C++ による区間解析ライブラリの作成について

戸川哲夫・榊原道夫*

岡山理科大学大学院理学研究科

*岡山理科大学理学部応用数学科

(1995年9月30日　受理)

1. はじめに

数値計算においていくつかの誤差のタイプがある。そのうち一般に計算の結果生じる丸め誤差がある。区間解析[1]は,そのような誤差に対して上限と下限の幅を与える事ができる。またその区間の幅は,できるだけ狭いことが望まれる。区間解析はまた方程式の求解,電子回路の精度保証つきの数値計算の解析など様々な分野に応用可能である。このように,初期データにおける丸め誤差や計算過程における伝播誤差を推定する事ができる。しかしながら,区間演算は一般の実数演算に比べて少し異なる。また従来の言語(例えばFORTRAN)を用いプログラミングすると非常に複雑なものとなり,多くのステップ数を必要とする。そこで,区間演算のライブラリをプログラミング言語C++[2]を用いて作成し,区間解析用のツールを開発した。ライブラリの構成及び使用法について述べるとともに,開発したライブラリを用いた応用例を紹介する。

2. 区間演算の定義

実数 x, y に対して, $x \in X = [\underline{X}, \bar{X}]$, $y \in Y = [\underline{Y}, \bar{Y}]$ を考える。ここで, $\underline{X}$, $\bar{X}$ はそれぞれ区間 X の下限と上限を表す。$\underline{Y}$, $\bar{Y}$ についても同様である。区間ベクトルを $X = (X_1, X_2, \cdots, X_n)$ と表す。ここで, X_1, X_2, $\cdots$, X_n はそれぞれベクトルの要素である。区間マトリックスを

$$A = \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nn} \end{pmatrix}$$

と表す。ここで, A_{ij} $(i, j = 1, 2, \cdots, n)$ はマトリックスの要素である。

2.1 四則演算

四則演算の定義を表1に示す。

表1 四則演算

加算	$X+Y=[\underline{X}+\underline{Y}, \bar{X}+\bar{Y}]$
減算	$X-Y=[\underline{X}-\bar{Y}, \bar{X}-\underline{Y}]$
乗算	$X\cdot Y=[\min\{\underline{X}\cdot\underline{Y}, \bar{X}\cdot\bar{Y}, \underline{X}\cdot\bar{Y}, \bar{X}\cdot\underline{Y}\}, \max\{\underline{X}\cdot\underline{Y}, \bar{X}\cdot\bar{Y}, \underline{X}\cdot\bar{Y}, \bar{X}\cdot\underline{Y}\}]$
除算	$X/Y=[\min\{\underline{X}/\underline{Y}, \bar{X}/\bar{Y}, \underline{X}/\bar{Y}, \bar{X}/\underline{Y}\}, \max\{\underline{X}/\underline{Y}, \bar{X}/\bar{Y}, \underline{X}/\bar{Y}, \bar{X}/\underline{Y}\}]$ ただし $0\notin Y$

2.2 集合演算

X, Y に対して，もし $\underline{X}>\bar{Y}$ または $\underline{Y}>\bar{X}$ ならば

$$X\cap Y=\phi$$

である。集合演算に対する論理積，論理和を表2に示す。

表2 集合演算に対する論理積，論理和

論理積	もし，$X\cap Y\neq\phi$ ならば
	$X\cap Y=[\max(\underline{X}, \underline{Y}), \min(\bar{X}, \bar{Y})]$
	もし区間ベクトル X, Y の対応するすべての要素が空ならば
	$X\cap Y=(X_1\cap Y_1, X_2\cap Y_2, \cdots, X_n\cap Y_n)$
論理和	もし，$X\cap Y\neq\phi$ ならば
	$X\cup Y=[\min(\underline{X}, \underline{Y}), \max(\bar{X}, \bar{Y})]$
	もし区間ベクトル X, Y の対応するすべての要素が空ならば
	$X\cup Y=(X_1\cup Y_1, X_2\cup Y_2, \cdots, X_n\cup Y_n)$

2.3 区間の間に成り立つ関係式

区間の間に成り立つ関係式を表3に示す。

表3 区間の間に成り立つ関係式

推移的関係	$X<Y\Longleftrightarrow\bar{X}<\underline{Y}$
	$X>Y\Longleftrightarrow\underline{X}>\bar{Y}$
包含関係	$X\subseteq Y\Longleftrightarrow\underline{Y}\leq\underline{X}$ かつ $\bar{X}\leq\bar{Y}$
	$X\supseteq Y\Longleftrightarrow\underline{Y}\geq\underline{X}$ かつ $\bar{X}\geq\bar{Y}$
等値関係	$X=Y\Longleftrightarrow\underline{X}=\underline{Y}$ かつ $\bar{X}=\bar{Y}$

2.4 区間に対する数学関数

区間に対する数学関数を表4に示す。

表 4　区間に対する数学関数

区間 X の幅 w	$w(X)=\bar{X}-\underline{X}$
区間ベクトル X の幅 w	$w(X)=\max\{w(X_1), w(X_2), \cdots, w(X_n)\}$
区間マトリックス A の幅 w	$w(A)=\max_{ij} w(A_{ij})$
X の中点 m	$m(X)=\frac{1}{2}(\underline{X}+\bar{X})$
区間ベクトル X の中点 m	$m(X)=(m(X_1), m(X_2), \cdots, m(X_n))$
区間マトリックス A の中点 m	$m(A)=\begin{pmatrix} m(A_{11}) & m(A_{12}) & \cdots & m(A_{1n}) \\ m(A_{21}) & m(A_{22}) & \cdots & m(A_{2n}) \\ \vdots & \vdots & \vdots & \vdots \\ m(A_{n1}) & m(A_{n2}) & \cdots & m(A_{nn}) \end{pmatrix}$
X の絶対値 $\lvert X\rvert$	$\lvert X\rvert = \max(\lvert\underline{X}\rvert, \bar{X})$
区間ベクトル X のノルム $\lVert X\rVert$	$\lVert X\rVert = \max\{\lvert X_1\rvert, \lvert X_2\rvert, \cdots, \lvert X_n\rvert\}$
区間マトリックス A のノルム $\lVert A\rVert$	$\lVert A\rVert = \max_i \sum_j \lvert A_{ij}\rvert$
X の平方根	$\sqrt{X}=[\sqrt{\underline{X}}, \sqrt{\bar{X}}]$　ただし，$\underline{X}\geq 0$ かつ $\bar{X}\geq 0$
X の指数関数	$\exp X=[\exp\underline{X}, \exp\bar{X}]$
X の自然対数	$\log X=[\log\underline{X}, \log\bar{X}]$
X の sin 関数	$\sin X=[\min_{x\in X}(\sin x), \max_{x\in X}(\sin x)]$
X の cos 関数	$\cos X=[\min_{x\in X}(\cos x), \max_{x\in X}(\cos x)]$
X の tan 関数	$\tan X=[\tan\underline{X}, \tan\bar{X}]$　ただし，$\pi(2i-1)/2\in X$（i は整数）

3．区間解析ライブラリについて

区間解析ライブラリは，オブジェクト指向言語である C++ により作成した。C++ は，C と互換性を持ち，オブジェクト指向的機能を付加したものである。C++ には情報隠蔽，クラス継承，演算子の多重定義などがある。ライブラリを作成するにあたって，C++ を用いたのは，主にこの演算子の多重定義という概念があるためである。演算子の多重定義ができることにより，既存の演算子を，改めて別の意味のある演算子として定義することができる。例えば，$X=[0,1]$，$Y=[1,2]$ という区間のデータを持つオブジェクトを考える。この時，X と Y の加算はプログラム上で $X+Y$ とする事ができる。これは，一般に非常に分かり易く，結果が $[1,3]$ となることは容易に予想される。このように C++ の演算子の多重定義を用いライブラリを作成すれば，ユーザーがこれを利用する時，非常に使い易くなる。区間解析ライブラリは，3 つのクラスから構成される。すなわち，区間演算クラス，入出力クラス，エラークラスである。これらのクラスについて順次述べていく。

3．1　区間演算クラス

区間演算クラスは，6 つのクラスから構成される。そのうち 3 つのクラスは，

Interval クラス ⟶ I_Vector クラス ⟶ I_Matrix クラス

である。一番左が親クラスで，その右以降はその子クラスである。これらは，区間演算ク

ラスにおいて中核をなす部分である。残りの3つのクラスは,

I_Math クラス ⟶ IV_Math クラス ⟶ IM_Math クラス

である。これらのクラスは，区間演算における数学関数(区間関数)が定義されている。

3.1.1 Interval クラス

Interval クラスは，区間の演算ができるように定義されている。このクラスの使用法について以下に示す。

[オブジェクトの初期化]

C++ では，コンストラクタを用いることにより自動的にオブジェクトの初期化が可能である。Interval 型のオブジェクトを生成する時，引数をつけることで初期化できる。初期化の仕方は4種類あり，それらを表5に示す。

表5 Interval 型のオブジェクトの初期化

引数の数	引数†	プログラム上の例	
なし	—	Interval x;	//x = [0.0, 0.0]
1	dT	Interval x(1.0);	//x = [1.0, 1.0]
1	IT	Interval x(1.0);	//x を宣言
		Interval y(x);	//y = [1.0, 1.0]
2	dT	Interval x(1.0, 2.0);	//x = [1.0, 2.0]

†: dT: double, IT: Interval

ここでは，dT, IT をそれぞれ double, Interval 型のデータ型としている。このように，引数無しか又は引数を付けることで異なった初期化が可能である。

[演算子の使用法]

Interval クラスには，21個の演算子が多重定義されている。これらの演算子を使用することにより，2節で示したような演算が可能となる。まず，四則演算に関する演算子の使用法を表6に示す（2.1を参照)。

ここでは，x, y, z をそれぞれ x = [1.0, 2.0], y = [2.0, 3.0], z = [0.0, 0.0] の区間を持つ Interval 型のオブジェクトとしている。この表において，例えば「+」演算子は，左，右オペランドの型が dT 又は IT となっているが，これは

(dT)+(IT), (IT)+(dT) 又は (IT)+(IT)

の3つの型の組み合わせが可能であることを示している。つまり，左右オペランドのうちの一方の型は IT で，もう一方は dT か又は IT でなければならない。

次に，集合演算に関する演算子の使用法を表7に示す（2.2を参照)。

表6　四則演算に関する演算子の使用法

演算子	意味	左オペランド†	右オペランド†	返値†	プログラム上の例‡
=	代入	IT	dT, IT	IT	z = x; //z = [1.0, 2.0]
+	加算	—	IT	IT	z = +x; //z = [1.0, 2.0]
+	〃	dT, IT	dT, IT	IT	z = x+y; //z = [3.0, 5.0]
+=	〃	IT	dT, IT	IT	z += y; //z = [2.0, 3.0]
++	〃	IT	—	IT	++x; //x = [2.0, 3.0]
++	〃	—	IT	IT	x++; //x = [2.0, 3.0]
−	減算	—	IT	IT	z = −x; //z = [−2.0, −1.0]
−	〃	dT, IT	dT, IT	IT	z = x−y; //z = [−2.0, 0.0]
−=	〃	IT	dT, IT	IT	z −= x; //z = [−3.0, −2.0]
−−	〃	IT	—	IT	−−x; //x = [0.0, 1.0]
−−	〃	—	IT	IT	x−−; //x = [0.0, 0.0]
*	乗算	dT, IT	dT, IT	IT	z = x*y; //z = [2.0, 6.0]
*=	〃	IT	dT, IT	IT	z *= y; //z = [0.0, 0.0]
/	除算	dT, IT	dT, IT	IT	z = x / y; //z = [0.3, 1.0]
/=	〃	IT	dT, IT	IT	z /= y; //z = [0.0, 0.0]

†: 型　‡: x = [1.0, 2.0], y = [2.0, 3.0], z = [0.0, 0.0]

表7　集合演算に関する演算子の使用法

演算子	意　味	左オペランド†	右オペランド†	返値†	プログラム上の例‡
&	論理積	dT, IT	dT, IT	IT	z = x & y; //z = [2.0, 3.0]
&=	〃	IT	dT, IT	IT	z &= y; //z = [0.0, 0.0]
|	論理和	dT, IT	dT, IT	IT	z = x | y; //z = [1.0, 4.0]
|=	〃	IT	dT, IT	IT	z |= y; //z = [0.0, 0.0]

†: 型　‡: x = [1.0, 3.0], y = [2.0, 4.0], z = [0.0, 0.0]

表7において，もしも共通集合が無く空集合となった時，0が返されることに注意する。（この時，空集合となったことを警告する。3．3を参照。）

次に，区間における関係式に関する演算子の使用法を表8に示す（2．3を参照）。

ここでは，返されるデータ型はすべてiT: intである（データは0又は1のみ返される）。

3．1．2　I_Vectorクラス

I_Vectorクラスは，区間ベクトルにおける演算を提供している。このクラスの使用法について以下に示す。

［オブジェクトの初期化］

I_Vector型のオブジェクトの初期化の仕方は6種類あり，それらを表9に示す。

ここでは2つのことに注意する。1つは，引数無しの場合2つの要素を持つベクトルに初

表8 関係式に関する演算子の使用法

演算子	意味	左オペランド†	右オペランド†	返値†	プログラム上の例‡
==	等値関係	dT, IT	dT, IT	iT	w = x == y; //w = [0.0, 0.0]
!=	〃	dT, IT	dT, IT	iT	w = x != y; //w = [1.0, 1.0]
<	推移的関係	dT, IT	dT, IT	iT	w = x < y; //w = [1.0, 1.0]
>	〃	dT, IT	dT, IT	iT	w = x > y; //w = [0.0, 0.0]
<=	包含関係	dT, IT	dT, IT	iT	w = x <= z; //w = [1.0, 1.0]
>=	〃	dT, IT	dT, IT	iT	w = x >= y; //w = [0.0, 0.0]

†: 型 iT: int ‡: x = [1.0, 2.0], y = [3.0, 4.0], z = [1.1, 1.5], w = [0.0, 0.0]

表9 I_Vector 型のオブジェクトの初期化

引数の数	引数†	プログラム上の例	
なし	—	I_Vector vx;	//vx = ([0.0, 0.0], [0.0, 0.0])
1	iT	I_Vector vx(2);	//vx = ([0.0, 0.0], [0.0, 0.0])
1	IVT	I_Vector vx(2);	//vy = ([0.0, 0.0], [0.0, 0.0])
2	順に iT, dT	I_Vector vx(2, 1.0);	//vx = ([1.0, 1.0], [1.0, 1.0])
2	順に iT, IT	Interval x(1.0, 2.0);	//x = [1.0, 2.0] を宣言
		I_Vector vx(2, x);	//vx = ([1.0, 2.0], [1.0, 2.0])
3	順に iT, dT, dT	I_Vector vx(2, 1.0, 2.0);	//vx = ([1.0, 2.0], [1.0, 2.0])

†: 型 IVT: I_Vector

期化されるという点である。2つ目は，第一引数の型は iT であるという点である。なぜなら，それはベクトルの要素を決める数だからである。

［演算子の使用法］

I_Vector クラスには，21個の演算子が多重定義されている。これらの演算子を使用することにより，区間ベクトルの演算が可能となる。まず，区間ベクトルの四則演算に関する演算子の使用法を表10に示す。

ここでは3つのことに注意する。1つは，次のような点である。例えば「+」演算子に関するプログラム上の例で

$$vz = 1.0 + vx;$$

とある。この場合，左オペランドのデータの型は dT であるが，これは2つの要素を持つ区間ベクトル，即ち，([1.0, 1.0], [1.0, 1.0]) として考えなければならないという点である。IT に対しても同様である。2つ目は，ベクトルの各要素への参照は括弧演算子である ( ) を用いるという点である。ただし，返値の型は Interval 型なので，各要素へ代入するデータ型は dT 又は IT でなくてはならない。また，ベクトルの最初の要素は0から始まる。3つ目は，I_Vector クラスにおける「++」又は「--」演算子の使用法は，オペランド

表10 四則演算に関する演算子の使用法

演算子	意味	左オペランド†	右オペランド†	返値†	プログラム上の例‡
( )	要素の参照	IVT	iT	IT	vz(1) = x; //vz =([0.0, 0.0], [1.0, 2.0])
=	代入	IT	dT,IT,IVT	IVT	vz = vx; //vz =([1.0, 2.0], [1.0, 2.0])
+	加算	—	IVT	IVT	vz = +vx; //vz =([1.0, 2.0], [1.0, 2.0])
+	〃	dT,IT,IVT	dT,IT,IVT	IVT	vz = vx+vy; //vz =([3.0, 5.0], [3.0, 5.0])
+=	〃	IVT	dT,IT,IVT	IVT	vz += vx; //vz =([1.0, 2.0], [1.0, 2.0])
++	〃	IVT	—	IVT	++vx; //vx =([2.0, 3.0], [2.0, 3.0])
++	〃	—	IVT	IVT	vx++; //vx =([2.0, 3.0], [2.0, 3.0])
−	減算	—	IVT	IVT	vz = −vx; //vz =([−2.0, −1.0], [−2.0, −1.0])
−	〃	dT,IT,IVT	dT,IT,IVT	IVT	vz = vx−vy; //vz =([−2.0, 0.0], [−2.0, 0.0])
−=	〃	IVT	dT,IT,IVT	IVT	vz −= vx; //vz =([−2.0, −1.0], [−2.0, −1.0])
−−	〃	IVT	—	IVT	−−vx; //vx =([0.0, 1.0], [0.0, 1.0])
−−	〃	—	IVT	IVT	vx−−; //vx =([0.0, 1.0], [0.0, 1.0])
*	乗算	dT,IT,IVT	dT,IT,IVT	IVT	vz = vx*vy; //vz =([2.0, 6.0], [2.0, 6.0])
*=	〃	IVT	dT,IT,IVT	IVT	vz *= vx; //vz =([0.0, 0.0], [0.0, 0.0])
/	除算	dT,IT,IVT	dT,IT,IVT	IVT	vz = vx / vy; //vz =([0.3, 1.0], [0.3, 1.0])
/=	〃	IVT	dT,IT,IVT	IVT	vz /= vx; //vz =([0.0, 0.0], [0.0, 0.0])

†: 型 ‡: vx =([1.0, 2.0], [1.0, 2.0]), vy =([2.0, 3.0], [2.0, 3.0]), vz =([0.0, 0.0], [0.0, 0.0])

表11 集合演算に関する演算子の使用法

演算子	意味	左オペランド†	右オペランド†	返値†	プログラム上の例‡
&	論理積	dT,IT,IVT	dT,IT,IVT	IVT	vz = vx & vy; //vz =([2.0, 3.0], [2.0, 3.0])
&=	〃	IVT	dT,IT,IVT	IVT	vz &= vy; //vz =([0.0, 0.0], [0.0, 0.0])
|	論理和	dT,IT,IVT	dT,IT,IVT	IVT	vz = vx | vy; //vz =([1.0, 4.0], [1.0, 4.0])
|=	〃	IVT	dT,IT,IVT	IVT	vz |= vy; //vz =([0.0, 0.0], [0.0, 0.0])

†: 型 ‡: vx =([1.0, 3.0], [1.0, 3.0]), vy =([2.0, 4.0], [2.0, 4.0]), vz =([0.0, 0.0], [0.0, 0.0])

の各要素に1加算又は2減算すると解釈されるという点である。

次に，集合演算に関する演算子の使用法を表11に示す（2．2を参照)。

次に，区間ベクトルにおける関係式に関する演算子の使用法を表12に示す(2．3を参照)。

ここでは，返されるデータ型はすべてiT型である。

3．1．3 I_Matrixクラス

I_Matrixクラスは，区間マトリックスにおける演算を提供している。このクラスの使用法について以下に示す。

［オブジェクトの初期化］

I_Matrix型のオブジェクトの初期化の仕方は6種類あり，それらを表13に示す。

表12 関係式に関する演算子の使用法

演算子	意味	左オペランド†	右オペランド†	返値†	プログラム上の例‡
==	等値関係	dT,IT,IVT	dT,IT,IVT	iT	vw = vx == vy; //vw = ([0.0, 0.0], [0.0, 0.0])
!=	〃	dT,IT,IVT	dT,IT,IVT	iT	vw = vx != vy; //vw = ([1.0, 1.0], [1.0, 1.0])
<	推移的関係	dT,IT,IVT	dT,IT,IVT	iT	vw = vx < vy; //vw = ([1.0, 1.0], [1.0, 1.0])
>	〃	dT,IT,IVT	dT,IT,IVT	iT	vw = vx > vy; //vw = ([0.0, 0.0], [0.0, 0.0])
<=	包含関係	dT,IT,IVT	dT,IT,IVT	iT	vw = vx <= vz; //vw = ([1.0, 1.0], [1.0, 1.0])
>=	〃	dT,IT,IVT	dT,IT,IVT	iT	vw = vx >= vy; //vw = ([0.0, 0.0], [0.0, 0.0])

†: 型　‡: vx = ([1.0, 2.0], [1.0, 2.0]), vy = ([3.0, 4.0], [3.0, 4.0]), vz = ([1.1, 1.5], [1.5, 1.5]), vw = ([0.0, 0.0], [0.0, 0.0])

表13 I_Matrix 型のオブジェクトの初期化

引数の数	引数†	プログラム上の例	
なし	—	I_Matrix mx;	//mx = ([0.0, 0.0], [0.0, 0.0], //[0.0, 0.0], [0.0, 0.0])
1	IT	I_Matrix mx;	//mx を宣言
		I_Matrix my(mx);	//my = ([0.0, 0.0], [0.0, 0.0], //[0.0, 0.0], [0.0, 0.0])
2	順に iT,iT	I_Matrix mx(2, 2);	//mx = ([0.0, 0.0], [0.0, 0.0], //[0.0, 0.0], [0.0, 0.0])
3	順に iT,iT,dT	I_Matrix mx(2, 2, 1.0);	//mx = ([1.0, 1.0], [0.0, 0.0], //[0.0, 0.0], [1.0, 1.0])
3	順に iT,iT,IT	Interval x(1.0, 2.0);	//mx を宣言
		I_Matrix mx(2, 2, x);	//mx = ([1.0, 2.0], [0.0, 0.0], //[0.0, 0.0], [1.0, 2.0])
4	順に iT,iT,dT,dT	I_Matrix mx(2, 2, 1.0, 2.0);	//mx = ([1.0, 2.0], [0.0, 0.0], //[0.0, 0.0], [1.0, 2.0])

†: 型

ここで注意する点は，2つある。1つは，引数を付ける場合，第一，第二引数の型は iT でなければならない点である。2つ目は，次のようなものである。プログラム上の例，とりわけ引数なしのオブジェクト初期化の例において，mx の要素をコメントとして次のように示している。

```
//mx = ([0.0, 0.0], [0.0, 0.0],
//[0.0, 0.0], (0.0, 0.0])
```

これは，上が mx の1行目を示し，下が mx の2行目を示しているという点である。

［演算子の使用法］

I_Matrix クラスには，11個の演算子が多重定義されている。これらの演算子を使用することにより，区間マトリックスの演算が可能となる。区間マトリックスの四則演算に関する演算子の使用法を表14に示す。

表14　四則演算における使用法

演算子	意味	左オペランド†	右オペランド†	返値†	プログラム上の例‡
( )	要素の参照	IMT	iT	IT	mz(0, 1) = x; //mz = ([0.0, 0.0], [1.0, 2.0], //[0.0, 0.0], [0.0, 0.0])
=	代入	IT	dT,IT,IVT,IMT	IMT	mz = mx; //mz = ([1.0, 2.0], [1.0, 2.0], //[1.0, 2.0], [1.0, 2.0])
+	加算	—	IMT	IMT	mz = +mx; //mz = ([1.0, 2.0], [1.0, 2.0], //[1.0, 2.0], [1.0, 2.0])
+	〃	dT,IT,IVT,IMT	dT,IT,IVT,IMT	IMT	mz = mx+my; //mz = ([3.0, 5.0], [3.0, 5.0], //[3.0, 5.0], [3.0, 5.0])
+=	〃	IMT	dT,IT,IVT,IMT	IMT	mz += mx; //mz = ([1.0, 2.0], [1.0, 2.0], //[1.0, 2.0], [1.0, 2.0])
++	〃	IMT	—	IMT	++mx; //mx = ([2.0, 3.0], [1.0, 2.0], //[1.0, 2.0], [2.0, 3.0])
++	〃	—	IMT	IMT	mx++; //mx = ([2.0, 3.0], [1.0, 2.0], //[1.0, 2.0], [2.0, 3.0])
−	減算	—	IMT	IMT	mz = −mx; //mz = ([−2.0, −1.0], [−2.0, −1.0], //[−2.0, −1.0], [−2.0, −1.0])
−	〃	dT,IT,IVT,IMT	dT,IT,IVT,IMT	IMT	mz = mx−my; //mz = ([−2.0, 0.0], [−2.0, 0.0], //[−2.0, 0.0], [−2.0, 0.0])
−=	〃	IMT	dT,IT,IVT,IMT	IMT	mz −= mx; //mz = ([−2.0, −1.0], [−2.0, −1.0], //[−2.0, −1.0], [−2.0, −1.0])
−−	〃	IMT	—	IMT	−−mx; //mx = ([0.0, 1.0], [1.0, 2.0], //[1.0, 1.0], [0.0, 1.0])
−−	〃	—	IMT	IMT	mx−−; //mx = ([0.0, 1.0], [1.0, 2.0], //[1.0, 1.0], [0.0, 1.0])
*	乗算	dT,IT,IVT,IMT	dT,IT,IVT,IMT	IMT	mz = mx*my; //mz = ([2.0, 6.0], [2.0, 6.0], //[2.0, 6.0], [2.0, 6.0])
*=	乗算	IMT	dT,IT,IVT,IMT	IMT	mz *= mx; //mz = ([0.0, 0.0], [0.0, 0.0], //[0.0, 0.0], [0.0, 0.0])
/	除算	dT,IT,IVT,IMT	dT,IT,IVT,IMT	IMT	mz = mx / my; //mz = ([0.3, 1.0], [0.3, 1.0], //[0.3, 1.0], [0.3, 1.0])
/=	除算	IMT	dT,IT,IVT,IMT	IMT	mz /= mx; //mz = ([0.0, 0.0], [0.0, 0.0], //[0.0, 0.0], [0.0, 0.0])

†: 型　IMT: I_Matrix
‡: x = [1.0, 2.0], vx = ([0.0, 1.0], [0.0, 1.0]), mx, my, mz: 2×2 のマトリックス,
mx = ([1.0, 2.0], [1.0, 2.0], [1.0, 2.0], [1.0, 2.0]), my = ([2.0, 3.0], [2.0, 3.0], [2.0, 3.0], [2.0, 3.0]), mz = ([0.0, 0.0], [0.0, 0.0], [0.0, 0.0], [0.0, 0.0])

I_Matrix クラスにおける「++」又は「−−」演算子の使用法は，マトリックスの対角要素に対して 1 加算又は減算する。

3．1．4　I_Math クラス

I_Math クラスは，Interval 型のオブジェクトに対する数学関数の使用を提供している。このクラスには，7 個の関数が多重定義され，5 個の関数が新たに定義されている。これら数学関数の使用法を表15に示す（2．4 を参照）。

表15 数学関数の使用法

関数	第一引数†	第二引数†	返値†	プログラム上の例‡
sqrt	IT	—	IT	z = sqrt(x); //z = [1.0, 1.4]
cos	IT	—	IT	z = cos(x); //z = [−1.0, 1.0]
sin	IT	—	IT	z = sin(x); //z = [0.0, 1.0]
tan	IT	—	IT	z = tan(x/3); //z = [0.0, 1.7]
exp	IT	—	IT	z = exp(x); //z = [2.8, 7.8]
log	IT	—	IT	z = log(x); //z = [0.0, 0.6]
inf	IT	—	dT	z = inf(x); //z = [1.0, 1.0]
sup	IT	—	dT	z = sup(x); //z = [2.0, 2.0]
mid	IT	—	dT	z = mid(x); //z = [1.5, 1.5]
width	IT	—	dT	z = width(x); //z = [1.0, 1.0]
iabs	IT	—	dT	z = iabs(x); //z = [2.0, 2.0]
pow	IT	iT	IT	z = pow(x, 2); //z = [1.0, 4.0]

†: 型 ‡: x = [1.0, 2.0], z = [0.0, PI]

表16 数学関数の使用法

関数	第一引数†	返値†	プログラム上の例‡
mid	IVT	dT	vz = mid(vx); //vz = ([1.5, 1.5], [1.5, 1.5])
width	IVT	dT	vz = width(vx); //vz = ([1.0, 1.0], [1.0, 1.0])
norm	IVT	dT	vz = norm(vx); //vz = ([2.0, 2.0], [2.0, 2.0])

†: 型 ‡: vx = ([1.0, 2.0], [1.0, 2.0]), vz = ([0.0, 0.0], [0.0, 0.0])

3.1.5 IV_Math クラス

IV_Math クラスは，I_Vector 型のオブジェクトに対する数学関数の使用を提供している。このクラスには，2個の関数が多重定義され，1個の関数が新たに定義されている。これら数学関数の使用法を表16に示す（2.4を参照）。

3.1.6 IM_Math クラス

IM_Math クラスは，I_Matrix 型のオブジェクトに対する数学関数の使用を提供している。このクラスには，3個の関数が多重定義され，これら数学関数の使用法を表17に示す（2.4を参照）。

3.2 入出力クラス

入出力クラスは，3つのクラスから成っている。それらは

I_IOStream クラス ⟶ IV_IOStream クラス ⟶ IM_IOStream クラス

表17 数学関数の使用法

関数	第一引数†	返値†	プログラム上の例‡
mid	IMT	dT	mz = mid(mx); //mz = ([1.5, 1.5], [1.5, 1.5], //[1.5, 1.5], [1.5, 1.5])
width	IMT	dT	mz = width(mx); //mz = ([1.0, 1.0], [1.0, 1.0], //[1.0, 1.0], [1.0, 1.0])
norm	IMT	dT	mz = norm(mx); //mz = ([4.0, 4.0], [4.0, 4.0], //[4.0, 4.0], [4.0, 4.0])

†: 型 ‡: mx, mz は 2×2 のマトリックス,
mx = ([1.0, 2.0], [1.0, 2.0], , mz = ([0.0, 0.0], [0.0, 0.0],
//[1.0, 2.0], [1.0, 2.0]) //[0.0, 0.0], [0.0, 0.0])

表18 Interval オブジェクトに対する入出力の使用法

	演算子	左オペランド†	右オペランド†	返値†	形式
入力‡	>>	isT	IT	isT	dT, , dT
出力‡	<<	osT	IT	osT	浮動小数点(桁数 15)

†: 型 isT: istream, osT: ostream, x: x = [1.0, 2.0] のデータを持つ Interval 型オブジェクト
‡: プログラム上の例
入力 cin >> x; //1.0, , 2.0, と入力
出力 cout << x; //[1.0 ··· 0e+00, 1.0 ··· 0e+00] と出力

である。これらのクラスは，区間の入出力の処理を行う。

3.2.1 I_IOStream クラス

I_IOStream クラスは，Interval 型のオブジェクトに対する入出力を可能にしている。これらのクラスの使用法を表18に示す。

isT: istream, osT: ostream は，それぞれ C++ に標準で用意されている iostream ライブラリの中にあるクラスである。入出力には，その他に用意されているクラスのオブジェクト cin, cout を用いる。入力の仕方は，ディスプレイ上で double 型のオブジェクトの入力後，続けてコンマを 2 つ付けさらに double 型のオブジェクトを入力する。入力形式を誤ると再度入力し直す。出力については，浮動小数点形式で小数点以下15桁まで表示する。

3.2.2 IV_IOStream クラス

IV_IOStream クラスは，I_Vector 型のオブジェクトに対する入出力処理が可能となるように定義されている。表19にそれらの使用法を示す。

I_Vector 型のオブジェクトに対する入力の仕方は，表18で示した形式でベクトルの要素の数だけ入力する。逆に，出力は要素の数だけ横に続けて出力する。

表19 I_Vector オブジェクトに対する入出力の使用法

	演算子	左オペランド†	右オペランド†	返値†	形　式
入力‡	>>	isT	IT	isT	要素数だけ入力
出力‡	<<	osT	IT	osT	要素数だけ横に入力

†: 型　isT: istream,　osT: ostream,　vx = ([1.0, 2.0], [1.0, 2.0])
‡: プログラム上の例
　入力 cin >> vx; //1.0, , 2.0 1.0, , 2.0 と入力
　出力 cout << vx; //([1.0 ··· 0e+00, 1.0 ··· 0e+00], [1.0 ··· 0e+00, 1.0 ··· 0e+00]) と出力

表20 I_Matrix オブジェクトに対する入出力の使用法

	演算子	左オペランド†	右オペランド†	返値†	形　式
入力‡	>>	isT	IT	isT	行毎に入力
出力‡	<<	osT	IT	osT	行毎に改行し出力

†: 型　isT: istream,　osT: ostream,　mx: 2×2 のマトリックス, mx = ([1.0, 2.0], [2.0, 3.0],
[3.0, 4.0], [4.0, 5.0])
‡: プログラム上の例
　入力 cin >> mx; //1.0, , 2.0 2.0, , 3.0 3.0, , 4.0 4.0, , 5.0 と入力
　出力 cout << mx; //([1.0 ··· 0e+00, 2.0 ··· 0e+00], [2.0 ··· 0e+00, 3.0 ··· 0e+00],
//[3.0 ··· 0e+00, 4.0 ··· 0e+00], [4.0 ··· 0e+00, 5.0 ··· 0e+00]) と出力

3.2.3 IM_IOStream クラス

IM_IOStream クラスは，I_Matrix 型のオブジェクトに対する入出力処理が可能となるように定義されている。表20にそれらの使用法を示す。

ここでの入力形式は，1行目の最初の要素から逐次入力し，2行目3行目と続く。逆に出力は，1行目の要素を出力した後改行し，次に2行目を出力する。

3.3 エラークラス

エラークラスは，3つのクラスから成っている。それらは，

I_Error クラス ⟶ IV_Error クラス ⟶ IM_Error クラス

である。これらのクラスは，ユーザーに対するエラー表示，またそれぞれのエラーに応じた処理を行う。

3.3.1 I_Error クラス

I_Error クラスは，区間演算に対するエラーの処理を行う。このクラスは，表21に示すように他のオブジェクトから5種類のメッセージのうち1つが送られてくると，そのメッセージの種類に応じて，エラー処理を行う。

表21 I_Error クラスのエラー処理

メッセージ	メッセージの意味	エ ラ ー 処 理	プログラム上の例‡
Divide_By0	0 を含む区間で除算	エラー出力し強制終了	z = x / z;
Interchange	区間のパラメータの設定の誤り	エラー出力し強制終了	Interval y(1, 0);
Empty_Set	空集合	警告の出力	z = x & z;
Input	データ入力の形式が誤り	エラー出力し再度入力	1, 2 と入力
Power_Negative	べき乗の数が負	エラー出力し強制終了	z = pow(x, −2);

‡: x = [1.0, 2.0], z = [0.0, 0.0]

表22 IV_Error クラスのエラー処理

メッセージ	メッセージの意味	エ ラ ー 処 理	プログラム上の例‡
Inconsistent	ベクトルの大きさが不一致	エラー出力し強制終了	vx+vw;
Memory_Over	メモリの確保不可	エラー出力し再度入力	I_Vector vy(10000);

‡: vx = ([1.0, 2.0], [1.0, 2.0]), vz = ([0.0, 0.0], [0.0, 0.0]), vw = ([2.0, 3.0], [2.0, 3.0], [2.0, 3.0])

表23 IM_Error クラスのエラー処理

メッセージ	メッセージの意味	エ ラ ー 処 理	プログラム上の例‡
D_InconsisTent	マトリックスの大きさが不一致	エラー出力し強制終了	mx+my;

‡: mx, my は 2×2 のマトリックス, mx = ([1.0, 2.0], [1.0, 2.0], , my = ([0.0, 0.0], [0.0, 0.0],
[1.0, 2.0], [1.0, 2.0]) [0.0, 0.0], [0.0, 0.0],
[0.0, 0.0], [0.0, 0.0])

3.3.2 IV_Error クラス

IV_Error クラスは, 区間ベクトルの演算に対するエラー処理を行う。このクラスも同様に, 表22で示されるメッセージの種類に応じて, それぞれエラー処理を行う。メッセージ, Inconsistent に対するエラーが発生するのは, 左オペランドと右オペランドのベクトルの大きさが異なる時である。

3.3.3 IM_Error クラス

IM_Error クラスは, 区間マトリックスに対するエラー処理を行う。このクラスも同様に, 表23で示されるメッセージの種類に応じて, エラー処理を行う。

3.4 定義されたクラスの格納場所

既に述べた各クラスのプロトタイプ, その内容は, それぞれ表24で示されるファイルで定義されている。

表24 各クラスを定義したファイル

プロトタイプ	プロトタイプの内容	クラス
INTERVAL. H	INTERVAL. CPP	Interval, I_Vector, I_Matrix
IMATH. H	IMATH. CPP	I_Math, IV_Math, IM_Math
IIOSTRM. H	IIOSTRM. CPP	I_IOStream, IV_IOStream, IM_IOStream
IERROR. H	IERROR. CPP	I_Error, IV_Error, IM_Error

4. 区間解析ライブラリの使用例

この節では，前節で述べたライブラリを実際に使用した例を述べる。非線形方程式の根を見つけるための方法としていろいろな反復法がある。その中でも，一般に収束の速い反復法として知られるニュートン法がある。このニュートン法を区間という概念に拡張したものが区間ニュートン法である[3,4]。今，ある連続で微分可能な関数 $f(x)$ に対して $f(x)=0$ の解を見つけるための問題を考える。また，$f(x)$ とその導関数 $f'(x)$ に対して，区間に拡張した $F(X)$, $F'(X)$ を考える。以下に区間ニュートン法のアルゴリズムを示す。

［アルゴリズム］

もしも $0 \notin F'(X^{(0)})$ ならば，$X^{(k+1)}=X^{(k)}$ となるまで以下の反復公式を繰り返す。

$$N(X^{(k)}) := m(X^{(k)}) - \frac{f(m(X^{(k)}))}{F'(X^{(k)})}$$
$$X^{(k+1)} := X^{(k)} \cap N(X^{(k)}) \qquad (k=0,1,2,\cdots)$$

［例］

初期区間 $X_0=[1.0, 2.5]$ における $f(x)=\sin x+\cos x=0$ の根を求めるプログラムと実行結果を以下に示す。

［プログラム］

```
#include "interval.cpp"
#include "iiostrm.cpp"
#include "imath.cpp"

double f( double& x ){ return sin(x) + cos(x); };//関数fの定義
Interval deriv( Interval& x ){ return cos(x) - sin(x); };//導関数f'の定義
int criter( Interval& x ){ return  !( 0.0 <= deriv(x) ); };//判定条件の定義

void main ( void ){
    Interval   x, old_x, n;
    double     m;  // 区間の中点
    int        IterCount=0;  //反復回数
    cin >> x; // 区間の入力
```

```
    if ( criter(x) ){    //条件を満たすかどうか判定する
        do{
            cout << " nx" << "^("<<IterCount << ")=" << x << endl;
            old_x = x;
            m = mid(x);
            n = m - f(m) / deriv(x);
            x = n & x;
            IterCount ++;
        } while(x  != old_x); // 終了条件
        cout << "反復回数:" << IterCount  << "近似区間:" << x;
    }
    else cout << "Criterion not satisfied !¥n";
}
```

[実行結果]

```
x^(0)=[1.000000000000000e+00, 2.500000000000000e+00]
x^(1)=[2.197349053273453e+00, 2.500000000000000e+00]
x^(2)=[2.355275043201859e+00, 2.357650289344867e+00]
x^(3)=[2.356194171125610e+00, 2.356194808111474e+00]
x^(4)=[2.356194490192345e+00, 2.356194490192345e+00]
x^(5)=[2.356194490192345e+00, 2.356194490192345e+00]
反復回数:6  近似区間:[2.356194490192345e+00, 2.356194490192345e+00]
```

5. 結　　論

作成されたライブラリを用いる事により，区間，区間ベクトル，区間マトリックスに対する演算が可能である。また，ライブラリを C++ で作成する事によって，これを利用する時，非常にプログラミングが容易になる。今後，更に信頼性を高め充実したライブラリにする必要がある。

参考文献

1) Ramon E, Moore, Method And Applications Of Interval Analysis. Siam. Philadelphia, (1979).
2) 山下　浩，黒羽裕章，黒岩健太郎，1992 C++ プログラミングスタイル.
3) R. Klatte, U. Kulish, A. Wiethoff, C. Lawo, M. Rauch. C-XSC. Springer-Verlag, (1993).
4) R. Haumer, M. Hocks, V. Kulisch, D. Ratz. Numerical Toolbox for Verified Computing I.

Interval Arithmetic Library with C++

Tetsuo TOGAWA and Michio SAKAKIHARA*
Graduate School of Science,
**Department of Applied Mathematics,*
Faculty of Science,
Okayama University of Science,
Ridai-cho 1-1, Okayama 700, Japan
(Received September 30, 1995)

The interval arithmetic has been applied to some fields of engineering and science. In order to develop a programming to solve those problems we present a convenient programming tool in this paper. The construction of the library and an example of its application is introduced.