科学技術計算に適した多倍長数値計算環境の構築と数値的に不安定なスキームの直接計算の実現

藤原宏志†

偏微分方程式の高精度かつ大規模数値計算に適する高速な多倍長数値計算環境 exflib の設計と実装を行った．本計算環境は FORTRAN90 または C++ 言語から利用可能であり，ポリモルフィックなインタフェースを提供している．本論文では，大規模数値計算で多く利用される FORTRAN90 への対応とアセンブリ言語でのライブラリの設計について論じる．さらに，典型的な逆問題の数値計算において，FORTRAN 用の多倍長計算環境と比較して本計算環境の高速性とメモリ利用について示す．また，数値的に不安定なスキームに対して多倍長数値計算を利用することで，計算誤差の急激な増大に対する多倍長計算の有効性と数値解析理論への応用例を示す．

Design of a Multiple-precision Arithmetic Environment for Scientific Numerical Computations and Direct Computations of Numerically Unstable Schemes

Hiroshi Fujiwara†

We design and implement a fast multiple-precision arithmetic package 'exflib' for the purpose of large scale numerical computations of partial differential equations. The package are works with FORTRAN90 or the programming language C++ and main arithmetics are written in an assembly language. We give a remark on compatibility of programs implemented in an assembly language and FORTRAN90 compilers. Numerical results for a typical inverse problem are given to compare the proposed library with a FORTRAN multiple-precision arithmetic package. We also show an important application to numerically unstable problems and numerical analysis.

1. 緒　　言

科学技術計算では，実数は浮動小数点数によって近似して扱われ，それらの演算は浮動小数点演算とよばれる[1),2)]．標準的な数値計算環境では IEEE754 倍精度数が利用されており，その仮数部の精度は 10 進法では約 16 桁である[3)]．浮動小数点数による近似を丸め（rounding）とよび，丸めに際しては丸め誤差（rounding error）とよばれる誤差が混入する．浮動小数点演算では，演算結果を計算機内部で保持するためにも丸めが行われる．数値計算結果が信頼できるものであるためには，浮動小数点演算に特有の現象である丸め誤差とその増大，特に桁落ちや情報落ちに対する注意が必要である．

近年の計算力学では，非破壊検査に代表される逆問題（inverse problems）に対する数値的手法の重要性が高まっている．逆問題は多くの場合，Hadamard の意味で非適切な問題（ill-posed problems）となり，その直接離散化スキームの数値計算においては，計算過程で丸め誤差が著しく増大する数値的に不安定な問題（ill-conditioned problems）の数値計算が要求される．IEEE754 倍精度による数値的に不安定な問題の数値計算では，丸め誤差が急激に増大して数値計算は破綻し，直接的な手法では信頼できる数値計算は不可能であるとされていた．そこで逆問題の数値計算では，正則化法などの数学的安定化手法を利用した数値計算が行われている．正則化法は元の問題を Hadamard の意味で適切（well-posed）な問題で近似する手法で[4)]，問題を精度良く近似するためには正則化パラメータを小さく設定する必要がある．その場合，正則化法を適用して得られる方程式は数学的にはある位相で安定となるが，その離散スキームは数値的に不安定となる．そのため現在はパラメータを比較的大きな値に設定しての計算が行われるが，その計算結果では満足を得られない場合が多く，数値的に不安定なスキームに対す

† 京都大学大学院情報学研究科
Graduate School of Informatics, Kyoto University

る新たな手法の確立が嘱望されている．

丸め誤差の問題に対し，計算機上で実数を高精度で近似する手法はいくつか提案されているが，計算力学における数値計算を高速に実現するためには，仮数部の精度を多く確保する多倍長精度（multiple-precision）浮動小数点演算が適していると考えられる．浮動小数点演算では丸め誤差は不可避であるが，精度を任意に設定可能な多倍長計算環境では丸め誤差を任意に小さくすることができ，その増大が，数値計算の結果として要求される精度に達しなければ，仮想的に丸め誤差を除去した数値計算が実現される．

これまでにも多くの多倍長計算環境が提案されている．科学技術計算での利用においては，数論，代数学，暗号理論を目的としたものが多く，計算全体で利用するメモリ量は問題とならないが，数万桁を超える精度が要求されるため，高精度演算アルゴリズムの設計と実装が行われきた．これに対して本研究の多倍長計算環境は，数百桁から数千桁の精度で，計算力学に現れる偏微分方程式や積分方程式の数値シミュレーションの実現を目的としていることが特徴である．計算力学においては FORTRAN が多く利用されているが，既存の FORTRAN 多倍長計算環境を大規模問題に適用する際には利用するメモリ量も問題となることが指摘されており，高速性と利用メモリ量が問題となる大規模計算の視点からは，FORTRAN による多倍長計算環境の整備は十分とはいえなかった．

著者はこれまでに C++ 言語から利用可能な多倍長計算環境の設計と実装を行ったが[5]，本研究では計算力学での多倍長計算の利用を念頭におき，FORTRAN90 から利用可能なインタフェースの設計と実装を行った．次章において，計算環境の設計について論じ，アセンブリ言語による FORTRAN90 ライブラリの実装における注意を与える．本環境を既存の FORTRAN 多倍長計算環境として高速性に定評のある FMLIB を用いた大規模計算と比較したところ，高速な演算とメモリの節約を実現していることが確認された．これについて 3 章で，典型的な逆問題の数値計算について，先行研究と比較した結果を述べる．

計算力学においては高精度スキームの開発が重要な課題であるが，それが安定性を有さなければ，丸め誤差の増大によって数値計算が破綻する．この問題点に対して，4 章において，一階双曲型偏微分方程式を例に不安定な高精度スキームの直接計算による実現における多倍長計算の有効性について論じる．

2. 本研究での多倍長計算環境の設計と実装

2.1 多倍長数のデータ構造

提案する計算環境では，実数の近似に浮動小数点型を利用する．この浮動小数点型は指数を 1 ビット，基数を 2 として指数に 63 ビット，仮数の精度は暗黙の省略の 1 と合わせて $1+64n$ ビットを有し，メモリ中には**図 1** に示す 1 次元配列で保持する．配列の各要素 f_i は 64 ビット長符号なし整数であり，図 1 に示すデータ構造が表す値は

$$(-1)^s \times \left(1 + \sum_{i=1}^{n} f_i \times 2^{-64i}\right) \times 2^e$$

である．ここで e はメモリ中に保持されるバイアスつき指数 e_b に対して $e = e_b - (2^{62} - 1)$ である．

仮数部の精度は 10 進では約 $(1+64n) \times \log_{10} 2 \approx 19.3n$ 桁に相当し，精度を決定するパラメータ n はユーザ・プログラムのコンパイル時に決定される．IEEE754 倍精度および代表的な拡張精度型との比較を**表 1** に示す．

2.2 多倍長計算環境の構成

提案する計算環境は，四則や組み込み関数，丸めなどの演算を実現するライブラリ・ファイルと，FORTRAN90 または C++ 言語に対応するのインタフェース・ファイルの 2 つから構成される．円周率などの数学定数および 10 進法 2 進法変換に必要な定数を $1+64\times1023$ ビットの精度でライブラリ・ファイルの内部に保持している．この精度は 10 進法で約 19,709 桁に相当し，ライブラリ・ファイルのコンパイル時に決定する．組み込み関数については，C 言語で数学関数として提供されるものを利用可能である．

インタフェース・ファイルでは，FORTRAN90 と

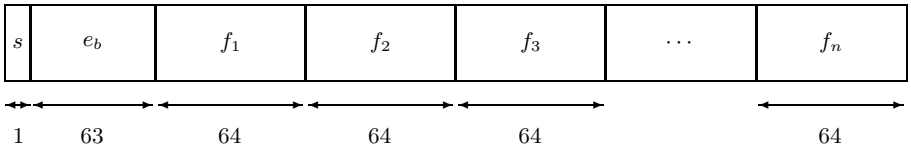

図 1 提案する多倍長精度・浮動小数点型のメモリ構造
Fig. 1 Memory structure of proposed multiple-precision floating point data type.

表 1 浮動小数点数の型
Table 1 Floating point types.

型	合計のビット幅	仮数部の精度*（ビット）	10 進法での仮数部の精度	指数部のビット幅	表現可能な値の範囲
IEEE754 倍精度	64	53	約 16.0 桁	11	10^{-308}〜10^{308}
IA32 拡張精度	80	65	約 19.6 桁	15	10^{-4932}〜10^{4932}
SPARC 拡張精度	128	113	約 34.0 桁	15	10^{-4932}〜10^{4932}
POWER 拡張精度	128	$\geq 106^{\dagger}$	約 31.9 桁以上	11	10^{-308}〜10^{308}
本設計の多倍長精度	$64+64n$	$1+64n$	約 $19.3n$ 桁	63	$10^{-10^{18}}$〜$10^{10^{18}}$

(*) 暗黙の省略の 1（hidden one）を含む.
(†) POWER の拡張精度数は 2 つの IEEE754 倍精度数の和で表される.

C++ 言語の多重定義によりポリモルフィックな演算子と関数を提供しており，四則演算や組み込み関数を利用する文は組み込みの型と同じ記法で実現される．そのため，特に数値計算ユーザにとっては使いやすいものとなっており，既存の単精度あるいは倍精度での利用を想定したユーザプログラムの移植性も高い．

これら四則演算や組み込み関数については，他の多倍長計算環境においてもポリモルフィズムによるインタフェースが提供されることが多いが，本環境の特徴は，多倍長型の変数への代入において，リテラル定数およびその四則演算を記述した文字列を利用可能な点である．FORTRAN の場合に具体例を示す．多倍長精度型変数 `A` に対して定数を代入するには，

```
A = 0.1
```

と記述するのが自然であると考えられる．しかし右辺の値はコンパイル時に単精度数または倍精度数と解釈されるため，多倍長数の `A` に要求される精度を有さない可能性がある．これに対し本環境では，定数の代入において，

```
A = '0.1'
```

のように，数値を文字列として代入することで，多倍長精度の定数の代入を実現している．右辺の文字列中には，10 進数，数学定数と四則演算を記述することが可能であり，

```
B = '#PI / 2'
C = '(1.23 + #E) * #G'
```

などの代入文が可能である．文字列中の式は，再帰下降法によりプログラム実行時に構文解析と演算が実行される．数学定数は，現在のところ円周率（`#PI`），自然対数の底（`#E`），Euler の定数（`#G`）を提供している．

2.3 演算アルゴリズムの設計と実装

本設計の計算環境は 100 桁から数千桁の精度での高速演算に適した浮動小数点演算アルゴリズムを採用している．多倍長数どうしの加減算[6] および多倍長数と整数の乗除算には，演算コストが $O(n)$ の筆算を利用している．多倍長数どうしの乗算には，精度が低い場合には $O(n^2)$ の筆算を，精度が高い場合には $O(n^{\log_2 3})$ の Karatsuba 法[7] を利用している．高速な乗算法の 1 つに高速 Fourier 変換に基づくアルゴリズムがあるが，上述の精度の範囲では Karatsuba 法が高速であった．また多倍長数どうしの除算には $O(n^2)$ の Ozawa の方法[8] を 64 ビット計算機に適した形に修正して利用している[5]．

本設計の計算環境は，64 ビットでの動作を念頭においており，アセンブリ言語によりアーキテクチャ固有の演算とフラグを利用して高速な演算を実現している．さらに，符号なし整数の 64 ビットすべてに有効な情報を保持し，省メモリでの多倍長数の表現が可能となる．32 ビット計算機やいくつかのアーキテクチャにおいては，精度と速度の観点から，浮動小数点数の演算を中心に利用するほうが有利であり，FORTRAN 用の多倍長計算環境である FMLIB ではこの設計をとっているが，[9] 演算の最小単位となる浮動小数点数の仮数部すべてに有意義な情報を保持することができず，メモリの利用において無駄が生じる．

2.4 FORTRAN90 の引数評価の非互換性

FORTRAN90 では 1 次元配列の引数評価がコンパイラによって異なる．すなわち，1 次元配列を引数とする際に，従来の FORTRAN77 と同様に配列の先頭アドレス（直接ポインタ）を利用するものと，配列の先頭アドレスを保持しているアドレス（間接ポインタ）を利用するものがある．これら 2 つの引数評価には互換性がなく，これは，ライブラリの生成とそれを利用するユーザ・プログラムのコンパイルに異なるコンパイラを利用すると計算結果が正しくならない場合があることを意味する．あらかじめコンパイルされて配布されるライブラリの利用や，FORTRAN90 と C 言語との間でのライブラリの相互利用に際しては注意が必要である．代表的なコンパイラの引数評価を**表 2** に示す．

本設計では，多倍長の演算演算を実現する関数の引数として，演算数などに図 1 の 1 次元配列を，また，

表 2 コンパイラの引数評価
Table 2 Argument evaluations of compilers.

コンパイラ（ベンダ）	スカラ型	1 次元配列
FORTRAN90 (GNU, Intel, Sun)	参照渡し	間接ポインタ渡し
FOTRAN90 (Fujitsu, IBM, PGI)	参照渡し	直接ポインタ渡し
FORTRAN77	参照渡し	直接ポインタ渡し
C 言語，C++言語	値渡し	直接ポインタ渡し

```
op_f90_:
    参照渡し / 間接ポインタ仕様のエントリ

    1 次元配列型を，間接ポインタから
    直接ポインタに変換する．
    (配列の先頭アドレスで上書きする．)

op_f77_:
    参照渡し / 直接ポインタ仕様のエントリ

    スカラ型を参照渡しから値渡しに変換する．
    (ポインタの内容でアドレスを上書きする．)

op_c:
    値渡し / 直接ポインタ仕様のエントリ

    プログラムの本体．
    1 次元配列は直接ポインタ渡しとして，
    スカラ型は値渡しとして記述する．

    return
```

図 2 演算プログラムの複数の引数仕様への対応
Fig. 2 Prototype for three argument evaluation manners.

```
C VEND_A, VEND_B, VEND_C, VEND_D は
C コンパイラ固有の識別子であるとする

C 間接ポインタ渡し仕様のコンパイラ
#if (defined VEND_A) || (defined VEND_B)
#define INTERFACE 90
#endif

C 直接ポインタ渡し仕様 (FORTRAN77 と互換)
#if (defined VEND_C) || (defined VEND_D)
#define INTERFACE 77
#endif

C 適切なエントリに置換する
#if (INTERFACE == 90)
#define op_FRT op_f90
#elif (INTERFACE == 77)
#define op_FRT op_f77
#else
#error Unkown Compiler.
#endif

C 型宣言
  INTEGER*8 :: int
  INTEGER*8, DIMENTION(10) :: array
  INTERFACE
    SUBROUTINE op_FRT(s_arg, v_arg)
      INTEGER*8 :: s_arg
      INTEGER*8, DIMENSION(:) :: v_arg
    END SUBROUTINE op_FRT
  END INTERFACE

C 演算の呼び出し例 : op_FRT は実際には
C 存在せず，プリプロセサにより
C op_f90 または op_f77 に置換される．

  CALL op_FRT(int, array)
```

図 3 FORTRAN90 でのプリプロセサによる関数名の置換
Fig. 3 Name binding by preprocessor in FORTRAN90.

演算精度の指定に整数型を利用する．表 2 にあげたコンパイラから利用可能であるためには，3 種類の仕様に対応する必要がある．これには，コンパイル時オプションで引数評価仕様を指定する，あるいはコンパイラごとにライブラリ・ファイルおよびインタフェース・ファイルを作成する方法が考えられるが，本計算環境ではユーザの利便性を考慮して，アセンブリ言語で提供する演算関数において各引数評価仕様に対応する複数のエントリを設け，プリプロセサを利用して適切なエントリを結び付ける手法を採用した．

演算を実現する関数では，表 2 にあげた 3 種類の仕様に対応するエントリをプログラムの先頭に設ける（**図 2**）．図 2 を例に詳述すると，この演算関数は 3 つのエントリ op_f90_，op_f77_，op_c を有する．最初のエントリの後では 1 次元配列引数に対して間接ポインタを直接ポインタに変換し，次のエントリの後ではスカラ型引数を参照渡しから値渡しに変換する．プログラムの本体は最後のエントリに続いて記述され，1 次元配列引数は直接ポインタで，スカラ型引数は値渡しとして処理する．

ユーザが多倍長演算を行う際には，インタフェース・ファイルで定義されるポリモルフィックな演算子をとおして上述の演算関数が呼び出される．演算関数は利用するコンパイラに対応するエントリから実行されなければならず，演算子が適切なエントリを選択する

表 3 多倍長計算環境の演算速度（単位：マイクロ秒）
Table 3 Execution time of arithmetic (unit: micro sec.).

10 進法での桁数と演算	Maple	Mathematica	mpfun90	FMLIB	MPFR	Pari	本設計
100 桁，乗算	22	1.5	1.7	1.1	0.24	0.29	0.14
除算	26	5.9	2.3	1.4	0.52	1.2	0.33
1000 桁，乗算	100	17	60	17	6.2	15	3.7
除算	93	50	63	20	16	23	8.3
10000 桁，乗算	3400	580	na	1105	300	1400	280
乗算	3000	1800	na	1251	730	1500	630

計算環境：Opteron246 (2.0 GHz)．mpfun90，FMLIB は Opteron150 (2.4 GHz) を使用，na: not available

表 4 MPI 並列計算における Gauss 消去法に必要な計算時間（単位：秒）
Table 4 Execution time of MPI program for Gauss elimination with pivotting (unit: sec.).

10 進法での桁数	2000				3000			
行列のサイズ	200	400	800	1000	200	400	800	1000
1 CPU 使用	41	311	2493	4920	87	649	5140	10029
2 CPU 使用	23	161	1263	2462	49	337	2614	5051
4 CPU 使用	15	87	651	1254	31	182	1334	2578
8 CPU 使用	12	54	349	660	23	106	710	1339
16 CPU 使用	9.5	35	193	365	19	73	389	725

計算環境：Opteron246 (2.0 GHz)，MPI

必要がある．C++ 言語では図 2 の op_c を演算関数として呼び出す．FORTRAN90 では，プリプロセサにより利用しているコンパイラを判別して，呼び出す演算関数の名前をコンパイラの引数仕様に適したエントリ op_f90 あるいは op_f77 に置換する．プリプロセサは多くの FORTRAN90 コンパイラでサポートされており，コンパイラの判別に利用するコンパイラ固有の識別子と引数仕様は FORTRAN90 のプログラムに埋め込む形でインタフェース・ファイルに記述される（**図 3**）．

上述の手法では，コンパイラごとに複数のファイルを用意する必要はなく，ラッパ（wrapper）を利用する手法に比して呼び出しにかかるオーバヘッドが解消される．プリプロセス作業はユーザ・プログラムのコンパイル作業の一部として自動的に行われ，引数仕様を指定するコンパイル・オプションも不要である．

2.5 対応するアーキテクチャと演算速度

本設計の多倍長計算環境は，AMD64 および EM64T で 64 ビットのライブラリとして動作する．また，パーソナル・コンピュータとして普及している IA32 アーキテクチャでは 32 ビットライブラリを提供している．UNIX において GCC（GNU Compiler Collection）といくつかの商用コンパイラでの動作を確認している．また Windows オペレーティングシステムにおいては商用の C++ 言語コンパイラと GCC での動作を確認している．

64 ビット・アーキテクチャの 1 つである Opteron プロセサにおいて，他の多倍長計算環境との速度の比較を**表 3** に示す[11]．本研究で提案する環境が，他と比較して高速な演算を実現していることが分かる．

本環境は MPI または OpenMP による並列計算も可能である．Hilbert 行列[12] に対して軸選択つき Gauss 消去法に要した計算時間を**表 4** に示す．10 進法で 3000 桁，1000 次元の問題において，1CPU での逐次実行と 16CPU での並列計算の計算時間の比は $10029/725 = 13.8$ であり，並列計算が有効であることが分かる．

なお，本提案の多倍長計算環境 exflib は，インターネットを介して公開されており，利用可能である[13]．

3. 多倍長計算による大規模計算における演算時間と使用するメモリ量

本環境は大規模計算での利用を念頭に置いており，演算速度の比較では四則などの個々の演算の速度だけでなく，具体的な問題の数値計算に要する計算時間も合わせて論じることが計算力学の視点から必要であると考えられる．また，大規模問題への適用においては，必要とするメモリ量の検討も重要である．本章では，熱伝導に関連する典型的な逆問題の数値計算[14] をベンチマークとして採用し，提案する環境 exflib の高速性と使用するメモリ量を論じる．

採用したベンチマーク問題は，次の偏微分方程式

$$\frac{\partial u}{\partial t} = -\frac{\partial^2 u}{\partial x^2}, \quad 0 < t < 1, -1 < x < 1, \tag{1a}$$

$$u(0, x) = \cos\frac{\pi}{2}, \quad -1 < x < 1, \tag{1b}$$

$$u(t, -1) = u(t, 1) = 0, \quad 0 < t < 1 \tag{1c}$$

表 5 熱伝導逆問題の数値計算に要する時間（精度 10 進 150 桁，単位：秒）
Table 5 Execution time of Backward Heat Equation with 150 decimal digits (unit: sec.).

スペクトル次数	連立方程式の次元	計算環境 A	計算環境 B	計算環境 C	計算時間の比 ($\frac{B}{C}$)
30	961	1879	284	78	3.64
40	1681	10581	1503	428	3.51
50	2601	40285	5529	1624	3.40
60	3721	—	16169	4815	3.36
70	5041	—	40296	11915	3.38
80	6561	—	89247	26900	3.32
90	8281	—	191380	54712	3.50

計算環境 A：FMLIB，Xeon (2.0 GHz)
計算環境 B：FMLIB，Dual Xeon (2.0 GHz) 3 台と Dual Xeon (2.4 GHz) 1 台の 8CPU による並列計算
計算環境 C：exflib，Opteron246 (2.0 GHz)

に対する Chebyshev 多項式に基づくスペクトル選点法による離散化問題の数値計算である．そこでは，離散化におけるスペクトル次数を N とすると，$(N+1)^2$ 次の連立一次方程式の解法が数値計算の中心となる．系 (1) は Hadamard の意味で非適切であり，その離散化は数値的に不安定である．ここでは高精度離散化手法であるスペクトル法と多倍長計算の組合せにより，不安定スキームの直接計算を実現している．

スペクトル選点法による数値計算に要した計算時間を**表 5** に示す．計算環境 A および B は先行研究[14] からの引用で，多倍長計算には FMLIB が用いられている．計算環境 C が本研究で提案する exflib での数値計算を 1CPU で実行したときの計算時間である．計算の精度は，いずれも 10 進法で 150 桁である．計算環境 B では PVM による 8CPU の並列計算によって計算の高速化と大規模数値計算の実現に成功しているが，exflib による計算環境 C では，計算環境 B に比してさらなる高速化が実現されていることが分かる．

メモリ量について論じると，10 進法で 150 桁の数の情報量は約 500 ビットに相当し，3721 次，6561 次の行列の保持に必要なメモリ量を理論的に見積もるとそれぞれ約 0.8 GB，約 4.0 GB となる．一方，計算環境 A では $N = 60$ に対する数値計算において 4 GB ではメモリが不足することが報告されており，exflib を利用した計算環境 C では，$N = 60$，$N = 90$ の数値計算に要したメモリはそれぞれ約 1.0 GB，4.6 GB であり，メモリの利用においても FMLIB に比して exflib が有利であるといえる．

多倍長計算による大規模数値計算では，計算時間と必要とするメモリ量が問題となるが，本研究で提案する exflib は，従来環境に比して少ないメモリ量で，高速な計算を実現していることが分かる．

4. 数値的に不安定な計算スキームの直接計算

4.1 定数係数の一階双曲型方程式の収束性と安定性

本章では，一階双曲型方程式の初期値問題

$$\frac{\partial u}{\partial t}(t,x) = \frac{\partial u}{\partial x}(t,x), \quad u(0,x) = u_0(x) \qquad (2)$$

に対し，収束性を有するが安定ではない高精度差分スキームによる数値計算に多倍長計算を行い，有効性を示す．この例は比較的小規模のものであるが，本章で述べる不安定スキームの直接計算を大規模計算に適用するには，前章で示したとおり，本研究で提案する高速な多倍長計算が有効である．

系 (2) に対する離散化は，時間方向には離散化パラメータ Δt の前進差分を利用するものとし，空間方向の離散化に離散化パラメータ Δx の前進差分を適用すると，

$$\frac{u^{k+1}(x) - u^k(x)}{\Delta t} = \frac{u^k(x+\Delta x) - u^k(x)}{\Delta x}, \qquad (3)$$

空間方向に中心差分を適用すると，

$$\frac{u^{k+1}(x) - u^k(x)}{\Delta t} = \frac{u^k(x+\Delta x) - u^k(x-\Delta x)}{2\Delta x}, \qquad (4)$$

空間方向に後退差分を適用すると，

$$\frac{u^{k+1}(x) - u^k(x)}{\Delta t} = \frac{u^k(x) - u^k(x-\Delta x)}{\Delta x} \qquad (5)$$

を得る．ここで $u^k(x)$ は $u(k\Delta t, x)$ 相当値である．

式 (2) に対する厳密解は $u(t,x) = u_0(t+x)$ で与えられるが，解析的な初期値 $u_0(x)$ に対しては，いずれの差分方程式の厳密解も，Δt，$\Delta x \to 0$ のとき，数値解が厳密解に広義一様収束することが Hayakawa によって証明されている[15]．一方で安定性を有するのは，特性曲線を考慮した式 (3) において $\Delta t/\Delta x \leq 1$ のときのみであり，これ以外では L^2 あるいは L^∞ の枠組みでの安定性は保証されない．

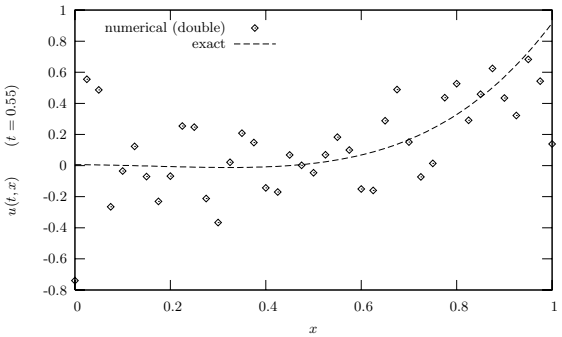

図 4 定数係数 (2) の中心差分 (4) による倍精度での計算結果（◇：数値解，破線：厳密解）

Fig. 4 Results with double precision for constant coefficient (2) with central difference(4) (◇: numerical solution, dotted line: exact solution).

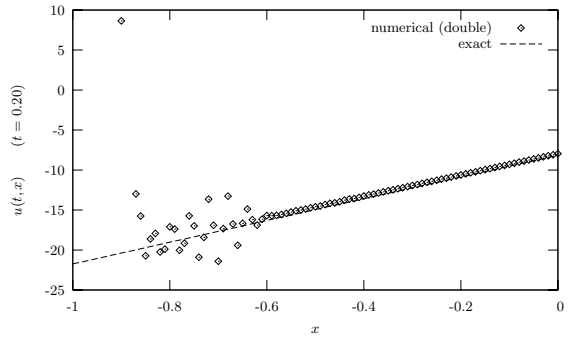

図 6 変数係数 (6) の中心差分 (7) による倍精度での計算結果（◇：数値解，破線：厳密解）

Fig. 6 Results with double precision for variable coefficient (6) with central difference (7) (◇: numerical solution, dotted line: exact solution).

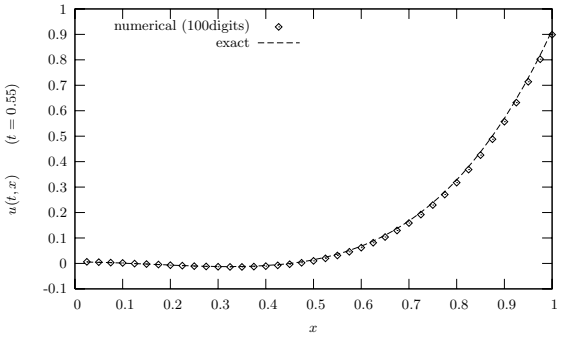

図 5 定数係数 (2) の中心差分 (4) による 100 桁での計算結果（◇：数値解，破線：厳密解）

Fig. 5 Results with 100 decimal digits for constant coefficient (2) with central difference (4) (◇: numerical solution, dotted line: exact solution).

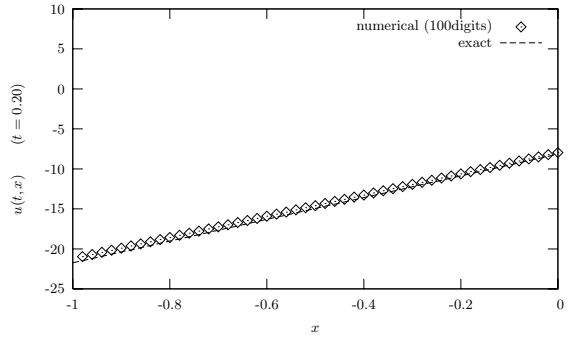

図 7 変数係数 (6) の中心差分 (7) による 100 桁での計算結果（◇：数値解，破線：厳密解）

Fig. 7 Results with 100 decimal digits for variable coefficient (6) with central difference (7) (◇: numerical solution, dotted line: exact solution).

実際，

$$u_0(x) = x\left(x-\frac{1}{3}\right)\left(x-\frac{2}{3}\right)(x-1)$$

として $\Delta t = \Delta x = 0.005$ にとった場合，中心差分 (4) をもちいて倍精度で数値計算を行うと $t = 0.55$ において**図 4** に示す数値解を得る．図中，◇ が示す数値解は，破線で示す厳密解とは著しく異っている．一方，同一の離散化パラメータ下で，本研究で提案する多倍長計算により 100 桁の精度で $t = 0.55$ で得られる数値計算結果を**図 5** に示す．この場合，厳密解を十分に近似する数値解が得られており，図 4 に現れた数値計算の破綻は丸め誤差に起因することが分かる．

空間方向の離散化について論じると，中心差分 (4) の離散化誤差は $O(\Delta x^2)$ で，前進差分と後退差分の $O(\Delta x)$ に比して高精度スキームを与えるが，数値計算では安定性の欠如のために倍精度計算では数値計算が破綻する．これに対し，多倍長計算を用いることで，中心差分による高精度スキームでの数値計算が実現される．

4.2 変数係数の一階双曲型方程式の数値計算

定数係数である式 (2) では，特性曲線が一定の方向を向いており，上流（up-wind）スキームを 1 つ採用すれば安定した数値計算を行うことが可能である．しかし変数係数の場合には特性方向に応じて差分公式や Δt と Δx の比を適切に選ぶ必要があり，複雑になる．

例として

$$\frac{\partial u}{\partial t}(t,x) = (5x-3)\frac{\partial u}{\partial x}(t,x), \tag{6a}$$

$$u(0,x) = 5x-3 \tag{6b}$$

を考える．この場合，厳密解は $u(t,x) = (5x-3)e^{5t}$ で与えられる．式 (6) の数値計算において，時間方向に前進差分を，空間方向には依存領域が最も広く，高

精度な中心差分を適用すると，次の差分スキームを得る．

$$\frac{u^{k+1}(x)-u^k(x)}{\Delta t} = (5x-3)\frac{u^k(x+\Delta x)-u^k(x-\Delta x)}{2\Delta x} \quad (7)$$

このスキームにより，$\Delta t = \Delta x = 0.01$ として $t = 0.2$ における倍精度での数値解を**図 6** に，100 桁の精度での数値解を**図 7** に示す．倍精度では丸め誤差の影響のため数値計算が破綻しているが，多倍長計算では厳密解を十分に近似する数値解が得られる．

Hayakawa は定数係数の場合に収束証明を与えるもので，変数係数の式 (6) に対する収束証明は現在までに与えられていないが，100 桁での数値計算結果を評価すると，解析関数の範疇においては，変数係数の方程式においても Hayakawa の結果と同様の広義一様収束性が成立することが，数値例をともなって示唆される．この数値解析理論に対する示唆は，倍精度の数値計算では丸め誤差の増大のために不可能であったことである．

5. 結　　言

本論文では，大規模数値計算の高精度数値シミュレーションを念頭においた高速な多倍長数値計算環境の設計と実装を行い，数値的に不安定なスキームの数値計算における多倍長計算の有効性を確認した．本環境は，典型的な逆問題の大規模計算において，従来環境に比して少ないメモリ量での高速な演算を実現していることを示した．これは大規模数値計算に多倍長計算を適用するうえで本質的な改善であり，従来の多倍長計算では計算時間とメモリ量の制約から実現不可能とされていた大規模数値計算を，exflib により実現しうることを示している．

具体的な逆問題への適用については，問題ごとの特徴を利用した離散化手法と合わせて論じる必要がある[16]．現在，著者を含む研究グループにより，逆散乱問題の数値解析において，解析関数を核に持つ第 1 種積分方程式の多倍長計算が進められており，従来には得られなかった高精度な数値計算を短時間で実現することに成功している[17]．

多倍長計算では，多倍長数の記憶に大容量のメモリが必要であり，演算に多くの CPU 時間が要求される．計算力学に現れる大規模数値計算の実現には，並列計算が必要であり，精度に応じて多倍長数の演算と通信コストの比が異なることを考慮した新たな指標が必要となる．多倍長計算は，演算理論の立場からの高精度計算の手法であり，実際の数値計算では種々の誤差に対処する必要がある．本研究における高速な多倍長計算に加えて数学的安定化法，高精度離散化法の個々の研究だけでなく，これらを併用する数値解析理論と手法の進展は，計算力学の種々の問題に対して信頼性の高い数値解析手法の実現に寄与するものと考えている．

謝辞 本研究の遂行にあたり，京都大学の磯祐介教授に貴重なご助言をいただきました．また日本学術振興会科学研究費（課題番号 17740057，16340024，17654023）および京都大学学術メディアセンタースーパーコンピュータ共同研究制度の助成をいただきました．

参 考 文 献

1) Goldberg, D.: What every computer scientist should know about floating-point arithmetic, *ACM Computing Surveys*, Vol.23, pp.5–48 (1991).
2) Knuth, D.E.: *The Art of Computer Programming Volume 2 : Semi Numerical Algorithms*, 3rd edition, Addison-Wesley (1998).
3) IEEE Standard for Binary Floating-Point Arithmetic: ANSI/IEEE std 754-1985 (1985). Reprinted in *SIGPLAN* 22, pp.9–25 (1987).
4) Tikhonov, A.N. and Arsenin V.: *Solutions of ill-posed problems*, Wiley (1977).
5) 藤原宏志，磯　祐介：64 bit 計算環境に適した多倍長数値計算環境の構築と非適切問題の数値計算，情報処理学会論文誌，Vol.44, pp.925–931 (2003).
6) Goldberg, D.: *Computer Arithmetic* (1996). Appendix in 18).
7) Karatsuba, A. and Ofman, Y.: Multiplication of multidigit numbers on automata, *Soviet Physics Doklady*, Vol.7, pp.595–596 (1963). (English translation).
8) Ozawa, K.: A fast $O(n^2)$ division algorithm for multiple-precision floating-point numbers, *J. of Information Processing*, Vol.14, pp.354–356 (1991).
9) Smith, D.M.: A Fortran package for floating-point multiple-precision arithmetic, *Transactions on Mathematical Software*, Vol.17, pp.273–283 (1991).
10) Bailey, D.H: the Fortran-90 Multiprecision System. https://crd.lbl.gov/~dhbailey/mpdist
11) http://www.medicis.polytechnique.fr/~pphd/mpfr/timings.html
12) Higham, N.J.: *Accuracy and Stability of Numerical Algorithms*, SIAM (2002).
13) http://www-an.acs.i.kyoto-u.ac.jp/~fujiwara/exflib/index.html

14) 竹内敏己，今井仁司，磯　祐介：熱伝導方程式に関する逆問題の無限精度並列数値シミュレーション，第 52 回理論応用力学講演会，講演論文集，pp.197–198 (2003).
15) Hayakawa, K.: Convergence of finite difference scheme and analytic data, *Publ. Res. Inst. Math. Sci.*, Vol.24, pp.759–764 (1988).
16) Imai, H. and Takeuchi, T.: Some Advanced Applications of the spectral collocation method, *GAKUTO Internat. Ser. Math. Sci. Appl.*, Vol.17, pp.323–335 (2002).
17) Fujiwara, H. and Iso Y.: *Application of multiple-precision arithmetic to direct numerical computation of inverse acoustic scattering* (submitting).
18) Patterson, D.A. and Hennessy, J.L.: *Computer Architecture: A Quantitative Approach*, 2nd edition, Morgan Kaufmann (1996).

(平成 18 年 10 月 10 日受付)
(平成 19 年 2 月 21 日採録)

藤原　宏志

昭和 51 年生．平成 11 年京都大学理学部卒業．平成 15 年同大学大学院情報学研究科博士後期課程修了．京都大学博士（情報学）．現在，京都大学大学院情報学研究科助手．非適切問題の数値解析，多倍長数値計算の研究に従事．日本数学会，日本応用数理学会に所属．