第 1 章

数値計算の基礎

本章では，導入としていくつかの簡単な例をとおして，数値計算とはどのようなものか，そして数値計算を行う上でどのような点に注意すべきかを述べることにする．

1.1 アルゴリズム

数学的には同じ答えが得られる計算であっても計算の方法を工夫することにより計算量を減らすことができることがある．例を 2 つほどあげよう．

例題 x^{32} の計算

ふつうに計算すれば,

$$x \times x \times \cdots \times x \tag{1.1}$$

というように x を 31 回掛け算することになる．しかし,

$$a = x^2 \tag{1.2}$$

とおき，同様に

$$\begin{aligned} b &= a^2 (= x^4) \\ c &= b^2 (= x^8) \\ d &= c^2 (= x^{16}) \\ e &= d^2 (= x^{32}) \end{aligned} \tag{1.3}$$

と計算すれば掛け算は 5 回ですむ.

例題 多項式の計算

$$y_4 = a_0 x^4 + a_1 x^3 + a_2 x^2 + a_3 x + a_4 \tag{1.4}$$

の右辺を計算することを考える．このまま計算すると，第 1 項に対しては x^4 の計算に 3 回の掛け算が必要であり，それに a_0 を掛けるので合計 4 回の掛け算が必要である．同様に 2，3，4 項の計算にはそれぞれ 3，2，1 回の掛け算が必要なので全体では掛け算は

$$4 + 3 + 2 + 1 = 10 \text{ 回} \tag{1.5}$$

必要になる．また足し算は 4 回となる．ただし，x^4 を計算する場合には，すでに x^2 と x^3 の計算は済んでいるのでそれを利用することにすれば，掛け算の回数は

$$4 + 1 + 1 + 1 = 7 \text{ 回} \tag{1.6}$$

に減る.

一方，上式は

$$\begin{aligned} y_1 &= a_0 x + a_1 \\ y_2 &= y_1 x + a_2 \\ y_3 &= y_2 x + a_3 \\ y_4 &= y_3 x + a_4 \end{aligned} \tag{1.7}$$

という計算に分解できる．このことは上から順に代入することにより確かめられる．ここで，それぞれの式では 1 回の掛け算と 1 回の足し算を行っているため，合計 4 回の掛け算と 4 回の足し算で計算できる．

これらの例ではある数値を計算するために 2 つの計算法を比較した．一般に，目的となる数値を得るために行う一連の計算手順をアルゴリズムとよんでいるが，上の例のように，同一の結果を得る**アルゴリズム**はひとつではない．計算量の観点からいえば，上の 2 つの例ではあとに述べたものの方が優れているといえる．

1.2　漸化式と反復法

前節の 2 番目の例を一般化して，n 次多項式

$$y_n = a_0x^n + a_1x^{n-1} + a_2x^{n-2} + \cdots + a_{n-1}x + a_n \tag{1.8}$$

の値を求める問題を考える．この場合も

$$\begin{aligned} y_1 &= a_0x + a_1 \\ y_2 &= y_1x + a_2 \\ &\vdots \\ y_n &= y_{n-1}x + a_n \end{aligned} \tag{1.9}$$

とおき，上から順に $y_1, y_2, \cdots, y_n$ を計算していけばよい．この手続きは，以下のようにまとめられる：

$$\begin{aligned} y_0 =& a_0 \text{とおく} \\ i =& 1, 2, \cdots, n \text{ の順に次式を計算する：} \\ & y_i = y_{i-1}x + a_i \end{aligned} \tag{1.10}$$

これが多項式の値を求めるひとつのアルゴリズムである．

y_i を数列と考えたとき，式 (1.10) のように数列の近接の項間に関係式が与えられた場合，その関係式を漸化式という．**漸化式**は数値計算ではいたるところに現れる．

漸化式の応用例として，2 次方程式

$$x^2 - x - 1 = 0 \tag{1.11}$$

を考える．この方程式は

$$x = 1 + \frac{1}{x} \tag{1.12}$$

と変形できる．そこで，この式から漸化式

$$x_{i+1} = 1 + \frac{1}{x_i} \tag{1.13}$$

をつくってみよう．そして，$x_0 = 1$ からはじめて，$x_0, x_1, x_2, \cdots$ を計算すると

$$1, 2, 1.5, 1.6667, 1.6250, 1.6154, 1.6190, 1.6176, \cdots \tag{1.14}$$

となる．

この数列から，上の漸化式の値はある一定の数に近づくことが予想できる．それでは，どのような数に近づくのであろうか．答えはもとの 2 次方程式のひとつの根

$$\alpha = \frac{1+\sqrt{5}}{2} = 1.6181\cdots \tag{1.15}$$

である．なぜなら，一定値 α に落ち着いたとすれば，漸化式の右辺の x_i も左辺の x_{i+1} もともに α となるため，α は方程式

$$\alpha = 1 + \frac{1}{\alpha} \tag{1.16}$$

を満足するからである．逆にいえば，漸化式 (1.13) は 2 次方程式 (1.11) の根を求めるひとつの方法になっている．このように漸化式を利用して方程式の根を求める方法を反復法とよぶ．また**反復法**に利用される漸化式を特に**反復式**とよんでいる．

方程式 (1.11) を解く反復式は一通りではない．たとえば，式 (1.11) から

$$x = \sqrt{x+1} \tag{1.17}$$

という式も得られ，少し変わったものとしては

$$x = \frac{x^2+1}{2x-1} \tag{1.18}$$

という式にも変形できる．なぜ後者の式を選んだかは次章で明らかになる．そこで，これらの式からそれぞれ次の反復式

$$x_{i+1} = \sqrt{x_i+1} \tag{1.19}$$

$$x_{i+1} = \frac{x_i^2+1}{2x_i-1} \tag{1.20}$$

が得られる．共に $x_0 = 1$ からはじめて順次計算を進めれば，式 (1.19) では

$$1, 1.4142, 1.5538, 1.5981, 1.6118, 1.6161, 1.6174, \cdots \tag{1.21}$$

となり，式 (1.20) では

$$1, 2, 1.6667, 1.6190, 1.6180, \cdots \tag{1.22}$$

となる．式 (1.19) では数列は単調増加しながら正解に近づく．一方，式 (1.20) では他の 2 つの反復式より速く正解に近づいていることがわかる．

1.3 誤差

コンピュータでは最終的には電圧の高低でふたつの状態を区別する．そこで例えば電圧の高い場合を 1，低い場合を 0 とすれば，内部の状態は 2 進数で表されることになる．したがって，数値も最終的には 2 進数で表現される．その場合，コンピュータは無限桁の計算ができるわけではないので，数値は 16 桁とか 32 桁といった有限の桁数で表される．一方，実数を小数で表したとき無限桁になることがふつうであり，またたとえば 0.1 のように，たとえ 10 進数では有限桁の数であっても 2 進数では無限桁になってしまうこともある．このような場合には，表現しきれない桁に対しては切り捨てや四捨五入が行われる．したがって，コンピュータには必然的に誤差が入ることになる．このように，本来無限桁の数を有限桁で表現するために生じる誤差を**丸め誤差**とよんでいる．

別の種類の誤差もある．このことを理解するために，三角関数や指数関数の値など，本来は四則演算では計算できない値を求めること考えてみよう．実はコンピュータで三角関数や指数関数の値を計算する場合には，これらの関数を四則演算で計算可能な近似式で代用している．具体的には多項式を用いることが多いが，その場合，数学的には無限の項をもった多項式を用いないと正確には一致しない．一方，コンピュータでは無限項の計算はできないため，有限項で打ち切ってしまう．このとき必然的に誤差が生じるが，このような誤差を**打ち切り誤差**とよんでいる．

誤差はコンピュータでは避けられないものであるため，それが計算結果に悪影響を及ぼさないようにアルゴリズムの側で注意する必要がある．以下にアルゴリズムの選択が特に重要な例を 2 つあげることにする．

例題　$x(\sqrt{x^2+1}-x)$ の計算

仮にあるコンピュータの有効数字が 8 桁であったとしよう．$x=10^4$ のときの関数値を計算してみよう（正確な値は $0.49999999875\cdots$ である）．このとき根号内は正確には 10^8+1 となるが，有効数字が 8 桁なので 1 は無視され，10^8 とみなされてしまう．このように大きさが極端に違う 2 数の加減を行うとき，小さな数が無視される現象を**情報落ち**という．したがって，計算結果は

$$10^4(\sqrt{10^8+1}-10^4)=10^4(\sqrt{10^8}-10^4)=10^4(10^4-10^4)=0 \qquad (1.23)$$

となり，正解からは大きくはずれてしまう．実はこの場合の根号内の 1 は大切な情報を含んでいたことになる．

次にもとの式を次のように変形してみよう．

$$\begin{aligned} x(\sqrt{x^2+1}-x) =& x\frac{(\sqrt{x^2+1}-x)(\sqrt{x^2+1}+x)}{\sqrt{x^2+1}+x} \\ =& \frac{x}{\sqrt{x^2+1}+x} \end{aligned} \tag{1.24}$$

この式に $x = 10^4$ を代入して情報落ちを考慮に入れて計算すれば

$$\frac{10^4}{\sqrt{10^8+1}+10^4} = \frac{10^4}{10^4+10^4} = 0.50000000 \tag{1.25}$$

となり，正解に近い数値が得られることがわかる．

有効数字がもっと多いコンピュータを用いて情報落ちが防げたとしよう．しかし，この場合でも例題の式をそのままの形で計算することはあまりよい方法とはいえない．その理由は以下のとおりである．すなわち，$\sqrt{10^8+1} = 10000.00005$ であるが，そこに現れる 0 を含めた各数字は有効数字であり重要な意味をもつ．このとき $\sqrt{10^8+1} - 10^4$ を計算すれば 0.00005 となるが，ほぼ同じ大きさをもつ 2 つの数の差をとったため 5 より左の有効数字が失われてしまうからである．同じようなことが，8 個の有効数字をもつ 2 つの数の引き算

$$0.12345687 - 0.12345678 \tag{1.26}$$

についてもいえる．このとき計算結果の有効数字は 1 になる．このようにほぼ等しい 2 つの数の差を計算したとき，有効数字の殆どが失われる現象を桁落ちとよんでいる．**桁落ち**は数値計算でもっとも注意しなければならない現象のひとつである．

例題 係数の絶対値が極端に異なる 2 次方程式

2 次方程式

$$ax^2 + bx + c = 0 \tag{1.27}$$

の根は，ふつう根の公式

$$x = \frac{-b \pm \sqrt{b^2-4ac}}{2a} \tag{1.28}$$

で求める．しかし，b^2 が $4ac$ よりずっと大きい場合には問題がおきる．

なぜなら，そのようなときには

$$\sqrt{b^2-4ac} \sim |b| \tag{1.29}$$

であるため，上式の分子の計算において，+ または − の計算のどちらかで桁落ちがおこるからである．この場合，桁落ちを防ぐには以下のようにすればよい．まず，$b > 0$ のときは

$$x_1 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \tag{1.30}$$

に対しては桁落ちは起こらないため，この式を用いてひとつの根を求める．もうひとつの根は，公式を用いずに根と係数の関係

$$x_1 x_2 = \frac{c}{a} \tag{1.31}$$

から求れば桁落ちは起こらない．$b < 0$ の場合も同様にして，ひとつの根を

$$x_2 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \tag{1.32}$$

から求め，もうひとつの根を根と係数の関係から求めればよい．

上記の例題にそった考え方で，2 次元方程式の根を求めるプログラムを以下のプログラム 1 に示す．

プログラム 1　2 次方程式

```
Option Explicit

Private Sub CMD_Calculation_Click()

    Dim A As Double
    Dim B As Double
    Dim C As Double
    Dim D As Double
    Dim X1 As Double
    Dim X2 As Double
    Dim MSG As String

    A = Range("_A")
    B = Range("_B")
    C = Range("_C")

    Call Calculation(A, B, C, MSG, X1, X2)

    Range("_MSG") = MSG
    Range("_X1") = X1
    Range("_X2") = X2
End Sub
```

```
Sub Calculation(A As Double, _
                B As Double, _
                C As Double, _
                ByRef MSG As String, _
                ByRef X1 As Double, _
                ByRef X2 As Double)

   Dim D  As Double

   If A = 0# Then
       MSG = "２次の係数が０なので再入力して下さい "
       Exit Sub
   End If
   D = B * B - 4# * A * C
   If D < 0 Then
       X1 = -B / (2# * A)
       X2 = Sqr(-D) / (2# * A)
       MSG = "解は共役複素数です。"
   ElseIf D = 0 Then
       X1 = -B / (2# * A)
       MSG = "解は重根です。"
       X2 = 0
   Else
       If (B < 0#) Then
           X1 = (-B + Sqr(D)) / (2# * A)
       Else
           X1 = (-B - Sqr(D)) / (2# * A)
       End If
       X2 = C / (A * X1)
       MSG = "解は２実根です。"
   End If

End Sub
```

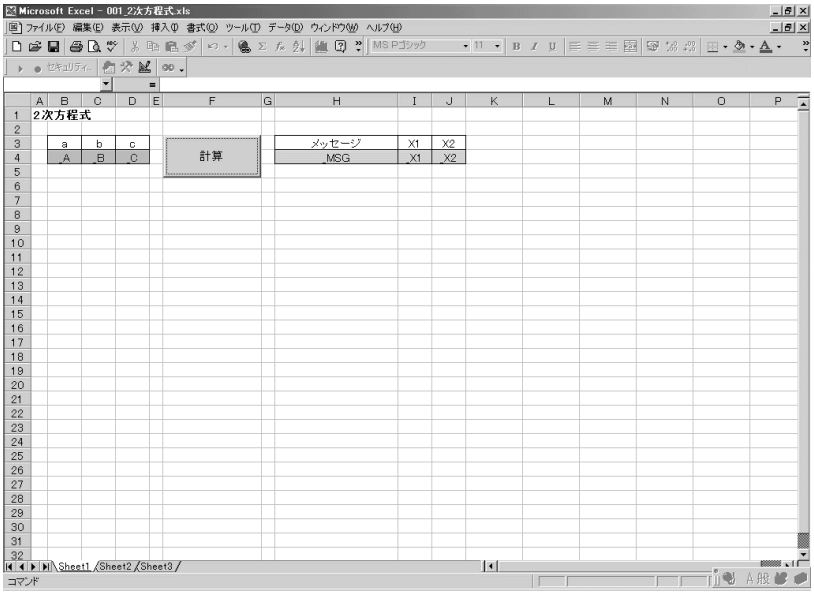

図 1.1　二次方程式のセルの名前

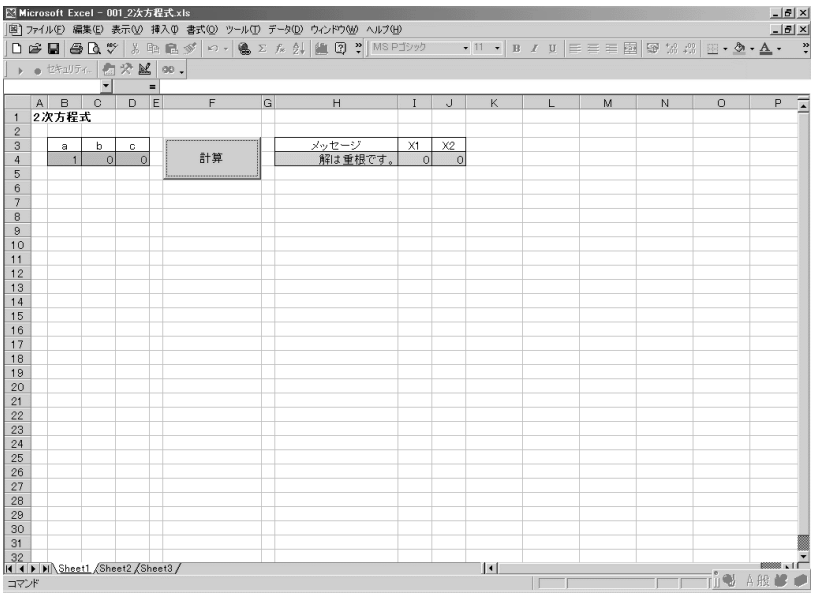

図 1.2　二次方程式の実行結果