偏微分方程式の解の計算機援用存在証明法のための C++を用いた精度保証付き数値計算ライブラリの構築

関根 晃太[1,a)]

概要：数値計算とはコンピュータを使って数学的に記述された手では解けない問題を近似的に解く手法であり，現代の科学には欠かせない技術である．しかし，近似的にしか解が求まらないため，正しい解に近いかどうかさえわからない．精度保証付き数値計算とは数値計算で発生する全ての誤差を把握し，数学的に正しい結果を数値計算によって得ることをいう．即ち，IEEE 754 で規格化された浮動小数点数の計算で発生する丸め誤差から Newton 法などの反復解法の打切り誤差，さらには有限要素法などの偏微分方程式の近似解を求める際に発生する離散化誤差まで全ての誤差を把握して計算する手法である．さらに，数学的に解がわかっていない問題に適応すれば，計算機を用いた数学の証明にもなる．しかし，精度保証付き数値計算は浮動小数点数の丸めモードの制御のプログラムから数学の知識まで必要になるため気軽にユーザーは利用しにくい状況にある．そこで本講演では，計算機を用いた偏微分方程式の解の存在証明を最終目的とした数値線形代数ライブラリの構築法と著者が作成した VCP Library について紹介する．

キーワード：精度保証付き数値計算，計算機援用存在証明法，C++11，VCP Library

1. はじめに

数値計算とはコンピュータを使って，数学的に記述された問題を解く手法であり，現代の科学には欠かせない技術である．一般的に利用するコンピュータは実数を IEEE 754 Standard [1] で定められた浮動小数点数に近似して演算を行う．例えば，実数「1.2」は IEEE 754 Standard で定められている倍精度浮動小数点数では真の値で表すことが出来ないため近似される．また，浮動小数点数同士の演算結果も浮動小数点数で表せない場合，切り捨てや切り上げによって近似することで浮動小数点数にする．例えば「1」も「3」も倍精度浮動小数点数で記述可能だが，「1/3」の結果は倍精度浮動小数点数で記述することはできない．このように浮動小数点数の演算によって発生する誤差を丸め誤差という．そのため，浮動小数点数の演算は非常に高速である反面，演算を経て得られた結果は丸め誤差を含む近似値となる．

数値計算を行う上では，丸め誤差の他にも様々な誤差がある．例えば，コンピュータを用いて積分値を求めたい場合に利用する数値積分法は

$$\int_a^b f(x)dx \approx \sum_{i=1}^{N} f(x_i)w_i$$

[1] 東洋大学情報連携学科
[a)] sekine123@iniad.org

のように積分を有限和に近似して計算を行う．また，非線形方程式 $f(x) = 0$ の解を求める際に良く利用される Newton 法

$$x_{i+1} = x_i - f'[x_i]^{-1} f(x_i)$$

は「実数計算」を「無限回」繰り返すことによって真の解が得られる．しかし，コンピュータでは「浮動小数点演算」を「有限回」繰り返した結果しか得られないため，真の解が得られているわけではない．

このようにコンピュータで得られた結果は必ずしも正しいわけではない．そのために近似解がどの程度正しい結果を得ているのか確かめる手段が必要であり，さらに正しさを確かめた結果，必要に応じて近似解の改善も必要となる．誤差が含まれた近似解がどの程度正しい結果を得ているか確かめる手段として精度保証付き数値計算法がある．精度保証付き数値計算とは，連立一次方程式をはじめとした固有値問題，積分，微分方程式などの解きたい問題の真の解 x^* と近似解 $\hat{x}$ に対し，コンピュータを用いて数学的に厳密な誤差の上界を得る手法である．

さらに，精度保証付き数値計算法は非線形偏微分方程式の境界値問題などにも適応される．非線形偏微分方程式は無限次元問題でありコンピュータではそのまま計算できない．そのため，有限要素法などの方法を用いて離散化し，有限次元問題に近似してからコンピュータで計算する．また，非線形偏微分方程式の真の解も無限次元になり，真の解がそもそも存在するか解析的に証明されていない問題も多くある．精度保証付き数値計算では，無限次元問題から有限次元問題に離散化した際の離散化誤差まで考えなければならない．そのために有限要素法などを利用して離散化し，コンピュータで得られた近似解に加え関数解析の知識を利用することで，「コンピュータを用いて非線形偏微分方程式の解の存在性を証明」しつつ，「無限次元問題の真の解 u^*」と「有限次元問題の近似解 $\hat{u}$」に対し数学的に厳密な誤差の上界を与える．このことから，偏微分方程式の解に対する精度保証付き数値計算はしばしば「計算機援用存在証明法」とも呼ばれる．

しかし，偏微分方程式の解に対する計算機援用存在証明法は有限次元問題に対する精度保証付き数値計算法の結集であり，プログラムが非常に複雑になってしまう．そのために，精度保証付き数値計算ライブラリを利用することで好ましい．現状ある有名な精度保証付き数値計算ライブラリは以下の 2 つである:

S.M. Rump, INTLAB, 言語: MATLAB [2] 数値線形代数に対する精度保証付き数値計算法の専門家である S.M. Rump 氏が作成したライブラリ．MATLAB に実装されている BLAS と LAPACK を利用した高速な行列，数値線形代数に関わる精度保証付き数値計算が強力．

柏木雅英, kv Library, 言語: C++ [3] 常微分方程式の解に対する精度保証付き数値計算法が専門家である柏木雅英氏が作成したライブラリ．常微分方程式の精度保証付き数値計算法が実装されている．また，Double-Double 演算 (疑似 4 倍精度演算) や MPFR を用いた高精度なスカラー計算に対する精度保証付き数値計算法が実装されている．

その上で，偏微分方程式の解に対する計算機援用証明法では，「数値線形代数に関わる高速な精度保証付き数値計算」と「高精度なスカラー計算に対する精度保証付き数値計算法」に加え「数値線形代数に関わる高精度な精度保証付き数値計算」が必要になる．そのため現状では，無限次元を扱うための数学である関数解析の知識のみならず，有限次元の様々な精度保証付き数値計算までも把握できる研究者しか偏微分方程式の解に対する計算機援用証明法に関するプログラムを構築することは難しい．そのために本論文では偏微分方程式の解に対する計算機援用証明法で必要となる数値線形代数ライブラリを以下のように定義する:

定義 1 (応用上で必要な数値線形代数ライブラリ)

- 様々な型で行列計算，連立一次方程式，固有値問題を扱える
- double 型は最適化された BLAS, LAPACK を利用し高速に実行できる

- 上記 2 つに対して精度保証付き数値計算も可能
- すべてを変数の型のみの変更で切り替えが可能

本論文では，定義 1 を満たす数値線形代数ライブラリの作成する手段として，Policy-based design を用いて作成する．また，応用として作成したライブラリを用いて実際に偏微分方程式の解に対する計算機援用証明法を行う．さらに，Policy-based design の特徴を生かした数値線形代数の新しい開発モデルを提案する．本論文で紹介するプログラムは VCP Library として http://verified.computation.jp/に公開しているため，詳細を知りたい場合は参照されたい．VCP Library では C++11 を仮定しており，またスカラーの計算に関しては柏木雅英氏が作成した kv Library を利用することも想定して作成している [3]．

第 2 章では，精度保証付き数値計算に関する既存の結果として，「スカラーに対する精度保証付き数値計算の基礎である機械区間演算」と，「スカラーに対する精度保証付き数値計算の限界とその解決策である事後誤差評価」について紹介する．第 3 章では，定義 1 を満たす数値線形代数ライブラリの設計方法について紹介する．第 4 章では，Policy-based design の特徴を生かした数値線形代数の新しい開発モデルについて提案する．第 5 章では，実際に Emden 方程式の定常解として非自明解が存在することを計算機を用いて証明する．

2. スカラーや行列の演算に関する精度保証付き数値計算の既存の結果

ここではスカラーや行列の演算に関する精度保証付き数値計算の既存の結果を示す．詳細については書籍 [4] を参考にされたい．

2.1 倍精度浮動小数点数における機械区間演算

$\mathbb{R}$ を実数全体の集合とし，$\mathbb{F}$ を IEEE 754 Standard で定められた倍精度浮動小数点数の集合とする．但し，オーバーフローは起こらないと仮定する．また，閉区間を $[a,b] := \{x \in \mathbb{R} \mid a \le x \le b,\ a,b \in \mathbb{R}\}$ と表記する．

精度保証付き数値計算の基本は真の値を包含する集合を作成することである．例えば 10 進数のイメージでは，円周率 $\pi = 3.14158\cdots$ のような実数は閉区間を用いて $\pi \in [3.14, 3.15]$ のように真の値を包含する区間 $[3.14, 3.15]$ を作成する．また演算においても，$1/3 \in [0.3333, 0.3334]$ のように真の解を包含する閉区間を作成する．このような，閉区間をコンピュータで作成するには以下の 2 つの技術が必要である:

- 丸めモードの制御
- 区間演算

まず, 丸めモードの制御に関して紹介する．IEEE 754 Standard では「浮動小数点数の四則演算」，及び「平方根」の 5 つの演算結果に限り結果に対する切り捨てや切り上げなどの丸めに関するモードが定められている．即ち，sin や cos などの初等関数に関しては利用できないので，別の方法が必要になるので注意が必要である．

例えば $a,b \in \mathbb{F}$ としたとき，四則演算は以下の通りである：

上向き丸め: $\overline{\circ} : \mathbb{F} \times \mathbb{F} \to \mathbb{F}$ とし，$\inf\{x \in \mathbb{F} \mid x \ge a \circ b\}$

下向き丸め: $\underline{\circ} : \mathbb{F} \times \mathbb{F} \to \mathbb{F}$ とし，$\sup\{x \in \mathbb{F} \mid x \le a \circ b\}$

最近点丸め: $\tilde{\circ} : \mathbb{F} \times \mathbb{F} \to \mathbb{F}$ とし，最も近い浮動小数点数に丸める．

例えば，以下のように C99 準拠の C 言語コンパイラでは fenv.h を使用することで，丸めモードの変更を行う fesetround 関数が利用できる:

```
#include<fenv.h>

int main(void) {
 // 下向き丸めに変更
 fesetround(FE_DOWNWARD);
 // 上向き丸めに変更
 fesetround(FE_UPWARD);
 // 最近点丸めに変更
 fesetround(FE_TONEAREST);
}
```

一方で，丸めモードの変更が簡単ではない言語や，そもそも丸めモードの変更を行うことを禁止している言語があるため注意が必要である．

丸めモードの変更を用いることで例えば 1/3 の

結果を包含する区間 $[a,b]$, $a,b \in \mathbb{F}$ は次のようなプログラムで得られる:

```
#include<fenv.h>

int main(void) {
 double a, b;
 fesetround(FE_DOWNWARD);
 a = 1.0/ 3.0;
 fesetround(FE_UPWARD);
 b = 1.0/ 3.0;
}
```

但し，上記のプログラムはコンパイラの最適化によりプログラムの実行順序が変更されないことが前提となる[*1]．

このように，浮動小数点数同士の演算であれば丸めモードの変更により目的とする真の値を包含する閉区間を得ることが出来る．次に，閉区間同士の演算である区間演算を定義する必要がある．例えば，2/3 を包含した閉区間を $[0.666, 0.667]$，π を包含した閉区間を $[3.14, 3.15]$ としたとき，$2/3 \times \pi$ を得るには閉区間同士の積 $[0.666, 0.667] \times [3.14, 3.15]$ はどの様に定義するかということである．

2 つの閉区間 $[a_l, a_u]$, $[b_l, b_u]$ に対する機械区間演算を導入する．四則演算 $f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ に対し，値域 $f([a_l, a_u], [b_l, b_u]) = \{z \in \mathbb{R} \mid z = f(x, y),\ x \in [a_l, a_u],\ y \in [b_l, b_u]\}$ を包含する閉区間 $[c_l, c_u]$ を求めることが基本である．以下が機械区間四則演算の定義とする:

加算： $[a_l,\ a_u] + [b_l,\ b_u] := [a_l \underline{+} b_l,\ a_u \overline{+} b_u]$

減算： $[a_l,\ a_u] - [b_l,\ b_u] := [a_l \underline{-} b_u, a_u \overline{-} b_l]$

乗算： $[a_l,\ a_u] \times [b_l,\ b_u] :=$
$[\min\{a_l \underline{\times} b_l, a_u \underline{\times} b_l, a_l \underline{\times} b_u,\ a_u \underline{\times} b_u\},$
$\max\{a_l \overline{\times} b_l, a_u \overline{\times} b_l, a_l \overline{\times} b_u, a_u \overline{\times} b_u\}]$

除算： $[a_l,\ a_u]/[b_l,\ b_u] :=$
$[\min\{a_l \underline{/} b_l, a_u \underline{/} b_l, a_l \underline{/} b_u,\ a_u \underline{/} b_u\},$
$\max\{a_l \overline{/} b_l, a_u \overline{/} b_l, a_l \overline{/} b_u, a_u \overline{/} b_u\}]$, ただし $0 \notin [b_l,\ b_u]$

ここで，演算子 $\circ$ に対し $\overline{\circ}$ は上向き丸めで演算，$\underline{\circ}$ は下向き丸めでを意味する．

実際に機械区間演算をプログラムするには，上端と下端を持つ構造体 interval などを定義し，さらに定義した構造体 interval に対する機械区間四則演算を演算子多重定義等を用いて実装する．kv Library では上記の区間演算をさらに改良した実装になっており，C++のテンプレートを用いて $\mathrm{interval} < \mathrm{double} >, \mathrm{interval} < \mathrm{dd} >, \mathrm{interval} < \mathrm{mpfr} < N >>$ などの double 型に限らず高精度な計算にも対応できるようになっている．

2.2 機械区間演算の限界と回避策

機械区間演算を利用すれば丸め誤差を把握することが可能である．しかし，区間演算は演算ごとに過大評価を行っているため多くの区間演算を繰り返すと区間の幅が非常に大きくなってしまうことや，丸めモードの制御が多いため実行時間が長くなるなどの問題点がある．

例えば，$A \in \mathbb{F}^{n \times n}$, $b \in \mathbb{F}^n$ における連立一次方程式

$$Ax = b$$

の真の解 $x^* \in \mathbb{R}^n$ を包含する区間ベクトルを求める問題を考える．そのときに，Gauss の消去法に基づくアルゴリズムで生じる計算を全て機械区間演算で実行すれば，得られた結果は真の解 $x^* \in \mathbb{R}^n$ を包含する区間になる．しかし，倍精度浮動小数点数で機械区間演算したとしても区間幅が増大してしまうために考えている問題の次元が n が数十次元程度すら扱えなくなってしまう．例えば次の章で紹介する VCP Library を利用することで，簡単に試すことが出来る:

```
// 例1 VCP Library ( 区間Gauss )
#include<iostream>
#include<omp.h>
#include<kv/interval.hpp>
#include<kv/rdouble.hpp>
#include<vcp/matrix.hpp>
using namespace vcp;
int main(void) {
 int n = 50; //Set dimension
 matrix<kv::interval<double>> A, b, x;
```

*1 現在は C99 の fenv.h にある丸め変更に伴う最適化の抑制を行う FENV_ACCESS が実装されているコンパイラは少ないため，volatile 修飾子にて最適化を抑制する．

```
 A.rand(n); //Make a random matrix
 b.ones(n, 1); //Make vector with 1
 b = A*b; //Matrix-vector product
 x = lss(A, b); //Solve x s.t. Ax = b
 std::cout << x(0) << std::endl;
}
```

このプログラムは $n = 50$ とし，$e = (1, \cdots, 1)^T$ とすると $Ae \in b$ としているため，$Ax = b$ を満たす真の解 x^* は 1 になる．さらに，連立一次方程式の解を求める関数 lss() は LU 分解による前進消去と後退代入により解 x を求めているため，全ての計算に機械区間演算を利用する kv Library の interval 型をテンプレートで渡すと得られた集合 x は真の解 x^* を包含する結果になるはずである．この結果を表示してみると，もちろん乱数の値によっても変化するが著者が試した限り，例えば $[-224705, 224707]$ のような閉区間が得られた．このように 1 を包含する閉区間でも $[-224705, 224707]$ のように区間幅が大きい区間では意味がなくなってしまう．また $n = 100$ とすると，区間幅の増大が影響し 0 除算が発生し，計算できなくなってしまう．

偏微分方程式などの応用問題の近似解を求める際には次元は数千から数十万以上にまでに登る可能性がある．そのため，$n = 100$ 程度で利用できなくなってしまうと応用には適さない．そこで，良く知られている回避策として，連立一次方程式 $Ax = b$ の近似解 $\hat{x}$ を求めてから，真の解 x^* との差を計算するアルゴリズムに切り替える．例えば，近似解 $\hat{x}$ と真の解 x^* の差を計算する定理として山本の定理 [5] が良く知られている．C を行列 (or ベクトル) とし，$|C|$ を行列 (or ベクトル)C の成分毎の絶対値を表す記号とする．また，$\|C\|$ として行列 (or ベクトル)C の最大値ノルムとする．

定理 1 (山本の定理 [5]) 行列 $A \in \mathbb{R}^{n\times n}$, ベクトル $b \in \mathbb{R}^n$, 行列 $R \in \mathbb{R}^{n\times n}$ をそれぞれ与えられているとする．$I \in \mathbb{R}^{n\times n}$ を単位行列とし $e = (1, 1, \cdots, 1)^T \in \mathbb{R}^n$ は全成分が 1 のベクトルとする．もし

$$\|I - RA\| < 1$$

を満たすならば，A は逆行列を持ち，$Ax = b$ を満たす真の解 x^* と与えられた近似解 $\hat{x} \in \mathbb{R}^n$ に対し

$$|x^* - \hat{x}| \leq |R(b - A\hat{x})| + \frac{\|R(b - A\hat{x})\|}{\|I - RA\|}|I - RA|e$$

が成り立つ．

実際には定理 1 に現れる R は行列 A の近似逆行列とし，近似解 $\hat{x}$ と近似逆行列 R は既存のライブラリ等を用いて求めて計算すれば良い．例えば次の章で紹介する VCP Library を利用することで，簡単に試すことが出来る:

```
// 例2 VCP Library(BLAS, LAPACK, 精度保証)
#include<iostream>
#include<omp.h>
#include<kv/interval.hpp>
#include<kv/rdouble.hpp>
#include<vcp/pidblas.hpp>
#include<vcp/matrix.hpp>
using namespace vcp;
int main(void) {
 std::cout.precision(17);
 int n = 10000; // dimension
 matrix< kv::interval< double >,
         pidblas > A, b, x;
 A.rand(n);
 b.ones(n, 1);
 b = A*b;
 x = lss(A, b);
 std::cout << x(0) << std::endl;
}
```

このプログラムは $n = 10000$ とし，先ほどと同様に真の解 x^* が 1 になるように設定している．そのとき，真の解 x^* を包含する閉区間 $[0.9999999759705449, 1.0000000023973821]$ となった．この値は乱数や行列 A の性質により変化するが，直接 Gauss の消去法に機械区間演算を適応した場合では求められなかった 1 万次元の問題でも解くことが可能になっている．

このように精度保証付き数値計算は，「よく利用されるアルゴリズム」+「機械区間演算」のように単純には実装できず，新たに誤差を解析するための定理の実装が必要になる．

3. 定義 1 を満たす数値線形代数ライブラリの構築

本章では定義 1 を満たす数値線形代数ライブラリを Policy-based design に基づいて構築する．図 1 に現在ある良く知られた数値線形代数ライブラリと定義 1 を満たすために必要な型の組み合わせを示す．

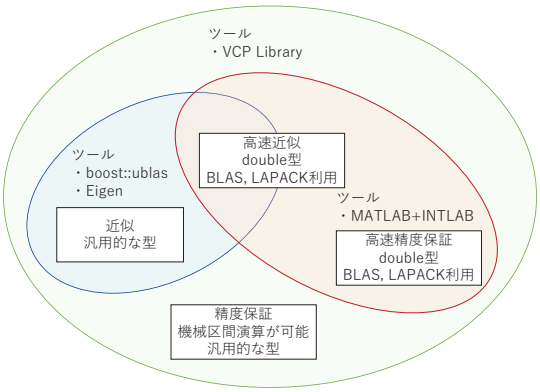

図 1 定義 1 を満たすライブラリに必要な条件とツールの対応

図 1 にあるように今回紹介する VCP Library は全ての型を扱えるように構築している．例えば，以下のようなプログラムで今までの既存のライブラリにはなかった機械区間演算が可能な汎用的な型の精度保証が可能となる．ここでは，機械区間演算が可能な汎用的な型として，kv Library にある疑似 4 倍精度型である dd 型とその機械区間演算型 interval 型を利用している．

```
// 例3 VCP Library(kv::dd, 精度保証)
#include<iostream>
#include<omp.h>
#include<kv/interval.hpp>
#include<kv/dd.hpp>
#include<kv/rdd.hpp>
#include<vcp/imats.hpp>
#include<vcp/matrix.hpp>
using namespace vcp;
 int main(void) {
 std::cout.precision(34);
 int n = 1000; // dimension
 matrix< kv::interval< kv::dd >,
         imats< kv::dd > > A, b, x;
 A.rand(n);
 b.ones(n, 1);
 b = A*b;
 x = lss(A, b);
 std::cout << x(0) << std::endl;
}
```

このプログラムは $n = 1000$ とし，連立一次方程式 $Ax = b$ の真の解 x^* が 1 になるように設定している．そのとき，真の解 x^* を包含する閉区間 [0.99999999999999999999999999020023735, 1.00000000000000000000000000007358396] となった．この結果からもわかるように例 2 の場合に比べ double の限界精度のおよそ 16 桁よりも精度が良く，およそ 25 桁まで 0 が並ぶ結果となった．一方で，BLAS 等の既存の高速なプログラムを使用しないため，非常に計算時間が掛かってしまうことに注意する．

また，例 1 から例 3 までは kv Library の機械区間演算型である interval 型を利用していたが，double 型のみに限定し BLAS と LAPACK を用いた高速な近似解を求めることも可能である:

```
// 例4 VCP Library(double, BLAS, LAPACK)
#include<iostream>
#include<omp.h>
#include<vcp/pdblas.hpp>
#include<vcp/matrix.hpp>
using namespace vcp;
int main(void) {
 std::cout.precision(17);
 int n = 20000; // dimension
 matrix< double, pdblas > A, b, x;
 A.rand(n);
 b.ones(n, 1);
 b = A*b;
 x = lss(A, b);
 std::cout << x(0) << std::endl;
}
```

このプログラムは $n = 20000$ とし，連立一次方程式 $Ax = b$ の真の解 x^* が 1 になるように設定している．このプログラムは精度保証付き数値計算ではないので近似解 1.000000000274329 のように求

まる．また，この値も乱数行列 A が変わる毎回の実行ごとに変化する．

VCP Library の例 1 から例 4 を通して見てみるとわかるように，変数の型名 matrix(Host クラス) に入れるテンプレート引数 (Policy クラス) とそれに対応する必要なヘッダーファイルの include が大きく違っている点で，それ以外には本質的な変更はない．そのため，ユーザーは精度保証の有無や精度に関わらずユーザー自身が実装したい重要なアルゴリズムのプログラムに専念し，必要に応じて変数の型を変更すればアルゴリズムが変更できる．この点が Policy-based design を用いた VCP Library の matrix クラスの特徴である．連立一次方程式の解を求める関数以外にも実対称行列に対する一般化固有値問題の固有値を求める関数もあるので詳細は VCP Library の web ページ http://verified.computation.jp/を参照されたい．

3.1 matrix クラスの設計

matrix クラス (Host クラス) は 2 つのテンプレート引数_T と_P を必要とし，_T には行列の要素の型，_P には行列演算に関するアルゴリズムを定義したクラス (Policy クラス) を与える:

```
vcp::matrix< _T, _P >
```

また，matrix クラスは Policy クラス_P を継承している:

```
template<typename _T, class _P=mats<_T>>
class matrix : protected _P {
 ...
};
```

matrix クラス内では，以下のような 2 点のみ役割を担っている:

- Policy クラス_P を継承することで「ある規則に従って定義されているメソッド変数とメソッド関数」のみを呼び出し，ユーザーが利用しやすい仕様に変形する
- matrix クラス内でのみ move コンストラクタ (C++11) を考え，余分なコピーによるメモリーの圧迫を低減し，高速化をはかる．特に，アルゴリズムの作成に専念すべき Policy クラス_P では move コンストラクタを考えなくても良いようにする．

そのため，Policy クラス_P を「ある決められた規則に基づくメソッド変数とメソッド関数」を持つように作成し，ユーザーは「Policy の型名」と「matrix 型 (Host クラス) の使い方」さえ理解すれば，自由に「型」,「アルゴリズム」,「精度保証の有無」などが選択可能になる (図 2)．

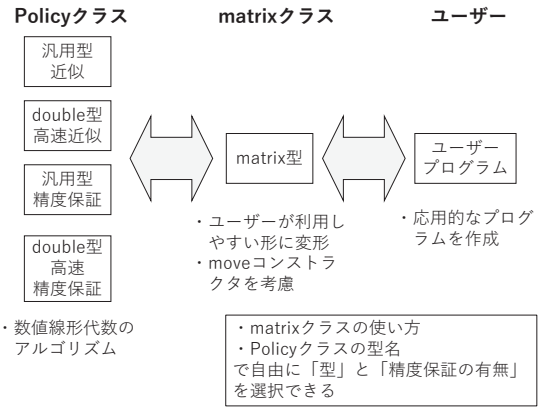

図 2　VCP Library の matrix クラスの考え方

3.2 VCP Library の Policy クラス

現在，VCP ライブラリで提供している Policy クラスは以下の 4 つである:

- mats<_T >
  - _T 型の近似汎用型 Policy
- pdblas
  - double 型の高速近似 Policy
  - BLAS，LAPACK が必要
- imats<_T (, _P) >
  - _T 型の精度保証汎用型 Policy
  - kv ライブラリの interval 型が必要
  - _P は精度保証付き数値計算で必要となる近似解を計算する行列用 Policy．記述しない場合は_P = mats< _T >となる．
- pidblas
  - double 型の高速精度保証 Policy
  - LAPACK と丸め変更可な BLAS が必要
  - kv ライブラリの interval 型が必要

また，図 3 に示すように各 Policy の継承の関係は mats が根幹にある．そのため，新しい機能を追加したければ mats に追加し，matrix クラスから呼び出せるようにすれば，すべての Policy で利用可能になる．

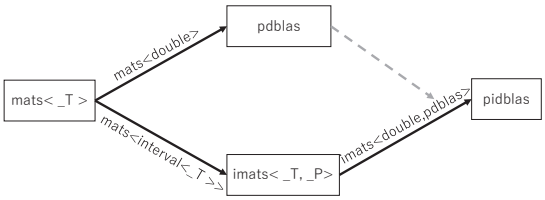

図 3　Policy 間の継承の関係

また，必要に応じて mats を継承した Policy 内で変更したいアルゴリズムに関するメソッド関数を書き換えれば，別のアルゴリズムを持つ Policy を作成できる．そのため，Policy クラスの管理や拡張，一部のみアルゴリズムを変更するなどが容易な設計にしてある．

4.　数値線形代数の新しい開発モデル

第 3 章では VCP Library の Policy-based design を用いた数値線形代数ライブラリの設計思想を紹介した．特に，

- 数値線形代数のアルゴリズムを記述する Policy クラス
- 関数の呼び出し方法などの使い勝手とメモリー削減 (move) を記述した matrix クラス (Host クラス)

とわけることで，管理や拡張，一部のみアルゴリズムを変更するなどが容易な設計にしてある．

数値線形代数のアルゴリズムの記述場所を Policy クラスとしたライブラリは，アルゴリズムを開発する研究者と数値線形代数を応用する研究者の双方にメリットがある．ここでは，Policy-based design を用いた数値線形代数ライブラリの特徴を生かした開発モデルを提案する．まず，図 4 に従来の数値線形代数の開発モデルを示す．現在，数値線形代数のアルゴリズムは無数にある．例えば，LU 分解をみても Crout 法や Doolittle 法，さらにブロック化の手法など多岐にわたる．これらのアルゴリズムの選択は現状，言語やライブラリの開発者が行っている．そのため，ユーザーは使いたいアルゴリズムを選択するには言語やライブラリを横断するか，あるいはユーザー自身でアルゴリズムを実装しなければならない．また，アルゴリズム開発者にとっても，有名な言語やライブラリに実装されない限り，開発者自身がライブラリを作成するか，あるいは論文のみの公開に留まってしまう．

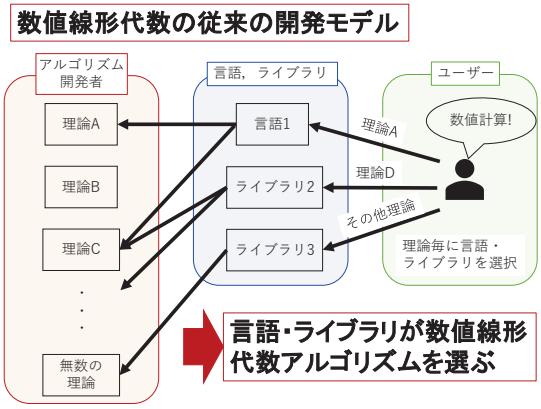

図 4　数値線形代数の従来の開発モデル

それに対し，数値線形代数のアルゴリズムの記述場所を Policy クラスとしたライブラリであれば，図 5 に示すような開発モデルが構築できる．数値線形代数のアルゴリズム開発者は既存の Policy クラスの一部のみ自身のアルゴリズムに差し替えて，開発者自身の Policy クラスを提供でき，自身のアルゴリズムを応用してくれるユーザーまですぐに届けることが可能になる．また，ユーザーは Host クラスに渡す Policy クラスを変更のみで，提供されている最新のアルゴリズムを使用可能になる．そのため，ユーザ-は作成したい計算に適切な数値線形代数アルゴリズムを最後に決定することが可能である．

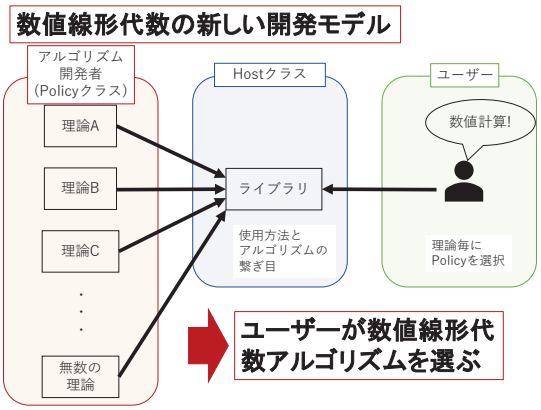

図 5　数値線形代数の新しい開発モデル

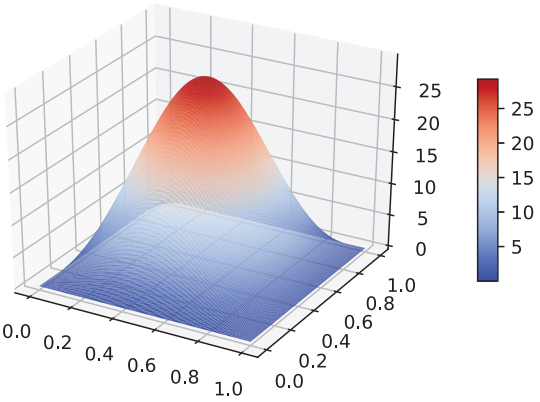

図 6　Emden 方程式の近似解 ($N = 50$)

このように Policy-based design を用いた数値線形代数ライブラリでは，数値線形代数のアルゴリズム開発者・ユーザーともにメリットがある開発モデルを構築することが可能になる．

5. 応用例

VCP Library の応用例として実際に楕円型偏微分方程式の解の計算機援用存在証明を行ってみる．領域 $\Omega = (0,1)^2$ とし，定常 Emden 方程式の Dirichlet 境界値問題

$$\begin{cases} -\Delta u = u^2 & \text{in } \Omega, \\ u = 0 & \text{on } \partial\Omega, \end{cases} \tag{1}$$

の非自明解の存在を計算機を用いて証明する．

有限次元の基底 ϕ_i 張られた空間 V_N とし，計算機で求める近似解 $\hat{u} \in V_N$ は

$$\hat{u}(x,y) = \sum_{i=1,j=1}^{N} u_{i,j}\phi_i(x)\phi_j(y)$$

とする．但し，Legendre 多項式

$$P_i = \frac{(-1)^i}{i!}\left(\frac{d}{dx}\right)^i x^i(1-x)^i,\ i = 0,1,\cdots$$

を用いた基底

$$\phi_i(x) = \frac{1}{i(i+1)}x(1-x)\frac{dP_i}{dx}(x),\ i = 1,2,\cdots$$

を利用する．$N = 50$ のときの定常 Emden 方程式 (1) の近似解を図 6 に示す．

$L^2(\Omega)$ を Lebesgue 2 乗可積分な関数全体の集合とし，$H^1(\Omega)$ を 1 階の L^2-Sobolev 空間とする．また，$H_0^1(\Omega) := \{u \in H_0^1(\Omega) \ :\ u = 0 \text{ on } \partial\Omega\}$ とし，$H_0^1(\Omega)$ の共役空間を $H^{-1}(\Omega)$ とする．そのときに，$H_0^1(\Omega)$ 内に定常 Emden 方程式 (1) の解が存在することを Newton-Kantorovich の定理を用いて証明する:

定理 2 (Newton-Kantorovich の定理の系) 非線形作用素 $F : H_0^1(\Omega) \to H^{-1}(\Omega)$ とし，方程式 $F(u) = 0$ の近似解を $\hat{u} \in H_0^1(\Omega)$ とする．非線形作用素 F の $\hat{u}$ における Fréchet 微分 $F'[\hat{u}] : H_0^1(\Omega) \to H^{-1}(\Omega)$ は正則であり，不等式

$$\|F'[\hat{u}]^{-1}\|_{\mathcal{L}(H^{-1},H_0^1)} \le K$$

を満たす正の定数 K が存在すると仮定する．また，正の定数 δ を残差ノルムの上界

$$\|F(\hat{u})\|_{H^{-1}} \le \delta$$

を満たすとする．$\bar{B}$ を $\bar{B}(\hat{u}, 2K\delta) := \{u \in H_0^1(\Omega) \ :\ \|u - \hat{u}\|_{H_0^1} \le 2K\delta\}$ を満たす閉球とする．D を $D \supset \bar{B}$ を満たす開球とする．正の定数 G を

$$\|F'[w_1] - F'[w_2]\|_{\mathcal{L}(H_0^1,L^2)} \le G\|w_1 - w_2\|_{H_0^1},\quad \forall w_1, w_2 \in D$$

を満たすとする．もし，$K^2\delta G \le 1/2$ を満たすならば，方程式 $F(u) = 0$ を満たす解 $u^* \in H_0^1(\Omega)$ は存在し，

$$\|u^* - \hat{u}\|_{H_0^1} \leq \frac{1 - \sqrt{1 - 2K^2\delta G}}{KG} \qquad (2)$$

を満たす．その上で，解 u^* は D 内に一意である．

定理 2 を利用するには，定数 K, δ, G を精度保証付き数値計算を利用して求めて，さらに $K^2\delta G$ が 1/2 以下であれば解の存在が証明できる．さらに，近似解 $\hat{u}$ と真の解 u^* の H_0^1 ノルムの意味での差も不等式 (2) で計算可能である．詳しい定数の求め方は，参考文献 [6], [7], [8], [9], [10] を参照されたい．

近似解 $\hat{u}$ は kv Library の疑似 4 倍精度型である dd 型を挿入した Policy クラス mats <kv::dd >の連立一次方程式の近似用のソルバーで求めている．さらに定数 K は kv Library の interval <double > 型を挿入し，さらに BLAS と Lapack で高速した精度保証付き数値計算用の Policy クラス pidblas の固有値問題のソルバーを利用している．

計算機環境は CPU: Intel Xeon Phi Processor 7290(1.5 GHz, 72Core, 288 Thread)，Memory: 384GByte (DDR4)，OS: CentOS 7.2.1，Compiler: g++ 4.8.5，kv Library 0.4.43[3], Intel MKL version 2017 を利用した．

$N = 20, 40, 60$ の場合における定数 K, δ, G の値を表 1 に，Newton-Kantorovich の定理の十分条件 $K^2\delta F \leq 1/2$ と $H_0^1(\Omega)$ ノルムの意味での誤差 $\|u^* - \hat{u}\|_{H_0^1}$ を表 2 に示す．定数 K と G は基底の項数 N によって大きく変化していない．一方で，残差にあたる定数 δ は基底の項数が増えるごとに小さくなっていることがわかる．また，すべての場合において Newton-Kantorovich の定理の十分条件 $K^2\delta F \leq 1/2$ を満たしているため，近似解の周辺に真の解が存在することが証明できた．さらに，$H_0^1(\Omega)$ ノルムの意味での誤差 $\|u^* - \hat{u}\|_{H_0^1}$ も基底の項数が増えるごとに小さくなっていることが見て取れる．

表 1 定数 K, δ, G の結果

N	K	δ	G
20	2.063	4.24×10^{-5}	4.39×10^{-2}
40	2.063	4.34×10^{-10}	4.39×10^{-2}
60	2.063	8.26×10^{-12}	4.39×10^{-2}

表 2 解の計算機援用証明法の結果

N	$K^2\delta G$	$\|u^* - \hat{u}\|_{H_0^1}$
20	7.93×10^{-6}	9.18×10^{-5}
40	8.09×10^{-11}	9.37×10^{-10}
60	1.55×10^{-12}	1.79×10^{-11}

表 3 に実行時の計算時間を示す．計算時間は増やすほどの項数を増加していることが見て取れる．特に最も時間が掛かる計算は，近似解の最大値と最小値を求める計算であった．この計算は，真に最大値や最小値を包含する区間を作成するために，行列計算は利用できず，スカラー計算にのみにとどまり，また並列度も非常に高いため改善する余地は今のところない．そのため，数値線形代数を利用する近似解や定数 K の求める計算時間は主要時間になっていないことがわかる．

表 3 計算時間 [msec]

N	近似解	最大・最小値	$K(N = 40)$	δ
20	9068	79301	16530	2388
40	58529	544292	20321	16821
60	413523	1686771	29503	63934

6. まとめと今後の課題

本論文では，偏微分方程式の解の計算機援用証明法のための数値線形代数ライブラリを Policy-based design に乗っ取り作成した．また，Policy-based design の特徴を生かした数値線形代数の新しい開発モデルを提案した．さらに，応用として作成したライブラリを用いて実際に Emden 方程式の定常解に対する計算機援用存在証明法を行い，真の解の存在性を証明した．

現状 Policy である mats< _T >が「行列の操作」,「行列同士の演算」,「スカラーと行列の演算」,「連立一次方程式」,「固有値問題」,「一般化固有値問題」のすべてが含まれているため非常に大きなプログラムとなっている．そこで今後の課題としては，mats< _T >も管理しやすいように Policy を導入することも可能である．また，精度保証付き数値計算のテクニックの一つである一部の計算のみ高精度計算を利用する Policy は現状ないため，

今後より複雑な偏微分方程式の解の存在証明を行う際には作成する必要がある.

謝辞 本研究は JSPS 科研費，課題番号 16K17651 の支援を受けたものである．また本研究は，文部科学省ポスト「京」萌芽的課題「極限の探究に資する精度保証付き数値計算学の展開と超高性能計算環境の創成」の一環として実施したものである．また本研究は，JST，CREST「モデリングのための精度保証付き数値計算論の展開」の支援を受けたものである．kv Library の開発者である早稲田大学 柏木雅英先生には VCP Library について多くの助言を頂き心から感謝の気持ちと御礼を申し上げたく、謝辞にかえさせて頂きます.

参考文献

[1] IEEE Standard for Floating-Point Arithmetic, Std 754-2008, (2008).
[2] S.M. Rump, INTLAB - INTerval LABoratory, In Tibor Csendes, editor, Developments in Reliable Computing, pp.77104. Kluwer Academic Publishers, Dordrecht, (1999).
[3] M. Kashiwagi: kv - C++ による精度保証付き数値計算ライブラリ, http://verifiedby.me/kv/
[4] 大石進一: 精度保証付き数値計算, コロナ社 (2000).
[5] T. Yamamoto: Error Bounds for Approximate Solutions of Systems of Equations, Japan J. Appl. Math., 1, 1, pp.157–171 (1984).
[6] M.T. Nakao: A numerical approach to the proof of existence of solutions for elliptic problems, Japan J. Indust. Appl. Math. 5, pp.313–332 (1988).
[7] 中尾充宏, 渡部善隆: 実例で学ぶ精度保証付き数値計算:理論と実装, サイエンス社 (2011).
[8] M. Plum: Computer-assisted existence proofs for two-point boundary value problems, Computing. 46, pp.19–34 (1991).
[9] M. Plum: Computer-assisted proofs for semilinear elliptic boundary value problems, Japan J. Indust. Appl. Math. 26, 2-3, pp.419-442 (2009).
[10] K. Tanaka, A. Takayasu, X. Liu, and S. Oishi: Verified norm estimation for the inverse of linear elliptic operators using eigenvalue evaluation, Japan J. Indust. Appl. Math, 31, 3, pp.665–679 (2014).