いつ反復計算をやめるべきか？

～収束判定基準の設定方法～

渡部 善隆 *

この記事は、数値計算において頻繁に用いられる反復計算の収束判定基準と収束判定用定数の設定方法について簡単にまとめたものです。本論は 4 章です。2 章と 3 章は 4 章で使用する用語の説明にあてられています。ノルムや誤差について十分な知識をお持ちの方は 2 章や 3 章を飛ばしてお読みいただいて結構です。

反復計算が確実に真の解に向かって行くという保証 (収束証明) と、反復を途中で打ち切った時の誤差限界を与えることは数値解析学の重要な使命です。しかし、すべての反復計算においてこの二つ (収束証明と誤差限界) が理論的に保証されるわけではありません。したがって、実際に反復計算を用いる場合に極めて重要な設定項目は、どのような収束判定基準を用いていつ反復を終了するかを決定することです。そのとき、計算の限界の検出という意味で根拠を持ついくつかの判定基準があることを知っているのは決して無駄ではないと思います。

1 反復計算とは

1.1 問題設定

実数体 $\boldsymbol{R}$ 上の n 次元ベクトル空間を $\boldsymbol{R}^n$ で表し，以下では n 次元 Euclid 空間と同一視します． $\boldsymbol{R}^n$ から $\boldsymbol{R}^n$ 自身への写像 $\boldsymbol{f} = (f_1, f_2, \ldots, f_n)^T$ に対して，方程式:

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0} \tag{1}$$

をみたす実ベクトル $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^T \in \boldsymbol{R}^n$ を計算機によって数値的に求める問題を考えます[1]．“T” は転置記号です． $f_i (1 \leq i \leq n)$ それぞれは $\boldsymbol{R}^n$ から $\boldsymbol{R}$ への写像です．したがって (1) は

$$f_i(\boldsymbol{x}) = 0 \qquad (1 \leq i \leq n)$$

と書くこともできます．方程式の数と未知数の個数は同じ n 個とします[2]．また， n は次元の意味でも用います．たとえば，方程式 $x = \cos x$ は $f(x) := x - \cos x$ と変形することにより (1) として見ることができます．連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ も $\boldsymbol{f}(\boldsymbol{x}) := \boldsymbol{b} - A\boldsymbol{x}$ とすれば (1) の形になります．さらに，非線形微分方程式など多くの関数方程式は，差分法，有限要素法，境界要素法などの離散化によって (1) に帰着されます．

1.2 反復計算

$\boldsymbol{f}$ は通常非線形[3]です．したがって $\boldsymbol{f}(\boldsymbol{x})$ は，図 1 のようにベクトル $\boldsymbol{x}$ の要素が複雑に絡まりあった形をしています．

* 九州大学大型計算機センター・研究開発部　E-mail: watanabe@cc.kyushu-u.ac.jp

[1] なお，この記事の内容は，ほぼそのまま複素数にも拡張することができます．

[2] 方程式の数と未知数の個数が異なると，途端に話がややこしくなります．ひとことでややこしくなる理由を説明すると，Jacobi 行列 (ドイツの数学者 Carl Gustav Jacob Jacobi さん (1804–1851) に由来します．ヤコビアン, Jacobian 行列ともいいます．) が非正則になるからです．

[3] 任意の $\boldsymbol{x}, \boldsymbol{y} \in \boldsymbol{R}^n$, $\alpha, \beta \in \boldsymbol{R}$ に対して $\boldsymbol{f}(\alpha\boldsymbol{x} + \beta\boldsymbol{y}) = \alpha\boldsymbol{f}(\boldsymbol{x}) + \beta\boldsymbol{f}(\boldsymbol{y})$ が成立するとき， $\boldsymbol{f}$ は線形写像であるといいます．非線形写像とは，このような性質を持たないもの，と定義されます．

$$
\begin{cases}
f_1(\boldsymbol{x}) = \dfrac{9\,\mathcal{P}\,x_1}{4} + \dfrac{9\,x_2\,x_3}{8\sqrt{2}} + \dfrac{\mathcal{P}\,\mathcal{R}\,x_7}{\sqrt{2}} \\
f_2(\boldsymbol{x}) = \dfrac{81\,\mathcal{P}\,x_2}{4} + \dfrac{9\,x_1\,x_3}{8\sqrt{2}} + \dfrac{\mathcal{P}\,\mathcal{R}\,x_8}{\sqrt{2}} \\
f_3(\boldsymbol{x}) = \dfrac{-9\,x_1\,x_2}{4\sqrt{2}} + 9\,\mathcal{P}\,x_3 + \sqrt{2}\,\mathcal{P}\,\mathcal{R}\,x_9 \\
f_4(\boldsymbol{x}) = 36\,\mathcal{P}\,x_4 + \sqrt{2}\,\mathcal{P}\,\mathcal{R}\,x_{10} \\
f_5(\boldsymbol{x}) = -2\,x_5 + \dfrac{x_2\,x_7}{2\sqrt{2}} + \dfrac{x_1\,x_8}{2\sqrt{2}} - \dfrac{x_4\,x_9}{\sqrt{2}} + \sqrt{2}\,x_4\,x_9 - \dfrac{x_3\,x_{10}}{\sqrt{2}} + \sqrt{2}\,x_3\,x_{10} \\
f_6(\boldsymbol{x}) = -8\,x_6 - \dfrac{x_1\,x_7}{\sqrt{2}} - \sqrt{2}\,x_3\,x_9 \\
f_7(\boldsymbol{x}) = -\dfrac{x_1}{\sqrt{2}} - \dfrac{x_2\,x_5}{2\sqrt{2}} + \dfrac{x_1\,x_6}{\sqrt{2}} - \dfrac{3\,x_7}{2} + \dfrac{3\,x_3\,x_8}{4\sqrt{2}} + \dfrac{3\,x_2\,x_9}{4\sqrt{2}} \\
f_8(\boldsymbol{x}) = -\dfrac{x_2}{\sqrt{2}} - \dfrac{x_1\,x_5}{2\sqrt{2}} - \dfrac{3\,x_3\,x_7}{4\sqrt{2}} - \dfrac{9\,x_8}{2} - \dfrac{3\,x_1\,x_9}{4\sqrt{2}} \\
f_9(\boldsymbol{x}) = -\sqrt{2}\,x_3 - \dfrac{x_4\,x_5}{\sqrt{2}} + \sqrt{2}\,x_3\,x_6 - \dfrac{3\,x_2\,x_7}{4\sqrt{2}} + \dfrac{3\,x_1\,x_8}{4\sqrt{2}} - 3\,x_9 \\
f_{10}(\boldsymbol{x}) = -\sqrt{2}\,x_4 - \dfrac{x_3\,x_5}{\sqrt{2}} - 6\,x_{10}
\end{cases}
\quad \Longleftarrow
$$

左の次元 $n = 10$ の非線形方程式は，定常熱対流問題から導かれる非線形偏微分方程式の解をスペクトル法によって離散近似して得られたものです．方程式を満たす $\boldsymbol{x}$ を求めることで，流れ関数と温度場の近似解が出てきます．$\mathcal{P}$, $\mathcal{R}$ は既知の定数です．

図 1: $\boldsymbol{f}(\boldsymbol{x})$ の具体例[4]

また $\boldsymbol{f}$ が線形であったとしても，n が大きくなると手計算や数式処理ソフトを用いて “数学的に” 解くことが難しくなってきます．そこで浮動小数点演算[5]による反復計算が登場します．計算の大雑把な手順は次のようになります:

【反復計算の手順】

- 最初に適当な値を選んで第 1 近似値とする; $\quad \boldsymbol{x}^{(0)}$
- それを材料にして「もっとよい近似値」を (浮動小数点演算によって) 求める; $\quad \boldsymbol{x}^{(0)} \to \boldsymbol{x}^{(1)}$
- ある停止条件を満たすまで改良を繰り返す; $\quad \boldsymbol{x}^{(0)} \to \boldsymbol{x}^{(1)} \to \cdots \to \boldsymbol{x}^{(k)} \to \cdots$

この手順を総称して**逐次近似法** (*successive approximation method*) と呼びます．実際の計算では，問題に応じて第 1 近似値の設定方法 (こちらも非常に大切です)，次の近似値の決定方法，反復終了の条件をそれぞれ設定する必要があります．

1.3 Newton-Raphson 法

あらゆる形の $\boldsymbol{f}$ に対応した逐次近似法のアルゴリズムは見つかっていません．もっとも有名なものは **Newton-Raphson 法** (*Newton-Raphson's method*)[6]の反復計算:

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \boldsymbol{f}'(\boldsymbol{x}^{(k)})^{-1}\boldsymbol{f}(\boldsymbol{x}^{(k)}) \qquad k = 0, 1, 2, \ldots$$

でしょう[7]．たとえば 1 次元の問題 $f(x) := x - \cos x = 0$ を Newton-Raphson 法の式で書くと

$$x^{(k+1)} = x^{(k)} - \frac{x^{(k)} - \cos x^{(k)}}{1 + \sin x^{(k)}} \qquad k = 0, 1, 2, \ldots$$

となります．一般の n では $\boldsymbol{f}'(\boldsymbol{x}^{(k)})$ は $\boldsymbol{f}$ の $\boldsymbol{x}^{(k)}$ での微分を表す $n \times n$ Jacobi 行列です[8]．

[4] 詳細は `http://www.cc.kyushu-u.ac.jp/RD/watanabe/RESERCH/MANUSCRIPT/OHP/AppMathSym-1998/intro.html` を参照してください．

[5] 浮動小数点に代表される計算機内の数の表現方法については，たとえば [4], [9] を参照してください．

[6] **Newton 法** (*Newton's method*) とも呼びます．Joseph Raphson さん (1648–1715) はイギリスの数学者です．

[7] その他の手法として，regula-falsi 法，二分法 (bisection method)，一松法，Traub 法，Muller 法などがあります．詳しくは [14], [3] を参照してください．

[8] “$\boldsymbol{f}'(\boldsymbol{x}^{(k)})^{-1}\boldsymbol{f}(\boldsymbol{x}^{(k)})$” は，素直に読むと Jacobi 行列 $\boldsymbol{f}'(\boldsymbol{x}^{(k)})$ の逆行列を計算してベクトル $\boldsymbol{f}(\boldsymbol{x}^{(k)})$ との積を求めることになります．しかし実際の計算で逆行列を求めることは少なく，連立 1 次方程式 $\boldsymbol{f}'(\boldsymbol{x}^{(k)})\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{x}^{(k)})$ を数値的に解きます．

Newton-Raphson 法は理論的適用範囲が広く，実用的にも強力であることから広く使われている逐次近似法の一種です．ただし，万能ではありません．第 1 近似値の決定方法，$\boldsymbol{f}$ が微分不可能またはわからない場合の対処方法，行列 $\boldsymbol{f}'(\boldsymbol{x}^{(k)})$ が悪条件[9]になり連立 1 次方程式が数値的に解けなくなった場合どうするのか，n が大きい場合の連立 1 次方程式を解く手間など，解決すべき案件が次々と登場します．そして，最後は必ず「いつ反復計算をやめるべきか」という大切な問題が待ち受けています．繰り返しを止めるのが早過ぎると，十分な精度が得られません．逆に止めるのが遅過ぎると，計算時間が余分にかかることになりますし，意味のない計算を繰り返していることすらあるのです．なお，Newton-Raphson 法についての理論的な解説は [3], [10], [12], [16] をご覧ください．

1.4　反復計算と反復法

数値解析の分野では，連立 1 次方程式に対する逐次近似法を「反復法」と呼ぶことが多くなりました (たとえば [1], [2], [7] のタイトルをご覧ください)[10]．したがって，この記事では「反復法」という用語を使うのは避け，「反復計算」という言葉を使います

計算機は有限桁の計算しかできません．この記事ではあたかも無限桁の実数値が演算によって得られるような書き方をすることもあります．しかし，実際の数値計算では，有限の浮動小数点体系上の演算 (詳しくは [3], [5], [9] を参照してください) となることに注意してください．

2　ノルム

2.1　ノルムの定義

反復計算を途中でやめるには，それぞれの反復で得られるベクトルの「大きさ」あるいは「長さ」を調べることが必要です．$\boldsymbol{R}$ では絶対値で「大きさ」を評価します．$\boldsymbol{R}^n$ の場合は n 個の要素から成るベクトルの「大きさ」を総合的に測る指標が必要です．しかしどれでも良いわけでなく，直観と合致するいくつかの性質を有することが少なくとも要請されます．この要請をまとめたのが次の**ノルム** $(norm)$[11] の定義です:

【ノルムの定義】

$\boldsymbol{R}^n$ のノルム $\|\ \|$ とは，次の 3 つの性質を満たす実数値関数である:

1. 任意の $\boldsymbol{x} \in \boldsymbol{R}^n$ に対して，$\|\boldsymbol{x}\| \geq 0$ かつ，$\|\boldsymbol{x}\| = 0 \Leftrightarrow \boldsymbol{x} = \boldsymbol{0}$.
2. 任意の $\boldsymbol{x} \in \boldsymbol{R}^n$, $\alpha \in \boldsymbol{R}$ に対して，$\|\alpha\boldsymbol{x}\| = |\alpha|\|\boldsymbol{x}\|$.
3. 任意の $\boldsymbol{x}, \boldsymbol{y} \in \boldsymbol{R}^n$ に対して，$\|\boldsymbol{x}+\boldsymbol{y}\| \leq \|\boldsymbol{x}\| + \|\boldsymbol{y}\|$ $\left(\begin{array}{l}\text{三角不等式または}\\ \text{Minkowski の不等式}\end{array}\right)$.

ここで，“$|\alpha|$” は実数 α の絶対値です．$n = 1$ としたときの絶対値はノルムの定義を満たします．“$\Leftrightarrow$” は必要十分条件を意味します．ノルムが小さいこととすべての成分が小さいことは等価になります (証明はたとえば [11] をご覧ください).

[9] $A\boldsymbol{x} = \boldsymbol{b}$ の解 $\boldsymbol{x}$ の変化が A や $\boldsymbol{b}$ の変化の何倍に拡大されるのかを表す数を条件数と呼びます (例えば [11] を参照してください)．条件数が大きくなる問題を悪条件と呼びます．

[10] 連立 1 次方程式の数値解法は，大きく「直接法」「反復法」「共役勾配法」(共役勾配法を反復法に分類することもあります) に分類することができます．[15] の簡単な紹介記事を参照してください．

[11] 英語で「標準」「平均」という意味です．「ノルマ」(ラテン語の $norma$ に由来) の意味もあります．

2.2 よく使われるノルム

ノルムの定義を満たす関数は無限個存在します[12]．しかし，ここでは数値計算でよく使われるノルムを表 1 に紹介します．なお，各ノルムを識別するため “$\|\ \|$” の右下に記号をつけます．ベクトル $\boldsymbol{x}$ の要素は $x_i\,(1 \le i \le n)$ で表しています．

表 1: 数値計算でよく使われる $\boldsymbol{R}^n$ のノルム

ノルムの定義	名称
$\|\boldsymbol{x}\|_1 = \sum_{i=1}^{n} \|x_i\|$	1 ノルム， 1-ノルム，絶対ノルム，絶対和ノルム， l_1 ノルム
$\|\boldsymbol{x}\|_2 = \left(\sum_{i=1}^{n} \|x_i\|^2\right)^{\frac{1}{2}}$	2 ノルム， 2-ノルム， 2 乗ノルム， Euclid ノルム, l_2 ノルム
$\|\boldsymbol{x}\|_\infty = \max_{1\le i\le n} \|x_i\|$	最大ノルム，一様ノルム， ∞ ノルム， ∞-ノルム，無限ノルム， l_∞ ノルム

各ノルムの値が定数 $a(>0)$ となる $\boldsymbol{x} \in \boldsymbol{R}^2$ の全体をプロットすると， $\|\boldsymbol{x}\|_1 = a$ は “◇”, $\|\boldsymbol{x}\|_2 = a$ は “○”, $\|\boldsymbol{x}\|_\infty = a$ は “□” の形をしています (詳しくは [11], [13] を参照してください).

一般に有限次元ノルム空間の任意の 2 つのノルムは同値です[13]．したがって理論上はどんなノルムを採用しても構いません．しかし数値計算を行なう場合は，次節で説明するように次元 n に注意する必要があります．

2.3 次元に対する依存性

表 1 の 3 つのノルムの中では 2 ノルムが最も有名だと思います．しかし，実際の数値計算では最大ノルムがよく使われます．理由は，四則演算が必要ないことと，次元 n に依存しないからです．定義から分かる通り， 1 ノルムと 2 ノルムは次元 n に依存します．例として

$$\boldsymbol{x} = (\sin(1), \sin(2), \ldots, \sin(n))^T$$

に対する各ノルムの値を n を変えて計算した結果を表 2 にあげます．計算は Fortran の倍精度 (IEEE 形式; 64 ビット) で行ない，表示数字以下は切捨てています．

表 2: $\boldsymbol{x} = (\sin(1), \sin(2), \ldots, \sin(n))^T$ に対するノルムの値

n	$\|\boldsymbol{x}\|_1$	$\|\boldsymbol{x}\|_2$	$\|\boldsymbol{x}\|_\infty$
1	0.84	0.84	0.841
10	6.48	2.23	0.989
100	63	7	0.999
1000	636	22	0.999
10000	6366	70	0.999
100000	63662	223	0.999
500000	318309	500	0.999

[12] 任意の固定した正則行列 A に対して $\|\boldsymbol{x}\|_A := \|A\boldsymbol{x}\|_2$ がノルムの定義を満たすからです.

[13] 2 つのノルム $\|\ \|_a$, $\|\ \|_b$ が同値であるとは，ある正数 c_1, c_2 が取れて，任意の $\boldsymbol{x}$ に対して $c_1\|\boldsymbol{x}\|_a \le \|\boldsymbol{x}\|_b \le c_2\|\boldsymbol{x}\|_a$ が成り立つことをいいます．この不等式から $1/c_2\|\boldsymbol{x}\|_b \le \|\boldsymbol{x}\|_a \le 1/c_1\|\boldsymbol{x}\|_b$ も成り立ちます．有限次元ノルム空間のノルムの同値性の証明は関数解析の教科書を参照してください.

最大ノルムの値が変わらないのに対し，1 ノルムと 2 ノルムは n に応じて値が大きくなっていきます．これら 3 つのノルムの間には以下の不等式が成り立ちます:

$$\begin{aligned}
\|\boldsymbol{x}\|_2 &\leq \|\boldsymbol{x}\|_1 \leq \sqrt{n}\|\boldsymbol{x}\|_2, \\
\|\boldsymbol{x}\|_\infty &\leq \|\boldsymbol{x}\|_2 \leq \sqrt{n}\|\boldsymbol{x}\|_\infty, \\
\|\boldsymbol{x}\|_\infty &\leq \|\boldsymbol{x}\|_1 \leq n\|\boldsymbol{x}\|_\infty.
\end{aligned}$$

大規模な数値計算を行なうときには，ノルムによっては次元に依存することを知る必要があります．また，2 ノルムの計算には自乗の計算が必要なため，他のノルムに比べて計算途中でオーバーフローやアンダーフローが起きやすいことにも注意が必要です (たとえば [1] をご覧ください)[14].

2.4 ノルムの計算

1 ノルム，2 ノルム，最大ノルムを求める Fortran プログラムを紹介します．最大ノルムを用いた具体的な応用例は 4.5節をご覧ください．それぞれベクトル $\boldsymbol{x}, \boldsymbol{y} \in \boldsymbol{R}^n$ に対し $\boldsymbol{x} - \boldsymbol{y}$ のノルムを倍精度で計算するものです[15]．大きさ n で宣言した 1 次元配列 x, y が $\boldsymbol{x}, \boldsymbol{y}$ に対応しています．添字 i は整数型，ノルムの値は変数 s に返ります[16].

● 1 ノルム

```
s=0.0D0                          ! 最初は 0 に設定
do i=1,n                         ! 1 から n まで動かす
  s=s+abs(x(i)-y(i))             ! 各要素の差の絶対値を足す
end do                           ! do 文の終り
```

Fortran 90 の配列演算機能と SUM 関数を使うと，さらに簡単に記述できます．

```
s=sum(abs(x-y))                  ! s に 1 ノルムの値が返る
```

ただし，SUM 関数 (および後に出てくる MAXVAL 関数) を使うとき，配列 x, y を n より大きい値で宣言している場合には，すべての要素を 0 に初期化してください．

● 2 ノルム

```
s=0.0D0                          ! 最初は 0 に設定
do i=1,n                         ! 1 から n まで動かす
  s=s+(x(i)-y(i))**2             ! 各要素の差の自乗を足す
end do                           ! do 文の終り
s=sqrt(s)                        ! 平方根を計算
```

Fortran 90 の配列演算機能と SUM 関数を使うと，さらに簡単に記述できます:

```
s=sqrt(sum((x-y)**2))            ! s に 2 ノルムの値が返る
```

[14] だからといって「2 ノルムは使わない方がいい」とは一概に言えません．2 ノルムは我々が普段使っている (Euclid) 距離そのものであり，理論的にも大切なものだからです．

[15] プログラムの中で倍精度だとわかる箇所は，s の初期設定 “s=0.0D0” だけです．“D0” を付けたのは用心のためで，おそらくすべての処理系で “s=0” または “s=0.0” でも変な「ゴミ」はたまらないはずです．また，組み込み関数 abs, sqrt, max は総称名です．総称名の関数は引数に対応した値を返却します．したがってそれぞれ dabs, dsqrt, dmax1 と同じ値を得ます．

[16] 個人的な趣味によってプログラムは小文字で記述しています．Fortran は大文字 / 小文字を区別しません．

●最大ノルム

```
s=0.0D0                          ! 最初は 0 に設定
do i=1,n                         ! 1 から n まで調べる
  s=max(abs(x(i)-y(i)),s)        ! |x(i)-y(i)| と s の大きい方を s に再設定
end do                           ! do 文の終り
```

Fortran 90 の配列演算機能と MAXVAL 関数を使うと，さらに簡単に記述できます:

```
s=maxval(abs(x-y))               ! s に最大ノルムの値が返る
```

●計算速度

表 3 は VPP700/56 のスカラーモード (オプション -Wv,-sc 指定) で計算した結果です．$1 \le i \le n$ に対して x(i)=sin(i), y(i)=cos(i) で配列を作成しています．$n = 100000$，ノルムの計算には Fortran 90 の組み込み関数を使用しました[17]．最適化オプションはシステムの省略値を採用しています．時間はノルムの計算に要した CPU 時間です．単位はミリ秒です．

表 3: ノルムの計算時間 (VPP700/56 スカラーモード)

ノルム	単精度	倍精度	4 倍精度
1 ノルム	8	26	898
2 ノルム	5	7	2415
最大ノルム	23	33	636

プログラムを見ると，最大ノルムが一番速そうな気がします．その意味では 4 倍精度の結果は納得できる値です．しかし，単精度と倍精度では 2 ノルムが優秀でした．これは，VPP700/56 の Fortran コンパイラ (Fujitsu Fortran90/VP V10L10 L98061) が最適化を頑張っているからです[18]．また，ベクトル化を施すと単精度・倍精度はすべての値が 1 ミリ秒以下で処理されました[19]．

他の幾つかの計算機システムでも試した結論として，単精度と倍精度で計算する場合，ノルムを計算する過程で必要になる計算時間に比べると，ノルムの計算時間にあまり神経質にならなくてもいいのではないかと思います．ただし，4 倍精度以上[20]を用いる場合は計算機によって性能に差が出る可能性があります．

2.5 行列ノルム

ベクトルのノルムと同様，行列のノルムも定義することができます．たとえば $n \times n$ 行列 $A = (a_{ij})$ の最大ノルム $\|A\|_\infty$ は最大の行絶対値和

$$\|A\|_\infty = \max_{1 \le i \le n} \sum_{j=1}^{n} |a_{ij}|$$

で定義されます．行列ノルムは線形計算の誤差解析や感度解析などで重要な役割を担います．詳しくは [11] などを参照してください．

[17] もうひとつのプログラムとの極端な差はありませんでした．

[18] 最適化レベルを下げると差がなくなります．

[19] 4 倍精度はベクトル化の対象外なので値はほとんど変わりません．

[20] 現在の JIS Fortran の規格に「4 倍精度」はありません．実は単精度も倍精度もなくて，「処理系は，基本種別より高い精度をもつ表現方法を少なくとも一つは用意しなければならない．それを DOUBLE PRECISION と指定することもできる．」と書いてあるだけです．4 倍精度はメーカーが拡張仕様として提供している機能であり，すべての Fortran システムで利用できるとは限りません．

3　誤差の基礎知識

この章では，前章で定義したノルムを用いて「誤差」の基本的な用語を説明します．絶対誤差，相対誤差は次章の反復計算の収束判定基準には直接関係しません．しかし，理論的な数値解析ではもっとも重要な概念のひとつですので，知っていて損はありません．また，真の解が既知の問題に対してあるアルゴリズムを適用し，その有効性を確かめたりする場合には，誤差の知識が必要になります．

3.1　誤差とは

以下は概念的な式です．数値計算による近似値は，真値 (存在を仮定します) に誤差が加わったもの，つまり

$$\text{近似値} = \text{真値} + \text{誤差}$$

と考えます．これを移項すると，誤差の定義は

$$\text{誤差} = \text{近似値} - \text{真値}$$

となります．つまり誤差を厳密に考える場合，符号も考慮する必要があります[21]．ただし数値計算の教科書では，符号を無視して誤差のノルムを考えることが多いようです．本稿も誤差をノルムで定義することにします．

○絶対誤差

真値 $\boldsymbol{x}$ と近似値 $\hat{\boldsymbol{x}}$ との差のノルム (ノルムの種類は問いません)

$$\|\boldsymbol{x} - \hat{\boldsymbol{x}}\| \quad (= \|\hat{\boldsymbol{x}} - \boldsymbol{x}\|)$$

を**絶対誤差** ($absolute\ error$)，あるいは単に「誤差」と呼びます[22]．

○相対誤差

真値 $\boldsymbol{x} \neq \boldsymbol{0}$ と近似値 $\hat{\boldsymbol{x}}$ に対して

$$\frac{\|\boldsymbol{x} - \hat{\boldsymbol{x}}\|}{\|\boldsymbol{x}\|} \tag{2}$$

を**相対誤差** ($relative\ error$) と呼びます．相対誤差は真値に対する絶対誤差の割合 (相対的な大きさ) を表します．

通常，数値計算で求めたい正体不明の真値 $\boldsymbol{x}$ のノルム $\|\boldsymbol{x}\|$ を評価することはできません．しかし，なんらかの方法で

$$\frac{\|\boldsymbol{x} - \hat{\boldsymbol{x}}\|}{\|\hat{\boldsymbol{x}}\|} \tag{3}$$

を評価できる場合があります．よって (3) を相対誤差として定義することもあります．ここで，$\hat{\boldsymbol{x}}$ が $\boldsymbol{x}$ に十分近くなれば，(2) も (3) も非常に近い値を取りながら 0 に近づくことに注意してください．どちらにしても，相対誤差の定義を，採用したノルムの種類とともに論文に明記することが大事です．

[21] 真値 − 近似値 を「補整」と呼びます．

[22] 本によっては $\hat{\boldsymbol{x}} - \boldsymbol{x}$ を「誤差」，$\|\hat{\boldsymbol{x}} - \boldsymbol{x}\|$ を「絶対誤差」と分けて定義することもあります．また「絶対誤差」の「絶対」の由来については，相対誤差との対比として「絶対」の名前が付いたと書いてある本と，“絶対値ではかった誤差” であることから名付けられた，と書いてある本に分かれます．

3.2　なぜ誤差の種類が二つあるのか？

○絶対誤差の問題点

絶対誤差の問題点として，真値 $\boldsymbol{x}$ の大きさによって近似の意味が変わってしまうことがあげられます．簡単のため 1 次元で考えます．真の値 x と近似値 $\hat{x}$ の絶対誤差が同じ 0.1 であったとしても，$x = 100$, $\hat{x} = 99.9$ の場合と $x = 0.2$, $\hat{x} = 0.1$ の場合では「どちらも誤差が小さいよ」とは一概に言えません．図 2 を見ると，近似値 0.1 と同じ大きさ (0.1) の誤差が「とんでもなく大きい」という状況も十分に考えられます．

図 2: $|x - \hat{x}| = 0.1$ となる x と $\hat{x}$ (左右のスケールは異なります)

相対誤差は誤差の影響の「重大さ」を比較的よく表します．今の例では $x = 100$, $\hat{x} = 99.9$ の相対誤差は 0.001，$x = 0.2$, $\hat{x} = 0.1$ の相対誤差は 0.5 になり，相対誤差の方が図 2 の『見た感じ』をよく説明していることと思います．

また，“$\log_{10}$(相対誤差の逆数)” を近似値の**有効桁数**といい，「真値と何桁一致しているのか」を調べる《指標》としてよく使われます．たとえば $x = \underline{1000}$, $\hat{x} = \underline{1001}$ は相対誤差の逆数 10^3 より有効桁数は 3 となります．ここで，場合によっては有効桁数は「一致する桁数」を反映しないことに注意してください．たとえば，「一致する桁数」がひと桁も合っていない $x = 100$, $\hat{x} = 99.9$ の有効桁数も同じく 3 となります[23]．

○閑話

ごくわかりやすい例をもう一つあげて，絶対誤差と相対誤差の違いを見てみます．幼稚園や小学生の低学年の頃には，年が一つ上の人はずいぶん「おとな」に見えたものです．しかし，齢を重ねるにしたがい，だんだんと年の差を感じなくなってきます．この感覚は相対誤差を使うと納得できます．

女の子の年齢を x，男の子の年齢を $\hat{x}$ とします．女の子が 5 才，男の子が 6 才のとき (図 3)，絶対誤差は 1，相対誤差は 0.2 です．この二人が風雪を乗り越え，無事 99 才と 100 才を迎えたとします (図 4)．このとき，絶対誤差は変わらず 1 です．一方，相対誤差は $1/99 \approx 0.0101$ になっており，年齢 (誤差) が接近していることがわかります．

図 3: 6 才と 5 才 (イメージ)

図 4: 100 才と 99 才 (イメージ)

○相対誤差の問題点

相対誤差の問題点は，(2) または (3) の分母が 0 または 0 に非常に近くなると，計算不可能で実行を打ち切られたり，オーバーフローを起こす危険があることです．プログラムを組む場合には，分母の値が 0 に近くなったら警告を出すか，絶対誤差に切替えるなどの工夫が必要です．なお [4] に絶対誤差と相対誤差を組み合わせた評価方法が紹介されています．

[23] 有効桁数はこのような「よい近似であるのにひと桁も一致していない」という不合理をなくすために導入されました．有効桁数は「精度桁数」とも呼ばれます．また，有効桁数の定義をもって「一致する桁数」だと定める本もあります．

3.3 マシン・イプシロン

現在ほとんどすべての計算機で採用されている浮動小数点体系は，数学的な実数体系とは違い，有限個の要素から成る集合です．計算機の中では，実数を有限の 0 と 1 の組合せで近似します．したがって，計算のたびに，本来実数であって欲しい真値 x を適当な浮動小数点 $\hat{x}$ で近似する操作が入ります．つまり**誤差が発生する**わけです (詳しくは [3], [5], [9] を参照してください)．

マシン・イプシロン (*machine epsilon*)[24]は

$$1 + \varepsilon > 1 \tag{4}$$

を満たす最小の浮動小数点数で定義されます．左辺の足し算は浮動小数点演算での加算を意味します．マシン・イプシロンは浮動小数点演算における最大の相対誤差をあらわす数と考えることができ，反復計算の収束判定にとって重要な数です．なぜならば，どんな単純な演算であっても，最悪でマシン・イプシロンの大きさの誤差が入り込むことを覚悟する必要があるからです．

ここで，マシン・イプシロンは最小の正の浮動小数点数では ない ことに注意してください．最小の数は後の数値例で見る通り，もっと小さい値です．マシン・イプシロンは，計算機の採用している浮動小数点の形式，精度によって異なります． [5] には (4) の定義に基づくマシン・イプシロン生成プログラムが紹介されています．

Fortran 90 の規格からは，組み込み関数を用いてマシン・イプシロンの値を調べることができるようになりました．以下は VPP700/56 がサポートしている各数値を問い合わせるプログラムです．処理系によっては 8 バイト整数型と 4 倍精度実数型は未サポートの場合があります．

```
program machine_epsilon         ! プログラムの開始 (プログラム名は適当に記述します)
  implicit none                 ! 暗黙の型宣言の抑止
  integer(kind=4) :: i          ! 4 バイト整数型の宣言
  integer(kind=8) :: j          ! 8 バイト整数型の宣言
  real(kind=4)    :: x          ! 実数型の宣言
  real(kind=8)    :: y          ! 倍精度実数型の宣言
  real(kind=16)   :: z          ! 4 倍精度実数型の宣言
  intrinsic epsilon,huge,tiny   ! 組み込み手続きを明確にする
                                !     (ソースプログラムの改行に深い意味はありません)
  write(6,*) epsilon(x)         ! 実数型のマシン・イプシロンの出力
  write(6,*) epsilon(y)         ! 倍精度実数型のマシン・イプシロンの出力
  write(6,*) epsilon(z)         ! 4 倍精度実数型のマシン・イプシロンの出力
  write(6,*)                    !     (一行空ける)
  write(6,*) huge(i)            ! 4 バイト整数型の最大値を出力
  write(6,*) huge(j)            ! 8 バイト整数型の最大値を出力
  write(6,*) huge(x)            ! 実数型の最大値を出力
  write(6,*) huge(y)            ! 倍精度実数型の最大値を出力
  write(6,*) huge(z)            ! 4 倍精度実数型の最大値を出力
  write(6,*)                    !     (一行空ける)
  write(6,*) tiny(x)            ! 実数型の正の最小数を出力
  write(6,*) tiny(y)            ! 倍精度実数型の正の最小数を出力
  write(6,*) tiny(z)            ! 4 倍精度実数型の正の最小数を出力
end program machine_epsilon     ! プログラムの終り
```

表 5 にプログラムで使用した数値問い合わせ関数を紹介します．

[24]「計算機イプシロン」「機種精度」「マシン・エプシロン」とも言います．ギリシャ語と英語の発音では「エプシロン」が近いはずです．ここでは「イプシロン」と書く本が多かったのであえて逆らいませんでした．また，「マシン・イプシロンが浮動小数点演算の最大の相対誤差を表す」ことは，IEEE 標準 754 以外の規格では厳密に成立しません．それでもほとんどがマシン・イプシロンの 2 倍か 3 倍で評価できます．

表 5: Fortran の数値問い合わせ関数 (一部)

関数名	機能
EPSILON	引数の体系におけるマシン・イプシロンを求める
TINY	引数の体系における正の最小数を求める
HUGE	引数の体系における最大値を求める

以下はプログラムを VPP700/56 で翻訳・実行した結果です:

```
1.19209290E-07                                  ⇐ 実数型のマシン・イプシロン = 2^-23
2.220446049250313E-16                           ⇐ 倍精度実数型のマシン・イプシロン = 2^-52
1.92592994438723585305597794258492 73E-0034    ⇐ 4 倍精度実数型のマシン・イプシロン = 2^-113

2147483647                                      ⇐ 4 バイト整数型の最大値 = 2^31 − 1
9223372036854775807                             ⇐ 8 バイト整数型の最大値 = 2^63 − 1
3.40282347E+38                                  ⇐ 実数型の最大値 = (1 − 2^-24)2^128
1.797693134862316+308                           ⇐ 倍精度実数型の最大値 = (1 − 2^-53)2^1024
1.18973149535723176508575932662800 70E+4932     ⇐ 4 倍精度実数型の最大値 = (1 − 2^-112)2^16384

1.17549435E-38                                  ⇐ 実数型の正の最小数 = 2^-126
2.225073858507201-308                           ⇐ 倍精度実数型の正の最小数 = 2^-1022
3.36210314311209350626267781732175 26E-4932     ⇐ 4 倍精度実数型の正の最小数 =2^-16382
```

3.4 収束の定義

この章の最後に，収束の定義を紹介します．

【収束の定義】

$\boldsymbol{R}^n$ のベクトル列 $\boldsymbol{x}^{(k)} = (x_1^{(k)}, \ldots, x_n^{(k)})^T$ がベクトル $\boldsymbol{a} = (a_1, \ldots, a_n)^T$ に対して

$$x_i^{(k)} \to a_i \ (k \to \infty) \quad (1 \le i \le n) \tag{5}$$

となるとき，$\{\boldsymbol{x}^{(k)}\}$ は $\boldsymbol{a}$ に**収束する** $(converge)$ という．

(5) をまとめて “$\lim_{k\to\infty} \boldsymbol{x}^{(k)} = \boldsymbol{a}$” と書くこともあります[25]．

ノルムの性質より，“$x_i^{(k)} \to a_i \ (k \to \infty)$” は “$\|\boldsymbol{x}^{(k)} - \boldsymbol{a}\| \to 0 \ (k \to \infty)$” と同値になります (証明はたとえば [11] をご覧ください)．したがってベクトル列の収束は，絶対誤差が 0 に収束することと同じです．しかし，計算機を使って (5) の収束を直接確かめることは困難です．浮動小数点演算によって発生する誤差を完全に回避するためには無限桁が必要な上に，極限までベクトル列を追いかけるためには無限の時間が必要だからです．反復計算では，ベクトル列 $\boldsymbol{x}^{(k)}$ をいつ収束したと《見なす》のかが大事になります．

以上の準備のもと，次章から収束判定基準を紹介します．お待たせしました．

[25] 極限の定義をさらに書き下す教科書では，収束の定義を次のように書くことがあります: 『任意の $\varepsilon > 0$ に対し，ある K が定まり，$k > K$ であれば $\|\boldsymbol{x}^{(k)} - \boldsymbol{a}\| < \varepsilon$ が必ず成り立つ．』

4　収束判定基準の設定方法

反復列 $\boldsymbol{x}^{(k)}$ が収束の定義 (5) の条件を満たすかどうかを有限の資源しか持たない計算機で確認することは，真の解 $\boldsymbol{a}$ の形が具体的にわかっていたとしても困難です．問題によっては，ある条件をチェックすることで収束性や誤差評価が得られることもあります．しかし，理論的な裏付けが得られない場合でも，本章で説明する様々な基準を参考にして，『えいやっ』で反復を停止することがないように努めなければなりません．

この章では，収束判定基準[26]の代表的な設定方法を紹介します．名の通った数値計算ライブラリには必ず収束判定基準が明示してあり，なおかつ利用者がいろいろな基準値を設定できる仕様になっているはずです．数値計算ライブラリが利用者にとってブラックボックスとなる危険を避けるためにも，収束判定の意味を理解しておくことが必要です．

4.1　残差基準

もともとの問題は「$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$ を満たす $\boldsymbol{x}$ を計算機を用いた反復計算により求める」ということでした．したがって，反復計算により更新されるベクトル $\boldsymbol{x}^{(k)}$ を (1) に代入して，ノルムが十分小さいかどうか調べるという方法が考えられます．この判定方法を**残差基準の判定**[27]といいます．

【残差基準による判定】

収束判定用の定数 $\varepsilon > 0$ を決めておき，

$$\|\boldsymbol{f}(\boldsymbol{x}^{(k)})\| < \varepsilon \tag{6}$$

ならば反復は収束したと《見なして》計算を打ち切る．

$\boldsymbol{f}(\boldsymbol{x}^{(k)})$ を方程式 (1) の**残差** $(residual)$ と呼びます．たとえば連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ では "$\boldsymbol{b} - A\boldsymbol{x}^{(k)}$" または "$A\boldsymbol{x}^{(k)} - \boldsymbol{b}$" が残差になります．

○相対的な残差基準

$\boldsymbol{x}^{(k)}$ との相対的な残差

$$\frac{\|\boldsymbol{f}(\boldsymbol{x}^{(k)})\|}{\|\boldsymbol{x}^{(k)}\|} < \varepsilon$$

を判定基準に採用することもあります．相対的な残差基準の判定は，共役勾配法系の連立 1 次方程式の反復解法によく用いられます[28]．詳しくは [1] を参照してください．

○要素ごとの残差基準

(6) はノルムを使った基準です．ノルムはベクトルの大きさをはかる大切な指標です．しかし，感覚的に表現すると，ノルムは n 個の異なる値を持つ要素を「べちゃっ」と押しつぶして，ある 1 つの (1 次元の) 値で代表させたものです．この値を見るだけでは，それぞれの要素の値が極端に異なっていたり，特定の要素だけが反復のたびに大きく変動しているといった情報を得ることは困難です．したがって，よりきめ細かい評価を必要とするならば，$f_i(\boldsymbol{x}^{(k)})(1 \leq i \leq n)$ それぞれに対して残差基準を設定する方法も考えられます[29]．

---

[26]「反復停止規準」「収束基準」「収束判定条件」「停止規則」「停止条件」ともいいます．

[27]「ε 法」と呼ばれることもあります．

[28]理由はアルゴリズムに残差の計算が出てくるからです．連立 1 次方程式では $\dfrac{\|\boldsymbol{b} - A\boldsymbol{x}^{(k)}\|}{\|\boldsymbol{b}\|} < \varepsilon$ もよく判定に使われます．

[29]少なくとも，要素ごとの値を出力するプログラムを書いておくことはデバッグに有効です．

○残差基準の注意点

残差 $\boldsymbol{f}(\boldsymbol{x}^{(k)})$ の計算では，同じような大きさの値の引き算がよく出てきます．そのため，桁落ち[30]による精度損失が起こる可能性が高くなります．結果が思わしくないときには，残差の計算部分の桁数を多くとってみることをお勧めします[31]．また，残差 $\boldsymbol{f}(\boldsymbol{x}^{(k)})$ は「ゼロになって欲しいベクトル」です．もし不用意にこの値を分母にもってくると，計算が破綻する危険があることに注意してください．

4.2　誤差基準

数値計算は一定の桁数の有限操作であることから，修正量がある程度小さくなると，あとは事実上近似解の値が変わらなくなったり，微小な振動が続いたりします．そうなった場合，これ以上計算を続けても無駄だという判断のもと反復を停止する方法が**誤差基準の判定**[32]です．

【誤差基準の判定 (1)】

収束判定用の定数 $\varepsilon > 0$ を決めておき，

$$\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}\| < \varepsilon \tag{7}$$

ならば反復は収束したと《見なして》計算を打ち切る．

もし $\boldsymbol{x}^{(k)}$ が真の解に十分近いならば，(7) は絶対誤差に近いことが期待されます．
また，相対的な修正量を判定基準にする方法もよく使われます．

【誤差基準の判定 (2)】

収束判定用の定数 $\varepsilon > 0$ を決めておき，

$$\frac{\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}\|}{\|\boldsymbol{x}^{(k-1)}\|} < \varepsilon \tag{8}$$

ならば反復は収束したと《見なして》計算を打ち切る．

もし $\boldsymbol{x}^{(k)}$ が真の解に十分近いならば，(8) は相対誤差に近いことが期待されます．相対的な判定基準 (8) を用いることの利点は，3.2節で説明した通りです．

○要素ごとの誤差基準

残差基準と同じく，きめ細かい評価としてベクトル $\boldsymbol{x}^{(k)}$, $\boldsymbol{x}^{(k-1)}$ の個々の要素に対して判定基準 (7), (8) を設定することも考えられます．

4.3　収束判定基準から誤差評価がいえるのか？

残念ながら，(6), (7), (8) の判定によって得られた $\boldsymbol{x}^{(k)}$ と (1) の真の解 $\boldsymbol{a}$ (とおきます) に対して，絶対誤差または相対誤差評価:

$$\|\boldsymbol{x}^{(k)} - \boldsymbol{a}\| < \varepsilon, \qquad \frac{\|\boldsymbol{x}^{(k)} - \boldsymbol{a}\|}{\|\boldsymbol{a}\|} < \varepsilon$$

[30] 絶対値がほぼ等しい 2 つの数を加減して答が著しく小さくなる現象です．小さくなった分，相対誤差が増大し，有効数字が失われます．

[31] 単精度の場合は倍精度で，倍精度のときは 4 倍精度または拡張倍精度での計算になります．実際，多くの数値計算ライブラリでこの方法による桁落ち回避がなされています．

[32] 「δ 法」と呼ばれることもあります．

は**一般に成立しません**．問題 ($\boldsymbol{f}$ の形) と反復法の手順によっては，収束判定条件とその他の条件を組み合わせることで理論的な誤差限界を導くことができる場合もあります[33]．しかし一般には，収束判定用の定数 ε は「真の解との誤差」でも「真の解と一致する桁数」でもありません．単に計算する人が勝手に決めた数です． $\boldsymbol{x}^{(k)}$ は $\boldsymbol{a}$ から遠く離れているかもしれません．また，そもそも真の解 $\boldsymbol{a}$ が存在しない可能性もあります． [16] に 1 次元のわかりやすい例が紹介されています．図 5 として引用します．

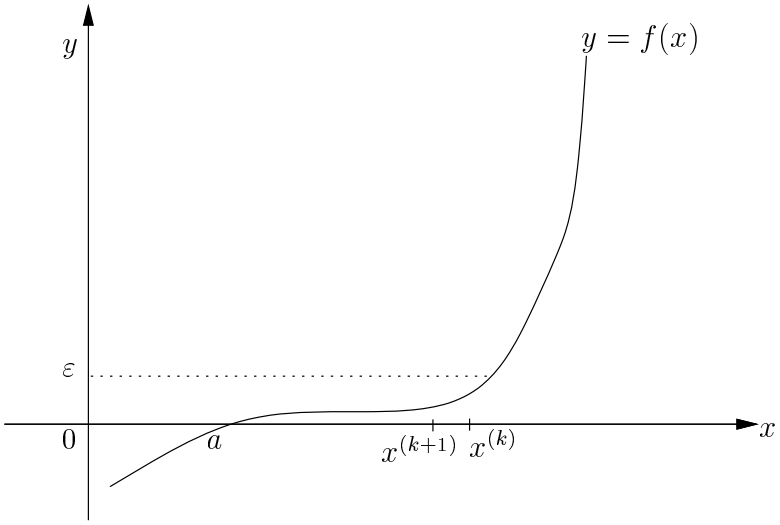


図 5: 真の解が離れている例

図 5 で ε を収束判定用の定数， a を $f(x) = 0$ の解， $x^{(k)}$ の次の反復が $x^{(k+1)}$ になったとします．このとき， $|x^{(k+1)} - x^{(k)}| < \varepsilon$ という条件を設定していたならば，反復は停止します．しかし，どう見ても $|a - x^{(k+1)}| < \varepsilon$ は成り立ちません．また， $\|f(x^{(k)})\| < \varepsilon$ という条件でも反復は停止します．しかし，同じく真の解はまだまだ先です．微分を次の反復の重要な手がかりとして用いる Newton-Rapson 法の立場から図を眺めると，残差基準の判定はグラフの傾斜がなだらかになる部分，誤差基準の判定法は $\boldsymbol{f}(\boldsymbol{x})$ が急変する部分で判断を誤る危険があると言えます．

4.4 どっちを選ぶ？

では，理論的な知見が何もない問題の近似解を反復計算によって求める場合，残差基準と誤差基準のどちらを選べばよいのでしょうか？数値解析や数値シミュレーションの本を読めばおわかりのように，はっきりいってバラバラです[34]．数値解析の分野では，本で調べたり人に聞いてみた限り，残差基準を採用する人が多いようです[35]．数値計算ライブラリの中には残差基準と誤差基準を問題に応じて選択できるものもあります．

残差基準と誤差基準はそれぞれ「計算の限界」という意味で根拠を持っています．したがって，浮動小数点演算の限界 (具体的にはマシン・イプシロンです． 4.6節を参照してください) 以下の値を収束判定用の定数に設定する時には，残差基準または誤差基準のどちらか一方の条件が成立した場合に反復を打ち切るようにプログラムを組むことをお勧めします．もし両方の条件の成立を待って反復を終了するように設定していた場合，一方の条件がいつまでも満足されないという状況も十分に考えられるからです (さらに詳しい考察として [14] をお勧めします)．

しかしながら， 4.3節で説明したように，得られた近似解 $\boldsymbol{x}^{(k)}$ が真の解 $\boldsymbol{a}$ に十分近いという保証は与えられていないことに十分注意してください．理論的な保証をえるためには，近似解 $\boldsymbol{x}^{(k)}$ のまわりでの $\boldsymbol{f}$ のふるまいを厳密に観察する必要があります． 4.7節ではこのような手法について簡単に触れます．

[33] よく見る評価式は，ある定数 $C > 0$ に対して $\|\boldsymbol{x}^{(k)} - \boldsymbol{a}\| \leq C\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}\|$ というものです． C の値が具体的にわかることは稀で，存在しか分かっていないことがほとんどです．とんでもなく大きな値かもしれません．それでもアルゴリズムの正当性の主張には説得力を持ちます．

[34] 一方の基準だけを紹介して，他の基準を紹介していない本もたくさんあります．経験的に，両方を紹介している本は良い本です．もちろん，逆は成り立ちません．

[35] 説得力のある意見が [10] に書いてあります．

4.5 Fortran プログラム例

図 1 で紹介した非線形写像 $\boldsymbol{f}$ に対し，方程式 $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$ を Newton-Raphson 法で数値的に解く Fortran プログラムを紹介します[36]．メインプログラム “Newton_Raphson” からは $\boldsymbol{f}$ を計算する “Generate_Function”， Jacobi 行列を求める “Generate_Jacobian”，連立 1 次方程式を Gauss の消去法で解く SSL II の “DVLAX” を外部サブルーチンとして呼び出しています[37]．精度は倍精度です．

初期値はすべての値を 1 に設定しています． $\mathcal{R} = 10$, $\mathcal{P}$=1 です．最大反復回数は 100 回に設定しています．収束判定の基準として，マシン・イプシロンの 2 倍の epsz に対して (6) または (8) が満たされたとき反復を停止し，近似解の値を出力したうえで計算を終了します．ノルムは最大ノルムです．

```
program Newton_Raphson                                        ! メインプログラムの開始
  implicit none                                               ! 暗黙の型宣言の抑止
  integer(kind=4),parameter   :: n=10                         ! 次元をパラメータで設定
  real(kind=8),dimension(n)   :: v,f,u,vw                     ! ベクトルの宣言
  real(kind=8),dimension(n,n) :: G                            ! Jacobi 行列の宣言
  real(kind=8),dimension(2)   :: error                        ! 残差基準, 誤差基準用変数の宣言
  real(kind=8)                :: P,R,epsz                     ! 定数, 収束判定用の定数の宣言
  integer(kind=4)             :: i,k,imax,icon,isw,is         ! 整数型変数の宣言
  integer(kind=4),dimension(n):: ip                           ! dvlax で利用する配列の宣言
  intrinsic abs,maxval                                        ! 組み込み手続きを明示
  external  dvlax,Generate_Function,Generate_Jacobian         ! 外部手続きを明示
  v=1.0D0                                                     ! 反復の初期値を 1 に設定
  P=1.0D0                                                     ! 定数値 P の設定
  R=10.0D0                                                    ! 定数値 R の設定
  epsz=2*epsilon(epsz)                                        ! 収束判定用の定数を設定
  imax=100                                                    ! 反復回数の最大値を設定
  isw=1                                                       ! LU 分解を行なう (dvlax)
     do k=1,imax                                              ! 反復開始
       u=v                                                    ! 前回の反復列を u に格納
       call Generate_Function(u,v,P,R)                        ! f(u) の値を計算
       call Generate_Jacobian(u,G,P,R)                        ! Jacobian を計算
       call dvlax(G,n,n,v,epsz,isw,is,vw,ip,icon)             ! 連立 1 次方程式を解く (dvlax)
         if(icon /= 0) then                                   !-+
           write(6,*) 'Error occurred in DVLAX:',icon         ! +-dvlax の正常終了を確認
           stop                                               ! | 正常終了でない場合は停止
         end if                                               !-+
       v=u-v                                                  ! 新しい反復列を v に格納
       call Generate_Function(v,f,P,R)                        ! 残差を計算
       error(1)=maxval(abs(f))                                ! 残差の最大ノルムを計算
       error(2)=maxval(abs(v-u))/maxval(abs(u))               ! 相対的な誤差基準の値を計算
       write(6,'(I3,2E15.7)') k,error(1),error(2)             ! 反復回数と各値の書きだし
       if( error(1)<epsz .or. error(2)<epsz ) then            !-+
         write(6,*) 'approximate solution: '                  ! |
         do i=1,n                                             ! +- ★収束判定★
           write(6,'(i4,E25.13)') i,v(i)                      ! |   残差基準と誤差基準の一方が
         end do                                               ! |   満たされた場合，結果を書き
         exit                                                 ! |   ループを抜ける
       end if                                                 !-+
     end do                                                   ! 反復を繰り返す
end program Newton_Raphson                                    ! メインプログラムの終了
```

プログラムのサブルーチン部分です． 1 ページに収めるため，無理して詰め込んでいます． “&” は，文章の継続の印です． “intent” で引数の特性 (入力，出力，入出力) を陽に指定しています．

[36] http://www.cc.kyushu-u.ac.jp/RD/watanabe/RESERCH/MANUSCRIPT/KOHO/CONVERGE/Newton-Raphson.f90

[37] センターで公開している科学技術計算用サブルーチンライブラリです．DVLAX の引数の詳細はたとえば kyu-vpp の man dvlax で調べることができます． SSL II の利用できないシステムにプログラムを移植するときは，この部分を差し替える必要があります．またセンターでこのプログラムを実行するときは，ファイルのサフィックスを “.f90” にしてください．

```
subroutine Generate_Function(v,f,P,R)
  implicit none
  real(kind=8),dimension(10),intent(in)  :: v
  real(kind=8),dimension(10),intent(out) :: f
  real(kind=8),intent(in)                :: P,R
  real(kind=8)                           :: S
  intrinsic sqrt
  S=sqrt(2.0D0)
       f(1)= (9*P*v(1))/4  + (9*v(2)*v(3))/(8*S) + (P*R*v(7))/S
       f(2)= (81*P*v(2))/4 + (9*v(1)*v(3))/(8*S) + (P*R*v(8))/S
       f(3)= (-9*v(1)*v(2))/(4*S) + 9*P*v(3) + S*P*R*v(9)
       f(4)= 36*P*v(4) + S*P*R*v(10)
       f(5)= -2*v(5) + (v(2)*v(7))/(2*S) + (v(1)*v(8))/(2*S) - (v(4)*v(9))/S +      &
              S*v(4)*v(9) - (v(3)*v(10))/S + S*v(3)*v(10)
       f(6)= -8*v(6) - (v(1)*v(7))/S - S*v(3)*v(9)
       f(7)= -(v(1)/S) - (v(2)*v(5))/(2*S) + (v(1)*v(6))/S -  (3*v(7))/2.0D0 +      &
              (3*v(3)*v(8))/(4*S) + (3*v(2)*v(9))/(4*S)
       f(8)= -(v(2)/S) - (v(1)*v(5))/(2*S) - (3*v(3)*v(7))/(4*S) - (9*v(8))/2.0D0 - &
              (3*v(1)*v(9))/(4*S)
       f(9)= -(S*v(3)) - (v(4)*v(5))/S + S*v(3)*v(6) - (3*v(2)*v(7))/(4*S) +        &
              (3*v(1)*v(8))/(4*S) - 3*v(9)
       f(10)= -(S*v(4)) - (v(3)*v(5))/S - 6*v(10)
end subroutine Generate_Function
```

“;” を用いると，一行に複数の命令を記述できます．この機能は Fortran 90 からです．

```
subroutine Generate_Jacobian(v,G,P,R)
  implicit none
  real(kind=8),dimension(10),intent(in)     :: v
  real(kind=8),dimension(10,10),intent(out) :: G
  real(kind=8),intent(in)                   :: P,R
  real(kind=8)                              :: S
  intrinsic sqrt
  S=sqrt(2.0D0)
     G=0.0D0
     G(1,1) = (9*P)/4.0D0            ;  G(7,1)  = -(1/S)+v(6)/S
     G(1,2) = (9*v(3))/(8*S)         ;  G(7,2)  = -v(5)/(2*S) + (3*v(9))/(4*S)
     G(1,3) = (9*v(2))/(8*S)         ;  G(7,3)  = (3*v(8))/(4*S)
     G(1,7) = (P*R)/S                ;  G(7,5)  = -v(2)/(2*S)
     G(2,1) = (9*v(3))/(8*S)         ;  G(7,6)  =  v(1)/S
     G(2,2) =  (81*P)/4.0D0          ;  G(7,7)  = -1.5D0
     G(2,3) =  (9*v(1))/(8*S)        ;  G(7,8)  = (3*v(3))/(4*S)
     G(2,8) = (P*R)/S                ;  G(7,9)  = (3*v(2))/(4*S)
     G(3,1) = (-9*v(2))/(4*S)        ;  G(8,1)  = -v(5)/(2*S) - (3*v(9))/(4*S)
     G(3,2) = (-9*v(1))/(4*S)        ;  G(8,2)  = -(1/S)
     G(3,3) = 9*P                    ;  G(8,3)  =  (-3*v(7))/(4*S)
     G(3,9) = S*P*R                  ;  G(8,5)  = -v(1)/(2*S)
     G(4,4) = 36*P                   ;  G(8,7)  = (-3*v(3))/(4*S)
     G(4,10)= S*P*R                  ;  G(8,8)  = -4.5D0
     G(5,1) = v(8)/(2*S)             ;  G(8,9)  = (-3*v(1))/(4*S)
     G(5,2) = v(7)/(2*S)             ;  G(9,1)  = (3*v(8))/(4*S)
     G(5,3) = -(v(10)/S) + S*v(10)   ;  G(9,2)  = (-3*v(7))/(4*S)
     G(5,4) = -(v(9)/S) + S*v(9)     ;  G(9,3)  = -S + S*v(6)
     G(5,5) = -2.0D0                 ;  G(9,4)  = -(v(5)/S)
     G(5,7) =  v(2)/(2*S)            ;  G(9,5)  = -(v(4)/S)
     G(5,8) =  v(1)/(2*S)            ;  G(9,6)  = S*v(3)
     G(5,9) =  -(v(4)/S) + S*v(4)    ;  G(9,7)  = (-3*v(2))/(4*S)
     G(5,10)=  -(v(3)/S) + S*v(3)    ;  G(9,8)  = (3*v(1))/(4*S)
     G(6,1) = -(v(7)/S)              ;  G(9,9)  = -3.0D0
     G(6,3) = -(S*v(9))              ;  G(10,3) = -(v(5)/S)
     G(6,6) = -8.0D0                 ;  G(10,4) = -S
     G(6,7) = -(v(1)/S)              ;  G(10,5) = -(v(3)/S)
     G(6,9) = -(S*v(3))              ;  G(10,10)= -6.0D0
end subroutine Generate_Jacobian
```

表 6 は VPP700/56(IEEE 形式 64 ビット) で計算した反復回数，最大ノルムで測った (6) の残差, (8) の誤差の値です． 9 回の反復で停止条件を満たしました．

表 6: VPP700/56 での実行結果

反復回数	残差基準の判定値	誤差基準の判定値
1	$0.3462929E+01$	$0.2169360E+01$
2	$0.3055435E+02$	$0.2986887E+01$
3	$0.7771962E+01$	$0.4004562E+00$
4	$0.1856027E+01$	$0.3439725E+00$
5	$0.3602276E+00$	$0.2363468E+00$
6	$0.3191305E-01$	$0.9436822E-01$
7	$0.3268139E-03$	$0.1087901E-01$
8	$0.3519620E-07$	$0.1160453E-03$
9	$0.4108034E-16$	$0.1285530E-07$

4.6 収束判定用の定数の設定

理論的な収束証明が保証されていたり，あらかじめ真の解がわかっていて，アルゴリズムのテストとして反復計算を実行している場合を除けば，収束判定用の定数の選び方には十分な注意が必要です．

○マシン・イプシロン以下に設定しない

浮動小数点演算では，どのような簡単な演算でも最悪でマシン・イプシロンの誤差が混入します．数値計算は四則演算と sin, cos などの初等関数に代表される関数の集合体です．そして，演算の度に浮動小数点演算の誤差はどんどん蓄積しているかも知れません[38]．したがって，収束判定用の定数にマシン・イプシロン以下の値を設定するのは危険です．反復が判定基準に到達しない可能性が高いからです．

例を挙げます． $n \times n$ 行列 $A = (a_{ij})$， n 次ベクトル $\boldsymbol{b} = (b_i)$ を

$$a_{ij} = \begin{cases} \sqrt{\dfrac{2}{n+1}} \sin\left(\dfrac{ij\pi}{n+1}\right) & (i \neq j) \\ \sqrt{\dfrac{2}{n+1}} \sin\left(\dfrac{ij\pi}{n+1}\right) \times 2n & (i = j) \end{cases}, \qquad b_i = \sum_{j=1}^{n} a_{ij}$$

で定義し，連立 1 次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ を SOR 法[39]を用いて数値的に解きます．加速係数 ω は 1.5 に設定しました．計算機は wisdom(FUJITSU S-4/1000E)，精度は IEEE 形式の単精度，倍精度，および富士通拡張仕様の 4 倍精度を用いました．図 6 は $n = 10$ の残差の最大ノルムを描いたものです．

最初はそれぞれの精度型とも同じように残差ノルムが小さくなっていきます (線が重なっています)．反復が進むと，まず単精度のマシン・イプシロン近くで単精度の残差が振動を始めました．以下，倍精度，4 倍精度のマシン・イプシロンに近付くとそれぞれの精度の残差が振動するようになり，それ以上小さくなりません．したがって，この例で収束判定用の定数にマシン・イプシロン以下を設定すると，いつまでたっても反復が終らないことになります[40]．

---

[38]実際は都合よく打ち消しあって誤差の累積がほとんどないこともあります．しかし，問題の特異性が強い場合には，あっという間に有効桁が誤差で汚されることもあります．

[39]SOR 法 (Successive Over-Relaxation method) は，過剰緩和法，逐次過緩和法ともいいます．行列 A を下三角部分 L，対角部分 D，上三角部分 U を用いて $A = L + D + U$ と分解し，加速係数 $1 < \omega < 2$ により $\boldsymbol{x}^{(k)} := (D + \omega L)^{-1}(\omega \boldsymbol{b} + ((1-\omega)D - \omega U)\boldsymbol{x}^{(k-1)})$ で反復計算を行なう手法です． $\omega = 1$ のときを Gauss-Seidel 法と呼びます．詳しくは [7] を参照してください．

[40]数値計算ライブラリの中には，利用者が収束判定用の定数としてマシン・イプシロン以下の値を入力した場合，勝手に値を修正するものもあります．もちろん，マニュアルにはその旨きちんと書いてあります．

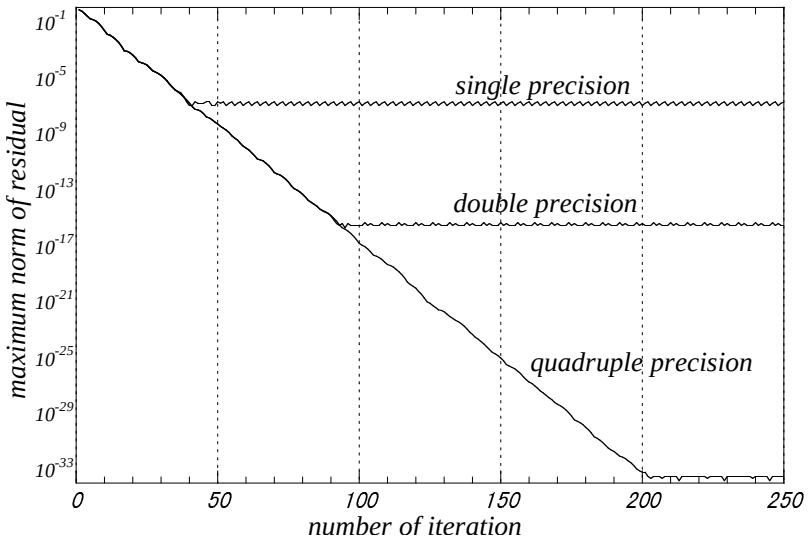


図 6: SOR 法による残差ノルムの推移

特に大規模な数値計算では，安全を考えて，マシン・イプシロンのある倍数を収束判定用の定数に設定するべきでしょう[41]．なお [14] では，残差が妙な挙動を始めた時に反復を終了する方法が紹介されています[42]．

○次元数に注意

次元数 n が大きくなると，解く問題が大規模になります．問題が大規模になると，演算数が増大します．演算数が増大すると，よりいっそう浮動小数点演算の誤差の累積を覚悟しなくてはいけません．n をパラメータとして調整可能なようにプログラムを作成しておくと，小規模なテスト問題を大規模な数値シミュレーションに拡張して使用することができます．このような (ほとんど同じ) プログラムを実行する場合でも，大きな n に対しては小さな n の時よりも「甘い」収束判定基準を設定しないと反復が終わらないかもしれません．

○ノルムの種類に注意

ノルムの定義で説明したように，1 ノルムと 2 ノルムは次元に依存します．したがってこれらのノルムを用いて収束判定を行なう場合は，前の項目と同じ理由で収束判定基準を調整する必要があります．具体的には，次元に依存しない最大ノルムを基準に 2.3節の不等式評価を考えて，(6) や (7) の評価では 1 ノルムはマシン・イプシロンの n 倍，2 ノルムはマシン・イプシロンの $\sqrt{n}$ 倍より小さく しない 方がいいでしょう．

○解が重根を持つ場合

解が重根を持っていたり，非常に近い場所にある場合，収束判定用の定数を大きめに設定する必要が出てきます．あらかじめ解の性質がわかっている場合には判定基準を調整するようにしてください．

4.7　近似解を使って真の解の存在を確認してやめる

残差基準または誤差基準で得られた近似解をもとに，真の解の存在を厳密な誤差限界とともに証明してしまうという手法があります．手法は，(1) を有限次元の不動点問題に帰着させ，浮動小数点演算に

[41] 残差基準の場合，「ある倍数」のおおよその見積りが可能です．詳しくは [10] をご覧ください．少なくともパラメータとして収束判定用の定数を自由に調整できるようなプログラムの作成をお勧めします．

[42] 九州大学に在籍されていた，(故) 占部 実先生が提唱された方法であることから「占部の方法」と呼ばれています．

よる誤差の影響をすべて考慮した上で不動点定理の条件を計算機でチェックすることによって解の存在と (場合によっては一意性を) 数学的に証明しようというものです.

もちろん，すべての問題が不動点定理を満足するとは限りません．また，解の存在のチェックには近似解を求めるよりも多くのコストがかかります．しかし，収束判定基準に自信がなくても，解の存在が誤差評価付きで保証されていれば，もちろん安心して計算を打ち切ることができるでしょう．詳しくは [6], [8], [10] をご覧ください.

また，1 次元では $f(x) = 0$ を満たす x が欲しいわけですので，得られた近似解 $\hat{x}$ よりほんの少し右とほんの少し左で (数直線で考えています) 関数値を計算し，符合が逆転していることを調べることで誤差評価が可能です[43].

5 まとめ

最後に，反復計算で得られた結果を論文にまとめたり研究集会で発表するときの「マナー」を箇条書きにします.

◇収束判定基準は何か

残差基準か誤差基準か，その他の基準であるかをはっきり書きます.

◇その基準を選択した理由

「収束証明の条件となるから」「経験的によい精度を得ているから」「この本に書いてあるから」など，何か考えておきましょう.

◇ノルムの種類

1 ノルム，2 ノルム，最大ノルム，その他のノルム，要素ごとに調べた，など．1 ノルムと 2 ノルムが次元に依存することに注意しましょう.

◇収束判定用の定数の値とその根拠

「いろいろ調べてここが限界でした」「マシン・イプシロンで大丈夫でした．それ以下は意味がないので設定していません」など，すぐ何か返答できるようにしておきましょう．残差基準では理論的な見積りが可能な場合があります．また連立 1 次方程式では理論的な誤差限界がわかることもあります．それぞれの問題に応じて調べておきましょう.

◇計算精度

最近の大規模な数値計算においては，単精度での計算結果は受け入れられなくなってきています．メモリーを倍にするために桁数を半分にするよりは，計算サイズは小さくても十分な桁数を確保することをお勧めします．なぜなら「正しくない計算は無意味」だからです[44].

◇計算機の名前，浮動小数点の形式

計算機名は必ず書くようにしましょう．きわどい収束状況を見せる場合は浮動小数点の形式 (何進・仮数部何桁・指数部何桁) も説明すると親切です．速度を測る場合にはコンパイラのバージョンや計算機の性能データも必要です.

◇著作権に配慮する

既存のプログラムを用いた場合は名前，文献などを書くようにしましょう.

[43] 関数 f が連続であるという仮定つきです．厳密な評価のためには，浮動小数点演算の誤差を見積もることが必要です．しかし，ある程度の信頼性は得られます.

[44] 以前「実験から得られる入力データは所詮 3 桁程度しか合ってないから，計算も単精度で十分だ．倍精度なんてメモリーの無駄使いだ．」という意見を聞いたことがあります．これは，数値シミュレーションにおける実験誤差と数値計算の誤差のレベルを一緒にした誤解です．数値計算結果が無意味になってしまえば，実験データそのものが無意味になってしまいます.

参考文献に簡単な説明と 1999 年 1 月現在の価格 (税別) を追加しています．専門書は品切れになってもなかなか増刷してくれません．古典的な名著といわれながら増刷，復刻されない本もたくさんあります．「この本が手元に欲しいなぁ」と思ったら，すぐに注文することをお勧めします．

参考文献

[1] Richard Barrett, Michael Berry, Tony F. Chan, James Demmel, June Donato, Jack Dongarra, Victor Eijkhout, Roldan Pozo, Charles Romine, and Henk Van der Vorst: *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*, SIAM, Philadelphia, PA, 1994.

《URL》`http://www.netlib.org/templates/Templates.html`

《邦訳》長谷川 里美, 長谷川 英彦, 藤野 清次 訳:『反復法 Templates』, 応用数値計算ライブラリ, 朝倉書店, ISBN 4-254-11401-X, 1996.

大規模疎行列に対する連立 1 次方程式の反復解法のアルゴリズムをきれいにまとめている本です．反復の停止条件も詳しく書かれています．上記 URL から PostScript ファイルが入手できます．また， FORTRAN, MATLAB のソースプログラムも入手することができます．邦訳は第 1 版の訳本です． 4,600 円 (訳本).

[2] 藤野 清次, 張 紹良:『反復法の数理』, 応用数値計算ライブラリ, 朝倉書店, ISBN 4-254-11404-4, 1996.

連立 1 次方程式の反復解法についての最新のアルゴリズム，ベクトル計算機における具体的なチューニング手法などが紹介されています．数値計算の原著論文，経歴，写真をまとめた第一章は貴重な資料です． 4,200 円.

[3] 一松 信:『数値解析』, 新数学講座 13, 朝倉書店, ISBN 4-254-11443-5, 1982.

やや理論的に踏み込んだ数値解析学の入門書です．内容もさることながら，切れ味鋭い小さい活字の部分が大変参考になります． 3,000 円.

[4] 川上 一郎:『数値計算』, 理工系の数学入門コース 8, 岩波書店, ISBN 4-00-007778-3, 1989.

数値計算の基礎をわかりやすく解説しています．特に計算機内の数値表現の説明が参考になります．具体的な Fortran プログラムも添付されています． 2,330 円.

[5] 森 正武:『FORTRAN 77 数値計算プログラミング (増補版)』, 岩波コンピュータサイエンス, 岩波書店, ISBN 4-00-007684-1, 1987.

FORTRAN 77 を用いた数値計算を行なう人は必携の本です． IEEE 規格と Fortran 95 に対応した新版を期待しています． 3,200 円.

[6] 中尾 充宏, 山本 野人:『精度保証付き数値計算〜コンピュータによる無限への挑戦〜』, チュートリアル: 応用数理の最前線, 日本評論社, ISBN 4-535-78258-X, 1998.

有限次元および無限次元の問題の解の存在，一意性と誤差限界を数学的に保証する数値計算法 (精度保証付き数値計算) を解説した日本初の本です．実践的なアルゴリズムも付いています．章の途中で文体が急に変わる妙も楽しめます． 3,000 円.

[7] 仁木 滉, 河野 敏行:『楽しい反復法』, 共立出版, ISBN 4-320-01582-7, 1998.

見事なタイトルです． Gauss-Seidel 法， SOR 法のアルゴリズム，収束条件が詳しく説明されています．コーヒーブレイクに登場する「H 教授」の博識には恐れ入ります． 2,000 円.

[8] 大石 進一:『精度保証数値計算』, 応用数理, Vol.8, No.4 (1998) pp.42–54.

数値計算の特集記事の一つです． IEEE 標準 754 の浮動小数点の丸めの特長と OS が提供する丸めモード変換命令を用いた実用的な精度保証付き数値計算を解説しています.

[9] 小澤 一文:『数値計算法 第 2 版』, 情報処理入門シリーズ 4, 共立出版, ISBN 4-320-02804-X, 1996.

B5 版の大きさです． Fortran 90 のプログラムと図が豊富に入っています． 2,300 円.

[10] 杉原 正顯, 室田 一雄:『数値計算法の数理』, 岩波書店, ISBN 4-00-005518-6, 1994.

数値計算法の数理的・数学的基礎を解説しています．解答付きの豊富な演習問題も用意されています．数値解析を生業にしようと思う方ならば買って損はありません． 1998 年に加筆・修正版の第 2 刷が出版されました． 6,600 円.

[11] 杉浦 洋:『数値計算の基礎と応用 – 数値解析学への入門 –』, 新 情報教育ライブラリ M-11, サイエンス社, ISBN 4-7819-0856-X, 1997.

たいへん読みやすい本です．線形変換の誤差解析が詳しく書かれています．実用的なアルゴリズムも豊富に記述されています． 1,800 円.

[12] 洲之内 治男:『数値計算』, サイエンスライブラリ理工系の数学 =15, サイエンス社, ISBN 4-7819-0137-9, 1978.

半年程度の講義にあわせた必須の内容を重点的にまとめてあります．理論面・実用面ともバランスのとれた教科書です．Fortran による別冊の演習本もあります． 1,600 円.

[13] 戸川 隼人:『計算機のための誤差解析の基礎』, Information & Computiong-37, サイエンス社, ISBN 4-7819-0550-1, 1974.

丸め誤差の累積，近似計算の誤差などをわかりやすく解説した本です．以前はサイエンスライブラリ情報電算機の 1 冊でした． 1989 年の第 4 刷からソフトカバーに変更になっています．現在，残念ながら品切れ，重版未定です．書店で見かけたら買っておきましょう.

[14] 戸川 隼人:『新装版 UNIX ワークステーションによる科学技術計算ハンドブック［基礎篇 C 言語版］』, サイエンス社, ISBN 4-7819-0868-3, 1998.

1992 年に出版されたものを定価を大幅に下げた新装版としてまとめたものです．ハンドブックの 560 ページの原稿と 130 本の C プログラムを一人で，しかも，半年で書きあげられたそうです．文章は明解かつ含蓄に富んでいます． C を用いて数値計算を行なう人は必携の本です． 3,800 円.

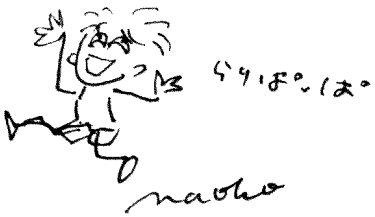
[15] 渡部 善隆:『連立 1 次方程式の基礎知識〜および Gauss の消去法の安定性について〜』, 九州大学大型計算機センター広報, Vol.28, No.4 (1995), pp.291–349.

《URL》`http://www.cc.kyushu-u.ac.jp/RD/watanabe/RESERCH/education.html`

むやみに多い連立 1 次方程式の数値解法を分類してみた記事です．上記 URL 経由で原稿の PostScript ファイルを入手できます.

[16] 山本 哲朗: 数値解析入門, サイエンスライブラリ現代数学への入門 =14, サイエンス社, ISBN 4-7819-0155-7, 1976.

タイトルの通り，数値解析学の入門書としてお勧めです．姉妹篇に『数値解析演習』があります． 1,800 円.



論文完成之図
illustration: H^2 Naoko