&+ A_{99}A_{98}\cdots A_2A_1(f(c_0) - c_1) \qquad (4.10)
\end{aligned}
$$

となり、最後の項のように行列の乗算をかなりの回数行わなければならなく、ステップ回数が増す毎に計算時間が長くなってしまう。

そこで計算時間を短くするために、Wrapping Effect の影響を考慮する範囲をあらかじめ決めておき、その範囲でだけ、式 (4.9) のような式変形をして、Wrapping Effect の影響を抑止していく。

例えば、Wrapping Effect の影響を考慮する範囲を前 2 回とすると、

$$
\begin{aligned}
&\vdots \\
I_3 &= f(c_2) + A_2(f(c_1) - c_2) + A_2A_1(I_1 - c_1) \\
I_4 &= f(c_3) + A_3(f(c_2) - c_3) + A_3A_2(I_2 - c_2) \\
I_5 &= f(c_4) + A_4(f(c_3) - c_4) + A_4A_3(I_3 - c_3) \\
&\vdots \\
I_{100} &= f(c_{99}) + A_{99}(f(c_{98}) - c_{99}) + A_{99}A_{98}(I_{98} - c_{98}) \\
&\vdots \\
I_n &= f(c_{n-1}) + A_{n-1}(f(c_{n-2}) - c_{n-1}) + A_{n-1}A_{n-2}(I_{n-2} - c_{n-2}) \qquad (4.11)
\end{aligned}
$$

のように計算していく。

区間幅の増大をできるだけ抑えるためには、計算時間が、現実的に許され得るぎりぎりになるまで、Wrapping Effect の影響を考慮する範囲を広げていく必要がある。

第 5 章

数値実験

本章では、前章までに述べたアルゴリズムをすべて実装したC++のプログラムを以下のような正規形の2次元の一階常微分方程式に対して実行する。

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = f(x(t),t) \quad , \quad x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \quad , \quad f(x(t),t) = \begin{pmatrix} -2tx_1 + t \\ -x_2 + t \end{pmatrix},$$
$$x(0) = \begin{pmatrix} 0.0 \\ 0.0 \end{pmatrix} \quad , \quad t \in [0,1] \tag{5.1}$$

以下のことについてそれぞれ数値実験を行い、検証していく。

1, 第3章2節のアルゴリズムに従って、$x(t_{i+1}) = \phi(v, t_s, t_e)$ を計算する。

2, 実際に長時間積分を行うこと、また、平均値形式を用いることの有効性について、さらにWrapping Effectの影響を考慮することの有効性について

3, 第3章2節の $\phi(v, t_s, t_e)$ 計算の際の、STEP-2における近似解の生成における次数 m を変えたときのその実行結果の移行について

4, 式(5.1)において、定義域を分割する際のその一つ一つの刻み幅 $(= t_2 - t_1 = t_3 - t_2 = \cdots = t_m - t_{m-1})$ を変えたときのその実行結果の移行について

5,Wrapping Effectを考慮する範囲を変えたときのその実行結果の移行について

6, 長時間積分が行える t の範囲について

第1章2節で述べたように本論文の目的は区間幅の広がりを抑えながら式(1.1)における $x(b)$ の値を精度保証付きで求めることであるから、2〜5の比較については、それぞれの t における $x(t)$ の値の精度をもって比較する。よって、2〜5に示していく数値は、すべて、$x(t)$ の数値解ではなく、その数値解の区間幅である。

当然、区間幅が小さければ小さいほど精度が高いということを示す。

また、m を近似解生成時の次数とし、h を刻み幅とする。

なお、以下の数値例において、例えば

$$0.277570966888^{474}_{649} = [0.277570966888474, 0.277570966888649]$$

$$0.306569659740^{598}_{600} = [0.306569659740598, 0.3065696597406]$$

である。

また、本章で行う数値実験は全て“Free BSD2.2.8 コンパイラ:gcc version2.7.2.1 CPU:Pentium II 350MHz メモリ:128MB ”で行った。

5.1 $x(t_{i+1}) = \phi(v, t_s, t_e)$ の計算

本節では式 (5.1) において、$x(t_{i+1}) = \phi(v, t_s, t_e)$ の計算の例として、$h = 0.1$,$m = 10$ として計 $x(1) = \phi(x(0.9), 0.9, 1)$ を計算した数値例を**アルゴリズム 3.1** に沿って示していく。なお、数値例を $[a_0, a_1, a_2, \cdots, a_n]$ のように示すときは、ベキ級数の表記である。また、第 m 項だけはその数値を区間で表す。

Step-1 べき級数ベクトル X を

$$X = \begin{pmatrix} 0.277570966888^{474}_{649} \\ 0.306569659740^{598}_{600} \end{pmatrix}$$

のように初期化し、$m = 0$ とする。

Step-2(近似解の生成) Step-2(1)- Step-2(3) を 10 回繰り返す。

$m = 1$

$$X = \begin{pmatrix} [0.277570966888^{474}_{649}, 0.400372259600^{432}_{747}] \\ [0.306569659740^{598}_{600}, 0.5934303402594^{00}_{02}] \end{pmatrix}$$

$m = 2$

$$X = \begin{pmatrix} [0.277570966888^{474}_{649}, 0.400372259600^{432}_{747}, -0.137906000529^{9321}_{8863}] \\ [0.306569659740^{598}_{600}, 0.5934303402594^{00}_{02}, 0.203284829870^{299}_{300}] \end{pmatrix}$$

$\vdots$

$m = 9$

$$X = \begin{pmatrix} [0.277570966888^{474}_{649}, 0.400372259600^{432}_{747}, -0.137906000529^{9321}_{8863}, -0.184171239416^{514}_{029} \\ [0.306569659740^{598}_{600}, 0.5934303402594^{00}_{02}, 0.203284829870^{299}_{300}, -0.0677616099567667^{7}_{3} \\ , 0.151830058001^{1644}_{2092}, 0.0190096748^{56584}_{60136}, -0.0563129217998^{8347}_{5789} \\ , 0.016940402489191^{6}_{7}, -0.00338808049783^{4}_{2}, 0.000564680082973^{3}_{6} \\ , 0.00904912992^{388783}_{405508}, 0.0120421762^{69823}_{70839}, -0.00441935300^{31791}_{26043}] \\ , -8.066858328186^{52}_{47}\text{e-}05, 1.00835729102331\text{e-}05, -1.1203969900259^{1}_{0}\text{e-}06] \end{pmatrix}$$

$m = 10$

$$X = \begin{pmatrix} [0.277570966888^{474}_{649}, 0.400372259600^{432}_{747}, -0.137906000529^{9321}_{8863}, -0.184171239416^{514}_{029} \\ [0.306569659740^{598}_{600}, 0.5934303402594^{00}_{02}, 0.203284829870^{299}_{300}, -0.0677616099567667^{7}_{3} \\ , 0.151830058001^{1644}_{2092}, 0.0190096748^{56584}_{60136}, -0.0563129217998^{8347}_{5789} \\ , 0.016940402489191^{6}_{7}, -0.00338808049783^{4}_{2}, 0.000564680082973^{3}_{6} \\ , 0.00904912992^{388783}_{405508}, 0.0120421762^{69823}_{70839} \\ , -8.066858328186^{52}_{47}\text{e-}05, 1.00835729102331\text{e-}05 \\ , -0.00441935300^{31791}_{26043}, -\mathbf{0.00161295170}26^{\mathbf{499}}_{\mathbf{1924}}] \\ , -1.1203969900259^{1}_{0}\text{e-}06, \mathbf{1.1203969900259}^{\mathbf{0}}_{\mathbf{1}}\text{e-}07] \end{pmatrix}$$

Step-3 $T = [t_s, 1, 0, \cdots, 0]$(order m) とする。

Step-4 $Y = f(X,T)$ を Type-II PSA で計算し、$Y = v + \int_0^t Y dt$ を計算し、定義 2.1 によって Y の次数を $m+1$ から m に減次する。

$$
Y = \left(\begin{array}{l}
[0.277570966888^{474}_{649}, 0.400372259600^{432}_{747}, -0.137906000052^{9321}_{8863}, -0.184171239416^{514}_{029} \\
[0.306569659740^{598}_{600}, 0.593430340259^{00}_{02}4, 0.203284829870^{299}_{300}, -0.067761609956766^{7}_{3} \\
,0.151830058002^{1644}_{2092}, 0.019009674886^{56584}_{60136}, -0.056312921799^{8347}_{5789} \\
,0.016940402489191^{6}_{7}, -0.0033880804978383^{4}_{2}, 0.00056468008297305^{3}_{6} \\
,0.009049129923^{88783}_{405508}, 0.012042176219^{69823}_{70839} \\
,-8.066858328186^{52}_{47}\text{e-}05, 1.00835729102331\text{e-}05 \\
,-0.004419353004^{31791}_{26043}, -\mathbf{0.000^{414697254999 44}_{252231477282381}}] \\
,-1.12039699002591^{1}_{0}\text{e-}06, \mathbf{1.0^{1854271820536}_{2872814538742}}\text{e-}07]
\end{array} \right)
$$

Step-5

$$
r = \max_{1 \le i \le n} |Y_i^{(m)} - X_i^{(m)}| \tag{5.2}
$$

を計算する。

$$
r = \left(\begin{array}{l}
0.001360720225 \\
1.018542718\text{e-}08
\end{array} \right)
$$

Step-6 $Let\ x_i^{(m)} = x_i^{(m)} + [-2r, 2r] for 1 \le i \le n.$

これにより、$X^{(m)}$ は次のようになる。

$$
X^{(m)} = \left(\begin{array}{l}
[-0.004334392153, 0.001108488748] \\
[9.166884464\text{e-}08, 1.324105534\text{e-}07]
\end{array} \right)
$$

Step-7 $Y = f(X,T)$ を Type-II PSA で計算し、$Y = v + \int_0^t Y dt$ を計算し、定義 2.1 によって Y の次数を $m+1$ から m に減次する。

これにより、$Y^{(m)}$ は次のようになる。

$$Y^{(m)} = \begin{pmatrix} [-0.001234424076, 0.0006879024966] \\ [1.00002376\text{e-}07, 1.045395208\text{e-}07] \end{pmatrix}$$

Step-8(存在検証) 全ての i について $Y_i^{(m)} \subset X_i^{(m)}$ が成立すれば、Y に式 (5.1) の解が存在することが Schauder の不動点定理により検証される。**Step-6** と **Step-7** の結果から $Y_i^{(m)} \subset X_i^{(m)}$ が成立する。よって、解が存在する。

Step-9(解の改良) Step-9(1)-Step-9(2) を $Y_i^{(m)}$ が十分小さくなるまで繰り返す。

これにより、$Y_i^{(m)}$ が、以下のように改良される。

$$\begin{aligned}
Y_i^{(m)} &= \begin{pmatrix} [-0.001234424076, 0.0006879024966] \\ [1.00002376\text{e-}07, 1.045395208\text{e-}07] \end{pmatrix} \\
&\Downarrow \\
Y_i^{(m)} &= \begin{pmatrix} -0.00^{1103659987}_{03829955664} \\ 1.0^{25361062}_{38576864}\text{e-}07 \end{pmatrix} \\
&\Downarrow \\
Y_i^{(m)} &= \begin{pmatrix} -0.000^{7769752442}_{4281686153} \\ 1.0^{25980911}_{36503812}\text{e-}07 \end{pmatrix} \\
&\Downarrow \\
&\vdots \\
&\Downarrow \\
Y_i^{(m)} &= \begin{pmatrix} -0.000^{725156184}_{5589244746} \\ 1.0^{26175428}_{36437182}\text{e-}07 \end{pmatrix}
\end{aligned}$$

Step10 $\phi(v, t_s, t_e)$ は

$$\phi = \begin{pmatrix} \sum_{i=0}^{m} Y_1^{(i)} \triangle^i \\ \vdots \\ \sum_{i=0}^{m} Y_n^{(i)} \triangle^i \end{pmatrix} \tag{5.3}$$

で得られる。

これにより、$\phi(x(0.9), 0.9, 1)$ は、以下のようになる。

$$\phi(x(0.9), 0.9, 1) = \begin{pmatrix} 0.3160602794\,14^{277}_{506} \\ 0.3678794411\,7144^{1}_{4} \end{pmatrix}$$

5.2 長時間積分の実行、また平均値形式を用いることの有効性についての検証

本節では実際に長時間積分を行い、平均値形式を用いて長時間積分を行うことの有効性について、さらに Wrapping Effect の影響を考慮することの有効性について検証する。

また、例題の解析解との数値の比較を行い、正しい値が得られている事を確認する。

平均値形式を用いずにそれぞれの t における初期値を区間として長時間積分を行った場合と、平均値形式を用いて長時間積分を行った場合、さらに Wrapping Effect の影響を前 2 回で考慮した場合とを、以下の条件のもと比較する。

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = f(x(t), t) \quad , \quad x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \quad , \quad f(x(t), t) = \begin{pmatrix} -2tx_1 + t \\ -x_2 + t \end{pmatrix}, \tag{5.4}$$

$$x(0) = \begin{pmatrix} 0.0 \\ 0.0 \end{pmatrix} \quad , \quad t \in [0, 1] \tag{5.5}$$

$$m = 10 \quad , \quad h = 0.01 \tag{5.6}$$

次の表がその実行結果である。なお、数値解そのものではなく、その数値解の区間幅を示す。

表 5.1: 平均値形式を用いた場合と用いない場合の比較

t	平均値形式を用いない場合	平均値形式を用いた場合	Wrapping Effect を考慮した場合
0	$\begin{pmatrix} 0.0 \\ 0.0 \end{pmatrix}$	$\begin{pmatrix} 0.0 \\ 0.0 \end{pmatrix}$	$\begin{pmatrix} 0.0 \\ 0.0 \end{pmatrix}$
⋮	⋮	⋮	⋮
0.99	$\begin{pmatrix} 3.241851232\text{e-}14 \\ 2.803313137\text{e-}14 \end{pmatrix}$	$\begin{pmatrix} 1.587618925\text{e-}14 \\ 1.842970220\text{e-}14 \end{pmatrix}$	$\begin{pmatrix} 1.665334536\text{e-}15 \\ 1.665334536\text{e-}15 \end{pmatrix}$
1	$\begin{pmatrix} 3.358424649\text{e-}14 \\ 2.886579864\text{e-}14 \end{pmatrix}$	$\begin{pmatrix} 1.620925615\text{e-}14 \\ 1.887379141\text{e-}14 \end{pmatrix}$	$\begin{pmatrix} 1.665334536\text{e-}15 \\ 1.665334536\text{e-}15 \end{pmatrix}$

まず、$t = 1$ までアルゴリズムどおりに、長時間積分がきちんと行えたことが確認できた。また上の表より、平均値形式を用いない場合に比べて平均値形式を用いた場合は精度が高くなっていることが分かる。また、Wrapping Effect の影響を前 2 回で考慮した場合は、さらに精度が良くなっている事が分かる。

例題の解析的に求めた一般解は、初期値を式 (5.5) のようにすると、それぞれ

$$x(t) = \begin{pmatrix} -\frac{1}{2}e^{-t^2} + \frac{1}{2} \\ t - 1 + e^{-t} \end{pmatrix} \tag{5.7}$$

である。よって、解析解を $x^*(1)$ とすると、

$$x^*(\,1\,) = \begin{pmatrix} 0.31606027941427883299 \\ 0.36787944117144233402 \end{pmatrix} \tag{5.8}$$

である。対して、平均値形式を用いない場合の数値解を $x(1)$ とすると、

$$x(\,1\,) = \begin{pmatrix} 0.3160602794142^{6201311}_{9559736} \\ 0.367879441171^{2845624}_{5732204} \end{pmatrix} \tag{5.9}$$

平均値形式を用いた場合の数値解を $x(1)_{mvf}$ とすると

$$x(\,1\,)_{mvf} = \begin{pmatrix} 0.3160602794142^{704508}_{8666006} \\ 0.367879441171^{3356326}_{5243705} \end{pmatrix} \tag{5.10}$$

さらに Wrapping Effect を考慮した場合の数値解を $x(1)_{we}$ とすると

$$x(\,1\,)_{we} = \begin{pmatrix} 0.316060279414^{772276}_{93881} \\ 0.36787944117144^{216749}_{383283} \end{pmatrix} \tag{5.11}$$

である。よって、

$$x^*(\,1\,) \in x(\,1\,)_{we} \subset x(\,1\,)_{mvf} \subset x(\,1\,) \tag{5.12}$$

となり、区間の幅は異なるが、すべてに真の解が含まれていることが示された。

5.3 次数 m についての検証

本節では、同じ条件のもとで近似解生成時の次数 m を変えると実行結果がどう移行するかについて検証する。

平均値形式を用い, さらに Wrapping Effect の影響を前 2 回で考慮した上で以下の条件のもと、それぞれ $m = 7$,$m = 8$,$m = 9$,$m = 10$,$m = 11$,$m = 12$,$m = 13$,$m = 15$ の場合につい

て比較する。

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = f(x(t),t) \quad , \quad x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \quad , \quad f(x(t),t) = \begin{pmatrix} -2tx_1 + t \\ -x_2 + t \end{pmatrix},$$

$$x(0) = \begin{pmatrix} 0.0 \\ 0.0 \end{pmatrix} \quad , \quad t \in [0,1]$$

$$h = 0.1 \tag{5.13}$$

とする。

次の表がその実行結果である。

表 5.2: 次数 m を変えたときの実行結果の比較

t	$m=7$	$m=8$	$m=9$	$m=10$
0	$\begin{pmatrix} 0.0 \\ 0.0 \end{pmatrix}$	$\begin{pmatrix} 0.0 \\ 0.0 \end{pmatrix}$	$\begin{pmatrix} 0.0 \\ 0.0 \end{pmatrix}$	$\begin{pmatrix} 0.0 \\ 0.0 \end{pmatrix}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
1	$\begin{pmatrix} 5.122429148\text{e-}10 \\ 3.053668429\text{e-}13 \end{pmatrix}$	$\begin{pmatrix} 2.205985394\text{e-}11 \\ 4.49640325\text{e-}15 \end{pmatrix}$	$\begin{pmatrix} 7.963074644\text{e-}13 \\ 1.276756478\text{e-}15 \end{pmatrix}$	$\begin{pmatrix} 4.113376306\text{e-}14 \\ 1.33226763\text{e-}15 \end{pmatrix}$
t	$m=11$	$m=12$	$m=13$	$m=15$
0	$\begin{pmatrix} 0.0 \\ 0.0 \end{pmatrix}$	$\begin{pmatrix} 0.0 \\ 0.0 \end{pmatrix}$	$\begin{pmatrix} 0.0 \\ 0.0 \end{pmatrix}$	$\begin{pmatrix} 0.0 \\ 0.0 \end{pmatrix}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
1	$\begin{pmatrix} 2.498001805\text{e-}15 \\ 1.443289932\text{e-}15 \end{pmatrix}$	$\begin{pmatrix} 1.554312234\text{e-}15 \\ 1.554312234\text{e-}15 \end{pmatrix}$	$\begin{pmatrix} 1.665334537\text{e-}15 \\ 1.720845688\text{e-}15 \end{pmatrix}$	$\begin{pmatrix} 1.887379142\text{e-}15 \\ 1.998401444\text{e-}15 \end{pmatrix}$

上の表より、m が大きくなっていくとどんどん精度がよくなり、$-2tx_1 + t$ は $m = 12$ のときに、$-x_2 + t$ は $m = 9$ のときに最も精度が高くなり、それ以降は精度がどんどん悪く

なっている事が分かる。この様な結果を得た事についての考察は次章で詳しく述べる。

5.4 刻み幅についての検証

本節では、同じ条件のもとで、定義域を分割する際のその一つ一つの刻み幅 h を変えると実行結果がどう移行するかについて検証する。

平均値形式を用い, さらに Wrapping Effect の影響を前 2 回で考慮した上で以下の条件のもと、それぞれ $h = 1$,$h = 0.1$,$h = 0.01$,$h = 0.001$ の場合について比較する。

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = f(x(t),t) \quad , \quad x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \quad , \quad f(x(t),t) = \begin{pmatrix} -2tx_1 + t \\ -x_2 + t \end{pmatrix},$$

$$x(0) = \begin{pmatrix} 0.0 \\ 0.0 \end{pmatrix} \quad , \quad t \in [0,1]$$

$$m = 10 \tag{5.14}$$

とする。

次の表がその実行結果である。

表 5.3: 刻み幅 h を変えたときの実行結果の比較

t	$h = 1$ の場合	$h = 0.1$ の場合	$h = 0.01$ の場合	$h = 0.001$ の場合
0	$\begin{pmatrix} 0.0 \\ 0.0 \end{pmatrix}$	$\begin{pmatrix} 0.0 \\ 0.0 \end{pmatrix}$	$\begin{pmatrix} 0.0 \\ 0.0 \end{pmatrix}$	$\begin{pmatrix} 0.0 \\ 0.0 \end{pmatrix}$
⋮	⋮	⋮	⋮	⋮
1	$\begin{pmatrix} 0.0007575757576 \\ 2.505210883\text{e-}08 \end{pmatrix}$	$\begin{pmatrix} 4.113376306\text{e-}14 \\ 1.33226763\text{e-}15 \end{pmatrix}$	$\begin{pmatrix} 1.665334537\text{e-}15 \\ 1.665334537\text{e-}15 \end{pmatrix}$	$\begin{pmatrix} 1.665334537\text{e-}15 \\ 1.665334537\text{e-}15 \end{pmatrix}$

上の表より、h を小さくしていくにつれて精度が高くなるが、$h = 0.01$ からは精度が変わっていないことが分かる。この様な結果を得た事についての考察は次章で詳しく述べる。

5.5 Wrapping Effectについての検証

本節では、同じ条件のもとで、Wrapping Effect を考慮する範囲を変えると実行結果がどう移行するかについて検証する。

平均値形式を用いた上で以下の条件のもと、それぞれ前 2 回,3 回,5 回について考慮する場合を比較する。

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = f(x(t),t) \quad , \quad x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \quad , \quad f(x(t),t) = \begin{pmatrix} -2tx_1 + t \\ -x_2 + t \end{pmatrix},$$
$$x(0) = \begin{pmatrix} 0.0 \\ 0.0 \end{pmatrix} \quad , \quad t \in [0,1]$$
$$m = 10 \quad , \quad h = 0.1 \tag{5.15}$$

とする。

次の表がその実行結果である。

表 5.4: Wrappinf Effect を考慮する範囲を変えたときの実行結果の比較

t	前 2 回の場合	前 3 回の場合	前 5 回の場合
0	$\begin{pmatrix} 0.0 \\ 0.0 \end{pmatrix}$	$\begin{pmatrix} 0.0 \\ 0.0 \end{pmatrix}$	$\begin{pmatrix} 0.0 \\ 0.0 \end{pmatrix}$
⋮	⋮	⋮	⋮
1	$\begin{pmatrix} 4.113376306\text{e-}14 \\ 1.33226763\text{e-}15 \end{pmatrix}$	$\begin{pmatrix} 4.923839114e-14 \\ 1.554312234\text{e-}15 \end{pmatrix}$	$\begin{pmatrix} 6.322720125\text{e-}14 \\ 1.887379142\text{e-}15 \end{pmatrix}$

上の表より、考慮する範囲を大きくするほど精度が悪くなっていることが分かる。この様な結果を得た事についての考察は次章で詳しく述べる。

5.6 長時間積分が行える t の範囲について

前節までは、決められた定義域 $t \in [0,1]$ での長時間積分についての数値実験を行ってきた。定義域を定めずに長時間積分を行うと、$x(t_{i+1}) = \phi(v, t_s, t_e)$ を計算する際の式 (3.10) が成立する限り、長時間積分を続けることができる。しかし現実には、あるところで式 (3.10) が成立しなくなり、長時間積分がそれ以上行えなくなる。

長時間積分をできるだけ長い間行うことができるようにするには、式 (3.10) が成立するように、条件を設定する必要がある。その条件にあたるのは、近似解の生成時の次数 m である。この m を大きくすることによって近似解をより真の解に近づけることによって式 (3.10) が成立しやすいようにすれば、長時間積分を行える範囲を広げられることが期待できる。そこで、同じ条件のもとで近似解の生成時の次数 m を変えると長時間積分が行える範囲がどう移行するかについて検証する。

そこで、平均値形式を用い、さらに Wrapping Effect の影響を前 2 回で考慮した上で、以下の条件の下、$m = 5$,$m = 10m = 20$,$m = 30$ の場合について t がいくつのときまで実行できるかについて比較する。

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = f(x(t),t) \quad , \quad x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \quad , \quad f(x(t),t) = \begin{pmatrix} -2tx_1 + t \\ -x_2 + t \end{pmatrix},$$

$$x(0) = \begin{pmatrix} 0.0 \\ 0.0 \end{pmatrix} \quad , \quad t \in [0,1]$$

$$h = 0.1 \tag{5.16}$$

とする。

表 5.5: 次数 m を変えたときの長時間積分を行える範囲の比較

	$m = 10$ の場合	$m = 20$ の場合	$m = 50$ の場合	$m = 100$ の場合
t	1.2	1.6	2.5	3.5

上の表がその実行結果である。

上の表より m を大きくすればするほど、長時間積分が行える範囲が広がっていることが分かる。この様な結果を得た事についての考察は次章で詳しく述べる。

第 6 章

まとめ

正規形の連立一階常微分方程式の初期値問題に対する精度保証付き数値計算について、文献 [1] の方法を平均値形式と組み合わせることによって、区間幅の広がりを抑えながら $x(b)$ の値を精度保証付きで求めることが本論文の目的であったが、第 5 章 2 節の数値実験の結果によってその目的が達成できた。アルゴリズムどおりに、真の値を含みつつ長時間積分が行えたことが確認でき、また平均値形式を用いる有効性を十分に示すことができた。また、Wrapping Effect を考慮することにより、さらに精度を高めることができることが示された。

次に、第 5 章 3 節の結果により、近似解を生成する際にその次数を大きくすれば、より精度が高くなることが示された。

これは論理的に考えても当然で、ある区間において、その区間での真の関数曲線を近似する際に、例えば直線となる 1 次関数で近似するよりも曲線となる 2 次関数、2 次関数よりも 3 次関数で近似した方が真の関数により近づくからである。しかしながら、あるところまで次数を大きくすると、それ以降は精度が悪くなっている。これは、次数を高くしたことによって、式 (3.11) などの計算において、あまりにも小さな数が結果として出るために情報落ちが起こり、正しい値を結果としていないためである。このプログラムは倍精度の浮動小数点システム (C++においての double 型) で実装している。よって、倍精度浮動小数点数の精度は 10 進数では約 16 桁であるから、それを超えた値については情報落ちが生じる。

次に、第 5 章 4 節の結果により、刻み幅を小さくすれば、より精度が高くなることが示された。

これは、刻み幅を小さくすることによって分割したそれぞれの区間が小さくなるので関数を近似する際により真の関数に近づくからであると考えられる。しかし、$h = 0.01$ からは精度が変わっていない。この数値実験において $h = 0.00001$ より小さい値のようなステップ数が 10 万を超えるような場合は、メモリ不足のために実行ができない。そのため、$h = 0.01$ からは精度が変わらないことについて、はっきりとした結論が出すことができなかった。

しかし理論的に考えると、刻み幅を小さくしていくと、次数のときと同様に、それによって式(3.11)などの計算においてあまりにも小さな数が結果として出るために情報落ちが起こり、正しい値が結果として得られずに精度が悪くなることが予想される。

次に、第5章5節の結果により,Wrapping Effectを考慮させる範囲を大きくすると、精度が悪くなるという結果が得られた。しかし、この結果は考慮する範囲を前2回としているときから、前述したような情報落ちが生じていると考えられる。Wrapping Effectを抑止するために、ステップ毎にかなり小さい値をもった行列を掛け合わせるので、これにより情報落ちがすぐに生じると考えられる。よって、理論的に考えてもそうだが、情報落ちが生じないのであれば、Wrapping Effectを考慮させる範囲を大きくすればするほど、精度が良くなることが期待できる。

以上のことから、今回のように倍精度の浮動小数点数システムで長時間積分を行う場合、どうすれば最も良い精度をもった数値解が得られるか、ということについて考える。第5章3,4節から、近似する次数を大きくし、刻み幅を小さくすれば精度を高められるということは証明された。しかしながら、実際に精度を高くしようと極端に次数を大きくしたり、極端に刻み幅を小さくしたり、また第5章5節からWrapping Effectを考慮させる範囲を大きくすると、上で述べたように、情報落ちが生じてしまい、正しい値を結果として得ることができない。よって、Wrapping Effectを考慮させる範囲を前2回とし、情報落ちが生じないで実行できるようなぎりぎりの次数、刻み幅を設定すれば、最も良い精度をもった数値解が得られる。

次に、第5章5節の結果により、できるだけ長い時間まで長時間積分を行うためには、近似解生成時の次数mを大きくすることが必要であることが分かった。

しかしながら、実際には、次数mを大きくしていくと前述したように情報落ちが生じてしまうので、正しい値を得られない。よって、形としてはあたかも長時間積分を続けているように見えるが、実際には正しい値に基づいて長時間積分を行っているのではなく、全

く関係のない値について長時間積分を行っていることになるので、意味がない。

また、この問題を別としても、次数を大きくすればするほど、その計算時間も跳躍的に長くなってしまうので、現実的に許される範囲の計算時間に抑えなければならない。

以上のことより、できるだけ長い時間まで長時間積分を行うためには、情報落ちが生じない範囲で、次数をぎりぎりまで大きくする必要がある。

最後に、これからの課題について述べる。まず、今回作成したプログラム上での問題としては、刻み幅を小さくしてステップ数を大きくすると、メモリ不足のため実行ができなくなるので、メモリ不足を解消するようにプログラムを改良する必要がある。また、プログラムの中で計算時間を短縮でき得る部分があると思われるので、その部分を改良し、計算時間をできるだけ短縮しなければならない。

また、根本的な課題として、今回の数値実験において大きな壁となった情報落ちの解消がある。

拡張倍精度の浮動小数点数システムを採用すれば、情報落ちを生じることなく計算が行える範囲が多少は広がることを予想できるが、これでは全面的な解決策とはならない。

とすれば、解決策として、計算結果の精度を大きく上げることができる多倍長演算が浮上してくる。しかしながら、多倍長演算を行うには多大な計算時間を要するので、計算時間を現実的に許される範囲に抑えるためには、多倍長演算は不適切である。

この通り、いまのところは具体的な解決策が見出せない。よって、計算時間を抑えつつ、情報落ちが生じないで計算ができるような解決策を見つけることを今後の大きな課題として取り組んでいきたい。

第 7 章

謝辞

本研究を進めるに当たり、終始懇切丁寧な御指導及び御激励を賜り、その他の多くの面でもいろいろと面倒を見て下さり御助言を与えて下さいました柏木 雅英助教授に深く感謝致します。

また、卒業論文中間報告の際など、御指導、御鞭撻を賜りました大石 進一教授、柏木研究室助手 相馬 隆郎氏に深く感謝致します。

また、本研究を進めるに当たり、その基本的なことに関する説明から本論文の作成に至るまで、終始懇切丁寧な御指導、御教示をして下さいました柏木研究室博士課程2年 宮田 孝富氏に深く深く感謝致します。

また、日常生活においていろいろとお世話になりました、柏木研究室修士課程1年 高崎 大輔氏、岩折 朱希嗣氏、また大石研究室の皆様に深く感謝致します。

また、同じ非線形班として意見の交換や協力をして下さいました柏木研究室4年 桜井 幹夫氏、金谷 卓充氏、川上 修氏、小泉 健氏、白井 健一氏、波多野 伸哉氏、深谷 光統氏、渡部 啓氏に深く感謝致します。

また、CG班でありながら御助言をして下さいました柏木研究室4年 洲濱 陽一氏、村竹 範彦氏、山田 浩之氏、吉田 直史氏に感謝申し上げます。

最後に研究だけでなく日頃の生活の中でお世話になりました柏木研究室の皆様に改めて深く感謝致します。

参考文献

[1] 柏木 雅英, 大石 進一：”ベキ級数演算を用いた常微分方程式の精度保証付き計算”, 電子情報通信学会技術研究報告,NLP94-91,pp.65-70,(1995-01).

[2] 大石 進一：”応用解析セミナー 数値計算”, 裳華房,1999.

[3] 柏木 雅英: ”精度保証付きシミュレーション [1]-区間解析-”, 日本シミュレーション学会”シミュレーション第 18 巻第 4 号 (平成 11 年 12 月)”.

[4] 水上 孝一 他 4 人の共著:”コンピュータによる数値計算”, 朝倉書店.