情報処理 II No.5

計算機で方程式を解く

桂田 祐史

1995 年 5 月 26 日

1 はじめに – 今週の目標

今回の例題は次のものです。

例題 5-1: 二分法によって、方程式 $\cos x - x = 0$ の解を計算せよ。

例題 5-2: Newton 法によって、方程式 $\cos x - x = 0$ の解を計算せよ。

「方程式を解け」という問題はしょっちゅう現れます。基本的で大事な方程式はその性質を学んできましたが、それ以外にも色々な方程式があることは分かるでしょう。方程式は解が存在しても、紙と鉛筆の計算で具体的に解くのが難しいことがしばしばあります[1]。この例題の方程式もそういうものの一つで、解がただ一つあることは簡単に分かりますが (後の補足を参照)、その解を簡単な式変形等で求めることは出来そうにありません。これに計算機でチャレンジしよう、というのが今週の例題です。

計算機で方程式を扱う場合には、計算機ならではのやり方があります。次週以降の常微分方程式もそうですが、有限回の計算で真の解 (無限精度の解) を求めることをあきらめて、真の解を求めるには無限回の演算が必要だが、有限桁の要求精度を持つ解 (近似解) はそこそこの回数の基本的な演算 (四則や初等関数の計算) で求まるような方法 — **近似解法** — を採用する、というものです。従ってアルゴリズムは、大抵**繰り返し**のあるものになります。というわけで一言、

どんなプログラミング言語であれ、繰り返し計算になれることは大事です。

ここで解説する近似解法は、適用できる範囲はかなり広く、計算機を使って計算することになる人は、今後も何度もお世話になることと思います。

[1] 方程式によっては、人間の手計算では実際的な解法がないものもある、というかそういうものの方が多いわけですが、大学二年次までの段階では、うまく解けるような問題を扱うことの方が多かったので、ピンと来ないかもしれません。

2　方程式の分類

方程式といっても色々なものがありますが、ここでは微分や積分を含まない、有限次元の、「普通の」もの、つまり既知関数

$$f : \mathbf{R}^n \supset \Omega \longrightarrow \mathbf{R}^m$$

を用いて表される、未知数 x についての方程式

$$f(x) = 0$$

について考えます。

2.1　線形方程式 — まあまあ簡単

f が x の一次式である場合、方程式を線型方程式と呼びます。これは、f が適当な行列 $A \in M(m, n, \mathbf{R})$, ベクトル $b \in \mathbf{R}^n$ を用いて $f(x) = Ax - b$ と表されるということで、方程式はいわゆる (連立)1 次方程式です。この場合は有限回の四則演算で解が求まります。A が n 次正則行列であった場合は $x = A^{-1}b$ ですね。計算機における解法も、みなさんも既に何らかの方法[2]を習ったことがあるでしょう。この問題はみかけよりも奥が深く、また非常に応用範囲が広いので、実に精力的に研究されていて、面白い手法も少なくないのですが、この講義では紹介を見送ります。

問題 5.1:　連立 1 次方程式を解くための共役勾配法 (CG method) について調べ、プログラムを書いて実験せよ。(少し手間がかかります。)

2.2　非線形方程式 — なかなか難しい

ここでは非線形方程式について考えましょう。もっとも簡単な非線形方程式は、2 次以上の代数方程式

$$a_n x^n + a_{n-1} x^{n-1} + \cdots + a_2 x^2 + a_1 x + a_0 = 0 \quad (a_n \neq 0)$$

でしょう。「n **次代数方程式は** n **個の根を持つ**」ことは常識ですね。また、次数 n が 2 の場合は「2 次方程式」で根の公式は中学校以来良く知っているはずです。さらに n が 3,4 である場合も、(2 次方程式ほどポピュラーではありませんが) 根の公式[3]があります。ところが、「n **が** 5 **以上の場合は、四則とべき根のみを有限回用いた根の公式は存在しない**」ことが証明されています。

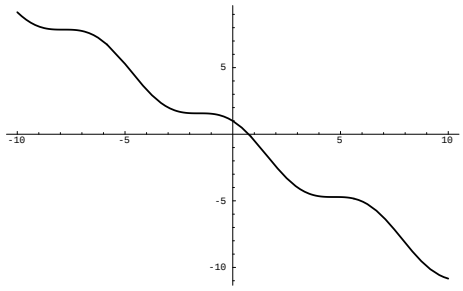
比較的簡単なはずの代数方程式でもこんな調子なのですから、より一般の**非線形方程式を簡単な式変形のみで解くことは運が良くない限り駄目だ**、ということになります。

問題 5.2:　3 次代数方程式を解くための Cardano の方法について調べ、プログラムを書いて実験せよ。

[2]例えば掃き出し法、Gauss の消去法など。
[3]もっとも、とても複雑で、紙と鉛筆で計算するのは (少なくとも私は) うんざりしてしまいます。

3　非線形方程式を計算機で解く

ここでは例題の方程式 $\cos x - x = 0$ について考えましょう。$f(x) = \cos x - x$ とおいて、f を少し調べてみれば (このすぐ後に書いておきます)、この方程式にはただ一つの解があって、それは区間 $(0, 1)$ にあることが分かります。大雑把に言うと、グラフの概形は $y = -x$ をサイン・カーブ風に波打たせたものになっています。この唯一の解を求めることが目標です。



方程式が区間 $(0, 1)$ に unique な解を持つことの証明　まず $f'(x) = -\sin x - 1 \leq 0$ $(x \in \mathbf{R})$ で、特に $x = \frac{\pi}{2} + 2n\pi$ $(n \in \mathbf{Z})$ 以外のところでは $f'(x) < 0$ であるから、f は狭義の単調減少関数である。そして $f(0) = 1 > 0$, $f(1) = \cos 1 - 1 < 0$ ゆえ、方程式 $f(x) = 0$ は区間 $(0, 1)$ 内に少なくとも一つの解を持つが (中間値の定理)、f の単調性からそれは $\mathbf{R}$ 全体でただ一つの解であることが分かる。

3.1　二分法

微積分で基本的な**中間値の定理**を復習してみましょう。

定理 1 (中間値の定理) $f : [\alpha, \beta] \to \mathbf{R}$ を連続関数、$f(\alpha)f(\beta) < 0$ とすると、$f(c) = 0$ となる $c \in (\alpha, \beta)$ が存在する。

(つまり $f(\alpha)f(\beta) < 0$ となる α, β があれば、方程式 $f(x) = 0$ の解 $x = c$ が区間 (α, β) 内に存在する、ということです。)

この定理の証明の仕方は色々ありますが、代表的なものに**区間縮小法**を使ったものがあります。それは以下のような筋書きで進みます。

次の手順で帰納的に数列 $\{a_n\}$, $\{b_n\}$ を定める。

(i) $a_0 = \alpha$, $b_0 = \beta$ とする。

(ii) 第 n 項 a_n, b_n まで定まったとして、$c_n = (a_n + b_n)/2$ とおき、$f(a_n)f(c_n) < 0$ なら $a_{n+1} = a_n$, $b_{n+1} = c_n$, そうでないなら $a_{n+1} = c_n$, $b_{n+1} = b_n$ とする。

すると、

$$a_0 \leq a_1 \leq a_2 \leq \cdots \leq a_n \leq a_{n+1} \leq \cdots, \quad \cdots \leq b_{n+1} \leq b_n \leq \cdots \leq b_2 \leq b_1 \leq b_0$$

$$a_n < b_n \leq b_0 \quad (n \in \mathbf{N}) \quad \text{さらに} \quad a_0 \leq a_n < b_n \quad (n \in \mathbf{N}),$$

$$b_n - a_n = (\beta - \alpha)/2^n \to 0 \quad (\text{as } n \to \infty),$$

$$f(a_n)f(b_n) \leq 0 \quad (n \in \mathbf{N}).$$

これから

$$\lim_{n\to+\infty} a_n = \lim_{n\to+\infty} b_n = c, \quad \alpha < c < \beta$$

と収束して

$$f(c) = 0$$

が成り立つことが分かります。

以上の証明の手続きから、$f(\alpha)f(\beta) < 0$ となる α, β が分かっている場合に、方程式 $f(x) = 0$ の近似解を求めるアルゴリズムが得られます (以下では $\leftarrow$ は変数への代入を表します)。

(0) 目標とする誤差 ε を決める。

(1) $a \leftarrow \alpha$, $b \leftarrow \beta$ とする。

(2) $c \leftarrow (b+a)/2$ として $f(a)f(c) < 0$ ならば $b \leftarrow c$、そうでなければ $a \leftarrow c$ とする

(3) $|b-a| \geq \varepsilon$ ならば (1) に戻る。そうでなければ c を解として出力する。

注意 1 (反復の停止のための ε) 目標とする誤差としては、単精度の場合は解の絶対値の推定値 *(*本当に大雑把な、桁数の目安がつく位のもので構いません*)* の大きさに 10^{-7} をかけた数程度にするのが適当です[4]。それ以上小さく取っても、使用している浮動小数点数の体系の能力を越えてしまうことになるので意味がありません。この問題の場合は解は $(0,1)$ の間の真ん中位にありますから、ざっと 1 程度ということで、$\varepsilon = 10^{-7}$ 位でいいでしょう *(*この数を表すのに、*Fortran* では "`1.0e-7`" と書きます*)*。

3.2 Newton 法

非線形方程式を解くためのもう一つの代表的な方法が **Newton** 法です。

これは f が微分可能な関数で、方程式 $f(x) = 0$ の近似解 x_0 が得られている時、漸化式

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad (n = 0, 1, 2, \cdots)$$

で数列 $\{x_n\}_{n=0,1,2,\cdots}$ を定めると、適当な条件の下で

$$\lim_{n\to+\infty} x_n = x_*$$

と収束し、極限 x_* は方程式の解になっている:

$$f(x_*) = 0$$

ということを利用したもので、実際のアルゴリズムは次のようになります。

(1) 適当な初期値 x_0 を選ぶ。

(2) $x \leftarrow x_0$

(3) $x \leftarrow x - f(x)/f'(x)$ とする。

(4) まだ近似の程度が十分でないと判断されたら (3) に戻る。そうでなければ x を解として出力する。

[4]倍精度の場合には 10^{-15} をかけます。

3.3 二分法 vs. Newton 法

ここで紹介した二つの方法はどちらが優れているのでしょうか？それぞれの長所・短所をあげてみましょう。

二分法 — f が微分可能でなくとも連続でありさえすれば適用できる。しかし f は一変数実数値関数でない場合は適用が難しい (特に実数値であることはほとんど必要であると言ってよい)。$f(\alpha)f(\beta) < 0$ なる α, β が見つかっていれば、確実に解が求まるが、収束はあまり速くない。一回の反復で 2 進法にして 1 桁ずつ精度が改善されていく程度である。

Newton 法 — 適用するには少なくとも f が微分可能である必要がある。微分可能であっても f の実際の計算が難しい場合もあるので、そういう場合も適用困難になる。一方 f は多変数ベクトル値関数でも構わない (それどころか無限次元の方程式にも使うことが出来る)。適切な初期値を探すことは、場合によってはかなり難しい。求める解が重解でない場合には、十分真の解に近い初期値から出発すれば 2 次の収束となり (合っている桁数が反復が一段進むごとに 2 倍になる)、非常に速い。

総合的に見て「まずは Newton 法を使うことを考えよ、それが困難ならば二分法も考えてみよ。」というところでしょうか。

問題 5-3: 色々な方程式を二分法、Newton 法で解いてみなさい。初期値の選び方を変えて、収束するか、しないか試して見なさい。収束の判定条件には注意を払うこと。最終的に得られる精度や、必要な反復の回数はどの程度になるか。

詳しく解説はしませんが、初等関数なども計算機の中では、四則演算などの簡単な演算の組合せで計算されていることが多いです。

問題 5-4: 平方根 $\sqrt{a}$ や立法根 $a^{1/3}$ を方程式の解の形に定式化して、Newton 法で解いて見なさい。その結果を Fortran の組み込み関数 `sqrt` や巾乗演算子[5]で計算した値と比較してみなさい。

特に逆関数の計算にも使えます (y が与えられているとして方程式 $f(x) = y$ を x について解けば、$x = f^{-1}(y)$ が得られるはず)。

問題 5-5: Fortran の組み込み関数 `asin`, `acos`, `atan` は使わずに、arcsin, arccos, arctan を計算するプログラムを作りなさい。

3.4 Newton 法の意味

Newton 法の式の意味を簡単に説明しましょう。微分の定義によると、x が x_* に十分近いところでは、f は「接線の式」で近似されます:

$$f(x) \fallingdotseq f'(x_*)(x - x_*) + f(x_*).$$

今 x_* が $f(x) = 0$ の解に十分近いとすると、$f(x) = 0$ の代わりに

$$f'(x_*)(x - x_*) + f(x_*) = 0$$

[5] `a**b` で `a` の `b` 乗が計算できます。例えば `a**(1.0/3)` で `a` の立法根が計算できます。

を解くことにより、x_* よりも精度の高い近似解が得られると考えるのは自然でしょう。実際に実行すると、まず移項して

$$f'(x_*)(x - x_*) = -f(x_*)$$

両辺に $[f'(x_*)]^{-1}$ をかけて

$$x - x_* = -[f'(x_*)]^{-1} f(x_*),$$

ゆえに

$$x = x_* - [f'(x_*)]^{-1} f(x_*) = x_* - \frac{f(x_*).}{f'(x_*)}.$$

多変数の場合も、$[f'(x_*)]^{-1}$ を Jacobi 行列[6]の逆行列と考えれば、まったく同様に Newton 法が使えます。

問題 5-6: 連立方程式

$$\begin{aligned} x^2 - y^2 + x + 1 &= 0 \\ 2xy + y &= 0 \end{aligned}$$

を Newton 法を用いて解くプログラムを作れ。ヒント:

$$\vec{x} = \begin{pmatrix} x \\ y \end{pmatrix},$$

$$f(\vec{x}) = \begin{pmatrix} x^2 - y^2 + x + 1 \\ 2xy + y \end{pmatrix}$$

とおくと、方程式は $f(\vec{x}) = 0$ と書ける。f の $\vec{x}$ における Jacobi 行列を $f'(\vec{x})$ とすると、Newton 法の式は

$$\vec{x}_{n+1} = \vec{x}_n - [f'(\vec{x}_n)]^{-1} f(\vec{x}_n)$$

となる。初期値 $\vec{x}_0$ を $\begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ -1 \end{pmatrix}$ として実験せよ。

4 例題を解くプログラム例

```
* reidai5-1.f -- 二分法 (bisection method) で方程式 f(x)=0 を解く
      program bisect
      integer i,MAXITR
      parameter (MAXITR = 100)
      real alpha,beta,a,b,c,eps
      real fa,fb,fc,f
      external f
*
      write(*,*) ' α , β , ε ='
      read(*,*) alpha, beta, eps
```

[6]Jacobi 行列については、もうすぐ微分積分学で習います。

```
*
      a = alpha
      b = beta
      fa = f(a)
      fb = f(b)
      if (fa * fb .gt. 0) then
         write(*,*) ' f(a) f(b) > 0 なのであきらめます。'
      else
         do i = 1,MAXITR
            c = (a + b) / 2
            fc = f(c)
            if (fc .eq. 0.0) then
               exit
            else if (fa * fc .le. 0.0) then
*     左側 [a,c] に根がある。
               b = c
               fb = fc
            else
*     左側 [a,c] には根がないかもしれない。[c,b] にあるはず。
               a = c
               fa = fc
            endif
            write(*,2000) a,fa,b,fb
            if ((b - a) .le. eps) then
               exit
            endif
         end do
      endif
      write(*,1000) c,fc
 1000 format(' f(', E16.8,')=',E24.16)
 2000 format(2(' f(', E16.8, ')=', E16.8))
      end
*****************************************************************
      real function f(x)
      real x
      f = cos(x) - x
      end

* reidai5-2.f -- Newton 法で方程式 f(x)=0 を解く
      program Newton
      integer i,MAXITR
      parameter (MAXITR = 100)
      real f,dfdx,x,dx,eps
```

```
      external f,dfdx
*
      write(*,*) ' x0, ε='
      read(*,*) x, eps
*
      do i = 1, MAXITR
         dx = - f(x) / dfdx(x)
         x = x + dx
         write(*,1000) x, f(x)
         if (abs(dx) .le. eps) then
            exit
 endif
      end do
*
      write(*,1000) x,f(x)
 1000 format(' f(', E16.8,')=', E24.16)
* 1000 format(' f(', E24.16,')=', E24.16)
      end
*****************************************************************
      real function f(x)
      real x
      f = cos(x) - x
      end
*****************************************************************
* 関数 f の導関数 (df/dx のつもりで名前をつけた)
      real function dfdx(x)
      real x
      dfdx = - sin(x) - 1.0
      end
```

5 おまけ – 計算機における演算

5.1 Fortran で使える「実数」は浮動小数点数

この授業では、情報処理・演習等で Fortran による初歩のプログラミングを習得した人を対象にしていますが、Fortran では普通の数学に現れる実数を**浮動小数点数**[7](**floating point numbers**, 型名で言うと `real`=`real*4`, `real*8`=`double precision`, `real*16`) というもので表わします。詳しいことは省略しますが、演習に使用している SPARCstation などの内部で

[7]自然科学等で使われる、(例えばアボガドロ数が 6.02×10^{23} というような) 仮数部×指数部という、いわゆる**指数形式の表示**に似た表現法。

は浮動小数点数は有限桁の 2 進数で表現されていることに注意して下さい[8]。

有限桁であることから、当然「有限の精度しか持っていない」ことになります。例えば SPARCstation では、Fortran の単精度実数型は 10 進数に換算して 7 桁程度[9]の精度を持ち、表現できる数 ($\neq 0$) の絶対値の範囲はおおよそ $10^{-38} \sim 10^{38}$ 程です。(倍精度実数型では、それぞれ 16 桁弱、$10^{-308} \sim 10^{308}$ となっています。) 当然 0 も使えますが、かなり特別な扱い方をしています[10]。

有限桁でしか演算が出来ないために生じる誤差を**丸め誤差**(round off error) と呼びますが、真面目な計算には、丸め誤差の発生を小さく押さえるための各種の手法が必要になります。しかし、この種の面倒な話に気を取られると肝心の本質が見えにくくなってしまいます。そこで、この情報処理 II では、丸め誤差についての、その種の細かい話は取り扱わないことにします。

5.2 実数は理想の産物

一つのシステムで使用可能な浮動小数点数全体の集合は、実数全体の集合が持っているようなきれいな性質を持っていません。しかし良く考えてみると、数学で現れる実数というものは相当な理想化がされたものであることが分かります。無限桁の精度を許せば、無限の情報を収めることが可能になります (特に、デジタル情報は、それがどんなに大きくても、有限であれば、一つの実数で表現することが出来てしまいます)。

現実の世界の連続量を測定したデータは有限桁の精度しか持たないことに注意しましょう。浮動小数点数はベストではないかもしれませんが (多分ベストではないでしょうが)、実際的な観点からは十分役に立っています。

[8]現在のパソコンやスーパーコンピューターでも 2 進数が採用されています。一部の大型機では 16 進数が採用されています。

[9]正確には 2 進法で 24 桁。倍精度の方は 2 進法で 53 桁。

[10]実は無限大 $\pm\infty$ も扱えるようになっています。