非線型方程式、特に代数方程式の解法

桂田 祐史

2007年3月24日

目 次

第 1 章　非線型方程式概説 **4**
- 1.1 不動点定理に基づく反復法 . . . 4
- 1.2 Newton 法 . . . 5
  - 1.2.1 基礎事項 . . . 5
  - 1.2.2 Newton 法を用いた初等的な計算 . . . 9
  - 1.2.3 停止則について . . . 11
  - 1.2.4 Kantorovich の定理 . . . 12
  - 1.2.5 減速 Newton 法 . . . 13
  - 1.2.6 実例 . . . 13
- 1.3 連続変形法 . . . 13
- 1.4 その他 . . . 13

第 2 章　代数方程式の解法 **14**
- 2.1 序 . . . 14
  - 2.1.1 落書き . . . 14
  - 2.1.2 1 次元代数方程式の解法概観 . . . 14
  - 2.1.3 連立代数方程式 . . . 17
- 2.2 連立法, 特に Durand-Kerner 法 . . . 18
  - 2.2.1 Durand-Kerner 法 — 名前の由来 . . . 18
  - 2.2.2 DK 法 — Durand の解釈 . . . 19
  - 2.2.3 復習: 根と係数の関係 . . . 20
  - 2.2.4 DK 法 — Kerner の解釈 . . . 21
  - 2.2.5 解の精度の検証 . . . 22
  - 2.2.6 DK 法の長所・短所 . . . 23
  - 2.2.7 初期値の取り方の重要性 . . . 23
  - 2.2.8 Aberth の初期値, DKA 法 . . . 23
  - 2.2.9 Ehrlich-Aberth 法 . . . 24
  - 2.2.10 根の大きさの限界 . . . 25
  - 2.2.11 伊理の初期値 . . . 25

付 録 A おもちゃ箱 (その他の方法) **26**
- A.1 平野法 . . . 26
- A.2 Horner 法 . . . 28
- A.3 Graffe 法 . . . 28
- A.4 Bairstow Hitchcock の方法 . . . 28

A.5 Strum の方法 30
A.5.1 スツルムの定理 30
A.5.2 ユークリッドの互除法による Strum 列の生成 32
A.5.3 3 重対角行列の固有多項式と Strum 列 34
A.5.4 直交多項式の作る Strum 列 36
A.5.5 一般化された Strum 列 38
A.6 Bernoulli 法 39
A.7 商差法 40
A.8 次数低下 40
A.9 Jenkins and Traub 法 41
A.10 その他 41
A.10.1 Euler 法 41
A.10.2 ハレー法 42
A.10.3 ポメンタール法 42

付 録B 応用解析 IV の数値実験 **43**
B.1 6701 号室のワークステーションの利用 43
B.2 応用解析 IV ホームページ 43
B.3 C++ について 44
B.3.1 まずはサンプル・プログラム 44
B.3.2 C++ の (とても簡潔な) 紹介 45
B.3.3 サンプル・プログラムを読むための C++ の知識 45
B.3.4 g++ — 実験に用いる C++ 処理系 46
B.4 グラフィックス・ライブラリィ GLSC について 46
B.5 素朴な Newton 法の実行例 48
B.6 DK 法のサンプル・プログラム 49
B.6.1 ソースプログラム `DK3.C` 49
B.6.2 コンパイル＆実行例 52

付 録C 情報処理 II から **54**
C.1 レポート課題 54
C.2 例題を解くプログラム例 55
C.2.1 Newton 法の場合 55
C.2.2 二分法の場合 55

付 録D 多次元の Newton 法の例 **57**
D.1 ターゲット問題 57
D.2 復習 57
D.3 ターゲット問題の差分近似 58
D.4 Newton 法 58
D.5 サンプル・プログラム 58

メモ

まずは Kantorovich の定理, それから山本 [15] を勉強することだ。

- 2003 年度 Gröbner 基底がらみで面白い本 (齋藤・竹島・平野 [3]) が出た。誰か卒研でやらないかな。
- 連立法が Newton 法であることの証明は、山本 [15] にある。これを読解して解説するようにしたい。
- Kantorovich の定理の証明を消化して書いておこう。
- 篠原 [4] にある占部[1]の定理。

[1] 篠原能材氏の恩師故占部実氏のこと。

第1章　非線型方程式概説

有限次元の線型方程式については、問題の規模がさほど大きくなければ十分に満足できる解法があるが、非線型方程式一般に適用できる解法はない。

- 不動点定理に基づく反復法
- Newton 法

1.1　不動点定理に基づく反復法

与えられた方程式をそれと同値な

$$x = F(x) \tag{1.1}$$

に変換してそれを解くと解釈できる方法が多い。

方程式 (1.1) の解 x のことを F の不動点と呼び、方程式 (1.1) の解の存在を保証する定理を不動点定理と呼ぶ。

ここでは (1.1) のタイプの方程式を不動点型の方程式と呼ぶことにする。

適当に選んだ X の要素 x_0 から漸化式

$$x_{k+1} = F(x_k) \quad (k = 0, 1, 2, \cdots) \tag{1.2}$$

で定めた列 $\{x_k\}k \in \mathbf{N}$ が極限 $x_\infty \in X$ を持つことがある。F が連続であれば x_∞ は F の不動点である (これは (1.2) において $k \to \infty$ の極限を取れば良い)。このことを背景に列 $\{x_k\}_{k\in\mathbf{N}}$ を実際に計算して、十分大きな番号 k に対する x_k を方程式の近似解として採用する方法を反復法と呼ぶ。

定理 1.1.1 (Banach の不動点定理 (縮小写像に関する不動点定理)) (X, d) は完備距離空間で、$F : X \to X$ を縮小写像とする。すなわち

$$d(F(x), F(y)) \leq kd(x, y) \quad (x,\, y \in X)$$

を満たす $k \in [0, 1)$ が存在すると仮定する。このとき F の不動点が一意的に存在し、それは X の任意の要素 x_0 を初期値とした反復法

$$x_{k+1} = F(x_k) \quad (k = 0, 1, 2, \cdots)$$

で生成される列 $\{x_k\}$ の極限として得られる。

証明　有名なので省略する。例えば Schwartz [23] を参照せよ。■

注意 **1.1.1** Lipschitz 定数 < 1 の Lipschitz 条件を満たす写像は縮小写像である。ゆえに f が $\|f'(x)\| \leq \exists L < 1$ を満たすならば縮小写像である。 ■

注意 **1.1.2 (Banach の不動点定理の簡単な拡張)** F そのものでなく、適当な自然数 p に対して F^p が縮小写像になる場合にも、F の不動点は一意的に存在し、それは反復法で得られる。ただし F^p とは

$$F^1 \stackrel{\text{def.}}{=} F, \quad F^{m+1} = F^m \circ F \quad (m \in \mathbf{N})$$

で帰納的に定義される写像である。

定理 **1.1.2 (Brouwer の不動点定理)** D を $\mathbf{R}^n$ の有界閉凸集合とし、$f\colon D \to D$ が連続とするとき、f は少なくとも一つの不動点を持つ。

証明　Zeidler [24], 増田 [10] 等を参照せよ。 ■

1.2 Newton 法

山本哲朗先生の一連の著作、特に山本 [15] を見よ。

1.2.1 基礎事項

X を Banach 空間、U を X の開集合、$f\colon U \to X$ を Fréchet 微分可能な写像とするとき方程式

$$f(x) = 0 \tag{1.3}$$

を考える。適当な $x_0 \in U$ を取って漸化式

$$x_{k+1} = x_k - (f'(x_k))^{-1} f(x_k) \tag{1.4}$$

で列 $\{x_k\}_{k\in\mathbf{N}}$ を生成したとき、これが極限 x_∞ を持つことがある。このとき x_∞ は (1.3) の解となる。

以下 $(f'(x_k))^{-1}$ のことを $f'(x_k)^{-1}$ と書く。

そこで十分大きな番号 k を取って、x_k を (1.4) の近似解として採用する方法が考えられるが、それを **Newton 法**と呼ぶ。形式的には

$$F(x) := x - f'(x)^{-1} f(x)$$

で定められる F の不動点を反復法で求めていることになる[1]。

[1] F の定義域を考えると、Banach の不動点定理をすぐに適用できるような状況にはないことが分かる。しかし次のことに注意しよう。簡単のため、1 次元とする。$F'(x) = f(x)f''(x)/f'(x)^2$ なので、解の十分近くでは $|F'(x)| \leq \exists L < 1$. これから解を中心とする十分小さな閉球を X とすると、$F|_X$ は縮小写像である。

注意 **1.2.1 (Newton 法という名前について)** Newton が 3 次方程式の近似解法に利用したことに基づくそうだが、慣用の形に一般化したのは Raphson なので、Newton-Raphson (ラフソン) 法と呼ばれることが多い。■

注意 **1.2.2 (準 Newton 法)** f の値の計算コストが高い場合は、(1.4) でなく

$$x_{k+1} = x_k - f'(x_0)^{-1} f(x_k)$$

で列 $\{x_k\}_{k\in\mathbf{N}}$ を生成することがある。この方法を準 **Newton 法**, 簡易 Newton 法, von Mises (フォン・ミーゼ) 法、あるいは修正 Newton 法などと呼ぶ。これは線形収束となる。■

定義 **1.2.1** (p 次収束) $p \geq 1$ に対して、$\{x^{(k)}\}_{k\in\mathbf{N}}$ が a に p 次収束するとは、正定数 L が存在して ($p = 1$ のときは $L < 1$ を要請する)、十分大きい任意の k に対して

$$\|x^{(k+1)} - a\| \leq L\|x^{(k)} - a\|^p$$

が成り立つことをいう。特に $p = 1$ のとき、線形収束ともいい、L を収束率と呼ぶ。なお、

$$\lim_{k\to\infty} \frac{\left\|x^{(k+1)} - a\right\|}{\|x^{(k)} - a\|} = 0$$

のとき、超 1 次収束 (superlinear convergence) するという。

線形収束の場合、

$$\|x^{(k)} - a\| \leq L^k \|x^{(0)} - a\|$$

が成り立つ。つまり等比数列的 (あるいは指数関数的) に収束するわけである。

定理 **1.2.1 (Newton)** Ω を $\mathbf{R}^n$ の開集合、$f\colon \Omega \to \mathbf{R}^n$ を C^2-級の写像、$a \in \Omega$, そして $f(a) = 0$, $f'(a)$ が正則とするとき、a に十分近い x_0 から始めて、Newton 法で作った列 $\{x_k\}_{k\in\mathbf{N}}$ は a に 2 乗収束する。すなわち正定数 C が存在して

$$\|a - x_{k+1}\| \leq C\|a - x_k\|^2 \quad (k \in \mathbf{N}).$$

証明　(概略) f は C^2-級で、$f(a) = 0$ と仮定すると、十分 a に近い x_k に対して

$$\begin{aligned}
x_{k+1} - a &= \left[x_k - f'(x_k)^{-1} f(x_k)\right] - a \\
&= x_k - a - f'(x_k)^{-1} f(x_k) \\
&= x_k - a - f'(x_k)^{-1} \left(f(x_k) - f(a)\right) \\
&= x_k - a - f'(x_k)^{-1} \left[f'(x_k)(x_k - a) - O\left((a - x_k)^2\right)\right] \\
&= O\left((a - x_k)^2\right).
\end{aligned}$$

詳しくは例えば山本 [15] を見よ。■

重根の場合は収束の速さが線形収束になり (つまり「収束が遅い」)、実際の数値計算で得られる最終的な精度も低い。

注意 **1.2.3** 上の定理では舞台を $\mathbf{R}^n$ としたが、Banach 空間 X でも大丈夫である。それには Banach 空間 X の開集合 Ω で定義された写像 $f\colon \Omega \to X$ の 2 回微分可能性を定義する必要があるが、f が x^* で Fréchet 微分可能で、線型写像

$$T\colon X \to \mathcal{L}(X,X) := \{\varphi; \varphi\colon X \to X \quad \text{連続線形}\}$$

が存在して、

$$f(x) = f(x^*) + f'(x^*)(x-x^*) + T(x-x^*)(x-x^*) + S(x;x^*)$$

によって $S(x;x^*)$ を定義するとき

$$\lim_{x\to x^*} \frac{\|S(x;x^*)\|}{\|x-x^*\|^2} = 0$$

が成り立つとき、f は x^* で 2 回強微分可能、T を 2 階Fréchet 微分という。T は $T\colon X\times X \to X$ という有界双線型写像とも見なせる。— ここの説明は一松によるものだが、Schwartz [23] も見ておこう。 ■

微積分の演習問題から

次の命題は微積分の演習書によく登場する。

命題 **1.2.1 (凸関数の Newton 法)** $f\colon [a,b] \to \mathbf{R}$ が 2 回微分可能で、

$$f''(x) > 0 \quad (x \in [a,b]), \quad f(a)f(b) < 0$$

を満すならば、方程式 $f(x)=0$ の解 s は (a,b) 内にただ一つだけ存在し、

$$x_0 = \begin{cases} b & (f(b) > 0 \text{ のとき}) \\ a & (f(a) > 0 \text{ のとき}) \end{cases}$$

を初期値とする Newton 法で定まる列 $\{x_n\}_{n\in\mathbf{N}}$ の極限となる:

$$\lim_{n\to\infty} x_n = s.$$

証明 (それなりに長い)

1. (存在) $f(s)=0$ となる $s \in (a,b)$ が存在することは中間値の定理から分かる。
2. (一意性)

   $$f(s_1) = f(s_2) = 0, \quad a < s_1 < s_2 < b$$

   を満たす s_1, s_2 が存在したと仮定すると、Rolle の定理から

   $$\exists z \in (s_1, s_2) \quad \text{s.t.} \quad f'(z) = 0.$$

   仮定 $f'' > 0$ から、

   $$f' < 0 \quad ((a,z) \text{ 内で}) \quad \therefore f(a) > f(s_1) = 0,$$
   $$f' > 0 \quad ((z,b) \text{ 内で}) \quad \therefore f(b) > f(s_2) = 0.$$

これは $f(a)f(b) < 0$ に反する。ゆえに $f(s) = 0$ となる s は unique である。

3. (Newton 法の収束) 以下、$f(b) > 0$ の場合に $x_0 = b$ を初期値とする Newton 法が収束することを示す ($f(a) > 0$ の場合も同様にできる)。$f(x) = 0$ の unique な解を s とすると、

$$f(x) > 0 \quad (x \in (s, b]), \tag{1.5}$$

$$f'(x) > 0 \quad (x \in [s, b]) \tag{1.6}$$

が成り立つ。

(1.5) の証明　もしも $f(x^*) \leq 0,\ x^* \in (s, b]$ となる x^* があれば、中間値の定理より $f(x^{**}) = 0,\ x^{**} \in [x^*, b)$ となる x^{**} が存在することになるが、これは $f(x) = 0$ の解の一意性に反する。■

(1.6) の証明　まず $f(s) = 0,\ h > 0$ のとき $f(s + h) > 0$ であるから、

$$f'(s) = \lim_{h \downarrow 0} \frac{f(s + h) - f(s)}{h} \geq 0.$$

ところが、もし $f'(s) = 0$ ならば $f'' > 0$ より、$f'(x) < 0\ (x \in (a, s))$. ゆえに $f(a) > f(s) = 0$. これは矛盾であるから $f'(s) \neq 0$. ゆえに $f'(s) > 0$. 仮定 $f'' > 0$ から (1.6) が導かれる。■

さて、

$$x_0 = b, \quad x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

で数列 $\{x_n\}_{n \in \mathbf{N}}$ を決めようとするわけであるが、次の不等式が成り立つことが分かれば、$\{x_n\}$ は well-defined で、s を下界とする単調減少数列であることが分かる。

$$x \in (s, b] \quad \Longrightarrow \quad s < x' < x, \quad \text{where} \quad x' := x - \frac{f(x)}{f'(x)}. \tag{1.7}$$

(1.7) の証明　既に示したように $f(x) > 0,\ f'(x) > 0$ より

$$\frac{f(x)}{f'(x)} > 0 \quad \therefore \quad x' = x - \frac{f(x)}{f'(x)} < x.$$

さらに

$$\begin{aligned} x' - s &= x - \frac{f(x)}{f'(x)} - \left(s - \frac{f(s)}{f'(x)} \right) \quad (\because f(s) = 0) \\ &= (x - s) - \frac{f(x) - f(s)}{f'(x)} \\ &= \frac{f'(x)(x - s) - (f(x) - f(s))}{f'(x)} \\ &= \frac{f(s) - [f(x) + f'(x)(s - x)]}{f'(x)} \\ &= \frac{f''(x + \theta(s - x))(s - x)^2/2}{f'(x)}, \quad \exists\theta \in (0, 1) \quad (\because \text{Taylor の定理}) \\ &> 0. \blacksquare \end{aligned}$$

ラスト　$\{x_n\}_{n\in\mathbf{N}}$ は s を一つの下界とする単調減少数列であるから、収束する。その極限を x_∞ とすると、

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

で $n \to \infty$ として

$$x_\infty = x_\infty - \frac{f(x_\infty)}{f'(x_\infty)}.$$

これから $f(x_\infty) = 0$. 解の一意性から $s = x_\infty$. すなわち

$$\lim_{n\to\infty} x_n = s. \blacksquare$$

T 先生の講義の思い出　Newton 法の説明をしていただいたときのこと。Newton 法はいつも収束するわけではないと言って、反例の存在を図で示そうとし始めたのだが、なかなかうまく行かない。見ると関数の二階導関数の符号が一定のグラフばかり書いている。微積分の勉強で上の命題を知っていた私は、少しどきどきしながら「変曲点がないと... 」とつぶやいた。無事に T 先生の耳に届き、すぐに了解して反例のグラフを描かれました。めでたし、めでたし。 ■

1.2.2　Newton 法を用いた初等的な計算

例 1.2.1 (平方根) $a > 0$ に対して、$\sqrt{a}$ を $f(x) = x^2 - a = 0$ の根と考えて、$x_0 > 0$ から始めて

$$x_{k+1} = x_k - \frac{x_k^2 - a}{2x_k} \tag{1.8}$$

により列 $\{x_k\}_{k\in\mathbf{N}}$ を生成すると、

$$x_{k+1} - \sqrt{a} = \frac{(x_k - \sqrt{a})^2}{2x_k}.$$

なお (1.8) は、

$$x_{k+1} = \frac{1}{2}\left(x_k + \frac{a}{x_k} \right)$$

のように整理することもできるが、数値計算においては、丸め誤差の観点から不利である。■

例 **1.2.2** (平方根の逆数) $1/\sqrt{a}$ を $f(x) = 1/x^2 - a = 0$ の根と考えて、

$$x_{k+1} = x_k + \frac{(1/x_k)^2 - a}{2/(x_k)^3} = x_k \left(\frac{3}{2} - \frac{a}{2}(x_k)^2 \right)$$

あらかじめ $a/2$ などを計算しておけば、除算なしで、乗算 3 回だけで計算可能である。■

例 **1.2.3** (立方根) $\sqrt[3]{a}$ を $f(x) = x^3 - a = 0$ の根と考えると、

$$x_{k+1} = \frac{2}{3} \left(x_k + \frac{a}{(x_k)^2} \right)$$

となるが、$f(x) = x^2 - \dfrac{a}{x} = 0$ の根と考えると

$$x_{k+1} = x_k \frac{(x_k)^3 + 2a}{2(x_k)^3 + a}$$

となる。 ■

例 **1.2.4** (逆数) $1/a$ を $f(x) = a - 1/x = 0$ の根と見なすと

$$x_{k+1} = x_k(2 - ax_k)$$

となる。■

newton.c

例 1.2.5 (プログラム例)

```
/*
 * newton.c -- Newton 法で方程式 f(x)=0 を解く
 *  コンパイル: cc -o newton newton.c -lm
 *  実行: ./newton
 */

#include <stdio.h>
#include <math.h>

int main()
{
    int i, maxitr = 100;
    double f(double), dfdx(double), x, dx, eps;

    printf(" 初期値 x0, 許容精度ε =");
    scanf("%lf %lf", &x, &eps);

    for (i = 0; i < maxitr; i++) {
        dx = -f(x) / dfdx(x);
        x += dx;
        printf("f(%24.15e)=%9.2e\n", x, f(x));
        if (fabs(dx) <= eps)
            break;
    }
    return 0;
}

double f(double x)
{
    return cos(x) - x;
}

/* 関数 f の導関数 (df/dx のつもりで名前をつけた) */
double dfdx(double x)
{
    return -sin(x) - 1.0;
}
```

1.2.3 停止則について

小さな正の数 ε や δ を与えて

$$|\Delta x_k| < \varepsilon \quad (\text{ただし } \Delta x_k := x_{k+1} - x_k)$$

あるいは

$$|f(x_k)| < \delta$$

のとき、x_k が十分良い近似解となったと判断して反復を停止するという考えで書かれているプログラムが多いが、ε や δ をどう取るべきかまで考えておかないと不十分であろう。

例えば代数方程式

$$\sum_{i=0}^{n} a_i x^i = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 = 0$$

を Newton 法で解く場合には、ε_M を計算機イプシロンとして、

$$|f(x_k)| \leq \left(\sum_{i=0}^{n} |a_i||x_k|^i \right) \varepsilon_M$$

が良いと言われている (杉原・室田 [5] p.64 を見よ)。

1.2.4 Kantorovich の定理

Newton 法の収束域、収束速度を与えるだけでなく、解の存在そのものまで保証できる優れた定理である。

$h < 1/2$ が単根の場合、$h = 1/2$ が重根の場合に相当する。

またこの定理の仮定は、精度保証付きの計算で確認可能な可能性が高いものであることに注意しよう。

定理 1.2.2 (Newton-Kantorovich) D を $\mathbf{R}^n$ の開凸集合とし、$f\colon D \to \mathbf{R}^n$ は微分可能で、$x^{(0)} \in D$ で、次の条件が満たされるとする。

(a) Jacobi 行列 f' は D で Lipschitz 連続、すなわち

$$\exists L > 0 \quad \text{s.t.} \quad \|f'(x) - f'(y)\| \leq L\|x - y\| \quad (x, y \in D)$$

が成り立つ。

(b) $f'(x^{(0)})$ は正則行列で、

$$\|f'(x^{(0)})^{-1}\| \leq a, \quad \|f'(x^{(0)})^{-1} f(x^{(0)})\| \leq b, \quad h := abL \leq \frac{1}{2}.$$

(c) $t^* := (1 - \sqrt{1-2h})/(aL)$, $U := B(x^{(0)}; t^*)$ とするとき、

$$\overline{U} \subset D.$$

このとき次の (1), (2) が成り立つ。

(1) $\overline{U}$ の中に $f(x) = 0$ の解 x^* がただ一つ存在する。

(2) $x^{(0)}$ を初期値とする Newton 法による列 $\{x_k\}_{k \in \mathbf{N}}$ が定義され、$x^{(k)} \in \overline{U}$ であり、

$$\|x^{(k)} - x^*\| \leq \frac{\left(1 - \sqrt{1-2h}\right)^{2^k}}{2^k aL} \quad (k = 0, 1, 2, \cdots).$$

証明　杉原・室田 [5] pp.71–75 を見よ (これは Zeidler [24] の証明を改良したものだとか)。他にも山本 [16] や、Rall [22] に別証があるとか。 ■

この証明を読んでみよう。それから、この定理を利用して精度保証するプログラムを書いてみよう。

1.2.5　減速 Newton 法

杉原・室田 [5] を見よ。

1.2.6　実例

杉原・室田 [5] の例 4.1, 4.2 など。

1.3　連続変形法

杉原・室田 [5] の §4.4 を見よ。

1.4　その他

- 歴史
- Kantorovich の定理の証明
- Aitken 加速, Steffensen 反復

第2章　代数方程式の解法

2.1　序

2.1.1　落書き

この章に書いてあるのは、あくまでも桂田個人の学習ノートであり、書きかけのものも多く、単なる写経も多い。

まじめにやる必要がある場合 (実際に問題を解く、数値解析の講義をする場合) は、一松 [8], 杉原・室田 [5], 伊理・藤野 [2] などを一読すること。

杉原・室田 [5] はきちんとまとまっていて面白い。読み物としては一松 [8] が面白い。山本・北川 [17] は数値例が豊富で楽しい。

一松先生曰く

- 古くて新しい難問である。4000 年以上昔のバビロニアの粘土板にもあるくらい歴史があるが、コンピューターの時代になってかえって難しくなったとも言える (扱える数の精度が低く、範囲が狭い)。
- すべての問題をたった一つの方法で解くのは困難である。すくなくとも「序盤向き」の解法と「終盤向き」の解法を使い分けるのがよい。

  序盤向き　精度は低くても、根の位置をつきとめ、他の根と分離する。
  　二分法 (Strum 列を利用した二分法)、商差法

  終盤向き　大域収束性はなくとも、十分よい近似値から始めれば速く収束する。
  　Newton 法、Bairstow 法、Durand-Kerner 法

伊理・藤野 [2]「数値計算の常識」では

Newton 法だ、と。

杉原・室田 [5]「数値計算法の数理」では

- 平野法、連立法 (Durand-Kerner 法) について論じている。

2.1.2　1 次元代数方程式の解法概観

(今のところ、この項はガラクタ箱になってしまっている。)

2 次方程式

2 次方程式 $ax^2+bx+c=0$ は根の公式

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

がある。高校数学では係数 a, b, c は実数であるが、複素数にしてもこの式のままで大丈夫である (なお、複素数の平方根は、実数の範囲で計算できる — Ahlfors の有名な複素解析の教科書に説明がある)。

この根の公式のまま計算しようとすると、$b^2 \gg 4|ac|$ の場合に桁落ち (cancellation) が起きてしまい、精度が低下してしまう。

例えば $b>0$ の場合、

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

の計算で桁落ちが起こるので、

$$x_1=\frac{-b+\sqrt{b^2-4ac}}{2a}=\frac{(-b+\sqrt{b^2-4ac})(-b-\sqrt{b^2-4ac})}{2a(-b-\sqrt{b^2-4ac})}=\frac{-2c}{b+\sqrt{b^2-4ac}}$$

のように分子の有理化を行うか、桁落ちの起こらない方の根

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

を先に計算してから、根と係数の関係 $x_1x_2=c/a$ から

$$x_2=\frac{c}{ax_1}$$

と計算する (この式の右辺を整理すると上と同じになる)。■

3 次方程式

いわゆる **Cardano** の方法がある[1]。かつてはこの方法は数値計算には向かないと考えられていたが (例えば森口 [14])、最近では見直されているそうである (「特に 1 実根 2 複素根を持つ実 3 次方程式の標準算法として見直されてきている」 — 一松 [8])。

まず

$$x^3+a_2x^2+a_1x+a_0=0$$

を

$$x+\frac{a_2}{3}=X$$

と置き換えて、

$$X^3-3px+2q=0,\quad p=\left(\frac{a_2}{3}\right)^2-\frac{a_1}{3},\quad q=\frac{a_0}{2}-\frac{a_1a_2}{6}+\left(\frac{a_2}{3}\right)^3 \tag{2.1}$$

[1] Cardano (1501–1576). 『偉大な学術すなわり代数学の法則について』の中で $\sqrt{-1}$ という記号を使った。「10 をその積が 40 になる 2 つの数に分解すると $5\pm\sqrt{-15}$ になる」。

に変換する。次に

$$X = u + v, \quad uv = p$$

とすると、(2.1) は

$$u^3 + v^3 = -2q, \quad u^3 v^3 = p^3$$

となる。ゆえに

$$t^2 + 2qt + p^3 = 0 \tag{2.2}$$

を解けば u^3, v^3 を得る。実行すると

$$u^3, v^3 = -q \pm \sqrt{q^2 - p^3}.$$

ここで場合わけをする。

(I) もし $q^2 \geq p^3$ ならば、(2.2) は 2 実根

$$t_1 = -q + \sqrt{q^2 - p^3}, \quad t_2 = -q - \sqrt{q^2 - p^3}, \quad \omega = e^{2\pi i/3} = \frac{-1 + \sqrt{3}i}{2}$$

を持つ。その三乗根 (実数！) から

$$\begin{aligned} x_1 &= {t_1}^{1/3}\omega + {t_2}^{1/3}\omega^2 - \frac{a_2}{3}, \\ x_2 &= {t_1}^{1/3}\omega^2 + {t_2}^{1/3}\omega - \frac{a_2}{3}, \\ x_3 &= {t_1}^{1/3} + {t_2}^{1/3} - \frac{a_2}{3}. \end{aligned}$$

(II) $q^2 < p^3$ ならば、$t^{1/3}$ の計算には、複素数の 3 乗根を要する。注意すべきは ${t_1}^{1/3}{t_2}^{1/3} = uv = p$ であるから、${t_1}^{1/3}$ を求めるごとに対応する ${t_2}^{1/3}$ を

$${t_2}^{1/3} = \frac{p}{{t_1}^{1/3}}$$

で定める。これらを数値計算するには極座標に直して計算すればよい。この場合は、代数方程式は 3 実根を持つが、それは実数の四則と累乗根だけでは求められないことが証明されている (不還元の場合と呼ばれる)。 ■

3 次方程式の数値計算の際の注意事項　一松 [8] の §21 を見よ。なお、渡辺・名取・小国 [18] には Fortran サブルーチンが載っている。 ■

実根のみを持つ実係数代数方程式の解きにくさ

例 2.1.1 (Wilkinson の例)

$$p(x) = \prod_{j=1}^{20}(x - j) = (x-1)(x-2)\cdots(x-19)(x-20) = x^{20} - 210x^{19} + \cdots + 20! = 0$$

を考える。係数が変化したとする。これは $q(x)$ を定まった方程式、ε を小さな正数として

$$p(x) + \varepsilon q(x) = 0$$

を解くことになる。その結果、根 x_0 が $x_0 + \delta$ に変わったとする。

$$0 = p(x_0 + \delta) + \varepsilon q(x_0 + \delta) = p(x_0) + \delta p'(x_0) + \cdots + \varepsilon q(x_0) + \varepsilon\delta q'(x_0) + \cdots .$$

大ざっぱに

$$\delta \fallingdotseq \varepsilon q(x_0)/p'(x_0).$$

$q(x) = x^{19}$ とすると、

$$\delta \fallingdotseq C\varepsilon, \quad C = \pm\frac{x_0^{19}}{(x_0 - 1)!(20 - x_0)!}.$$

C は $x_0 = 16$ に対して最大で、そのとき

$$|C| \fallingdotseq 2.4 \times 10^9.$$

Wilkinson は x^{19} の係数 210 が $210 + \varepsilon$ に変わったとき、$x_0 = 16$ に ε のほぼ 24 億倍の誤差が入ることを確かめている。$\varepsilon = 2^{-23} \fallingdotseq 10^{-7}$ とすると、もはや ε の 2 乗以上の項が無視できない。実際にはこのときには、20 個の実根のうち 10 個が, 5 対の複素根に化けてしまい、その虚数部は最大 2.8 にもなっている。■

2.1.3　連立代数方程式

一松 [9] 第 3 章 6 節「連立代数方程式」の写経。

計算機代数の話題はまだ数多くあるが、近年有名になった「グレブナー基底」について略説する。

多変数多項式に関する連立代数方程式を解く問題は良く現れるが、一般的な解法はほとんどない。理論上ではつぎつぎに消去法によって変数を消去してゆけば解けるはずだが、それを正直に実行すると、たちまち大きな係数を持つ高次方程式が現れて、手におえなくなる。

これまでは与えられた方程式を適当に組み合わせて、因数分解できる形に直す方法が主力だった。しかしこれを下手に実行すると、考察すべき場合の個数が激増し、解をうまくまとめるのが厄介いになる。

変数に対称性がある場合には、それを生かしてうまく解ける場合もあった。ただしそれにこだわるのは得策でなく、対称性を無視して消去法を使うほうがよいことも多い。

この問題は、一般化すると、多項式イデアルに関する所属判定問題になる。イデアルとは、歴史的な由来のある述語だが、いくつかの基底要素 $p_1, \cdots, p_m$ に多項式 $u_1, \cdots, u_m$ をかけて加えて表わされる

$$I = \{u_1 p_1 + \cdots + u_m p_m\}$$

という形の多項式の族と考えてよい。

一変数の場合には、このようなイデアルが、$p_1, \cdots, p_m$ の最大公因子 d の倍式全体として表される (単項イデアル)。この d は「よい基底」である。

多変数の場合には、単項でないイデアルがふつうであり、「よい基底」を求めることが重要である。

連立代数方程式 $p_1 = 0, \cdots, p_m = 0$ を解くには、原理的には $p_1, \cdots, p_m$ の生成するイデアルの任意の基底の共通零点を求めればよい。しかし基底が悪いと、この操作が実行困難である。

イデアルに関する所属判定問題とは、与えられた多項式 f が、そのイデアル I に含まれるかどうかを判定する算法を問う問題である。その応用例は後述する。

もし具体的に $u_1, \cdots, u_m$ を求めて

$$f = u_1 p_1 + \cdots + u_m p_m$$

と書くことができれば f は I に含まれる、他方もし変数 $x_1, \cdots, x_n$ にたいして適当な値 $a_1, \cdots, a_n$ で

$$p_1(a_1, \cdots, a_n) = 0, \cdots, p_m(a_1, \cdots, a_n) = 0, f(a_1, \cdots, a_n) \neq 0$$

である組がみつかれば、f はイデアル I には含まれない。

その昔、何日も所属問題がうまく解けずに苦しみ、試しに数値を代入したら、前期のような組がみつかり、f が I に属さないことがわかった、そして結果的にそれまでの計算はむだだった、という実例がある。現在でもこのような予備テストは重要である。しかしこのような $a_1, \cdots, a_n$ の組がみつからない (いつでも $f(a_1, \cdots, a_n) = 0$ となる) とき、f が I に含まれそうだと予想されても証明にはならない。

この判定には I の基底自体が重要である。じっさいに基底のとり方が悪いと、奇妙な現象が起こる。

例えば変数の個数を 3 とし、x_1, x_2, x_3 を x, y, z と書くことにして、次の基底を考える。

$$p_1 = x^3yz - xz^2, \quad p_2 = xy^2z - xyz, \quad p_3 = x^2y^2 - z^2$$

これにたいして $f = x^2yz - z^3$ の所属問題を考えてみよう。

原則として最も高い次数の項を合わせて消去する。しかし f の主項 x^2yz は、p_1, p_2, p_3 のどれを使っても、そのままでは簡約できない。この f は ${}^x p_2 + z p_3$ と表現できるので、イデアルに属する。このことは x^2y^2z というおなじ項を加減して変形すればわかるが、これを機械的に求めることはできない。

この例は f が悪いというよりも、イデアルの基底が不足しているのである。これにさらに新しい基底を補充して、基底 (base) を固める必要がある。

多項式イデアルの基底を補充して、標準的な基底を整備する理論は、かなり昔から考えられていた。たとえば広中平祐氏が、フィールズ賞の対象となった「特異点の還元」の研究中に、その種の「標準基底」を扱っている。したがって、これを「広中基底」とよぶべきかもしれない。

しかしその具体的な構成算法を示したのはブーフバーガーであり、彼は恩師の名をとって、それをグレブナー基底とよんだ。今日その名で広く使われているので、以下でもそうよんで論じることにする。

それはある意味では、連立一次方程式を解くガウスの消去法の一般化である。対称式については基本対称式が「グレブナー基底」に相当する。一般のグレブナー基底は、これらの概念の一般化に相当する。

2.2 連立法, 特に Durand-Kerner 法

n 次代数方程式を n 個の初期値 $\{x_i^{(0)}\}_{i=1}^n$ から始めて、

$$x_i^{(k+1)} = x_i^{(k)} + \Delta x_i^{(k)} \quad (i = 1, \ldots, n)$$

の形の漸化式で計算していく方法を連立法と呼ぶ。

2 次の連立法 (Durand-Kerner 法)、3 次の連立法 (Ehrlich-Aberth 法) を解説する。

2.2.1 Durand-Kerner 法 — 名前の由来

実際には古くからあるとも言われているが (山本 [15])、フランスの Durand が古く提唱し、ドイツの Kerner が別の立場から解釈を与えた (1966 年) ため、今では Durand-Kerner 法と呼ばれるのが普通である、と一松先生は書いているが、杉原・室田 [5] では「2 次の連立法」と呼んでいる。

2.2.2 DK 法 — Durand の解釈

代数方程式

$$p(x) \equiv x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0 = 0$$

の根を $\xi_1, \cdots, \xi_n$ とすると

$$p(x) = (x - \xi_1) \cdots (x - \xi_n),$$

$$p'(\xi_i) = \prod_{j \neq i} (\xi_i - \xi_j) = (\xi_i - \xi_1) \cdots (\xi_i - \xi_{i-1})(\xi_i - \xi_{i+1}) \cdots (\xi_i - \xi_n). \tag{2.3}$$

Newton 法では現在の近似値 $x^{(k)}$ から次の近似値 $x^{(k+1)}$ を求めるには

$$x^{(k+1)} = x^{(k)} - \frac{p(x^{(k)})}{p'(x^{(k)})} \tag{2.4}$$

とする。ここで導関数の計算をしないで済ませるために、この分母を (2.3) で代用することを考える。つまり $x^{(k)}$ が ξ_i の近似値だと考えて $p'(x^{(k)})$ を $p'(\xi_i)$ で置き換えるわけである。各根 ξ_i に対する近似値 $x_i^{(k)}$ が $i = 1, \cdots, n$ について全部そろっているとして、(2.4) の代わりに

$$x_i^{(k+1)} = x_i^{(k)} - \frac{p(x_i^{(k)})}{\displaystyle\prod_{j \neq i} \left(x_i^{(k)} - x_j^{(k)} \right)}. \tag{2.5}$$

を用いる。

Durand は (2.5) の収束性を示し、最終的に 2 乗収束になることを証明した。 ■

$\xi_1, \cdots, \xi_n$ がすべて相異なり、$x_i^{(k)}$ が ξ_i に十分近いとして、$\varepsilon_i^{(k)} := x_i^{(k)} - \xi_i$ とおけば、

$$\begin{aligned} \varepsilon_i^{(k+1)} &= \varepsilon_i^{(k)} \left[1 - \frac{\prod_{j \neq i}(x_i^{(k)} - \xi_j)}{\prod_{j \neq i}(x_i^{(k)} - x_j^{(k)})} \right] \\ &= \varepsilon_i^{(k)} \left[1 - \prod_{j \neq i} \left(1 + \frac{\varepsilon_j^{(k)}}{x_i^{(k)} - x_j^{(k)}} \right) \right] \end{aligned}$$

2 次の項 $\varepsilon_j^{(k)} \cdot \varepsilon_\ell^{(k)}$ を無視すると

$$\varepsilon_i^{(k)} \fallingdotseq -\varepsilon_i^{(k)} \sum_{j \neq i} \frac{\varepsilon_j^{(k)}}{x_i^{(k)} - x_j^{(k)}}.$$

$$|x_i^{(k)} - x_j^{(k)}| \geq \frac{1}{C} \quad (C \text{ は } i, j, k \text{ によらない})$$

とすれば

$$\left| \varepsilon_i^{(k+1)} | \leq C | \varepsilon_i^{(k)} \cdot \sum_{j \neq i} | \varepsilon_j^{(k)} \right|$$

これは 2 乗収束を意味するが、さらに詳しくみると

定理 2.2.1 (DK 法の誤差)

$$|\text{次の誤差}| \leq C \times |\text{自分自身の誤差}| \times |\text{他の誤差の和}|$$

定理 2.2.2 (重心の不変性) 任意の初期値に対して、(2.5) によって作られる $x_i^{(k+1)}$ $(i = 1, 2, \cdots, n)$ は

$$\sum_{i=1}^{n} x_i^{(k+1)} = -a_{n-1} = \sum_{i=1}^{n} \xi_i \tag{2.6}$$

を満たす。すなわち、近似根 $x_1^{(k)}, \cdots, x_n^{(k)}$ の重心は一定で、根 $\xi_1, \cdots, \xi_n$ の重心に等しい。ゆえに誤差 $\varepsilon_i^{(k+1)} = x_i^{(k+1)} - \xi_i$ の和は 0 である。

$$\sum_{i=1}^{n} \varepsilon_i^{(k+1)} = 0.$$

証明　(そんなに簡単ではない。一松 [8] の §20 や、山本・北川 [17] を参照。) ■

2.2.3　復習: 根と係数の関係

この項の内容は代数学の入門書に解説されていることが多い。色々あるが、例えば山崎 [19] をあげておく。

2 次方程式の場合

$$a_2x^2 + a_1x + a_0 = a_2(x - \xi_1)(x - \xi_2)$$

より

$$-a_2(\xi_1 + \xi_2) = a_1, \quad a_2(\xi_1\xi_2) = a_0.$$

3 次方程式の場合

$$a_3x^3 + a_2x^2 + a_1x + a_0 = a_3(x - \xi_1)(x - \xi_2)(x - \xi_3)$$

より

$$-a_3(\xi_1 + \xi_2 + \xi_3) = a_2, \quad a_3(\xi_2\xi_3 + \xi_3\xi_1 + \xi_1\xi_2) = a_1, \quad -a_3(\xi_1\xi_2\xi_3) = a_0.$$

これを一般化して

$$a_nx^n + a_{n-1}x^{n-1} + \cdots + a_1x + a_0 = a_n(x - \xi_1)(x - \xi_2)\cdots(x - \xi_n)$$

について、

$$(-1)^s a_n \times (\xi_1, \cdots, \xi_n \text{ から } s \text{ 個取って作った積の和}) = a_{n-s} \quad (s = 1, 2, \cdots, n).$$■

定義 2.2.1 (基本対称式) 不定元 $X_1, X_2, \cdots, X_n$ の整式

$$S_s(X_1, X_2, \cdots, X_n) = \sum_{1 \le i_1 < i_2 < \cdots < i_s \le n} X_{i_1} X_{i_2} \cdots X_{i_s} \quad (s = 1, 2, \cdots, n)$$

$(i_1, \cdots, i_s$ は $1, \cdots, n$ から取った長さ s の増加列で ${}_nC_s$ 個ある$)$ を $X_1, X_2, \cdots, X_n$ の s 次基本対称式と呼ぶ。

定理 2.2.3 (根と係数の関係) 可換体 K 上の n 次の整式

$$f(X) = \sum_{j=0}^{n} a_j X^j = a_n X^n + a_{n-1} X^{n-1} + \cdots + a_1 X + a_0 \quad (a_n \neq 0)$$

のすべての根 $\xi_1, \xi_2, \cdots, \xi_n$ について

$$(-1)^s \frac{a_{n-s}}{a_n} = S_s(\xi_1, \xi_2, \cdots, \xi_n).$$

定義 2.2.2 (対称式) 整域 K における不定元 $X_1, X_2, \cdots, X_n$ の整式で、不定元をどのように入れ換えても等しいものがあったとき、その整式を $X_1, X_2, \cdots, X_n$ の対称式であるという。

定理 2.2.4 不定元 $X_1, X_2, \cdots, X_n$ の任意の対称式 f は、ある n 不定元の整式 g によって

$$f = g(S_1, S_2, \cdots, S_n)$$

と表される。

その他、判別式、終結式などが勉強しておくべき内容である。

2.2.4 DK 法 — Kerner の解釈

$X_1, \cdots, X_n$ の s 次基本対称式 (s 個の積全体の和) を $S_s(X_1, \cdots, X_n)$ とおく:

$$S_s(X_1, \cdots, X_n) = \sum_{i_1 < i_2 < \cdots < i_s} X_{i_1} X_{i_1} \cdots X_{i_s} = \sum_{i_1 < i_2 < \cdots < i_s} \prod_{j=1}^{s} X_{i_j}.$$

根と係数の関係から

$$f_s(\xi_1, \cdots, \xi_n) := (-1)^s S_s(\xi_1, \cdots, \xi_n) - a_{n-s} = 0 \quad (s = 1, 2, \cdots, n). \tag{2.7}$$

定理 2.2.5 (Kerner 1966) Durand の反復 (2.5) は、(2.7) を $\xi_1, \cdots, \xi_n$ についての連立方程式とみなして、それを Newton 法で解く反復計算に他ならない。

証明　(書きかけ) 与えられた $z_1, \cdots, z_n$ に対して

$$\prod_{j=1}^{n}(x-z_j) = Q(x) = x^n + b_{n-1}x^{n-1} + \cdots + b_1 x + b_0$$

とおくと、

$$\begin{aligned}\frac{\partial Q}{\partial z_j} &= \left(\frac{\partial b_{n-1}}{\partial z_j}\right)x^{n-1} + \cdots + \left(\frac{\partial b_1}{\partial z_j}\right)x + \left(\frac{\partial b_0}{\partial z_j}\right) \\ &= -(x-z_1)\cdots(x-z_{j-1})(x-z_{j+1})\cdots(x-z_n).\end{aligned}$$

また根と係数の関係

$$b_{n-k} = (-1)^k S_k(z_1, \cdots, z_n)$$

が成り立つ。ここで $x = z_i$ とおくと、$-Q'(z_i)\delta_{ij}$ となるから、

$$\left(-\frac{z_i^{n-1}}{Q'(z_i)}, \cdots, -\frac{z_i}{Q'(z_i)}, -\frac{1}{Q'(z_i)}\right)$$

は $J(z)^{-1} = (\partial b_{n-i}/\partial z_j)^{-1}$ の第 i 行と一致する。ゆえに $J(z)^{-1}f(z)$ の第 i 成分は

$$\begin{aligned}\frac{-z_i^{n-1}}{Q'(z_i)}(b_{n-1}-a_{n-1}) - \cdots - \frac{1}{Q'(z_i)}(b_0-a_0) &= -\frac{z_i^n}{Q'(z_i)}(b_n-a_n) - \frac{z_i^{n-1}}{Q'(z_i)}(b_{n-1}-a_{n-1}) - \frac{1}{Q'(z_i)}(b_0-a_0) \\ &= \frac{-Q(z_i)+P(z_i)}{Q'(z_i)} = \frac{P(z_i)}{Q'(z_i)}.\end{aligned}$$

ゆえに ... の各成分を比較して ... を得る。■

系 **2.2.1** 代数方程式の根がすべて相異なるとき、DK 法は根に十分近いところで 2 次の収束となる。

証明　Newton 法の一般的な性質である。 ■

2.2.5　解の精度の検証

次の定理は Durand-Kerner 法に限らず、連立法一般に利用できる便利な定理である。

定理 **2.2.6 (Smith (1970))** $z_1, \cdots, z_n$ を相異なる複素数とするとき、

$$p(z) = z^n + a_{n-1}z^{n-1} + \cdots + a_1 z + a_0 = 0$$

のすべての根は閉円板

$$\Gamma_\ell = \{z \in \mathbf{C}; |z - z_\ell| \le r_\ell\}, \quad r_\ell = \frac{n|p(z_\ell)|}{\prod_{j\neq\ell}(z_\ell - z_j)} \quad (\ell = 1, 2, \cdots, n)$$

の合併に含まれる。その連結成分の一つが m 個の閉円板からなれば、その中にちょうど m 個の根がある。

証明　山本 [15] に載っていた定理だが証明は書いていない。杉原・室田 [5] ではステートメントも少し詳しく、証明つきである。 ■

2.2.6 DK 法の長所・短所

DK 法の長所

- すべての根を同時に求める反復法。並列演算向き。
- 導関数の計算が不要 (多項式を計算する手続きさえあれば計算できるので便利)。
- 各根の収束の速さがほぼ一様。
- Newton 法と異なり暴走の危険性が少ない。

短所

- 能率が今一つと言われている。ただし、初期値次第とも言える。
- 代数方程式専用であって、他への応用がない。
- 近接根が求めにくい (一般的な困難と言えるが)。
- 一つの根だけが欲しい場合には無駄が大きい。

2.2.7 初期値の取り方の重要性

例 2.2.1 (DK 法が収束しない初期値) $p(x) \in \mathbf{R}[x]$, $x_i^{(0)} \in \mathbf{R}$ $(i = 1, 2, \ldots, n)$ とすると、

$$x_i^{(k)} \in \mathbf{R} \quad (i = 1, 2, \ldots, n,\ k \in \mathbf{N})$$

となるので、虚根には収束し得ないことが分る。実際の数値例については、山本・北川 [17] p.61 を見よ。■

2.2.8 Aberth の初期値, DKA 法

以下に示す連立法の初期値は Aberth (1973) により提唱された。アルゴリズム全体を **DKA** 法と呼ぶ。

$$p(z) = z^n + a_{n-1}z^{n-1} + \cdots + a_0 = 0$$

の根を求める DK 法の初期値 $z_j^{(0)}$ $(j = 1, 2, \cdots, n)$ として、

$$z_j^{(0)} = -\frac{a_{n-1}}{n} + r_0 \exp\left[\sqrt{-1}\left(\frac{2\pi(j-1)}{n} + \frac{\pi}{2n}\right)\right] \quad (j = 1, 2, \cdots, n) \tag{2.8}$$

ただし r_0 は閉円板

$$\overline{B}(-a_{n-1}/n; r_0) := \{z \in \mathbf{C}; |z + a_{n-1}/n| \leq r_0\}$$

がすべての根を含むように選ぶ。例えば

$$r_0 = \frac{|a_{n-1}|}{n} + 1 + \max_{0 \le i \le n-1} |a_i|.$$

$p(z) = 0$ の根は行列

$$\begin{pmatrix} 0 & & & & -a_0 \\ 1 & 0 & & & -a_1 \\ & \ddots & \ddots & & \vdots \\ & & \ddots & 0 & -a_{n-2} \\ & & & 1 & -a_{n-1} \end{pmatrix}$$

の固有値である。Gerschgorin の定理から、それらはすべて

$$|z| \le 1 + \max_{0 \le i \le n-1} |a_i|$$

に含まれる。

また

$$x_j^{(1)} + \frac{a_1}{n} = \left(1 - \frac{1}{n}\right)\left(x_j^{(0)} + \frac{a_1}{n}\right) + O\left(\left(x_j^{(0)} + \frac{a_1}{n}\right)^{-1}\right)$$

が成り立つので、r_0 が十分大きいいとき、$x_j^{(k)}$ $(j = 1, 2, \cdots, n)$ は、円の中心 (根の重心) $-a_1/n$ に向かって「直進」する。このことを「最初は 1 次収束」、「終盤は 2 次収束」と言われることもあるが、中盤がどうなるかは分からず、収束証明になっているわけではない。

2.2.9 Ehrlich-Aberth 法

$$x_i^{(k+1)} = x_i^{(k)} - \frac{p(x_i^{(k)})}{p'(x_i^{(k)}) - p(x_i^{(k)}) \sum_{j \ne i} \dfrac{1}{x_i^{(k)} - x_j^{(k)}}}$$

という漸化式に基づく方法は 3 次収束する。この式の導出は杉原・室田 [5] にある。

2.2.10　根の大きさの限界

定理 2.2.7 (根の大きさの評価)

$$p(z) = \sum_{k=0}^{n} a_k z^k = a_n z^n + a_{n-1} z^{n-1} + \cdots + a_1 z + a_0 \quad (a_n \neq 0)$$

に対して、$K = \{k; 0 \leq k \leq n-1, \quad a_k \neq 0\}$ の要素数を κ とし、$\kappa \geq 1$ と仮定する。このとき、実係数多項式

$$q(x) = |a_n| x^n - \sum_{k=0}^{n-1} |a_k| x^k$$

は unique な正の実根 r をもち、$p(z)$ の任意の根 α に対して

$$|\alpha| \leq r \leq r^* := \max_{k \in K} (\kappa |a_k / a_n|)^{1/(n-k)}$$

が成り立つ。

$q(x) = 0$ の根 r は、$x = r^*$ を初期値とする Newton 法により求められる。
(杉原・室田 [5] を見よ。)

2.2.11　伊理の初期値

$$z_j^{(0)} = -\frac{a_{n-1}}{n a_n} + r_0 \exp\left[\sqrt{-1}\left(\frac{2\pi(j-1)}{n} + \frac{3}{2n}\right)\right] \quad (j = 1, 2, \cdots, n) \tag{2.9}$$

伊理 [1] p.152 による。森口 [14] の解説によると「特に $3/(2n)$ だけ位相をずらせて、実軸や虚軸と特別な関係にならないようにしてあるのが特色である。」とある。

付 録A　おもちゃ箱 (その他の方法)

A.1　平野法

(この節の内容は全面的に杉原・室田 [5] のつまみ食いである。)

平野法は Newton 法を少し修正したものであるが、大域的収束性を保証できるという大きな長所がある。1960 年代後半に

- 平野 菅保, 代数方程式の解法および誤差, 第 8 回プログラミングシンポジウム報告集, 情報処理学会, 1967.
- B.Dejon and K.Nickel, A never failing fast convergent root-finding algorithm, Constructive Aspects of the Fundamental Theorems of Algebra (B.Dejon and P.Henrici, eds.), John Wiley, New York, 1969, pp.1–35.

によって独立に提案された算法である。

アルゴリズムの解説と収束を保証する定理については、

> 室田 一雄, 平野の変形 Newton 法の大域的収束性, 情報処理学会論文誌, Vol.21 (1980), pp.469–474.

で初めて証明された。

$$p(z) = \sum_{k=0}^{n} a_{n-k} z^k$$

$$z^{(\nu+1)} = z^{(\nu)} + \Delta z^{(\nu)} \quad (\nu = 0, 1, \cdots)$$

$\Delta z^{(\nu)}$ は次のように考えて決める。$p(z)$ を $z^{(\nu)}$ の回りで

$$p(z^\nu + \zeta) = \sum_{k=0}^{n} c_{n-k} \zeta^k \tag{A.1}$$

と展開する。もちろん

$$c_{n-k} = \frac{p^{(k)}(z^{(\nu)})}{k!} \quad (k = 0, 1, \cdots, n),$$

特に

$$c_n = p(z^{(\nu)}), \quad c_{n-1} = p'(z^{(\nu)})$$

通常の Newton 方では、$z^{(\nu)}$ が根に十分近いことを前提にして、(A.1) の右辺から $k = 0$ と $k = 1$ に対応する項だけを取り出して、

$$p(z^{(\nu)+\zeta}) \fallingdotseq c_{n-1}\zeta + c_n \tag{A.2}$$

と近似して、これを 0 にする値

$$\zeta = -\frac{c_n}{c_{n-1}} = -\frac{p(z^\nu)}{p'(z^{(\nu)})}$$

を修正量としている。平野法では、

$$p(z^{(\nu)} + \zeta) \fallingdotseq c_{n-k}\zeta^k + c_n \quad (k = 1, 2, \cdots, n)$$

という 2 項近似を考え、これから定まる

$$\zeta_k := \left(-\frac{c_n}{c_{n-k}}\right)^{1/k} \qquad (c_{n-k} = 0 \text{ のときは } \zeta_k = \infty \text{ とおく})$$

を修正量の候補とし、それらの中で絶対値が最小のもの (これを ζ_m と書く) を修正量に採用する。

補題 **A.1.1** ($|\zeta_m|$ が余裕を持って最小ならば関数値は減少する) $0 < \beta < 1$ とする。

$$|\zeta_m| \le \frac{\beta}{1+\beta}|\zeta_k| \quad (1 \le k \le n,\ k \ne m)$$

ならば

$$|p(z^{(\nu)} + \zeta_m)| \le \beta \left|p(z^{(\nu)})\right|$$

である。

証明　杉原・室田 [5] を見よ。■

$z^{(\nu)}$ が根 α の十分良い近似値ならば $m = 1$ となる (すなわち Newton 法と一致)。さらに α が単根であることを仮定すると、補題の条件が成り立つ。

$0 < \beta < 1$, $\lambda > 1$ は減速の仕方を決めるパラメーターである。例えば $\beta = 3/4$, $\lambda = 2$ とする。

平野法の第 ν 段の修正量の決定

(1) c_k $(k = 0, 1, 2, \cdots, n)$ を組み立て除法で計算する。$\mu := 1$ とおく。

(2) $\zeta_k := (-\mu c_n / c_{n-k})^{1/k}$ $(k = 1, 2, \cdots, n)$ とおく。

(3) $|\zeta_k|$ が最小である k $(1 \le k \le n)$ を m とおく。

(4) $\left|p(z^{(\nu)+\zeta_m})\right| \le (1 - (1-\beta)\mu)|c_n|$ ならば、$\Delta z^{(\nu)} := \zeta_m$ として第 ν 段を終了し、そうでなければ $\mu := \mu/\lambda$ として、(2) に戻る。

(2) における k 乗根の分枝は、$|z^{(\nu)} + \zeta_k|$ が最も小さくなるなものを選ぶ。具体的には ϕ, $\psi_k \in [0, 1)$ を

$$\phi = \frac{\arg z^{(\nu)}}{2\pi}, \quad \psi_k = \frac{\arg(-c_n/c_{n-k})}{2\pi},$$

で定めて、$k(1/2 - \phi) - \psi_k$ に最も近い整数 j_k を求め、

$$\zeta_k := \left|\mu\frac{c_n}{c_{n-k}}\right|^{1/k} \exp\left[\frac{2\pi i(\psi_k + j_k)}{k}\right] \tag{A.3}$$

とする。

定理 A.1.1 (室田 1980 — 平野法の大域的収束性) 任意の初期値 $z^{(0)} \in \mathbf{C}$ に対して、平野法の近似列 $\{z^{(\nu)}\}$ は、

$$\left|p(z^{(\nu+1)})\right| \leq [1-(1-\beta)\theta]\left|p(z^{(\nu)})\right|$$

を満たす。ここで、

$$\theta = \left(\frac{\beta}{1+\beta}\right)^{2n^3} \lambda^{-n}.$$

ゆえに $\{z^{(\nu)}\}$ は $p(z)$ のある根に収束する。

証明　元々は室田 [11] による。杉原・室田 [5] 第 5 章を見よ。■

A.2 Horner 法

古くは天元術と言われたとか。一松 [8] にちょこっと説明があるが...。

A.3 Graffe 法

根の自乗を根とする方程式を次々に作る。URR のように広い範囲を数値を表現できるようなシステムでは有効性が高いかもしれないという指摘がある。

A.4 Bairstow Hitchcock の方法

n 次代数方程式

$$p_n(x) = a_0x^n + a_1x^{n-1} + \cdots + a_n, \quad a_0 \neq 0, \quad n \geq 2$$

が与えられたとき、2 次の因数 $x^2 - ux - v$ を求めることを考える。

$$p_n(x) = (x^2 - ux - v)(b_0x^{n-2} + b_1x^{n-3} + \cdots + b_{n-2}) + b_{n-1}(x-u) + b_n$$

とおいて、係数を比較すると

$$\text{(A.4)} \qquad \begin{cases} b_0 = a_0 \\ b_1 = a_1 + ub_0 \\ b_k = a_k + ub_{k-1} + vb_{k-2} \quad (k = 2, 3, \cdots, n) \end{cases}$$

が得られる。b_k は (u, v) の関数と考えられるが、我々が求めたいものは $b_{n-1} = b_n = 0$ となるような (u, v) である。そこで u, v に関する連立 1 次方程式

$$b_{n-1}(u, v) = 0, \quad b_n(u, v) = 0$$

であると考えることができる。これを Newton 法により解くのがベアストウ・ヒッチコックの方法である。詳しい計算手順は省略する。例えば森 [12] に載っている。

一松による説明

$$x^n + a_1 x^{n-1} + \cdots + a_{n-1}x + a_n = 0$$

の二次の因数 $x^2 - px - q$ を求めよう。商を $x^{n-2} + b_1 x^{n-3} + \cdots + b_{n-2}$, 剰余を $b_{n-1}x + b_n$ とおくと、

$$b_1 = a_1 + p, \quad b_2 = a_2 + pb_1 + q, \quad b_k = a_k + pb_{k-1} + qb_{k-2} \quad (k = 3, 4, \cdots, n-1, n)$$

という漸化式で $\{b_k\}$ が計算できる。次に $\{c_k\}$ を

$$c_1 = b_1 + p, \quad c_2 = b_2 + pc_1 + q, \quad c_k = b_k + pc_{k-1} + qc_{k-2} \quad (k = 3, 4, \cdots, n-1, n)$$

という漸化式で計算すると、

$$\frac{\partial b_k}{\partial p} = c_{k-1}, \quad \frac{\partial b_k}{\partial q} = c_{k-2}$$

となることが示される。解くべき方程式は

$$b_n(p, q) = 0, \quad b_{n-1}(p, q) = 0$$

であるから、

$$b_n + c_{n-1}\Delta p + c_{n-2}\Delta q = 0$$
$$b_{n-1} + c_{n-2}\Delta p + c_{n-3}\Delta q = 0$$

$$p_{\nu+1} = p_\nu + \frac{b_n c_{n-3} - b_{n-1}c_{n-2}}{(c_{n-2})^2 - c_{n-1}c_{n-3}}, \quad q_{\nu+1} = q_\nu + \frac{b_{n-1}c_{n-1} - b_n c_{n-2}}{(c_{n-2})^2 - c_{n-1}c_{n-3}}$$

という反復を行う。初期値には $p_0 = q_0 = 0$ が使われるが、これを Bairstow 法とよぶ。除法を次数の低い方から進める変形を Hitchcock 法と呼ぶ。

一松 [8] から

「Bairstow 法は、これまで、実係数の代数方程式の複素根を、実数計算だけで求めるのに便利な方法として、広く使われてきた。実係数なら複素根は必ず共役複素数の対になって現れ、実 2 次因子の根になるからである。しかし機械的にこれを使って失敗した例が多い。その多くは、Newton 法には大域的収束性が保証されないのに、適当な初期値からいい加減に始めた場合である。特に虚数部が小さい根 $\alpha \pm i\beta$ があると、あたかも α に実根があるかのように見え、他の実根 γ と合わせた 2 次因子 $(x-\alpha)(x-\gamma)$ の近似が計算されかねない。ある段階に進むと、$x = \alpha$ の付近に実根がないため、分母が 0 近くなって、収束しかかった反復列が大きく飛ぶという現象を生ずる。これを防ぐための手法もいろいろ研究されている。筆者のささやかな経験では、根の分布を調べて十分よい初期値から始めることにし、複素数のよい計算体系が利用できる場合には、複素数の 1 変数の Newton 法を使うようにした方が無難と思う。」

A.5　Strum の方法

(Fourier 解析で有名な Fourier による先駆的な研究がある ([] を見よ)。行列の固有値問題でよく利用されている。一般の代数方程式の数値解法としては、数値的不安定性が心配になる。)

Strum 列は数値計算法のあちこちで顔を出す。ここで簡単にまとめておく。なお、以下の記述は森 [12], 一松 [8] から適当に抜き書きしたものである。高木 [6] を読んで必要なところを書いた方がよいか。

固有値問題のノートにも Strum 列のことを書いておいた。時々見比べて、マージすること。

A.5.1　スツルムの定理

実軸上の区間 $[a,b]$ における多項式 $f(x)$ の実根の個数に関する有名な定理。

定義 A.5.1 (Strum 列) $f(x) \in \mathbf{R}[x]$, $[a,b]$ を $\mathbf{R}$ の区間とする。このとき次の 4 つの条件を満足する実係数多項式列

$$f_0(x) = f(x), f_1(x), f_2(x), \cdots, f_\ell(x)$$

は区間 $[a,b]$ において **Strum 列**をなすという。

(1) $\forall x \in [a,b]$ に対して、隣り合う二つの多項式 $f_k(x),\ f_{k+1}(x)$ は同時に 0 にはならない。

(2) $x_0 \in [a,b]$, $k \in \{1,2,\cdots,\ell-1\}$ において $f_k(x_0) = 0$ となれば、$f_{k-1}(x_0)f_{k+1}(x_0) < 0$.

(3) 列の最後の多項式 $f_\ell(x)$ は $[a,b]$ において一定の符号を持つ。

(4) $x_0 \in [a,b]$ において $f(x_0) = 0$ ならば $f'(x_0)f_1(x_0) > 0$.

この定理によって、区間 $[a,b]$ 内に存在する $f(x) = 0$ の解を数を知ることができるが、二分探索の技法と組み合わせることで、解が存在する区間を好きなだけ小さくすることが出来、その区間内の適当な点を解の近似値として採用するという近似解法ができる。これを **Strum** の方法と呼ぶ。

定義 A.5.2 (符号変化の回数) 実係数多項式の列 $f(x) = f_0(x),\ f_1(x),\ \cdots,\ f_\ell(x)$ が区間 $[a,b]$ で Strum 列をなすとき、$f(x)$ の根でない $x \in [a,b]$ に対して、関数値の列

$$f_0(x), f_1(x), \cdots, f_\ell(x)$$

を左から右に見ていったときの符号の変化の回数 $N(x)$ が定義できる。

- x において、すべての k につき $f_k(x) \neq 0$ が成り立つ場合の $N(x)$ の定義は明らか。
- ある k について $f_k(x) = 0$ となった場合は、仮定 (Strum 列の条件 (3) と x が $f(x)$ の根でないという仮定) から $k = 0, \ell$ はありえず $1 \leq k \leq \ell - 1$ で、その場合も Strum 列の条件 (2) によって、$f_{k-1}(x)f_{k+1}(x) < 0$ となることから、向う三軒両隣では

$f_{k-1}(x)$	$f_k(x)$	$f_{k+1}(x)$
$+$	0	$-$

または

$f_{k-1}(x)$	$f_k(x)$	$f_{k+1}(x)$
$-$	0	$+$

のいずれかのパターンしかありえない。そこで、この 3 項で 1 回符号変化したと数えることにする。

例 A.5.1 $+, +, -, 0, +, +, -$ では 3 回と数える。

定理 A.5.1 (Strum) 実係数多項式の列 $f(x) = f_0(x),\ f_1(x),\ \cdots,\ f_\ell(x)$ は区間 $[a,b]$ で Strum 列をなし、$f(a)f(b) \neq 0$ であるとする。このとき、$f(x) \neq 0$ なる $x \in [a,b]$ に対して、関数値の列

$$f_0(x), f_1(x), \cdots, f_\ell(x)$$

を左から右に見ていったときの符号の変化の回数を $N(x)$ とすると、方程式 $f_0(x) = 0$ の区間 $[a,b]$ における解の個数は $N(a) - N(b)$ である。

証明　$f(x) = 0$ の解の個数は有限個である。そのうち $[a,b]$ 内にあるものを大きさの順に並べて

$$x_1 < x_2 < \cdots < x_n$$

とする。$n+1$ 個の区間

$$I_0 = [a, x_1), \quad I_1 = (x_1, x_2), \quad I_2 = (x_2, x_3), \quad I_{n-1} = (x_{n-1}, x_n), \quad I_n = (x_n, b]$$

の合併において $N(x)$ は定義できる。以下では

(1) 各区間 I_j において $N(x)$ は定数である:

$$\exists \{n_j\}_{j=0}^{n} \quad \text{s.t.} \quad N(x) = n_j \quad (x \in I_j,\ j = 0, 1, \cdots, n)$$

(2) $n_{j+1} - n_j = -1$ $(j = 0, 1, \cdots, n-1)$

であることを証明する (この二つから容易に $N(b) - N(a) = n$ が導かれる)。

(1) の証明　I_j 内のある部分区間 J において、すべての多項式 $f_k(x_0) \neq 0$ とすると、$f_k(x)$ の符号は J で一定であるので (符号が変化するには 0 にならないといけない)、$N(x)$ の値は変化しないことが分かる。

そこで、ある多項式 $f_k(x)$ の解 $x_* \in I_j$ を通過した場合を考える。Strum 列の条件から $k \neq \ell$, I_j の取り方から $k \neq 0$ であるから、$1 \leq k \leq \ell - 1$ である。Strum 列の条件 (2) から $f_{k-1}(x_*)f_{k+1}(x_*) < 0$ であるが、連続性から x_* の十分小さな近傍 V を取れば、そこで $f_{k-1}(x)$ と $f_{k+1}(x)$ は定符号となり、V 上で $f_{k-1}(x)f_{k+1}(x) < 0$ がなりたつ。それゆえ、列

$$f_0(x), f_1(x), \cdots, f_{k-1}(x), f_k(x), f_{k+1}(x), \cdots, f_\ell(x)$$

の部分列

$$f_{k-1}(x), f_k(x), f_{k+1}(x)$$

の符号については次の二つのいずれか一方だけしか起こらない。

(a)	$f_{k-1}(x)$	$f_k(x)$	$f_{k+1}(x)$		(b)	$f_{k-1}(x)$	$f_k(x)$	$f_{k+1}(x)$
	V 上 $-$		V 上 $+$			V 上 $+$		V 上 $-$

それゆえ、($f_k(x)$ の符号が何であっても) 任意の $x \in V$ について、部分列 $f_{k-1}(x)$, $f_k(x)$, $f_{k+1}(x)$ における符号の変化は 1 として $N(x)$ の値に算入される。これから十分小さな x_* の近傍で $N(x)$ の値に変化はないことが分かる。ゆえに $N(x)$ は I_j 上で定数であることが分かる。

(2) の証明　$f(x) = 0$ の解 x_j において考える。Strum 列の条件 (4) は $f'(x_j)$ と $f_1(x_j)$ が同符号であることを示している。例えば $f'(x_j) > 0$ の場合、十分小さな $\varepsilon > 0$ を取ると、

- $f(x) < 0$ $(x \in [x_j - \varepsilon, x_j))$, $f(x_j) = 0$, $f(x) > 0$ $(x \in (x_j, x_j + \varepsilon])$.
- $f_1(x) > 0$ $(x \in (x_j - \varepsilon, x_j + \varepsilon))$.

これから

$$n_{j+1} - n_j = N(x_j + 0) - N(x_j - 0) = -1$$

であることが分かる。$f'(x_j) < 0$ の場合も同様の議論で $n_{j+1} - n_j = -1$ であることが分かる。
■

A.5.2　ユークリッドの互除法による Strum 列の生成

多項式 $f(x) \in \mathbf{R}[x]$ が与えられたとき、$f_0(x) = f(x)$ と $f_1(x) = f'(x)$ から Euclid の互除法を行い、関数列 $f_0(x)$, $f_1(x)$, $\cdots$, $f_\ell(x)$ を作る:

$$f_0(x) := f(x), \quad f_1(x) := f'(x), \tag{A.5}$$

$$
\begin{aligned}
f(x) &= q_1(x)f_1(x) - f_2(x), \quad \deg f_2(x) < \deg f_1(x), \\
f_1(x) &= q_2(x)f_2(x) - f_3(x), \quad \deg f_3(x) < \deg f_2(x), \\
f_2(x) &= q_3(x)f_3(x) - f_4(x), \quad \deg f_4(x) < \deg f_3(x), \\
&\vdots \\
f_{k-1}(x) &= q_{k-1}(x)f_k(x) - f_{k+1}(x), \quad \deg f_{k+1}(x) < \deg f_k(x), \\
&\vdots \\
f_{\ell-2}(x) &= q_{\ell-1}(x)f_{\ell-1}(x) - f_\ell(x), \quad \deg f_\ell(x) < \deg f_{\ell-1}(x), \\
f_{\ell-1}(x) &= q_\ell(x)f_\ell(x).
\end{aligned} \tag{A.6}
$$

(普通の互除法と異なり、$f_{k-1}(x)$ を $f_k(x)$ で割ったときの普通の剰余 $\times(-1)$ を $f_{k+1}(x)$ とする。)

よく知られているように $f_\ell(x)$ は $f(x)$ と $f'(x)$ の最大公約多項式であるから、$f(x) \in \mathbf{R}[x]$ が重根を持たない場合、$f_\ell(x) \equiv$ 定数 $(\neq 0)$ となることに注意しよう。以下この場合に $f(x) = 0$ の解 (根) を求めることを考える。$f(x)$ が重根を持つ場合は $f(x)$ の代わりに $g(x) = f(x)/f_\ell(x)$ を考えることで同様の議論ができる。

定理 A.5.2 $f(x) \in \mathbf{R}[x]$ が重根を持たないとき、$f(a)f(b) \neq 0$ を満たす任意の区間 $[a, b]$ において、$f(x)$ と $f'(x)$ から互除法で作った剰余列 (詳しくは (A.5), (A.6) 参照)

$$
f(x) = f_0(x), f'(x) = f_1(x), f_2(x), \cdots, f_\ell(x)
$$

は Strum 列をなす。

証明　まず $f_\ell(x) \equiv$ 定数 であるから、Strum 列の条件 (3) は満たされている。次にある $x_0 \in [a, b]$, ある $k \in \{0, 1, \cdots, \ell - 1\}$ に対して

$$
f_k(x_0) = f_{k+1}(x_0) = 0
$$

となったとすると、式 (A.6) から $f_{k+2}(x_0)$ 以降の $f_j(x_0)$ もすべて 0 になり、特に $f_\ell(x_0) = 0$. これは $f_\ell(x)$ が定数関数 $(\neq 0)$ であることに矛盾する。ゆえに Strum 列の条件 (1) が満たされる。次にある点 x_0, ある $k \in \{1, 2, \cdots, \ell - 1\}$ に対して $f_k(x_0) = 0$ となったとすると、式 (A.6) から

$$
f_{k-1}(x_0) = -f_{k+1}(x_0).
$$

ゆえに Strum 列の条件 (2) も満たされる (条件 (3) から上式の値は 0 にならないことに注意)。最後に x_0 が $f(x)$ の根であるとき、$f(x)$ が重根を持たないという仮定から $f'(x_0) \neq 0$ で、

$$
f'(x_0)f_1(x_0) = f'(x_0)^2 > 0
$$

となり、条件 (4) も満たされる。 ■

A.5.3　3 重対角行列の固有多項式と Strum 列

$$T = \begin{pmatrix} a_1 & b_1 & 0 & & 0 \\ b_1 & a_2 & b_2 & & \\ & \ddots & \ddots & \ddots & \\ & & b_{N-2} & a_{N-1} & b_{N-1} \\ 0 & & 0 & b_{N-1} & a_N \end{pmatrix}$$

を実対称三重対角行列とする。

注意 **A.5.1** (ちょっとした) 実は以下の議論で T の対称性はあまり本質的でなく、対角線をはさんで対称の位置にある成分が同符号、すなわち

$$t_{k,k-1}t_{k-1,k} > 0 \quad k = 2, \cdots, N$$

が成り立つことが本質的であるという指摘が一松先生の本にあった。なるほど。行列を Hessenberg 形にする算法で、特に与えられた行列が実対称であった場合には、結果が実対称三重対角になってしまうということで、(以下 Strum の方法を用いて固有値を求める段階になると) 実対称性はある意味では必要性はあまりないのだが、最初に実対称性がないと三重対角化できないわけで、まあ必ずついてくる「おまけ」ですな。 ■

$b_k \neq 0$ $(k = 1, 2, \cdots, N-1)$ と仮定する。もしある k に対して $b_k \neq 0$ ならば

$$T = \begin{pmatrix} T' & O \\ O & T'' \end{pmatrix}$$

となり、T', T'' の固有値を求める問題に帰着できるから、一般性は失われない。

$p_k(\lambda)$ を $\lambda I - T$ の第 k 主座行列式とする $(k = 0, 1, \cdots, N)$。すなわち

$$p_k(\lambda) := \begin{cases} \det(\lambda I_k - T_k) & (k = 1, 2, \cdots, N) \\ 1 & (k = 0) \end{cases} . \tag{A.7}$$

ただし、

$$I_k = k \text{ 次の単位行列}, \quad T_k = \begin{pmatrix} a_1 & b_1 & 0 & & 0 \\ b_2 & a_2 & b_1 & & \\ & & & & \\ & & b_{k-2} & a_{k-1} & b_{k-1} \\ 0 & & 0 & b_{k-1} & a_k \end{pmatrix} .$$

すぐ分かる命題を二つ。

補題 A.5.1 (漸化式) (A.7) で定義された $\{p_j(\lambda)\}_{j=0}^N$ について、以下が成り立つ。

$$\begin{cases} p_0(\lambda) &= 1, \\ p_1(\lambda) &= \lambda - a_1, \\ p_{k+1}(\lambda) &= (\lambda - a_{k+1})p_k(\lambda) - {b_k}^2 p_{k-1}(\lambda) \quad (k = 1, 2, \cdots, N-1), \\ p_N(\lambda) &= \det(\lambda I - T). \end{cases}$$

証明　行列式の行に関する展開定理を用いる。■

補題 A.5.2 (Strum 列であること) (A.7) で定義された $\{p_j(\lambda)\}_{j=0}^N$ を逆順に並べた多項式列

$$p_N(\lambda), p_{N-1}(\lambda), \cdots, p_1(\lambda), p_0(\lambda)$$

は任意の閉区間 $[a,b]$ において Strum 列である。すなわち

(1) 隣り合う二つの多項式 $p_k(\lambda)$, $p_{k+1}(\lambda)$ は共通根を持たない。

(2) ある $\lambda_0 \in \mathbf{R}$ において $p_k(\lambda_0) = 0$ ならば $p_{k-1}(\lambda_0)p_{k+1}(\lambda_0) < 0$ $(k = 1, 2, \cdots, N-1)$.

(3) 列の最後の多項式 $p_0(\lambda)$ は $\mathbf{R}$ において定符号である。

(4) ある $\lambda_0 \in \mathbf{R}$ において $p_N(\lambda_0) = 0$ ならば $p'_N(\lambda_0)p_{N-1}(\lambda_0) > 0$.

証明

(1) もしも $p_k(\lambda_0) = p_{k+1}(\lambda_0) = 0$ とすると、漸化式から ${b_k}^2 p_{k-1}(\lambda_0) = 0$. $b_k = 0$ と仮定したから $p_{k-1}(\lambda_0) = 0$. これを繰り返すと

$$0 = p_{k+1}(\lambda_0) = p_k(\lambda_0) = p_{k-1}(\lambda_0) = \cdots = p_2(\lambda_0) = p_1(\lambda_0) = p_0(\lambda_0).$$

これから

$$p_0(\lambda_0) = 0$$

これは $p_0 \equiv 1$ に矛盾する。

(2) $p_k(\lambda_0) = 0$ を漸化式に代入すると $p_{k+1}(\lambda_0) = -{b_k}^2 p_{k-1}(\lambda_0)$. 前項より左辺 $\neq 0$. これから $p_{k+1}(\lambda_0)$, $p_{k-1}(\lambda_0)$ は異符号である。 ■

(3) $p_0(\lambda) \equiv 1$ であるから明らか。

(4) 漸化式

$$p_k(\lambda) = (\lambda - a_k)p_{k-1}(\lambda) - {b_{k-1}}^2 p_{k-2}(\lambda) \tag{A.8}$$

を微分すると、

$$p'_k(\lambda) = p_{k-1}(\lambda) + (\lambda - a_k)p'_{k-1}(\lambda) - {b_{k-1}}^2 p'_{k-2}(\lambda). \tag{A.9}$$

(A.8) と (A.9) から

$$p'_k(\lambda)p_{k-1}(\lambda) - p_k(\lambda)p'_{k-1}(\lambda) = \beta_{k-1}^2 \left(p'_{k-1}(\lambda)p_{k-2}(\lambda) - p_{k-1}(\lambda)p'_{k-2}(\lambda)\right) + p_{k-1}^2(\lambda) \tag{A.10}$$

が得られる。ここで

$$q_k(\lambda) := p'_k(\lambda)p'_{k-1}(\lambda) - p_k(\lambda)p_{k-1}(\lambda)$$

とおくと (A.10) は

$$q_k(\lambda) = p_{k-1}(\lambda)^2 + \beta_{k-1}(\lambda)^2 q_{k-1}(\lambda) \quad (k = 2, 3, \cdots, N).$$

となる。ところで

$$q_1(\lambda) = p_1'(\lambda)p_0(\lambda) - p_1(\lambda)p_0'(\lambda) = p_1'(\lambda) = 1 \cdot 1 - (\lambda - \alpha_1) \cdot 0 = 1 > 0$$

であるから、以下帰納的に

$$q_k(\lambda) > 0 \quad (k = 2, 3, \cdots, N).$$

特に

$$q_N(\lambda) = p_N'(\lambda)p_{N-1}(\lambda) - p_N(\lambda)p_{N-1}'(\lambda) > 0$$

であるが、$p_N(\lambda) = 0$ であるから

$$p_N'(\lambda)p_{N-1}(\lambda) > 0. \quad \blacksquare$$

符号の変化数の計算に関する注意　$p_N(a) \neq 0$ なる a に対して

$$N(a) \overset{\text{def.}}{=} \text{“}\{p_0(a), p_1(a), \cdots, p_N(a)\}\text{” の符号の変化数}$$

とおく (逆順であっても符号の変化数は同じになる)。例えば

$p_0(a)$	$p_1(a)$	$p_2(a)$	$p_3(a)$	$p_4(a)$	$p_5(a)$	$p_6(a)$	$p_7(a)$	$p_8(a)$	$p_9(a)$	$p_{10}(a)$
$+$	$-$	$-$	$-$	0	$+$	$+$	$+$	$+$	$-$	$-$

では $N(a) = 3$.

この符号の変化数は (特別な注意をせずに) 数値的に安定して計算できる。つまり絶対値が非常に小さくて、符号の判別がつきにくい場合も、「符号の変化数」そのものは疑いがなく計算できる。例えば

$p_{k-1}(a)$	$p_k(a)$	$p_{k+1}(a)$
$+$	絶対値小	$-$

または

$p_{k-1}(a)$	$p_k(a)$	$p_{k+1}(a)$
$-$	絶対値小	$+$

において $p_k(a)$ が正であっても負であっても 0 であっても符号の変化数の計算にとっては影響がない。注意すべきは Strum 列の条件 (iv) から

$p_{k-1}(a)$	$p_k(a)$	$p_{k+1}(a)$
$+$	絶対値小	$+$

または

$p_{k-1}(a)$	$p_k(a)$	$p_{k+1}(a)$
$-$	絶対値小	$-$

のような場合 (もしこうなったら符号の変化数の計算がむつかしい) が起こり得ないことである。

A.5.4　直交多項式の作る Strum 列

(まあここは単なる覚え書き。後で肉付けするかもしれない。)

$w\colon [a, b] \to \mathbf{R}$ は連続で、有限個の点で 0 になる他は正で、

$$\sup_{k \in \mathbf{N}} \int_a^b x^k w(x)\, dx < \infty$$

を満たすような関数とする。このとき $[a,b]$ 上の実数値連続関数全体の集合に

$$(u,v)_w := \int_a^b u(x)v(x)w(x)\,dx$$

で定義される内積を導入して、内積空間としたものを $H_w(a,b)$ とする。

関数列 $\{1,x,\cdots,x^n\}$ から Gram-Schimidt の直交化法によって得られる直交多項式系を $\{p_0(x),p_1(x),\cdots,p_n(x)\}$ とする。

命題 A.5.1 $H_w(a,b)$ において関数列 $\{1,x,\cdots,x^n\}$ から Gram-Schimidt の直交化法によって得られる直交多項式系を $\{p_0(x),p_1(x),\cdots,p_n(x)\}$ とし、$p_n(x)$ の最高次の係数を μ_n とすると、

$$\inf\{\|q\|_w; q(x)\in\mathbf{R}[x], \deg q(x)=n, q(x)\text{ の最高次係数}=1\} = \|p_n/\mu_n\|_w.$$

命題 A.5.2 $H_w(a,b)$ において関数列 $\{1,x,\cdots,x^n\}$ から Gram-Schimidt の直交化法によって得られる直交多項式系を $\{p_0(x),p_1(x),\cdots,p_n(x)\}$ とすると、$p_n(x)$ の根はすべて単根で、区間 $[a,b]$ の内部にある。

命題 A.5.3 $H_w(a,b)$ において関数列 $\{1,x,\cdots,x^n\}$ から Gram-Schimidt の直交化法によって得られる直交多項式系を $\{p_0(x),p_1(x),\cdots,p_n(x)\}$ とすると、次のような 3 項漸化式が存在する。

$$p_k(x) = (\alpha_k x+\beta_k)p_{k-1}(x) - \gamma_k p_{k-2}(x) \quad (k=1,2,\cdots).$$

ただし $p_{-1}(x)\equiv 0$ とし、

$$\begin{aligned}\alpha_k &= \frac{\mu_k}{\mu_{k-1}}, \quad \beta_k = -\frac{\alpha_k(xp_{k-1},p_{k-1})_w}{\lambda_{k-1}},\\ \gamma_k &= \frac{\alpha_k(xp_{k-1},p_{k-2})_w}{\lambda_{k-2}} = \frac{\mu_k\mu_{k-2}\lambda_{k-1}}{\mu_{k-1}^2\lambda_{k-2}}.\end{aligned}$$

ただし、$\lambda_k=(p_k,p_k)_w$, $\mu_k = p_k(x)$ の最高次係数。

命題 A.5.4 $H_w(a,b)$ において関数列 $\{1,x,\cdots,x^n\}$ から Gram-Schimidt の直交化法によって得られる直交多項式系を $\{p_0(x),p_1(x),\cdots,p_n(x)\}$ とすると (最高次の係数が正であるようにしておくと)、

$$p_n(x),p_{n-1}(x),\cdots,p_1(x),p_0(x)$$

は区間 $[a,b]$ において Strum 列をなす。

A.5.5　一般化された Strum 列

一松先生の本にあった Strum 列の定義の条件は森先生の本よりも少し緩い条件であった (つまり一般化してあると言える)。明示していないけれど $p_k(\lambda)$ の連続性は仮定されている、と思う。

次の性質を持つ関数列 $\{p_k(\lambda)\}_{k=0}^n$ を **Strum 系**と言う。

(1) 各 $p_k(\lambda)$ の解は有限個; $p_k(\lambda_0) = p_{k+1}(\lambda_0) = 0$ はありえない。

(2) $p_k(\lambda_0) = 0$ のとき $p_{k-1}(\lambda_0)p_{k+1}(\lambda_0) < 0$.

(3) $p_0(\lambda)$ は定符号。

もちろん対応する Strum の定理の結論は少し複雑になる。

定理 A.5.3 (一般化された Strum の定理) $\{p_k(\lambda)\}_{k=0}^n$ を Strum 系とする。$p_n(a)p_n(b) \neq 0$ $(a < b)$ のとき、

$$N(a) - N(b) = \sum_{p_n(\lambda_0)=0} \chi(\lambda_0).$$

ただし $\chi(\lambda_0)$ は、λ を λ_0- から λ_0+ に変化させるときの $p_n(\lambda)/p_{n-1}(\lambda)$ の符号変化を表す量で

$$\chi(\lambda_0) := \begin{cases} 1 & (- \text{ から } + \text{ に変化}) \\ 0 & (\text{符号の変化なし}) \\ -1 & (+ \text{ から } - \text{ に変化}). \end{cases}$$

証明　略 ■

系 A.5.1 $N(a) \neq N(B)$ ならば $[a, b]$ 内に $p_n(\lambda) = 0$ の解がある。

系 A.5.2 (実対称三重対角行列の固有値問題) n 次実対称三重対角行列

$$T = \begin{pmatrix} a_1 & b_1 & 0 & & 0 \\ b_1 & a_2 & b_2 & & \\ & \ddots & \ddots & \ddots & \\ & & b_{n-2} & a_{n-1} & b_{n-1} \\ 0 & & 0 & b_{n-1} & a_n \end{pmatrix}, \quad b_k \neq 0 \quad (k = 1, 2, \cdots, n-1)$$

から $p_k(\lambda) = \det(\lambda I_k - T_k)$ (T_k は T の第 k 次首座行列) として作った Strum 列 $p_n(\lambda)$, $p_{n-1}(\lambda)$, $\cdots$, $p_1(\lambda)$, $p_0(\lambda) \equiv 1$ について、n 個の固有値はすべて実単根であり、各固有値 λ_0 において、つねに $\chi(\lambda_0) = 1$ である。ゆえに $p_n(a)p_n(b) \neq 0$ となる $a < b$ に対して

$$N(a) - N(b) = \text{区間 } [a, b] \text{ 内の } f(x) = 0 \text{ の根の個数}.$$

そして $p_k(\lambda)$ の隣接した根の中間に、必ず $p_{k-1}(\lambda) = 0$ の根がある $(k = 1, 2, \cdots, n)$。

証明　略。 ■

A.6 Bernoulli 法

与えられた解析関数の零点のうちで原点に最も近いものを求める方法と、その多項式への応用 (ベルヌイの方法) を述べる。

補題 A.6.1 (ケーニッヒ)

$$h(z) = c_0 + c_1 z + c_2 z^2 + \cdots, \quad c_0 \neq 0$$

を $D_R := |z| < R$ における有理型関数とする。またこの関数は D_R には $z = z_r$ にただ一つの単純な極を持つものとする。このとき

$$\frac{c_k}{c_{k+1}} = z_r \left[1 + O\left(\left| \frac{z_r}{\rho} \right|^{k+1} \right) \right].$$

ただし ρ は $|z_r| < \rho < R$ を満たす数である。

複素平面上の円板 $D_R = \{z \in \mathbf{C}; |z| < R\}$ で定義された有理型関数

$$f(z) = a_0 + a_1 z + a_2 z^2 + \cdots, \quad a_0 \neq 0$$

が、D_R 内に唯一の零点 z_a を持つとする。また

$$g(z) = b_0 + b_1 z + b_2 z^2 + \cdots$$

を D_R で正則で、$g(z_a) \neq 0$ を満たす関数とする。このとき

$$h(z) = \frac{g(z)}{f(z)} = c_0 + c_1 z + c_2 z^2 + \cdots$$

は補題の条件を満足する。ゆえに

$$\lim_{k\to\infty} \frac{c_{k+1}}{c_k} = \frac{1}{z_a}.$$

これから $f(z) = 0$ の絶対値最小の零点が得られる。

$g(z) = h(z)f(z)$ を

$$b_0 + b_1 z + b_2 z^2 + \cdots = (a_0 + a_1 z + a_2 z^2 + \cdots)(c_0 + c_1 z + c_2 z^2 + \cdots)$$

から

$$\begin{aligned} a_0 c_0 &= b_0 \\ a_0 c_1 + a_1 c_0 &= b_1 \\ a_0 c_2 + a_1 c_1 + a_2 c_0 &= b_2 \\ &\cdots\cdots\cdots\cdots\cdots\cdots \end{aligned} \tag{A.11}$$

ベルヌイ法　上述の方法を

$$p_n(z) = a_0 z^n + a_1 z^{n-1} + \cdots + a_{n-1} z + a_n = 0$$

$$f(z) := z^n p_n(z^{-1})$$

とおくと $f(z)$ も n 次多項式である。そして $f(z)$ の絶対値最小の零点 z_a と $p_n(z)$ の絶対値最大の零点 z_p との間には

$$z_p = \frac{1}{z_a}$$

の関係がある。

(この項書きかけ)

一松先生曰く、

> (代数方程式の) 最大根を求めるには古くから (Daniel) Bernoulli 法が知られている (本質的には (固有値問題を解くための) 累乗法と同じ)。

A.7　商差法

商差法 (**quotient-difference algorithm**, QD 法) は、昔から知られていたが、1950 年代初めの Rutishauser が導入し、後に Henrici が発展させた方法である。

一松 [8] (1982) に解説がある。「現在でも低次 (10 次以下) で実根の多い代数方程式の、序盤ないし中盤むきの手法としては有用であり、多くの関連話題とともに、理論的には重要である。しかし、汎用とするには難点が多く、一つの補助算法と考えた方がよい。(中略) 以上のような理由で、現在では実用価値が低下しているので、算法の解説のみにとどめ、定理の証明を省略した。」

A.8　次数低下

Durand-Kerner 法のように代数方程式 $p(x) = 0$ の全根を同時に探索する方法を採用した場合には必要ないが、例えば Newton 法のような方法で、一つの根の近似根 ξ を求めた後、$p(x)$ を $(x - \xi)$ で割って、1 次低い代数方程式に還元することを次数低下あるいは減次 (**deflation**) と呼ぶ。

次数低下をする際に、除法を「いつでも最高次の項から行う」のは危ない (一松 [8] の §21 を見よ)。

この点については、櫻井 [7] に詳しい。

n 次多項式 $f(x)$ を $x - \alpha$ で割った商を $q(x)$, 剰余を r とおくと、

$$f(x) = q(x)(x - \alpha) + r$$

であり、$f(\alpha) = r$ であるから、α が $f(x)$ の根のときは $r = 0$ となる。

また $Q(x)$, R を

$$f(x) = Q(x)(x - \alpha) + Rx^n$$

を満たすように決めた場合にも $f(\alpha) = 0$ のときには $R = 0$ であり、従って $q(x) = Q(x)$ であるが、数値計算で求める場合、丸め誤差の観点からは二つのやり方に差がある。前者を高次項からの減次、後者を低次項からの減次という。

高次項からの減次　$f(x) = a_0 + a_1 x + \cdots + a_n x^n$ を因子 $x - \alpha$ によって高次項から減次した多項式 $q(x) = q_0 + q_1 x + \cdots + q_{n-1} x^{n-1}$ を求める。

1. $q_{n-1} := a_n$ とおく。
2. $i = n-1, n-2, \cdots, 1$ の順に、以下を繰り返す。
   $q_{i-1} := q_i \alpha + a_i$.
3. $q(x) := q_0 + q_1 x + \cdots + q_{n-1} x^{n-1}$ とおく。

低次項からの減次　$f(x) = a_0 + a_1 x + \cdots + a_n x^n$ を因子 $x - \alpha$ によって高次項から減次した多項式 $Q(x) = Q_0 + Q_1 x + \cdots + Q_{n-1} x^{n-1}$ を求める。

1. $Q_0 := -a_0/\alpha$ とおく。
2. $i = 1, 2, \cdots, n-1$ の順に、以下を繰り返す。
   $Q_i := (Q_{i-1} - a_i)/\alpha$.
3. $Q(x) := Q_0 + Q_1 x + \cdots + Q_{n-1} x^{n-1}$ とおく。

絶対値の小さな根については高次項からの減次、絶対値の大きな根については低次項からの減次をした方が誤差の影響が少なくなる。絶対値が中くらいの場合、$|a_k \alpha^k|$ が最大となる $k = k_{\max}$ を探し、

$$\widetilde{q}(x) = q_0 + q_1 x + \cdots + q_{k_{\max}} x^{k_{\max}} + Q_{k_{\max}+1} x^{k_{\max}+1} + \cdots + Q_{n-1} x^{n-1}.$$

を採用すると良い結果が得られることがある (そうだ)。

A.9　Jenkins and Traub 法

Jenkins–Traub 法については、Jenkins and Traub [20], Ralston and Rabinowitz [21]

A.10　その他

この節の記述は櫻井 [7] による。

A.10.1　Euler 法

f を x_0 の近傍で 2 次式で近似すると

$$f(x) \fallingdotseq f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2$$

そこで

$$f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 = 0$$

の 2 根のうち x_0 に近い方を新たな近似解とする。

A.10.2　ハレー法

A.10.3　ポメンタール法

付 録B　応用解析IVの数値実験

2003 年度の応用解析 IV の内容から。

本日は、C++ で書かれたサンプル・プログラムを応用解析 IV ホームページから入手して、DK 法の簡単な数値実験を行なう。

B.1　6701 号室のワークステーションの利用

数学科の学生は 6701 号室にあるワークステーション、パソコンを利用することができる。ユーザー名は情報科学センターのワークステーションと (とりあえず) 同じユーザー名となるが、独立に管理している (だから、例えばパスワードは一応別物[1]となる)。

6701 号室の利用については、以下の WWW ページを読むこと。

http://www.math.meiji.ac.jp/howto/

B.2　応用解析 IV ホームページ

この講義のホームページ

http://www.math.meiji.ac.jp/~mk/lecture/ouyoukaiseki4/

が用意されていて、サンプル・プログラムはそこから入手できる。

なお、講義ノートなどは、受講者のみに公開している (一般には公開していない)。ユーザー名は ouyoukaiseki4 で、パスワードは　　である。

サンプル・プログラムを手元のコンピューターにコピーする方法　Netscape を使う場合、サンプル・プログラムをアクセスしてから、File メニューの「名前を付けて保存 (S)」を選択する。IE を使う場合は...

[1] 自分で同じものを選ぶことは可能である。

B.3 C++ について

B.3.1 まずはサンプル・プログラム

quadratic-equation.C

```
 1  // quadratic-equation.C --- 複素係数の 2 次方程式を解く。
 2  //
 3  //    コンパイルは（6701 号室では例えば）
 4  //      g++ -o quadratic-equation quadratic-equation.C
 5  //
 6  //    注意：素朴な（桁落ちの対策などしていない）アルゴリズムを使っている。
 7
 8  #include <iostream.h>
 9  #include <complex.h>
10
11  int main()
12  {
13    complex<double> a, b, c, D, x1, x2;
14
15    cout << "複素係数の 2 次方程式 a x^2+b x+c=0 (a ≠ 0) を解きます。" << endl;
16    cout << " 複素数は ( ) でくくり、実部と虚部をカンマ , で区切って表す。"
17         << endl;
18    cout << " 例えば 1+2i は (1,2) と表わします。" << endl;
19    cout << "入力してください。" << endl;
20
21    cout << "a="; cin >> a;
22    cout << "b="; cin >> b;
23    cout << "c="; cin >> c;
24
25    cout << "a=" << a << ", b=" << b << ", c=" << c << endl;
26
27    D = sqrt(b * b - 4 * a * c);
28    x1 = (-b + D) / (2 * a);
29    x2 = (-b - D) / (2 * a);
30    cout << "x1=" << x1 << endl;
31    cout << "x2=" << x2 << endl;
32
33    return 0;
34  }
```

実行例

```
oyabun% g++ -o quadratic-equation quadratic-equation.C
oyabun% ./quadratic-equation
複素係数の 2 次方程式 a x^2+b x+c=0 (a ≠ 0) を解きます。
 複素数は ( ) でくくり、実部と虚部をカンマ , で区切って表す。
 例えば 1+2i は (1,2) と表わします。
入力してください。
a=1
b=1
c=1
a=(1,0), b=(1,0), c=(1,0)
x1=(-0.5,0.866025)
x2=(-0.5,-0.866025)
oyabun%
```

B.3.2 C++ の (とても簡潔な) 紹介

C++ は C 言語を基礎にして作られたプログラミング言語である。

大抵の C プログラムは C++ プログラムでもある

という言葉が示すように、C++ は C 言語を拡張したものと考えることができる。主な拡張点は、オブジェクト指向プログラミング[2]を実現しやすくする機能である。

今回は、C 言語よりも複素数を扱うプログラミングがしやすいという理由から、C++ を採用した (上のプログラムを C 言語で書くとどうなるかを想像してみよう)。

B.3.3 サンプル・プログラムを読むための C++ の知識

入出力に C 言語の関数 (printf() 等) を使うことも可能だが、普通は >>, << 演算子を使うことが推奨されている。これを使う場合、プログラムの先頭付近で #include <iostream.h> としておく必要がある。

標準出力への出力 (通常は画面への表示)

cout << 式; あるいは cout << 式$_1$ << 式$_2$ << ··· << 式$_n$;
(なお行末を表わす名前 endl (END of Line?) を覚えておこう。)

標準入力からの入力 (通常はキーボードからの入力)

cin >> 変数名;

複素数データの処理

実験に用いる C++ 処理系では、complex<double> というクラス[a]が用意されている。

- プログラムの先頭部分で #include <complex.h> とする。
- complex<double> 変数名; で複素数を表現する変数が定義できる。
- complex<double> クラスのデータに対して、通常の演算子 +, -, *, / で四則演算が出来る。比較演算子 ==, != もある。また real(), imag(), abs(), sqrt() などの関数も用意されている (意味は多分名前で分かってもらえると思う)。
- (もちろん) <<, >> 演算子を用いての入出力が可能である。その際、$x + \sqrt{-1}y$ ($x, y \in \mathbf{R}$) は (x,y) のように丸括弧 (,) を用いて 2 次元ベクトルのように表現する。例えば虚数単位 $\sqrt{-1}$ は (0,1) となる。

[a] クラスというのはオブジェクト指向言語における概念であるが、ここでは説明しない。型を拡張したものぐらいに思っておけばサンプル・プログラムは理解可能である。

[2] これが何であるかは説明しない。

B.3.4 g++ — 実験に用いる C++ 処理系

6701 号室のワークステーションでは、GCC というプログラミング言語処理系があり、C, C++, Objective C, Fortran 77 などのプログラミング言語で書かれたプログラムを扱うことができる。

- GCC で C++ プログラムをコンパイルするには、通常 g++ というコマンドを使う。オプション等の使い方は C コンパイラーである gcc に準じる。例えば、サンプル・プログラム quadratic-equation.C のコンパイル＆実行には、

```
oyabun% g++ quadratic-equation.C
oyabun% ./a.out
```

  または

```
oyabun% g++ -o quadratic-equation quadratic-equation.C
oyabun% ./quadratic-equation
```

  のようにすれば良い。

- GCC では、C++ で書かれたプログラムのファイル名末尾は .C (大文字の C であることに注意) とする習慣である。

B.4 グラフィックス・ライブラリィ GLSC について

サンプル・プログラム DK3.c では、収束の様子を図示するために、C, C++ や Fortran プログラムから利用可能なグラフィックス・ライブラリィ GLSC[3] を利用している。

GLSC について詳しいことが知りたければ、

http://www.math.meiji.ac.jp/~mk/labo/howto/index.html#GLSC

を参照すること。

[3] 龍谷大学で開発された。ftp://ftp.st.ryukoku.ac.jp/pub/ryukoku/software/math/ で公開されている。

サンプル・プログラムの中で GLSC を使っているところ

```
// GLSC のヘッダーファイルをインクルード
extern "C" {
#define G_DOUBLE
#include <glsc.h>
}

  .......

  // グラフィックス・ライブラリィ GLSC の初期化
  g_init("DKGRAPH", 220.0, 220.0);
  g_device(G_BOTH);
  // 座標系の指定
  g_def_scale(0,
              real(g) - 1.1 * R, real(g) + 1.1 * R,
              imag(g) - 1.1 * R, imag(g) + 1.1 * R,
              10.0, 10.0, 200.0, 200.0);
  g_sel_scale(0);
  .......

  // 線種、マーカーの指定
  g_def_line(0, G_BLACK, 0, G_LINE_SOLID);
  g_sel_line(0);
  g_def_marker(0, G_RED, 2, G_MARKER_CIRC);
  g_sel_marker(0);

  .......

  // 初期値と初期値を置く円を描く
  g_circle(real(g), imag(g), r0, G_YES, G_NO);
  for (i = 0; i < n; i++)
    g_marker(real(x[i]), imag(x[i]));

  .......

      // 図示する
      g_move(real(x[i]), imag(x[i]));
      g_plot(real(newx[i]), imag(newx[i]));
      g_marker(real(newx[i]), imag(newx[i]));

  .......

  // GLSC 終了の処理
  cout << "終了にはグラフィックスのウィンドウをクリックして下さい。" <<endl;
  g_sleep(-1.0);
  g_term();
```

上のプログラム断片を理解するための注

- g は根の重心の座標
- R は $\{z \in \mathbf{C}; |z-g| \leq R\}$ がすべての根を含むことが保証できる正数 (講義で解説済み)
- x[], newx[] はそれぞれ $\{x_i^{(k)}\}_{i=1}^n$, $\{x_i^{(k+1)}\}_{i=1}^n$ を記憶する変数

B.5　素朴な Newton 法の実行例

complex-newton.C

```
 1  /*
 2   * complex-newton.C -- Newton 法で方程式 f(x)=0 を解く
 3   *  コンパイル: g++ -o complex-newton complex-newton.C
 4   *  実行: ./complex-newton
 5   */
 6
 7  #include <iostream.h>
 8  #include <complex.h>
 9
10  int main()
11  {
12      int i, maxitr = 100;
13      complex<double> f(complex<double>), dfdz(complex<double>), x, dx;
14      double eps;
15
16      cout << " 初期値 x0, 許容精度ε =";
17      cin >> x >> eps;
18
19      cout.precision(16);
20      for (i = 0; i < maxitr; i++) {
21          dx = - f(x) / dfdz(x);
22          x += dx;
23          cout << "f(" << x << ")=" << f(x) << endl;
24          if (abs(dx) <= eps) break;
25      }
26      return 0;
27  }
28
29  /*
30   * この関数の例は杉原・室田『数値計算法の数理』岩波書店, p.67 による
31   *   f(z) = z^3-2z+2
32   *   1 実根 (≒-1.769292354238631),2 虚根 (≒ 0.8846461771193157 ± 0.5897428050222054)
33   *   を持つが、原点の近くに初期値を取ると、0 と 1 の間を振動する。
34   */
35
36  complex<double> f(complex<double> z)
37  {
38    /* return z * z * z - 2 * z + 2; */
39    return z * (z * z - 2) + 2;
40  }
41
42  /* 関数 f の導関数 (df/dx のつもりで名前をつけた) */
43  complex<double> dfdz(complex<double> z)
44  {
45    return 3 * z * z - 2;
46  }
```

complex-newton 実行結果

```
oyabun% g++ -o complex-newton complex-newton.C
oyabun% ./complex-newton
 初期値 x0, 許容精度ε =(0.07,0.07) 1e-15
f((1.000479890500462,0.01402105439035678))=(0.9998905285720316,0.01405867910352982)
f((0.008690071144917044,0.08327942929892811))=(1.982439704951537,-0.1671175729050391)
f((0.9899874424723547,0.002680506223014176))=(0.9902658531083521,0.002520260799472426)
f((-0.06300187281112724,0.01783194417259117))=(2.125813775994601,-0.03545722093219378)
(中略)
f((-9.992007221626409e-16,0))=(2.000000000000002,0)
f((0.9999999999999999,0))=(1,0)
f((-9.992007221626409e-16,0))=(2.000000000000002,0)
f((0.9999999999999999,0))=(1,0)
f((-9.992007221626409e-16,0))=(2.000000000000002,0)
f((0.9999999999999999,0))=(1,0)
f((-9.992007221626409e-16,0))=(2.000000000000002,0)
f((0.9999999999999999,0))=(1,0)
f((-9.992007221626409e-16,0))=(2.000000000000002,0)
f((0.9999999999999999,0))=(1,0)
f((-9.992007221626409e-16,0))=(2.000000000000002,0)
f((0.9999999999999999,0))=(1,0)
f((-9.992007221626409e-16,0))=(2.000000000000002,0)
oyabun%
```

B.6 DK 法のサンプル・プログラム

B.6.1 ソースプログラム DK3.C

```
 1  // DK3.C -- DKA 法で代数方程式の根を求める
 2  //
 3  //   数学科 6701 号室のワークステーション環境でのコンパイル
 4  //     (グラフィックス・ライブラリィ GLSC を使っているので
 5  //      情報科学センターではコンパイルできない。悪しからず。)
 6  //     g++ -o DK3 DK3.C -I/usr/local/include -lglscd -lX11 -lsocket
 7  //   あるいは
 8  //     ccmg+ DK3.C
 9  //
10  //  Win32 環境下の glscwin でも (若干の修正の下に) 動作させられることが
11  //  分かったので、画面に表示するメッセージを変更しました (2000/10/18)。
12
13  #include <iostream.h>
14  #include <iomanip.h>   // setprecision() のため
15  #include <math.h>
16  #include <complex.h>   // complex<double> のため
17
18  // GLSC のヘッダーファイルをインクルード
19  extern "C" {
20  #define G_DOUBLE
21  #include <glsc.h>
22  }
```

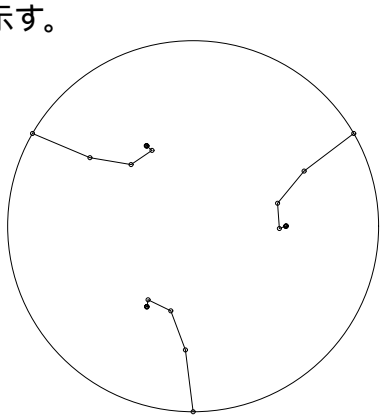
```
23
24  // プロトタイプ宣言
25  complex<double> polynomial(int, complex<double> *, complex<double>);
26  complex<double> bunbo(int, complex<double> *, int);
27
28  // 解きたい代数方程式の次数の最高値
29  #define MAXN 100
30
31  int main()
32  {
33    int i, k, n;
34    complex<double> a[MAXN+1], x[MAXN], newx[MAXN], dx[MAXN];
35    complex<double> g, I(0,1);
36    double r0, R, max, pi;
37
38    // 数学定数（円周率）の準備
39    pi = 4 * atan(1.0);
40
41    // 表\示の桁数を 16 桁にする
42    cout << setprecision(16);
43
44    // 方程式の入力
45    cout << "次数 n を入力してください（1 ≦ n ≦" << MAXN << "): ";
46    cin >> n;
47    if (n > MAXN || n <= 0) {
48      cerr << "次数は" << MAXN << "以下の自然数として下さい。" << endl;
49      exit(0);
50    }
51    for (i = 0; i <= n; i++) {
52      cout << (n-i) << "次の係数を入力してください: ";
53      cin >> a[i];
54    }
55
56    // 多項式を最高次の係数で割って monic にする
57    cout << "monic にします。" << endl;
58    for (i = 1; i <= n; i++)
59      a[i] /= a[0];
60    a[0] = 1;
61    cout << "修正した係数" << endl;
62    for (i = 0; i <= n; i++)
63      cout << "a[" << i << "]=" << a[i] << endl;
64
65    // Aberth の初期値を配置する円の決定
66    g = - a[1] / n;
67    cout << "根の重心" << g << endl;
68    max = 0;
69    for (i = 1; i <= n; i++)
70      if (abs(a[i]) > max)
71        max = abs(a[i]);
72    cout << "max|a_i|=" << max << endl;
73    r0 = abs(g) + 1 + max;
74    cout << "根は重心中心で、半径 r0=" << r0 << "の円盤内にある" << endl;
75
76    cout << "円の半径（分からなければ上の値を指定してください): ";
77    R = r0;
78    cin >> r0;
79    if (r0 > R)
80      R = r0;
```

```
  cout << "図は根の重心を中心として半径 " << R
#ifdef __CYGWIN__
       << "の円が表\示できるようにします。" << endl;
#else
       << "の円が表示できるようにします。" << endl;
#endif

  // Aberth の初期値
  cout << "初期値" << endl;
  for (i = 0; i < n; i++) {
    double theta;
    theta = 2 * i * pi / n + pi / (2 * n);
    x[i] = g + r0 * exp(I * theta);
    cout << x[i] << endl;
  }

  // グラフィックス・ライブラリィ GLSC の初期化
  g_init("DKGRAPH", 120.0, 120.0);
  g_device(G_BOTH);
  // 座標系の指定
  g_def_scale(0,
      real(g) - 1.1 * R, real(g) + 1.1 * R,
      imag(g) - 1.1 * R, imag(g) + 1.1 * R,
      10.0, 10.0, 100.0, 100.0);
  g_sel_scale(0);
  // 線種、マーカーの指定
  g_def_line(0, G_BLACK, 0, G_LINE_SOLID);
  g_sel_line(0);
  g_def_marker(0, G_RED, 2, G_MARKER_CIRC);
  g_sel_marker(0);

  // 初期値と初期値を置く円を描く
  g_circle(real(g), imag(g), r0, G_YES, G_NO);
  for (i = 0; i < n; i++)
    g_marker(real(x[i]), imag(x[i]));

  // 反復
  for (k = 1; k <= 1000; k++) {
    double error;
    cout << "第" << k << "反復" << endl;
    for (i = 0; i < n; i++) {
      dx[i] = polynomial(n, a, x[i]) / bunbo(n, x, i);
      newx[i] = x[i] - dx[i];
      // 図示する
      g_move(real(x[i]), imag(x[i]));
      g_plot(real(newx[i]), imag(newx[i]));
      g_marker(real(newx[i]), imag(newx[i]));
    }
    // 更新
    for (i = 0; i < n; i++) {
      x[i] = newx[i];
      cout << x[i] << endl;
    }
    // 変化量を計算する（ここは非常に素朴）
    error = 0.0;
    for (i = 0; i < n; i++)
      error += abs(dx[i]);
    cout << "変化量=" << error << endl;
```

```
139       // 変化量が小さければ収束したと判断して反復を終了する。
140       if (error < 1e-12)
141         break;
142     }
143
144     // GLSC 終了の処理
145     cout << "終了には" << endl;
146     cout << " グラフィックスのウィンドウをクリック (X の場合)。" << endl;
147     cout << " グラフィックスのウィンドウをクローズ (Windows の場合)。" << endl;
148     g_sleep(-1.0);
149     g_term();
150
151     return 0;
152   }
153
154   // 多項式の値の計算 (Horner 法)
155   complex<double> polynomial(int n,
156      complex<double> *a,
157      complex<double> z)
158   {
159     int i;
160     complex<double> w;
161     w = a[0];
162     for (i = 1; i <= n; i++)
163       w = (w * z + a[i]);
164     return w;
165   }
166
167   //  Π (z i-z j) の計算
168   // j ≠ i
169   complex<double> bunbo(int n,
170         complex<double> *z,
171         int i)
172   {
173     int j;
174     complex<double> w(1,0);
175     for (j = 0; j < n; j++)
176       if (j != i)
177         w *= (z[i] - z[j]);
178     return w;
179   }
```

B.6.2　コンパイル＆実行例

$z^3 - 1 = 0$ を解いてみた例を示す。



```
oyabun% ccmg+ DK3.C
oyabun% ./DK3
次数 n を入力してください (1 ≦ n ≦ 100): 3
3 次の係数を入力してください: 1
2 次の係数を入力してください: 0
1 次の係数を入力してください: 0
0 次の係数を入力してください: -1
monic にします。
修正した係数
a[0]=(1,0)
a[1]=(0,0)
a[2]=(0,0)
a[3]=(-1,0)
根の重心 (-0,-0)
max|a_i|=1
根は重心中心で、半径 r0=2 の円盤内にある
円の半径 (分からなければ上の値を指定してください): 2
図は根の重心を中心として半径 2 の円が表示できるようにします。
初期値
(1.732050807568877,0.9999999999999999)
(-1.732050807568877,1.000000000000001)
(-3.673940397442059e-16,-2)
第 1 反復
(1.196367205045918,0.5944978830179634)
(-1.113033871712585,0.7388354503153702)
(-0.08333333333333359,-1.333333333333333)
変化量=2.015564437074637
第 2 反復
(0.9103764382644928,0.2474704563982415)
(-0.6695039210592528,0.6646738943447257)
(-0.2408725172052399,-0.912144350742967)
変化量=1.34906218596283

   (途中省略)

第 6 反復
(0.9999999987110025,4.085881563506899e-10)
(-0.4999999997093489,0.86602540246384)
(-0.4999999990016535,-0.8660254028724281)
変化量=0.0001103168208393921
第 7 反復
(1,-1.053338202674773e-18)
(-0.5,0.8660254037844386)
(-0.5,-0.8660254037844386)
変化量=4.056615492443548e-09
第 8 反復
(1,0)
(-0.5,0.8660254037844387)
(-0.5,-0.8660254037844387)
変化量=1.665556150858263e-16
終了には
 グラフィックスのウィンドウをクリック (X の場合)。
 グラフィックスのウィンドウをクローズ (Windows の場合)。
oyabun%
```

付 録C　情報処理IIから

C.1　レポート課題

課題 1

色々な方程式を二分法、Newton 法で解いてみよ。初期値の選び方を変えて、収束するかしないか、試しなさい。収束の判定条件には注意を払うこと (特に理由なく低い精度の答を出しただけの場合は減点)。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。Newton 法の収束のための十分条件を本などで探すことができたら、それも説明すること。

ここでは解説しないが、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。

課題 2

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 法で解いて見よ。その結果を C 言語のライブラリィ関数 `sqrt()` や `pow()` [1] で計算した値と比較してみよ。

特に逆関数の計算にも使える (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) ことに注意しよう。

課題 3

C 言語のライブラリィ関数 `asin()`, `acos()`, `atan()` は使わずに、arcsin, arccos, arctan を計算するプログラムを作り、実験せよ。

課題 4

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:

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

[1] `pow(a,b)` で `a` の `b` 乗が計算できる。例えば `pow(2.0, 1.0/3.0)` で 2 の立法根が計算できる。

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

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。

C.2 例題を解くプログラム例

二分法のプログラム bisection.c[2], newton.c[3] は情報処理 II の WWW ページ `http://www.math.meiji.ac.jp/~mk/syori2/` から入手できる。WWW ブラウザー「ファイル・メニュー」の「名前をつけて保存」を用いてセーブする。2003 年度情報科学センターの教育用情報処理室の Windows 環境からは、z: ドライブ (UNIX 環境のホームディレクトリィ) にセーブするのが便利であろう。

C.2.1 Newton 法の場合

1 を初期値として、許容精度 10^{-15} を指示して Newton 法で解かせたのが以下の結果 (入力は `1 1e-15`)。

```
isc-xas06% gcc -o newton newton.c -lm
isc-xas06% ./newton
 初期値 x0, 許容精度ε =1 1e-15
f(   7.503638678402439e-01)=-1.89e-02
f(   7.391128909113617e-01)=-4.65e-05
f(   7.390851333852840e-01)=-2.85e-10
f(   7.390851332151607e-01)= 0.00e+00
f(   7.390851332151607e-01)= 0.00e+00
isc-xas06%
```

注意 **C.2.1** Newton 法の繰り返しを停止させるための良いルールを独力で発見するのはかなり難しい。上の例題プログラムの採用したルールは、多くのプログラムで採用されているやり方ではあるが、いつでもうまく行く方法とは言えない。現時点で「これが良い方法」と見なされている方法は一応あるが、結構複雑なのでここでは説明しない。

C.2.2 二分法の場合

区間 $(0, 1)$ 内に解があることがわかるから、二分法で許容精度 10^{-15} を指示して解かせたのが以下の結果 (入力は `0 1 1e-15`)。関数値が、区間の左端では正、右端では負になったまま区間が縮小して行くのを理解しよう。

[2] `http://www.math.meiji.ac.jp/~mk/syori2/cprog/bisection.c`
[3] `http://www.math.meiji.ac.jp/~mk/syori2/cprog/newton.c`

```
isc-xas06% gcc -o bisection bisection.c -lm
isc-xas06% ./bisection
 探す区間の左端α , 右端β , 許容精度ε =0 1 1e-15
f(   5.000000000000000e-01)= 3.78e-01, f(   1.000000000000000e+00)=-4.60e-01
f(   5.000000000000000e-01)= 3.78e-01, f(   7.500000000000000e-01)=-1.83e-02
f(   6.250000000000000e-01)= 1.86e-01, f(   7.500000000000000e-01)=-1.83e-02
f(   6.875000000000000e-01)= 8.53e-02, f(   7.500000000000000e-01)=-1.83e-02
f(   7.187500000000000e-01)= 3.39e-02, f(   7.500000000000000e-01)=-1.83e-02
f(   7.343750000000000e-01)= 7.87e-03, f(   7.500000000000000e-01)=-1.83e-02
f(   7.343750000000000e-01)= 7.87e-03, f(   7.421875000000000e-01)=-5.20e-03
f(   7.382812500000000e-01)= 1.35e-03, f(   7.421875000000000e-01)=-5.20e-03
中略
f(   7.390851332151556e-01)= 8.55e-15, f(   7.390851332151627e-01)=-3.44e-15
f(   7.390851332151591e-01)= 2.55e-15, f(   7.390851332151627e-01)=-3.44e-15
f(   7.390851332151591e-01)= 2.55e-15, f(   7.390851332151609e-01)=-4.44e-16
f(   7.390851332151600e-01)= 1.11e-15, f(   7.390851332151609e-01)=-4.44e-16
f(   7.390851332151600e-01)= 1.11e-15
isc-xas06%
```

付 録D　多次元の Newton 法の例

非線形の微分方程式を離散化して解く際に Newton 法を使う例。

D.1　ターゲット問題

次の非線形 2 点境界値問題の解を差分法で求める。

$$-u''(x) = u(x)^2 \qquad (x \in (0,1)), \tag{D.1}$$

$$u(0) = u(1) = 0. \tag{D.2}$$

D.2　復習

対応する線形問題は 1 次元 Poisson 方程式の Dirichlet 境界値問題である。

$$-u''(x) = f(x) \quad (x \in (0,1)) \tag{D.3}$$

$$u(0) = u(1) = 0 \tag{D.4}$$

$h = 1/N$, $x_i = ih$, $u_i = u(x_i)$ $(i = 0, 1, 2, \cdots, N)$ とおき、u_i の近似値 U_i を差分法で求めることにする。

$$A = \frac{1}{h^2}\begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}, \quad \vec{U} = \begin{pmatrix} U_1 \\ U_2 \\ \vdots \\ U_{N-1} \end{pmatrix}, \quad \vec{f} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_{N-1} \end{pmatrix}$$

とおくと、差分方程式は

$$A\vec{U} = \vec{f}$$

になる。

D.3　ターゲット問題の差分近似

これは次の方程式で $\vec{U}$ を定めるのが自然だろう。

$$A\vec{U} = \begin{pmatrix} (U_1)^2 \\ (U_2)^2 \\ \vdots \\ (U_{N-1})^2 \end{pmatrix}. \tag{D.5}$$

(D.5) は非線形方程式である

D.4　Newton 法

$$\vec{F}(\vec{U}) := A\vec{U} - \begin{pmatrix} (U_1)^2 \\ (U_2)^2 \\ \vdots \\ (U_{N-1})^2 \end{pmatrix}$$

とおくと、(D.5) は

$$\vec{F}(\vec{U}) = \vec{0} \tag{D.6}$$

となる。

Newton 法とは、初期値 $\vec{U}^{(0)}$ を適当に選んで、後は漸化式

$$\vec{U}^{(k+1)} = \vec{U}^{(k)} - \left(\vec{F}'(U^{(k)})\right)^{-1} \vec{F}(\vec{U}^{(k)}) \tag{D.7}$$

でベクトル列 $\{\vec{U}^{(k)}\}_{k\in\mathbf{N}}$ を定めるというものである。

$$\vec{F}'(\vec{U}) = A - 2\begin{pmatrix} U_1 & & & \\ & U_2 & & \\ & & \ddots & \\ & & & U_{N-1} \end{pmatrix}$$

であるから、ヤコビ行列 $\vec{F}'(\vec{U})$ は三重対角行列である。

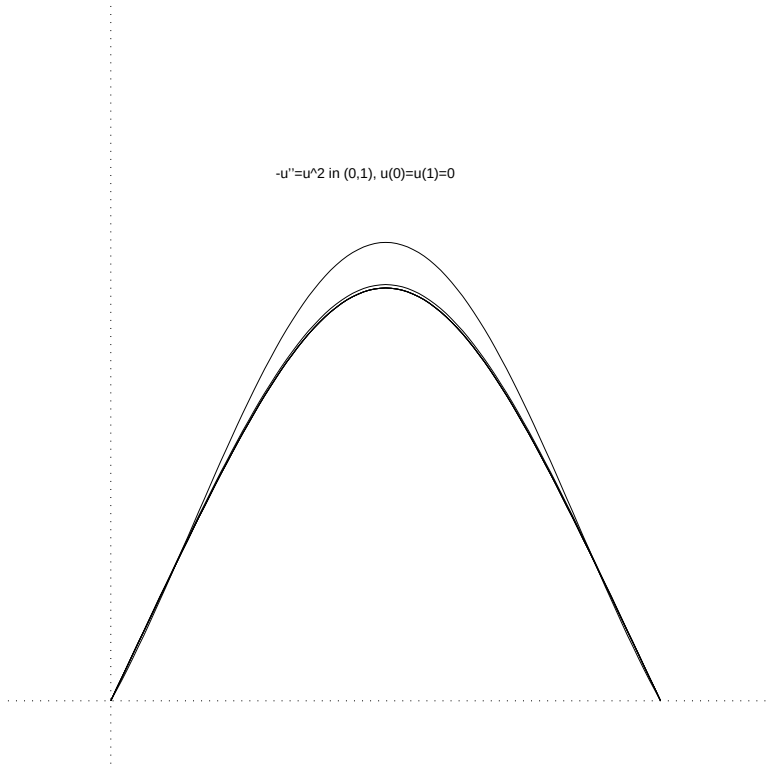
(D.7) において逆行列が現れるが、逆行列を計算せずに、連立 1 次方程式を解く形で計算を遂行すべきことに注意しよう。

D.5　サンプル・プログラム

```
/*
 * Newton.c
 *  非線形 2 点境界値問題
 *    -u''=u^2 in (0,1), u(0)=u(1)=0
 *  を差分法で離散化して得られる非線形方程式を Newton 法で解く。
```

```
 */

#include <stdio.h>
#include <math.h>
#include <matrix.h>
#include "fplot.h"
#include "trid-lu.h"

void mul_mv(int n,
            vector ab,
            vector al, vector ad, vector au,
            vector b)
{
  int i, nm1 = n - 1;
  ab[0]   =                   ad[0] * b[0] + au[0] * b[1];
  for (i = 1; i < nm1; i++)
    ab[i] = al[i] * b[i-1] + ad[i] * b[i] + au[i] * b[i+1];
  ab[nm1]   = al[nm1] * b[nm1-1] + ad[nm1] * b[nm1];
}

double norm(int n, vector x)
{
  return sqrt(dotprod(n, x, x));
}

int main()
{
  int N, i, k;
  vector al,ad,au,akl,akd,aku;
  vector U, x;
  double h, h2, du, H;

  N = 100;
  h = 1.0 / N;
  h2 = h * h;
  al = new_vector(N+1); ad = new_vector(N+1); au = new_vector(N+1);
  akl = new_vector(N+1); akd = new_vector(N+1); aku = new_vector(N+1);
  U = new_vector(N+1);
  x = new_vector(N+1);

  /* 初期値 */
  printf("H (10 位で OK)="); scanf("%lg", &H);
  for (i = 0; i <= N; i++)
    U[i] = H;
  U[0] = U[N] = 0.0;
  /* A */
  for (i = 1; i < N; i++) {
    al[i] = - 1.0 / h2; ad[i] = 2.0 / h2; au[i] = - 1.0 / h2;
  }
  openpl(); fspace2(-0.2, -2.0, 1.2, 20.0);

  fmove(0.3, 15.0);
  label("-u''=u^2 in (0,1), u(0)=u(1)=0");
  linemod("dotted");
  fline(-0.2, 0.0, 1.2, 0.0); fline(0.0, -2.0, 0.0, 20.0);
  linemod("solid");

  for (k = 1; k < 100; k++) {
```

```
    /* A U^k の計算 */
    mul_mv(N - 1, x+1, al+1, ad+1, au+1, U+1);
    /* F(U^k) の計算 */
    for (i = 1; i < N; i++)
      x[i] -= U[i] * U[i];
    /* F'(U^k) の計算 */
    for (i = 1; i < N; i++) {
      akl[i] = al[i]; akd[i] = ad[i] - 2 * U[i]; aku[i] = au[i];
    }
    /* F'(U^k)^{-1} U(U^k) の計算 */
    trid(N-1, akl+1, akd+1, aku+1, x+1);
    /* */
    du = norm(N-1, x+1);
    printf("du=%g\n", du);
    /* U^{k+1} の計算 */
    for (i = 1; i < N; i++)
      U[i] -= x[i];
    /* */
#ifdef NONE
    for (i = 0; i <= N; i++)
      printf("U[%d]=%g\n", i, U[i]);
#endif
    fmove(0.0, U[0]);
    for (i = 1; i <= N; i++)
      fcont(i * h, U[i]);
    if (du < 1.0e-12)
      break;
  }
  {
    double min, max;
    max = U[0]; min = U[0];
    for (i = 1; i <= N; i++) {
      if (U[i] > max)
        max = U[i];
      else if (U[i] < min)
        min = U[i];
    }
    printf("min=%g, max=%g\n", min, max);
  }
  mkplot("Newton.plot");
  closepl();
  return 0;
}
```


-u''=u^2 in (0,1), u(0)=u(1)=0

関連図書

[1] 伊理 正夫, 数値計算, 朝倉書店 (1981).

[2] 伊理 正夫, 藤野 和建, 数値計算の常識, 共立出版 (1985).

[3] 齋藤 友克, 竹島 卓, 平野 照比古, グレブナー基底の計算 実践篇　Risa/Asir で解く, 東京大学出版会 (2003).

[4] 篠原 能材, 数値解析の基礎, 日新出版 (初版 1978, 5 版 1997).

[5] 杉原 正顯, 室田 一雄, 数値計算法の数理, 岩波書店 (1994).

[6] 高木 貞治, 代数学講義 改訂新版, 共立出版 (1965).

[7] 名取 亮 編, 長谷川 秀彦, 櫻井 鉄也, 桧山 澄子, 周 偉東, 花田 孝郎, 北川 高嗣, 数値計算法 第 5 章「非線形方程式」(櫻井 鉄也 執筆), オーム社 (1998).

[8] 一松 信, 数値解析、朝倉書店 (1982).

[9] 一松 信, 数学とコンピュータ, 共立出版 (1995).

[10] 増田 久弥, 非線型数学, 朝倉書店 (1985).

[11] 室田 一雄, 平野の変形 Newton 法の大域的収束性, 情報処理学会論文誌, Vol.21 (1980), pp.469–474.

[12] 森 正武, 数値解析, 共立出版 (1973).

[13] 森口 繁一, 数値計算術, 共立出版 (19??).

[14] 森口 繁一, 数値計算工学, 岩波書店 (1989).

[15] 山本 哲朗, 数値解析入門 [新訂版], サイエンス社 (2003).
(1976 年初版発行の定番本の待望の改訂版)

[16] 山本 哲朗, Newton 法とその周辺, 数学, Vol.37 (1985), pp.1–15.

[17] 山本 哲朗, 北川 高嗣, 数値解析演習, サイエンス社 (1991).

[18] 渡部 力 (わたなべ つとむ), 名取 亮 (まこと), 小国 力 (おぐに つとむ) 監修, Fortran77 による数値計算ソフトウェア 第 11 章「代数方程式」 (小野寺 嘉孝 執筆), 丸善 (1989).

[19] 山崎 圭次郎, 基礎代数, 岩波講座 応用数学, 岩波書店 (1994).

[20] M.A.Jenkins and J.F.Traub, Commun. ACM, 15 (1972), pp.97–99.

[21] A.Ralston and P.Rabinowitz, A First Course in Numerical Analysis, McGraw-Hill, second edition (1978).

[22] L.B.Rall, Computational Solution of Nonlinear Operator Equations, John Wiley, New York (1969).

[23] L.Schwartz, 解析学 I, 東京図書 (19??).

[24] E.Zeidler, Nonlinear Functional Analysis and Its Applications I, Springer–Verlag Berlin, 1986.