応用数理実験

桂田 祐史

2001年11月15日

目 次

第 1 章　計算機における数の表現 **4**
1.1 はじめに 4
1.2 予備的な知識 4
1.3 整数 5
1.4 浮動小数点数 (floating-point numbers) 6

第 2 章　連立 1 次方程式に対する直接法 **11**
2.1 序 (線形代数の復習) 11
2.2 Gauss の消去法 — 計算量の観点から 12
2.2.1 計算量についての準備 12
2.2.2 色々な解き方の比較 13
2.3 Gauss の消去法の丸め誤差解析 14
2.3.1 記号、言葉 14
2.3.2 丸め誤差解析の要約 14
2.3.3 ベクトルと行列のノルム 15
2.3.4 行列の条件数 17
2.4 参考書 18

第 3 章　連立 1 次方程式に対する CG 法 **19**

第 4 章　固有値問題 **20**
4.1 一般的な注意 20
4.1.1 問題の定式化 20
4.1.2 線形代数の復習 21
4.1.3 どう立ち向かうべきか 21
4.2 解法についての概観 22
4.2.1 相似変換 22
4.2.2 Jacobi 法 — 温故知新 22
4.2.3 巾乗法 (power method) 24
4.2.4 固有ベクトルから固有値を求める方法 24
4.2.5 逆反復法 25
4.2.6 シフト法 26
4.3 三重対角化の手法 26
4.3.1 Householder 法 26
4.3.2 Lanczos(ランチョス) 法 30
4.4 二分法 30

4.4.1 伝統的な説明 . . . 31
4.4.2 Sylvester の慣性律による説明 . . . 32
4.5 QR 法 . . . 33
4.5.1 正則行列の QR 分解 . . . 33
4.5.2 QR 変換 . . . 35
4.5.3 QR 法の歴史 . . . 39
4.6 特異値 . . . 39
4.7 固有値計算のためのパッケージ . . . 39
4.8 参考書 . . . 39
4.9 収束の速さ . . . 39

第 5 章 常微分方程式の初期値問題 40
5.1 序 . . . 40
5.2 初期値問題の設定 . . . 40
5.3 常微分方程式の初期値問題の復習 . . . 41
5.3.1 数学理論 . . . 41
5.3.2 数値解法 . . . 42
5.4 基本的な用語 . . . 42
5.5 典型的なスキーム . . . 44
5.5.1 Runge-Kutta 法とその一族 . . . 44
5.5.2 線形多段法 . . . 47
5.5.3 その他の方法 . . . 48
5.6 数値的安定性 . . . 48
5.7 Stiff problem (硬い問題) . . . 50
5.8 参考書 . . . 51
5.9 おまけ — 実際的な誤差の推測 . . . 52
5.10 常微分方程式の初期値問題 補足 . . . 53
5.10.1 数値解法の次数 (order) . . . 53
5.10.2 スキームの数値的安定性 . . . 54

第 6 章 今後の執筆予定？ 55
6.1 代数方程式 . . . 55
6.2 計算機代数 . . . 55
6.3 FFT . . . 55
6.4 初等関数の計算法 . . . 55
6.5 特殊関数 . . . 55

付 録 A Strum の方法 56
A.1 スツルムの定理 . . . 56
A.2 ユークリッドの互除法による Strum 列の生成 . . . 59
A.3 3 重対角行列の固有多項式と Strum 列 . . . 60
A.4 直交多項式の作る Strum 列 . . . 63
A.5 一般化された Strum 列 . . . 64

付 録B C 言語と行列 **66**
B.1 はじめに 66
B.2 1 次元配列の方法 66
B.3 ポインター配列の方法 67
B.4 二つの方法の優劣 68
B.5 サンプル・プログラム 68

第1章　計算機における数の表現

1.1　はじめに

計算機の中では、数をどのように表現しているのか？

> 「実数体系という名称はあまり適切とは思えないが、微積分をはじめ高等な解析学の基盤をなしているのがこの実数体系である。そのために、すべての実数を現実の有限な計算機で表現することが不可能なことをつい忘れがちになる。しかし、実数体系が理論的解析をやさしくしている部分は、実際の計算では逆にそれ無しでやってゆかなければならないのである。」 — Forsythe (森 正武 訳)

ここで FORTRAN や C 等のプログラミング言語であらかじめ用意されているデータ型について列挙してみよう。

FORTRAN intger, real, real*8, real*16, complex, complex*16, complex*32

C short, unsigned short, int, unsigned int, float, double, long double

色々あるが、いずれもあらかじめ定まった量の記憶域を用いて数を表現することに注意しよう。従って**有限個の数しか表現することはできない**。

これらのデータ型は大きく二つに分類することが出来る。整数のみを表現できるものと、それ以外のもの — 実数（あるいは複素数）を表現出来る — ものである。

（多くの Lisp や数式処理言語では、bignum とか「無限多倍長」と呼ばれるデータ型があり、これは数を表すために用いる記憶域の大きさが動的に変化する。しかし、この場合も表す数の個数は有限個であることは変わらないし、また数表現の基本的な考え方は後述のものと同じである。）

1.2　予備的な知識

現在の（ほとんど）すべての電子計算機の内部では、データは 0 または 1 の値を取りうる数（**ビット**bit と呼ぶ）の列として表される（要するにデジタルということ）。これが「計算機内部では数は 2 進法だ」と言われる由縁である。

数を 2 進表示すると、桁数が多くなって人間には読みにくいので、4 桁ずつまとめて **16 進法**にすることが行われている。表記に使う文字は 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F の 16 文字である。（一部のシステムでは 3 桁ずつまとめた **8 進法**も使われる。）

8 ビットを**バイト**(byte) と呼ぶことが多い（過去には 8 ビットでない 1 バイトが存在したが）。

実際の計算機では適当な個数のビットをまとめて処理することが多い。いわば 2 進法の算盤（そろばん）が並んでいるようになっている。例えばパソコンや多くのワークステーションでは主記憶装置はバイト単位に番地をつけられて、読み書きの最少単位はバイトである。また **CPU**（central

processing unit, 中央演算装置）には**レジスター（置数器）**と呼ばれる演算用の算盤があるが、この桁数をそのマシンのビット数ということがある（ビット数は現在では 8,16,32,64 のような 2 のべきであることが多い)。例えば SPARCstation は 32 ビット・マシンである。このように CPU のレジスターに収められるサイズのデータはその計算機に取って最も基本的なデータの取り扱い量であるとみなすことが出来る。そのため、その量のことをその計算機の**ワード** (**word**) と呼ぶこともある。例えば、SPARCstation では 1 word = 32 bits である。

1.3 整数

整数[1]は数値シミュレーションではデータとして使われることは滅多にないが、簡単に解説しておく。

m ビット・マシンでは、m ビットのデータ (ワード) を標準の整数データを表現するために使うことが多い。m ビットで 2^m 個のデータを表現することが出来る ($m = 16$ では 65536 個、$m = 32$ では 4294967296 個)。正の数だけあれば間に合うならば、ごく自然に $0, 1, \cdots, 2^m - 1$ の数を対応させることが出来る (実際 $b_{m-1}, b_{m-2}, \cdots, b_1, b_0$ を m ビットのデータとして、$N = 2^{m-1}b_{m-1} + \cdots + 2^k b_k + \cdots + 2^2 b_2 + 2b_1 + b_0$ を対応させればよい)。C 言語で “unsigned (符号無)” を冠したデータ型はまさにこれである。

しかし大抵は負の数をも必要とする。素朴に考えると、1 ビット (例えば b_{m-1}) を符号を表すのに用いて、残りの $m-1$ ビットで絶対値を表すようにする方法が思い浮かぶ。これは**絶対値表示**と呼ばれ、ある程度使われて来た。しかし現在ポピュラーな方法は**補数表示**と呼ばれる方法である。大抵は **2 の補数表示**であるが、1 の補数というものもある。

$b_{m-1}b_{m-2}\cdots b_1 b_0$ というビットの列は 1 の補数表示では次のように解釈される。

- $b_{m-1} = 0$ ならば $N = b_0 + 2b_1 + 2^2 b_2 + \cdots + 2^k b_k + \cdots + 2^{m-2}b_{m-2}$. $(0 \le N \le 2^{m-1} - 1$ となる)
- $b_{m-1} = 1$ ならば $N = -(c_0 + 2c_1 + 2^2 c_2 + ... + 2^k c_k + ... + 2^{m-2}c_{m-2})$. ただし $c_i = 1 - b_i$ とした。 $(-2^{m-1} + 1 \le N \le 0$ となる)

要するに 1 の補数表示は本質的には絶対値表示とあまり変わらない。

例　8 ビットの場合、-1 を表すには 00000001 の各桁を反転して 11111110

2 の補数表示では、ビット列 $b_{m-1}b_{m-2}\cdots b_1 b_0$ の表す数は

$$N = b_0 + 2b_1 + 2^2 b_2 + \cdots + 2^k b_k + \cdots + 2^{m-2}b_{m-2} - 2^{m-1}b_{m-1}$$

とする。$-2^{m-1} \le N \le 2^{m-1} - 1$ となる。

2 の補数表示を用いる理由は、符号無しの場合の演算をする場合と回路が共通化出来ること、複数のワードを用いて広い範囲の数を表現する**多倍長整数**の演算の実現が簡単なことである。

演算をすることによって、結果が m ビットの範囲に収まらなくなった場合は、とにかく下位 m ビットは残し、溢れ出たものは適当に処理する (ことが多い)。加算、減算ならば溢れる

[1]いわゆる「固定小数点数」もこの範疇に含まれると思って良い。

のは 1 ビットなので、CPU に備わっているフラグに記憶する。乗算の場合は、溢れたものを捨ててしまうか、別のレジスターに収めることにする。加減算についてはデータを符号無しと考えても、符号有りと考えても、結果の下位 m ビットは同じものになることに注意。10 進数で説明すると

	9	9	9		9	9	9	-1
	9	9	8		9	9	8	-2
1|	9	9	7	1|	9	9	7	-3

m ビット・マシンで 2 の補数表示を使うのは、加減乗算については、2^m を法とする剰余系を考えて、代表元として $-2^{m-1}, -2^{m-1}+1, \cdots, -1, 0, 1, 2, \cdots, 2^{m-1}$ を取っているとして理解できる。

これに対して割算については、正数同士の場合はあまり問題がないが、負数を含む演算の結果についてはあまり合理的な説明は出来ない。割り切れない場合、剰余の符号をどう取るか、それに関連して商 (整数) をどうするか。($-7 \div 2$ は "-3 余り -1" なのか、 "-4 余り 1" なのか。割る数×商＋剰余が割られた数にならない場合もある（ひどい)。)

1.4 浮動小数点数 (floating-point numbers)

> 「実際の計算機で浮動小数点表現が実現される形は、ここで議論する理想的なものとは異なるかもしれない。しかしその差異は小さく、**丸め誤差**という基本的な問題を扱う場合には常にほとんど無視することが出来る」 — Forsythe

数値計算で扱う数の範囲は極めて広い。整数や固定小数点数では不便である (ENIAC を開発した von Neumann はそれくらい何でもないと思ったらしいが)。科学で現れる数の表現には指数形式と呼ばれるものがあるが、これを取り入れたものが、以下に説明する**浮動小数点数**である。

以下に見るように、浮動小数点数の体系は数学的にはあまりきれいなものではなく、実用的な観点からも完璧なものからはほど遠いものである。

この体系とつき合うには、物理や化学の実験でおなじみの有効数字の概念を勉強した時の感覚が多少は役に立つが、かなり独特のものがあると思う。

基数 β を固定する時、0 でない任意の実数 x は

$$(*) \qquad x = \pm\left(\frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \frac{d_3}{\beta^3} + \cdots\right) \times \beta^m$$

の形に表現することが出来る。ただし、$d_1, d_2, d_3, \cdots$ は

$$0 < d_1 \leq \beta - 1, \quad 0 \leq d_k \leq \beta - 1 \quad (k = 2, 3, \cdots)$$

を満たす整数である。($x = \pm\alpha \times \beta^m, \quad 1/\beta \leq \alpha < 1$, m は整数)

そこで計算機の内部では上式を有限項で打ち切った

$$\pm\left(\frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \frac{d_3}{\beta^3} + \cdots + \frac{d_n}{\beta^n}\right) \times \beta^m$$

で表現する。ただし m の範囲にも制限がつく： $-m_{_L} \leq m \leq m_{_U}$。

これを β 進 n 桁浮動小数点表示と呼び、この形に表現される数を浮動小数点数 (floating-point numbers) という。

現在の実際の計算機で採用されている基数は、主として 2, 10, 16 である (パソコン、ワークステーションでは 2、スーパーコンピューターを除く大型機では 16 が多い)。

通常は最上位の桁 d_1 は (上に書いたように) 0 にならないようにしておく。この条件を満たしている表示を正規化 (normalize) された浮動小数点表示という。

$$f = \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \frac{d_3}{\beta^3} + \cdots + \frac{d_n}{\beta^n}$$

を仮数部 (mantissa) または小数部 (fraction)、β^m を指数部、m を指数 (exponent) という。

0 は浮動小数点表示においては特別扱いする。通常はその仮数部を all zero ($d_1 = d_2 = \cdots = d_n = 0$)、指数 $m = -m_L$ とすることにより表現する。

条件 $x_1 \neq 0$, $-m_{_L} \leq m \leq m_{_U}$ を満たす全ての $(*)$ に、この 0 を加えたもので一つの浮動小数点の体系 F が出来る。

例 (ミニ浮動小数点体系)． $\beta = 2, n = 3, m_{_L} = 1, m_{_U} = 2$ とすると？是非各自で F の要素を書き出して、数直線上にメモって欲しい。(まず原点に関して対称。最大の絶対値 $(1/2+1/4+1/8)2^2 = 7/2$. 0 でない最小の絶対値 $(1/2)2^{-1} = 1/4$. 各々の区間 $[1/4, 1/2), [1/2, 1), [1, 2), [2, 4)$ では等間隔。)

数直線上に表してみると、左右に広漠な空きがあること、またゼロの付近もひどく空いていることに気付く。これは一般的な特徴である。

実際の計算機では、符号、仮数部、小数部のそれぞれを 2 進数 (ビットの列) で表すことになる ($\pm\beta^n f$ が前節で説明した方法で表現される)。

SPARCstation 上の FORTRAN では、単精度 (32bits) の場合、符号 1 ビット、指数部 7 ビット、仮数部 24 ビット (ただしいわゆるケチ表現をしている)。倍精度の場合は、符号 1 ビット、指数部 10 bit、仮数部 53 bit である。

問題点

(i) 絶対値が極端に大きい数、極端に小さい数は表現できない。入力が出来ないのはもちろんであるし、 演算の結果がそうなることもある（オーバーフロー (overflow)、アンダーフロー (underflow))。

(ii) F は有限集合なので、ほとんどすべての数は近似的にしか表現できない。入力の際も演算の際も近似をする必要がある。例. $\beta = 1/2$ では $\frac{1}{10}$ すら表現できない。

$$\frac{1}{10} = (0.0001100110011001100\cdots)_2$$

(iii) 演算に伴う丸め誤差, 情報落ち, 狭い意味の丸め（演算の結果は F に属さないので適当に丸める)。

(iv) 桁落ち

大雑把に言って、**浮動小数点数の体系は相対誤差を一定量で押えるようになっている**。 F に属する数の絶対値の最大値を U, $F \setminus \{0\}$ に属する数の絶対値の最小値を L とし、$D(F) = \{x; L \leq |x| \leq U\}$ とおく。$x \in D(F)$ に対し、

$fl(x) =$ “x に最も近い F の元（2 つある時はどちらか一方を適当なルールで決める）”

とおく。$fl(x)$ は x を近似的に表現する F の要素で最良のものだと考えられる。このとき

$$\left| \frac{fl(x) - x}{x} \right| \leq \frac{1}{2} \beta^{1-n}$$

演算の際の丸めについて。 $x, y \in F$ としても $x + y \in F$ であるとは限らない。計算機の中の加算 $\oplus$ はうまく実現した場合で

$$x \oplus y = fl(x + y)$$

が成立する。 $x \oplus y$ と $x + y$ の差を浮動小数点加算 $\oplus$ で生じた丸め誤差と呼ぶ。

$\oplus$ はあまり良い性質を持たない。（F が体のような代数系にならないのはまあ仕方ないにしても）可換則は OK であるが、結合則や分配則は成立しない。このことは誤差の解析を難しくする原因になっている。

例. $s_N = \sum_{n=1}^{N} \frac{1}{n}$ を計算することを考える。$N = 100000$ とした計算例。

```
oyabun% test1
s=1.209085083007812e+01          n が小さい方から足していった（単精度）
s=1.209015274047852e+01          n が大きい方から足していった（単精度）
s=1.209014612986334e+01          n が小さい方から足していった（倍精度）
S=1.209014612986341e+01          n が大きい方から足していった（倍精度）
```

consider $1.0 + 0.010 + 0.010 + 0.010 + 0.010 + 0.010 + 0.010 + 0.010 + 0.010 + \cdots + 0.010$

これは情報落ち（桁揃えの段階で仮数部が捨てられる）と呼ばれる現象。

計算機イプシロン (machine epsilon)　計算機の精度を特徴付ける量として、計算機イプシロンと呼ばれるものがある。普通、$1 + \varepsilon$ が 1 より大きくなるような ε のうちで最小のもの、と定義される:

$$\varepsilon_M \stackrel{\text{def.}}{=} \min\{\varepsilon \in F; 1 \oplus \varepsilon > 1\}.$$

厳密な値を知らなくても、大体の値が分かれば十分。次のような FORTRAN プログラムの断片で調べることが出来る。

```
      EPS = 1.0
 10   EPS = EPS * 0.5
      EPSP1 = EPS + 1.0
      IF (EPSP1 .GT. 1) GOTO 10
```

演習 自分が使える言語処理系で machine epsilon を求めよ。単精度、倍精度、両方求めること。

その他の実数表現法

上で説明した浮動小数点数以外の実数表現法も提唱されている。特に溢れを回避するための工夫として、指数部可変長の形式（松井正一、伊理正夫)、URR（universal representation of real numbers, 浜田穂積）などがある。

計算結果の精度保証をするための区間演算と、それをサポートするための実数表現法も重要な話題である。

IEEE Standard 754

浮動小数点数については、IEEE（アメリカ電気電信技術者協会のこと）の規格が、有名かつ重要である。特に 1985 年に公表された P754 という 2 進浮動小数点数演算規格は、現在のパソコン、ワークステーションのマイクロプロセッサーに広く採用されているので、特に微妙な計算を必要とする場合は、規格を学ぶ価値がある。この規格では、∞ 等の NAN (not-a-number, 非数)、不正規化数の導入、例外処理等の工夫がある。（もちろんデータの可搬性も保証される。）

IEEE 単精度 指数 8, 仮数 24 最小の非正規数 1.401e-45 最小の正規数 1.175e-38 最大の数 3.403e+38 b31（符号 s）, b30..b23（指数 e）,b22..b0（仮数部 f）(implicit MSB があるので、仮数部は実質 24 bits) 正規数値 ($0 < e < 255$)

$$(-1)^s \times 2^{e-127} \times 1.f$$

非正規数値 ($e = 0$)

$$(-1)^s \times 2^{e-126} \times 0.f$$

符号つきの 0 ($e = 0$)

$$(-1)^s \times 0$$

符号つきの ∞ ($e = 255$)

$$s = u; e = 255; f = .000 \cdots 000$$

非数 ($e = 255$) シグナルを発生するもの

$$s = u; e = 255; f = .0uuu \cdots uuu \quad \text{少なくとも 1 bit は 0 でない}$$

シグナルを発生しないもの

$$s = u; e = 255; f = .1uuu \cdots uuu$$

IEEE 倍精度 指数 11, 仮数 53 最小の非正規数 4.941e-324 最小の正規数 2.225e-308 最大の数 1.798e+308 b63（符号 s）, b62..b52（指数 e）,b51..b0（仮数部 f）(implicit MSB があるので、仮数部は実質 53 bits) 正規数値 ($0 < e < 2047$)

$$(-1)^s \times 2^{e-1023} \times 1.f$$

非正規数値 ($e = 0$)

$$(-1)^s \times 2^{e-1022} \times 0.f$$

符号つきの 0 ($e = 0$)

$$(-1)^s \times 0$$

符号つきの ∞ ($e = 2047$)

$$s = u; e = 2047; f = .000 \cdots 000$$

非数 ($e = 2047$) シグナルを発生するもの

$$s = u; e = 2047; f = .0uuu \cdots uuu \quad \text{少なくとも 1 bit は 0 でない}$$

シグナルを発生しないもの

$$s = u; e = 2047; f = .1uuu \cdots uuu$$

NaN – Not a Number の略。日本語では非数と訳される。

denormalized number （非正規数）subnormal number の古い言い方。

subnormal number (非正規数) この規格では、下駄ばき表現の指数が 0 になっている、ゼロでない浮動小数点数のこと。

exception（例外）算術例外とは、ある算術演算を試みた時に、一般的に受け入れられる結果が生成されないことを意味する。

gradual underflow （漸近的アンダーフロー）浮動小数点演算がアンダーフローした時、0 の代わりに非正規化数を返すこと。

	大きい数×大きい数	+Inf	オーバーフロー
	大きい数× (-大きい数)	-Inf	オーバーフロー
	正数/0.0	+Inf	0 除算
IEEE 例外	負数/0.0	-Inf	0 除算
	0.0/0.0	NaN	演算不可能
	小さい数/大きい数	非正規化数	アンダーフロー
	2.0/3.0	丸めが起こる	結果不正確

IEEE 拡張倍精度 指数 15, 仮数 64

IEEE 4 倍精度 指数 15, 仮数 113 最小の非正規数 6.475e-4966 最小の正規数 3.362e-4932 最大の数 1.190e+4932

第2章　連立1次方程式に対する直接法

$N \in \mathbf{N}$, $K = \mathbf{R}$ or $\mathbf{C}$, $A \in M(N;K)$, $b \in K^N$ とするとき[1]、未知ベクトル x に関する方程式

$$Ax = b$$

を連立 1 次方程式と呼ぶ。これを解くためには、色々な方法があるが、

1. 直接法 — (丸め誤差がなければ) 有限回の四則演算で、正確な解が得られる方法
2. 反復法 — 有限回の演算では正確な解を得られないかも知れないが、十分精度がよくなることは保証されていて、そこで打ち切った近似解で満足する、という方法

の二つに大きく分類される。ここでは、直接法の代表的なアルゴリズムである Gauss の消去法に焦点を当てて、考察する。

2.1　序 (線形代数の復習)

1. 逆行列があれば解決。

$$\det A \neq 0 \Leftrightarrow A^{-1} \text{ 存在、}$$
$$\text{このとき } \forall b \text{ に対して、一意可解で } x = A^{-1}b.$$

(B を A の余因子行列とするとき、

$$A\,{}^tB = {}^tB\,A = \det A \cdot I$$

であることが示される[2]。これから分かる。)

2. 有限回の四則演算で解は求まる。例えば、
   - Cramer の公式
   - (Jordan の) 消去法 （掃き出し法）

しかし、実際上はこれだけでは不満がある。問題点として

(1) 効率上の問題。「大きい問題を速く解きたい!」
   (a) (時間) 計算量
   (b) 空間計算量

(2) 精度 (丸め誤差) の問題。精度保証の問題。「実際に得た解の精度を良くしたい、見積もりたい」

[1] $\mathbf{N}$ = 自然数全体の集合、$\mathbf{R}$ = 実数全体の集合、 $\mathbf{C}$ = 複素数全体の集合。
[2] ${}^tB$ は B の転置行列を表す。本によっては B^T と書いてあるものも多い。

2.2 Gauss の消去法 — 計算量の観点から

ごく一般的な問題を解くには、Gauss の消去法を使うのがよい。それはなぜか?、がテーマ。

2.2.1 計算量についての準備

ここでは、時間計算量に関しては、乗除算の回数で測ることにする[3]。

例 2.2.1 線形計算で基本的と思われる演算における計算量の見積り:

1. N 次正方行列と N 次元ベクトルのかけ算

$$A, x \longrightarrow Ax \quad N^2 \text{ 回}$$

2. 二つの N 次正方行列のかけ算（ごく普通にやった場合[4]）

$$A, B \longrightarrow AB \quad N^3 \text{ 回}$$

ここで、クイズ。 ABx を計算する場合は?

注 計算量の議論をする時は、ごく粗いオーダーを示すだけのときが多い。例えば、正確に数えて $N^3/3 + N^2/2 + 2$ 回である場合、低次の項を無視して、$N^3/3$ 回と言ったり、Landau の記号を用いて、$O(N^3)$ である、と言ったりする。

空間計算量とは、要するに、どれだけのメモリーが必要になるか、である。プログラムがどの程度の量のメモリーを使用するか、

- 必要とあれば、見積もれるようになっておかなければならない。
- 新しいプログラムで計算する時、あるいは同じプログラムを使い続ける場合でも、これまでとは大きくパラメーターを変える時、必要とするメモリー量を見積もるべきである。

今回のような問題では、行列を記憶するのに必要とするメモリーの量だけ計算すれば、普通は十分である。N 次正方行列には、N^2 個の成分がある。通常、倍精度浮動小数点数は 1 つ 8 バイト (1 バイトとは、通常 8 ビットのことを指す。以下では「バイト」を “B” と略記する。) だから、$8N^2$ B 必要になる。例えば $N = 1000$ ならば、8×1000^2 B = 約 8 MB 必要になる[5]。

問 普段使用しているシステムで、N 次正方行列が、無理無く記憶できるのは、N がどれくらいまでであるか、概算せよ (パソコン、ワークステーション)。

---

[3] このことの妥当性についての議論には、あまり深入りしない。

[4] 実は、行列のかけ算に関しては、Strassen の算法というものがあり、N^3 よりも低いオーダーの計算量で済むことが知られている。しかし、その方法が効力を発揮するのは、N がかなり大きい時であること、実際に行列のかけ算が必要になることはあまりないことから、ここでは、詳しく述べない。

[5] M = K^2, $K = 1024 = 2^{10}$. ちなみに G=K^3, T= K^4.

2.2.2 色々な解き方の比較

逆行列なんか、誰も使わない。。。

1. 普通の掃き出し法 (Jordan の消去法)。

$$Ab \longrightarrow \cdots \longrightarrow \begin{pmatrix} 1 & & * \\ & \ddots & \vdots \\ & & 1 \quad * \end{pmatrix}$$

   これは乗除算約 $N^3/2$ 回が必要。

2. Cramer の公式。$N+1$ 個の行列式が現れる。普通に計算すると $O(N^4)$ 回の乗除算。丸め誤差の観点からも損。

3. A^{-1} を掃き出し法で求めてから、$A^{-1}b$ を計算する。N^3 回の乗除算が必要になり、損。他の理由もあってダメ。

4. Gauss の消去法。$N^3/3$ 回の乗除算で OK.

Gauss の消去法には、上に述べた以外にも、大きな長所がある。それは、係数行列 A が疎(そ)である場合は、その性質を保ったまま計算しやすく、従って問題がさらに効率的に解ける、ということである。例えば、A が三重対角行列である場合、逆行列を計算するのはやはり $O(N^3)$ であるが、Gauss の消去法を用いると、$O(N)$ で済む。

もしかすると、逆行列のファンである人（特に学生）は多いかもしれない。大昔は、学生でなくても、連立一次方程式を解くには、まず逆行列を求めようとする人が多かった — 現在、こんなことをしたら、研究者・技術者として、化石扱いにされるのがオチである。

Gauss の消去法による、連立一次方程式の解法は、実際には LU 分解の形で扱われることが多い。まず、前進消去の段階では、$Ax = b$ を表す

$$[A|b]$$

から始めて、行に関する基本変形を繰り返すことによって、対角線よりも下の部分に 0 が並ぶ形を導く。これはある方程式を表現しているが、それを $Ux = c$ と書くと、U は上三角行列である。Gauss の消去法の後半部分である、後退代入の原理は、「上三角行列を係数行列にもつ連立一次方程式は、簡単な代入操作で解ける」ということである。

実は、基本変形は、左から基本行列と呼ばれる行列を掛けることに他ならないが、今の場合は、ある下三角行列 L が存在して、

$$U = LA, \quad c = Lb$$

となる。

L, U を記憶しておくと、別の b' が与えられた時に、$Ax = b'$ の解を

$$U^{-1}Lb'$$

として計算できる。

2.3 Gauss の消去法の丸め誤差解析

数値解析の黎明期には重要視された問題で、今でもアウトラインを知っておくべきものではあるが、「最重要なもの」、「数値解析理論の典型」であるとは考えて欲しくない。

2.3.1 記号、言葉

以下では $K = \mathbf{R}$ または $K = \mathbf{C}$, また N は自然数とする。

$$\begin{aligned} M(K;N) &= K \text{ の要素を成分とする } N \text{ 次正方行列全体の集合,} \\ GL(K;N) &= K \text{ の要素を成分とする } N \text{ 次正則行列全体の集合} \end{aligned}$$

$A \in GL(K;N)$, $b \in K^N$ に対して、連立 1 次方程式

$$Ax = b$$

を浮動小数点数を用いた計算機上で「解く」と、丸め誤差のために x そのものは得られず、それとは異なる $x_* \in K^N$ が得られる。このとき $x - x_*$ を (絶対) 誤差、$b - Ax_*$ を (絶対) 残差と呼ぶ (「誤差」= "error", 「残差」= "residual").

2.3.2 丸め誤差解析の要約

未知数の個数 N がさほど大きくない場合に、Gauss の消去法に代表される直接法は、連立 1 次方程式を解くための有力な手段である。

ところで、現在の計算機で普通に計算する場合、有限桁の演算精度しか持っていないことに起因する丸め誤差が生じることを避けられない。すると

- なるべく高精度の「解」を得たい（その方法を知りたい）
- 実際に得られる「解」の精度を見積もりたい（その方法を知りたい）

のような要求が出てくることになる。(ここで、かっこ「」をつけて「解」としたのは、近似的な解でしかないということを表したかった。)

消去法の際に生ずる丸め誤差を解析すると、実際には大変うまく行くことが多いのに、およそ現実とはかけはなれた、悲観的にならざるを得ない、と主張する研究結果が出た。

> von Neumann and Goldstein (1947) — 素朴な (前進) 誤差解析で Gaußの消去法を解析した。これを信じると 100 元の方程式を解くのは絶望的と考えられるが、実際には楽楽解けてしまう。

この状況を解決したのが、J.H.Wilkinson[6] の研究である。彼は**後退誤差解析**の手法を用いて、連立 1 次方程式を解く過程を徹底的に解析して、この問題にケリをつけた。彼の得た重要な結論の一つは

[6]Rounding errors in algebraic process, 1963

ピボットの部分選択法を採用した Gauss の消去法では、ほとんどすべての場合に[7]、
相対残差が小さくなることが保証される。例えば:

$$\frac{\|b - Ax_*\|}{\|A\|\|x\|} \leq \rho\beta^{-n},$$

ただし用いた浮動小数点数の体系は β 進 n 桁であるとした。

というものである。この稿では、これを認めることにして、後はなるべく自前で議論することにしよう。

要点は、

（残念ながら）残差が小さくても、誤差も小さいとは保証されない！

係数行列の条件数が小さければ、誤差が小さくなることが保証される。

の二点である。

後退誤差解析とは　連立 1 次方程式 $Ax = b$ を浮動小数点演算で「解く」と、真の解の代わりに、x に「近い」x_* が得られるが、これは元の問題を摂動した問題、例えば $(A + \Delta A)x_* = (b + \Delta b)$ の正確な解になっているとみなすことができる。こうして

- どういう摂動問題の解になっているか
- 摂動問題の解になっているのならば、真の解をどの程度近似していることになるか

と考察が分解されることになる。以下の節では、この後者の部分を扱うことになる。

2.3.3　ベクトルと行列のノルム

定義 2.3.1 (ベクトルのノルム) $\|\cdot\|$ が K^N のノルムとは、写像

$$\|\cdot\|: K^N \ni x \longmapsto \|x\| \in \mathbf{R}$$

で以下の条件を満たすもののこと。

(i) $\forall x \in K^N \quad \|x\| \geq 0$, そして $\|x\| = 0 \Leftrightarrow x = 0$.

(ii) $\forall \lambda \in K, \forall x \in K^N \ \|\lambda x\| = |\lambda|\|x\|$.

(iii) $\forall x \in K^N, \forall y \in K^N \ \|x + y\| \leq \|x\| + \|y\|$.

例 2.3.1

$$\|x\|_p = \left(\sum_{i=1}^{N} |x_i|^p\right)^{1/p} \quad (1 \leq p < \infty), \quad \|x\|_\infty = \max_{i=1,2,\cdots,N} |x_i|$$

[7] この気持ちの悪い言いまわしが気になる人もいるだろうが、浮動小数点数の体系は、オーバーフローやアンダーフローを始めとする破局的なエラーが起こり得るものだから、ある程度までは仕方がない。

特に

$$\|x\|_1 = \sum_{i=1}^{N} |x_i|, \quad \|x\|_2 = \sqrt{\sum_{i=1}^{N} |x_i|^2}.$$

定理 2.3.1 (有限次元ベクトル空間のノルムの同値性) K^N の任意の二つのノルムは互いに同値である。すなわち $\|\cdot\|, \|\cdot\|'$ を K^N のノルムとすると、

$$\exists m > 0, \exists M > 0 \forall x \in K^N \quad m\|x\|' \leq \|x\| \leq M\|x\|'.$$

従って、どのノルムで考えても、収束や連続性の概念は一致する。

例 2.3.2 K^N において

$$\|x\|_\infty \leq \|x\|_2 \leq \sqrt{N}\|x\|_\infty, \quad \|x\|_\infty \leq \|x\|_1 \leq N\|x\|_\infty.$$

これを見ると N が大きいときは、同値と言っても、かなり値が違い得ることが見てとれる。反復法の停止則や、誤差の見積もりをする際には、「どれでも同じ」などと素朴に考えることは出来ない。

定義 2.3.2 (行列の作用素ノルム) K^N のノルム $\|\cdot\|$ を一つ定めたとき、N 次正方行列の空間 $M(K, N)$ に次のようにして定められるノルムを作用素ノルム、またはベクトルのノルムに付随する行列ノルムという。

$$\|A\| = \sup_{x \in K^N \setminus \{0\}} \frac{\|Ax\|}{\|x\|}.$$

命題 2.3.1 (i) 確かに作用素ノルムはノルムである。

(ii) $\|Ax\| \leq \|A\|\|x\|$ $(A \in M(K, N), x \in K^N)$.

(iii) $\|AB\| \leq \|A\|\|B\|$ $(A, B \in M(K, N))$.

以下では K^N にノルムを定めたとき、$M(K, N)$ には必ず作用素ノルムをいれて考えることと約束する。

練習問題 $M(K; N)$ のノルムを

$$\|A\|_1 = \sup \frac{\|Ax\|_1}{\|x\|_1}, \quad \|A\|_2 = \sup \frac{\|Ax\|_2}{\|x\|_2}, \quad \|A\|_\infty = \sup \frac{\|Ax\|_\infty}{\|x\|_\infty} \quad (A \in M(K; N))$$

とおくとき、各ノルムの同値性を示せ。

2.3.4 行列の条件数

定義 2.3.3 (行列の条件数) K^N にノルムが定められているとき、$A \in GL(K,N)$ に対して、条件数 (condition number) $\mathrm{cond}(A)$ を

$$\mathrm{cond}(A) = \|A\|\|A^{-1}\|$$

で定義する。

命題 2.3.2 (条件数の性質) (i) $\mathrm{cond}(A) \geq 1$.

(ii) $\forall \lambda \in K^*$ $\mathrm{cond}(\lambda A) = \mathrm{cond}(A)$. 特に $A = \lambda I$ のとき $\mathrm{cond}(A) = 1$.

(iii) $Ax = b$, $(A + \Delta A)(x + \Delta x) = b$ となっているならば

$$\frac{\|\Delta x\|}{\|x + \Delta x\|} \leq \mathrm{cond}(A) \frac{\|\Delta A\|}{\|A\|}.$$

$\|A^{-1}\Delta A\| < 1$ を仮定すると

$$\frac{\|\Delta x\|}{\|x\|} \leq \frac{\mathrm{cond}(A)}{1 - \|A^{-1}\Delta A\|} \frac{\|\Delta A\|}{\|A\|}.$$

(iv) $Ax = b$, $A(x + \Delta x) = (b + \Delta b)$ となっているならば

$$\frac{\|\Delta x\|}{\|x\|} \leq \mathrm{cond}(A) \frac{\|\Delta b\|}{\|b\|}.$$

(v) $Ax = b$, $(A + \Delta A)(x + \Delta x) = (b + \Delta b)$, $\|A^{-1}\| \leq \frac{1}{\|A^{-1}\|}$ となっているならば

$$\frac{\|\Delta x\|}{\|x\|} \leq \frac{\mathrm{cond}(A)}{1 - \mathrm{cond}(A)\frac{\|\Delta A\|}{\|A\|}} \left(\frac{\|\Delta A\|}{\|A\|} + \frac{\|\Delta b\|}{\|b\|} \right).$$

上の命題の「実際的な意味」を考えるために、次の注意をしておこう。ピボットの選択をした Gauss の消去法で、まともに計算が進んだ場合は、

$$\frac{\|\Delta A\|}{\|A\|}, \quad \frac{\|\Delta b\|}{\|b\|}$$

は、用いている浮動小数点数の体系の計算機イプシロンの（ある程度は見積もることの出来る、そう大きくはない）定数倍程度の量であると考えられる。

$\mathrm{cond}(A)$ は問題の解きにくさを計る目安となる。つまり $\mathrm{cond}(A)$ が大きいと解きにくい。特に

$\mathrm{cond}(A)$ が計算機イプシロンの逆数程度以上に大きい場合はまともな計算は望めない。

2.4 参考書

1. 戸川 隼人、BASIC による線形代数入門、共立出版やさしいところから、一歩一歩、読者を線形計算に導いてくれる本。

2. 名取 亮、線形計算、朝倉書店
   いわゆる教科書風の本で、この分野ではとても珍しい。プログラムは載っていないが、サービス精神に溢れていて、分かりやすい。

3. 、数値計算の常識、共立出版
   含蓄に富んだ話が多く、座右に置くことを勧めたい本であるが、今回は「逆行列よ、さようなら」を見てもらいたい。

第3章　連立1次方程式に対するCG法

(卒業研究で使うことがあるので「卒研テキスト」に移した。その他に「自分のための線形代数」に何か書くかもしれない。)

第4章　固有値問題

4.1　一般的な注意

- 直接法はない (固有方程式と同値で、それは解けない)
- 対称、非対称の差は大きい
- 中継地点 (三重対角行列、Hessenberg 形) を経由すべし

4.1.1　問題の定式化

いま A を N 次正方行列とする。この時

$$Ax = \lambda x, \quad x \neq 0$$

を満たす $\lambda \in \mathbf{C}$ を A の**固有値** (eigenvalue), $x \in \mathbf{C}^N \setminus \{0\}$ を A の固有値 λ に属する**固有ベクトル** (eigenvector) と呼ぶ。

固有値問題 — 固有値、固有ベクトルを求める問題 — は非線形問題であり、有限回の四則演算では解けない ($N \geq 5$ のときは巾根を求める操作を用いても解けない)。これを解くには、何らかの意味での反復法が必要である。

これに類似した問題に、**一般化固有値問題**と**特異値問題**がある。ここでは名前をあげておくだけにとどめるが、将来問題に出会った時に、固有値問題の親戚であると気がつけば良い。いずれの問題も、固有値問題のアルゴリズムを修正したもので解くことが出来る。

一般化固有値問題　行列 A, B が与えられた時、方程式

$$Ax = \lambda Bx, \quad x \neq 0$$

を満たすスカラー λ, ベクトル x を求める問題を一般化固有値問題という。これは 2 次形式の固有値問題などに関係して現れる。$B = I$ (単位行列) の場合は通常の固有値問題になる。応用上は B は多くの場合、正定値対称行列になる。B が正則である場合、上の方程式に B^{-1} をかけて $A' = B^{-1}A$ とおくと

$$A'x = \lambda x, \quad x \neq 0$$

となって、形式的には通常の固有値問題に帰着するが、このやり方は大抵の場合、得策ではない (A, B が対称であっても A' が対称であるとは限らなくなるなどの理由がある)。

特異値問題　一言でいうと正方行列でないような行列に対して固有値問題を一般化したものである。

4.1.2 線形代数の復習

線形代数学で学んだことをいくつか復習しておこう。

定理 4.1.1 固有値は、固有方程式と呼ばれる代数方程式

$$\det(\lambda I - A) = 0$$

の根である。従って N 次正方行列の固有値は重複度を込めて数えると N 個ある。

逆に任意の代数方程式に対して、それを固有方程式にもつ行列が存在するので、数学的には「固有値問題は代数方程式の問題と同値である」。

定理 4.1.2 (i) Hermite 行列の固有値は実数であり、ユニタリ行列で対角化できる。

(ii) 実対称行列の固有値は実数であり、直交行列で対角化できる。

4.1.3 どう立ち向かうべきか

- 固有値を求めるのに、固有方程式を解こうとするのは得策ではない[1]。固有値問題を解くには、以下に説明する固有値問題用の解法を採用すべきである。
- 実際的な観点からは、固有値問題の解法に使用される各種方法の原理を理解し、自分が解こうとしている問題にあった方法 or プログラムを選択できるようになることを目指すのが良い。
- 解こうとしている問題については、以下のことに留意して考えよう。

  (i) 問題は対称 (=行列が実対称行列または Hermite 行列) かどうか。

  (ii) すべての固有値が欲しいのかどうか。

  (iii) 固有ベクトルは必要かどうか。

  (ii), (iii) は要するに「ムダはやめよう」ということである。
- 対称な問題は非対称な問題と比べて解きやすい。

  「対称な問題を解くのはサイエンスであるが、非対称な問題を解くのはアートである。」

  という言葉があるくらいである。

[1] そもそも次数 n が 4 以下でないと根の公式はないし、代数方程式のための数値解法で解ける問題の範囲は、固有値問題のための数値解法で解ける範囲よりも狭い。

4.2 解法についての概観

歴史的なことを述べると、以前は対称な問題は Jacobi 法と呼ばれるアルゴリズムで解かれるのが普通であった。しかし Jacobi 法は N が小さな (せいぜい数十) である場合には実用的であるが、N の大きな問題を解くのに採用するのは得策ではない。今では行列の三重対角化や Hessenberg 形への変換を利用した解法が主流である（対称行列の場合は三重対角化、非対称行列の場合は Hessenberg 行列への変換を行なうことになる)。行列を三重対角化（resp. Hessenberg 化）するとは、相似変換を施して、行列を三重対角行列（resp. Hessenberg 行列）に変換することをいう。この変換の方法は色々あるが、いずれも $O(N^3)$ の演算量で済む。後で述べるように相似変換で固有値は不変なので、変換後の「簡単な」行列の固有値を求めれば、元の行列の固有値が求まったことになる。三重対角行列や Hessenberg 行列に対する固有値問題は、元の行列よりも簡単に解ける、というのが基本的なアイディアである。

色々な細かな技法（部品）を、利用者の要求に応じて組み合わせることで、望ましい解法が出来上がる。以下、それらの原理を説明する。三重対角化や Hessenberg 化のためのアルゴリズムの解説は次節にまわして、ここでは実際に固有値を求める諸方法を解説する。

4.2.1 相似変換

固有値問題でもっとも基本的な方法は**相似変換**である。これは行列 A を正則行列 P によって、$P^{-1}AP$ に変換することを意味する。

定理 4.2.1 相似変換により固有値は不変である。

に注意しよう。A が実対称な場合には P として直交行列が使われる。この場合 $P^{-1} = {}^tP$ であり、計算が簡単になることの他に、様々な利点がある。

直交行列として、Householder 行列を採用したものは、**鏡映変換**と呼ばれてよく使われる。Householder 行列については後述する。

4.2.2 Jacobi 法 — 温故知新

1960 年代まで主流であった **Jacobi 法**について簡単に説明しよう。これは実対称行列を 2 次元の回転変換により変形していって、対角成分以外の成分の絶対値を小さくしていくというものであり、行列のサイズ N が 10 程度の小さなものであれば現在でも実用的である。

なぜ「対角成分以外の成分の絶対値を小さくしていく」のか？これについて説明しよう。まず次の定理は簡単であるが重要である。

定理 4.2.2 対角行列の固有値は対角成分である。

一般の行列の場合も次の定理が成り立つ。おおざっぱにまとめると「行列の固有値は対角成分に近く、そのずれは非対角成分の大きさによる」。

定理 4.2.3 (Gerschgorin の円板定理) $A=(a_{ij})$ に対して

$$\Delta_i=\left\{z\in\mathbf{C};|z-a_{ii}|\le\sum_{1\le j\le N,j\ne i}|a_{ij}|\right\}\quad(i=1,2,\cdots,N)$$

とおくと、

$$\sigma(A)\equiv A\text{ の固有値全体}\subset\bigcup_{i=1}^{n}\Delta_i.$$

もし Δ_i のうちで k 個が連結成分をなせば、その中に k 個の固有値がある。

証明　$Ax=\lambda x$, $x=(x_1,\ldots,x_N)\ne 0$ とする。今 x の絶対値最大の成分を x_k とする:

$$\max_{i=1,2,\ldots,N}|x_i|=|x_k|.$$

方程式 $Ax=\lambda x$ の第 k 成分を書くと

$$\sum_{j=1}^{N}a_{kj}x_j=\lambda x_k.$$

これから

$$(\lambda-a_{kk})x_k=\sum_{1\le j\le N,j\ne k}a_{kj}x_j.$$

(以下 $1\le j\le N, j\ne k$ を単に $j\ne k$ と書くことにする。) よって

$$|\lambda-a_{kk}|\le\sum_{j\ne k}|a_{kj}|\left|\frac{x_j}{x_k}\right|\le\sum_{j\ne k}|a_{kj}|,$$

したがって $\lambda\in\Delta_k\subset\bigcup_{i=1}^{N}\Delta_i$.

つぎに

$$D=\mathrm{diag}(a_{11},\cdots,a_{NN}),\quad A_t=(1-t)D+tA\quad(t\in[0,1])$$

とおくと、A_t に対する円板は

$$\Delta_i(t)=\{z\in\mathbf{C};|z-a_{ii}|\le t\sum_{j\ne i}|a_{ij}|\}$$

である。A_i の固有値は t について連続的に変化し、$t=0$ のとき $\Delta_i(0)=\{a_{ii}\}$ にそれぞれ重複個数だけある。t を増加させれば、いくつかの円板が拡大して合流するが、k 個の $\Delta_i(t)$ が合流すれば、その中に A_t の k 個の固有値が存在する。ゆえに $t=1$ に達したとき、Δ_i の k 個が連結成分をなせば、その中に $A_1=A$ の k 個の固有値が存在する。 ■

4.2.3 巾乗法 (power method)

ここでも問題は対称であると仮定する。
与えられた行列の絶対値最大の固有値を求めるための、累乗法あるいは巾乗法[2]を説明する。
行列 A の固有値 $\{\lambda_i; i = 1, 2, \cdots, n\}$ は絶対値の順に番号づけられているとする:

$$|\lambda_1| \geq |\lambda_2| \geq \cdots |\lambda_n|.$$

また、$\{u_i; i = 1, 2, \cdots, n\}$ は $\{\lambda_i\}$ に対応する A の固有ベクトルからなる正規直交基底とする。
ここで話を簡単にするために次の仮定をおく。

$$\text{仮定:} \quad |\lambda_1| > |\lambda_2|.$$

この時、適当な x_0 ($x_0 \notin Span\{u_2, u_3, \cdots, u_n\}$) を選んで

$$\left\{ \begin{array}{lcl} y_{k+1} & := & Ax_k \\ x_{k+1} & := & y_{k+1}/\|y_{k+1}\| \end{array} \right.$$

によりベクトル列 $\{x_k; k = 0, 1, \cdots\}$ を定めると、

$$\lim_{k\to\infty} x_k = \pm u_1$$

となる。実際には十分大きな番号 k を取れば、$x_k = \pm u_1$ とみなしてよい。

大雑把な説明　まず適当な $\{c_i\}$ を選ぶことにより

$$x_0 = \sum_{i=1}^{n} c_i u_i$$

と展開されることに注意する。ここで $c_1 \neq 0$ である。

$$\begin{array}{rcl} A^k x_0 & = & \displaystyle\sum_{i=1}^{n} c_i A^k u_i = \sum_{i=1}^{n} c_i \lambda^k u_i \\ & = & \displaystyle c_1 \lambda_1^k \left\{ u_1 + \sum_{i=2}^{n} \left(\frac{\lambda_i}{\lambda_1}\right)^k c_i u_i \right\} \end{array}$$

ここで $x_k = A^k x_0/\|A^k x_0\|$ であることに注意すると、 $x_k \to \pm u_1$ $(k \to \infty)$ となることが分かる。

4.2.4 固有ベクトルから固有値を求める方法

誤差がなければ、$Ax = \lambda x$ の適当な ($x_i \neq 0$ となる i に対する) 成分に対応する方程式

$$\sum_{j=1}^{n} a_{ij} x_j = \lambda x_i$$

[2]冪乗法あるいは幂乗法が正しい？

の両辺を x_i で割れば λ が求まるが、x が近似的な固有ベクトルでしかない場合には、以下に解説する Rayleigh 商を用いる方法の方が良い。

定義 4.2.1 (Rayleigh 商) N 次正方行列 A と、$x \in \mathbf{C}^N$ に対して

$$\frac{(Ax, x)}{(x, x)}$$

を **Rayleigh 商**という。

x を A の近似固有ベクトルとする時、Rayleigh 商

$$\lambda' = \frac{(Ax, x)}{(x, x)}$$

は x に対応する固有値の良い近似になる。粗く言って

$$\text{固有値の誤差} = O(\text{固有ベクトルの誤差}^2).$$

命題 4.2.1 A が正規行列 $(AA^* = A^*A)$ で、その固有値を $\lambda_1, \cdots, \lambda_N$ とし、それに対応する固有ベクトルからなる正規直交基底を $x_1, x_2, \cdots, x_N$ とする。いま単位ベクトル v の Rayleigh 商 $\mu = v^*Av$ が

$$\|Av - \mu v\| = \varepsilon, \quad |\lambda_j - \mu| \geq \delta > \varepsilon \quad (j = 2, \ldots, N)$$

を満たせば、次の意味で μ は λ_1 にごく近い:

$$|\lambda_1 - \mu| \leq \frac{\varepsilon^2}{\delta(1 - \varepsilon^2/\delta^2)}.$$

さらに v は次の意味で x_1 に近い: $|c| = 1$ である定数 c に対して

$$\|cv - x_1\|^2 \leq (\varepsilon/\delta)^2 + (\varepsilon/\delta)^4.$$

証明　省略。一松を見よ。

4.2.5　逆反復法

絶対値が最小の固有値 (上の記号で λ_n) を求める方法である。A^{-1} の固有値は $\lambda_1^{-1}, \cdots, \lambda_n^{-1}$ であり、絶対値最大のものは λ_n^{-1} となるから、A^{-1} に巾乗法を適用することにより、λ_n^{-1} が求まり、その逆数を取れば λ_n が得られる。

計算上の注意:　反復の各段階で

$$y_{k+1} := A^{-1}x_k$$

という計算が必要になるが、これは

$$Ay_{k+1} = x_k$$

という y_{k+1} に関する連立一次方程式を解くことにより実行する (いつものことであるが、A^{-1} を計算するのは馬鹿馬鹿しい)。

4.2.6 シフト法

既に得られている近似固有値の精度を改良したい、あるいは、ある指定した値に最も近い固有値を求める方法である。

行列 A の固有値 λ_i に対する近似固有値 λ'_i が分かっているとしよう。この時、

$$A' = A - \lambda'_i I$$

の固有値は

$$\lambda_1 - \lambda'_i, \lambda_2 - \lambda'_i, \cdots, \lambda_n - \lambda'_i$$

となる。λ'_i は λ_i の近似値ということで、絶対値が最小なのは $\lambda_i - \lambda'_i$ であると期待できる。よって、A' に対して逆反復法を適用すれば、この値 (それを $\Delta\lambda$ とおこう) が高精度に計算できる。こうして

$$\lambda_i := \lambda'_i + \Delta\lambda$$

により λ_i が求まる。

4.3 三重対角化の手法

実対称行列に対する三重対角化の手法を学ぶ。

1. Givens 法
2. Householder 法
3. Lanczos 法

与えられた実対称行列 $A = (a_{ij})$ に対し、適当な正則行列 P を求めて

$$P^{-1}AP = \text{三重対角}$$

とする。特に P として直交行列を取る。

歴史的には、Jacobi 法の変形として Givens 法が最初に現われたが、その後現われた Householder 法は演算回数が約半分となるなど利点が多く普及している。

4.3.1 Householder 法

u を $\mathbf{R}^n$ の単位ベクトルとする。u に直交する超平面

$$W_u \equiv \{x \in \mathbf{R}^n; (x, u) = 0\}$$

に関する対称移動を表す変換を U とすると、

$$U = I - 2uu^T$$

である。($\overrightarrow{\mathrm{OP}} = x$ なる点 P から W_u に下ろした垂線の足を Q とすると、$\overrightarrow{\mathrm{QP}} = (x,u)u$. それゆえ、 $Ux = x - 2\vec{QP} = x - 2(x,u)u = (I - 2uu^T)x$)

定義 4.3.1 (鏡映変換、Householder 行列) $\mathbf{R}^n$ の単位ベクトル u に対し、行列 $U = I - 2uu^T$ で定まる線形変換をベクトル u に対する**鏡映変換**、U を **Householder 行列** (または基本直交行列) と呼ぶ。

補題 4.3.1 (鏡映変換の性質) U を Householder 変換とするとき、

(i) U は直交変換である。

(ii) U は対称変換である。

証明

(i) $UU^T = I$ を計算で確かめてもよいが、U が長さを変えないことから明らかである。

(ii) これは $U = I - 2uu^T$ であることから分かる。 ■

参考　N 次元の任意の直交変換は、N 個以下のベクトルに関する鏡映変換の積に書ける（Cartan)。

補題 4.3.2 $\|x\| = \|y\|$ なる $x, y \in \mathbf{R}^n$ に対して、

$$u = \frac{x-y}{\|x-y\|}, \quad U = I - 2uu^T$$

とおくと $Ux = y$.

Householder 行列による三重対角化

$$A_0 = A,$$

$$A_1 = U_0 A_0 U_0 = \begin{bmatrix} * & * & 0 & \cdots & 0 \\ * & * & * & \cdots & * \\ 0 & * & * & \cdots & * \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & * & * & \cdots & * \end{bmatrix}, \quad U_0 = I - 2u_0 u_0^T$$

$$A_2 = U_1 A_1 U_1 = \begin{bmatrix} * & * & 0 & 0 & \cdots & 0 \\ * & * & * & 0 & \cdots & 0 \\ 0 & * & * & * & \cdots & * \\ 0 & 0 & * & * & \cdots & * \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & * & * & \cdots & * \end{bmatrix}, \quad U_1 = I - 2u_1 u_1^T$$

$$\cdots\cdots\cdots$$

$$A_{N-2} \;=\; U_{N-3}A_{N-3}U_{N-3} = \begin{bmatrix} * & * & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ * & * & * & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & * & * & * & \ddots & 0 & 0 & 0 & 0 \\ 0 & 0 & * & * & \ddots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \ddots & * & * & 0 & 0 \\ 0 & 0 & 0 & 0 & \ddots & * & * & * & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & * & * & * \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & * & * \end{bmatrix}, \quad U_{N-3} = I - 2u_{N-3}u_{N-3}^T$$

のように $N-2$ 回の Householder 変換で三重対角化する。この 1 ステップ (A_{k-1} から A_k を作るところ) を発見的に説明しよう。面倒なので A_{k-1} を単に A と書き、

$$A = \left[\begin{array}{cc|c} & & 0 \\ & C & \\ & & * \quad b^T \\ \hline 0 & * & \\ & b & B \end{array}\right]$$

とブロック分けしておく。$u = u_k$ の最初の k 成分を 0 とおく、すなわち

$$u = u_k = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ * \\ \vdots \\ * \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \\ v \\ \\ \end{pmatrix}$$

とすると

$$U = I_N - 2uu^T = \begin{bmatrix} I_k & O \\ O & Q \end{bmatrix}, \quad Q = I_{N-k} - 2vv^T$$

となるので、

$$\text{新}\, A = UAU$$

とすると対応するブロックは

$$\begin{aligned} \text{新}\, C &= C, \\ \text{新}\, B &= QBQ, \\ \text{新}\, b &= Qb \end{aligned}$$

目標は

$$\text{新}\, b = Qb = \begin{pmatrix} s \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

の形にすることである。補題 4.3.2 を考えると $s = \pm\|b\|$ とすればよい。

$$s \text{ の符号} = -b \text{ の第 1 成分 } (-b_1) \text{ の符号}$$

となるように選ぶと桁落ちが起こりにくい。まとめると

$$(\heartsuit_k) \quad \begin{cases} s & = & -\text{sign}\,(b_1) \cdot \|b\|, \\ \text{新}\, b & = & \begin{pmatrix} s \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \\ v & = & \dfrac{b - \text{新}\, b}{\|b - \text{新}\, b\|}, \\ Q & = & I_{N-k} - 2vv^T \end{cases}$$

全体としては

for $k := 1$ to $N - 2$ do
begin
　　第 k 列の第 $k+1$ 成分以降を b とする。
　　$(\heartsuit_k)$ により s, 新 b, v, Q を作る。
　　新 $B = QBQ$ を計算する。
end

色々な工夫が可能である。まとめると

$$\begin{cases} s & = & -\text{sign}\,(b_1) \times \|b\| \\ w & = & b - \text{新}\, b = \text{第 1 成分は旧} b \text{ の第 1 成分} - s, \text{ その他の成分はそのまま} \\ \|w\|^2 & = & 2\{s^2 - (b \text{ の第一成分}) \times s\} \\ p & = & \dfrac{Bw}{(\|w\|^2/2)} \\ \alpha & = & w^T p / \|w\|^2 \\ q & = & p - \alpha w \\ \text{新}\, B & = & B - wq^T - qw^T \quad (\text{対称なので半分の計算で OK}) \end{cases}$$

注: この文書の過去の版で、$\|w\|^2$ の計算式の符号を間違えて

$$\|w\|^2 = 2\{s^2 + (b \text{ の第一成分}) \times s\}$$

のようにしていました。ごめんなさい。 ■

4.3.2 Lanczos(ランチョス) 法

実対称行列 A が直交行列 P で三重対角化されたとする:

$$B = P^{-1}AP = \begin{pmatrix} \alpha_1 & \beta_1 & 0 & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{N-2} & \alpha_{N-1} & \beta_{N-1} \\ 0 & & 0 & \beta_{N-1} & \alpha_N \end{pmatrix}.$$

つまり $AP = PB$ であるが、$P = (u_1 u_2 \cdots u_N)$ とおくと、

$$A(u_1 u_2 \cdots u_N) = (u_1 u_2 \cdots u_N) \begin{pmatrix} \alpha_1 & \beta_1 & 0 & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{N-2} & \alpha_{N-1} & \beta_{N-1} \\ 0 & & 0 & \beta_{N-1} & \alpha_N \end{pmatrix}$$

となるから

$$\begin{aligned} Au_1 &= \alpha_1 u_1 + \beta_1 u_2 \\ Au_2 &= \beta_1 u_1 + \alpha_2 u_2 + \beta_2 u_3 \\ &\cdots \\ Au_i &= \beta_{i-1} u_{i-1} + \alpha_i u_i + \beta_i u_i \\ &\cdots \\ Au_N &= \beta_{N-1} u_{N-1} + \alpha_N u_N \end{aligned}$$

第 k 行と u_k の内積を作ると

$$\alpha_k = {u_k}^T A u_k \tag{4.1}$$

また

$$v_{k+1} \stackrel{\text{def.}}{\equiv} Au_k - (\beta_k u_{k-1} + \alpha_k u_k) \tag{4.2}$$

とおくと、$v_{k+1} = \beta_k u_{k+1}$, $\|u_{k+1}\|$ であるから、

$$\beta_k = \|v_{k+1}\| \tag{4.3}$$

$$u_{k+1} = \frac{v_{k+1}}{\beta_k} \tag{4.4}$$

4.4 二分法

二分法 (**bisection method**, **Sturum method**) を解説する。

- これは実対称行列専用の方法である。
- 固有値は固有多項式の根であるが、Strum 列の理論によって、ある区間内の固有値の個数を計算することが出来る。このことと、いわゆる二分探索 (binary search) の方法を組み合わせて得られるのが、固有値計算手法としての二分法である。

4.4.1 伝統的な説明

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

を実対称三重対角行列とする。

$b_k \neq 0$ $(k = 1, 2, \cdots, N-1)$ 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

$$p_k(\lambda) \overset{\text{def.}}{=} \begin{cases} \det(\lambda I_k - T_k) & (k = 1, 2, \cdots, N) \\ 1 & (k = 0) \end{cases}.$$

ただし、

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

すぐ分かることは、

補題 4.4.1

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

補題 4.4.2 $\{p_k(\lambda)\}_{k=0,1,\cdots,N}$ は Strum 列である。すなわち

(i) $p_0(\lambda)$ は定符合。

(ii) $p_k(\lambda)$ の根は有限個 $(k = 1, 2, \ldots, N)$.

(iii) $p_k(\lambda)$, $p_{k+1}(\lambda)$ は共通根を持たない。

(iv) $p_k(\lambda_0) = 0 \Longrightarrow p_{k-1}(\lambda_0)p_{k+1}(\lambda_0) < 0$ $(k = 1, 2, \cdots, N-1)$.

証明　Strum 列については別に一つの章をもうけて説明するので、そこで述べることにする。

符号の変化数　$\{p_i(\lambda)\}_{i=0}^N$ を Strum 列とするとき、$p_N(a) \neq 0$ なる a に対して

$$N(a) \stackrel{\text{def.}}{\equiv} \text{“}\{p_0(a), p_1(a), \cdots, p_N(a)\}\text{”} \text{ の}\textbf{符号の変化数}$$

とおく。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。

Strum の定理によって、$p_N(a)p_N(b) \neq 0$ なる $[a, b]$ において、$[a, b]$ 内の零点の個数は $N(a) - N(b)$ であることが分かる。

4.4.2　Sylvester の慣性律による説明

線形代数でおなじみの **Sylvester の慣性律**「行列の符号数は座標変換で変化しない」を復習しよう。正則行列 M に対して

$$\begin{aligned} \pi(M) &= M \text{ の固有値のうち正のものの個数} \\ \zeta(M) &= M \text{ の固有値のうち } 0 \text{ のものの個数} \\ \nu(M) &= M \text{ の固有値のうち負のものの個数} \end{aligned}$$

とおく。

定理 4.4.1 *(Sylvester の慣性律による説明)* A を 実対称行列、S を正則行列で $B = S^T AS$ とするとき

$$\pi(A) = \pi(B), \quad \zeta(A) = \zeta(B), \quad \nu(A) = \nu(B).$$

与えられた $\sigma \in \mathbf{R}$ に対して、$A - \sigma I$ を LDL 分解する、すなわち

$$A - \sigma I = LDL^T$$

ただし L は下三角行列、D は対角行列とする:

$$L = \begin{pmatrix} * & 0 & 0 & & & 0 & & \\ * & * & 0 & & & & & \\ * & * & * & & & & & \\ & & & & & & & \\ \vdots & \vdots & \vdots & & \ddots & \ddots & \ddots & \\ * & * & * & \cdots & & * & 0 & 0 \\ * & * & * & \cdots & & * & * & 0 \\ * & * & * & \cdots & & * & * & * \end{pmatrix}, \quad D = \begin{pmatrix} d_1 & & & 0 \\ & d_2 & & \\ & & \ddots & \\ 0 & & & d_N \end{pmatrix}$$

このとき定理から

$$\pi(A - \sigma I) = \pi(D), \quad \zeta(A - \sigma I) = \zeta(D), \quad \nu(A - \sigma I) = \nu(D)$$

であるが、$\pi(D), \zeta(D), \nu(D)$ はすぐに分かる。従って、行列 A の固有値について、任意に与えられた実数 σ よりも大きいもの、小さいものの個数が勘定できることになる。

4.5 QR 法

この文書は主に一松 [?] による。

4.5.1 正則行列の QR 分解

$A \in GL(n; \mathbf{C})$ の列ベクトルを $\boldsymbol{a}_1, \cdots, \boldsymbol{a}_n$ とする:

$$A = (\boldsymbol{a}_1 \ \boldsymbol{a}_1 \ \cdots \boldsymbol{a}_n). \tag{4.5}$$

この $\boldsymbol{a}_1, \cdots, \boldsymbol{a}_n$ を Gram-Schmidt の直交化法して $\boldsymbol{q}_1, \cdots, \boldsymbol{q}_n$ を作る。つまり

$$\boldsymbol{v}_k \overset{\text{def.}}{=} \boldsymbol{a}_k - \sum_{i=1}^{k-1} \langle \boldsymbol{a}_k, \boldsymbol{q}_i \rangle \boldsymbol{q}_i, \quad \boldsymbol{q}_k \overset{\text{def.}}{=} \frac{\boldsymbol{v}_k}{\|\boldsymbol{v}_k\|} \quad (k = 1, 2, \cdots, n)$$

として計算するわけだが、中間変数 r_{ik} $(1 \leq i \leq k \leq n)$ を導入して、次のように書き換えておこう。

$$\boldsymbol{v}_k \overset{\text{def.}}{=} \boldsymbol{a}_k - \sum_{i=1}^{k-1} r_{ik} \boldsymbol{q}_i, \quad r_{ik} \overset{\text{def.}}{=} \langle \boldsymbol{a}_k, \boldsymbol{q}_i \rangle, \tag{4.6}$$

$$\boldsymbol{q}_k \overset{\text{def.}}{=} \frac{\boldsymbol{v}_k}{r_{kk}}, \quad r_{kk} \overset{\text{def.}}{=} \|\boldsymbol{v}_k\| \quad (k = 1, 2, \cdots, n) \tag{4.7}$$

さて

$$Q \overset{\text{def.}}{=} (\boldsymbol{q}_1\ \boldsymbol{q}_2\ \cdots\ \boldsymbol{q}_n), \quad R \overset{\text{def.}}{=} \begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ & r_{22} & \cdots & r_{2n} \\ & & \ddots & \vdots \\ \mathbf{0} & & & r_{nn} \end{pmatrix}$$

とおくと、Q は unitary 行列で、

$$A = QR.$$

ここまで A は複素行列として計算してきたが、実行列である場合は、Q, R も実行列 (したがって Q は実直交行列) である。

定義 4.5.1 (QR 分解) 正則行列 $A \in GL(n;\mathbf{C})$ に対して、

$$A = QR \quad (Q \text{ は unitary 行列, } R \text{ は対角線分がすべて正の上三角行列})$$

となる Q, R を見い出すことを A を **QR 分解**すると言う。

上の議論から任意の正則行列は QR 分解可能であることが分かったが、実はこれは一意的である。

命題 4.5.1 (QR 分解の一意性) 任意の $A \in GL(n;\mathbf{C})$ に対して、QR 分解はただ一つしかない。

証明 $A = QR$ は

$$\boldsymbol{a}_k = \sum_{i=1}^{k} r_{ik}\boldsymbol{q}_i \quad (k = 1, 2, \cdots, n)$$

と書ける。$k = 1$ についての条件

$$\boldsymbol{a}_1 = r_{11}\boldsymbol{q}_1$$

から

$$r_{11} = |r_{11}| = \frac{\|\boldsymbol{a}_1\|}{\|\boldsymbol{q}_1\|} = \|\boldsymbol{a}_1\|,$$

$$\boldsymbol{q}_1 = \frac{\boldsymbol{a}_1}{r_{11}}.$$

次に $k = 2$ についての条件

$$\boldsymbol{a}_2 = r_{12}\boldsymbol{q}_1 + r_{22}\boldsymbol{q}_2$$

から、まず $\boldsymbol{q}_1$ との内積を取って、

$$r_{12} = \langle \boldsymbol{a}_1, \boldsymbol{q}_1, . \rangle$$

これから

$$\boldsymbol{v}_2 \overset{\text{def.}}{=} \boldsymbol{a}_2 - r_{12}\boldsymbol{q}_1$$

は計算できて、

$$\boldsymbol{v}_2 = r_{22}\boldsymbol{q}_2$$

であるから、

$$r_{22} = |r_{22}| = \frac{\|\boldsymbol{v}_2\|}{\|\boldsymbol{q}_2\|} = \|\boldsymbol{v}_2\|, \quad \boldsymbol{q}_2 = \frac{\boldsymbol{v}_2}{r_{22}}.$$

以下 $k = 3, \cdots, n$ と順に同様に計算できる。■

別証明　A の二つの QR 分解

$$A = Q_1 R_1, \quad A = Q_2 R_2$$

があれば、$Q_1 R_1 = Q_2 R_2$ から

$$Q_2^* Q_1 = R_2 R_1^{-1}. \tag{4.8}$$

この等式の左辺は unitary 行列である。実際

$$(Q_2^* Q_1)(Q_2^* Q_1)^* = Q_2^* Q_1 Q_1^* Q_2^{**} = Q_2^* Q_2 = I.$$

また (4.8) の右辺は上三角行列で、対角成分はすべて正である。

unitary かつ上三角かつ対角成分がすべて正という行列は単位行列に限られることは容易に証明できる。ゆえに

$$Q_2^* Q_1 = R_2 R_1^{-1} = I.$$

これから

$$Q_1 = Q_2, \quad R_1 = R_2.$$■

注意 4.5.1 (QR 分解を利用した連立 1 次方程式の解法) $A = QR$ という QR 分解があるとき、

$$Ax = b$$

は

$$QRx = b$$

であるから

$$x = R^{-1}(Q^* b)$$

上三角行列の逆行列をかける計算は、乗算 $n^2/2$ 回程度の計算量で計算できるから、x は乗算 $3n^2/2$ 回程度の計算量で計算できることが分かる。QR 分解は $O(n^3)$ 程度の計算量で求まることは分かるから、Gauss の消去法に基づく LU 分解とオーダーだけは匹敵する計算法である。(決して有利な方法ではないが。) ■

4.5.2　QR 変換

$$A = QR$$

という QR 分解が得られたとき、順番を変えて掛け算した行列 A_1 を作る。

$$A_1 = RQ$$

これは $R = Q^*A$ より

$$A_1 = Q^*AQ$$

となるので、実は A を Q で相似変換したものである。これを **QR 変換**と言う。

行列 A が与えられたとき、

$$\begin{aligned} A_0 &= A = Q_0R_0, \quad A_1 = R_0Q_0, \\ A_1 &= Q_1R_1, \quad A_2 = R_1Q_1, \\ A_2 &= Q_2R_2, \quad A_3 = R_2Q_2, \\ &\cdots \\ A_k &= Q_kR_k, \quad A_{k+1} = R_kQ_k, \\ &\cdots \end{aligned} \tag{4.9}$$

と QR 変換を繰り返すと、A_m の対角線の下側の成分はすべて 0 に収束する。

注意 4.5.2 (たまにある誤解を正す) このことを「A_k は上三角行列に収束する」と言うことがあるが、対角線の上にある成分が特定の値に収束するわけではない。言い換えると、特定の上三角行列 U があって、

$$\lim_{k\to\infty} A_k = U$$

となるわけではない。 ■

一般性の追求はほどほどにして、次の定理を証明しよう。

定理 4.5.1 (QR 法の原理) $A \in GL(n;\mathbf{C})$ で、その固有値 λ_j $(j = 1, 2, \cdots, n)$ はすべて相異なり、

$$|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$$

と仮定する。このとき $k \to \infty$ とすると、A_k の対角線より下のすべての成分は 0 に収束し、A_k の対角成分は $\lambda_1, \cdots, \lambda_n$ に収束する。

証明

ステップ 1 (A^{k+1} の QR 分解) まず

主張 1

$P_k \overset{\text{def.}}{=} Q_0Q_1\cdots Q_{k-1}Q_k$ とおくと

$$A_k = P_{k-1}^*AP_{k-1}. \tag{4.10}$$

実際、

$$\begin{aligned} A_1 &= R_0Q_0 = (Q_0^*A)Q_0 = Q_0^*AQ_0, \\ A_2 &= R_1Q_1 = (Q_1^*A_1)Q_1 = Q_1^*(Q_0^*AQ_0)Q_1 = Q_1^*Q_0^*AQ_0Q_1, \\ A_2 &= R_2Q_2 = (Q_2^*A_2)Q_2 = Q_2^*(Q_1^*Q_0^*AQ_0Q_1)Q_2 = Q_2^*Q_1^*Q_0^*AQ_0Q_1Q_2, \end{aligned}$$

$$
\begin{aligned}
&\cdots \\
A_k &= R_kQ_k = (Q_k^*A_{k-1})Q_k = Q_k^*(Q_{k-1}^*\cdots Q_1^*Q_0^*AQ_0Q_1\cdots Q_{k-1})Q_k, \\
&\cdots
\end{aligned}
$$

となる (帰納法で証明すべきかもしれないが)。

主張 2

$U_k \stackrel{\text{def.}}{=} R_kR_{k-1}\cdots R_1R_0$ とおくと $A^{k+1} = P_kU_k$ であり、これは A^{k+1} の QR 分解である。

まず

$$
\begin{aligned}
P_kU_k &= (Q_0Q_1\cdots Q_{k-1}Q_k)(R_kR_{k-1}\cdots R_1R_0) = (Q_0Q_1\cdots Q_{k-1})(Q_kR_k)(R_{k-1}\cdots R_1R_0) \\
&= P_{k-1}A_kU_{k-1}
\end{aligned}
$$

で、主張 1 より $P_{k-1}A_k = AP_{k-1}$ だから、

$$P_kU_k = AP_{k-1}U_{k-1}.$$

この式を再帰的に用いて、

$$P_kU_k = AP_{k-1}U_{k-1} = A(AP_{k-1}U_{k-2})\cdots = A^kP_0U_0 = A^kQ_0R_0 = A^kA = A^{k+1}.$$

一方、P_k は unitary 行列の積であるから unitary 行列、U_k は対角線分が正である上三角行列の積であるから対角成分が正である上三角行列である。ゆえに $A^{k+1} = P_kU_k$ は A^{k+1} の QR 分解である。

ステップ 2 (A_k の挙動) A は固有値がすべて相異なるから対角化が可能である。すなわち $\exists M \in GL(n;\mathbf{C})$ s.t.

$$M^{-1}AM = \Lambda, \quad \Lambda = \mathrm{diag}(\lambda_1,\cdots,\lambda_n).$$

M を $M = QR$ と QR 分解、M^{-1} を $M^{-1} = LU$ (ただし L の対角成分はすべて 1) と LU 分解して、

$$A^{k+1} = M\Lambda^{k+1}M^{-1} = (QR)\Lambda^{k+1}(LU) = QR(\Lambda^{k+1}L\Lambda^{-(k+1)})\Lambda^{k+1}U. \tag{4.11}$$

ここで $\Lambda^{k+1}L\Lambda^{-(k+1)}$ は下三角行列で、対角成分はすべて 1 に等しく、$i > j$ のとき (i,j) 成分は

$$\ell_{ij}\left(\frac{\lambda_i}{\lambda_j}\right)^{k+1}$$

に等しい (ただし ℓ_{ij} は L の (i,j) 成分)。そこで

$$\Lambda^{k+1}L\Lambda^{-(k+1)} = I + E_k$$

とおくと、$\lim_{k\to\infty} E_k = O$. 上式を (4.11) 代入して、

$$
\begin{aligned}
A^{k+1} &= QR(\Lambda^{k+1}L\Lambda^{-(k+1)})\Lambda^{l+1}U \\
&= QR(I+E_k)(R^{-1}R)(\Lambda^{l+1}U) = QR(I+E_k)R^{-1}(R\Lambda^{l+1}U) \\
&= Q(I+RE_kR^{-1})(R\Lambda^{l+1}U)
\end{aligned} \tag{4.12}
$$

この $I + RE_kR^{-1}$ を QR 分解する:

$$I + RE_kR^{-1} = S_kT_k. \tag{4.13}$$

$I + RE_kR^{-1} \to I$ $(k \to \infty)$ で、QR 分解の連続性は明らかだから、

$$\lim_{k\to\infty} S_k = I, \quad \lim_{k\to\infty} T_k = I.$$

(4.13) を (4.12) に代入して、

$$A^{k+1} = Q(S_kT_k)(R\Lambda^{k+1}U) = (QS_k)(T_kR\Lambda^{k+1}U). \tag{4.14}$$

この右辺は unitary 行列と、上三角行列の積の形になっている。

ステップ 3　(QR 分解との関係) $U = (u_{ij})$ として、

$$D_1 = \mathrm{diag}\left(\frac{\overline{\lambda_1}}{|\lambda_1|}, \frac{\overline{\lambda_2}}{|\lambda_2|}, \cdots, \frac{\overline{\lambda_n}}{|\lambda_n|}\right), \quad D_2 = \mathrm{diag}\left(\frac{\overline{u_{11}}}{|u_{11}|}, \frac{\overline{u_{22}}}{|u_{22}|}, \cdots, \frac{\overline{u_{nn}}}{|u_{nn}|}\right)$$

とおく。これを使って (4.14) を

$$A^{k+1} = (QA_kD_2^*D_1^{k*})[(D_1^kD_2)(T_kR)(D_1^kD_2)^*(D_1\Lambda)^k(D_2U)]$$

のように書き換えると、これは A^{k+1} の QR 分解になる。すでに $A^{k+1} = P_kU_k$ という QR 分解が得られていたので、QR 分解の一意性から、

$$P_k = QS_kD_2^*D_1^{k*}. \tag{4.15}$$

ステップ 4　(4.10) と (4.15) から

$$A_{k+1} = P_k^*AP_k = D_1^kD_2S_k^*(Q^*AQ)S_kD_2^*D_1^{k*}.$$

$$A = M\Lambda M^{-1} = (QR)\Lambda(R^{-1}Q^*)$$

より

$$Q^*AQ = R\Lambda R^{-1}.$$

この行列は (右辺に注目することで) 上三角行列で、かつ対角線上に $\lambda_1, \cdots, \lambda_n$ がこの順にならんでいる。$k \to \infty$ のとき、$S_k \to I$ だから

$$A_{k+1} \to D_1^k(D_2R\Lambda R^{-1}D_2^*)D_1^{*k}$$

この右辺は上三角行列で、対角成分は $\lambda_1, \lambda_2, \cdots, \lambda_n$ である。■

この定理から $k \to \infty$ とするときの収束の速さは

$$\left(\frac{\lambda_i}{\lambda_j}\right)^k$$

という因子に関わることが分かる。

4.5.3 QR 法の歴史

1961 年に Francis が複素行列の固有値の計算法として導入したもの。

一松 信、数値解析、朝倉書店 (1983)

4.6 特異値

いつの日か。やはり

一松 信、数値解析、朝倉書店 (1983)

4.7 固有値計算のためのパッケージ

4.8 参考書

[1] 名取亮、線形計算、朝倉書店

[2] 戸川隼人、マトリクスの数値計算、オーム社

[3] F. シャトラン、行列の固有値問題、シュプリンガー・フェアラーク東京

4.9 収束の速さ

$\{a_n\}$

$$\lim_{n\to\infty} a_n = a_\infty$$

$$\varepsilon_n \stackrel{\text{def.}}{\equiv} \|a_n - a_\infty\|$$

定義 4.9.1 *(線形収束)*

$$\exists L \in (0,1)\ s.t.\ \varepsilon_{n+1} =\leq L\varepsilon_n \quad (n \in \mathbf{N})$$

定義 4.9.2 *(p 次の収束)* $p \geq 2$

$$\exists L > 0\ s.t.\ \varepsilon_{n+1} =\leq L\varepsilon_n^p \quad (n \in \mathbf{N})$$

第5章　常微分方程式の初期値問題

5.1　序

微分方程式の解は関数であり、これは関数空間の要素としてとらえるのが (数学では) 普通である。(大抵の場合、関数空間は無限次元空間で、問題を複雑にしている。)

微分方程式は解析的[1]に解けないことが多い。たとえ解けても便利でないことがある[2]。

近似解法 — 解の有限的な近似表現を求める。

- 有限次元の関数空間の要素で近似する、が基本。
- 特に連続変数を離散変数に置き換えて近似する**離散変数法**が有力。

残念ながら

常微分方程式の初期値問題に限っても万能の方法はない。

プロでない平均的ユーザーとしては、実際的にはとりあえず Runge-Kutta 法を使い、不満があれば他の方法を考える、くらいで良いだろう。

この講義では、基礎概念について簡単に紹介した後で、

1. 刻み幅の自動調節 (adaptive stepsize control)
2. 硬い方程式 (stiff problem)

などの進んだ話題についてコメントする。詳しいことを知りたい場合は、まず (後述する) 三井斌友「数値解析入門」などを見るとよい。

5.2　初期値問題の設定

この稿では 1 階正規形の常微分方程式の初期値問題を扱う。すなわち

$$\frac{dx}{dt} = f(t,x) \quad (t \in I) \tag{5.1}$$

$$x(t_0) = x_0 \tag{5.2}$$

を満たす $x = x(t)$ を求めることを考える。ここで I は $t_0 \in \mathbf{R}$ を含む $\mathbf{R}$ の区間で、

$$\begin{cases} f : \mathbf{R}^{n+1} \supset \Omega \to \mathbf{R}^n \quad \text{連続}, \\ (t_0, x_0) \in \Omega \end{cases}$$

は与えられているとする。

[1]「解析的に解く」とは、不定積分を取る、四則演算を施す、逆関数を取る、初等関数に代入するなどで求めたり — いわゆる**求積法**で解いたり —、解を級数で表現したりすることを指す。

[2]例えば無限級数で解を表したとして、その値を計算するのは大変である。

注意　1 階正規形の方程式は十分一般的である。高階の方程式も 1 階にできるから。

5.3　常微分方程式の初期値問題の復習

5.3.1　数学理論

- 局所解の存在を保証するには f の連続性を仮定するだけで十分。
- 一意性は f の連続性だけでは不十分。

例 5.3.1 (無限個の解の分岐) 次の常微分方程式の初期値問題には無限個の解が存在する。

$$\begin{cases} x' = x^{1/2} & (t > 0) \\ x(0) = 0. \end{cases}$$

実際 $\forall t_0 \geq 0$ に対して

$$x(t) \overset{\text{def.}}{=} \begin{cases} 0 & (t \leq t_0) \\ \dfrac{(t-t_0)^2}{4} & (t > t_0) \end{cases}$$

は解である。

- f が次に示す “変数 x に関する**局所 Lipschitz 条件**” を満たせば一意性が成り立つ。$\forall (t,x) \in \Omega$, $\exists V : (t,x)$ の近傍, $\exists L_V > 0$ s.t.

$$\|f(t,x_1) - f(t,x_2)\| \leq L_V \|x_1 - x_2\| \quad ((t,x_1),\,(t,x_2) \in V).$$

f がこの条件を満たすための分かりやすい十分条件として、f が C^1-級であることがあげられる。すなわち

$$f \in C^1(\Omega) \Longrightarrow f : \text{局所 Lipschitz.}$$

- 大域的な存在について。$(t,x(t)) \to \partial\Omega$ または $\|x(t)\| \to \infty$ となるまで左右に延長できる (**延長不能解の存在定理**)。

  このうち $(t,x(t))$ が f の定義域 Ω の境界に近付くという条件は分かりやすいが、$\|x(t)\| \to \infty$ の方は見慣れない人もいるかも知れない。

例 5.3.2 (爆発解)

$$\begin{cases} x' &= x^2 \\ x(0) &= 1 \end{cases}$$

の解は

$$x(t) = \frac{1}{1-t} \quad (t \in (-\infty, 1))$$

であり、

$$\lim_{t \uparrow 1} x(t) = \infty.$$

5.3.2 数値解法

$I=[a,b]$ とする。$N\in\mathbf{N}$ に対し、I を N 個の小区間に分ける:

$$a=t_0<t_1<t_2<\cdots<t_N=b.$$

このとき、各 t_j における x の値 $x(t_j)$ の近似値 x_j を求めることを考える方法を**離散変数法** (**discrete variable method**) と呼ぶ。

区間の分割の仕方としては、例えば

$$h=\frac{b-a}{N},\quad t_j=a+jh\quad (j=0,1,\cdots,N).$$

のように等分割することが多い。

前進 Euler 法 微分係数 $x'(t)$ を前進差分商

$$\frac{x(t+h)-x(t)}{h}$$

で近似して作った漸化式

$$x_{j+1}=x_j+hf(t_j,x_j)\quad (j=0,1,2,\cdots)$$

で $\{x_j\}_{j=0}^N$ を計算する。

後退 Euler 法 前進差分商のかわりに後退差分商

$$\frac{x(t+h)-x(t)}{h}$$

で近似して得られる漸化式

$$x_{j+1}=x_j+hf(t_{j+1},x_{j+1})\quad (j=0,1,2,\cdots)$$

で $\{x_j\}_{j=0}^N$ を計算する。x_{j+1} を求めるために、方程式を解く必要がある (こういう方法を一般に**陰解法**と呼ぶ)。

Runge-Kutta 法

$$\begin{cases} k_1 &= hf(t_j,x_j)\\ k_2 &= hf(t_j+h/2,x_j+k_1/2)\\ k_3 &= hf(t_j+h/2,x_j+k_2/2)\\ k_4 &= hf(t_j+h,x_j+k_3)\end{cases}$$

$$x_{j+1}=x_j+\frac{1}{6}\left(k_1+2k_2+2k_3+k_4\right)$$

で $\{x_j\}_{j=0}^N$ を計算する方法を (古典的、あるいは 4 次の) **Runge-Kutta 法**と呼ぶ。

5.4 基本的な用語

なるべく一般的な解法で説明しよう。

段数, 多段法　k **段法**とは x_{j+k} を定めるために

$$(5.3)\quad \begin{aligned} x_{j+k} &= a_0 x_j + a_1 x_{j+1} + \cdots + a_{k-1} x_{j+k-1} + h\Phi(t_j, t_{j+1}, \cdots, t_{j+k}, x_j, x_{j+1}, \cdots, x_{j+k}) \\ &\equiv L(t_j, t_{j+1}, \cdots, t_{j+k}, x_j, x_{j+1}, \cdots, x_{j+k}, h) \end{aligned}$$

のように x_j, x_{j+1}, $\cdots$, x_{j+k} を含んだ方程式を用いる方法のことである。このような方程式 (あるいは方法) のことを**スキーム**(**scheme**) と呼ぶ。

ここで a_0, $\cdots$, a_{k-1} は $\displaystyle\sum_{i=0}^{k-1} a_i = 1$ を満たす定数。 Φ は f によって定まる、微分・積分などの無限小演算を含まない写像で、 $f \equiv 0$ ならば $\Phi \equiv 0$ となるものである。この $(k, \{a_i\}_{i=0}^{k-1}, \Phi)$ が方法を特徴づける。

問　Euler 法、Runge-Kutta 法がこの形になっていることを確かめよ。

- 上の整数 k をスキームの**段数** (**step number**) と呼ぶ。
  (Euler 法、Runge-Kutta 法では $k = 1$ である。)
- $k \geq 2$ なるスキームを**多段法** (**multistep method**) と呼ぶ。
- Φ が x_{j+k} によらないように表される時、**陽解法**(**explicit method**) であるとよび、そうでない場合を**陰解法** (**implicit method**) と呼ぶ。陰解法では x_{j+k} を求めるために、(一般には非線型の) 方程式を解かねばならないので、ほとんどの場合に反復法が必要になり、面倒であるが、次数が高くて安定性のよい方法が作れる。

局所離散化誤差、公式の次数　解法 (5.3) の、t における**局所離散化誤差**(**local truncation error**) を

$$\tau(t,h) \overset{\text{def.}}{=} \frac{1}{h}\left[x(t+kh) - L(t, t+h, \cdots, t+kh, x(t), x(t+h), \cdots, x(t+kh))\right]$$

で定義する。これを用いて公式の**次数**(**order**, **位数**とも呼ぶ) を次のように定義する。

公式の次数が少なくとも m $\overset{\text{def.}}{\Leftrightarrow}$ C^m-級の一意解を持つ任意の初期値問題に適用した場合
$$\tau(h) \overset{\text{def.}}{=} \max_{t\in[a,b-kh]} |\tau(t,h)| = O(h^m) \quad (h \to 0)$$

この $\tau(h)$ を**大域的離散化誤差**(**global discretization error**, **global truncation error**) と呼ぶ。

注意 5.4.1 f があまり滑らかでないときなど、解がなめらかでない場合、次数 m の公式を用いても $\tau(h) = O(h^m)$ は期待できない。

要するに　あらく言って m 次の公式とは、Taylor 展開して考えたとき、m 次の項まで一致するものであって、次のような性質を持つ:

(i) (どういうわけか運良く) 第 k 段まで正しく計算出来たとすると

$$x(t_{j+1}) - x_{j+1} = O(h^{m+1}) \quad (h \to 0).$$

(ii) 実際は誤差が累積するので

$$x(t_N) - x_N = O(h^m) \quad (h \to 0).$$

この左辺を**全離散化誤差**(**total discretization error**) と呼ぶ。

収束のための条件　以下の三条件が成り立つ時、収束する。

(i) 公式が少なくとも 1 次以上 (**適合条件**(**consistency**) を満たす)。

(ii) Φ が $x_j, x_{j+1}, \cdots, x_{j+k}$ について Lipschitz 条件を満たす。

(iii) 初期値 $x_1, x_2, \cdots, x_{k-1}$ が適切に用意された。

5.5　典型的なスキーム

5.5.1　Runge-Kutta 法とその一族

前節に解説したタイプの公式のうち、$k = 1$ の場合、すなわち 1 段法を **Runge-Kutta 型公式**という。いくつかの (t, x) について、右辺の $f(t, x)$ を計算し、それらの重みつき平均によって、適当な次数の (=その次数までの真の解の Taylor 展開と一致するような) 公式を作っている。具体的には Runge-Kutta 型公式の一般形は

$$\begin{cases} x_{j+1} &= x_j + h\displaystyle\sum_{i=1}^{s} \mu_i k_i \\ k_i &= f\left(t_i + \alpha_i h, x_j + h\displaystyle\sum_{\ell=1}^{s} \beta_{i\ell} k_\ell\right) \quad (i = 1, \cdots, s) \end{cases}$$

の形に書くことが出来る。ここで s を**段数**(**number of stages**) と呼ぶ。段数とは、要するに 1 ステップ先に進めるために必要な f の計算回数である[3]。

段数 s と係数 $\{\alpha_i; 1 \le i \le s\}$, $\{\beta_{ij}; 1 \le i, j \le s\}$, $\{\mu_i; 1 \le i \le s\}$ を選ぶとスキームが定まることになる。それら係数を並べた

$$\begin{array}{c|ccc} \alpha_1 & \beta_{11} & \cdots & \beta_{1s} \\ \alpha_2 & \beta_{21} & \cdots & \beta_{2s} \\ \vdots & \vdots & \cdots & \vdots \\ \alpha_s & \beta_{s1} & \cdots & \beta_{ss} \\ \hline & \mu_1 & \cdots & \mu_s \end{array}$$

のような表を **Stetter's notation** と呼ぶ。

[3] 1 段法なのに「段数」とは？英語では step, stage と区別があるのだが、日本語では同じ「段」となってしまって紛らわしい。残念ながら、この用語はもう定着してしまっている。

定理 5.5.1 (Runge-Kutta 型公式の収束のための条件) Runge-Kutta 型公式が収束するためには次の二条件が成立することが必要十分。

(i) 適合性 ($\rho(1) = 0$, $\rho'(1) = \sum_{j=0}^{s} \beta_j$)

(ii) 安定

公式が**前進型** (**陽的**、**explicit**) であるとは $\beta_{ij} = 0$ $(i < j)$ が成り立つこと。そうでない場合を**陰的**(**implicit**) であるという。陰的な場合でも $\beta_{ij} = 0$ $(i \leq j)$ が成り立つ場合は**半陰的**(**semi-implicit**) であるという。

Runge-Kutta 型公式の特徴として、

(1) **自己出発的**(self-starting) である。すなわち、多段法 ($k \geq 2$) で計算の最初に必要な x_1, $x_2, \cdots, x_{k-1}$ を準備することなく、計算が開始できる。

(2) 計算の途中で刻み幅 (stepsize) h の変更が簡単である ($\rightarrow$ adaptive stepsize control に便利)。

(3) 次数を大きくしようとすると、f の値の計算回数が増える。
次数 m を与えたとき、explicit で少なくとも何 stage 必要か？陽的 Runge-Kutta 型公式の場合には、以下のようになる:

m	1	2	3	4	5	6	7	8	$\cdots$
s	1	2	3	4	6	7	9	10?	$\cdots$

ここで $m \geq 5$ のとき $s > m$ となることに注意しよう (この事実が 4 次の Runge-Kutta 法が人気のある理由の一つである)。なお、s 段の公式で実現できる最高の次数を**到達可能次数**と呼ぶ。上の表から陽的 Runge-Kutta 型公式の到達可能次数が分かる。なお陰的 Runge-Kutta 型公式では、s 段で $2s$ 次を到達する公式が存在することが分かっている。

歴史　Runge (1895), Heun (1900), Kutta (1901), Gill (1940 年代), Ceschino, Butcher, 田中正次 (1960〜)

後退 Euler 法　次の公式を**後退オイラー法**(**Backward Euler's rule**) と呼ぶ:

$$x_{j+1} = x_j + hf(t_{j+1}, x_{j+1}).$$

これまでの分類で言うと、段数 1, 次数 1, implicit, である。これは単純で次数が低いが、前進 Euler 法と異なり、多少の実用性がある (特に硬い方程式に使われることがある)。

埋め込み型の公式　E. Fehlberg (1970) は次の公式 RKF45 を提案した[4]。

$$x_{j+1} = x_j + h\left(\frac{16}{135}k_1 + \frac{6656}{12825}k_3 + \frac{28561}{56430}k_4 - \frac{9}{50}k_5 + \frac{2}{55}k_6\right), \tag{5.4}$$

ただし

$$\begin{aligned}
k_1 &= f(t_j, x_j) \\
k_2 &= f\left(t_j + \frac{1}{4}h, x_j + \frac{1}{4}hk_1\right) \\
k_3 &= f\left(t_j + \frac{3}{8}h, x_j + \frac{1}{32}h(3k_1 + 9k_2)\right) \\
k_4 &= f\left(t_j + \frac{12}{13}h, x_j + \frac{1}{2197}h(1932k_1 - 7200k_2 + 7296k_3)\right) \\
k_5 &= f\left(t_j + h, x_j + h\left(\frac{439}{216}k_1 - 8k_2 + \frac{3680}{513}k_3 - \frac{845}{4104}k_4\right)\right) \\
k_6 &= f\left(t_j + \frac{1}{2}h, x_j + h\left(-\frac{8}{27}k_1 + 2k_2 - \frac{3544}{2565}k_3 + \frac{1859}{4104}k_4 - \frac{11}{40}k_5\right)\right)
\end{aligned}$$

これは 6 段 5 次の公式であるが、それだけでなく

$$x^*_{j+1} = x_j + h\left(\frac{25}{216}k_1 + \frac{1408}{2565}k_3 + \frac{2197}{4104}k_4 - \frac{1}{5}k_5\right) \tag{5.5}$$

という値を作ると、x^*_{j+1} は $x(t_{j+1})$ に対して 4 次の近似値となる。この次数の差を利用してステップ幅の自動調節をしよう、というアイディア。

いま (5.4), (5.5) の局所離散化誤差をそれぞれ τ, τ^* とおくと

$$\begin{aligned}
\tau(t,h) &= C(t)h^5 + O(h^6) \\
\tau^*(t,h) &= C^*(t)h^4 + O(h^5)
\end{aligned} \tag{5.6}$$

である。あらかじめ許容誤差限界 $\varepsilon_{\mathrm{TOL}}$ を決めておき、第 j ステップでは

$$\|x^*_{j+1} - x_{j+1}\| \le \varepsilon_{\mathrm{TOL}}$$

が成り立ったとする。これは (5.6) より高次項を無視して

$$\|C^*(t_j)h^5\| \le \varepsilon_{\mathrm{TOL}}$$

が成り立つことを意味する。そのとき、次のステップにおけるステップ幅 $\hat{h}$ は、やはり

$$\|C^*(t_{j+1})\hat{h}^5\| = \varepsilon_{\mathrm{TOL}}$$

となるように選ぶべきであろう。ステップ幅が小さいときにはもっともであると思われる $C^*(t_j) = C^*(t_{j+1})$ を仮定して

$$\|C^*(t_j)\hat{h}^5\| = \varepsilon_{\mathrm{TOL}}.$$

[4] この公式は御覧の通り大変複雑である。筆者はこの公式で誤植のドジを犯したことがある (レポート課題のプリントで間違えた)。この公式を用いたプログラムを書く時は複数の資料でチェックすることをお勧めする。

ところで近似的に

$$\|C^*(t_j)\| = \frac{\|x^*_{j+1} - x_{j+1}\|}{h^5}$$

とみなせるから、これを代入して

$$\frac{\|x^*_{j+1} - x_{j+1}\|}{h^5}\hat{h}^5 = \varepsilon_{\mathrm{TOL}}.$$

すなわち

$$\hat{h} = h\sqrt[5]{\frac{\varepsilon_{\mathrm{TOL}}}{\|x^*_{j+1} - x_{j+1}\|}}. \tag{5.7}$$

すなわち第 $j+1$ ステップでは、(5.7) で定められるステップ幅 $\hat{h}$ で公式を適用すれば、$\|x_{j+2} - x^*_{j+2}\|$ も $\varepsilon_{\mathrm{TOL}}$ の限界内にあるであろう、と考える。実際には安全率を見込んで

$$\hat{h} = \alpha h\sqrt[5]{\frac{\varepsilon_{\mathrm{TOL}}}{\|x^*_{j+1} - x_{j+1}\|}}. \tag{5.8}$$

とする。ここで α は 1 より小さい正数で普通 $0.8 \sim 0.9$ とする。こうして、次のアルゴリズムが得られる。

許容限界 $\varepsilon_{\mathrm{TOL}}$ を定めて、(5.4), (5.5) により x_{j+1}, x^*_{j+1} を求めよ。
次のステップ幅を (5.8) で求めて、計算を続行せよ。

このアイディアのキーは、 s 段 m 次公式に、f の値を計算することなく $(m-1)$ 次公式を付随させるところにある。このような Runge-Kutta 型公式を、$(m-1)$ 次公式が m 次公式に埋め込まれているといい、埋め込み型 Runge-Kutta 法と呼ぶ。

5.5.2 線形多段法

k 段法の公式のところで、Φ が線形の場合、すなわち

$$d_0x_j + d_1x_{j+1} + \cdots + d_kx_{j+k} = h\left(\beta_0f_j + \beta_1f_{j+1} + \cdots + \beta_kf_{j+k}\right)$$

を**線形多段法** (linear multistep method) と言う。ただし $f_j \overset{\text{def.}}{=} f(t_j, x_j)$.

$$\begin{cases} \text{explicit(陽的)} & \overset{\text{def.}}{\Leftrightarrow} \quad \beta_k = 0 \\ \text{implicit(陰的)} & \overset{\text{def.}}{\Leftrightarrow} \quad \beta_k \neq 0 \end{cases}$$

$\rho(\lambda) \overset{\text{def.}}{=} d_0 + d_1\lambda + \cdots + d_k\lambda^k$ の根 $\lambda_1, \cdots, \lambda_k$ について

$$|\lambda_i| \leq 1 \quad (i = 1, \cdots, k), \quad |\lambda_i| = 1 \text{ となる } \lambda_i \text{ は単根。}$$

という条件が**安定条件**である。

多段法の特徴

(1) 出発にあたって、未知の値 $x_1, \cdots, x_{k-1}$ を何らの方法で求めねばならない。

(2) 計算の途中で stepsize h を変更するのが難しい。

(3) ステップあたりの f の計算回数が少ないままで、高次の公式が作れる。

(4) 安定性に注意が必要。

PC (予測子修正子法) (準備中)

5.5.3 その他の方法

Taylor 法

f の Taylor 展開を利用する方法である。簡単で $\left(\dfrac{d}{dx}\right)^k f$ の計算しやすい f に対して有効である。

補外法

補外法(**extrapolation method**) と呼ばれる一群の方法がある。特に **Richardson 補外**に基づく **Bulirsch-Stoer 法**が有力である。

5.6 数値的安定性

今まで区間 $[a,b]$ を固定して分割数 $N \to +\infty$ とした ($h = (b-a)/N$) ときの収束を考えた。応用上は長時間解を追跡したい場合がある。この場合、刻み幅 h を固定して $n \to +\infty$ ($t \to \infty$) としても変なことが起こらないようにしたい。

注意 5.6.1 (「安定」に関する注意) 常微分方程式の数値解法の話では「安定」という語が何度も出て来たが、状況により違う意味で使われることが多い。まとめておくと

1. 微分方程式の (平衡点, あるいは周期解の) 安定性　(離散化とは無関係)
2. 数値解法の安定性　(離散化に起因する不安定性が起こらない; 丸め誤差とは無関係)
3. 数値的安定性　(丸め誤差の増幅に起因する)

ただし、それぞれの場合にさらに細かい分類がある。 ■

簡単なテスト問題に適用したときの数値的安定性を調べる、というのが基本的な作業方針 (テスト問題の枠を離れたときにどうなるかの保証はないわけで、そういう意味ではあまりいばれないが)。

線形安定性解析　$\lambda \in \mathbf{C}$ を $\mathrm{Re}\lambda < 0$ なる定数として

$$\begin{cases} x'(t) &= \lambda x(t) \quad (t \in I \stackrel{\text{def.}}{=} (a, +\infty)) \\ x(a) &= x_0 \end{cases}$$

について適用してみる。この解は $x(t) = x_0 e^{\lambda t}$ であり、$\lim_{t\to\infty} x(t) = 0$ であるが、数値解法では

$$\exists h_0 > 0,\ \exists h_1 > 0 \text{ s.t. } \begin{cases} \forall h > h_1 & \lim_{j\to\infty} |x_j| = +\infty \\ 0 < \forall h < h_0 & \lim_{j\to\infty} |x_j| = 0 \end{cases}$$

ということも起こりうる。

1) 中点則. これはいわゆる中心差分商

$$x'(t) = \frac{x(t+h) - x(t-h)}{2h}$$

を用いて作った公式

$$\begin{cases} x_{j+1} = x_{j-1} + 2hf_j, \\ x_1 \text{は適当に定める} \end{cases}$$

のことを指す。これは任意の $h > 0$ に対して $\lim_{j\to\infty} |x_j| = +\infty$ を満たす。

2) R-K 型公式. 公式の段数を s, 次数を m とすると、

$$x_{j+1} = R(\lambda h)x_j$$

のように書ける。ここで $R(z)$ は

$$R(z) = \frac{s \text{ 次以下の多項式}}{s \text{ 次以下の多項式}}, \quad |R(z) - e^z| = O(|z|^m) \quad (|z| \to 0)$$

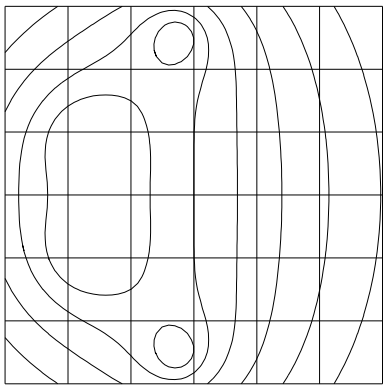
なる z の有理式である (特に公式が explicit の場合には $R(z)$ は z の多項式になる)。例えば Euler 法の場合 $R(z) = 1+z$, 古典的 Runge-Kutta 法の場合 $R(z) = 1+z+\frac{z^2}{2}+\frac{z^3}{3!}+\frac{z^4}{4!}$. この $R(z)$ に対して

$$\mathcal{R} \overset{\text{def.}}{=} \{z \in \mathbf{C}; |R(z)| < 1\}$$

を**絶対安定領域**といい、

$$\lambda h \in \mathcal{R} \Longrightarrow \lim_{j\to+\infty} x_j = 0$$

がなりたつ。絶対安定領域が左半平面 $\{z \in \mathbf{C}; \mathrm{Re}\ z < 0\}$ を含むような公式を **A-安定** (**A-stable**) という。A-安定な公式では、$\mathrm{Re}\ \lambda < 0$ なるとき、任意の $h > 0$ に対して数値解が 0 に収束する。次に示すのは Runge-Kutta 法の絶対安定領域である[5]。



なお、後述の硬い方程式の項を参照せよ。

[5] この図はどうやって描いたのか忘れてしまった。

3) LM 法. 公式

$$(\alpha_0 - z\beta_0)x_j + (\alpha_0 - z\beta_1)x_{j+1} + \cdots (\alpha_0 - z\beta_k)x_{j+k} = 0$$

に対して特性方程式を

$$(\alpha_0 - z\beta_0) + (\alpha_0 - z\beta_1)\xi + \cdots (\alpha_0 - z\beta_k)\xi^k = 0$$

で定義し、その根を $\xi_\ell(z)$ $(\ell = 1, \cdots, k)$ とする。この場合

$$\mathcal{R} \stackrel{\text{def.}}{=} \{z \in \mathbf{C}; |\xi_\ell(z)| < 1\}$$

とおくと

$$\lambda h \in \mathcal{R} \Longrightarrow \lim_{j\to\infty} x_j = 0.$$

5.7 Stiff problem (硬い問題)

以下は工学的見地からの説明である (新しい言葉に説明がついているが、数学的な定義になっていないものが多い)。

時定数 (じていすう) (time scale) とは、解が $\frac{1}{e}$ に減衰するのに必要な時間のことである[6]。 $x(t)$ が τ を正の定数として

$$x(t) = \exp\left(-\frac{t}{\tau}\right)$$

のように表されているのならば τ が時定数である。

定義 5.7.1 (硬い方程式のあいまい定義 — その 1) ある安定な微分方程式が解くべき全区間に比較して極めて小さい時定数をもつ、指数関数的に減衰する解を一つの特解として持つ時、その方程式は **stiff** である (硬い方程式である) と呼ばれる。

定義 5.7.2 (硬い方程式のあいまい定義 — その 2) 一つの問題の中に時定数が大きな部分と、小さな部分がある時、その方程式は **stiff** である (硬い方程式である) と呼ばれる。

こういう問題を数値解法で解く場合、数値的安定性の要請から小さな時定数にあわせて h を選ぶと、なかなか計算が進まない。これは数値解法に特有の困難であって、もとの問題は安定である。

例 $\lambda_1 << \lambda_2 < 0$ なる定数 λ_1, λ_2 に対して、方程式

$$\frac{d}{dt}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}$$

は硬い。

A-stable な方法があればよいが、

[6]半減期というものに似ている。

定理 5.7.1 (Dahlquist) (i) 陽的 Runge-Kutta 法は A-stable になり得ない。

(ii) 陽的線型多段法は A-stable になり得ない。

(iii) 陰的線型多段法も 3 次以上の公式は存在しない。

(iv) 2 次 A-stable 陰的線型多段法の中では台形則が「最適」である。

のような「否定的な」結果がある。そこで対応策として、

- A-stable はあきらめ、別の安定性を考える。Stiff stability (S-安定性) の概念。Gear（ギア）の方法。
- A-stable な陰的 Runge-Kutta はかなり次数の高い公式が作られている。

などが考えられる。

5.8 参考書

[1] 笠原 晧司、微分方程式の基礎、朝倉書店 (1982).
明治大学数学科の常微分方程式論の講義で長くテキストとして採用されている。数学科の教科書として標準的な内容。

[2] 三井 斌友（たけとも）、数値解析入門、朝倉書店 (1985).
常微分方程式の数値解析の専門家が書いた数少ない和書。特に理論的な解説をきちんとしてある本としてユニーク。
コンパクトにしたものに同じ著者の
微分方程式の数値解法 I, 岩波講座 応用数学、岩波書店 (1993)
がある。

[3] 戸川 隼人、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 (1992).
色々な Runge-Kutta 型公式 (made in Japan も多い) についてプログラムがたくさん載っている。

[4] 渡部 力、名取 亮、小国 力監修、Fortran 77 による数値計算ソフトウェア、丸善 (1982)
題名からするとプログラムのみの本のようであるが、そうではなく、常微分方程式の項は杉原正顕氏によるすぐれた解説がある。

[5] たくさん[7]、Numerical Recipes (in C), Cambridge University Press, 邦訳 技術評論社.
Bulirsch–Stoer 法が最善の方法であると信じている著者たちのプログラムが載っている。

[7] William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery.

5.9 おまけ — 実際的な誤差の推測

急速に α に収束する列 $\{x_n\}_{n\in\mathbf{N}}$ があるとき、

$$\varepsilon_n \stackrel{\text{def.}}{=} \|x_n - \alpha\|$$

で定義される誤差の大きさについて、十分先の番号 n に対しては

$$\varepsilon_{n+1} << \varepsilon_n$$

が成り立つので、

$$\|x_n - x_{n+1}\| \leq \|x_n - \alpha\| + \|x_{n+1} - \alpha\| = \varepsilon_n + \varepsilon_{n+1} \simeq \varepsilon_n.$$

よって

$$\|x_n - x_{n+1}\|$$

を x_n の誤差の大きさ ε_n の見積りとすることが出来る。

急速に収束しない列の場合はどうか？例えば n を分割数とした時の差分法の解 $u^{(n)}$ などでは、この仮定が成り立たないと思われる。そういう場合は例えば

$$x_j \stackrel{\text{def.}}{=} u^{(2^j)}$$

とすることによって、同じテクニックが使える。つまり例えば $n = 512$ のときの近似解と $n = 1024$ の時の近似解の差の大きさを、$n = 512$ の時の近似解の誤差の大きさの見積りとすることが出来る。

5.10　常微分方程式の初期値問題 補足

常微分方程式の初期値問題に関しては、そのうち一本にまとめて加筆した改定版を出します。

5.10.1　数値解法の次数 (order)

常微分方程式の初期値問題 (I.V.P.)

$$\frac{dx}{dt} = f(t,x) \quad (t \in I \stackrel{\text{def.}}{=} (a,b)) \tag{5.9}$$

$$x(a) = x_0 \tag{5.10}$$

に対する k 段法 (k-step formula)

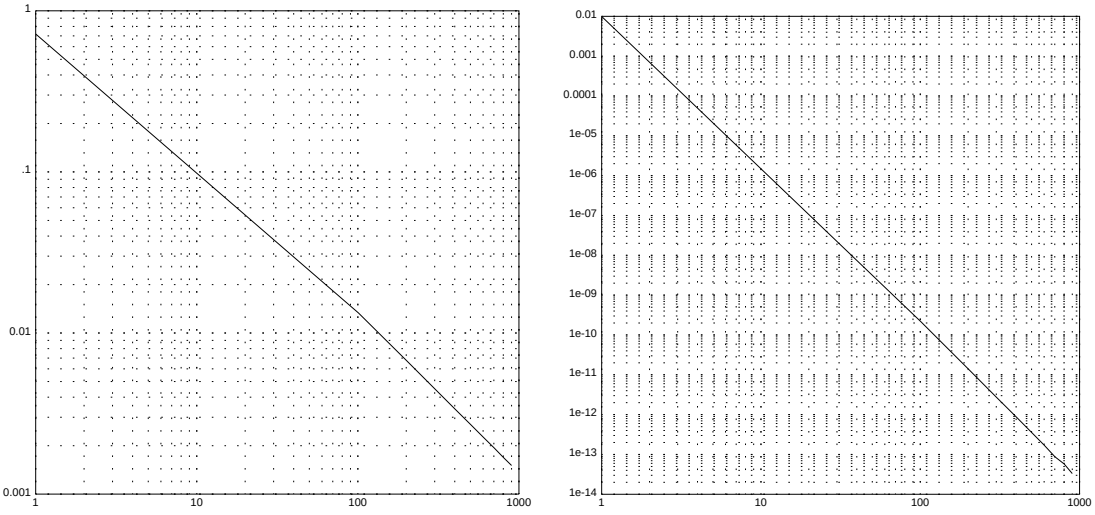
$$\begin{aligned} x_{n+k} &= a_0 x_n + a_1 x_{n+1} + \cdots + a_{k-1} x_{n+k-1} + h\Phi(t_n, t_{n+1}, \cdots, t_{n+k}, x_n, x_{n+1}, \cdots, x_{n+k}; h) \\ &\equiv L(t_n, t_{n+1}, \cdots, t_{n+k}, x_n, x_{n+1}, \cdots, x_{n+k}; h) \\ &\qquad (a_0, \cdots, a_{k-1} \text{ は定数}) \end{aligned}$$

の次数 (order) が (少なくとも) m であるとは、C^m-級の一意解を持つ任意の I.V.P. に適用した場合に、局所打ち切り誤差

$$\tau(t,h) \stackrel{\text{def.}}{=} \frac{1}{h}\left[x(t+kh) - L(t, \cdots, t+kh, x(t), \cdots, x(t+kh))\right]$$

が t に関して一様に $O(h^m)$ (as $h \downarrow 0$) であることと定義した。ここで h は刻み幅 (stepsize) である。多くの場合、累積打ち切り誤差 $\|x(b) - x_N\|$ も $O(h^m)$ になると期待される。

Euler 法は 1 次、古典的 Runge-Kutta 法は 4 次の公式である。そこで例えば $x'(t) = x$ $(t \in (0,1))$, $x(0) = 1$ という初期値問題に適用した場合の累積誤差を表示したものが次の図である (横軸は区間の分割数 N, 縦軸は累積打ち切り誤差で、いずれも対数目盛)。



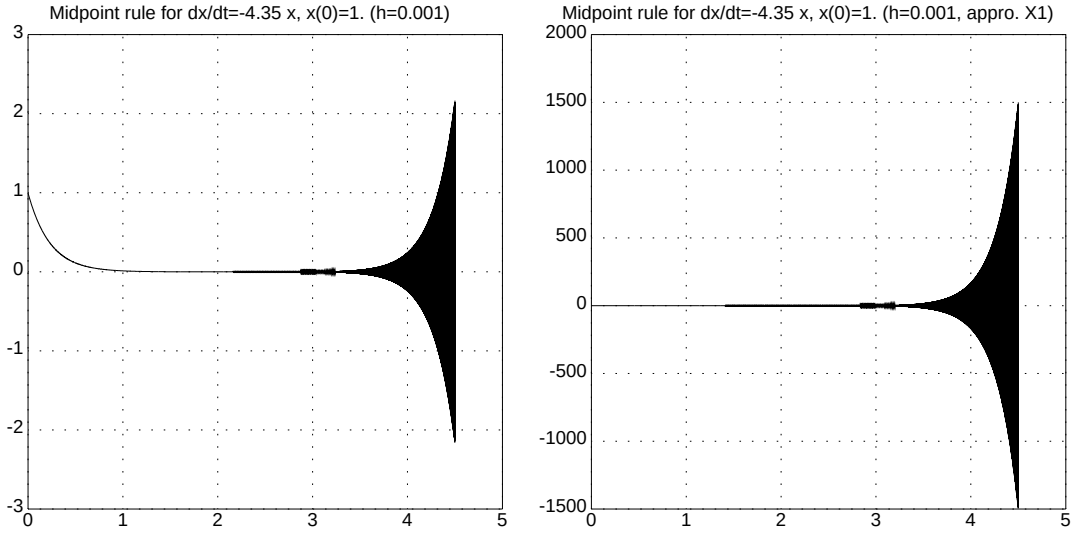
(左が Euler 法によるもの、右が Runge-Kutta 法によるもの)

実はこの問題の場合、 Euler 法では $x_{n+1} = (1+h)x_n$, Runge-Kutta 法では $x_{n+1} = (1 + h + h^2/2 + h^3/3! + h^4/4!)\, x_n$ となる。$x(t+h) = e^h x(t) = (1 + h + h^2/2 + h^3/3! + h^4/4! + \cdots + h^n/n! + \cdots)\, x(t)$ であるから、Euler 法は 1 次の項まで、Runge-Kutta 法は 4 次の項まであっていると言える。

5.10.2　スキームの数値的安定性

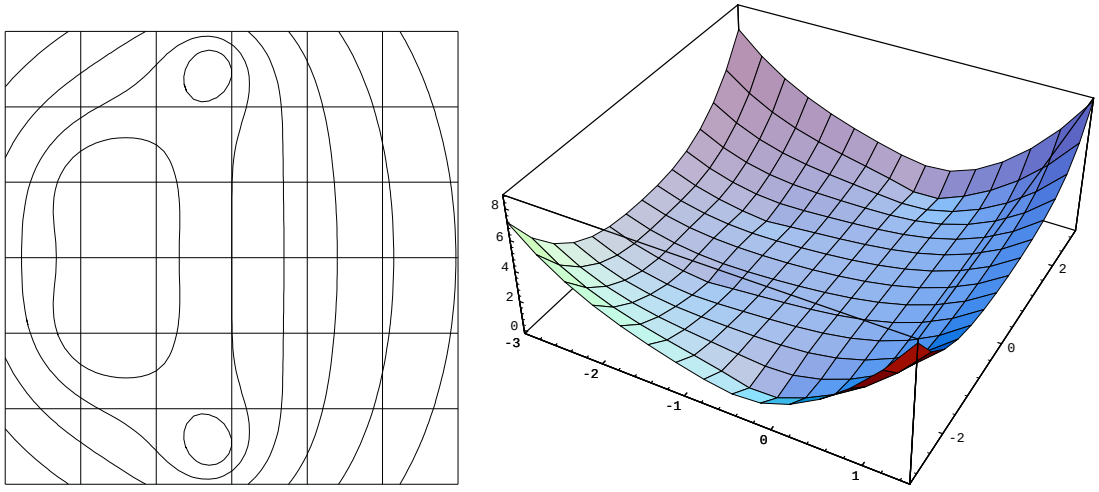
中点公式の不安定性

中心差分近似に基づく中点公式 $x_{n+1} = x_{n-1} + hf(t_n, x_n)$ で $x' = -\lambda x,\ x(0) = 1$ を解く ($\lambda = -4.35$)。左側は $x_1 = e^{-\lambda h}$ としたもの、右側は $x_1 = x_0 + hf(t_0, x_0)$ としたもの。



Runge-Kutta 法の絶対安定領域

カラーでお見せできないのが残念ですが。



なお、レベル 1 の等高線が負軸と交わる点の座標は (巾根で書けはするのだけれど...)、

$$-2.785293563405281623529759190085463034764757896716 9\cdots.$$

第6章　今後の執筆予定？

6.1　代数方程式

まとまった解説は案外と少ない。日本語で読める数少ない例外が

一松信、数値解析、朝倉書店 (1982)、第 4 章「代数方程式の数値解法」

である。

このノートに書くとしたら、まずは **Durand-Kerner 法** であろう。数学の講義で取り上げるとしたら、**Cardano の解法**についても話をした方がよいかもしれない。その他 **Bairstow 法**, **Graffe 法**, **Bernoulli 法**とか。

フランスの Durand が提唱し、ドイツの Kerner が別の角度から解釈を与えた (1969)。イギリスの Aberth が初期値の取り方について一つの提唱をした (1970)。三人の頭文字を取って **DKA 法**と呼ばれることもある。

6.2　計算機代数

「計算数学」

- 数式処理
- グレブナー基底

6.3　FFT

高速 Fourier 変換

6.4　初等関数の計算法

一松 信	初等関数の数値計算法	培風館		

6.5　特殊関数

付 録A　Strum の方法

Strum の方法とは、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 Strum の定理をもとにした多項式の根の計算法であり、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

1. 森 正武、数値解析、共立出版 (1973).
2. 一松 信、数値解析、朝倉書店 (1982).

から適当に抜き書きしたものである。

高木貞治、代数学講義 改訂新版、共立出版 (1965) の第 3 章にも書いてある。

A.1　スツルムの定理

定義 A.1.1 (Strum 列) $[a,b]$ を $\mathbf{R}$ の区間とするとき、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 $f'(x_0)f_1(x_0)>0$.

定義 A.1.2 (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.1.1 $+, +, -, 0, +, +, -$ では 3 回と数える。

定理 A.1.1 (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。

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。

定理の証明　$f(x) = 0$ の解の個数は有限個である。そのうち $[a,b]$ に含まれ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

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

の和集合において ($f(x)$ は 0 とならないので) $N(x)$ が定義できる。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 $N(a) - N(b) = n$ 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 $f_k(x) \neq 0$ とすると、$f_k(x)$ の符号は J で一定であるので (中間値の定理から、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 x_* の十分小さな近傍で $N(x)$ 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。

(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.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:

$$f_0(x) \stackrel{\text{def.}}{=} f(x), \quad f_1(x) \stackrel{\text{def.}}{=} f'(x), \tag{A.1}$$

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

(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の (-1) 倍を $f_{k+1}(x)$ とすることに注意しよう。そうする理由は以下の (A.3) を成立させるためである。)

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.2.1 $f(x) \in \mathbf{R}[x]$ 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.1), (A.2) 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.2) 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.2) から

$$f_{k-1}(x_0) = -f_{k+1}(x_0). \tag{A.3}$$

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.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.3.1 (対称性の仮定について) 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 の第 (i,j) 成分を t_{ij} と書くとき、

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

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 形にする算法を施すとき、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」ということでしょう。 ■

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

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

とブロック分けでき、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

$$p_k(\lambda) \stackrel{\text{def.}}{=} \begin{cases} \det(\lambda I_k - T_k) & (k = 1, 2, \cdots, N) \\ 1 & (k = 0) \end{cases} . \tag{A.4}$$

ただし、

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

すぐ分かる命題を二つ。

補題 A.3.1 (漸化式) (A.4) 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 行列式の展開定理を用いる。■

補題 A.3.2 (Strum 列であること) (A.4) 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 $p_0(\lambda) \equiv 1$ 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

(A.5) $$p_k(\lambda) = (\lambda - a_k)p_{k-1}(\lambda) - {b_{k-1}}^2 p_{k-2}(\lambda)$$

を微分すると、

(A.6) $$p'_k(\lambda) = p_{k-1}(\lambda) + (\lambda - a_k)p'_{k-1}(\lambda) - {b_{k-1}}^2 p'_{k-2}(\lambda).$$

$(A.6) \times p_{k-1}(\lambda) - (A.5) \times p'_{k-1}(\lambda)$ より

$$p'_k(\lambda)p_{k-1}(\lambda) - p_k(\lambda)p'_{k-1}(\lambda) = b_{k-1}^2 \left(p'_{k-1}(\lambda)p_{k-2}(\lambda) - p_{k-1}(\lambda)p'_{k-2}(\lambda)\right) + p_{k-1}(\lambda)^2$$

という漸化式が得られる。ここで

$$q_k(\lambda) \stackrel{\text{def.}}{=} p'_k(\lambda)p'_{k-1}(\lambda) - p_k(\lambda)p_{k-1}(\lambda)$$

とおくと、漸化式は

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

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

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

が示せる。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

$$N(a) \stackrel{\text{def.}}{\equiv} \text{“}\{p_0(a), p_1(a), \cdots, p_N(a)\}\text{”} \text{ の}\textbf{符号の変化数}$$

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.4 直交多項式の作る Strum 列

(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、条件

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

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

$$(u,v)_w \overset{\text{def.}}{=} \int_a^b u(x)v(x)w(x)\,dx$$

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。また $(\cdot,\cdot)_w$ に付随するノルムを $\|\cdot\|_w$ と書く:

$$\|u\|_w = \sqrt{(u,u)_w}.$$

自然数 n に対して、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。

命題 A.4.1 $H_w(a,b)$ 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.4.2 $H_w(a,b)$ 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.4.3 $H_w(a,b)$ 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.4.4 $H_w(a,b)$ 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 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.5.1 (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

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

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

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

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。 ■

付 録B　C 言語と行列

B.1　はじめに

微分方程式の数値シミュレーションのためのプログラムには、大きな行列が現れます。FORTRAN では、行列を表すのに、2 次元配列を用いるのが普通です。しかし C 言語には整合配列の機能がないので、2 次元配列を用いると、ポータビリティーのある関数を作るのが難しくなります。そこで、色々工夫することになります。

1 次元配列の方法 1 次元配列、あるいはポインターを用いて 1 次元的な連続した領域を確保し、添字計算を自前でプログラムする

ポインター配列の方法 ポインター配列、、あるいはポインターのポインターを用いて確保した領域を 2 次元配列的な記法でアクセスする

B.2　1 次元配列の方法

例えば $m \times n$ 型の行列 A を表現して、計算に用いるために

領域の確保

```
double a[m * n];
```

のように 1 次元配列 a[] を宣言するか、あるいは (こちらの方がずっと C らしいが)

```
double *a;
```

とポインターを宣言しておいて、

```
if ((a = (double *)malloc(sizeof(double) * m * n)) == NULL) {
    エラーの場合の処理
}
```

のように動的に記憶領域を確保する。

成分へのアクセス　(i, j) 成分 $(0 \leq i \leq m-1, 0 \leq j \leq n-1)$ をアクセスする時は面倒だが

```
a[n * i + j]
```

のように書くか、どこかで

```
#define A(i,j) a[n*(i)+(j)]
```

のようなマクロを定義しておいて

```
A(i,j)
```

と書く。

B.3 ポインター配列の方法

例えば $m \times n$ 型の行列 A を表現して、計算に用いるために

領域の確保 (準備その 1)

```
double *abody;
```

とポインターを宣言しておいて、

```
if ((abody = (double *)malloc(sizeof(double) * m * n)) == NULL) {
    エラーの場合の処理
}
```

のように動的に記憶領域を確保する。

ポインター配列の設定 (準備その 2)

```
double **a;
....
if ((a == (double *)(malloc(sizeof(double *) * m))) == NULL) {
    エラーの場合の処理
}
```

のようにポインターへのポインターを宣言して、必要な領域を確保した後で

```
for (i = 0; i < m; i++)
    a[i] = abody + i * n;
```

とアドレスをセットしておく。

成分のアクセス (i, j) 成分 $(0 \leq i \leq m-1, 0 \leq j \leq n-1)$ をアクセスする時は単に

```
a[i][j]
```

とかけば良い。

B.4 二つの方法の優劣

いずれが優れているか、決定打はないという感じである。

- 一次元配列の方法は成分へのアクセスがとにかく面倒である (マクロで一応は処理できるが、行列が増えるとイヤミ)。
- 一次元配列の方法は成分へのアクセスに、積和演算が必要で、CPU によっては時間がかかる。
- ポインター配列の方法は、余分なメモリーを必要とする。
- ポインター配列の方法は、成分へのアクセスに、余分なメモリー・アクセスを必要とする。システムによっては時間がかかる。
- ポインター配列の方法では、行列の行の交換が次のように非常に高速にできる。

```
double *ap;
...
ap = a[i]; a[i] = a[j]; a[j] = ap;
```

B.5 サンプル・プログラム

ポインター配列の方法で、行列を確保する関数 new_matrix() を以下に示す。

```
/*
 * test3.c --- GLSC で使うために連続した領域を確保することにした。
 */

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

typedef double scalar;
typedef scalar *vector;
typedef vector *matrix;

static maxm = 0;

/* m 行 n 列の行列を作る */
matrix new_matrix(m, n)
int m, n;
{
    int i;
    matrix a;
    vector abody;

    if ((a = (matrix)malloc((m + 1) * sizeof(vector))) == NULL)
        return NULL;
    if ((abody = (vector)malloc(m * n * sizeof(scalar))) == NULL)
        return NULL;
    for (i = 0; i < m; i++) {
        a[i] = abody;
```

```
        abody += n;
    }
    a[m] = NULL;
    if (m > maxm) maxm = m;
    return a;
}

void del_matrix(a)
matrix a;
{
    int i;
    vector atop;
    atop = a[0];
    for (i = 0; i <= maxm; i++) {
        if (a[i] == NULL)
             break;
        if (a[i] < atop)
             atop = a[i];
    }
    free(atop);
    free(a);
}

int
main()
{
    int m,n,i,j;
    matrix a;
    printf("m,n: "); scanf("%d %d", &m, &n);
    a = new_matrix(m,n);
    if (a == NULL) {
        fprintf(stderr, "new_matrix() に失敗\n");
        exit(1);
    }
    kuku(a,m,n);
    for (i = 0; i < m; i++) {
        for (j = 0; j < n; j++)
            printf("a[%d][%d]=%g ", i, j, a[i][j]);
        printf("\n");
    }
    del_matrix(a);
    return 0;
}

kuku(a,m,n)
matrix a;
int m,n;
{
    int i,j;
    for (i = 0; i < m; i++)
        for (j=0; j < n; j++)
            a[i][j] = (i + 1) * (j + 1);
}

#ifdef  __TURBOC__
double dummy() { return sin(0.0); }
#endif
```