数理解析研究所講究録
1147巻 2000年 5-15

高速精度保証付き数値計算

早稲田大学理工学部情報学科　大石　進一 (Shin'ichi Oishi)

1　はじめに

近年、区間解析の理論の進展により、数値計算のほとんどすべての分野について、近似解が与えられたとき、数値計算の誤差を完全に評価して、数学的に正しい結論を導けることが明らかにされつつある。

このようなことから、近年、精度保証付き数値計算の計算複雑度をいかに近似計算の計算複雑度とあまりかわらないレベルまでに減らすかという問題に大きな興味がもたれ始めている。本稿では、連立一次方程式の場合には、精度保証に要する手間は近似解をLU分解で求める手間と同じ計算量となることを示す。また、非線形方程式の近似解の精度保証付き数値計算を行う計算機環境を数値計算言語Scilab上に構築したので、精度保証付き数値計算が具体的にどのように行われるかの例として紹介したい。

2　浮動小数点数

準備として浮動小数点数の議論からスタートしよう。

2進数の浮動小数点数については1960年代にはいろいろな方式があったが、1985年にIEEE754という規格ができて、ほとんどすべてのCPUの浮動小数点プロセッサはこの規格に従うようになっている。提案グループの中心にKahanという優れた数値計算の学者がいたので、IEEE754規格は理論的にすっきりとした美しい体系をなしている。実世界においてもIntelのPentiumを始めとして、ほとんどのCPUがこの規格に従っていて、例外を捜すのが難しいくらいである。INTELのPentium CPUもこの規格に従っている。本稿では，浮動小数点数の集合を $\boldsymbol{F}$ で表す。IEEE 754では，四つの丸めのモードが指定できるように，コンピュータが設計されていることを要請している。その内の3つは次のように定義される。c を実数 ($c \in \boldsymbol{R}$) とする。

(1) 上向きの丸め　c 以上の浮動小数点数の中で最も小さい数に丸める。これを $\triangle : \boldsymbol{R} \to \boldsymbol{F}$ と表す。

(2) 下向きの丸め　c 以下の浮動小数点数の中で最も大きい数に丸める。これを $\nabla : \boldsymbol{R} \to \boldsymbol{F}$ と表す。

(3) 最近点への丸め　c に最も近い浮動小数点数に丸める。これを $\square : \boldsymbol{R} \to \boldsymbol{F}$ と表す。もし，このような点が2点ある場合には，仮数部の最後のビットが偶数である浮動小数点数に丸める。これを**偶数丸め方式** (round to even) という。

丸めの演算を写像として $\bigcirc : \boldsymbol{R} \to \boldsymbol{F}$ と書く。すなわち，$\bigcirc$ は $\triangle$，∇，$\square$ のいずれかと考える。IEEE 754では浮動小数点数演算 ($\boldsymbol{F}$ 上での四則演算) は丸めとの関係によりつぎのように定義されている。$\cdot \in \{+, -, \times, /\}$，$\bigcirc \in \{\triangle, \nabla, \square\}$ のとき

$$x \bigodot y = \bigcirc(x \cdot y) \text{ (任意の } x, y \in \mathbf{R} \text{ について)} \tag{1}$$

この式は，左辺の浮動小数点数の四則演算の結果 $x \bigodot y$ は，右辺の数学的に正しい (実数としての) 四則演算の結果 $x \cdot y$ を指定された丸めを行って得られた数 $\bigcirc(x \cdot y)$ に一致するように計算する規格を表している。

また，平方根も

$$(\sqrt{x})_{fp} = \bigcirc(\sqrt{x}) \text{ (任意の } x \in \boldsymbol{F} \text{ について)} \tag{2}$$

と，浮動小数点数演算によって計算された平方根 $(\sqrt{x})_{fp}$ は，正確な実数演算で計算された平方根 $\sqrt{x}$ を指定された丸めの方向へ丸めた数となる規格である。注意すべきことは，指数関数や三角関数などはこの

ような規格をみたすようには規定されていないことである。これはこれらの初等関数の値を精度保証付きで求めるためには別途工夫が必要であることを意味する。

C 言語では，このような丸めの制御ができる。

3 連立一次方程式の数値解の高速精度保証

連立一次方程式

$$Ax = b \tag{3}$$

は通常 LU 分解によって近似解が計算される。A が $n \times n$ 行列とすると，計算量は $n^3/3$ である。著者は，Rump 教授と共同で、近似解を求めて，さらに精度保証をするのに合計で $2n^3/3$ の計算量で済む高速精度保証法を考案した [?]。本章ではその成果を述べる。ただし，計算量の単位は倍精度浮動小数点数の乗除算を 1 回とする単位 (flops) である。

この高速精度保証法は，事前誤差解析を利用する。そこで，つぎに，必要となる事前誤差解析に関する議論をしておこう。

3.1 事前誤差解析

3.1.1 LU 分解計算の誤差の事前評価

$n \times n$ 行列 A, $a_{i,j} \in \boldsymbol{F}$ の LU 分解をガウスの消去法で求めたとする。ただし，丸めのモードは最近点への丸めで，途中に現れた数がすべて規格化浮動小数点数であったとする。このとき，計算された LU 分解の近似 $\tilde{L}$ と $\tilde{U}$ に対して次が成立する。

$$|\tilde{L}\tilde{U} - A| \leq \gamma_n |\tilde{L}||\tilde{U}| \tag{4}$$

この結果は N.J.Higham:Accuracy and Stability of Numerical Algorithms, SIAM (1996) による。

3.1.2 三角行列の逆行列計算の誤差の事前評価

$L \in \boldsymbol{R}^{n\times n}$ を下三角行列とする。丸め誤差が存在しないときには消去法によって計算した逆行列 X は左逆行列にも右逆行列にもなっている。

$$XL = I, \quad LX = I \tag{5}$$

しかし，丸め誤差が存在する場合，逆行列の近似 $\tilde{X}$ の右残差と左残差は一致しない。

$$|\tilde{X}L - I| \neq |L\tilde{X} - I| \tag{6}$$

ここでは，後に利用するので，左残差 $|\tilde{X}L - I|$ の事前誤差評価式を導こう。

下三角行列 L の逆行列 X の求め方としてつぎのような方法を用いる。今，$M \in \boldsymbol{R}^{m\times m}$ を下三角行列，$N \in \boldsymbol{R}^{m\times m}$ をその逆行列とする。このとき，

$$M' = \begin{pmatrix} \alpha & 0 \\ y & M \end{pmatrix}, \quad N' = \begin{pmatrix} \beta & 0 \\ z & N \end{pmatrix} \tag{7}$$

とする。$N'M' = I$ を第 1 列について解くと

$$\beta = \alpha^{-1}, \quad z = -\beta N y \tag{8}$$

を得る。これは，M' の逆行列 N' を M の逆行列 N から作る方法を示している。したがって，L の第 nn 成分からなる 1×1 行列の逆行列から出発して，1次元づつ次元を増やした行列の逆行列を計算し続けて，最終的に L の逆行列 X を計算することができる。このアルゴリズムを与えられた浮動小数点システム $\boldsymbol{F}$ 上で実行した結果得られた近似逆行列を $\tilde{X}$ としよう。このとき

$$|\tilde{X}L-I|\leq\gamma_n|\tilde{X}||L| \tag{9}$$

が成立することを証明しよう。この結果も上記の Higham のものであるが、証明はそこに記載されていなかったので、以下は著者による証明である。

n に関する帰納法を用いる。$n=1$ のときは明らかに成立する。そこで，$n-1$ まで成立したとしよう。

$$L=\begin{pmatrix}\alpha & 0\\ y & M\end{pmatrix},\quad X=\begin{pmatrix}\beta & 0\\ z & N\end{pmatrix} \tag{10}$$

とする。$XL-I$ を第1列について解けば

$$\beta=\alpha^{-1},\quad z=-\beta Ny \tag{11}$$

となる。これを浮動小数点数演算で実行すると

$$\begin{aligned}&\tilde{\beta}=\frac{1}{\alpha}(1+\delta),\quad |\delta|\leq u\\ &\tilde{z}=-\tilde{\beta}\tilde{N}y+\Delta(\tilde{\beta},\tilde{N},y),\\ &|\Delta(\tilde{\beta},\tilde{N},y)|\leq\gamma_{n-1}(1+\delta)|\beta||\tilde{N}||y|\end{aligned} \tag{12}$$

となる。これから，

$$\begin{aligned}&\tilde{\beta}\alpha=1+\delta\\ &\tilde{z}\alpha+\tilde{N}y=-\delta\tilde{N}y+\alpha\Delta(\tilde{\beta},\tilde{N},y)\end{aligned} \tag{13}$$

を得る。よって，$|\tilde{X}L-I|$ の第1列は，$u+(1+u)\gamma_{n-1}\leq\gamma_n$ より

$$\leq\begin{pmatrix}u\\ u|\tilde{N}||y|+\gamma_{n-1}(1+u)|\tilde{N}||y|\end{pmatrix}\leq\gamma_n|\tilde{X}||L|\text{ の第1列} \tag{14}$$

と評価される。第2列以下は $|\tilde{N}M-I|\leq\gamma_{n-1}|\tilde{N}||M|$ より満たされる。よって，n でも成立することがわかった。

3.2 連立一次方程式の高速精度保証

以上の準備の下で，LU 分解1回の手間で連立一次方程式のシャープな精度保証ができることを示そう。

$A\in\boldsymbol{R}^{n\times n}$ 行列が与えられ，IEEE 754 浮動小数点数システムの最近点への丸めモードで，LU 分解を使って

$$Ax=b \tag{15}$$

の近似解 $\tilde{x}$ が求められたとしよう。同時に A の LU 分解の近似 $\tilde{L}$ と $\tilde{U}$ も与えられているとしよう。この条件の下でできるだけ高速な精度保証を行うことを考える。ただし，保証できる精度は前節のアルゴリズムで計算するものとほぼ程度を要求するという意味でシャープな結果を得ることを条件とする (この条件を緩めるならば $O(n^2)$ の手間で精度保証できる)。

A の近似逆行列 R に対し，$\|RA-I\|_\infty < 1$ が成立すれば，$Ax=b$ の真の解 x^* が存在し，近似解 $\tilde{x}$ と真の解の間の誤差は

$$\|x^*-\tilde{x}\|_\infty \leq \frac{\|R(A\tilde{x}-b)\|_\infty}{1-\|RA-I\|_\infty} \tag{16}$$

と評価されることはよく知られている。この式をよく見ると，分子の $R(A\tilde{x}-b)$ の計算は高精度が要求されるのに対し，分母の $1-\|RA-I\|_\infty$ に現れる $\|RA-I\|_\infty$ の計算はたとえこの量が $\frac{1}{2}$ になっても，精度は2倍しか悪くならないことがわかる。しかも，$\|RA-I\|_\infty$ の計算に $2n^3$flops の計算量がかかっていた。そこで，$\|RA-I\|_\infty$ の計算に事前誤差評価式を用いて計算量を $O(n^2)$ に削減しようというのが基本的なアイディアである。また，近似逆行列として，$\tilde{L}$ の近似逆行列 X_L と $\tilde{U}$ の近似逆行列 X_U を計算して，$R=X_U\cdot X_L$ を用いる。ただし，2つの行列の積は計算しないことにする。この計算にはそれぞれ $\frac{n^3}{6}$flops づつ合計 $\frac{n^3}{3}$flops の計算量を要する。また，近似逆行列の計算はIEEE 754の最近点への丸めのモードで，事前誤差評価の節の方式によって行うものとする。

以下，X_L と X_U の計算以外の計算はすべて $O(n^2)$ で済ましてしまえることを見ていこう。

まず，

$$\|RA-I\|_\infty = \|X_UX_LA-I\|_\infty \tag{17}$$

を $O(n^2)$flops で計算するアルゴリズムを示そう。

$$\begin{aligned}
& \|X_UX_LA-I\|_\infty \\
\leq\ & \|X_UX_L(A-\tilde{L}\tilde{U})\|_\infty + \|X_UX_L\tilde{L}\tilde{U}-I\|_\infty \\
\leq\ & \gamma_n\| \,|X_U|\,|X_L|\,|\tilde{L}|\,|\tilde{U}|\,\|_\infty + \gamma_n\| \,|X_U|\,|\tilde{U}|\,\|_\infty \\
& +\gamma_n\| \,|X_U|\,|X_L|\,|\tilde{L}|\,|\tilde{U}|\,\|_\infty \\
=\ & 2\gamma_n\| \,|X_U|\,|X_L|\,|\tilde{L}|\,|\tilde{U}|\,\|_\infty + \gamma_n\| \,|X_U|\,|\tilde{U}|\,\|_\infty
\end{aligned} \tag{18}$$

ここで，$e=(1,1,\cdots,1)^t \in \boldsymbol{R}^n$ とすると

$$\| \,|X_U|\,|X_L|\,|\tilde{L}|\,|\tilde{U}|\,\|_\infty = \|(|X_U|(|X_L|(|\tilde{L}|(|\tilde{U}|e))))\|_\infty \tag{19}$$

および

$$\| \,|X_U|\,|\tilde{U}|\,\|_\infty = \|(|X_U|(|\tilde{U}|e))\|_\infty \tag{20}$$

が成り立つ。この両式は $\| \,|X_U|\,|X_L|\,|\tilde{L}|\,|\tilde{U}|\,\|_\infty$ および $\| \,|X_U|\,|\tilde{U}|\,\|_\infty$ が $O(n^2)$ で計算できることを示している。

また，

$$\| \,|X_U|\,|X_L|\,\|_\infty = \|(|X_U|(|X_L|e))\|_\infty \tag{21}$$

より $\|X_U\cdot X_L\|_\infty$ の評価も $O(n^2)$ でできることがわかる。

こうして，

$$\|A^{-1}(A\tilde{x}-b)\|_\infty \leq \frac{\|(X_U\cdot X_L)(A\tilde{x}-b)\|_\infty}{1-\|(X_U\cdot X_L)A-I\|_\infty} \tag{22}$$

の評価が X_L および X_L を $\tilde{L}$ および $\tilde{U}$ から計算する $\frac{n^3}{3}$flops の手間で実行できることがわかった。

例を示そう。上述の方法により，近似解とその精度保証をするC言語プログラムを作りその計算時間を測定した。実行時間の測定結果をまとめて表??に示す。また，参考のために，Intel の Pentium 用に最適化されたBLAS を用いたLAPACK を利用して作成したプログラムによる，同じパソコンでの計算時間も示す。

表 1: LU 分解を利用した近似解とその高速精度保証のための計算時間 (sec)

n	C	MATLAB	LAPACK
100	0	0.11	0.05
200	1	0.55	0.16
300	2	1.87	0.44
400	6	4	0.72
500	13	8	1.42
600	24	16	2.14
700	38	25	3.29
800	57	37	4.50
900	83	53	6.26
1000	112	72	8.29

4 Scilab による精度保証付き数値計算環境の構築

数値計算専用の高い品質で容易に計算できる数値計算インタプリタ言語として、欧米では MATLAB がよく用いられている。MATLAB は商用でソースコードは公開されていない。精度保証付き数値計算を行うためには、本研究者はオープンソースの言語を用いることがよいのではないかと考えている。そのような理由により、本研究者は MATLAB と同様に使いやすく、安定で、高速な数値計算を行うという目標をもった、オープンソースの数値計算用インタプリタ言語である Scilab(サイラボ) を用いて精度保証付き数値計算ライブラリを作成した。本章ではその概要について紹介する。

Scilab のホームページは http://www.inria.fr/ で、ここからソースをダウンロードすることができる。Scilab はフリーであるが、使用することを Inria に e-mail することが義務づけられている。Scilab をインストールしたとしよう。このとき、

```
$ scilab
```

と kterm から入力すると、Scilab が起動して X のグラフィックス画面が開き Scilab が起動する。このグラフィックス画面には

```
-->
```

とプロンプトが表示される。ここから Scilab のコマンドを入力して数値計算を開始する。グラフィック画面のヘルプボタンを押すことによって、Scilab のヘルプ画面を呼び出すことができる。Scilab を終了するためには

```
--> exit
```

とすればよい。

4.1 C 言語とのリンク

Scilab は Fortran や C で書かれた関数を呼び出して、実行する機能がある。これをリンクという。精度保証付き数値計算を行うためには、IEEE754 規格を利用して、浮動小数点数演算の丸めの方向を制御する必要がある。そこで、C 言語による浮動小数点数演算の丸めの制御のための関数を利用して、リンクにより Scilab 上で浮動小数点数の丸めの制御を行う方法を示そう。

Linux 上で Scilab を起動していると仮定しよう (もっと具体的には Vine Linux1.1 での動作を確認している)。次の内容の関数を up.c というファイルに保存したとしよう。これは C での上への丸めのモードの切り替えの命令である。

```
#include <fpu_control.h>

void up(void) { __setfpucw(_FPU_RC_UP|_FPU_DEFAULT); }
```

ここで、C コンパイラでオブジェクトファイル up.o を次のようにして作る。

```
$ cc -c up.c
```

同様にして、次の内容のファイルを down.c とする。これは C での下への丸めのモードの切り替えの命令である。

```
#include <fpu_control.h>

void up(void) { __setfpucw(_FPU_RC_DOWN|_FPU_DEFAULT); }
```

ここで、C コンパイラでオブジェクトファイル down.o を次のようにして作る。

```
$ cc -c up.c
```

さらに、次の内容のファイルを near.c とする。

```
#include <fpu_control.h>

void up(void)
{ __setfpucw(_FPU_RC_NEAREST|_FPU_DEFAULT); }
```

ここで、C コンパイラでオブジェクトファイル near.o を次のようにして作る。これは最近点への丸めの関数である。

```
$ cc -c up.c
```

以上の準備の下に、Scilab を起動する。

```
$ scilab
```

そして、次のようにして、C 言語の関数のオブジェクトファイルを Scilab 内で利用できるようにリンクする。

```
-->link('up.o','up','C');

-->link('down.o','down','C');

-->call('up');

-->c=1/10;

-->printf("%25.20lg\n",c)

    0.10000000000000000056

-->call('down');

-->c=1/10;

-->printf("%25.20lg\n",c)

  0.09999999999999991673
```

丸めが実際に行われている様子がわかるであろう。

4.2 非線形方程式の数値解の精度保証

前節までの準備をもとに、非線形方程式の数値計算で求めた数値解 (これは誤差を評価していない近似的なもの) の近くに真の解が存在するか否かを検証する精度保証を行う方法を示そう。

区間演算を始めとして、自動微分型など様々なデータ型が精度保証付き数値計算で必要となる。区間演算や自動微分型などをプログラムとして実装するためには、オブジェクト指向言語の演算子多重定義の機能を利用すると便利である。

ここでは、演算子多重定義の機能を用いた区間演算や自動微分型の実装例として、Scilab による実装例を示そう。Scilab はオブジェクト (C++流にいえばクラス) として tlist がある。tlist とはタイプ付リストのことで、tlist(タイプ,x1,x2,$\cdots$,xn) の形をしている。タイプはオブジェクト名であり、これでオブジェクトを識別する。x1,x2,$\cdots$,xn は Scilab の既存のオブジェクトである。区間オブジェクトとして tlist('intval',s,t) を定義する。これで区間 $[s,t]$ を表すことにする。ただし、s,t は同じタイプの実数, ベクトルまたは行列とする。Scilab において

```
-->x=tlist('intval',s,t);
```

とすると、x は区間型の tlist となる。x の要素は x(2) で s を x(3) で t を参照することができる。tlist では四則演算などを演算子多重定義ができ、演算子多重定義するには関数名が % で始まるものが割り当てられている。例えば、%intval_a_intval という関数名は関数と区間の和を演算子多重定義する関数を表す。その関数の定義例としてはつぎが考えられる。区間と区間の和の演算子多重定義の命令は

```
function x=%intval_a_intval(a,b)

   call('down'); x1=a(2)+b(2);

   call('up');   x2=a(3)+b(3);

   x=tlist('intval',x1,x2);

endfunction
```

で定義できる。同様に、減算、乗算、除算なども同様に定義できる。このような演算子多重定義を書いたファイルを intval.sci とし、Scilab のカレントディレクトリにおいておくと、

```
-->getf('intval.sci');
```

とすることで、区間型の tlist を呼び出し、演算子多重定義された区間型の四則演算を利用できる。

さて、Scilab では行列型が予め組み込まれており、成分が倍精度浮動小数点数の行列演算を高速に実行できる。しかし、成分が任意のオブジェクトとなる行列型は定義されていないので、これを tlist として実装しておくと便利である。そこで、著者は任意オブジェクトを要素にもてる tlist として

tlist(['mat','dim'],[m,n],x1,x2,$\cdots$,xn) の形の mat 型を定義している。x=mat(m,n) とすると、要素がすべて 0 に初期化された $m \times n$ 行列が確保される。evstr(s) は文字列 s を実行する命令で、インタプリタに特有の命令である。

```
function x=mat(m,n)

   s='tlist(["''mat''","''dim''"],[m,n]'; d=m*n;

   for i=1:d, s=s+',0', end; s=s+')';

   x=evstr(s);
```

```
endfunction
```

mat 型の加算、減算、乗算も演算子多重定義できる。以下は加算の例である。

```
function x=%mat_a_mat(a,b)
   c=a('dim'); d=c(1)*c(2); x=a;
   for i=3:2+d, x(i)=a(i)+b(i), end;
endfunction
```

このような関数の入ったファイルを mat.sci として、Scilab を立ち上げるディレクトリに入れておく。さて、このような準備の下で自動微分型を定義しよう。自動微分型を

```
tlist(['dif','dim','val','grad'],m,val,grad)
```

のタイプの tlist とする。val はスカラーで grad は $1 \times m$ の mat 型の tlist とする。自動微分型も演算子多重定義できる。加法の定義を以下に与えよう。

```
function x=%dif_a_dif(a,b)
   x=a;
   x('val')=a('val')+b('val');
   x('grad')=a('grad')+b('grad');
endfunction
```

こうして、自動微分型の定義がファイル autodif.sci に書き込まれ、Scilab を立ち上げるディレクトリに入っているとしよう。以上で、区間解析のための基本的な道具は揃った。次の章では、これらの準備の下に有限次元非線形方程式の精度保証付き数値計算法を示そう。

4.3 ニュートン法とクラフチック法

有限次元非線形方程式の近似解を求めるニュートン法のプログラムが前章までの道具により簡単に作成できることを示そう。また、その近似解の近くに真の解が存在することを検証するためのクラフチック法のプログラムも示そう。

4.3.1 ニュートン法のプログラム

$f : \boldsymbol{R}^n \to \boldsymbol{R}^n$ として非線形方程式 $f(x) = 0$ を考える。関数 f を定義する Scilab の関数を例えば

```
function [y]=func(x)
   y=x;  y(3)=x(3)*x(3)-x(4)*x(4)-3*x(3)+2;
   y(4)=2*x(3)*x(4)-3*x(4);
endfunction
```

のように定義する。このとき、近似解 x に対するニュートン法の一反復を与えるプログラムは次のように作ることができる。

```
function z=newton(x, name)
   ss=x('dim');  s1=ss(1);
   y=init_dif(x);  xx=val(y);
   s=name+'(y)';  zz=evstr(s);
   j=jacobi(zz);  v=val(zz);
   d=j\v; nx=xx-d; z=x;
   for i=1:s1,  z(i+2)=nx(i);  end;
endfunction
```

ただし、ファイル autodif.sci の中には次のような関数がある。

```
function x=init_dif(a)
   n=a('dim');  m=n(1);
   s='tlist(["''dif''","''dim''","''val''","''grad''"],m,0,0)';
   ss=evstr(s);  x=mat(m,1);
   for i=1:m,
      x(i+2)=ss;  x(i+2)('val')=a(i+2);
      x(i+2)('grad')=mat(1,m);  ii=i+2;
      x(i+2)('grad')(ii)=1;
   end;
endfunction
function x=jacobi(a)
   s=a('dim');  ss=s(1);  x=eye(ss,ss);
   for i=1:ss,
      for j=1:ss,  x(i,j)=a(i+2)(4)(j+2); end;
   end;
endfunction
function x=val(a)
   s=a('dim');  ss=s(1);  x=eye(ss,1);
   for i=1:ss;  x(i)=a(i+2)(3);  end;
endfunction
```

4.4 クラフチック法のプログラム

さて、ニュートン法のプログラムで求めた近似解の近くに真の解が存在するか否かを判定するクラフチック法のプログラムを示そう。少し長いが、精度保証付き数値計算のプログラムの典型となるので、全文をあげておこう。

```
function [T,H]=kraw(x, name)

   ss=x('dim');  dim=ss(1);

   e1=0.5*10^{-15};  h=i(-e1,e1);  ix=x;

   for i=1:dim;  ix(i+2) = x(i+2) + h;  end;

   s = name + '(ix)';  z = evstr(s);  v = x;

   for i=1:dim,  v(i+2)=z(i+2);  end;

   y=init_dif(x);  s=name + '(y)';  zz=evstr(s);

   j=jacobi(zz);  vv=val(zz);  d=j\vv;
rad=2*norm(d,'inf');

   R=inv(j); d1=max(10^{-13},abs((rad)));

   hh=i(-d1,d1);  ix=x;

   for i=1:dim,  ix(i+2) = x(i+2) + hh;  end;

   y=init_dif(ix);  F = evstr(s);

   Fc = eye(dim,dim);  Fw = eye(dim,dim);

   fc = ones(dim,1);  fw = ones(dim,1);

   call('up');

   for i=1:dim,

      fc(i) = (v(i+2)(2)+v(i+2)(3))/2;

      fw(i) = fc(i)-v(i+2)(2);

   end;

   for i=1:dim,

      for j=1:dim,

         Fc(i,j)=(F(i+2)(4)(j+2)(2)+F(i+2)(4)(j+2)(3))/2;

         Fw(i,j)=Fc(i,j)-F(i+2)(4)(j+2)(2);

      end;
```

```
    end;

    aR=abs(R);  ru = (-R)*fc + aR * fw;

    Mu = eye(dim,dim) + (-R)*Fc + aR * Fw;

    call('down');

    rd = (-R)*fc + (-aR)*fw;

    Md = eye(dim,dim) + (-R)*Fc + (-aR) * Fw;

    Mm = max(abs(Md),abs(Mu));

    call('up');  q = d1*Mm*ones(dim,1);  Hu = ru+q;

    call('down');

    Hd = rd+(-q); H=max(norm(Hu,'inf'),norm(Hd,'inf'));
T=d1;

  endfunction
```

必要な関数を Scilab に読み込み、関数 newton を利用して近似解を求める。十分よい近似解が得られたと思われたとき、関数 kraw に近似解と非線形方程式を記述する関数名を代入すると、近似解の精度保証がされる。$H < T$ であれば、近似解から最大値ノルムで T の範囲内に真の解が唯一つ存在することが示されたことになる。

以上、後半では著者による Scilab 上の精度保証ライブラリの概要について紹介した。精度保証付き数値計算のプログラミングの一例として参考にして頂ければ幸である。

参考文献

[1] Shin'ichi Oishi and Siegfried M. Rump: "Fast verification of solutions of matrix equations", submitted to *Numer. Math.*.

[2] 大石進一:精度保証付き数値計算、コロナ社 (2000)

[3] 大石進一:数値計算、裳華房 (1999)

[4] Shin'ichi OISHI: "Fast verification of eigenvalues and singular values of real symmetrix matrices", submitted to *Linear Algebra and its Applications*