C++で学ぶ
精度保証付き数値計算法の初歩

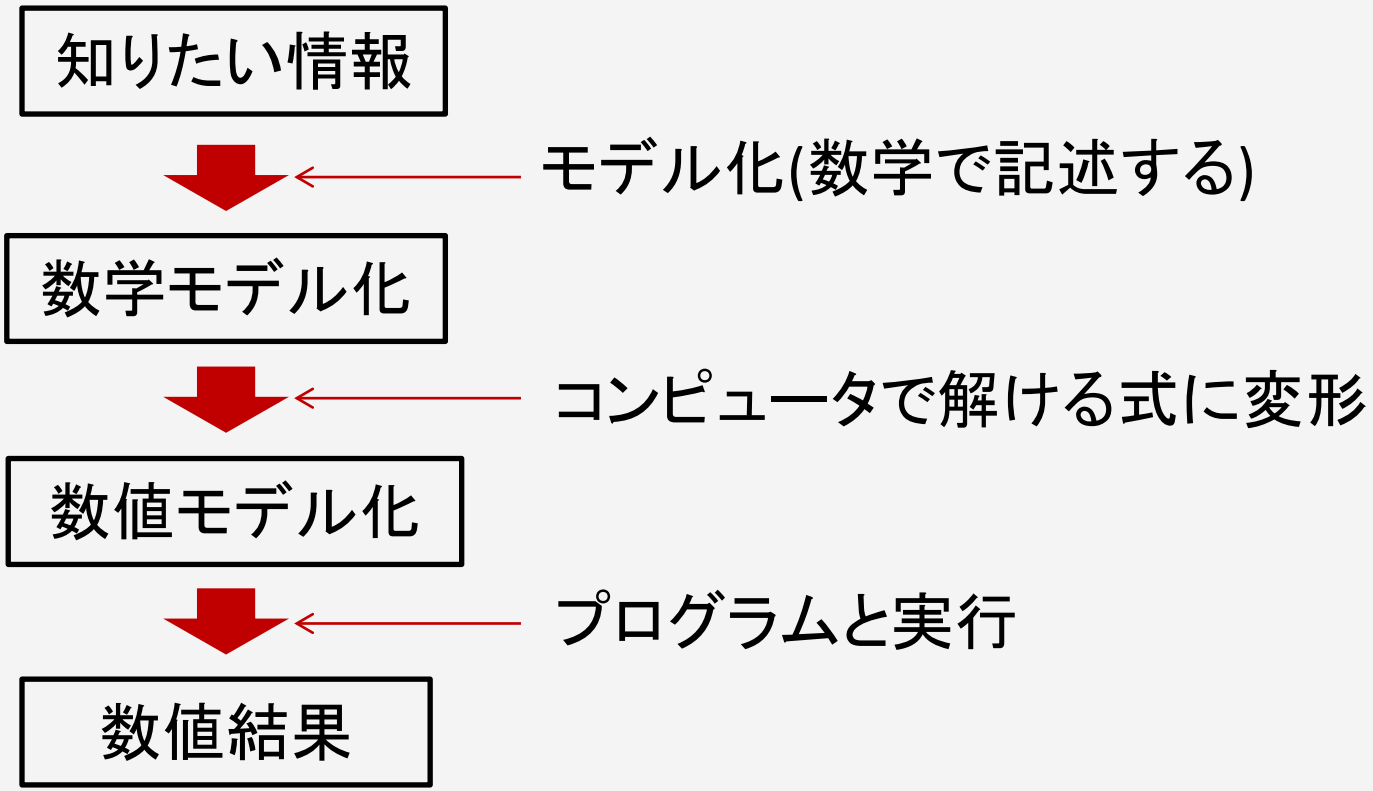
東洋大学 情報連携学部　関根 晃太

数値計算とは?

定義:
コンピュータを用いて，手で解くことが
困難な数学の問題を近似的に解く方法

数値計算とは?

数値計算：
コンピュータを用いて，手で解くことが
困難な数学の問題を近似的に解く方法

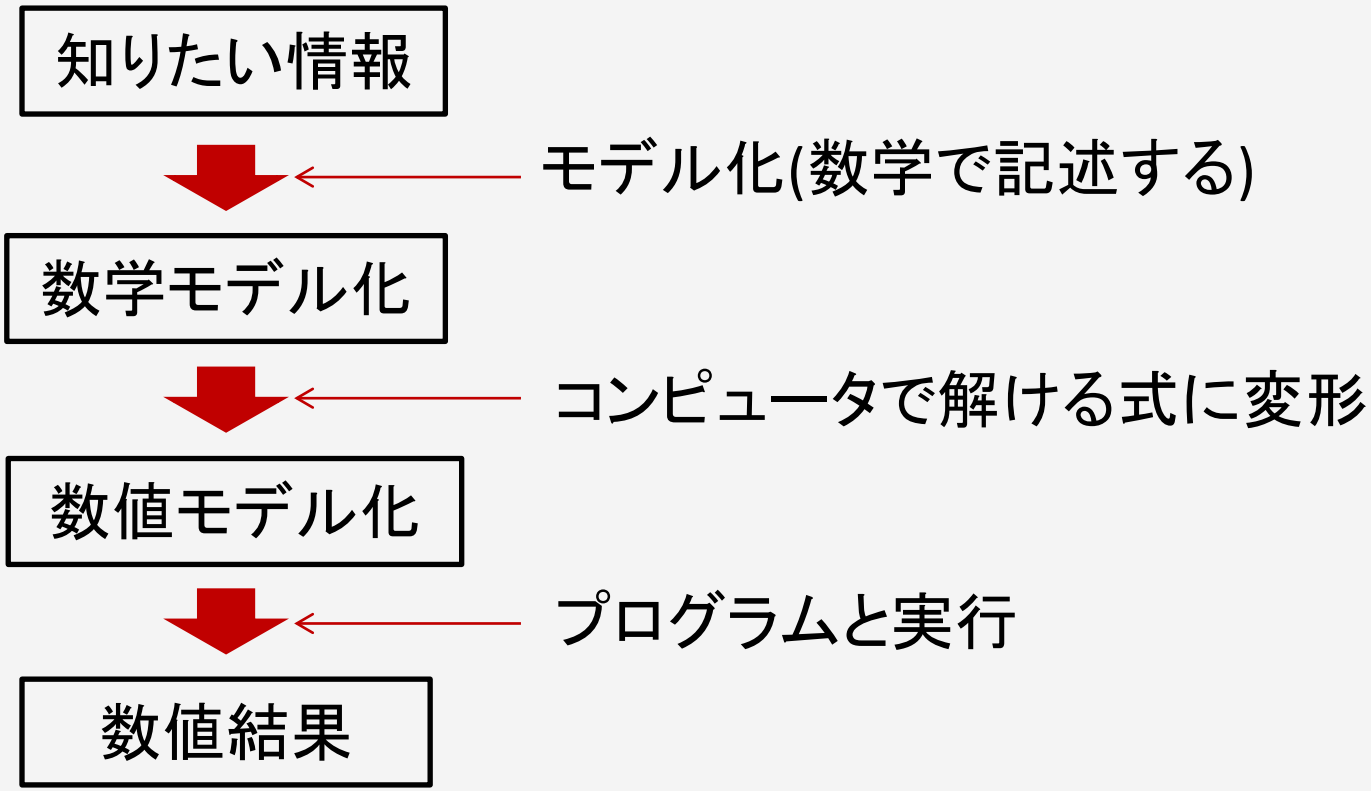


精度保証付き数値計算とは?

あなたのコンピュータは正しい結果を
返していますか?

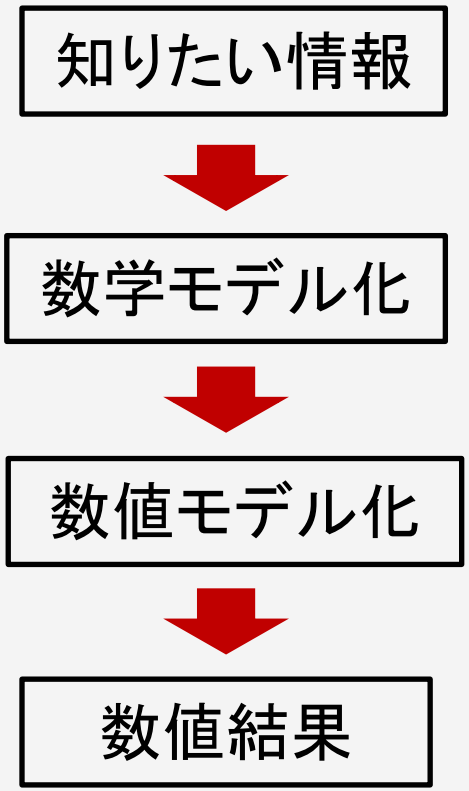
精度保証付き数値計算とは?

精度保証付き数値計算：
与えられた問題(数学モデル)の解の存在範囲もしくは
一意存在の範囲を丸め誤差の厳密評価を含めて特定する算法



精度保証付き数値計算とは?

精度保証付き数値計算：
与えられた問題(数学モデル)の解の存在範囲もしくは
一意存在の範囲を丸め誤差の厳密評価を含めて特定する算法



近似的に計算

誤差が入る!!

精度保証付き数値計算とは?

- 誤差について
  プログラムのどこで誤差が発生するか?

- 区間演算について
  四則演算の誤差はどのように把握するか?

- ライブラリについて
  もっと簡単に使えないか?

精度保証付き数値計算とは?

- 誤差について
  プログラムのどこで誤差が発生するか?

- 区間演算について
  四則演算の誤差はどのように把握するか?

- ライブラリについて
  もっと簡単に使えないか?

プログラムを少し書きます!

どこに誤差が発生するかわかりますか??

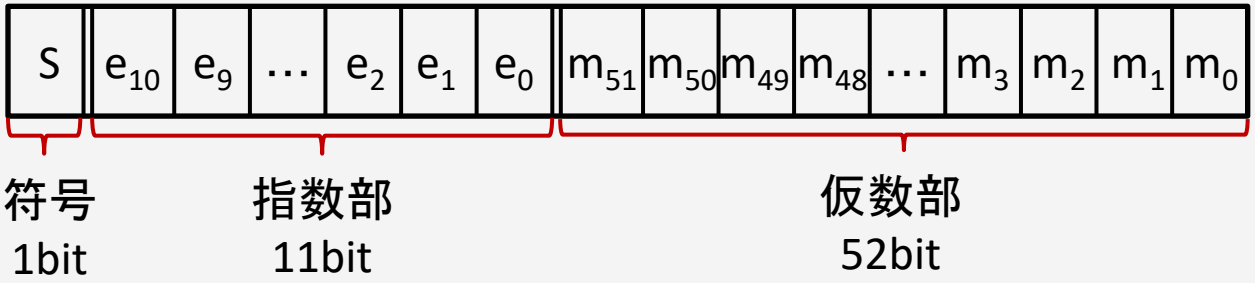
```
#include <iostream>

int main(void){

	double a = 0.1;
	double b = 0.01;
	double c = a + b;
	std::cout << c << std::endl;

	return 0;
}
```

精度保証付き数値計算とは?

コンピュータで扱える数値の一つ:
IEEE 754 Standardで定められている浮動小数点数

倍精度浮動小数点数の場合…
S, e_i, m_jを全て2進数

決まった数しか表現できない…



値: $(-1)^S \times (1+m) \times 2^{e-1023}$

整数:$e=e_0 \times 2^0 + \cdots + e_{10} \times 2^{10}$
小数:$m=m_0 \times 2^{0-52} + \cdots + m_{52} \times 2^{52-52}$

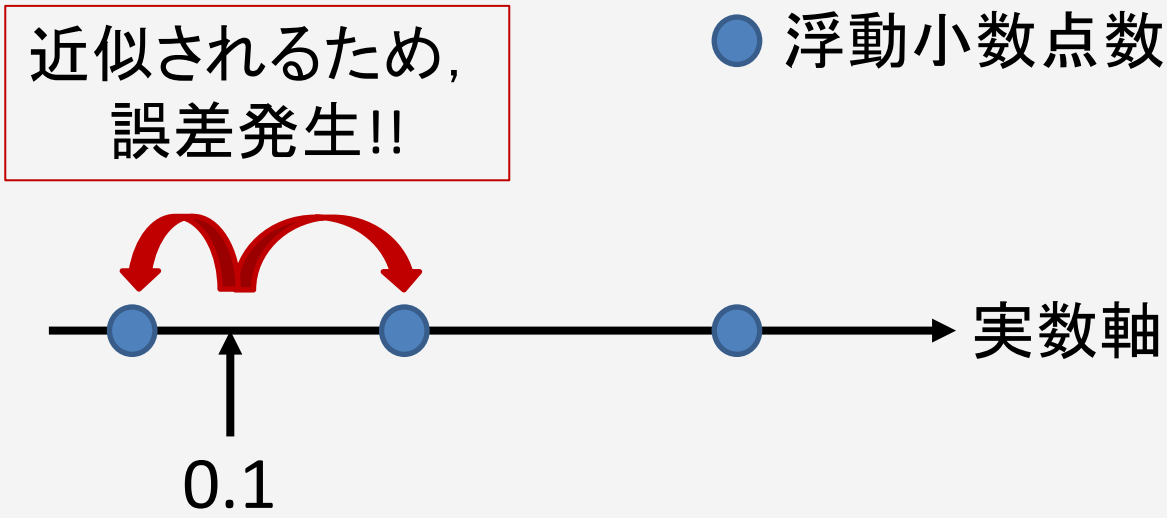
精度保証付き数値計算とは?

Q. 10進数の0.1と0.01は
倍精度浮動小数点数で表せますか?

精度保証付き数値計算とは?

Q. 10進数の0.1と0.01は
倍精度浮動小数点数で表せますか?

A. いいえ…



特に, 入力による誤差は非常に把握しにくい!!
(上に丸められたか, 下に丸められたか判断できない…)

プログラムを少し書きます!

どこに誤差が発生するかわかりますか??

```
#include <iostream>

int main(void){

    double a = 0.1;
    double b = 0.01;
    double c = a + b;
    std::cout << c << std::endl;

    return 0;
}
```

誤差が発生するので
そのままではダメ…

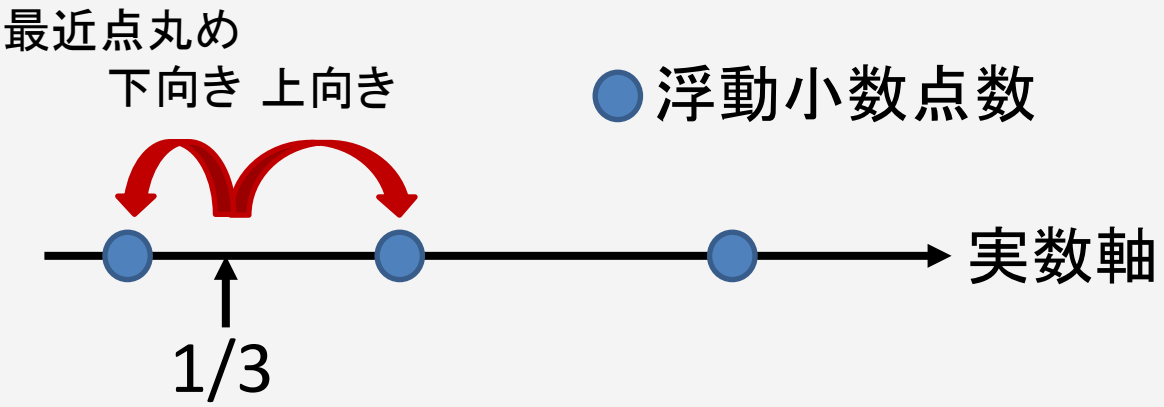
精度保証付き数値計算とは?

Q. 10進数の1と3は倍精度浮動小数点数で表せます.
その上で, 1/3の結果は浮動小数点数で表せますか?

精度保証付き数値計算とは?

Q. 10進数の1と3は倍精度浮動小数点数で表せます.
その上で, 1/3の結果は浮動小数点数で表せますか?

A. いいえ…
但し, IEEE754 Standardで定められている
丸めモードの指定をすることでどこの浮動小数点数に
いったか把握可能!!



プログラムを少し書きます!

どこに誤差が発生するかわかりますか??

```
#include <iostream>

int main(void){

    double a = 0.1;
    double b = 0.01;
    double c = a + b;
    std::cout << c << std::endl;

    return 0;
}
```

演算でも誤差が発生するのでそのままではダメ…

プログラムを少し書きます!

どこに誤差が発生するかわかりますか??

```
#include <iostream>

int main(void){

    double a = 0.1;
    double b = 0.01;
    double c = a + b;
    std::cout << c << std::endl;

    return 0;
}
```

出力は??

プログラムを少し書きます!

どこに誤差が発生するかわかりますか??

```
#include <iostream>

int main(void){

    double a = 0.1;
    double b = 0.01;
    double c = a + b;
    std::cout << c << std::endl;

    return 0;
}
```

出力は??
そもそも決められておらず,
何しているかわからない…
(浮動小数点演算が使われていたら, もちろんNG!)

プログラムを少し書きます!

どこに誤差が発生するかわかりますか??

```
#include <iostream>

int main(void){

    double a = 0.1;
    double b = 0.01;
    double c = a + b;
    std::cout << c << std::endl;

    return 0;
}
```

出力は??
そもそも決められておらず,
何しているかわからない…
(浮動小数点演算が使われていたら, もちろんNG!)

精度保証付き数値計算とは?

- 誤差について
  プログラムのどこで誤差が発生するか?

- 区間演算について
  四則演算の誤差はどのように把握するか?

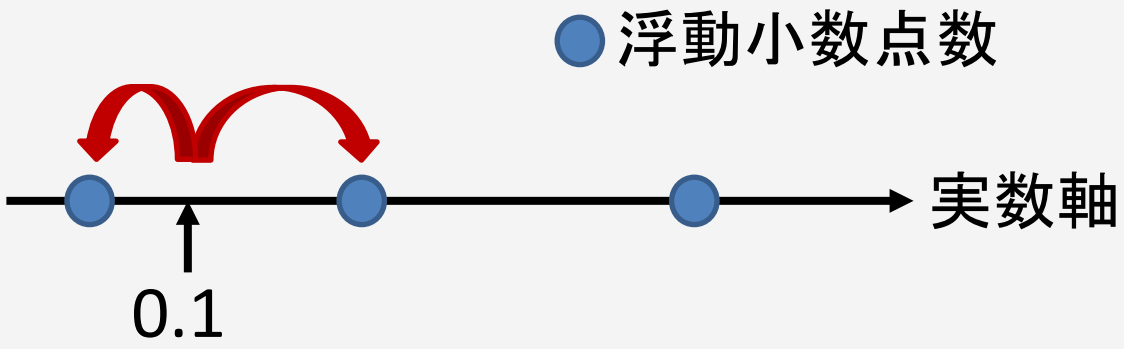
- ライブラリについて
  もっと簡単に使えないか?

精度保証付き数値計算とは?

精度保証付き数値計算では...

$$0.1 \in [0.10000, 0.10001]$$

のように区間で表現し，常に真の値を含むようにする!!



精度保証付き数値計算とは?

もしかすると,
精度保証付き数値計算を聞いたことがある人は...

丸め変更で簡単に何とかなるんじゃないの!?

って思うかもしれません

精度保証付き数値計算とは?

Q. 大抵のCPUがIEEE754 Standardに準拠しています.
では，丸めモードが標準で変更可能な
プログラミング言語は??

精度保証付き数値計算とは?

Q. 大抵のCPUがIEEE754 Standardに準拠しています.
では，丸めモードが標準で変更可能な
プログラミング言語は??

A. C99(C/C++)
Fortran 2003
MATLAB
Octave

・Pythonなどは標準では備えていない…
・C#は丸めモードの変更を非推奨

精度保証付き数値計算とは?

Q. 大抵のCPUがIEEE754 Standardに準拠しています.
では，丸めモードが標準で変更可能な
プログラミング言語は??

A. C99(C/C++)
Fortran 2003
MATLAB
Octave

C99を使った丸めモードの変更

```
#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 );
}
```

精度保証付き数値計算とは?

```
#include <iostream>
#include <fenv.h>

int main(void) {
    // 下向き丸めに変更
    fesetround(FE_DOWNWARD );
    double a_down = 0.1;

    // 上向き丸めに変更
    fesetround(FE_UPWARD );
    double a_up = 0.1;
}
```

精度保証付き数値計算とは?

```
#include <iostream>
#include <fenv.h>

int main(void) {
    // 下向き丸めに変更
    fesetround(FE_DOWNWARD );
    double a_down = 0.1;

    // 上向き丸めに変更
    fesetround(FE_UPWARD );
    double a_up = 0.1;
}
```

丸め変更は
四則演算(+, -, *, /)と
平方根(sqrt)のみ対応!

精度保証付き数値計算とは?

Q. なぜ下記のプログラムはダメなのでしょうか?

C99を使った1/3の包含

```
#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;
}
```

精度保証付き数値計算とは?

Q. なぜ下記のプログラムはダメなのでしょうか?

C99を使った1/3の包含

```
#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;
}
```

最適化後の例

```
#include <fenv.h>

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

コンパイラのオプションでコードが最適化されます…
(「-O0」で実行すると遅い…)

精度保証付き数値計算とは?

解決策(?)

C99にはFENV_ACCESSという丸めモードの変更に
伴う最適化の抑制を行うコマンドが制定されている!!

精度保証付き数値計算とは?

解決策(?)

C99にはFENV_ACCESSという丸めモードの変更に
伴う最適化の抑制を行うコマンドが制定されている!!

が，しかし，実装されているコンパイラは少ない…
(例えばgcc 9.1にはない…)

精度保証付き数値計算とは?

解決策(?)

C99にはFENV_ACCESSという丸めモードの変更に
伴う最適化の抑制を行うコマンドが制定されている!!

が，しかし，実装されているコンパイラは少ない…
(例えばgcc 9.1にはない…)

現状は，volatile修飾子で
部分的に最適化を抑制…

(が，しかし，最近制定されたc++20では，volatileは本来の目的以外で使用することを非推奨
将来はダメにするといわれ，現在はガクブル状態…)

精度保証付き数値計算とは?

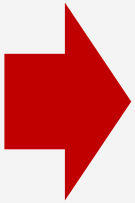
今までのまとめ

- 演算ごとにも誤差が発生する
- 入出力でも誤差が発生する
- 精度保証では真の値を包含した区間で表現
- 最適化によって演算順序が変わると
  精度保証できない

精度保証付き数値計算とは?

今までのまとめ

- 演算ごとにも誤差が発生する
- 入出力でも誤差が発生する
- 精度保証では真の値を包含した区間で表現
- 最適化によって演算順序が変わると
  精度保証できない

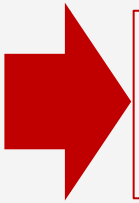


0から入出力, 四則演算をつくり,
そこから自分の解きたい問題まで…
ってやっていたら非常に大変, やだなぁ…

精度保証付き数値計算とは?

今までのまとめ

- 演算ごとにも誤差が発生する
- 入出力でも誤差が発生する
- 精度保証では真の値を包含した区間で表現
- 最適化によって演算順序が変わると
  精度保証できない



自分で区間演算を実装するのはやめて,
既存のライブラリを使いましょう!

2.有名な精度保証付き数値計算ライブラリ

INTLAB:

HP参照

言語:

MATLAB

特徴:

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・高精度な行列，数値線形代数に関わる精度保証 付き数値計算が強力．

S.M. Rump, INTLAB - INTerval LABoratory, In Tibor Csendes, editor, Developments in Reliable Computing, pp.77104. Kluwer Academic Publishers, Dordrecht, 1999.

http://www.ti3.tuhh.de/rump/intlab/

2.有名な精度保証付き数値計算ライブラリ

kv Library:

HP参照

言語:

C++

特徴:

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．

M. Kashiwagi: kv - C++ による精度保証付き数値計算ライブラリ,
http://veriedby.me/kv/

kv ライブラリを使った精度保証プログラム

ここからは実際に,

- インストール手順
- プログラミング

などを実演します!

プログラミングのメモのみ残しておきます.

kv ライブラリを使った精度保証プログラム

```
#include <iostream>

int main(void){

	double a = 0.1;
	double b = 0.01;
	double c = a + b;
	std::cout << c << std::endl;

	return 0;
}
```

```
#include <iostream>
#include <kv/interval.hpp>
#include <kv/rdouble.hpp>

typedef kv::interval< double > idouble;

int main(void){
	idouble a = idouble("0.1");
	idouble b = idouble("0.01");
	idouble c = a + b;
	std::cout << c << std::endl;

	return 0;
}
```

VCPを使った近似行列プログラム(1)

```
#include <iostream>
#include <vcp/matrix.hpp>
#include <vcp/matrix_assist.hpp>

typedef vcp::matrix< double, vcp::mats< double > > vmat;

int main(void){

	int n = 5;
	vmat A, B, C;

	A.zeros(n, n);
	B.rand(n, n);
	C = A + B;
	std::cout << C << std::endl;

	return 0;
}
```

BLASとVCPを使った近似行列プログラム

```
#include <iostream>
#include <vcp/matrix.hpp>
#include <vcp/matrix_assist.hpp>
#include <vcp/pdblas.hpp>

#include <vcp/vcp_timer.hpp>

//typedef vcp::matrix< double, vcp::mats< double > > vmat;
typedef vcp::matrix< double, vcp::pdblas > vmat;

int main(void){

	int n = 5000;
	vmat A, B, C;

	A.rand(n, n);
	B.rand(n, n);

	vcp::time.tic();
	C = A * B;
	vcp::time.toc();

	std::cout << C(0, 0) << std::endl;

	return 0;
}
```

BLASとVCPを使った近似連立一次プログラム

```
#include <iostream>
#include <vcp/matrix.hpp>
#include <vcp/matrix_assist.hpp>
#include <vcp/pdblas.hpp>

#include <vcp/vcp_timer.hpp>

//typedef vcp::matrix< double, vcp::mats< double > > vmat;
typedef vcp::matrix< double, vcp::pdblas > vmat;

int main(void){

	int n = 5000;
	vmat A, x, b;

	A.rand(n, n);
	b.ones(n, 1);
	b = A*b;

	vcp::time.tic();
	x = lss(A, b);
	vcp::time.toc();

	std::cout << x(0) << std::endl;

	return 0;
}
```

mpfr,kv,VCPを使った保証付き連立一次方程式

```
#include <iostream>

#include <kv/interval.hpp>
#include <kv/mpfr.hpp>
#include <kv/rmpfr.hpp>

#include <vcp/matrix.hpp>
#include <vcp/imats.hpp>
#include <vcp/matrix_assist.hpp>

#include <vcp/vcp_timer.hpp>

typedef kv::mpfr< 150 > kvmpfr;
typedef kv::interval< kvmpfr > impfr;
typedef vcp::imats< kvmpfr > policy;
typedef vcp::matrix< impfr, policy > vmat;
```

```
int main(void){

    int n = 200;
    vmat A, x, b;

    A.rand(n, n);
    b.ones(n, 1);
    b = A*b;

    vcp::time.tic();
    x = lss(A, b);
    vcp::time.toc();

    std::cout << x(0) << std::endl;

    return 0;
}
```

BLAS,kv,VCPを使った保証付き連立一次方程式

```
#include <iostream>

#include <kv/interval.hpp>
#include <kv/rdouble.hpp>

#include <vcp/matrix.hpp>
#include <vcp/pidblas.hpp>
#include <vcp/matrix_assist.hpp>

#include <vcp/vcp_timer.hpp>

typedef kv::interval< double > idouble;
typedef vcp::pidblas policy;
typedef vcp::matrix< idouble, policy > vmat;
```

```
int main(void){

    int n = 5000;
    vmat A, x, b;

    A.rand(n, n);
    b.ones(n, 1);
    b = A*b;

    vcp::time.tic();
    x = lss(A, b);
    vcp::time.toc();

    std::cout << x(0) << std::endl;

    return 0;
}
```