有理数計算による対称行列の固有値問題における特性多項式の因子探索

寒川　光, 早稲田大学理工学術院

平成 29 年 4 月 14 日

目 次

1　有理算術演算 **4**
1.1　十進 BASIC の有理数モード 4
1.2　有理算術演算と浮動小数点演算 5
1.3　有理算術演算の実装 6

2　対称行列の固有値の多重度解析アルゴリズム **8**
2.1　特性多項式を求めるアルゴリズム 9
2.1.1　ヘッセンベルグ変換 9
2.1.2　重複固有値 10
2.1.3　フロベニウス変換 11
2.2　整数行列の数値例 12
2.2.1　2×2 の熱伝導行列 13
2.2.2　分割の細分化と固有値の多重度と固有ベクトル 14
2.2.3　正方格子に対するグラフ・ラプラシアン行列 19
2.3　近似解で因子多項式を探索するアルゴリズム 20
2.3.1　根と係数の関係公式の応用 21
2.3.2　浮動小数点数の有理数変換と有理数の演算の精度 21
2.3.3　包含（inclusion）と縮小反復 24
2.4　プログラムの概要 25

3　テストプログラム dvsrch.cpp **27**
3.1　処理の概要 27
3.2　根と係数の関係公式と組合せのプログラミング 29
3.2.1　根と係数の関係公式 29
3.2.2　組合せのプログラミング（十進 BASIC による準備） 30
3.2.3　根と係数の公式と組合せのプログラミング（C++） 32
3.2.4　整数性判定と有理数での整数の抽出 33
3.3　多項式の格納と除算 34
3.3.1　多項式の格納形式 34
3.3.2　多項式の除算 35
3.4　包含チェック 36
3.4.1　包含チェック 36

3.4.2 ホーナー法 36
3.5 1 次の因子探し 39
3.6 2 次以上の因子探し 39

4 有理数係数の行列の解析プログラム dvsrch1.cpp 42
4.1 桁数の多い数値に対する縮小反復による精度改良 42
4.1.1 有理数による区間算術演算 44
4.1.2 interval 型の数を用いた因子探索 45
4.1.3 多項式係数の生成関数 50
4.2 計算例 54
4.2.1 s を指定した場合の計算例 54
4.2.2 G を指定した場合の計算例 55

5 倍精度浮動小数点行列の解析プログラム dvsrch2.cpp 56
5.1 倍精度浮動小数点数行列の読込み 56
5.1.1 浮動小数点行列のファイルへの出力 57
5.1.2 浮動小数点行列のファイルからの入力 58
5.2 チェックポイントとリスタート 59
5.3 近接根の分離 60
5.3.1 近接する固有値の分離（2 根の場合） 60
5.3.2 近接する固有値の分離（3 根の場合） 62
5.3.3 零を含むグループに対する包含の数値例 64
5.3.4 挟み撃ち法 66
5.3.5 精度改良の dvsrch1.cpp に対する変更点 67
5.3.6 その他の dvsrch1.cpp に対する変更点 68
5.4 固有ベクトルの計算 68
5.5 計算例 68
5.5.1 最適化オプションなしで CT2D を作成した場合 69
5.5.2 $A = K$ を選択した場合 71
5.5.3 $A = M^{-1}K$ を選択した場合 72
5.5.4 Cygwin 環境で最適化オプション -O3 で CT2D を作成し $A = S^{-1}KS^{-1}$ とした場合 73
5.5.5 Cygwin 環境で最適化オプション -O3 で CT2D を作成し $A = K$ とした場合 75
5.5.6 Cygwin 環境で最適化オプション -O3 で CT2D を作成し $A = M^{-1}K$ とした場合 75
5.5.7 多重精度算術演算のサポートに関して 75
5.5.8 デバッグ例 76

A ヤコビ法 79
A.1 ヤコビ法のアルゴリズム 79
A.2 ヤコビ法の数値例 80

概 要

有理算術演算は,「紙と鉛筆」による筆算の延長で，扱う計算対象の数値が有理数の範囲であれば誤差なしで計算できる．このため，数値計算で誤差理論を学ぶ前の学生に，計算機による正確な計算を体験させることができる．十進 BASIC の有理数モードはこれにあたる．しかし，扱う数値の範囲が無理数に広がると，区間算術演算などの工夫が必要になったり，あるいは計算を諦めることになる.「有理数計算プログラミング環境」は，浮動小数点計算の誤差解析や数式処理との融合，数学教育などを目的として,（C++に有理数の変数型 rational を追加することで）有理算術演算と浮動小数点算術演算をひとつのプログラムで混在して使用できる（十進 BASIC では混在使用ができない）．現在は便利な解析システムやツール類が広く行きわたり，計算機を使用して種々の解析を行うにも，汎用プログラミング言語を使用する必要がほとんどなくなった．そこで大学の初年度の，力学をカリキュラムに持たない学科でも理解できる HPC プログラミングの例題「対称行列の固有値の多重度解析」を作成した．対称行列の特性多項式を，有理算術演算によって，無平方多項式の形で正確に求める．これに浮動小数点計算によって得られた固有値の近似値を組み合わせて当てはめることで，特性多項式の因数分解された形を作る．前半は，ヘッセンベルグ変換とフロベニウス変換を使用するが，これらは基本変換を使用するだけなので，線形代数の授業で習う内容と考える．後半の当てはめは，根と係数の関係公式なので，高校時代から馴染み深い．プログラミング技術に長けていれば，実質的に数式処理システムの因数分解を代替できる．この演習を通して，数学知識をプログラミング技法で生かせれば何ができるかを知り，プログラムの高速化の楽しみを体験できる．この例題はたんに教育用の題材としてだけでなく,「有理数計算プログラミング環境」に必要な変数型やユーティリティ関数，また多重精度算術演算を含める必要性などを検討する目的で開発した．

「有理数計算プログラミング環境」の紹介は，第 1 章の「有理数行列の整数行列変換と浮動小数点数の有理数変換と有理算術演算の精度」節に含めたが，本資料の姉妹編である LongintRational.pdf に詳細に解説した．

1　有理算術演算

有理算術演算（rational arithmetic）は計算機科学の黎明期からの研究テーマである．Knuth 先生の *The Art of Computer Programming* 第 2 巻の後半 *Seminumerical Algorithms* の第 4 章「算術演算」に記載されている [1]．この章は，位取り記数法，浮動小数点数演算，多倍精度の算術演算，基数変換，有理算術演算，多項式算術演算，冪級数の処理の 7 つの節で構成されており，有理算術演算は 25 ページ程度である．しかし他の節との関連が深く，また定理の証明が演習問題の解答に回されており，有理算術演算を学ぶにはかなりの紙数を読む必要がある．4.5 節「有理算術演算」の冒頭から引用する．

> ある数値問題の答を「0.333333574」と表示する浮動小数点数ではなく，正確に 1/3 であると知ることが重要である場合は多い．算術演算を分数の近似値ではなく分数に対して行えば，丸め誤差をまったく蓄積することなしに多くの計算を実行できる．これによって快適な安心感を得るが，それは浮動小数点によって計算する場合にはなかなか得られない．その安心感は，計算精度をこれ以上は改善できないことを意味している[1]．

より精度の高い計算ではなく，**正確な計算**を実現するのが有理算術演算である．倍精度演算から 4 倍精度，8 倍精度と精度を上げる「多倍精度」演算も精度改善を目的とするが，正確な計算を実現するには工夫が必要である．「精度保証計算」は区間演算を用いる方法で，Knuth 書では浮動小数点数演算の節で，浮動小数点演算の丸め方式の改善方法として説明されている．これに対し有理算術演算の原理は単純で，小学生でも分かる分数（仮分数）の四則演算を，桁数が増えてもひたすら計算する．無理数を正確に計算することはできないが，数値の範囲が有理数に収まる計算式なら，計算誤差なしで計算できる．

1.1　十進 BASIC の有理数モード

汎用プログラミング言語で，有理算術演算が可能なものは少ない．十進 BASIC は，数学教育を目的に，文教大学の白石教授が開発された [2].「日本の宝」と称賛する数学者もおられる [3]．十進 BASIC は 5 つの計算モードを持つ[2]．10 進 15 桁，10 進 1000 桁，2 進（Intel x87 FPU 命令セットによる倍精度浮動小数点），2 進による複素数，有理数である．5 つのうちいずれかのモードを選択して実行するので，数値は単一の表現に統一され，浮動小数点数と有理数を同時に使用することはできない．

有理数モードは，プログラム単位ごとに “OPTION ARITHMETIC RATIONAL” を指定するか，実行前に “p/q” をクリックしてから実行する．有理数の分母分子の桁数は 10 進数で約 2000 桁と思われる．

“LET x=0.8” と “PRINT x” の 2 行のプログラムを有理数モードで実行すると 4/5 が表示される．分母と分子を独立に見たい場合は DENOM(x) と NUMER(x) を用いる．内積を計算するプログラム例を示す[3]．

```
OPTION ARITHMETIC RATIONAL
DIM x(5),y(5)
FOR i=1 TO 5        ! i =  1   2   3   4   5
   LET x(i)=1/i     ! x(i)=1   1/2 1/3 1/4 1/5
   LET y(i)=1/(7-i) ! y(i)=1/6 1/5 1/4 1/3 1/2
NEXT i
LET s=0
FOR i=1 TO 5
   LET t=x(i)*y(i)  ! t=1/6 1/10 1/12 1/12 1/10
   LET s=s+t
   PRINT i;x(i);y(i);t;NUMER(s);DENOM(s);s
```

[1] The rational arithmetic results in a comfortable feeling of security that is often lacking when floating-point calculations have been made, and it means that the accuracy of the calculation cannot be improved upon.
[2] 十進 BASIC の使用法は 30 ページの 3.2.2 項に解説した.
[3] 感嘆符以降はコメントである.

```
    NEXT i
    END
```

実行すると次の表示が得られる．

```
 1  1  1/6  1/6  1  6  1/6
 2  1/2  1/5  1/10  4  15  4/15
 3  1/3  1/4  1/12  7  20  7/20
 4  1/4  1/3  1/12  13  30  13/30
 5  1/5  1/2  1/10  8  15  8/15
```

2 行目の出力は $i = 2$，$\mathrm{x}(2) = \frac{1}{2}$，$\mathrm{y}(2) = \frac{1}{5}$，$\mathrm{t} = \frac{1}{10}$，NUMER(s) = 4，DENOM(s) = 15，$\mathrm{s} = \frac{4}{15}$ を示しており，$\frac{1}{6} + \frac{1}{10} = \frac{16}{60} = \frac{4}{15}$ と，約分されていることが分かる．

十進 BASIC で（組込み関数を使用せずに）四則演算だけで書かれたプログラムは，筆算で数値を丸めることなしに計算した結果と一致する[4]．

1.2 有理算術演算と浮動小数点演算

十進 BASIC による，連立 1 次方程式を解くための LU 分解サブルーチンを示す（軸選択や特異性チェックは省略する）．LU 分解は上三角行列 U は $u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik}u_{kj}$ で，下三角行列 L は $l_{ij} = \frac{1}{u_{jj}}\left(a_{ij} - \sum_{k=1}^{j-1} l_{ik}u_{kj}\right)$ で表される．第 k 段の基本操作（基本変換により変数を消去する）は，対角項 $a_{k,k}$ で k 列の $k+1$ 行以下の要素を割り，右下の $(n-k)\times(n-k)$ 小行列を第 k 列ベクトルと第 k 行ベクトルで更新（階数 1 更新）する．高校時代からおなじみのガウスの消去法である[5]．

```
    EXTERNAL SUB LUdecomp(a(,),n)          !  引数 a(,) は 2 次元配列
    FOR k=1 TO n-1
       FOR i=k+1 TO n
          LET a(i,k)=a(i,k)/a(k,k)         !  対角項 a(k,k) で割る
       NEXT i
       FOR j=k+1 TO n
          FOR i=k+1 TO n
             LET a(i,j)=a(i,j)-a(i,k)*a(k,j)
          NEXT i
       NEXT j
    NEXT k
    END SUB
```

10 進 15 桁モードで，次数 $n = 5$ のフランク行列 $a_{ij} = n - \max(i,j) + 1$ を配列 a(,) に作成し，call LUdecomp(a,5) で分解した後，配列 a(,) をプリントすると次の表示を得る．

```
5  4  3  2  1
.8  .8  .6  .4  .2
.6  .75  .75  .5  .25
.4  .5  .666666666666667  .666666666666667  .333333333333333
.2  .25  .333333333333333  .500000000000001  .5
```

[4]有理数モードで利用できる組込み関数は，整数べき，ROUND，CEIL，INT，MIN，MAX，正の平方根の小数点以下を切り捨てた数を返す INTSQR，引数の整数部分の 2 を底とする対数の整数部分を返す INTLOG2，最大公約数 GCD，乱数 RND などである．

[5]十進 BASIC は “LET” は記述する必要がなく（プログラム表示窓でカーソルを移動すると自動的に挿入してくれる），また字下げも自動的に処理してくれるという便利さがある． Fortran との親和性も高いので，Perl などでスクリプトを用意しておくと，Fortran77 → BASIC の変換や，逆方向の変換も比較的容易である．

筆算では a(4,3) は $\frac{2}{3}$ になるところだが，.66666666666667 になり，その後の計算も丸められている．この丸め誤差のために，解が整数（$=1$）になる問題を解いても，$0.999\cdots98$ になる．大学の数値計算の授業では「浮動小数点計算はいい加減な計算だからこんなもの」と教えてはいけない．ここから丸め誤差の追跡を教えるので，先生にも少しタフで，学生も着いてこられない．紙と鉛筆で LU 分解を書き下せば，第 3 段で 3 で割る計算が出てくるが，ここを分数で書けば丸め誤差なしで計算できる．紙と鉛筆ではなく，有理数モードで実行すれば次の結果を得る[6]．

```
5   4   3   2   1
4/5  4/5  3/5  2/5  1/5
3/5  3/4  3/4  1/2  1/4
2/5  1/2  2/3  2/3  1/3
1/5  1/4  1/3  1/2  1/2
```

浮動小数点小数が 10 進数なので，誤差の原因は見つけやすい．**正確な解との比較は，丸め誤差に起因する問題の解決の第一歩である．**.

浮動小数点計算では，結合則 "$(a+b)+c=a+(b+c)$" や "$(a\cdot b)\cdot c=a\cdot(b\cdot c)$" が成立しない．これが問題解決を複雑にする．結合則は，数が「数学的に数」と呼ばれるためには不可欠な条件である．

フランク行列を生成する "a(i,j)=n-MAX(i,j)+1" を，乱数 "a(i,j)=RND" に変更してみよう．上三角行列の $u_{5,5}$ は分子が "1065675308$\cdots$ 略 $\cdots$276927854473" の 92 桁，分母が "7918480142$\cdots$ 略 $\cdots$0000000000" の 80 桁の数になった．浮動小数点計算ではフランク行列でも乱数行列でも計算時間に差異はないが，**有理数モードでの算術演算の時間はデータ依存性が強い**．計測してみると，フランク行列は $O(n^3)$ と，浮動小数点計算の場合と同程度に収まるが，乱数行列は $O(n^5)$ になる [4]．この理由はフランク行列は行列式が 1 という特別な行列だからである[7]．

1.3 有理算術演算の実装

「有理数計算プログラミング環境」では，有理数は `rational` 型の変数で使用する．この数は，分母と分子を `longint` 型の多桁数で保持し，これに符号を付加することで表現する．四則演算を示す．

$$r_1 \pm r_2 = \frac{b}{a} \pm \frac{d}{c} = \frac{bc \pm ad}{ac} \tag{1}$$

$$r_1 \times r_2 = \frac{b}{a} \times \frac{d}{c} = \frac{bd}{ac} \tag{2}$$

$$r_1 \div r_2 = \frac{b}{a} \div \frac{d}{c} = \frac{bc}{ad} \tag{3}$$

r_1, r_2 は有理数で，その分母・分子は a, b, c, d などの多桁数で保持される．多桁数は，基数を $r=2^{32}$ として 32 ビット符号なし整数の配列に格納し，桁数 l を整数型で保持する．

$$z = \sum_{i=0}^{l-1} d_i r^i \tag{4}$$

n 桁と n 桁の数の積は $2n$ 桁になるので，式 (1),(2),(3) の ac, bc, ad, db の桁数は演算のたびに倍になる．そこで演算結果は，分母・分子の最大公約数（Greatest Common Divisor，GCD）で割り既約にする．d_i は各桁の数で $0 \le d_i < 2^{32}$ である．零は桁数が零（$l=0$）とする．**配列長は 64 から始めて，不足すると動的に倍，倍と拡張する可変長**とした（詳しくは 21 ページの 2.3.2 項に記す）．`longint` クラスは，式 (1),(2),(3) の左辺の演算の本体である四則演算の関数や，GCD 関数をもつ．

[6] フランク行列の場合は筆算が可能だが，一般の行列で正確な解を知るための筆算は n が小さくても不可能に近い．
[7] 三角行列の対角項の総積 $\prod_{i=1}^{n} u_{i,i} = 5\cdot\frac{4}{5}\cdot\frac{3}{4}\cdot\frac{2}{3}\cdot\frac{1}{2}=1$ である．

有理数を扱う **rational クラス**を longint クラスの上に構築した．有理数 r を多桁数 n と d と符号 s で保持する．

$$r = s \times \frac{n}{d} \tag{5}$$

$s = 0$ の場合 $r = 0$ である．rational クラスでは，式 (1),(2),(3) の有理算術演算を記号 $+$, $-$, $*$, $/$ でプログラミングするための演算子多重定義のほか，`min`，`max`，`floor`，`ceil` などの関数や，浮動小数点数を有理数に変換する関数などをもつ．分母 d が 2 べきのとき，$\mathrm{GCD}(n, d)$ は，分子 n の下位の 0 ビットを取り除いた数になる．

ここまでで，有理算術演算の問題点が計算時間にあることが分かる．Knuth 書の最初の版が書かれた時代（約半世紀前）と現在を比較すると，計算速度は飛躍的に増大した[8]．最初のスーパーコンピュータ（スパコン）と言われた Cray-1 は 1976 年に登場したが 160Mflop/s のピーク性能値であった．京速コンピュータの「京」は 10Pflop/s を意味し，16Pflop/s に達した Blue Gene/Q （2013 年）で 1 億倍である．しかしどんなに速くなっても，計算量のオーダーが高いと，計算可能な問題の規模は限定される．

有理算術演算の鍵は除算にある．計算機で行われる機械語の整数除算命令は，被除数を除数で割り，商と余りを 2 つのレジスタに返す．浮動小数点除算命令は，被除数を除数で割り，IEEE 754 浮動小数点標準の規格に合わせて丸められた商をを返す．Fortran や C は “`z=x/y`” において，変数が整数なら整数演算の結果である商を，変数が浮動小数点数なら浮動小数点演算の結果である丸められた商を左辺値に返す．BASIC や Perl などのインタプリータ型の言語では “`z=x/y`” において，小数点以下の数値も，電卓のように，数桁の丸められた商を左辺値に返す．このように，汎用プログラミング言語で記述された除算命令の左辺の値はいずれも正確ではない．加減算や乗算は桁数を拡張可能にしておけば正確に行えるが，除算は割り切れない場合があるので厄介である．有理算術演算では除算 (3) と乗算 (2) は，除数の分母と分子を入れ替えることを除けば，計算機にとって同じ処理である．**有理算術演算では，四則演算の結果は正確で，また有理数である**．数学では有理数は**体（field）**をなすという．計算機で正確な除算を実現するもうひとつの方法（整数が体をなすの）はモジュラー算術演算である[9]．

汎用プログラミング言語 Fortran は浮動小数点数を `real` と呼び，C は `float` や `double` と呼ぶ．有限のビット数で表現される数なので実数であるはずはなく，実数を誤差を許して近似する表現（representation）という意味で命名された．数学的な分類に従えば有理数である．具体例を挙げる．0.1 は 2 進数で $.0\dot{0}01\dot{1} = \frac{0}{2} + \frac{0}{4} + \frac{0}{8} + \frac{1}{16} + \frac{1}{32} + \cdots$ と循環小数になる．倍精度浮動小数点の “0.1” を正確に有理数に変換すると，循環小数の下位のビットが切り捨てられるために正確に 0.1 にはならず，次の分数になる．

$$0.1 \fallingdotseq \frac{3,602,879,701,896,397}{36,028,797,018,963,968} \tag{6}$$

分子の 10 倍から 2 を引くと分母に一致する．倍精度浮動小数点数の有意桁（significand）は 53 ビットあるので，10 進数で 16 あるいは 17 桁以降には丸め誤差が現れる．式 (4) で定義された数は，上限が限定された自然数の一部，式 (1),(2),(3) の r_1, r_2 はそれを分母・分子にもつ有理数である．

---

[8] Knuth 書の第 1 版は 1969 年に出版された．

[9] モジュラー算術演算は，整数論の原理（中国剰余定理）に基づく剰余計算を行う（除算は商は捨てて余りを返す）ことで，扱える整数の範囲は限定されるが正確な計算が実現される．これは，共通因子を持たないいくつかの「法」 $p_1, p_2, \ldots, p_r$ を用意し，数 u について直接的に計算する代わりに,「剰余」 $u \bmod p_1$, $u \bmod p_2$, ... , $u \bmod p_r$ について間接的に計算する（除算は商は捨てて余りを返す）．使用できる整数の範囲は，素数 p_i から構成される総積の範囲に限定されるが，オペランド整数は体をなす [1]．モジュラー算術演算は数式処理で使用され，Knuth 書では「多倍精度の算術演算」の 1 項として説明されている．

2 対称行列の固有値の多重度解析アルゴリズム

浮動小数点演算による対称行列の固有値問題は，数値計算の一般的なテーマである．全固有値を求める場合，ヤコビ法やギブンス法が用いられる．浮動小数点演算では，固有値が重なる「重根」なのか，接近していて異なる固有値なのかは分からない．つまり，数学的に興味のある部分には手が届かない．さらに，固有値が零近傍にある場合は，計算誤差が数値計算ライブラリやコンパイラに依存するので，同じプログラムでも固有値が正になったり負になったりする現象も現れる．

これに対し，有理算術演算による方法はアプローチが全く異なる．ヤコビ法やギブンス法は三角関数を用いるため，有理算術演算の利点である正確な計算ができない．そこで線形代数の教科書にあるように，対称行列を特性多項式に変換し，特性方程式の根を求める．固有値が有理数なら正確な解が，無理数でも必要な精度の解が得られる．本稿で扱う「多重度解析」は，浮動小数点演算によるプログラムから抽出された対称行列を，有理算術演算による方法で特性多項式を得る．ここまでは十進 BASIC でも計算可能であるが，さらに浮動小数点演算によって得られた近似固有値を利用して，その精度を高め，特性多項式の因数分解を得ることで，多重度の解析も行う．浮動小数点演算でコンパイラの最適化オプションを変更すると，浮動小数点演算では行列要素の下位のビットがわずかに変化し，その影響で重根が消滅する現象も解析する．有理数と浮動小数点数を混在させてプログラミングできる環境が，数値シミュレーションプログラムの開発で有用なツールとなるために有効な機能が何であるかを知ることも，次章以降で述べるプログラムを開発した目的であった．本章では次章以降で使用するアルゴリズムの数学的側面を解説する．固有値問題は，次の線形同次方程式の n 個の未知数 λ を決定する．

$$A\boldsymbol{x} = \lambda\boldsymbol{x}. \tag{7}$$

本稿で扱う行列 A は**対称**で，その要素は**有理数**とする．要素が有理数の行列は，全要素の分母の最小公倍数（Least Common Multiplyer，LCM）を掛けることによって整数行列に変換できる．固有方程式 (7) の係数を整数に変換すれば，整数を係数行列の要素とする対称行列の固有値問題となる．方程式 (7) は次式が成立したとき，非自明解をもつ．

$$\det(A - \lambda I) = 0. \tag{8}$$

式 (8) の左辺の行列式は展開されて次式を得る．

$$(-1)^n\lambda^n + p_{n-1}\lambda^{n-1} + p_{n-2}\lambda^{n-2} + \cdots + p_0 = 0 \tag{9}$$

これを係数行列 A の特性方程式（characteristic equation）と呼ぶ[10]．行列 A の由来が，幾何学的な対称性などで重根をもつはずでも，浮動小数点演算では丸め誤差のために重根か近似根かは判定できない．

多重度解析アルゴリズムのポイントを述べる．

- 特性方程式 (9) の左辺の多項式の全係数は，固有方程式 (7) の行列要素 a_{ij} に対する乗算と加減算だけで得られるので，行列要素が整数なら，特性方程式 (9) の係数 p_i も整数である．
- 多重度が高い（重根がある）場合は，ヘッセンベルグ（Hessenberg）変換の副対角項が零になる．
- 多重度が高い場合は，ヘッセンベルグ行列は複数のヘッセンベルグ小行列に分離され，各小行列をフロベニウス（Frobenius）変換して得られる特性方程式は重根を持たない（特性多項式が無平方（squarefree）である）．

---

[10] $n = 2$ の場合 $\begin{vmatrix} a_{11}-\lambda & a_{12} \\ a_{21} & a_{22}-\lambda \end{vmatrix} = \lambda^2 - (a_{11}+a_{22})\lambda + a_{11}a_{22} - a_{12}a_{21}$，$n = 3$ の場合 $\begin{vmatrix} a_{11}-\lambda & a_{12} & a_{13} \\ a_{21} & a_{22}-\lambda & a_{23} \\ a_{31} & a_{32} & a_{33}-\lambda \end{vmatrix} = -\lambda^3 + (a_{11}+a_{22}+a_{33})\lambda^2 - (a_{12}a_{21}+a_{23}a_{32}+a_{13}a_{31})\lambda + \det(A)$ になる．$p_n = (-1)^n$，$p_{n-1} = (-1)^{n-1}(a_{11}+a_{22}+\cdots+a_{nn})$，$p_0 = \det(A)$ である．

- 特性方程式の最高次の係数は 1 か -1 なので，決めなくてはいけない係数 $p_0, p_1, \cdots, p_{n-1}$ は n 個あり，近似固有値も n 個で等しい．整数行列なら「根と係数の関係公式」を用いて，近似固有値から「整数になるはずの係数」を求められる[11]．計算誤差を含む近似固有値から根と係数の関係公式で求める係数が，許容範囲内に整数があるかどうかの判定を「整数性」の判定と呼ぶことにする．

- 「整数性」の判定は絶対誤差評価で行うので，整数性判定には近似固有値の精度が十分になくてはならない．本稿で用いるアルゴリズムは，近似固有値を「包含」（inclusion）する区間を見つけ，2 分法で精度改良する．これらは有理算術演算による正確な計算で可能である．

はじめの 3 点は有理算術演算を用いた正確な計算ができれば可能である（十進 BASIC でもできる）．4 点め以降は，浮動小数点数を使うので，有理数と浮動小数点数を混在させてプログラミングできる環境が必要である．これらの項目の数学的説明を本章で行い，次章以降でプログラムに沿った説明を行う．

2.1 特性多項式を求めるアルゴリズム

A をヘッセンベルグ行列 H に変換すると，多重度が高い固有値が存在する場合は，副対角項に零要素が現れる．その後の計算は各ヘッセンベルグ小行列に対して独立に行える．

2.1.1 ヘッセンベルグ変換

次数 n の対称行列 A に基本変換行列 R を $(n-2)$ 回，**相似変換**（similarity transformation）の形で掛ける[12]．

$$H = R_{n-2} \cdots R_2 R_1 A R_1^{-1} R_2^{-1} \cdots R_{n-2}^{-1}. \tag{10}$$

計算順序は基本変換行列 R の添字の順序に行う．

浮動小数点演算では，このアルゴリズムは非対称行列に対して使用されることはあるが，ヘッセンベルグ行列が非対称行列であるため，対称行列に対して用いられることはめったにない[13]．**対称行列でも，計算が誤差なしで行われる場合は，対称性保持の必要はない**．これは有理算術演算でアルゴリズム選択の重要な特長である．

次数 5 のヘッセンベルグ行列を示す．

$$H = \begin{pmatrix} h_{11} & h_{12} & h_{13} & h_{14} & h_{15} \\ k_2 & h_{22} & h_{23} & h_{24} & h_{25} \\ & k_3 & h_{33} & h_{34} & h_{35} \\ & & k_4 & h_{44} & h_{45} \\ & & & k_5 & h_{55} \end{pmatrix}. \tag{11}$$

対角項の下の副対角項 $h_{s+1,s}$ を k_{s+1} で表す．ヘッセンベルグ行列への変換アルゴリズムは次のようになる;

第 s 段階までに，元の行列 $A \equiv A_1$ は A_s になっており，その最初の $(s-1)$ 行および列は上ヘッセンベルグ形である．第 s 段階は次の一連の操作からなる．

[11] 有理数係数の特性多項式を通分して，整数係数の多項式を作ると，近似固有値の数よりも決めなくてはならない係数の数が 1 つ多くなるので，根と係数の関係公式が使えない．

[12] $A\boldsymbol{x} = \lambda\boldsymbol{x}$ の両辺に H^{-1} を掛けると $H^{-1}A\boldsymbol{x} = \lambda H^{-1}\boldsymbol{x}$ になるが，$H^{-1}H = I = HH^{-1}$ なので

$$H^{-1}A\underbrace{HH^{-1}}_{=I}\boldsymbol{x} = \lambda H^{-1}\boldsymbol{x}$$

すなわち $(H^{-1}AH)H^{-1}\boldsymbol{x} = \lambda H^{-1}\boldsymbol{x}$ となる．$H^{-1}AH = B$，$H^{-1}\boldsymbol{x} = \boldsymbol{y}$ とおけば，$B\boldsymbol{y} = \lambda\boldsymbol{y}$ の固有方程式になる．λ は同じなので，**相似変換により固有値は保存**され，**固有ベクトルは H^{-1} が掛けられる**．

[13] 実数の固有値を持つはずの問題でも，非対称行列の固有値問題を，浮動小数点計算で解くと（固有値計算の反復計算で使用するアルゴリズムにもよるが），丸め誤差のために虚数部分に計算誤差が現れるので，`complex` 型で解く必要がある．

- 第 s 列の対角要素より下側で，絶対値最大の要素を捜す．それが零なら，次の 2 項目を飛ばし，この段階は終わる．そうでないなら，その最大要素のあった行の番号を s' と置く．
- 第 s' 行と第 $(s+1)$ 行を交換する．これは軸選択の手順である．この交換を相似変換にするため，列 s' と列 $(s+1)$ も交換する．
- $i = s+2, s+3, \ldots, n$ に対して，乗数

$$r_{i,s+1} = \frac{a_{is}}{a_{s+1,s}} \tag{12}$$

  を計算する．行 $(s+1)$ に $r_{i,s+1}$ を掛けて行 i から引く．この消去を相似変換にするため，列 i に $r_{i,s+1}$ を掛けて列 $(s+1)$ に加える[14]．第 s **列の要素** a_{is} **を消去するために** R **の** $s+1$ **列目に非零項を使うところが，ガウス消去法で用いる基本変換とは異なる**.

この段階が全体で $(n-2)$ 回必要である．

関数 elmhes は，数値計算分野では，非対称行列の固有値を計算するルーチンとして一般的に使用されるものとほぼ同じで，*Numerical Recipes* には ELMHES サブルーチンとして紹介されている [5, p. 478].

2.1.2 重複固有値

行列が縮重（derogatory）していて，複数の重複する固有値が存在する場合は，ヘッセンベルグ行列の副対角項に零要素が現れる．この計算は正確に行う必要があるので，有理算術演算を用いる（浮動小数点演算では正確に零にならず，小さな値が残るので，この目的には役に立たない）．この現象は，対称行列を対称 3 重対角行列に変換すると，非対角項に零要素が現れる現象に対応している．Wilkinson の古典的教科書から引用する．

行列 A が 3 重対角化された状態を考える．

$$SAS^{-1} = \begin{pmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_{n-1} \\ & & \beta_{n-1} & \alpha_n \end{pmatrix}.$$

> 対称行列が多重度 k を持つ場合，ギブンス法かハウスホルダー法によって 3 重対角化された行列は，少なくとも $(k-1)$ 個の副対角項を持つ．一方，ギブンス法かハウスホルダー法によって 3 重対角化された行列が零副対角項を持っても，それは多重度の高い固有値の存在を意味するとは限らない．

この引用に続く証明は，スツルム列の性質を使用し，非線形方程式を解く漸化式の計算に沿った方法でなされている [6, p. 300]．ヘッセンベルグ行列のスツルム列の計算式は 3 重対角行列の場合よりも複雑なので，ここではヘッセンベルグ行列の階数から証明する．

Proof. 対称行列 A が縮重していると仮定する．この場合，多重度の高い固有値を λ_j とすると，この固有値に対応するジョルダン細胞が複数存在する．A を変換して得られるヘッセンベルグ行列 H は，λ_j に対応する固有ベクトルを複数もつ．これは $(H-\lambda_j I)$ の階数が $(n-2)$ 以下であることを意味する．もし，副対角項 $k_i \neq 0,\ \ i = 2, \cdots, n$ とするなら，行列 $(H-\lambda_j I)$ の第 1 列から 第 $(n-1)$ 列は線形独立なので [6, p. 406], $(H-\lambda_j I)$ の階数が $(n-1)$ 以上であることを意味する．これは仮定に反する．したがって，$\beta_j = 0$ となる行列 A に対しては，必ず $k_{j+1} = 0$ となる． □

---

[14] $$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -r_{32} & 1 & 0 \\ 0 & -r_{42} & 0 & 1 \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & r_{32} & 1 & 0 \\ 0 & r_{42} & 0 & 1 \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & a'_{13} & a'_{14} \\ k_2 & h_{22} & a'_{23} & a'_{24} \\ 0 & a'_{32} & a'_{33} & a'_{34} \\ 0 & a'_{42} & a'_{43} & a'_{44} \end{pmatrix}$$

対称行列の 3 重対角化の場合と同様，ヘッセンベルグ行列が零副対角項を持っても，それは多重度の高い固有値の存在を意味するとは限らない．

なお，浮動小数点計算で 3 重対角化を行う場合は，$\beta_j = 0$ の判定は丸め誤差のために，閾値の設定に苦慮するが，計算誤差を含まない有理算術演算では「イコール零」でチェックできる．**浮動小数点演算による「丸め誤差を考慮した零判定」のようなプログラミングの難しさはない**．

$k_{r+1} = 0$ の場合は次のように書ける．

$$H = \begin{pmatrix} H_r & B_r \\ 0 & H_{n-r} \end{pmatrix},$$

H_r は次数 r，H_{n-r} は次数 $(n-r)$ で，どちらも上ヘッセンベルグ形である．行列式 (8) は次のように計算できる．

$$\det(H - \lambda I) = \det(H_r - \lambda I)\det(H_{n-r} - \lambda I),$$

つまり，$k_{r+1} = 0$ が検出されれば，後続の計算は 2 つの小行列に対して独立に進められる．繰り返しになるが，$k_{r+1} = 0$ が検出できるのは正確な計算の賜物で，浮動小数点演算ではできない．

2.1.3 フロベニウス変換

特性多項式を求めるために，上ヘッセンベルグ行列 H_r または H_{n-r} をフロベニウス標準形 F に変換する．この標準形は，有理数行列の要素間の四則演算（有理演算）のみで計算可能なので，有理標準系とも呼ばれる[15]．

本項では，行列の添字は（H_r に付けられた "r" は行列のサイズを表していたが），消去の段階を表す．消去は $(n-1)$ 段の主たるステップからなる．もとの行列は $H \equiv H_1$ とする．$n = 5$, $s = 3$, の場合には $R_3 H_3 R_3^{-1}$ を計算する．

$$\begin{pmatrix} 1 & & & -r_{14} & \\ & 1 & & -r_{24} & \\ & & 1 & -r_{34} & \\ & & & 1 & \\ & & & & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & h_{13} & h_{14} & h_{15} \\ k_2 & 0 & h_{23} & h_{24} & h_{25} \\ & k_3 & h_{33} & h_{34} & h_{35} \\ & & k_4 & h_{44} & h_{45} \\ & & & k_5 & h_{55} \end{pmatrix} \begin{pmatrix} 1 & & & r_{14} & \\ & 1 & & r_{24} & \\ & & 1 & r_{34} & \\ & & & 1 & \\ & & & & 1 \end{pmatrix}$$

i が 1 から s について，次の操作を行う [6, p. 406];

- $r_{i,s+1} = \dfrac{h_{is}}{k_{s+1}}$ を求める．
- 行 i から，$r_{i,s+1}\times$ 行 $(s+1)$ を引く．
- i が 1 から s について，$k_{i+1} r_{i,s+1}$ を $h_{i,s+1}$ に加える．

第 3 段の終了時点では次のようになる．

$$\begin{pmatrix} 0 & 0 & 0 & h'_{14} & h'_{15} \\ k_2 & 0 & 0 & h'_{24} & h'_{25} \\ & k_3 & 0 & h'_{34} & h'_{35} \\ & & k_4 & h_{44} & h_{45} \\ & & & k_5 & h_{55} \end{pmatrix}, \qquad h'_{ij} = h_{ij} + k_{i+1} r_{ij}.$$

[15] ジョルダン標準系は，一般に，体の拡大を必要とする [6, p. 16].

この変換操作を $(n-1)$ 回繰り返し，相似変換を完結させる．行列 X が得られる．

$$X = \begin{pmatrix} 0 & 0 & 0 & 0 & x_0 \\ k_2 & 0 & 0 & 0 & x_1 \\ & k_3 & 0 & 0 & x_2 \\ & & k_4 & 0 & x_3 \\ & & & k_5 & x_4 \end{pmatrix}.$$

フロベニウス行列は対角行列を相似変換 $F = D^{-1}XD$ の形で掛けて得られる．ここに対角行列 D は $d_1 = 1,\ d_2 = k_2,\ d_3 = k_2k_3,\ d_4 = k_2k_3k_4,\ d_5 = k_2k_3k_4k_5,$ である[16]．

$$F = \begin{pmatrix} 0 & 0 & 0 & 0 & p_0 \\ 1 & 0 & 0 & 0 & p_1 \\ & 1 & 0 & 0 & p_2 \\ & & 1 & 0 & p_3 \\ & & & 1 & p_4 \end{pmatrix}, \qquad p_i = \left(\prod_{j=0}^{n-2-i} k_{n-j} \right) x_i. \tag{13}$$

次数 5 の特性多項式が $-\lambda^5 + p_4\lambda^4 + p_3\lambda^3 + p_2\lambda^2 + p_1\lambda + p_0$ として得られる．最高次の係数は “−1” であるが n が偶数の場合は “+1” として，全係数 p_i の符号を反転させる．次数 n の特性方程式は式 (9) である[17]．ヘッセンベルグ行列を経由してフロベニウス行列を求める相似変換は，1 つの行列 T にまとめられ，これを用いて相似変換の形で $F = TAT^{-1}$ と記述できる．多重度が高い場合は F はブロック対角行列となる．

$$TAT^{-1} = \begin{pmatrix} F_1 & & & \\ & F_2 & & \\ & & \ddots & \\ & & & F_m \end{pmatrix}. \tag{14}$$

特性多項式は各フロベニウス小行列 F_i から得られる特性多項式の積になる．

$$p(\lambda) = \prod_{i=1}^{m} p_i(\lambda) \tag{15}$$

各特性多項式 $p_i(\lambda)$ は，重根をもたないので無平方（squarefree）である．

2.2 整数行列の数値例

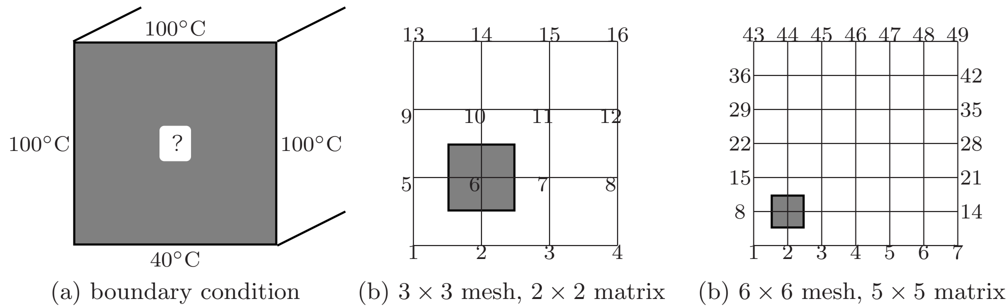
整数を行列要素とする 2 種類の行列を用いて，前節で述べた変換の具体的な姿を示す．はじめに熱伝導を表現する行列，次に正方格子に対するグラフ・ラプラシアン行列を用いて，多重度の高い固有値問題を計算する．

正方形断面の四角柱で，表面の温度を与えられた場合の熱伝導問題を，断面領域で考える [7, p. 6]．図 1 に，内部の未知数が 2×2 節点と 5×5 節点の場合を示す．1 つの差分要素を中図と右図に示した．熱量は上下左右の要素から，温度勾配と熱伝導係数に比例して流入する，5 点差分スキームを用いる[18]．解析領域を $l \times l$ の要素に分割すると，内部の節点（未知数）は $(l-1) \times (l-1)$ になる．$m = l - 1$ とおくと，未知数は $n = m \times m$ になる．未知数は内側の節点にあるので，行列の行・列番号は，境界節点の番号を詰めて振られる．3×3 の分割では未知数 2×2 になるので，行列の行・列番号では節点 6 が最初の行になり，節点 7 が 2 番目，節点 10 が 3 番目，節点 11 が 4 番目になる．四隅の節点 1 と 4 と 13 と 16 は数値解には関係しない．

[16] D を右から掛ける操作は，1 列目はそのまま，2 列目は k_2 倍，3 列目は k_2k_3 倍，…，n 列目は $k_2k_3\cdots k_n$ 倍する．D^{-1} を左から掛ける操作は，1 行目はそのまま，2 行目は k_2 で割り，3 行目は k_2k_3 で割り，…，n 列目は $k_2k_3\cdots k_n$ で割る．したがって $p_0 = k_2k_3k_4k_5x_0$，$p_1 = k_3k_4k_5x_1$，$p_2 = k_4k_5x_2$，$p_3 = k_5x_3$，$p_4 = x_4$ になる．なお，式 (13) の p_i は教科書の (52.6) 式とは異なる [6, p. 407]．

[17] 特性方程式を $(-1)^n \left(\lambda^n - p_{n-1}\lambda^{n-1} - p_{n-2}\lambda^{n-2} - \cdots - p_0\right) = 0$ と記述したほうが，式 (13) の係数 p_i の符号に忠実かもしれない．

[18] 5 点差分では，自身の点の温度が，上下左右の点の温度の平均になる．



(a) boundary condition (b) 3×3 mesh, 2×2 matrix (b) 6×6 mesh, 5×5 matrix

図 1: 四角柱の内部の温度分布

2.2.1 2×2 の熱伝導行列

熱伝導係数を 1 にすると係数行列は次のようになる．

$$A = \begin{pmatrix} 4 & -1 & -1 & 0 \\ -1 & 4 & 0 & -1 \\ -1 & 0 & 4 & -1 \\ 0 & -1 & -1 & 4 \end{pmatrix} \tag{16}$$

第 1 段の消去は $a_{3,1} = -1$ を消去する．$R_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ と $R_1^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ を用いて $R_1 A R_1^{-1}$ により $A' = \begin{pmatrix} 4 & -2 & -1 & 0 \\ -1 & 4 & 0 & -1 \\ 0 & 0 & 4 & 0 \\ 0 & -2 & -1 & 4 \end{pmatrix}$ となる．第 2 段の消去は，$a_{3,2}$ と $a_{4,2}$ を交換すると $a_{4,2} = 0$ であるため行われない[19]．3 列と 4 列を交換し，次の式 (17) のようになる．

行列は物理的な形状の対称性のために縮重して，ヘッセンベルグ行列は次のようになる．

$$H = \begin{pmatrix} 4 & -2 & 0 & -1 \\ -1 & 4 & -1 & 0 \\ & -2 & 4 & -1 \\ & & 0 & 4 \end{pmatrix} \tag{17}$$

$k_4 = 0$ なので，H は H_3 と H_1 に分かれる[20]．大きいほうの行列から

$$H_3 = \begin{pmatrix} 4 & -2 & 0 \\ -1 & 4 & -1 \\ & -2 & 4 \end{pmatrix}, \quad X_3 = \begin{pmatrix} 0 & 0 & 24 \\ -1 & 0 & 22 \\ & -2 & 12 \end{pmatrix} \tag{18}$$

[19] HesFrb.BAS を実行すると分かりやすい．2 進モードでは x87 FPU の命令セットの倍精度浮動小数点演算で実行される．5×5 分割では特性多項式の係数は精度が不足して正しく計算されない．

[20] 節点の順序付けを変更（例えば 2 と 4 を交換）しても，$k_4 = 0$ となる．

が得られ，特性多項式が得られて，それは整数の範囲で因数分解できる[21].

$$p_3(\lambda) = 48 - 44\lambda + 12\lambda^2 - \lambda^3 = (2-\lambda)(4-\lambda)(6-\lambda). \tag{19}$$

小さいほうの小行列 H_1 から特性多項式 $p_1(\lambda) = -\lambda + 4$ が得られる．

因数分解（因子探索）は小さいほうの H_1 から始めるので，1 次の因子 $\lambda - 4$ がはじめに得られる．次に $p_3(\lambda)$ は $\lambda - 4$ で割り切れる，商多項式 $\lambda^2 - 8\lambda + 12$ を因数分解することで $(2-\lambda)(6-\lambda)$ を得る．

行列 (16) の固有値の重根 $\lambda_2 = \lambda_3 = 4$ の場合，$\lambda I - A$ と重根 $\lambda = 4$ に対応する固有ベクトルは次のようになる．

$$4I - A = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}, \qquad \boldsymbol{v}_2 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ -1 \end{pmatrix}, \quad \boldsymbol{v}_3 = \begin{pmatrix} 0 \\ 1 \\ -1 \\ 0 \end{pmatrix}$$

固有ベクトル $\boldsymbol{v}_2$ は，行列の 1 列と 4 列に同じ値があるので打ち消し合って，$(A - 4I)\boldsymbol{v}_2 = 0$ になる．$\boldsymbol{v}_3$ は 2 列と 3 列に同じ値があるので打ち消し合って，$(A - 4I)\boldsymbol{v}_3 = 0$ になる．重根に対応する固有ベクトルの非零要素の位置は，図 1 の中の図では，節点番号で 6 と 11，および 7 と 10 で，両者は中央の要素の対角線の位置にある．

2.2.2　分割の細分化と固有値の多重度と固有ベクトル

分割を細かくすると行列サイズは大きくなり，多重度も増える．

3×3 の場合　係数行列 A をヘッセンベルグ行列に変換すると，$k_6 = k_9 = 0$ となり，3 つの小行列に分かれる．

$$H_5 = \begin{pmatrix} 4 & -2 & 0 & 0 & 0 \\ -1 & 4 & -\frac{3}{2} & 0 & 0 \\ & -2 & 4 & -2 & 0 \\ & & -\frac{3}{2} & 4 & -1 \\ & & & -2 & 4 \end{pmatrix}, \quad H_3 = \begin{pmatrix} 4 & -2 & 0 \\ -\frac{1}{2} & 4 & -\frac{1}{2} \\ & -2 & 4 \end{pmatrix}, \quad H_1 = 4$$

これらの小行列から得られる特性多項式は次のようになる．

$$\begin{aligned} p_1(\lambda) &= 4 - \lambda \\ p_3(\lambda) &= 56 - 46\lambda + 12\lambda^2 - \lambda^3 = (4-\lambda)(\lambda^2 - 8\lambda + 14) \qquad (20) \\ p_5(\lambda) &= 448 - 816\lambda + 520\lambda^2 - 150\lambda^3 + 20\lambda^4 - \lambda^5 \qquad (21) \\ &= (4-\lambda)(\lambda^2 - 8\lambda + 14)(\lambda^2 - 8\lambda + 8) \end{aligned}$$

因数分解は，$p_3(\lambda)$ は $p_1(\lambda)$ で割り切れ，$p_5(\lambda)$ は $p_3(\lambda)$ と $p_1(\lambda)$ で割り切れる．

多重度は 3 が最大になる．表 1 に固有値を示した．9 個の固有値のうち，3 個の固有値が有理数で，6 個が無理数である．固有値が有理数か無理数であるかは，対応する固有ベクトルを求める場合は重要になる．有理数固有値に対応する固有値ベクトルは正確に計算できる．

固有ベクトルの計算法は，固有値が有理数で正確に得られた場合は，ヘッセンベルグ・フロベニウス変換の過程は使わず，元の行列 A に戻り，$A - \lambda I$ を係数行列とする連立同次線形方程式の非自明解として得る．(紛らわしいが，$A - \lambda I$ を改めて行列 A と書くと)，$A\boldsymbol{x} = \boldsymbol{0}$ は $\det(A) = 0$ の場合に $\boldsymbol{x} = \boldsymbol{0}$ の自明解以外の非自明

[21]「デカルトの符号法則「実係数の方程式 $f(x) = a_0x^n + a_1x^{n-1} + \cdots + a_n = 0$ $(a_n \neq 0)$ の正根の数は $f(x)$ の符号変移の数に等しいか，それより偶数個少ない．ただし k 重根は k 個に数える」によると，式 (19) は -1, $+12$, -44, $+48$ で 3 回の符号変移より 3 個または 1 個の正根をもつ．$p_3(-\lambda)$ の符号変移は 1, $+12$, $+44$, $+48$ で零，つまり負根はない．両者と対称行列が実固有値を n 個持つことから，3 つの正根を持つことが言える．

表 1: 3×3 の場合の固有値

固有値 λ_i	近似固有値 e_i	正確な固有値 λ_i
λ_1	$1.171572876\cdots$	$4-2\sqrt{2}$
λ_2 λ_3	$2.5857\cdots$	$4-\sqrt{2}$
λ_4 λ_5 λ_6	4	4
λ_7 λ_8	$5.41421356\cdots$	$4+\sqrt{2}$
λ_9	$6.8284\cdots$	$4+2\sqrt{2}$
H_5 H_3 H_1		

解（non-trivial solution）を持つ．n よりも r だけ階数が低い $n \times n$ の行列 A の左上 $(n-r)\times(n-r)$ の小行列 A_{11} が正則とする．$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}\begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ で，上の式は $A_{11}\boldsymbol{x}_1 + A_{12}\boldsymbol{x}_2 = 0$ であり，A_{11} は正則なので，$\boldsymbol{x}_1$ と $\boldsymbol{x}_2$ の間には $\boldsymbol{x}_1 = -A_{11}^{-1}A_{12}\boldsymbol{x}_2$ の従属関係がある．ここで $\boldsymbol{x}_2$ を任意の r 要素からなるベクトルとしたとき，ベクトル $\begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{pmatrix}$ は $A\boldsymbol{x} = 0$ を満たす非ゼロのベクトルである．$\lambda_4 = \lambda_5 = \lambda_6 = 4$ の場合，$4I - A$ の階数が 3 つ落ちるので，任意の $\boldsymbol{x}_2$ ベクトルとして $(0\,0\,1)^T$，$(0\,1\,0)^T$，$(1\,0\,0)^T$ とする．この計算法による非自明解（固有ベクトル）を示す．

$$4I - A = \left(\begin{array}{ccc|ccc|ccc} 0&1&0&1&0&0&0&0&0 \\ 1&0&1&0&1&0&0&0&0 \\ 0&1&0&0&0&1&0&0&0 \\ \hline 1&0&0&0&1&0&1&0&0 \\ 0&1&0&1&0&1&0&1&0 \\ 0&0&1&0&1&0&0&0&1 \\ \hline 0&0&0&1&0&0&0&1&0 \\ 0&0&0&0&1&0&1&0&1 \\ 0&0&0&0&0&1&0&1&0 \end{array}\right), \quad \boldsymbol{v}_4 = \begin{pmatrix} 1\\0\\0\\0\\-1\\0\\0\\0\\1 \end{pmatrix}, \quad \boldsymbol{v}_5 = \begin{pmatrix} 0\\1\\0\\-1\\0\\-1\\0\\1\\0 \end{pmatrix}, \quad \boldsymbol{v}_6 = \begin{pmatrix} 0\\0\\1\\0\\-1\\0\\1\\0\\0 \end{pmatrix}$$

行列の各行で見ると，非零要素の位置は対角項の隣（+1）と 3 つ先（+3）の列にあるので，その間隔で 1 と -1 を配したベクトル $\boldsymbol{v}$ と行ベクトルの内積は零となる．この 3 本のベクトルが，階数 $9-3$ の行列 $4I - A$ の非自明解（A の 3 本の固有ベクトル）である．

なお，$\boldsymbol{v}_4$ と $\boldsymbol{v}_6$ は直交していない．**固有ベクトルの直交が保証されるのは，固有値が異なる場合だけ**である．この場合で記すと，3 重根に対応する固有ベクトルの 1 次結合 $\boldsymbol{v} = \alpha\boldsymbol{v}_4 + \beta\boldsymbol{v}_5 + \gamma\boldsymbol{v}_6$ はすべて固有ベクトルになるからである．$\boldsymbol{v}_6$ をシュミットの直交化で $\boldsymbol{v}_4$ に直交させると $\boldsymbol{v}_6' = \frac{1}{\boldsymbol{v}_4^T\boldsymbol{v}_4}(\boldsymbol{v}_4^T\boldsymbol{v}_6)\boldsymbol{v}_4 = \left(-\frac{1}{3}\ 0\ 1\ 0\ -\frac{2}{3}\ 0\ 1\ 0\ -\frac{1}{3}\right)^T$ になる．

浮動小数点演算（jacobev）で得られた固有ベクトルを示す．

$$\boldsymbol{u}_4=\begin{pmatrix}0.441059\\-0.017461\\0.252991\\0.017461\\-0.694050\\0.017461\\0.252991\\-0.017461\\0.441059\end{pmatrix},\quad \boldsymbol{u}_5=\begin{pmatrix}-0.329597\\0.300986\\0.450423\\-0.300986\\-0.120826\\-0.300986\\0.450423\\0.300986\\-0.329597\end{pmatrix},\quad \boldsymbol{u}_6=\begin{pmatrix}0.268016\\0.398877\\-0.328808\\-0.398877\\0.060792\\-0.398877\\-0.328808\\0.398877\\0.268016\end{pmatrix}$$

ヤコビ法では，初期値を単位ベクトルから始めて，それに固有値を求めるために作成した面内回転行列を繰り返し掛けて固有ベクトルを求めるので，互いに直交する $\boldsymbol{u}_4$，$\boldsymbol{u}_5$，$\boldsymbol{u}_6$ が得られる[22]．

$(4I-A)\boldsymbol{u}=0$ を満たすベクトル $\boldsymbol{u}$ の条件を考察する．ベクトル $\boldsymbol{u}$ の第 k 要素を $u(k)$ で表すと，行列 $\lambda I-A$ は $\lambda=4$ のとき対角項が零になり，奇数番目の行と列は，偶数番目の行と列と独立になる．$4I-A$ の各行の非零要素の位置から 9 つの条件が得られるが，これらを要約すると，固有ベクトルは次の条件を満たせばよいことが分かる（上は偶数番目の行，下は奇数番目の行から得られる）[23]．

$$u(5)=u(1)+u(3),\quad u(5)=u(7)+u(9)$$
$$u(2)=u(8)=-u(4)=-u(6)$$

$\boldsymbol{v}_4$,$\boldsymbol{v}_5$,$\boldsymbol{v}_6$ も $\boldsymbol{u}_4$,$\boldsymbol{u}_5$,$\boldsymbol{u}_6$ も，$\boldsymbol{v}'_6$ もすべてこの条件を満たしている．しかしヤコビ法による固有ベクトルは $4I-A$ が持っている構造的な特質,「奇数番目の行と列は，偶数番目の行と列と独立」とは無関係に非零要素が詰まっているので，教科書的には適切ではない．この場合は，教科書的な $\boldsymbol{v}_4$ と $\boldsymbol{v}_6$ が素直で,（重根だと分かっているので）直交化も必要ない．このようなベクトルが素朴なアルゴリズムで求まることは，有理算術演算の利点である．

無理数の固有値に対応する固有ベクトルの要素には無理数が入るので，正確な固有ベクトルを得ることはできない．しかし因数分解で因子 $(\lambda^2-8\lambda+8)$ が $\lambda_1=4-2\sqrt{2}$, $\lambda_9=4+2\sqrt{2}$ のペアに由来することが分かる．これから，浮動小数点演算で得られた固有ベクトルの近似値を[24]，正確な固有ベクトルに改良できる．これには，固有方程式 $A\boldsymbol{x}=\lambda\boldsymbol{x}$ の左辺の $\boldsymbol{x}$ に固有ベクトルを代入するとき，無理数 $\sqrt{2}$ がかかる項と有理数の項を分けて計算する．因子が 2 次式であるとき，そのペアが分かっているので，対応する近似固有ベクトルを用いて代入すると比較的簡単に正確なベクトルを得られる．λ_1 と λ_9 に対応する固有ベクトルを示す．

$$A=\left(\begin{array}{ccc|ccc|ccc}4&-1&0&-1&0&0&0&0&0\\-1&4&-1&0&-1&0&0&0&0\\0&-1&4&0&0&-1&0&0&0\\\hline -1&0&0&4&-1&0&-1&0&0\\0&-1&0&-1&4&-1&0&-1&0\\0&0&-1&0&0&4&0&0&-1\\\hline 0&0&0&-1&0&0&4&-1&0\\0&0&0&0&-1&0&-1&4&-1\\0&0&0&0&0&-1&0&-1&4\end{array}\right),\quad \boldsymbol{v}_1=\begin{pmatrix}1\\\sqrt{2}\\1\\\sqrt{2}\\2\\\sqrt{2}\\1\\\sqrt{2}\\1\end{pmatrix},\quad \boldsymbol{v}_9=\begin{pmatrix}1\\-\sqrt{2}\\1\\-\sqrt{2}\\2\\-\sqrt{2}\\1\\-\sqrt{2}\\1\end{pmatrix}$$

---

[22] 固有ベクトルを計算するアルゴリズムは，9 ページの「相似変換」の脚注を参照のこと．

[23] $u(2)+u(4)=0,\ u(1)+u(3)+u(5)=0,\ u(2)+u(6)=0,\ u(1)+u(5)+u(7)=0,\ u(2)+u(4)+u(6)+u(8)=0,\ u(3)+u(5)+u(9)=0,\ u(4)+u(8)=0,\ u(5)+u(7)+u(9)=0,\ u(6)+u(8)=0$

[24] $\boldsymbol{v}'_1=(0.250000\ 0.353553\ 0.250000\ 0.353553\ 0.500000\ 0.353553\ 0.250000\ 0.353553\ 0.250000\ )^T$ が浮動小数点演算の固有ベクトルである．4 倍すれば，0.353553 は 1.414212 になる．これを $\sqrt{2}$ として固有方程式に代入すれば正確な解であることが確かめられる．

表 2: 4×4 の場合の固有値

固有値 λ_i				近似固有値 e_i	正確な固有値 λ_i
λ_1				0.7639320225	$3-\sqrt{5}$
λ_2	λ_3			1.7639320225	$4-\sqrt{5}$
λ_4				2.7639320225	$5-\sqrt{5}$
λ_5		λ_6		3.0	3
λ_7	λ_8	λ_9	λ_{10}	4.0	4
λ_{11}		λ_{12}		5.0	5
λ_{13}				5.2360679775	$3+\sqrt{5}$
λ_{14}	λ_{15}			6.2360679775	$4+\sqrt{5}$
λ_{16}				7.2360679775	$5+\sqrt{5}$
H_9	H_3	H_3	H_1		

A の対角項は 4 で，その両隣と 3 つ前後に -1 がある．この配置に対応して，$\boldsymbol{v}_1$ は奇数番目の要素が 1 で，偶数番目の要素は $\sqrt{2}$ なので，$A\boldsymbol{v}_1$ の奇数番目の要素は $4-2\sqrt{2}=\lambda_1$，偶数番目の要素は $4\sqrt{2}-4=\sqrt{2}\lambda_1$ になる．なお $\boldsymbol{v}_1^T\boldsymbol{v}_9=0$ で，この場合，直交している．

4×4 の場合　固有値の分布を表 2 に示す．2 つの H_3 の特性多項式は異なっているので，割り切る関係は成立しない．16 個の固有値のうち，8 個が有理数で，7 個が無理数である．多重度は 4 になり，行列 $4I-A$ の各行で見ると，非零要素の位置は対角項の隣（+1）と 4 つ先（+4）の列にあるので，その間隔で 1 と -1 を配したベクトル $\boldsymbol{v}$ が，階数 $16-4$ の行列 $4I-A$ の非自明解（A の 4 本の固有ベクトル）である．節点の位置は，対角線上とたすき掛けの線上になる（「たすき掛け」の意味は，図のある 5×5 分割で後述する）．

5×5 の場合　図 1 の右のように 5×5 にすると，行列のサイズは $n=25$ になる．ヘッセンベルグ行列は 5 つの小行列 $H_{13}, H_9, H_1, H_1, H_1$ に分かれる．固有値の分布を表 3 に示す．多重度 5（$\lambda_{11}=\cdots=\lambda_{15}$）が存在する．11 個の固有値が有理数で，14 個の固有値が無理数である．

多重度 5 に対応する固有ベクトルは，2 通りの対角線上の節点をつなぐ 9,17,25,33,41 節点と 13,19,25,31,37 で 2 本，たすき掛けの 12,18,24,30,38,31,26,20 と 10,18,26,34,40,32,24,16 で 2 本，それに中央の点 25 とその周りの正方形 11,17,23,31,39,33,27,19 をつなぐ点に 1 と -1 をもつ 5 本である．

多重度 5 の固有値に対応する固有ベクトルは，行列 $4I-A$ の各行で見ると，非零要素の位置は対角項の隣（+1）と 5 つ先（+5）の列にあるので，その間隔で 1 と -1 を配したベクトル $\boldsymbol{v}$ が，階数 $25-5$ の行列 $4I-A$ の非自明解（A の 5 本の固有ベクトル）である．

$m \times m$ の場合　ヘッセンベルグ行列が H_a，H_b などのヘッセンベルグ小行列に分かれる状態を表 4 に示した．表では $4-\lambda$ の 1 次式の因子が複数現れる場合を “3*1” のように表した．すべてのケースで $\lambda=4$ の固有値が分割数に一致する多重度 m を持つ．また分割数 m の最大の小行列 H_a のサイズと $m+1$ の 2 番目の小行列 H_b のサイズが，$m=4$ と $m=11$ を例外として一致している．なお，本稿で使用するプログラム dvsrch.cpp では $m=13$ の場合は組合せ数が大きすぎて，`long long` 型の変数の最大値を超えるため計算できない（後述）．

一般的に，多重度 m は，行列 $4I-A$ の各行で見ると，非零要素の位置は対角項の隣（+1）と m 先（$+m$）の列にあるので，その間隔で 1 と -1 を配したベクトル $\boldsymbol{v}$ が，階数 $n-m$ の行列 $4I-A$ の非自明解（A の m 本の固有ベクトル）である．m が奇数の場合は，中央の点とその周りの正方形が入るが，偶数の場合は対角線とたすき掛けで偶数本の固有ベクトルの組が m 本構成される．

表 3: 5×5 の場合の固有値

固有値 λ_i					近似固有値 e_i	正確な固有値 λ_i
λ_1					.535898384862	$2(2-\sqrt{3})$
λ_2	λ_3				1.26794919243	$3-\sqrt{3}$
λ_4					2.0	2
λ_5	λ_6				2.26794919243	$4-\sqrt{3}$
λ_7	λ_8				3.0	3
λ_9	λ_{10}				3.26794919243	$5-\sqrt{3}$
λ_{11}	λ_{12}	λ_{13}	λ_{14}	λ_{15}	4.0	4
λ_{16}	λ_{17}				4.73205080757	$3+\sqrt{3}$
λ_{18}	λ_{19}				5.0	5
λ_{20}	λ_{21}				5.73205080767	$4+\sqrt{3}$
λ_{22}					6.0	6
λ_{23}	λ_{24}				6.73205080757	$5+\sqrt{3}$
λ_{25}					7.46410161514	$2(2+\sqrt{3})$
H_{13}	H_9	H_1	H_1	H_1		

表 4: 分割数 m に対する小行列サイズと因子多項式の次数 d

m	2	3	4	5	6	7	8	9	10	11	12	13
n	4	9	16	25	36	49	64	81	100	121	144	169
H_a	3	5	9	13	19	25	33	41	51	55	73	85
H_b	1	3	3	9	13	19	25	33	41	37	61	73
H_c		1	3	3*1	4*1	5*1	6*1	7*1	8*1	15	10*1	11*1
H_d			1							7		
H_e										7*1		
d	1	2	2	2	3	4	3	4	5	4	6	?

(a) 4×4 分割　　(b) 7×7 分割

図 2: 正方格子に対するグラフ・ラプラシアン行列

2.2.3　正方格子に対するグラフ・ラプラシアン行列

熱伝導行列では四角柱の表面温度を固定したが，この境界条件なしの場合を考える．グラフ理論では，節点を「頂点」といい，節点と節点の接続関係を「枝」という．グラフ・ラプラシアン行列は，各行，列がそれぞれ頂点に対応し，対角成分にはその頂点に接続する枝の数，非対角成分は枝がある部分は -1，ない部分は 0 が入る．図 1 の中のメッシュ図で m は正方形領域の一辺の分割数としたが，グラフ理論では，図 2 の左のように 1 方向の節点数とする．節点番号は，熱伝導問題でも問題に関与していなかった正方形断面の四隅は除外して数える．節点数は $(m-2)^2+4(m-2)$ になる．$m=4$ の場合は 12 節点で，図左のように接続したグラフは，次のラプラシアン行列（対称行列）になる．

$$A = \left(\begin{array}{cc|cccc|cccc|cc}
1 & & & -1 & & & & & & & & \\
 & 1 & & & -1 & & & & & & & \\
\hline
 & & 1 & -1 & & & & & & & & \\
-1 & & -1 & 4 & -1 & & & -1 & 0 & & & \\
 & -1 & & -1 & 4 & -1 & & 0 & -1 & & & \\
 & & & & -1 & 1 & & & & & & \\
\hline
 & & & & & & 1 & -1 & & & & \\
 & & & -1 & 0 & & -1 & 4 & -1 & & -1 & \\
 & & & 0 & -1 & & & -1 & 4 & -1 & & -1 \\
 & & & & & & & & -1 & 1 & & \\
\hline
 & & & & & & & -1 & & & 1 & \\
 & & & & & & & & -1 & & & 1
\end{array}\right) \tag{22}$$

図 1 の熱伝導行列は，図 2 では下の 1, 2 と左の 3, 7 と右の 6, 10 と上の 11, 12 節点の温度を固定したので，連立 1 次方程式の変数から除外する境界条件を与えた．2×2 の熱伝導行列の境界条件を外すと 4×4 のグラフ・ラプラシアン行列になる[25]．図には網掛けで，境界条件を指定した場合の行列要素を示した．

固有値の分布は表 5 のようになる．行列の各行の和は零なので，零固有値（固有ベクトルは全要素が 1 ）をもつ[26]．$\lambda=1$ の 4 重根が 3 つの H_1 で分離される．H_2 から得られる特性多項式は $(\lambda^2-5\lambda+2)$ である．H_7 から得られる特性多項式は $(-24\lambda+134\lambda^2-267\lambda^3+234\lambda^4-92\lambda^5+16\lambda^6-\lambda^7)$ と，定数項がなく，零固有値を持つことが分る．これは H_1 と H_2 の特性多項式で割り切れて $(\lambda^4-10\lambda^3+25\lambda^2-12\lambda)$ が得られ，$\lambda(\lambda-3)(\lambda^2-7\lambda+4)$ と分解される．

4 重根に対応する固有ベクトルは，節点 1 と 2 が 1 と -1 で他は零，節点 3 と 7 が 1 と -1 で他は零，節点 9 と 10 が 1 と -1 で他は零，節点 11 と 12 が 1 と -1 で他は零の 4 本である．**これらのベクトルは正方形状に由来する周辺での局所的なモード**なので，熱伝導行列の場合のように，行列全体をカバーするモードではないので，**分割を増やしても多重度は 4 のまま**である．

[25] 行列 (22) の対角項が 4 の行と列だけ残して，それ以外の行と列を取り除くと，式 (16) の 2×2 の熱伝導行列になる．

[26] 温度が平衡状態にある温度が得られたら，各節点に同じ温度を加えた温度分布もやはり平衡状態を保つことを意味する．

表 5: 4×4 分割のグラフ・ラプラシアン行列の固有値

λ_i					approx. e_i	exact λ
λ_1					2.37e-16	0
λ_2	λ_3				0.438447187	$0.5(5-\sqrt{17})$
λ_4					0.627718673	$0.5(7-\sqrt{33})$
λ_5		λ_6	λ_7	λ_8	0.99999999	1
λ_9					2.99999999	3
λ_{10}	λ_{11}				4.561552812	$0.5(5+\sqrt{17})$
λ_{12}					6.392281323	$0.5(7+\sqrt{33})$
H_7	H_2	H_1	H_1	H_1		

表 6 に m を 4 から 12 までの固有値の分布を示す．$m=7$ を例外として，多重度 4 は最後の 3 つの小行列と H_a に現れる．表にはヘッセンベルグ小行列の次数と，それが既知の因子で割切れる場合その次数を括弧に入れて示し，既知でない因子に分解される場合その次数を示した．例えば $m=9$ では，H_e は 1 次式の因子となり，H_d の 1 次の因子は既知の因子で割切れるので括弧を付けた．H_c も同様である．$H_b=H_{32}$ は既知の因子では割れずに 5 次，9 次，18 次の因子に分解される．$H_a=H_{42}$ は既知の 1，5，9，18 次の因子で割切れて 9 次式になり，1 次と 2 つの 4 次の因子に分解される．最下段に因子多項式の最高次の次数を示した．

熱伝導行列と比較すると，**次数の高い因子があるため，計算時間が長くなる**．多重度 4 に対応する固有ベクトルは，正方形の形状の周囲の局所モードで，図 2 の 7×7 の場合は，節点 1 と 6 が 1 と -1 で他は零，節点 5 と 12 が 1 と -1 で他は零，節点 34 と 41 が 1 と -1 で他は零，節点 40 と 45 が 1 と -1 で他は零である．

2.3 近似解で因子多項式を探索するアルゴリズム

前節では正確に計算された特性多項式の因数分解された形を述べたが，これは浮動小数点演算で得られた固有値の近似値を，根と係数の関係公式に当て嵌めて，整数性から誤差を取除いた整数の係数を抽出して除算することで見つけた[27]．この処理は，浮動小数点演算と有理算術演算の両者を使用する場合に一般的に使える方法と考えられる．本節で，探索のアルゴリズムの鍵となる精度改良について説明する．

表 6: グラフ・ラプラシアン行列の分割数 m に対する小行列サイズと因子多項式の最高次数 d

m	4	5	6	7	8	9	10	11	12
n	12	21	32	43	60	77	96	117	140
H_a	7	12	18	25	33	42-(1,5,9,18),1,4,4	52-(1,9,9,23),1,4,5	63-(1,9,14,28),1,1,4,5	75
H_b	2	6	7	15	14	32-5,9,18	23-23	42-14,28	34
H_c	1	1	4	1	10	1-(1)	18-9,9	9-9	28
H_d	1	1	1	1	1	1-(1)	1-(1)	1-(1)	1
H_e	1	1	1	2	1	1-1	1-(1)	1-(1)	1
H_f			1	1	1		1-1	1-1	1
d	2	4	7	10	14	18	23	28	

[27]数学知識としては，高校時代からお馴染みに根と係数の関係と組合せ ${}_nC_r$ しか使わない．プログラミングが達者でさえあれば，大学初年度でも，対称行列の特性方程式の因数分解をプログラミングによって得ることができる．

2.3.1　根と係数の関係公式の応用

倍精度計算で得られた固有値の近似値 e_i を小数で表 3 の第 2 列に示した.「有理数計算プログラミング環境」は，有理数と倍精度浮動小数点数を，1 つのプログラムで使用できる点が十進 BASIC とは異なる．行列 A が整数行列のときは特性方程式の係数も整数になる．特性多項式が次数の低い多項式の積に分解される場合，浮動小数点計算で得られた固有値を用いて，特性多項式の因数分解された形を，簡単なフィルタープログラムで作成できそうに思える．

特性多項式が 2 次式に因数分解されるとする．モニック 2 次方程式 “$x^2+rx+s=0$” に対する根と係数の関係公式 “$\alpha+\beta=-r$” と “$\alpha\beta=s$” を使う．2 つの候補 e_i と e_j を近似固有値から選び，$r=-(e_i+e_j)$ と $s=e_i\times e_j$ を作成し，

$$\mathrm{Floor}(|r|)-|r| \;\text{ or }\; \mathrm{Ceil}(|r|)-|r|$$
$$\text{and}$$
$$\mathrm{Floor}(|s|)-|s| \;\text{ or }\; \mathrm{Ceil}(|s|)-|s|$$

が許容範囲に入るかどうかを調べる．これを**整数性判定**ということにする[28]．範囲内であれば r と s を整数化して多項式の除算 $p_r(\lambda)\div(x^2+rx+s)$ を試み，剰余が零ならこの 2 次多項式は特性多項式の因子（divisor）であることが分かる．すべての i と j の組合わせを探索する．

例えば表 3 の 2 つめと 8 つめを選んで $\alpha=1.2679$, $\beta=4.7321$ と 5 桁で計算すると，$r=6.0000$, $s=5.9998\cdots$ となるので，許容範囲が 0.0002 で整数が係数の 2 次多項式 x^2-6x+6 が得られる．つまり “$x^2-6x+6=(x-(3-\sqrt{3}))(x-(3+\sqrt{3}))$” のペアを見つける[29]．

ここでの例題は，表 4 に示したように，未知数の分割が 5×5 以下は，整数の範囲での因数分解が 2 次以下の因子に分解される．6×6 の場合は 3 次式を含む．この場合はモニック 3 次方程式 “x^3+rx^2+sx+t” に対する公式 “$\alpha+\beta+\gamma=-r,\quad \alpha\beta+\beta\gamma+\gamma\alpha=s,\quad \alpha\beta\gamma=-t$” によって 3 つ子組を探す．36 個の近似固有値から 3 つ選んで有効数字 5 桁，$\alpha=0.95108$, $\beta=5.3569$, $\gamma=5.6920$ で計算すると，$r=-11.99998$, $s=40.9998\cdots$, $t=-28.9998\cdots$ が得られる．

7×7 の場合は 4 次式を含む．7×7 の場合，陰的ではあるが，解はフェラーリのべき根の公式から

$$\lambda=\pm\frac{\sqrt{8-2^{\frac{5}{2}}}}{2}-\sqrt{2}+4,\ \pm\frac{\sqrt{2^{\frac{5}{2}}+8}}{2}+\sqrt{2}+4.$$

を正確な解と考えることができる．表 4 の最下段から，9 分割以下，および 11 分割では 4 次多項式以下に分解されるので，べき根による n 次方程式の根の公式を正確な解と考えれば，正確な固有値を得ることができる．10 分割は 5 次多項式以下に，12 分割は 6 次多項式以下に分解される．

どの近似固有値がどの因子に繋がるかが分かるので，特性多項式の構成を考えることができる（固有値を数値として何 10 桁求めてみても，これは分からない）．また，前述したように，2 次の因子であれば，固有ベクトルを正確に得るためのヒントにもなる．

2.3.2　浮動小数点数の有理数変換と有理数の演算の精度

現実の浮動小数点プログラムが生成した行列を解析するのに先立って，前節で解析した整数行列に，人為的な操作を加えて桁数の多い問題を作り，因子探索を行う．これによってどの程度の精度改良をすれば，根と係数の関係から因子探索ができるかを調べる．

[28] 自然数が素数であることを英語では primality というが，数がある範囲内に整数をもつかどうかを，本稿では「整数性」という．この語に対する英語が見つからなかったので，primality に類似した造語 integer に ty を付して短縮した inty をコメントや関数名に使用する．

[29] $\{x-(p+\sqrt{q})\}\{x-(p-\sqrt{q})\}=x^2-2px+p^2-q$

代数方程式 $a_0x^n+\cdots+a_{n-1}x+a_n=0$ の解を $\alpha_1,\cdots,\alpha_n$ とするとき，その定数 k 倍 $k\alpha_1,\cdots,k\alpha_n$ を解とする方程式は,

$$a_0x^n+a_1kx^{n-1}+a_2k^2x^{n-2}+\cdots+a_{n-1}k^{n-1}x+a_nk^n=0 \tag{23}$$

である．k が 10 桁で，$n=10$ なら k^n は 100 桁になる．

行列要素が有理数の行列の特性多項式 (9) は，最高次の係数 a_0 が 1 か -1 の有理数係数である．全係数の分母の最小公倍数（LCM）を求めて掛ける（通分する）ことで，整数係数の多項式に変換すると，モニック（最高次の係数が 1 あるいは -1）でなくなり，未知数が $n+1$ 個になり，近似固有値の組合せから係数を整数性によって決めることができなくなる．そこで特性多項式を通分するのではなく，**もとの行列のほうを通分（分母の LCM 倍）**して，最高次の係数が 1 か -1 の整数係数の特性多項式を求めることにする．

熱伝導係数を 0.1 にすると，式 (16) の係数行列 A は次のように変わる．

$$A'=\begin{pmatrix} 0.4 & -0.1 & -0.1 & 0 \\ -0.1 & 0.4 & 0 & -0.1 \\ -0.1 & 0 & 0.4 & -0.1 \\ 0 & -0.1 & -0.1 & 0.4 \end{pmatrix} \tag{24}$$

全要素の分母の LCM は 10 なので，10 倍すれば行列 A になる．固有方程式の係数行列 A を 10 倍した行列 $A'=10A$ の固有値は，行列 A の固有値の 10 倍である．0.1 は 10 進数では循環小数（recurring decimal）にならないが，2 進数では循環小数になる．0.1 を倍精度浮動小数点数で作成して，これを有理数に変換してみよう．

倍精度浮動小数点数 a を，e を指数，d_i を 0 または 1 として 2 進数で次のように表す．

$$a=\left(\pm\sum_{i=0}^{52}d_i2^{-i}\right)\cdot2^e=A\cdot2^e \tag{25}$$

$B=A\cdot2^{52}$ は 2^{54} よりも小さい整数である．B を用いると式 (25) は，$a=B\cdot2^{e-52}$ となり，$C=52-e$ とおくと $a=B\div2^C$ と書け,「分子は B，分母は 2^C」と有理数表現される．**有意桁（significand）が 2 進数の浮動小数点数を有理数変換すると，分母 2 べき**になる．0.4 と 0.1 は次の有理数に変換される．64 ビットで表現されていた浮動小数点数を有理数変換すると，2 倍の 128 ビットに増える場合が多いが，増加は分母に零ビットを入れて数値の桁を合わせるためで，精度としてはどちらも 53 ビットしか使っていない．同じ情報を，浮動小数点数表示では指数部の 11 ビットに詰めるが，有理数は分母の自然数で表すのでビット数が増える．

0.1 と 0.4 を，式 (6) の再記を含めて記す．

$$\begin{aligned} 0.4 &\fallingdotseq \frac{3,602,879,701,896,397}{9,007,199,254,740,992}=\frac{3,602,879,701,896,397}{2^{53}} \\ 0.1 &\fallingdotseq \frac{3,602,879,701,896,397}{36,028,797,018,963,968}=\frac{3,602,879,701,896,397}{2^{55}} \end{aligned} \tag{26}$$

0.4 は「分母の 4 倍に 2 を加えると分子の 10 倍に一致」し，0.1 は「分子の 10 倍から 2 を引くと分母に一致」する．**15 または 16 桁の数に対して 2 だけずれている**．倍精度浮動小数点数の有意桁は 53 ビットあり，丸め誤差は最後の桁に入るので， $2^{53}\fallingdotseq10^{16}$ より，丸め誤差は分母と分子の 16 桁目あたりに現れる．

図 3 に有意桁 3 ビット，指数部 2 ビットで表される 2 進浮動小数点数と，分母・分子をそれぞれ 3 ビットで表される有理数の数直線上での位置をプロットした．3 ビットで表される数は 0 から 7 までの 8 通りである．浮動小数点数は，スケールファクターの役割を担う指数部が $2^0=1$ のとき 0 から 7，2^{-1} のとき 0.5 刻みで 0 から 3.5，2^{-2} のとき 0.25 刻みで 0 から 1.75，2^{-3} のとき 0.125 刻みで 0 から 0.875 までを表す．指数部が 4 通りで有意桁が 8 通りの積で 32 通りだが，同じ数に複数の表現があるところがあり，これよりも少ない 20 点しか表せない（正規化）．

有理数は分母も分子も 8 通りで，その組合せは 64 通りだが，$\frac{1}{2}=\frac{2}{4}$ のように重複があるので，36 点しか表せない．既約にすれば $\frac{2}{4}$ は存在しない．分母・分子を GCD で割る操作は，冗長性を取除くという意味で，浮

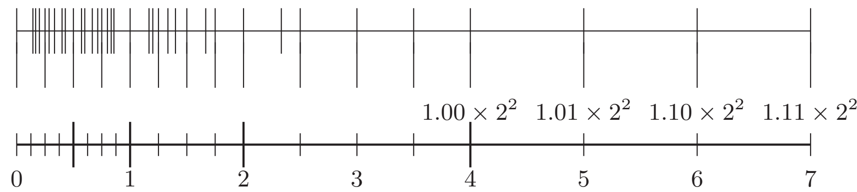


図 3: 有意桁 3 ビットと指数部 2 ビットの 2 進浮動小数点数と（下），分母・分子を 3 ビットで保持する有理数 (上）

動小数点演算の正規化（normalization）に対応する．分母・分子を 4 ビットにすると 256 通りから 144 点，5 ビットにすると 1024 通りから 616 点，6 ビットにすると 4096 通りから 2456 点，7 ビットにすると 16384 通りから 9916 点，8 ビットにすると 65536 通りから 39640 点が表せる． これらの点の数直線上での位置は，浮動小数点数のような規則性はなく，不規則に並ぶ．3 ビットの場合の 36 点を図 3 の上に示したが，零の隣から $\frac{1}{7}$，$\frac{1}{6}$，$\frac{1}{5}$，$\frac{1}{4}$，$\frac{2}{7}$，$\frac{1}{3}$，$\frac{2}{5}$，$\frac{3}{7}$，$\cdots$，3，$\frac{7}{2}$，4，5，6，7 と並ぶ[30]．浮動小数点数の点の位置と一致するものもあるが，一致しないものもある．図には一致する点を長く示した．

プログラミング言語 Fortran や C が，単精度とか倍精度で提供している浮動小数点数は，変数型を指定すれば数値を表すデータ形式は固定される．有理算術演算でも，桁数の上限を 100 桁程度にすれば，100 桁の分母と分子を割り当てることで有理数を表現でき，有理算術演算の実装もそれほど難しくないかもしれない．しかしこの程度の桁数では有理算術演算の役に立ちそうにない．「有理数計算プログラミング環境」では，**有理数型の精度は変数ごとに自動拡張**される．精度に関しては，浮動小数点数をレディメードに譬えるなら，有理算術演算は変数ごとにオーダーメードである．「有理数計算プログラミング環境」の実装は 6 ページの 1.3 節で紹介したが，有理数（rational 型の変数）は，式 (4) の多桁数（longint 型の変数）に符号を付加することで表わし，四則演算を式 (1),(2),(3) で行っている．配列長は 64 から始めて，不足すると動的に倍，倍と拡張する可変長とした．最大長は 32,768 としている（10 進数で約 31 万桁）[31]．図 3 は 3 ビットの場合を示したが，$32 \times 32768 = 2^{20} = 1,048,576$ ビットまで自動拡張される．通常のプログラミング言語が扱える符号なし整数は 64 ビットだが，longint 型は可変長の配列によって，約 100 万ビットまで使え，これが精度を支える．

分母 2 べきの有理数の和，差，積の分母は，式 (1),(2) から，分母の積なので 2 べきである．したがって，加算，減算，乗算だけで構成される計算式では**分母 2 べきは保存**される．除算が入ると，式 (3) より商の分母が ad となり，分母 2 べきが保存されない．分母 2 べきの場合，分子の最下位桁から続く零ビットの数を数えて，そのビット数だけ分母・分子を右シフトすれば既約になる（GCD をユークリッドの互除法で求めると除算に時間がかかる）．「有理数計算プログラミング環境」では，GCD 計算に 2 進 GCD アルゴリズムを使用しているので，分母または分子の一方が 2 べきの有理数は，2 べきを含まない有理数よりも GCD 計算が速い [1].

倍精度浮動小数点数を有理数変換すると，分母 2 べきであるが，アルゴリズムの選択で，加減算と乗算だけの計算式を選んだり，解が無理数になる場合は，中間解を**分母 2 べきに丸める**ことで高速化できる[32].

区間を縮小反復する場合，両端点は正確である必要はないので，有理数 $x = \frac{x_n}{x_d}$ を，分母 2 べきの有理数に丸める `roundrat(x,m)` 関数も用意した．引数の `m` は丸めの精度を指定する．分母・分子を GCD を求めて既約にするところは，四則演算のたびに行うが，それ以外に桁数を削減するための計算順序の工夫や，無理数解を追い込む場合の精度指定は，ユーザプログラムで行う（精度はオーダーメード，つまりユーザが指定して使う）.

図 3 に示した有理数の**不規則な位置は除算に効果的**である．有理算術演算では除算は式 (3) に示したように，分母と分子を逆にして掛けるので，商は正確である．ごく小規模で整数係数，かつ解も整数の連立 1 次方程式の問題を消去法で解いても，（前章で述べたように）分母に 3 が現れるような（循環小数が出てくる）場合は，浮

[30] 0, 1/7, 1/6, 1/5, 1/4, 2/7, 1/3, 2/5, 3/7, 1/2, 4/7, 3/5, 2/3, 5/7, 3/4, 4/5, 5/6, 6/7, 1, 7/6, 6/5, 5/4, 4/3, 7/5, 3/2, 5/3, 7/4, 2, 7/3, 5/2, 3, 7/2, 4, 5, 6, 7

[31] $\log_{10} 4294967296 = 9.633\cdots$ である．計算可能な最大の数は，乗算で FFT を使用しない場合は，再コンパイルで変更可能である．

[32] 例えば，無理数解を挟む区間の下限の分母がメルセンヌ素数 $2^p - 1$ のとき，これを 1 だけ大きくして 2^p にすると，速くなることが想像されよう．

動小数点演算では正確な解が得られない．多倍精度算術演算で，変数ごとに不足すると 8 倍精度，16 倍精度と精度を高める方式の処理系を作っても，循環小数は丸められて誤差を含むので，有理算術演算に敵わない．

53 ビットの精度をもつ倍精度浮動小数点数を有理数変換すると，53 ビットの有意桁は分子に，指数部のスケールファクタは 2 べきで分母に現れる．数直線上の点列では，分母 2 べきの数が浮動小数点数と位置が一致する．熱伝導係数を 10 進で 0.1 にした式 (24) の係数行列 A' は，2 進では 0.4 と 0.1 に式 (26) の有理数を代入して得られるが，丸め誤差のために A' とはすこし異なった行列 A'' になる．

$$A'' = 2^{-55} \begin{pmatrix} 4k & -k & -k & 0 \\ -k & 4k & 0 & -k \\ -k & 0 & 4k & -k \\ 0 & -k & -k & 4k \end{pmatrix}, \qquad k = 3,602,879,701,896,397 \tag{27}$$

この行列を 2^{55} 倍すると，整数行列 A''' が得られる．

$$A''' = kA \tag{28}$$

A''' の固有値は $A'''\boldsymbol{x} = (kA)\boldsymbol{x} = k(\lambda\boldsymbol{x})$ なので A の固有値の k 倍である．したがって行列 A''' の特性多項式を因数分解すると，因子多項式の次数は，もとの行列 A の特性多項式の因数分解後の因子多項式の次数と同じで，係数が式 (23) に従って大きくなる．このため，近似固有値はそれに合わせて高精度化しないと整数性を判定できない．

プログラム dvsrch1.cpp では，計算結果のチェックを数式処理システムを使わずにできる問題として，熱伝導率を倍精度浮動小数点数で 0.1 にして作成した行列を有理数変換した行列を使用する．なお，プログラム dvsrch2.cpp で解析する数値シミュレーションプログラムの倍精度の行列を整数化した行列の全要素の GCD は 1 で，式 (28) のような人為的に桁数を増やした行列とは異なる．

2.3.3　包含（inclusion）と縮小反復

浮動小数点数は正確に有理数に変換して計算でき，対称行列の特性多項式も正確に求められる．しかしその解は一般的には無理数なので，`rational` 型だけでは追究が難しい．

有理算術演算を用いると，対象の数値が有理数の範囲では，素朴なプログラミングによって正確に行うことができるが，**無理数は区間算術演算**（interval arithmetic）の考え方を取り入れる．区間算術演算では，存在（existence），唯一性（uniqueness），包含（inclusion）の 3 つを満たすことが重要である[33]．特性多項式が重根を 2 個と数えれば n 個の実根をもつこと（存在）は，係数行列が対称であることから（計算を正確に行えるので）保証される．残る 2 つは次のようにする．

まず，**解を 1 つだけ挟む区間** (a, b) を見つける．固有値 λ_1 から λ_n を昇順にとると，

$$\cdots < \lambda_{i-1} < a < \lambda_i < b < \lambda_{i+1} < \cdots \tag{29}$$

を満たせば，λ_i は区間 (a, b) に存在し，そしてこの区間には他の根は存在しない．この状態を「包含」と呼ぶ．包含は，区間 (a, b) 内に $p(\lambda) = 0$ の根が 1 つだけ存在する状態を指す．包含が成立したら，この区間 (a, b) を縮小するアルゴリズムを用いて区間幅を，精度要求を満たすまで「縮小反復」する．因子探しの前半は，n 次多項式に対して n 個の区間を見つけることで包含を達成する．λ_i の初期値は浮動小数点演算で得られた近似固有値である．

後半で，**近似固有値の大きさと探索対象の因子の次数から精度要求を決定**して，固有値の存在する区間を縮小反復してから，候補を選び，整数性を判定し，因子多項式の候補を作り，除算を行う．これによって対称行列の特性多項式の因数分解を得られる．誤差を含む近似固有値を測定データと見れば，当て嵌めによって現象分析をする逆問題を解く流れになる．

[33] ドイツ語で Existenz，Eindeutigkeit，Einschließung の 3 語の頭文字をとって 3E 法と呼ばれた．

2.4 プログラムの概要

次章以降で dvsrch，dvsrch1，dvsrch2 の 3 つのプログラムを解説する[34].「有理数計算プログラミング環境」の開発目的の 1 つは，浮動小数点演算による数値シミュレーションプログラムで丸め誤差に起因して発生した問題を，正確な解と比較することで特定することにある．この目的のために,「プログラミング環境」が備えるべき機能（多倍精度数を含めての変数型やユーティリティ関数）の整備を 3 つのプログラムの開発を通して行った．これらのプログラムは，生成された行列から有理算術演算を用いて特性多項式を得るが，同時に多重度の高い固有値がある場合はその存在を知る．これは浮動小数点演算ではできない．

テストプログラム dvsrch.cpp は，条件のよいデータに対して，正確な計算が最低限必要となる部分（行列変換と多項式の除算）だけを有理算術演算を用い，整数性判定は倍精度浮動小数点演算で行う．次の版である有理数係数の行列の解析 dvsrch1.cpp は，行列要素を浮動小数点数とし，これを有理数変換することで，桁数の多い問題を作る．この状態でも，倍精度で得られた近似固有値の精度を区間算術演算によって必要なところまで高めれば，dvsrch.cpp と同様に因子探索が可能なことを確かめる．dvsrch1.cpp で扱う問題は，病的に近接した固有値がなく，近似固有値の分離が容易な問題を扱うが，その次の版 dvsrch2.cpp は，近似固有値のグループ化で同じグループに属する複数の候補を分離する必要がある．

テストプログラム： 多重度 m の固有値をもつ熱伝導行列を，$m \times m$ の未知数で解析する．有理算術演算で正確な計算を行うのは，対称行列のヘッセンベルグ変換，フロベニウス変換，多項式除算で，**整数性判定は倍精度浮動小数点演算で行う**．これにより，整数行列に対してアルゴリズムが稼働するかどうかを確かめる．固有値は 0.1 から 10 の間に分布するので，整数性は倍精度浮動小数点演算で間に合う．特性多項式は表 4 の下段の d で示したように，低い次数の因子多項式に分解されるので，計算時間も短い．

有理数係数の行列の解析プログラム： 行列の要素を，倍精度浮動小数点数で生成してからこれを有理数変換して，全要素の分母の最小公倍数 LCM を求めて掛けることで整数行列にする．行列要素の桁数が増えるので，特性多項式の係数の桁数も大きくなる．また固有値も LCM 倍されるので大きい．したがって整数性判定は倍精度浮動小数点演算ではできない．**倍精度演算で得られた近似固有値を，正確な特性多項式に対して包含して interval 型の数に格納し，必要な精度まで精度改良する**（「interval 型の数」は「有理数計算プログラミング環境」で定義した，longint 型，rational 型に続く第 3 の変数型で，区間の上限と下限を有理数で保持する）．改良された近似固有値を用いて根と係数の関係公式を使用して，数 100 桁に及ぶ多項式係数を作成する．特性多項式は表 4 のように，低い次数の因子多項式に分解されるので，計算時間は短い．零固有値をもつ正方格子に対するグラフ・ラプラシアン行列も解析する．零固有値は特性多項式の定数項がゼロとして現れるので，簡単に見つけられる．因子多項式は高次になるので，組合わせの数が膨大になり，計算時間は長くなるため，有理算術演算に特有のチューニングを施す．また「有理数計算プログラミング環境」が，有理数引数に対しても再帰呼出し可能なことも確認する．

数値シミュレーションプログラムの行列を有理数係数の行列に変換して解析するプログラム： 数値シミュレーションで使用される物理的な意味が明確な，構造解析プログラム CT2D の行列の最も小さなものをファイル経由で読込み，有理数行列に変換する．構造解析の理論的には 3 つの零固有値をもつ問題であるが，浮動小数点演算で得られた行列は，厳密な零固有値は持たない．零となるはずの固有値は零近傍の小さな値をもち，構造物の形状対称性に由来する重根は重根として存在する（重根か近似固有値かは，倍精度浮動小数点演算では分からない）．係数行列の桁数は dvsrch1.cpp の行列の桁数と同程度であるが，ヘッセンベルグ行列の桁数は増え，ヘッセンベルグ変換とフロベニウス変換に計算時間がかかる．構造解析プログラム CT2D を 32 ビットのアーキテクチャである x87 FPU 命令セットを使用する環境でコンパイルするとき，最適化オプションを “-O3” にすると，多重度の分布が変わり，重根が消滅する．この現象の解析を行う．

[34]因子（divisor）を探す（search）の意で dvsrch とした．

dvsrch1.cpp では，近似固有値を 4 倍精度，あるいは 8 倍精度で得られれば，dvsrch.cpp と同様（倍精度を 4 あるいは 8 倍精度にするだけ）のプログラムで，因子探索ができるが，dvsrch2.cpp の扱う零近傍の固有値は，多倍精度演算では精度が不足して，因子探索には使えない．このため,「有理数計算プログラミング環境」としては,多倍精度数は実装せず，有理算術演算で無理数解を追究する方法が良いとの結論に達した．平方根や初等関数を用意し，精度を指定して近似値を得る方向で，今後の開発を進めることを考えている．

次章でテストプログラム dvsrch.cpp の作成を解説する．4 章で有理数係数の行列を分析するプログラム dvsrch1.cpp を解説する．5 章で倍精度浮動小数点行列を分析するプログラム dvsrch2.cpp を解説する．

3　テストプログラム dvsrch.cpp

有理算術演算（rational arithmetic）は，有理数を対象に正確な計算を行うが，対象の数値が無理数になると，要求される精度で妥協して誤差を含む近似解を求めるか，解の存在する区間を求めるのが普通の流れである．ここで行う「特性多項式の因子探し」はこの流れとは反対に，誤差を含む近似解を組合わせて，正確な因子多項式の因子を探すことで，因数分解を得る．近似解を測定されたデータ，根と係数の関係公式を理論式と見なせば，測定データに合致する理論式を見つけるデータ解析，あるいは**逆問題**の形になっている．プログラム dvsrch.cpp では素朴に倍精度浮動小数点演算で得られた近似解から候補の組合せを選んで，根と係数の関係公式によって多項式係数を作る．その係数の整数性判定を行い，小数点以下の小さな値を切り捨てるか切り上げるかして整数の係数を抽出し，整数係数の多項式で特性多項式を割ってみる．抽出された係数と特性多項式は有理数型で扱い，除算を有理算術演算で正確に行うので，ここに計算誤差は入らない．割り切れれば因子であるから，特性多項式を商多項式で置き換えて，特性多項式の次数を減らしてゆく．プログラム dvsrch.cpp で，**有理算術演算で正確な計算を行うのは，対称行列のヘッセンベルグ変換，フロベニウス変換，多項式除算で，整数性判定は倍精度浮動小数点演算で行う**ことでプログラミングを簡単にし，整数行列に対してアルゴリズムが稼働することを確かめる．

3.1　処理の概要

分割数 m を引数としてとる 230 行ほどの関数 eigvals が処理の流れを制御する．処理の概要を述べる．

- 係数行列 A を倍精度浮動小数点演算で 2 次元配列 `matrix<double> a(n,n)` に生成する（SetFtherm）[35]．有理数 2 次元配列 `rational_matrix ar(n+1,n+1)` に `cnvmat` 関数によって**有理数変換し**，ヘッセンベルグ変換 $A \to H$ する（elmhes）[36]．
- 近似固有値を倍精度計算で求め（jacobev）ソートする（bsort）[37]．近似固有値は 1 次元配列 `est` に格納され，グループ化し（selectev），各グループの下限 `eigvl` と上限 `eigvh`，そのグループに属する近似固有値の多重度 `mult` を抽出する．近似固有値に関する数値は倍精度浮動小数点数の配列に置かれる．グループ化は，ファイル全体をスコープ（有効範囲）とする変数 `eps` によって $\varepsilon = 10^{-9}$ で行う[38]．
- ヘッセンベルグ行列の副対角項 k_i のゼロ要素を探してヘッセンベルグ小行列の数を得る（nmhes）．

5 分割の場合は，H_{13}，H_9，H_1，H_1，H_1 の 5 つの小行列が得られるので，以下，各小行列に対して逆順に H_1，H_1，H_1，H_9，H_{13} と進める．

- 小行列 H_k をフロベニウス変換して特性多項式 $p(\lambda)$ を生成する（hesfrb）[39]．次数は変数 `nsize` に格納する．
- 既知の因子多項式がある場合はこれを `getpolynomial` 関数で取り出し，特性多項式 $p(\lambda)$ を割り（polydivchk），割り切れれば商多項式で $p(\lambda)$ を置き換え，次数 `nsize` を小さくする[40]．
- 近似固有値の $p(\lambda)$ に対する包含を調べる（chkinc）．$p(\lambda)$ が 1 次式なら，$p(a) \cdot p(b) < 0$ が条件になる．n 次多項式の場合，$p(a) \cdot p(b) < 0$ で，かつこのような区間が n 個見つかればよい．これらのケースでは，

---

[35] プログラミング言語 C は，他の言語で「配列の配列」と呼んでいるものを，2 次元配列と呼ぶ場合があるが，Fortran や Pascal のような 2 次元配列は扱えない.「有理数計算プログラミング環境」では 2 次元配列として使用するために `matrix<double>` 型を用意した．
[36] cnvmat 関数と elmhes 関数は ratutil クラスにある．
[37] jacobev のアルゴリズムであるヤコビ法の説明を付録に記す．
[38] 18 ページの表 3 の 5 分割の例の場合は，$n = 25$ の問題で表のように 13 グループに分かれる．
[39] hesfrb 関数は ratutil クラスにある．
[40] 表 3 の 5 分割の例で H_9 を扱う場合には，特性多項式は 9 次であるが，既知の $(4 - \lambda)$ で割り切れるため 8 次式 $(\lambda^8 - 32\lambda^7 + 436\lambda^6 - 3296\lambda^5 + 15079\lambda^4 - 42608\lambda^3 + 72312\lambda^2 - 67008\lambda + 25740)$ になる．

それぞれの区間に 1 つづつ根が存在するので，各区間で「包含（インクルージョン）が成立している」という．dvsrch.cpp では，**扱う行列の特質から，包含チェックを通過しなければ計算を停止する**[41]．

- 近似固有値 e_i の整数性を askinty 関数で調べ，整数らしければ e_i を有理数型の整数 e として抽出し，1 次式 $(-\lambda+e)$ で $p(\lambda)$ を割り，割り切れれば putpolynomial 関数によって因子として保存し，商多項式で $p(\lambda)$ を置き換え，nsize をデクリメントする[42]．

- 次数 $d=2,3,\ldots$ について以下の **2 次以上の因子探し**を行う．この処理は，次数 d が多項式の次数 nsize まで，近似固有値の組合せを作って因子を探すので，while(d<= nsize) ループで書かれている．

  はじめに，組合せループの前処理として，包含が成立している近似固有値だけを setecand 関数で evcan に詰めてセットし，各近似固有値が属するグループを配列 usev にセットする[43]．この時点で因子多項式の候補となる近似固有値の数が ncandi 個と決まり，作業用配列 idx を初期化し，組合せループの反復回数を得る（nn=comb(ncandi,d)）[44]．

  この後，その内側の for ループによる組合せループを最大 nn 回繰り返す．因子が見つかると，多項式の次数が小さくなるが，その次数の因子探しを継続する場合と，d をインクリメントする場合がある（この条件を reset フラグにセットする）．

  – nxtcmb 関数によって「組合せ」を作業用配列 idx に得る[45]．この配列に従って候補固有値を eigvl から figvl に詰めて格納する．根と係数の関係公式の 1 次項 $\sum_{j}^{d} e_j$ を求める．

  – 1 次項の整数性だけを先行して調べ（先行判定），整数と見なせれば，vietaterm 関数によって他の係数を cof[1] から cof[d] に求め，それらの整数性を調べる[46]．

  – 全係数が整数と見なせれば，係数を**有理数型の整数として抽出し**，それらによって整多項式 $v(\lambda)$ を作り，これで $p(\lambda)$ を割る．$p(\lambda)=v(\lambda)q(\lambda)$ なら割り切れて，$v(\lambda)$ を因子多項式とし保存し，商多項式 $q(\lambda)$ で $p(\lambda)$ を置き換え，nsize を減らす[47]．特性多項式も取り出された因子多項式の候補も有理数型なので，桁数が多くなっても除算は正確に行える．

    見つかった近似固有値を候補から除外して，その次数 d の因子探しを継続するか終えるかを配列 idx の状態から，d 個の組合わせの因子探しが完了したか途中かを判定し，継続する場合は reset フラグを立ててループを break する[48]．

倍精度浮動小数点演算で得られた近似固有値のグループ化を行う関数を示す．

[41] 5 分割の例で H_9 を扱う場合には，既知の因子で割られた 8 次多項式に対して，グループ 2,4,5,6,8,9,10,12 の 8 つで包含が成立し，これらのグループの近似固有値を組み合わせて因子多項式を探す．

[42] 5 分割で H_9 を扱う場合には，グループ 5 の $e_5=2.999\cdots$ と，グループ 9 の $e_9=5$ で割り切れて，特性多項式は 6 次式 $(\lambda^6-24\lambda^5+229\lambda^4-1104\lambda^3+2812\lambda^2-3552\lambda+1716)$ になる．

[43] usev は組合わせの番号からグループ番号へ間接参照するための配列で， usev[idx[j]] が j 番目のグループを指す．

[44] 初期化は，idx[1] から idx[d] に 1 から d を格納し，最後の idx[d] からは 1 を引く．5 分割で H_9 を扱う場合には，6 つの候補があり nn $={}_6C_2=15$ である．

[45] 組合せは d=2 なら 12,13,$\cdots$, ncandi,23,24,$\cdots$ のように idx[1] と idx[2] に入る．usev は組合わせの番号からグループ番号へのポインタで， usev[idx[j]] が j 番目のグループを指す．

[46] 全係数を作成してから整数性のチェックを行うと，プログラムは単純になるが計算時間がかかる．そこで総和で計算できる 1 次項だけを先に判定し，これを通過した場合だけ，他の係数を求める．この先行判定は有理算術演算では大きなチューニング効果を与える．なお第 d 次係数を総積で求めるより，1 次の係数を求めるほうが，桁数が少ないので速い場合が多い．

[47] 5 分割で H_9 を扱う場合には，6 つの候補から idx[1]=1, idx[2]=4 のとき，usev[1]=2, usev[4]=8 で，$e_2=1.2679\cdots$ と $e_8=4.7320\cdots$ が選択されると，係数 -6 と 6 が作られ，$(\lambda^6-24\lambda^5+229\lambda^4-1104\lambda^3+2812\lambda^2-3552\lambda+1716)\div(\lambda^2-6\lambda+6)=\lambda^4-18*\lambda^3+115*\lambda^2-306*\lambda^1+286$ と割り切れる．

[48] 5 分割で H_9 を扱う場合には，2 次式 $(\lambda^2-6\lambda+6)$ で割り切れると，6 つの候補が 4 つに減り，配列 evcan から 2 つを取り除いて，グループ 4,6,10,12 に関する因子探しを継続する．

selectev 関数

```
void selectev(const vector<double>& a, int n, int& ngrp, vector<double>& eigvl,
              vector<double>& eigvh, vector<int>& mult) {
   int i,ii,j=0;
   double x,am;
   am=fabs((a[n-1]-a[0])/((double)n));  /* abs(mean) */
   std::cout << "selectev Amean=" << am  << std::endl;
   eigvl[j+1]=a[0]; eigvh[j+1]=0; x=a[0]/am; ii=0; mult[j+1]=0;
   for(i=1; i < n; i++){
      if(fabs(a[i]/am-x) > eps){ j++; x = a[i]/am; eigvl[j+1]=a[i]; ii=i; mult[j+1]=0; }
      if(ii != i){ eigvh[j+1]=a[i]; mult[j+1]++; }
   }
   ngrp = j;
}
```

複数ある固有値の真ん中のものを代表として選び，`eps` を相対誤差判定で使用し，4 分割の表 2 では 16 個の近似固有値が 9，5 分割の表 3 では 25 個の近似固有値が 13 のグループに分類される．この関数は後述の章でも使用する．

2 次以上の因子を探す部分は，2 重のループ構造（次数 2 から特性多項式の次数の `while` ループと，組合わせのループ）の中に 3 つの `if` ブロックがあり，プログラムは複雑である．この理由は，候補数 `ncandi` が減ってゆき，これがループ構成で組合せループの終端を決めるからである．候補が減っても，常に候補の組合せをはじめから調べれば，プログラムは単純になるが，すでに**チェック済みの組合せを再びチェックする**ことになる[49]．これを避けるために `reset` フラグを用いてループを `break` する．

3.2 根と係数の関係公式と組合せのプログラミング

誤差を含む近似固有値から，誤差を除外した整数の多項式係数を抽出するところが鍵である．本節では，根と係数の関係公式と組合せのアルゴリズムとそのプログラミングを説明する．

3.2.1 根と係数の関係公式

高校の数学では「根と係数の関係」でお馴染みの基本対称式（elementary symmetric function）あるいは公式（Vieta's formula）を，次数 $n = 5$ の多項式 $p_0x^5 + p_1x^4 + p_2x^3 + p_3x^2 + p_4x + p_5$ を例に説明する．根を α_1 から α_5 とすると次のようになる（最高次の係数は 1 か -1 である．最高次の係数 $p_0 = 1$ の多項式をモニック多項式という）．

$$
\begin{aligned}
p_0 &= -1 \\
p_1 &= \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 + \alpha_5 \\
p_2 &= -(\alpha_1\alpha_2 + \alpha_1\alpha_3 + \alpha_1\alpha_4 + \alpha_1\alpha_5 + \alpha_2\alpha_3 + \alpha_2\alpha_4 + \alpha_2\alpha_5 + \alpha_3\alpha_4 + \alpha_3\alpha_5 + \alpha_4\alpha_5) \\
p_3 &= \alpha_1\alpha_2\alpha_3 + \alpha_1\alpha_2\alpha_4 + \alpha_1\alpha_2\alpha_5 + \alpha_1\alpha_3\alpha_4 + \alpha_1\alpha_3\alpha_5 + \alpha_1\alpha_4\alpha_5 + \alpha_2\alpha_3\alpha_4 + \alpha_2\alpha_3\alpha_5 + \alpha_2\alpha_4\alpha_5 + \alpha_3\alpha_4\alpha_5 \\
p_4 &= -(\underbrace{\alpha_1\alpha_2\alpha_3\alpha_4}_{\text{lack } \alpha_5} + \underbrace{\alpha_1\alpha_2\alpha_3\alpha_5}_{\text{lack } \alpha_4} + \underbrace{\alpha_1\alpha_2\alpha_4\alpha_5}_{\text{lack } \alpha_3} + \underbrace{\alpha_1\alpha_3\alpha_4\alpha_5}_{\text{lack } \alpha_2} + \underbrace{\alpha_2\alpha_3\alpha_4\alpha_5}_{\text{lack } \alpha_1}) \\
p_5 &= \alpha_1\alpha_2\alpha_3\alpha_4\alpha_5
\end{aligned}
$$

符号は，次数が奇数なら負，偶数なら正である．

[49]候補が 5 つあって，組合わせ (2,3) で因子が見つかると，候補は 3 つに減る．このとき 1 番目の候補に対するチェックは終わっているので，新しい組合わせ (2,3) から因子探しを始める．

プログラミングを行うにあたり，添字の並びのルールを調べる．各係数を形成する項 α_k の添字の並び方に着目する．定数項は総積 $p_n = p_5 = \prod_{k=1}^{n} \alpha_k$ である．p_1 と p_4 は互いに補（complement）の関係がある．つまり，p_1 は $k = 1, 2, \cdots, n$ の総和 $\sum_{k=1}^{n} \alpha_k$ で，総和の第 1 項は α_1 である．p_4 は 4 つの α の積の総和だが，その第 1 項は 1 から n までの α の項から α_1 が抜ける．p_1 の総和の第 2 項は α_2 で，p_4 の第 2 項は 1 から n までの α の項から α_2 が抜ける．$\cdots$．どちらも項数は n である．

p_2 と p_3 にも互いに補の関係があり，p_3 の項を後ろから逆に読むと，抜けた添字が p_2 の項を前から読む添字に一致する．$p_2 = \sum_{i1,i2}^{10} \alpha_{i1}\alpha_{i2}$ であり，$p_3 = \sum_{i1,i2,i3}^{10} \alpha_{i1}\alpha_{i2}\alpha_{i3}$ である．総和記号の上と下につく式を一般的に書くと次のようになる．

$$\sum_{i1,i2,\cdots,ir}^{{}_nC_r} \alpha_{i1}\alpha_{i2}\cdots\alpha_{ir} \tag{30}$$

3.2.2 組合せのプログラミング（十進 BASIC による準備）

もっとも基本的な「1 から 100 まで数える」ループプログラムを，C++ と 十進 BASIC で示す[50]．BASIC は 1960 年代に生まれた．当時の文字は 6 ビットコードだったので，プログラミングに小文字は使えなかった．7 ビットコードの ASCII コードが一般化した後に生まれた C や C++ では大文字と小文字は別の文字として区別するが，Fortran や BASIC では区別しない．十進 BASIC ではカーソルを移動すると，制御文字は大文字に自動的に変更され，ユーザの書いた変数名はそのままになる（字下げも自動的にブランクを挿入してくれる）．本稿では変数名は主に小文字を使用する．

C や Fortran の数値は，整数型か浮動小数点数型かを指定するが，BASIC は数値を電卓のように一通りに扱う（型を指定する必要はない）．C や C++ では `for` と初期化式 `int i=1;` を区切る「セパレータ」に括弧を使用するが，BASIC はセパレータにブランクを用いるので，`FOR` の次にブランクが必要である．1 から「1 つづつ増やして」を指定する部分は「再初期化式」というが，BASIC は `STEP 1` と指定する．増分が 1 の場合は省略可能である．C や C++ ではステートメントの終わりはセミコロンで示すが，60 年代のパンチカードにプログラムを記述した時代に生まれた言語ではセミコロンは使わない（改行コードでステートメントが終わる）．また，十進 BASIC は，セミコロンで区切って 1 行に複数のステートメントを書くことはできない．

正規化ループ

```
int n=100;
for(int i=1; i <= n; i++){
    .....; .....;
}
```

```
n=100
FOR i=1 TO n STEP 1
  .....
  .....
NEXI i
```

C や C++ では変数 `i` や `n` の変数型を `int` にすると 32 ビット整数型の最大値，2,147,483,647= 2G − 1 まで数えられ，それより大きな数まで数えたいときは変数型を，例えば `long long` 型などに変更する．十進 BASIC では p/q のアイコンをクリックして「有理数モード」にすれば，1000 桁の数まで数えられる．

順列，組合せ，重複順列，重複組合せの 4 つは，数え上げプログラミングのループ構成の基本である[51]．5 個のものから 2 個を選んで作る組合せは $n = 5$ として 2 重のループ構成で，5 個のものから 3 個を選んで作る組合せは 3 重のループ構成で作ることができる．

[50]最も基本的なループで「正規化ループ」（normalized loop）と呼ぶ．

[51]Combination.BAS の左の例で，`FOR i2=i1+1 TO n` を `FOR i2=i1 TO n` に変更すれば，重複組合せ ${}_5H_2$ になる．すべてのループを 1 からにすれば，重複順列 n^2 になり，内側のループに `IF i2<>i1 then` の条件を加えれば順列 ${}_5P_2$ になる．

Combination.BAS

```
LET n=5
FOR i1=1 TO n
   FOR i2=i1+1 TO n
      PRINT i1;i2
   NEXT i2
NEXT i1
```

```
LET n=5
FOR i1=1 TO n
   FOR i2=i1+1 TO n
      FOR i3=i2+1 TO n
         PRINT i1;i2;i3
      NEXT i3
   NEXT i2
NEXT i1
```

左のプログラムは次の出力を与える．

i1	1	1	1	1	2	2	2	3	3	4
i2	2	3	4	5	3	4	5	4	5	5

i1,i2 を 2 桁の数値として読むと，昇順になっている．右のプログラムは次の出力を与える．

i1	1	1	1	1	1	1	2	2	2	3
i2	2	2	2	3	3	4	3	3	4	4
i3	3	4	5	4	5	5	4	5	5	5

根と係数の関係公式では，与えられた n に対して，$r = 1$ から $r = n$ までを変数にして動かす必要がある．プログラムがコード生成して，生成されたコードに制御を渡してそのコードを実行する方式を**コード生成**（code gen）方式という[52]．この方式を使用すれば上記のループ構成でもよいが，コード生成方式を採用するまでもなく，ここでは Combination.BAS の r 重に**ネストしたループ構成を 1 重にループ変換**すればよい[53]．そこでまず，呼び出されるたびに表の i1,i2 や i1,i2,i3 の数列を引数の配列に返すサブルーチン nxtcmb （「次の組合せ」の意）を用意する．$n = 5$ で $r = 2$ なら，ith(1)=1，ith(2)=1 に初期化して呼ぶと配列 ith には 1,2 が返る．次は 1,3 が返る．$\cdots$．

CombSet.BAS の nxtcmb サブルーチン

```
EXTERNAL SUB nxtcmb (n,r,ith())
IF ith(r) = n THEN                    ! 最後の桁が n だったら
   FOR i=r-1 TO 1 STEP -1             ! 前の第 i 桁を探す．
      IF ith(i) <= n-(r-i)-1 THEN     ! 第 i 桁が大きくできれば
         LET ith(i)=ith(i)+1          ! 第 i 桁をインクリメントして
         FOR j=i+1 TO r               ! それ以降の桁を
            LET ith(j)=ith(i)+j-i     ! 第 i 桁に従ってセットする．
         NEXT j
         EXIT FOR
      END IF
   NEXT i
ELSE                                  ! 最後の桁が n でなかったら
   LET ith(r)=ith(r)+1                ! 最後の桁をインクリメントする．
END IF
END SUB
```

十進 BASIC には組合せの総数を与える comb 関数が用意されている．これを使って n と r を変数にして，組合せの添字を順番に得ることができる．プログラム CombSet.BAS は，n を与えると，${}_nC_1$，${}_nC_2$，${}_nC_3$，${}_nC_4$，${}_nC_5$ それぞれについて組合せを表示するが，各組合わせは次の SetInd サブルーチンで作成して表示している．

[52]現代のプロセッサは，命令とデータを分けて記憶域からフェッチする．コード生成はデータ領域に生成し，これを実行するときは命令としてフェッチすることでコード生成方式は実行される．

[53]「ループ変換技術（loop transformation technique）」は HPC プログラミングの根幹をなすプログラミング技術である．

CombSet.BAS の SetInd サブルーチン

```
EXTERNAL SUB SetInd (n,r)
DIM ith(r)
FOR i=1 TO r                        ! 配列の初期化
   LET ith(i)=i                     ! 配列の初期化
NEXT i                              ! 配列の初期化
LET ith(r)=ith(r)-1                 ! 配列の初期化
PRINT "comb(n,r)=";comb(n,r)
FOR i=1 TO comb(n,r)                ! _n C_r 回反復
   CALL nxtcmb(n,r,ith)             ! 次の組合せを ith[1] から [r] に得る
   FOR j=1 TO r
      PRINT ith(j);
   NEXT j
   PRINT
NEXT i
END SUB
```

CombSet.BAS によって 1 から n までの組合せループの 1 重化の方法を確認できる．

3.2.3 根と係数の公式と組合せのプログラミング（C++）

十進 BASIC の `comb` 関数を作成する．戻り値は 64 ビット整数（`long long` 型）とした．この変数型で扱える最大数は $2^{63}-1$ で，$n>61$ では桁溢れするので停止するようにした．

$$
{}_nC_r = \frac{n(n-1)\cdots(n-r+1)}{r(r-1)\cdots 2} \tag{31}
$$

を計算するので，r が 2 なら分子のどちらかは偶数である．r が 3 なら分子の内の 1 つは 3 の倍数である．したがって分子の 1 項と分母の 1 項を交互に計算すれば，中間桁溢れは起こらない[54]．

ratutil.cpp の comb 関数（long long 型）

```
long long comb(const int n, const int r){
    long long t=(long long)n;
    for(int i=1; i<r-1;i++){
        t=t*((long long)(n-i));
        t=t/((long long)(i+1));
    }
    return t;
}
```

ただし，64 ビット整数の除算は時間がかかるので，式 (31) の分子と分母を longint 型の変数で独立して乗算のみで計算してから，除算を 1 回としたほうが速い．ここでは longint 型の紹介を兼ねて，型変換の方法と longint クラスの除算関数 `ldivide` の使用例を示す．

```
int i;
longint t("1"), s("1"),x,y;  // how to set a number into longint variable
luint32_t ni;
long long result;
for(i=0; i<r;i++){
    ni=n-i;
    x=ni;      //  cast int to longint
    t=t*x;
}
for(i=1; i<=r;i++){
    ni=i;
```

[54]最終結果が桁溢れしなくても，それを算出する過程で桁溢れすることを「中間桁溢れ」という．

```
            if(ni != 0){ x=ni; s=s*x; }
        }
        x=longint::ldivide(t,s,&y);
        result=x;
        return result;
    }
```

nxtcmb 関数を示す.

ratutil.cpp の nxtcmb 関数

```
void nxtcmb(const int n, const int r, int ith[]){
    int i,j;
    if(ith[r]==n){
        for(i=r-1; i>0;i--){    // search position to add
            if(ith[i]<=n-(r-i)-1){
                ith[i]++;
                for(j=i+1; j<=r;j++) ith[j]=ith[i]+j-i;
                break;
            }
        }
    }else{ ith[r]++; }
}
```

倍精度配列 e に近似固有値が n 個格納されており，これから r 個の組合せを ${}_nC_r$ 選び出し，それらの積の和 p_r を倍精度浮動小数点数で返す関数 vietaterm を示す.

ratutil.cpp の vietaterm 関数

```
double vietaterm(const double e[], int n, int r){ // <== rational_vector
   int ith[r+1];
   for(int i=1; i<=r;i++) ith[i]=i;
   ith[r]=ith[r]-1;
   double t=0;                                    // <== rational
   long long nn=comb(n,r);
   for(long long i=1; i<=nn; i++){
      nxtcmb(n,r,ith);
      double s=1;                                 // <== rational
      for(int j=1; j<=r; j++) s=s*e[ith[j]];
      t=t+s;
   }
   if(r%2==1) t=-t;     ! 次数が奇数は符号反転
   return t;
}
```

同名の vietaterm で，コメントでマークした 2 行の変数型を rational とした rational 型で計算する関数も多重定義した（後述）.

“for(初期化式; 継続条件式; 再初期化式)” 文は，セミコロンで区切られた 3 つの式は独立に評価される．したがって，初期化式を long long i を用いているので，反復回数は 32 ビットを超えて $2^{63}-1$ まで実行できる．さらに多くの反復を実行する場合は，変数を rational とする.

3.2.4 整数性判定と有理数での整数の抽出

倍精度浮動小数点数 t が，絶対誤差で epsint の範囲内に整数が存在するか否かを判定する．t が絶対値の小さな数なら簡単にできるが，次章の dvsrch1.cpp で扱う大きな数ではこの方法の限界を超える.

rational クラスに次の倍精度数の場合の整数性チェックのための askinty 関数を用意した．第 2 引数の epsint 以内にある整数を有理数として抽出して第 3 引数に返す.

ratutil.cpp の askinty 関数（倍精度数）

```
int askinty(const double t, const double epsint, rational& vrat) {
   double u;
   int rtnval=0;
   if(fabs(floor(t)-t) < epsint){
      u=floor(t); vrat=Rdset(&u); rtnval=1;
   }
   if(fabs(ceil(t)-t) < epsint){
      u=ceil(t); vrat=Rdset(&u); rtnval=1;
   }
   return rtnval;
}
```

同名の関数で引数 `t` を rational 型で計算する関数も多重定義した（後述）.

3.3 多項式の格納と除算

多項式の表記は, $p(x) = a_3x^3 + a_2x^2 + a_1x + a_0$ のように冪と係数の添字を一致させる（降順の）書き方と，逆に $p(x) = a_0x^3 + a_1x^2 + a_2x + a_3$ のように最高次の係数添字をゼロに，定数項の添字を次数に一致させる（昇順の）書き方とがある．高校の数学では昇順を用いるが，多項式の乗算や除算を行う場合には降順のほうが自然にプログラミングできる．12 ページのフロベニウス変換式 (13) でも冪と行番号を一致させる降順が用いられる．本稿では，29 ページの根と係数の関係公式や，22 ページの方程式の解の変換公式 (23) では係数の添字とべきを一致させるために昇順を使用した．一般に両方の使用法が用いられる．

n 次多項式の $n+1$ 個の係数を配列 `a[0]` から `a[n]` に与えてこの多項式係数を `putpolynomial` 関数で格納し，また `getpolynomial` 関数で配列に取り出す．これらの操作では，多項式の項が昇順か降順かを決める必要はなく，使用者が昇順か降順かを理解していればよい．本例題プログラムでは降順で使用している．

3.3.1 多項式の格納形式

項数可変の複数の多項式を格納するのに，2 つの配列を用いて，一方の配列に多項式の先頭の項の置かれた添字を格納し，他方に多項式の項を詰めて格納する．各多項式の項数は，連続する添字配列の差として得られる．係数配列の先頭の添字の差 `ppolynomial[k]-ppolynomial[k-1]-1` が多項式の次数になる．したがって，次の `putpolynomial` 関数で格納でき

ratutil.cpp の putpolynomial 関数

```
int putpolynomial(const int nsize, const rational_vector& cof, int npolynomial,
                  vector<int>& ppolynomial, rational_vector& cpolynomial){
   int i,i1,i2;
   i1=ppolynomial[npolynomial]; i2=i1+nsize+1;
   for(i=i1; i<i2; i++) cpolynomial[i]=cof[i-i1];
   npolynomial++;
   ppolynomial[npolynomial]=i2;
   return npolynomial;
}
```

逆の `getpolynomial` 関数で取り出す．引数 `npolynomial` は値渡しである．何番目の多項式かを与えて取り出し，戻り値は取り出した関数の次数である．

ratutil.cpp の getpolynomial 関数

```
int getpolynomial(const int k, rational_vector& cof, const int npolynomial,
                  const vector<int>& ppolynomial, const rational_vector& cpolynomial){
   int i,i1,i2;
   if(k>npolynomial){
      cout << "getpolynomial error npolynomial=" << npolynomial <<
                               " smaller than the first argument=" << k << endl;
      exit(1);
   }
   i1=ppolynomial[k-1]; i2=ppolynomial[k];
   for(i=i1; i<i2; i++) cof[i-i1]=cpolynomial[i];
   return i2-i1-1;
}
```

使用例を示す.

put/getpolynomial 関数の使用

```
   // ******** get, put polynomials   ********************************
   int npolynomial=0;               // number of polynomials found  *
   vector<int> ppolynomial(n+2);    // pointer to  polynomials      *
   rational_vector cpolynomial(n*2); // coefficients of polynomials  *
   // **************************************************************
   ppolynomial[0]=0;   // 初期化
   int ksz=getpolynomial(iod,vrat,npolynomial,ppolynomial,cpolynomial);
   polytexformr(vrat,ksz);
```

取り出した関数を `polytexformr` で表示するが，表示する関数は昇順の場合は `polytexform` を使用する．`polytexformr` 関数も `polytexform` 関数も rational クラスに含まれる．また，有理数の分母が 1 の場合は “/1” を取り除く `polytexformint` 関数も用意した．特性多項式 (19) の `polytexformint` 関数による表示は次のように得られる．

polytexformint 関数の出力

```
-1*x^3 +12*x^2 -44*x^1 +48
```

これをコピー&ペーストで数式処理システムや LaTeX の原稿に入力できる．

3.3.2 多項式の除算

多項式の除算は次の ratutil クラスの関数で行う（高校の数学で「整式の除法」）．

ratutil.cpp の polydivchk

```
int polydivchk(rational_vector& u, const int m, const rational_vector& v,
                    const int n, rational_vector& q, rational_vector& r){
   for (int k=m-n; k>=0; k--){
      q[k] = u[n+k] / v[n];
      for (int j=n+k-1; j>=k; j--) u[j] = u[j] - q[k] * v[j-k];
   }
   int rtnval=1;                            !  剰余のチェック
   for (int j=0; j<=n-1; j++){
      r[j] = u[j];
      if(r[j] != rational::ZERO) rtnval=0;
   }
   return rtnval;
}
```

割り切れた場合は 1 を返し，剰余が非零の場合は 0 を返す．引数 `u` は変更される（入出力な）ので注意のこと．

3.4 包含チェック

包含は，区間両端での関数値と符号から判定する．関数値はホーナー法で計算するが，導関数の値もホーナー法で調べられる．本節では，十進 BASIC と C++で説明する．

3.4.1 包含チェック

selectev 関数でグループ化された近似固有値に対し，各グループに属する近似固有値の下限値が引数 `e` に，上限値が引数 `h` に与えられたとき，相対誤差 $\varepsilon = 10^{-9}$ を考慮して，特性多項式 $p(\lambda)$ に対して包含を調べる．近似固有値は倍精度，多項式係数は有理数で得られているので，近似固有値を rational 型の変数 `a` と `b` に有理数変換して格納し，有理算術演算によるホーナー法 `Horner` で関数値 $p(a)$ と $p(b)$ を正確に計算し，符号変化を調べることで包含を判定する．

dvsrch.cpp の chkinc 関数

```
int chkinc(const int n, const rational_vector& p, const double e,
           const double h, const int mult){
   double ta,tb;
   int rtnval=0;
   rational a,b,fa,fb;
   if(fabs(e) > eps){
      ta=e-fabs(e)*eps; tb=h+fabs(h)*eps;    // interval inflation
      if(mult==0) tb=e+fabs(e)*eps;          // 区間に 1 固有値しかない場合
   }else{
      ta=e-eps; tb=h+eps;                    // ゼロ近傍の固有値の場合
      if(mult==0) tb=e+eps;
   }
   a = Rdset(&ta); fa=Horner(n,p,a);         // rational 型に変換して p(a)
   b = Rdset(&tb); fb=Horner(n,p,b);         // rational 型に変換して p(b)
   if(fa*fb<rational::ZERO) rtnval=1;        // p(a) と p(b) が異符号なら 1
   return rtnval;
}
```

包含が成立していれば戻り値は 1 で，これを配列 `included` に格納する．dvsrch.cpp では，この時点での多項式次数 `nsize` と，包含の成立数を比較して，一致していた場合のみ計算を続行する．

3.4.2 ホーナー法

n 次の多項式

$$p_n(x) = a_0x^n + a_1x^{n-1} + a_2x^{n-2} + \cdots + a_{n-2}x^2 + a_{n-1}x + a_n \tag{32}$$

を $=0$ と置いた方程式を代数方程式（algebraic equation）と呼ぶ．係数 a_k は数学の教科書では複素数か実数として議論されるが，本稿では計算機で扱える数を扱うので有理数とする．多項式 (32) の $x=\xi$ における値 $p_n(\xi)$ は，ホーナー法によって計算する．

$$p_n(\xi) = (\cdots((a_0\xi + a_1)\xi + a_2)\xi + \cdots + a_{n-1})\xi + a_n \tag{33}$$

多項式の各項を，式 (32) の左から順に $a_0 \times x \times x \times x \cdots$，次に $a_1 \times x \times x \times x \cdots$ のように計算すると，$n(n+1)/2$ 回の乗算が必要になる．ホーナー法ではこれを n 回で行える．十進 BASIC で示す．

```
EXTERNAL FUNCTION Horner(n,a(),x)
OPTION BASE 0                       !  配列要素を 0 から数えるためのオプション
LET r=a(0)
FOR i=1 TO n
```

```
    LET r=r*x+a(i)
  NEXT i
  LET Horner=r
  END FUNCTION
```

$p_n(x)$ を $x-\xi$ で割ったとき，その商と剰余を $Q(x)$ と R で表せば

$$p_n(x) = (x-\xi)Q(x) + R \tag{34}$$

となる．これに $x=\xi$ を代入すれば

$$p_n(\xi) = R \tag{35}$$

が得られる．つまり，$p_n(\xi)$ の値は $p_n(x)$ を $x-\xi$ で割った剰余としても求められる．これは剰余定理（remainder theorem）として知られている．例えば $p_3(x) = x^3 - 15x - 4$ を $x-3$ で割った場合，組立除法（synthetic division）によって商と剰余を求めてみる．

$$\begin{array}{rrrrr|l} & 1 & 0 & -15 & -4 & 3 \\ +) & & 3 & 9 & -18 & \\ \hline & 1 & 3 & -6 & -22 & \end{array}$$

上段には多項式の係数を並べ，右端には $x-3$ の 3 を記す．次に最高次の係数 1 をそのまま下段に下ろし，それに上段右端の 3 を掛けた値 3 を（矢印に従って）中段右隣に記す．上段の値と中段の値の和を下段に記す．この操作を繰り返すと，商多項式の係数と剰余 $p_3(3) = (x^2+3x-6)(x-3) - 22$ が得られる．

組立除法の一般の場合を示す．

$$\begin{array}{cccccccc|l} a_0 & a_1 & a_2 & \cdots & a_{n-2} & a_{n-1} & a_n & & \xi \\ & a_0\xi & b_1\xi & \cdots & b_{n-3}\xi & b_{n-2}\xi & b_{n-1}\xi & & \\ \hline a_0 & b_1 & b_2 & \cdots & b_{n-2} & b_{n-1} & R & & \end{array}$$

となる．ここで，最下段の値 $b_i,\quad i=0,1,\ldots,n-1$ および R は，

$$\begin{aligned} b_1 &= a_0\xi + a_1 \\ b_2 &= b_1\xi + a_2 = (a_0\xi + a_1)\xi + a_2 \\ &\vdots \\ b_{n-1} &= b_{n-2}\xi + a_{n-1} = (\cdots((a_0\xi + a_1)\xi + a_2)\xi + \cdots + a_{n-2})\xi + a_{n-1} \\ R &= (\cdots((a_0\xi + a_1)\xi + a_2)\xi + \cdots + a_{n-1})\xi + a_n \end{aligned}$$

となる．すなわち組立除法による剰余 R の計算は，ホーナー法による $p_n(\xi)$ の計算順序に一致している．$p_n(\xi)$ は $p_n(x)$ を $x-\xi$ で割ったとき，その商を $Q(x)$，剰余を R で表わせば，

$$p_n(x) = (x-\xi)Q(x) + R, \qquad p_n(\xi) = R \tag{36}$$

である．両辺を x で微分すると

$$p_n'(x) = (x-\xi)Q'(x) + Q(x)$$

なので $p_n'(\xi) = Q(\xi)$ を得る．つまり，多項式 $p_n(x)$ の導関数 $p_n'(x)$ の $x=\xi$ での値 $p_n'(\xi)$ は，$Q(x)$ を $x-\xi$ で割ったとき，その剰余として求められる．

$$Q(x) = (x-\xi)T(x) + S, \qquad p_n'(\xi) = Q(\xi) = S \tag{37}$$

1 階導関数 $p_n'(x)$ の $x=\xi$ での値（接線の傾き）を求めるサブルーチンも作る．

```
EXTERNAL SUB Horner2(n,a(),x,r,s)
OPTION BASE 0
LET r=a(0)
LET s=a(0)
FOR i=1 TO n-1     ! i=1             i=2                i=3
   LET r=r*x+a(i)  ! b1=a(0)*x+a(1)  b2=b1*x+a(2)       b3=b2*x+a(3)
   LET s=s*x+r     ! s =a(0)*x+b1    s=(a(0)*x+b1)*x+b2 s=((a(0)*x+b1)*x+b2)*x+b3
NEXT i
LET r=r*x+a(n)
END SUB
```

2 階導関数まで考慮したサブルーチン Horner3 も作る．

```
EXTERNAL SUB Horner3(n,a(),x,r,s,t)
OPTION BASE 0
LET r=a(0)
LET s=a(0)
LET t=a(0)
FOR i=1 TO n-2     ! i=1             i=2                i=3
   LET r=r*x+a(i)  ! b11=a(0)*x+a(1) b12=b11*x+a(2)     b13=b12*x+a(3)
   LET s=s*x+r     ! b21=a(0)*x+b11  b22=b21*x+b12      b23=b22*x+b13
   LET t=t*x+s     ! t =a(0)*x+b21   t=(a(0)*x+b21)*x+b22 t=t*x+b23
NEXT i
LET r=r*x+a(n-1)
LET s=s*x+r
LET r=r*x+a(n)
LET t=t*2
END SUB
```

Horner2 サブルーチンを用いる例 NewtonSDFig.BAS を例題として加えた．このプログラムは，初期値（例えば 3）を与えると，$x^3 - 15x - 4 = 0$ の近似解をニュートン法反復で求め，収束までの過程を作画する．

Horner と Horner3 の C++ 版を，変数型を rational で，ratutil クラスに含めた．

```
rational Horner(const int n, const rational_vector& coef, const rational& x) {
   rational r = coef[0];
   for (int i=1; i <= n; ++i)  r = r*x + coef[i];
   return r;
}
void Horner3(const int n, const rational_vector& coef, const rational& x,
                              rational& r, rational& s, rational& t) {
   if(n >= 2){
      r = coef[0]; s = coef[0]; t = coef[0];
      for (int i=1; i < n-1; ++i) { r = r*x + coef[i]; s = s*x + r; t = t*x + s; }
      r = r*x + coef[n-1]; s = s*x + r; r = r*x + coef[n]; t = t*rational::TWO;
   }else if(n ==1){
      r = coef[0]*x + coef[1]; s = coef[1]; t = rational::ZERO;
   }else if(n ==0){
      r = coef[0]; s = rational::ZERO; t = rational::ZERO;
   }
}
```

3.5　1 次の因子探し

包含が成立しているグループに対して[55]，近似固有値を askinty 関数で評価して，整数と見なせればこの整数を rational 型の変数に抽出し，1 次式 $v(\lambda)$ の定数項として格納し，$v(\lambda)$ で $p(\lambda)$ を割ってみる（polydivchk 関数）．

1 次の因子探し

```
for(i=1; i<=ngrp1;i++){
   if(included[i]){                                    包含が成立していれば
      if(askinty(eigvl[i],epsint,vrat[0])){            整数性が満たされていれば
         for (j=0; j<=nsize; j++) urat[nsize-j]=p[j]; 特性多項式を urat に
         vrat[1]=-rational::ONE;                       割るほうの vrat を作り
         if(polydivchk(urat,nsize,vrat,1,qrat,rrat)){ 割り切れれば
            included[i]=0; nsize --;
            for (i=0; i<=nsize; i++) p[nsize-i]=qrat[i];  特性多項式を商多項式で置換え
            npolynomial=putpolynomial(1,vrat,npolynomial,ppolynomial,cpolynomial);
} } } }
```

割り切れれば戻り値が 1 で，このとき商多項式で特性多項式を置き換え，次数 nsize をデクリメントし，1 次式を因子多項式として putpolynomial 関数で保存する．

3.6　2 次以上の因子探し

ループ構成は，探索する多項式の次数 d を 2 次から nsize 次まで探索するが，因子多項式が見つかるたびに，特性多項式が因子で割られて，商多項式によって置き換えられるので，特性多項式の次数 nsize が小さくなる．このような**ループ端が変化する場合のループ構成**には注意が必要である．Fortran の DO ループや BASIC の FOR ループでは，ループの反復終端が反復中に変化しても，反復は反復開始前に設定した回数を実行する．これに対し，C や C++の for ループは，ループの反復終端が反復中に変化すると，変化に応じて反復は終了する．このようなプログラミング言語の差異を考慮して，dvsrch.cpp では while(d <= nsize) ループを用いた[56]．

はじめに探索のループで呼び出す setecand 関数は，包含が成立している近似固有値だけを選び，詰めて配列 evcan に格納し，また，次数 d 以下の番号でグループ番号を間接参照するための配列 usev を用意する．戻り値は選択された固有値の数である．関数名は「固有値（Eigenvalue）の候補（candidate）をセットする」の意である．

```
int setecand(const vector<double>& eigvl, const int ncan,
         const int included[], double evcan[], int usev[]){
   int j=0;
   for(int i=1; i<=ncan;i++){
      if(included[i]){ j++; usev[j]=i; evcan[j]=eigvl[i]; };
   }
   return j;
}
```

探索は 2 重ループで行う．外側の while ループは，因子多項式の次数 d の反復で，内側の for(ii=1; ii<=nn;ii++) のループで固有値の候補から組合せて因子多項式を作る反復である．外側のループ反復で，setecand の戻り値を ncandi に保存すると，ncandi 個から d 個を選ぶ組合せ ${}_{ncandi}C_d$ を準備する．noff は固有値の候補 evcan の先頭から noff 個はすでに探索済みで探索対象から除外する数が格納される．配列 idx を初期化し，nn=comb(ncandi,d) によって組合せの総数が決められる．

[55] for(i=1; i<= ngrp1; i++) ループを if(included[i]) で条件選択する．
[56] C 言語の while ループと同等のループ構成は，Fortran では do while(...) 文 と end do 文で，十進 BASIC では DO WHILE ... 文と LOOP 文で行う．BASIC ではブランクがセパレータなので，条件を括弧で囲わない．

内側のループの内部に 3 つの if ブロックがある．内側のループ反復中に候補多項式が特性多項式を割切る（因子が見つかる）と，特性多項式の次数が d だけ小さくなる．因子多項式を putpolynomial で登録し，included 配列を更新することで候補から除外する．この時点で配列 idx の状態によって，同じ次数 d の候補探しがまだ残っていて継続するか，あるいは終了するかで探索方法が変わる．継続の場合は，reset フラグを立てて break するが，候補が少なければ終了なので reset フラグを立てずに break する．ループを抜けて，外側の while ループで reset フラグを見て，次数 d をインクリメントするかしないかを選択する．

2 次以上の因子探し

```
d=2; noff=0;
while(d<=nsize){                   // search for polynomial of degree 2 to nsize
   ncandi=setecand(eigvl,ngrp1,included,evcan,usev); // set number of candidate eigenvalues
   for(i=1; i<d;i++) idx[i]=i+noff; idx[d]=d+noff-1; // initialize for nxtcmb
   nn=comb(ncandi,d); reset=0;    // since nn is type "long long", ngrp1-nuse should be "< 62"
   for(ii=1; ii<=nn;ii++){        // loop over nn combinations of candidates
      nxtcmb(ncandi,d,idx);       // select one of _(ncan-nused) C_d candidates
      u=0;
      for(j=1; j<=d; j++){ figvl[j]=evcan[idx[j]]; u=u+figvl[j];}
      if(askinty(u,epsint,vrat[0])){  // pass the first cofficient check, then ...
         int ok=1;
         for(int r=1; r<=d; r++){
            cof[r]=vietaterm(figvl,d,r); // make coefficient terms of Vieta's formula from figvr
            if(! askinty(cof[r],epsint,vrat[d-r])){ ok=0; break; } // if(not inty) ok=0, break
         }
         if(ok){                               // OK all terms satisfy inty, try trial division
            vrat[d]=rational::ONE;
            for (j=0; j<=nsize; j++) urat[nsize-j]=p[j];  // set urat with polynomial
            if(polydivchk(urat,nsize,vrat,d,qrat,rrat)){  // trial division divided up
               for (j=0; j<=nsize-d; j++) p[nsize-d-j]=qrat[j]; // set quatient for next p(\lambda)
               nsize = nsize-d;                                 // and decrease nsize
               npolynomial=putpolynomial(d,vrat,npolynomial,ppolynomial,cpolynomial);
               for (j=1; j<=d; j++) included[usev[idx[j]]]=0;   // upate array included[.]
               noff=idx[1]-1;                  // number of OFFed indecies
               if(d+noff>ncandi) break;        // there are no degree d left, then break for loop
               reset=1; break;     // candidates left for this degree d, set reset flag then break
            }    // if(polydivchk(urat,nsize,vrat,d,qrat,....)
         }       // if(ok){
      }          // if(askinty(u,epsint,vrat[0]))
   }             // for(ii=1; ii<=nn;ii++){
   if(reset==0){ d++; noff=0;} // if(reset) continue loop with degree d, otherwise increment d
}                // while(d<=nsize){
```

5 分割で H_9 を扱う場合には，グループ 2,4,6,8,10,12 の 6 つの候補から idx[1]=1, idx[2]=4 のとき，usev[1]=2, usev[4]=8 で，$e_2 = 1.2679\cdots \fallingdotseq 3-\sqrt{3}$ と $e_8 = 4.7320\cdots \fallingdotseq 3+\sqrt{3}$ が選択されると，係数 -6 と 6 が作られて特性多項式が 2 次式 $(\lambda^2 - 6\lambda + 6)$ で割り切れる．この組合せは ${}_6C_2 = 15$ のうちの 3 番目である．2 つのグループが除外され，setecand で 4 グループがリセットされるが，割り切れたときの idx[1]=1 なので，noff $= 0$ より，リセットされる idx は 1, 1 である（4 候補を始めから探索）．

次にグループ 4,6,10,12 の 4 つの候補から idx[1]=1, idx[2]=3 のとき，usev[1]=4, usev[3]=10 で，$e_4 = 2.2679\cdots \fallingdotseq 4-\sqrt{3}$ と $e_{10} = 5.7320\cdots \fallingdotseq 4+\sqrt{3}$ が選択されると，係数 -8 と 13 が作られて 2 次式 $(\lambda^2-8\lambda+13)$ で割り切れる．この組合せは ${}_4C_2 = 15$ のうちの 2 番目である．2 つのグループが除外され，setecand で 2 グループがリセットされるが，割り切れたときの idx[1]=1 なので，noff $= 0$ より，リセットされる idx は 1 である（2 候補を始めから探索）．

最後にグループ 6,12 の 2 つの候補から idx[1]=1, idx[2]=2 のとき，usev[1]=6, usev[3]=12 で，$e_6 = 3.2679\cdots \fallingdotseq 5-\sqrt{3}$ と $e_{12} = 6.7320\cdots \fallingdotseq 5+\sqrt{3}$ が選択されると，係数 -10 と 22 が作られて 2 次式 $(\lambda^2-10\lambda+22)$ で割り切れる．この組合せは ${}_4C_2 = 15$ のうちの 1 番目である．ncandi がなくなるので探索は終わる．この場

合，直観的に H_{13} から ${}_{13}C_r$ による多くの反復を想像しがちだが，このように次数の低い因子が見つかると，組合せループの反復は $3+2+1$ の 6 回で済む.

H_{13} は H_9 の特性多項式で割り切れるため，4 次式となり，これが 1 次式で割り切れるためすぐに計算が終わる.

熱伝導行列は構造が単純で高い多重度をもち，低い次数の因子多項式に分解されるため，組合せループを探索する回数も少なく，このアルゴリズムでも分解は短時間で終了する.

4　有理数係数の行列の解析プログラム dvsrch1.cpp

前章のテストプログラムで扱った行列の固有値は 0.1 と 10 の間に収まっていたので，浮動小数点計算で得られた固有値を，そのまま近似値として使用し，倍精度浮動小数点演算で整数性も判定できた．倍精度浮動小数点計算で得られた固有値の精度を 10 進で 15 桁とすると，近似固有値の小数点以下に 14 桁の有効数字があり，また因子多項式の係数も少ない桁数で収まっていたからである．しかし目標の倍精度浮動小数点演算で求められた行列の解析には，近似固有値の精度の問題を解決しなくてはならない．倍精度浮動小数点数は数学的には有理数である．行列要素を有理数変換し，さらに整数係数の特性多項式を得るには，有理行列の全要素の分母の LCM を求め，LCM 倍するので，特性多項式の係数は大きくなる．また固有値も LCM 倍されるため，倍精度計算で得られた近似値を使って整数性から因子を探すには，近似固有値の精度が不足する（dvsrch.cpp は，データの好条件に支えられて稼働した）．

計算機で行う「計算」で扱える整数は $2^{31}-1$ 以下とか $2^{63}-1$ 以下などと，大きさに制限があり，数学問題を計算機で確かめることができないことが多い．既存の多くの汎用プログラミング言語で扱える整数はごく小さな数だけである[57]．一方，浮動小数点演算は，限られた精度に丸めて計算するため，正確な計算はできない．精度の解決策として，倍精度を 4 倍精度に，4 倍精度でも不足すると 8 倍精度に，さらに 16 倍精度にと，多倍精度算術演算を導入する方法も考えられるが，この方法では機能しない問題がある[58]．

要求される有効数字の桁数は，整数性判定が小数点以下の数桁を見るので（絶対誤差評価的になり），固有値の小数点の上に何桁の数字があるかと，因子多項式の次数に依存する[59]．特性多項式の次数が与えられても，因数分解後の因子多項式の次数は分からない．そこで**近似固有値を 2 つの有理数型の変数で挟み，区間で捉える．探索すべき因子多項式の次数に応じて近似固有値の精度を，有理算術演算によって必要なところまで高める**．

テストに使用する行列は，前章で用いた行列の熱伝導率を 2 進数では正確に表せない 0.1 にすることで，行列要素の値を大きくする．また，零固有値を持つ問題（正方格子に対するグラフ・ラプラシアン行列）の多重度解析も行う．この行列はヘッセンベルグ変換によって複数の小行列に分割できるところまでは熱伝導行列に似ているが，特性多項式が低次の因子多項式に分解されにくい．このため，計算時間の配分が熱伝導行列の場合と異なる．これらの行列の解析を通して，有理算術演算に特有の高速化技法を整備する．

なお，dvsrch1.cpp で扱う行列は，ヘッセンベルグ小行列の固有値は，浮動小数点演算で得られた近似固有値のグループに 1 つだけ存在する（**小行列の固有値 2 つが，1 つのグループから選ばれて因子多項式を構成することがない）．この前提を設けることで，プログラムを単純にする**．次章で扱う数値シミュレーションプログラムの行列ではこの前提を外す．

4.1　桁数の多い数値に対する縮小反復による精度改良

包含が成立した固有値の近似値は，区間の上限と下限を正確に抑えられる．ここで**有理算術演算による 2 分法を用いて，包含が成立した区間を縮小反復**する．浮動小数点演算では，区間縮小には限界があるが，有理算術演算なので，原理的にはどこまでも区間幅を縮小できる．2 分法で挟み込まれた区間の上下限をそれぞれ，i 番目のグループに属する固有値の下限（lower）と上限（upper）を，interval 型の配列の要素 `eigint[i]` に格納し，因子多項式を作る段階で整数性の判定に必要な精度まで高める．これが有理算術演算で無理数を必要な精度まで求めて精度保証する基本方針であるが，桁数の増大にともない，「**整数性判定に近似固有値の小数点の下の 53 ビットだけを取り出して倍精度で判定**」と「**多項式係数を作成する順序の最適化**」の高速化が必要になる．はじめにこれらの改良点の要点を述べる．

[57] 十進 BASIC の有理数モードや Java の BigInteger クラスを用いると，大きな整数も計算することができる．

[58] 次章で述べる，零近傍に複数の固有値が密集する問題では，浮動小数点演算で得られた近似固有値の精度がなく，何倍長精度で計算すればよいかを決められない．

[59] 浮動小数点演算では相対誤差評価を用いることが多い．しかしここで用いる整数性の判定は，d 次の因子多項式を d 個の近似固有値から推定するとき,「整数かどうか」を使う．このために，絶対誤差評価になる．

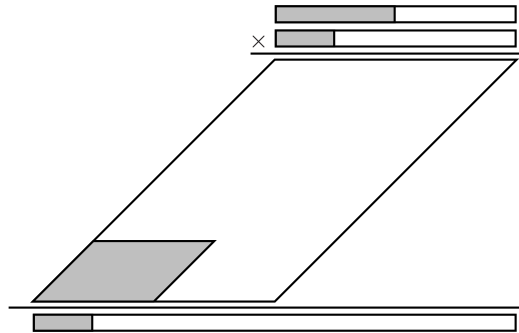


図 4: 乗算の正しい桁（グレー部分は正しい桁）

整数性判定に必要な精度について述べる．固有値 $3\pm\sqrt{3}$ の例で，有効数字を 3 桁として，$e_2 \fallingdotseq 1.27$，$e_{16} \fallingdotseq 4.73$ として，近似固有値の精度の問題を考えてみよう．この例では，抽出されるべき 2 次式の係数は -6 と 6 である．近似固有値の和は $-r = e_2 + e_{16} = 6.00$ で積は $s = e_2 \cdot e_{16} = 6.0071$ になる．係数行列を 10 倍すると固有値も 10 倍されて 12.7 と 47.3 になり，和は 60.0，積は 600.71 になる．次の桁に誤差が混入していて，12.74 と 47.34 だとしたら，和は 60.08 だが積は 603.1116 で，積の整数性は正しく得られない．つまり，整数性判定は，近似固有値の精度が，固有値の小数点よりも上の桁数と，根と係数の関係公式で使用する根の総積の項数に依存する．倍精度浮動小数点数を有理数化した場合は LCM が大きくなり，固有値自身も桁数の大きな数になる．行列 (27) の場合は $e_1 = 7,205,759,403,792,791$ と 16 桁の数になるので，少なくとも 16 桁以上の精度が要求される．このときの因子多項式の定数項がもっとも厳しい条件を与える．因子多項式の次数を d とした場合，定数項は総積 $p_d = \prod_{k=1}^{d} \alpha_k$ である．

s 桁と t 桁の数の積は st 桁になる．因子の次数を d とした場合，定数項は総積 $p_d = \prod_{k=1}^{d} \alpha_k$ である．被乗数の上位 u 桁と乗数の上位 v 桁が正しいとき，積で正しいのは u と v の小さいほうの桁数だけである（$u > v$ の場合，積の上から $v+1$ 桁目には誤差が入る可能性がある．図 4 のグレー部分は正しいと考える）．

必要な精度（ビット数）は，固有値の近似値 e_i から，小数点よりも上のビット数（2 進数表現したときの桁数）を $\log_2 e_i + 1$ によって得て，計算誤差の伝搬を考慮してこれに 2 を加えてから d 倍した値

$$d \cdot (\log_2 e_i + 3) \tag{38}$$

が，p_d の 2 進数での桁数（ビット数）なので，根と係数の関係式に代入する e_i の精度がこれ以上になるまで精度改良する．

因子多項式の次数は特性多項式からは分らないので，既知の因子で割った後の多項式に対して，まず 1 次式用に設定する．**固有値が有理数なら 1 次式で割りきれ**，分解対象の多項式の次数も下がる．次に候補の因子多項式の次数を，2 次から 3 次，4 次と上げてゆくループの中で，次数に応じて精度を高める．次数 d に関するループ中に精度改良を入れるので，不必要に桁数を増加させない．

整数性判定は，dvsrch.cpp でも p_1 についてだけ先行判定したが，dvsrch1.cpp では扱う数の桁が多いので，有理数と浮動小数点数の両方を使えるプログラミング環境に特有の高速化技法である**必要桁の抽出**を用いる．図 5 に 3 つの近似固有値を加えて p_1 を求める場合に，判定に必要となる桁の位置を示した．小数点よりも上の桁は外してよく，また小数点以下数 10 桁の小さな桁も判定から外す．近似固有値を（`floor()` 関数を用いて）小数点以下だけにし，これを倍精度型の変数に変換して小数点以下を 53 ビットの精度で取り出して倍精度浮動小数点演算で p_1 の小数部分を求めて先行判定する[60]．

[60]丸め誤差を含む計算なので，dvsrch.cpp でも使用した `epsint` のように設定にデリケートな要素を含む（問題に応じて変更しなければならないかもしれない）．

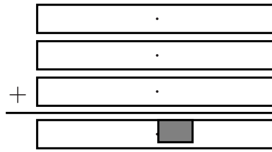


図 5: 整数性判定で必要な桁

次に，先行判定を通過した近似固有値の組合せから，**多項式係数** p_1，p_2，p_3，$\cdots$ **を求める計算**について述べる．根と係数の関係公式の乗算回数を最小にするには計算順序を最適化する（5 次の場合，公式通りに p_2，p_3，p_4，p_5 の順序で計算すると 49 回になるが，最適化すると項数と同じ 26 回にできる）．この順序には再帰呼出しを使う．

根と係数の関係公式は乗算と加減算だけで除算がない．そこで精度改良では，区間の上下限の有理数を構成する分母を 2 のべき乗にしておき，公式の計算でも「分母 2 べき」を続ける．区間縮小アルゴリズムを 2 分法にしておくと,（浮動小数点数を有理数変換した有理数は「分母 2 べき」なので），途中の計算も最後の精度改良された近似固有値も，**分母 2 べきのままなので** p_i **を高速計算**できる．

因子探索には,「必要桁の抽出」や「分母 2 べき」のような有理算術演算に特有のプログラミング技法によって高速化でき，正確な特性多項式と 10 数桁の精度の近似固有値を当て嵌めることで（数式処理システムを使わずとも）因数分解ができる．

以下，本節では，interval 型の紹介の後，interval 型による整数性の判定，多項式係数の生成についてのプログラムを説明する．

4.1.1 有理数による区間算術演算

Fortran の `complex` 型が，複素数の実部と虚部を 2 つの浮動小数点数で把握するように,「有理数計算プログラミング環境」では，2 つの有理数で区間の下限と上限を把握する `interval` 型の変数を設けた．これによって区間算術演算を数のようにプログラミングできる．

`interval` 型の変数同士，また `interval` 型と `rational` 型の変数の算術演算は，記号 $+$, $-$, $*$ でプログラミングできる（C++ の演算子多重定義を用いた）．

区間 $x = (a, b)$ を 2 つの有理数 a と b で表すには，“`interval x(a,b)`” と書く．区間は上限と下限が異符号（零を含む区間）はエラーとする場合がある．区間 x からその下限を rational 型の変数に取り出すには `x.lower()`，上限を取り出すには `x.upper()` を使用する．区間 $x = (a, b)$ の符号を反転すると $(-b, -a)$ になるが，これには `y=x.neg()` と書く．区間 $x = (a, b)$ の逆数 y は `y=x.inv()` と書く．

区間 $x_1 = (a_1, b_1)$ と $x_2 = (a_2, b_2)$ で表される 2 数の加算（乗算）は，**両区間が正の場合**は，下限同士の和（積）と上限同士の和（積）になる．

$$x_1 + x_2 = (a_1 + a_2, b_1 + b_2), \quad x_1 x_2 = (a_1 a_2, b_1 b_2)$$

減算は，下限は小さいほうから大きいほうを引き，上限は大きいほうから小さいほうを引く．

$$x_1 - x_2 = (a_1 - b_2,\ b_1 - a_2)$$

区間幅が 0.2 の 2 数 $x_1 = (19.9, 20.1)$ と $x_2 = (9.9, 10.1)$ の加算は，$x_1 + x_2 = (29.8, 30.2)$，減算は $x_1 - x_2 = (9.8, 10.2)$ で，区間幅はどちらも 0.4 に広がる．

区間は常に下限が $-\infty$ 側にあり，上限が $+\infty$ 側にあるので，加減算は簡単だが，乗算は符号を考慮する必要がある．両区間とも負の場合は積は正になるので反転して $x_1 x_2 = (b_1 b_2, a_1 a_2)$ になる．x_1 が負 x_2 が正の場合

は $x_1x_2 = (a_1b_2, b_1a_2)$ になり，x_1 が正 x_2 が負の場合は $x_1x_2 = (b_1a_2, a_1b_2)$ になる．この 4 通りを実装すれば，interval 型の数同士の乗算も，普通の数値に対するプログラムのように記述できる．

除算は逆数を掛ける．

interval 型の数同士の演算以外に，interval 型と rational 型の演算も，$+, -, *$ の記号で記述できる．

interval 型を用いると，**整数性は区間に整数が含まれるかどうかで判定**できる．これには，上限値の有理数の `floor()` と下限値の有理数の `ceil()` の一致で判定する[61]．

interval 型の変数 `eigint[i]` の表示（プリント）には `formatprn(eigint[i]]);` とすると，区間の上と下の値を比較して，差異の始まる位置を検出して表示する．後述する例であるが小数点以下 75 桁目に違いが生ずる場合を示す．

```
583767772594036.526309582779885448 (47 digits) 8436625927967713587
583767772594036.526309582779885448 (47 digits) 8436625928275423346
```

区間が整数を含む場合は次ように表示される．

```
64851834634135145.999999999999999999 (47 digits) 9999999999907331953
64851834634135146.000000000000000000 (47 digits) 215051817
```

`formatprn` 関数は，10 進変換された数字の一致からではなく，区間幅から表示する桁数を求める．

4.1.2 interval 型の数を用いた因子探索

整数性の判定値 `epsint` が隣接するグループの固有値よりも大きいと，正しい因子を見つけられない．$e_1 \fallingdotseq e_2$ のとき，$e_1 + e_3 \fallingdotseq e_2 + e_3$ であるから，本来は $e_2 + e_3$ で多項式係数を作るところを，同じ係数を先に $e_1 + e_3$ で作ってしまうからである．そこで dvsrch1.cpp では隣接グループの近似固有値の最小値を求めて，初期設定した `epsint` よりもこの値が小さい場合は，これによって置き換えることでこれを防いだ[62]．これはグループ化（`selectev` 呼出し）の直後に入れる．

epsint の固有値の接近による置換え

```
for (i=1; i<=ngrp1; i++){
   if(i==1){ eveps=epsint;
   }else{
      if(fabs(eigvl[i]-eigvl[i-1]) < eveps) eveps = fabs(eigvl[i]-eigvl[i-1]);
   }  }
if(epsint > eveps) epsint=eveps;
```

この更新された `epsint` は，2 分法反復 `bisect` 関数での収束判定値や，整数性の先行判定で使用する．

dvsrch1.cpp で扱うデータは，浮動小数点演算で得られた近似固有値で作成したグループに，2 つ以上の根が存在することがない（多重度の高い固有値は，別の小行列に分かれて現れる）という前提でプログラムを作っている．したがって，グループの下限と上限を使って**包含（inclusion）を作成**することができる．包含に先立って，次の `chkinc` 関数で，区間端点で関数値の積が負になることを確かめる．

[61] 多桁数の乗算で FFT を使用すると，畳込み演算は浮動小数点演算で行い，結果を整数に戻す際，精度チェックでは整数性を使用する．したがって有理算術演算では整数性は重要なアルゴリズムである．

[62] グラフ・ラプラシアン行列で “G 9” の場合に，グループ 7,12,22,28,35 で 5 次多項式 $(\lambda^5 - 14\lambda^4 + 69\lambda^3 - 142\lambda^2 + 112\lambda - 29)$ が生成されるが，$e_{11} = .7560\cdots$ で $e_{12} = .7561\cdots$ と 0.01 以内にあるため，`epsint=0.01` ではこの処理なしではエラーとなる．

chkinc 関数

```
int chkinc(const int n, const rat::vector<rational>& p, const double e, const double h,
        const int mult, rational& a, rational& b){ // a と b を引数に加えた
   double ta,tb;
   int rtnval=0;
   rational fa,fb;
   if(fabs(e) > eps){
      ta=e-fabs(e)*eps; tb=h+fabs(h)*eps;
      if(mult==0) tb=e+fabs(e)*eps;
   }else{
      ta=e-eps; tb=h+eps;   // consider eigenvalues close to zero
      if(mult==0) tb=e+eps;
   }
   a = Rdset(&ta); fa=Horner(n,p,a); b = Rdset(&tb); fb=Horner(n,p,b);
   if(fa*fb<rational::ZERO) rtnval=1;
   return rtnval;
}
```

戻り値は根を挟んでいれば 1 である．

chkinc 関数は，根を挟み込む区間 (a, b) を引数で返すので，次のように呼び出し，端点を eiglow[i] と eigup[i] として抑える．区間両端を interval 型の配列要素 eigint[i] に格納する．包含（inclusion）が特性多項式の次数に一致すれば包含成立で，不足していたら処理を中止する．

inclusion の成立と 1 次因子探索用の区間の縮小反復

```
   nexist=0;
   for(i=1; i<=ngrp1;i++){
      included[i] = chkinc(nsize,p,eigvl[i],eigvh[i],mult[i],eiglow[i],eigup[i]);   // 根を含むと 1 が返る
      nexist += included[i];  // 包含をカウント
      eigint[i]=interval(eiglow[i],eigup[i]); // interval 型の数に下限と上限を格納
   }
   if(nsize == nexist){                      // 包含（inclusion）成立
      for(i=1; i<=ngrp1;i++){
         if(included[i] > 0){
            eigvr[i]=Rdset(&eigvl[i]);        // 浮動小数点数を有理数に変換
            faeig=Horner(nsize,p,eiglow[i]); fbeig=Horner(nsize,p,eigup[i]);
            if(faeig*fbeig > rational::ZERO){
               cout << " ERROR check Horner .....faeig=" << faeig << " fbeig=" << fbeig << endl; exit(3);
            }
            int irtn=bisect(p,nsize,epsint,100,eigint[i],faeig,fbeig,eigvr[i]); // tol を epsint で縮小反復
            included[i] = abs(irtn); formatprn(eigint[i]);
         }
      }
   }else{
      std::cout << "ERROR nsize != nexist  STOP" << std::endl; exit(3); // 包含不成立なら処理を中止
   }
```

bisect 関数呼び出しは，1 次の因子を探すために，収束判定値に整数性判定値 epsint を指定して精度を改良する（bisect 関数は ratutil クラス）．必要な精度が与えられ，包含が成立すれば，近似固有値の精度は 2 分法反復で改良できる．

2 分法は高校の数学 B の教科書にも載っていた素朴なアルゴリズムで，根の存在する区間 $(a,\ b)$ の中点 $c = \dfrac{a+b}{2}$ で関数値を計算して，$p(a) \cdot p(c) < 0$ なら区間 (a, c) を，$p(c) \cdot p(b) < 0$ なら区間 (c, b) を選択する [8]．したがって，反復ごとに区間幅を半分にしてゆける（区間幅は反復 k 回で 2^{-k} に縮小される）．

収束判定には，関数値を使うか，区間幅を使うかの選択肢がある．正確な計算により多項式の零点を挟む区間を縮小するには，区間幅を収束判定値に用いるほうが使いやすい．bisect 関数は ratutil クラスに関数である．

```
int bisect(const rat::vector<rational>& p, const int n, const double tol, const int maxit,
     interval& ab, rational& fa, rational& fb, rational& c) {   // 引数 ab は interval 型
   rational a, b, fc, fdc, fddc;
   int it=0, itshift=0, rtnval;
```

```
   Horner(n,p,ab,fa,fb);
   a=ab.lower(); b=ab.upper();   //  switch from interval to two rational
   while( it < maxit){
      it++; totalit++;
      if(fa*fb > rational::ZERO){
         cout << " bisect(interval) error  STOP" <<  endl; exit(3);
      }
      if(it >= maxit){
         cout << " bisect(interval) MAXIT reached it=" << it << endl;
         rtnval=-it; break;
      }
      c = (a + b) / rational::TWO;
      Horner3(n,p,c,fc,fdc,fddc);
      if(fc == rational::ZERO){      // 有理数解のとき，c が正解になることがある
         a=c; fa=fc; b=c; fb=fc; rtnval=0; break; // 戻り値に 0 を入れる
      }
      if(fa*fc > rational::ZERO){    // 解は (c,b) 間にあり
         a=c; fa=fc;                 // 区間の下限 a を c で置換え，対応する関数値も置換え
      }else{                         // 解は (a,c) 間にあり
         b=c; fb=fc;                 // 区間の上限 b を c で置換え，対応する関数値も置換え
      }
      if((double)(b-a) < tol){       // 収束したら
         cout << " bisect(interval) converged  b-a=" << setprecision(16) << (double)(b-a) << endl;
         rtnval=it; break;           // 反復回数（正数）を返す
      }
   }
   ab=interval(a,b);                 // 区間引数を更新
   return rtnval;
}
```

これを用いると，区間 (a, b) に根が包含されたら，区間幅を指定した値まで確実に縮小できる．戻り値は，有理数解に収束した場合は 0 を，その他の収束の場合は反復回数である．指定された反復回数でも区間幅が `tol` 以下にならなかったら，反復回数の (-1) 倍を返す．bisect 関数は，1 次の因子を見つけるための精度改良として，区間幅が `epsint` よりも小さくなるように指定して使用する．なお，`Horner` も ratutil クラスの関数である．

dvsrch.cpp では **1 次の因子探索**を，`askinty` 関数に `epsint` を渡して判定していたが，interval 型の数に近似固有値を格納したので，区間が整数を挟むか否かで判定できる．1 次の因子探しの整数性判定は次のように書ける．interval 型変数の上限値を interval クラスのメンバー関数 upper() で抽出し，rational クラスのメンバー関数 floor() で小数点以下を捨て，これと下限値の ceil() の一致で判定する[63]．

```
      if(eigint[i].upper().floor() == eigint[i].lower().ceil()){
```

区間を使用することで，整数性判定に，`epsint` **のように閾値の設定が難しい値を不要にできた**．ただし，有理算術演算では `floor()` 関数は，longint 型の除算を行うため短時間で処理できない．このため，実行回数が多いカーネルでは工夫が必要になる．

2 次以上の因子探索は，dvsrch.cpp と同様に，近似固有値の候補を選び，根と係数の関係公式を使用して係数を作成して，その整数性を判定することで行うが，精度改良が必要になる．必要となる近似固有値の精度は，因子多項式の**項の次数に比例**して増える．プログラムでは，式 (38) の値を用いる．因子多項式の次数は特性多項式からは分からないので，既知の因子で割った後の多項式に対して，まず 1 次式用に設定する．固有値が有理数なら 1 次式で割りきれる．次に候補の因子多項式の次数を，2 次から 3 次，4 次と上げてゆくループの中で，次数に応じて精度を高める．次数 d に関するループ中に精度改良を入れるので，不必要に桁数を増加させることを

[63] `floor(t)` と書かないで `t.floor()` と書くのは，`floor` 関数が rational クラスのメンバー関数だからで，オブジェクト指向言語の流儀である．ratutil.h でメンバー関数として定義し，`public` としている．

避けられる．

ここで**整数性判定の高速化**について述べる．区間の上下限の有理数の小数点以下の桁数が増えると，整数性判定の高速化が必須になる．ここでは精度が十分に改良された固有値の小数部分だけを取り出して倍精度浮動小数点数に戻し，整数性判定を短時間で処理する．44 ページの図 5 に 3 つの近似固有値を加えて 1 次の係数を求める場合に，整数性判定に必要となる桁の位置を示した．小数点よりも上の桁は外してよく，また小数点以下数 10 桁の小さな桁も判定に関係ない．計算を高速に行うために，ここでは setecanint 関数に倍精度型の dfcand 配列を引数に追加して，小数点以下を 53 ビットの精度で取り出す．これには有理数型の固有値から，固有値を floor() で小数点以下を切り捨てた有理数を引くことで得られ，これを倍精度浮動小数点数にキャストする．

setecanint 関数

```
int setecanint(const rat::vector<rational>& eigvr, const vector<interval>& eab, const int ncan,
   const int included[], rat::vector<rational>& ercan, rat::vector<interval>& ercanab,
   vector<double>& dfcand, int usev[])
{
   int j=0;
   for(int i=1; i<=ncan;i++){
      if(included[i]){                        // 包含が成立した近似固有値だけを選ぶ
         j++;
         usev[j]=i; ercan[j]=eigvr[i]; ercanab[j]=eab[i];
         rational tmp=eab[i].upper();         // 区間の上限値
         dfcand[j]=(double)(tmp-tmp.floor()); // その小数点以下を取出し，double precision にする
      };
   }
   return j;    // return number of candidates set in array evcan[1] to [j]
}
```

因子多項式が 2 次以上になると，while ループの中で，多項式の次数 d に応じて精度改良を行う．

精度改良（2 次以上の因子探索用）

```
for(i=1; i<=ngrp1; i++){
   if(included[i]>=1){
      log2ev=log2(eigvl[i])+1;                  // log_2(e_i)
      int reqndeig=((int)log2ev+2)*d+accdouble; // Required Number of Digits d*( log_2(e_i)+3)
      if(chkonce(nsize,p,eiglow[i],eigup[i],faeig,fbeig)==0){
         cout << "chkonce ERROR then STOP" << endl; exit(3);
      }
      double stmp=pow(2,log2ev*d); double epstmp=1/stmp; // 2 進の桁数（ビット数）を 10 進桁数に変換
      int irtn=bisect(p,nsize,epstmp,reqndeig,eigint[i],faeig,fbeig,eigvr[i]);
      if(irtn < 0){
         cout << " bisect not converged STOP" << endl; exit(3);
      }
      if(irtn==0){  // 有理数の根に一致した場合だが，2 次以上ではないはず
         cout << " bisect encountered rational number solution" << endl;
      }
      formatprn(eigint[i]);
   }
}
```

chkonce 関数は，2 つの変数に対する関数値を Horner 関数で得て，2 つの関数値の積が正のときは戻り値を 0 にする．これはエラー検出が目的である．プログラムの開発段階では，エラーの早期発見は，デバッグの効率を高める効果がある．

この精度改良された近似固有値を使用して **2 次以上の因子探索**を行う．

2 次以上の因子探索ループ

```
d=2; noff=0; reset=0;
while(d<=nsize){                // search for polynomial of degree 2 to nsize
   if(d>nsize/2 && d!=nsize){  // d>nsize/2 は nsize までスキップする. d=nsize だけは近似固有値から作る多項式が
      d++; reset=0; noff=0; continue;// 特性多項式を割切るかどうかを確かめる (FACTORIZEONLY の#define なしの場合)
   }                                // reset, noff を初期化して continue
   if(reset==0){
      「上記の精度改良」がここに入る
   }
   ncandi=setecanint(eigvr,eigint,ngrp1,included,ercan,ercanint,dfcand,usev);
   for(i=1; i<d;i++) idx[i]=i+noff; idx[d]=d+noff-1; // initialize for nxtcmb
   nn=comb(ncandi,d); reset=0;      // since nn is type "long long", ngrp1-nuse should be "< 62"
   for(ii=1; ii<=nn;ii++){          // loop over nn combinations of candidates
      nxtcmb(ncandi,d,idx);         // select one of _(ncan-nused) C_d candidates
      double ud=0;
      for(j=1; j<=d; j++) ud+=dfcand[idx[j]];
      if(askinty(ud,epsint)){       //dvsrch1:  pass the first cofficient check, then ...
         for(j=1; j<=d; j++) figv[j]=ercanint[idx[j]];
         if(vietatermschk(figv,d,cor)){                                   // OK, try trial division
            for(int r=1; r<=d; r++) vrat[d-r]=cor[r]; vrat[d]=rational::ONE;
            for (j=0; j<=nsize; j++) urat[nsize-j]=p[j];
            if(polydivchk(urat,nsize,vrat,d,qrat,rrat){ // trial division divided up
               for (j=0; j<=nsize-d; j++) p[nsize-d-j]=qrat[j]; // set quatient for next p(\lambda)
               nsize = nsize-d;                             // and decrease nsize
               npolynomial=putpolynomial(d,vrat,npolynomial,ppolynomial,cpolynomial);
               for (j=1; j<=d; j++) included[usev[idx[j]]]=0;
               noff=idx[1]-1;  // number of OFFed indecies
               if(d+noff>ncandi) break;    // 次数 d の探索を終了
               reset=1; break;             // 次数 d の探索を継続
            }    // if(polydivchk(urat,nsize,....
         }      //  if(if(vietatermschk(figv,d,cor)){
      }        //   if(askinty(ud,epsint)){
   }          //    for(ii=1; ii<=nn;ii++){
   if(reset==0){ d++; noff=0; }
}          //      while(d<=nsize){
```

効果を計測するために，INTYDOUBLE を #define した[64].

T 指定と s 指定の比較 4×4 分割で整数行列 “T 4” を例とする（表 2）．はじめに 16 個の近似固有値をグループ化して 9 つの近似値が得られる．ヘッセンベルグ小行列をフロベニウス変換して特性多項式を得た段階で，近似固有値の各グループの上限に ε を加え，下限から ε を引いて，区間 (a_i, b_i) を定める．特性多項式は既知の因子で割ってから，包含を調べられる．2 番目に処理する H_3 の特性多項式は，$4-\lambda$ で割り切れて，商多項式で置き換えられ $p(\lambda)=\lambda^2-8\lambda+15$ になっている．$p(a_i)\cdot p(b_i)<0$ なら区間 i に 1 つ，あるいは奇数個の根がある．グループをすべて調べて，この次数が 2 の特性多項式 $p(\lambda)$ に対して，このような区間が 2 つ見つかれば，包含が成立する．4×4 分割の 2 番目に計算する小行列 H_3 の場合は，$e_4=3$ と $e_6=5$ だけが包含条件を満たす．

“s 4” の場合は，2 番目に処理する H_3 の特性多項式は，既知の因子 $(14,411,518,807,585,588-\lambda)$ で割られて次式になる．

$$p(\lambda)=\lambda^2-28,823,037,615,171,176\lambda+194,711,132,195,056,057,687,171,823,724,135$$

22 ページの式 (26) の倍精度浮動小数点数の 0.1 の分子を $k=3,602,879,701,896,397$ とおくと，

$$p(\lambda)=\lambda^2-8k\lambda+15k^2 \tag{39}$$

[64] **1 次の係数で行う判定は不完全**である．18 ページの表 3 の例で説明すると，$\pm\sqrt{3}$ を 3 組の固有値が持つので，和が整数になる組合せは 9 通りある．これは 2 次式の係数作成の例であるが，高次の係数でも同様の現象は現れる．したがってこの枝切りで落とされなかった組合せに対して，全係数を計算するときも，係数が確定した時点で整数性判定を行い，不要な係数計算を避ける必要がある．CT2D の最適化オプションを指定してコンパイルして実行したときの行列では，判定に用いる閾値が大きい（たとえば 0.01 を使う）と，先行判定を通過するものが非常に多くなり，性能を低下させる（“grep nok ファイル名” で調べられる）．ここでは dvsrch や dvsrch1 で使用した epsint よりも小さな，ファイル全体にスコープが及ぶ変数 $\texttt{eps}=10^{-9}$ を使用する．

である．これは “T 4” で得られる固有値を k 倍した値を固有値とする固有方程式の特性多項式を意味する．倍精度浮動小数点数を有理数変換した場合，分母は 2 べきなので，行列の全要素の LCM も 2 べきで $\mathrm{LCM} = 36,028,797,018,963,968 = 2^{55} = 10k - 2$ である．浮動小数点演算による丸め誤差の分布の差異が，固有値を，LCM 倍ではなく，k 倍し，因子多項式も式 (39) になった．24 ページの式 (28) に示したように，k は行列 A 全体にかかるので，因数分解の因子多項式の次数は “T 4” と同じ形になる[65]．

“T 5” のヘッセンベルグ小行列 H_9 の特性多項式は $-\lambda^9 + 36\lambda^8 - 564\lambda^7 + 5040\lambda^6 - 28263\lambda^5 + 102924\lambda^4 - 242744\lambda^3 + 356256\lambda^2 - 293772\lambda + 102960$ である．“s 5” で特性多項式を比較すると，H_9 の特性多項式の定数項は $105321 \cdots 4320$ で 146 桁の数値だが，k^9 で割ると商は 102960 で “T 5” の定数項と一致する．有理算術演算を用いると，数学的に成立する事柄を，数の大小に関係なくプログラムで確かめられる．

4.1.3 多項式係数の生成関数

計算を次に示すように，アンダーブレースで付記した数字の順番で行う．p_1 段の α_1 の 0 番から始める．p_2 段の $\alpha_1\alpha_2$ の 1 番を計算したら，下（p_3 段）に行き,「これ」に α_3 を掛けると，$\alpha_1\alpha_2\alpha_3$ は乗算 1 回でできる（「これ」はオーバーブレースを付加した $\alpha_1\alpha_2$ を指す）．同様に p_4 段に行き「これ」に α_4 を掛けると，$\alpha_1\alpha_2\alpha_3\alpha_4$ は乗算 1 回でできる（「これ」は $\alpha_1\alpha_2\alpha_3$ を指す）．同様に「これ」に α_5 を掛けると，$\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5$ は乗算 1 回でできる（「これ」は $\alpha_1\alpha_2\alpha_3\alpha_4$ を指す）．この関数は自分自身を呼び出す**再帰呼出し**で,「添字列」 1 や 12 や 123 などは引数の配列 `num` に，また「これ」も引数 `sp` に渡される．また,「次数ごとに使用した添字列の最後の数字」も引数の配列 `last` に保存する．ここまでで，次数の階層を 2 から 5 まで下り p_5 が完成した．以上の操作は，次数の階層を「下への移動」になる．

$$
\begin{aligned}
p_1 \quad & \underbrace{\alpha_1}_{0} \qquad \underbrace{\alpha_2}_{16-1} \qquad \underbrace{\alpha_3}_{23-1} \qquad \underbrace{\alpha_4}_{26-1} \\
p_2 = -(& \underbrace{\alpha_1\alpha_2}_{1} + \underbrace{\alpha_1\alpha_3}_{9} + \underbrace{\alpha_1\alpha_4}_{13} + \underbrace{\alpha_1\alpha_5}_{15} + \underbrace{\alpha_2\alpha 3}_{16} + \underbrace{\alpha_2\alpha_4}_{20} + \underbrace{\alpha_2\alpha_5}_{22} + \underbrace{\alpha_3\alpha_4}_{23} + \underbrace{\alpha_3\alpha_5}_{25} + \underbrace{\alpha_4\alpha_5}_{26}) \\
p_3 = & \underbrace{\overbrace{\alpha_1\alpha_2}\alpha_3}_{2} + \underbrace{\overbrace{\alpha_1\alpha_2}\alpha_4}_{6} + \underbrace{\overbrace{\alpha_1\alpha_2}\alpha_5}_{8} + \underbrace{\overbrace{\alpha_1\alpha_3}\alpha_4}_{10} + \underbrace{\overbrace{\alpha_1\alpha_3}\alpha_5}_{12} + \underbrace{\overbrace{\alpha_1\alpha_4}\alpha_5}_{14} + \underbrace{\overbrace{\alpha_2\alpha_3}\alpha_4}_{17} + \underbrace{\overbrace{\alpha_2\alpha_3}\alpha_5}_{19} + \underbrace{\overbrace{\alpha_2\alpha_4}\alpha_5}_{21} + \underbrace{\overbrace{\alpha_3\alpha_4}\alpha_5}_{24} \\
p_4 = -(& \underbrace{\overbrace{\alpha_1\alpha_2\alpha_3}\alpha_4}_{3} + \underbrace{\overbrace{\alpha_1\alpha_2\alpha_3}\alpha_5}_{5} + \underbrace{\overbrace{\alpha_1\alpha_2\alpha_4}\alpha_5}_{7} + \underbrace{\overbrace{\alpha_1\alpha_3\alpha_4}\alpha_5}_{11} + \underbrace{\overbrace{\alpha_2\alpha_3\alpha_4}\alpha_5}_{18}) \\
p_5 = & \underbrace{\overbrace{\alpha_1\alpha_2\alpha_3\alpha_4}\alpha_5}_{4}
\end{aligned} \tag{40}
$$

添字列の最後が “5” の場合は，上に戻る（順番 3 に戻る）.「リターン」すると，引数 `num` に，添字列 123，呼出されたときの「これ」である $\alpha_1\alpha_2\alpha_3$ が引数 `sp` に入手できる．次数 4 の「使用した添字列の最後」が 4 な（n よりも小さい）ので，4 に 1 を加えた 5 を用いて α_5 を掛けると $\alpha_1\alpha_2\alpha_3\alpha_5$ は 1 回の乗算でできる．この操作は「右への移動」になり，順番 5 の計算ができる．このように，次数の階層を下と上（リターン）と右に移動しながら計算すると，項数と乗算回数が一致する計算順序が得られる．

- 下への呼出しは次数 `r` を 1 増やして，添字の最後が n が出てくるまで，呼ばれたときの添字列の最後の数に 1 を加えた添字を加えて呼び出す．
- 添字の最後が n だったらリターンする（上への移動）．

[65] dvsrch1.cpp プログラムでは，浮動小数点演算で得られた近似固有値を倍精度浮動小数点演算で LCM 倍した．丸め誤差を含む近似値を，k 倍するところを LCM 倍していた．

- リターンしたとき，その次数の最後に使用した添字が n より小さければ，1 を加えた添字を次数 r のまま呼び出し（右への移動），添字の最後が n ならリターンする．

第 1 引数の配列 num は添字列，第 2 引数 sp が上記の説明の「これ」にあたる項が入り，第 3 引数に配列長の n，第 4 引数に次数 r，以下，近似固有値を配列 e，次数ごとに多項式係数を保存する配列 p，次数ごとに「使用した添字列の最後」を保存する配列を last，進む方向 dir（$=1$ のときは下へ移動，$=0$ のときは右へ移動）として，再帰呼出しを使用する mulvieta 関数を示す．なお，戻り値は整数性チェックの 1 か 0 である．

mulvieta

```
int mulvieta(vector<int>& num,const interval& sp,const int n,const int r,
const vector<interval>& e, vector<interval>& p,vector<int>& last,int dir) {
   int dgt, dgt1, rtnval=1;
   interval t;
   dgt=num[r];                // 引数 sp の積の最後の項
   if(dir == 1){              // 下に進むときは
      dgt1=dgt+1;             // 最後の項 dgt に 1 を加える
   }else{                     // 横に進むときは
      dgt1=last[r]+1;         // その次数で最後に掛けた項に 1 を加える
   }
   t=sp*e[dgt1];              // t = sp * \alpha_{dgt1}
   p[r]=p[r]+t;               // その次数の多項式係数に t を加える
   last[r]=dgt1;              // その次数で最後に掛けた項を記録

   if(r == n){                // check if the final term 123...n case
      if(p[r].upper().floor() != p[r].lower().ceil()) retern 0; // 整数性チェック
   }
   if(dgt1 < n){
      num[r+1]=dgt1;
      int inty=mulvieta(num,t,n,r+1,e,p,last,1);      // 下へ移動
      if(inty == 1){           <---- 再帰呼出しがリターンすると「この if 文」に制御が来る
         if(dgt1 <n) mulvieta(num,sp,n,r,e,p,last,0); // 最後の添字が n より小さければ右へ移動
      }else{
         rtnval=0;
      }
   }
   return rtnval;
}
```

配列 num を 1 として，2 番目の引数 sp に α_1 を入れて呼ぶと（式 (40) では p_1 段に 0 番で示した），式 (40) の 26 項のうち，α_1 を含む 15 項を計算して，途中結果を配列 p に残して帰ってくる．次に（式 (40) では p_1 段に 16-1 番で示した），α_2 を sp に入れて呼ぶと，網掛けで示した 16 から 22 番目の 7 項を計算して，配列 p は更新される．次に（式 (40) では p_1 段に 23-1 番で示した），α_3 を sp に入れて呼ぶと，23 から 25 番目の 3 項を計算して，配列 p は更新される．最後に（式 (40) では p_1 段に 26-1 番で示した），α_4 を sp に入れて呼ぶと，網掛けで示した 26 番目の 1 項を計算して，p_2 から p_5 は完成する．

関数呼出しでは，**プロシージャフレーム（アクティベーション・レコードとも呼ばれる）に引数リスト（リターンアドレスを含む）が置かれ，これがスタック領域に積まれてゆく**ので，何回同じ関数が呼び出されも引数リストは保存されている[66]．再帰呼出しから戻ると，“<----” で示した if ステートメントに制御が来る[67]．このとき，前回呼ばれたときの仮引数（パラメータ）mulvieta(123,e(1)*e(2)*e(3),5,4,e,p,last,1) の sp$=\alpha_1\alpha_2\alpha_3$ も得られる．ここで dgt1$<n$ なので，最後の引数 dir$=0$ で呼び出し，階層を右へ行く．

[66] もし引数リストが静的なプログラム領域に置かれると，同じ関数が 2 回呼ばれると，2 回目の呼出しで最初の引数リストを上書きするので，前回の引数もリターンアドレスも消される．

[67] 再帰呼び出しでプロシージャフレームに引数情報が置かれ，プロシージャフレームがスタックに積まれてゆく構造について，授業などで習っていない人は教科書を読むことをお勧めする [9]．また，再帰呼び出しとスタックに積まれた引数の状態に不慣れな人は，付録の HanoiStackMR.BAS プログラムを,「ステップ実行」で実行してみることをお勧めする．

この関数は最初の呼出しで，全係数を途中まで作成するので，整数性判定も含める．整数性を p_n が完成したときに判定して，整数でなければその先の計算を行わない．`mulvieta` 関数を次の `vietatermschk` 関数が呼出すことで，n 個の多項式係数が整数で，rational 型の配列に得られる．

vietatermschk

```
int vietatermschk(const vector<interval>& e, const int n, rational_vector& cor) {
   int i,r,inty,ok=1;
   vector<int> last(n+1), num(n+2);
   vector<interval> p(n+1);              // コンストラクタがゼロに初期化する
   for(i=1; i<n; i++){
      num[2]=i;
      for(int j=1; j<n; j++) last[j]=0;
      inty=mulvieta(num,e[i],n,2,e,p,last,1); // p_2 から p_n が完成
      if(inty==0){ ok=0; break;}
   }
   if(ok==1){
      interval t=e[1];
      for(r=2; r<=n; r++){ interval er=(e[r]);  t=t+er;  }  // 1 次項の総和計算
      if((t.upper()).floor() == (t.lower()).ceil()){        // p_1 の整数性チェック
         cor[1]=-(t.upper().floor());                        // 整数の係数を得る
      }else{
         ok=0;
      }
   }
   if(ok==1){
      for(r=2; r<=n; r++){
         if(p[r].upper().floor() == p[r].lower().ceil()){ // 整数性を満たしていれば
            if(r%2 == 1){                       // 奇数次なら
               cor[r]=-p[r].upper().floor(); // 整数の係数を取出し負の符号をつける
            }else{                              // 偶数次なら
               cor[r]=p[r].upper().floor();  // 整数の係数を取出す
            }
         }else{
            ok=0; break;
         }
      }
   }
   return ok;
}
```

C や C++はローカル変数や引数リストはプロシージャフレームに積むので，再帰呼び出しが稼働する.「有理数計算プログラミング環境」では，rational 型の変数は，桁数の増加とともに動的に配列を拡張するが，この配列そのものは 2 重にポイントされ，スタックに積まれるのは最初のポインタだけで，配列の本体は静的な領域に獲得されるので,（参照渡しでは）再帰呼び出し可能である．

計算時間を実行結果の桁数から考察する．5 次の因子を作る “s 10” の $n = 100$ の問題から例を示す．ヘッセンベルグ変換後の H_{51} から（dvsrch.cpp と同様）既知の因子で割られて 10 次の特性多項式になる．10 個の近似固有値を区間に入れて，5 次多項式用の精度に改良する．見付けられる因子に関係する近似固有値の区間幅と区間を示す.

```
bisect(interval) converged it=50 b-a=3.077097586687155e-76
     583767772594036.526309582779885448 (47 digits) 8436625927967713587
     583767772594036.526309582779885448 (47 digits) 8436625928275423346
bisect(interval) converged it=53 b-a=1.000150533063272e-80
     4973981023983810.402897873009471205 (51 digits) 9607109244384125545
     4973981023983810.402897873009471205 (51 digits) 9607109244484140599
bisect(interval) converged it=54 b-a=9.708652095377912e-83
     12360545839211637.931305869131004270 (54 digits) 9754735196749000320
     12360545839211637.931305869131004270 (54 digits) 9754735197719865529
```

```
 bisect(interval) converged it=56 b-a=5.006853152565665e-84
      20398280080415685.794626887998834475 (55 digits) 8585394269792407540
      20398280080415685.794626887998834475 (55 digits) 8585394270293092855
 bisect(interval) converged it=56 b-a=1.628300893324702e-84
      26535259917929975.344859787080804599 (55 digits) 9968771758050478038
      26535259917929975.344859787080804599 (55 digits) 9968771758213308127
```

これらの区間幅が 10^{-80} 程度の近似固有値の区間から，多項式係数 p_1 から p_5 を生成する．上記の `vietatermschk` 関数では，1 次の係数は変数 `t` に，2 次以降は変数 `p[r]` に現れる数値で，これを `formatprn(p[r])` によって表示すると，実際に計算した桁数も分かる．

```
first order term=-6.485183463413514e+16
       64851834634135145.999999999999999999 (47 digits) 9999999999907331953
       64851834634135146.000000000000000000 (47 digits) 215051817
2-th term=1.453843120389752e+33
     1453843120389751897397549617140207.999999999999999999 (30 digits) 9999999999940444156
     1453843120389751897397549617140208.000000000000000000 (30 digits) 138209312
3-th term=-1.309505467048489e+49
1309505467048 (24 digits) 5683597576439.999999999999999999999999999999999999999999986875:
1309505467048 (24 digits) 5683597576440.000000000000000000000000000000000000000000030458
4-th term=4.043992000072597e+64
4043992000072 (39 digits) 9995286931439.999999999999999999999999999988631:
4043992000072 (39 digits) 9995286931440.000000000000000000000000000026383
5-th term=-1.94266889222573e+79
1942668892225 (54 digits) 3322522936223.999999999996916:
1942668892225 (54 digits) 3322522936224.000000000007156
```

1 次の係数では精度は充分に余裕があるが，5 次では 10 桁の余裕になっている．この interval の数を `cor[r]=p[r].upper().f` によって整数として，次の 5 次多項式が得られる．

```
+                            1.*x^{5}
           -64851834634135146.*x^{4}
+14538431 (18 digits) 17140208.*x^{3}
-13095054 (35 digits) 7576440.*x^{2}
+40439920 (49 digits) 86931440.*x^{1}
-19426688 (65 digits) 2936224.*x^{0}
```

因子多項式の桁数は，p_1 は 17 桁，p_2 は 34 桁，p_3 から p_5 までは，50 桁，65 桁，80 桁と，約 15 桁ずつ増加しているが，区間演算では整数性の判定によってこの桁を正確に得るためにそれ以上の桁数を計算している．

s 桁と t 桁（$s < t$）の**乗算の計算時間**は st に比例すると見積もる．ここでは，多項式係数を p_1 から p_r まで素朴な順序で計算した場合で見積もる．この例の p_5 の場合，4 回の乗算で，$s^2 + 2s^2 + 3s^2 + 4s^2$ となる．一般的に k 項では $\dfrac{k(k-1)s^2}{2}$ の乗算の計算時間になる．p_r は ${}_nC_r$ 項の総和なので，

$$
{}_nC_r \frac{r(r-1)s^2}{2} \tag{41}
$$

の乗算の計算時間になる．vietaterm で $n-1$ 個の係数を求めると，

$$
\sum_{r=2}^{n} {}_nC_r \frac{r(r-1)s^2}{2} \tag{42}
$$

の乗算の計算時間にになる．dvsrch1.cpp では，精度改良を行うので，近似固有値の最初の精度 s は 80 桁とする．組合せの数が指数時間だが，それに桁数の増大が拍車をかけるので，n が大きくなり，次数が高くなると計算不能になる．言い換えると，適当な次数で因子多項式が見つかる場合でなければ,（計算時間の壁に当たって）因子探しはできない．

乗算回数を `vietatermschk` 関数で削減すると，大きな効果がある．“s 10” の計算の場合，`vietatermschk` 関数で計算順序を最適化した場合，最初の 15 項の計算には，2^{32} 進数で，有理数の分子の桁数の積の総和は 1029 桁であった．次の 22 項目までで 1457 桁，25 項目までで 1616 桁，26 項目までで 1663 桁であった．

“G 10” では，`#define FACTORIZEONLY` を指定しないと，$n = 96$ の行列から H_{23} の小行列が分離されるが，この小行列の特性多項式は分解されないので，23 個の近似固有値から 23 個の因子多項式の係数を作成する部分が計算時間の多くを占める．ここでは，約 800 万項を計算し，桁数は 5 億に及ぶ．ただしもとの行列要素の係数が整数なので，桁数は大きくない．この計算を，元の素朴な順序で計算すると，ジョブ全体で 747 秒かかるが，計算順序を最適化した `vietatermschk` を用いると 156 秒に短縮された．このように，乗算回数を `vietatermschk` 関数で削減すると，次数の多い問題では大きな効果がある．

4.2 計算例

実行時の入力データは，コマンド行引数に “T 4” のように英文字で行列の種類を，数字で分割数を指定する．T が dvsrch と同じ熱伝導行列，小文字の t が 10 進数の 10 で割った熱伝導率，s が 2 進数の 10 で割った熱伝導率を使用する熱伝導行列を指定する．G がグラフ・ラプラシアン行列を指定する．

4.2.1 s を指定した場合の計算例

“s m” を指定すると，因数分解の因子多項式の次数の形は dvsrch.cpp の場合と同じになる．したがって，小行列のサイズは 18 ページの表 4 に一致する．多重度 m が現れるので，H_1 が $m-2$ 個分離される．$n = m^2$ なので，次数 $m^2 - m + 2$ が 2 つの小行列に分かれる．

“s 10” 指定： $n = 100$ の行列 A が H_{51}, H_{41} と 8 つの H_1 に分割される．H_{41} の特性多項式は H_1 の特性多項式で割り切れて 40 次式になり，8 つの 5 次式で割り切れる．${}_{40}C_5 = 658,008$ で，割り切れる組合せは，66,903 番目，36,980 番目，18,032 番目，5,680 番目，624 番目に現れる．H_{51} の特性多項式は H_{41} の特性多項式で割り切れて 10 次式になり，2 つの 5 次式で割り切れる．次数 40 を相手に，5 次多項式で割ってゆくので，“s 11” の場合よりも組合せの数が多く，また 5 次式なので近似固有値の精度も高くなり，桁数の多い計算をしなけらばならない．計算時間を比較すると，11 の場合よりも 5 倍以上を要する．なお，`#define INTYDOUBLE` を指定しないで `vietatermschk` の効果を見ると，22.7 秒の計算時間（`#define FACTORIZEONLY` も指定しない）が，26.3 秒に伸びた．

“s 11” 指定： $n = 121$ の行列 A が，$H_{55}, H_{37}, H_{15}, H_7$ と 7 つの H_1 に分割される．H_7 の特性多項式は，既知の 1 次，2 次，4 次式で割り切れる．H_{15} の特性多項式は H_7 の特性多項式で割り切れて 8 次式になり，2 つの 1 次式と 3 つの 2 次式で割り切れる．H_{37} の特性多項式は既知の 1 次，2 次，4 次多項式因子で割り切れて 30 次式になり，3 つの 2 次式と 6 つの 4 次式で割り切れる．${}_{30}C_2 = 435$ で，割り切れる組合せは，131 番目，12 番目に現れる．${}_{24}C_3 = 2,024$，${}_{24}C_4 = 10,626$ で，割り切れる組合せは，1587 番目，733 番目，383 番目，63 番目に現れる．H_{55} の特性多項式は既知の 3 つの 1 次，7 つの 2 次，6 つの 4 次多項式因子で割り切れて 10 次式になり，2 つの 1 次式と 2 つの 2 次式と 4 次式で割り切れる．この場合は，特性多項式の次数は高いが，実際の除算は 4 次式までで済むので，計算時間も短い．実行結果を hoge に作成して “grep ii= hoge” で見ると，何次の因子が何番目に見つかったかが分かる．

```
ii=8 Divisor = +                            1.*x^{2}
ii=1 Divisor = +                            1.*x^{4}
ii=3 Divisor = +                            1.*x^{2}
ii=2 Divisor = +                            1.*x^{2}
ii=1 Divisor = +                            1.*x^{2}
ii=131 Divisor = +                            1.*x^{2}
ii=12 Divisor = +                            1.*x^{2}
ii=122 Divisor = +                            1.*x^{2}
ii=1587 Divisor = +                            1.*x^{4}
ii=733 Divisor = +                            1.*x^{4}
ii=383 Divisor = +                            1.*x^{4}
```

```
ii=63 Divisor = +                              1.*x^{4}
ii=9 Divisor = +                              1.*x^{4}
ii=1 Divisor = +                              1.*x^{4}
ii=12 Divisor = +                              1.*x^{2}
ii=3 Divisor = +                              1.*x^{2}
ii=1 Divisor = +                              1.*x^{4}
```

4.2.2 G を指定した場合の計算例

グラフ：ラプラシアン行列に対する計算結果について述べる．

"G 9" 指定：H_{42}，H_{32} と 3 つの H_1 に分かれ，H_{32} は既知の因子で割りきれない．32 次の特性多項式は 5 次，9 次，18 次の多項式に分解される．${}_{32}C_5 = 201,376$ で，因子は組合せが 111,182 番目である．${}_{27}C_6 = 296,010$，${}_{27}C_7 = 888,030$，${}_{27}C_8 = 2,220,075$，${}_{27}C_9 = 4,686,825$ で，組み合わせが 2,264,208 番目である．H_{42} は 1 次の因子 $(1-\lambda)$ と H_{32} の因子で割り切れて 9 次となり，これが 1 次（λ），4 次，4 次と分解される．このように，熱伝導行列の場合に比較すると，因子多項式の見つかるまでの反復回数が多い．

"G 10" 指定：H_{52}，H_{23}，H_{18} と 3 つの H_1 に分かれる．H_{18} は 2 つの 9 次式に分解されるが，H_{23} は分解されない．H_{52} は 1 次の因子 $(1-\lambda)$ と H_{18} の 2 つの 9 次式と H_{23} の因子で割り切れて 10 次となり，これが 1 次（λ），4 次，5 次と分解される．そのため，計算時間の大部分は 23 次多項式の係数の作成にかかる．この計算で，近似固有値の精度を高めるところを $e_2 = 0.08284674618487071$ に着目して見てみる．はじめは区間幅は 10^{-10} 程度であるが，

```
0.082846746184871:
0.082846746267718
            |
```

23 次多項式の係数の整数性を求めるためには 10^{-13} 程度にまでの精度改良で済んでいる．

```
0.082846746184871131:
0.082846746184911584
                |
```

#define FACTORIZEONLY を指定しないで，H_{23} の 23 次の因子多項式係数を作る vietatermschk 関数に大部分の計算時間が費やされる．なお，23 次多項式の係数を作成する時間は，実測で，148 秒であった．

"G 11" 指定：H_{63}，H_{42}，H_9 と 3 つの H_1 に分かれる．H_9 は分解されない．H_{42} は 14 次と 28 次に分解される．H_{63} は 1 次の因子 $(1-\lambda)$ と 9 次式，14 次式，28 次式の因子で割り切れて 11 次となり，これが 2 つの 1 次式（ひとつは零固有値），4 次，5 次と分解される．

なお，#define FACTORIZEONLY を指定しないと，H_{42} の 28 次の因子多項式係数を作る vietatermschk 関数に大部分の計算時間が費やされる．この時間は実測で 5923 秒であった．

グラフラプラシアン行列の場合，熱伝導行列のように，適当な次数の因子多項式が存在しないので，計算時間の大部分は，次数の大きな特性多項式の係数を作成する vietatermschk に集中する．

5 倍精度浮動小数点行列の解析プログラム dvsrch2.cpp

2 次元トラス構造解析プログラム CT2D から抽出した，倍精度浮動小数点演算で作成された対称行列の固有値の多重度を解析する．前章の行列との違いは，**ヘッセンベルグ小行列から得られる特性方程式が病的に近接した根を持つ**点にある．前章の行列の多重度の問題は，ヘッセンベルグ変換で解決され，小行列の特性方程式は近接根はなかった．本章で扱う CT2D の行列は，ヘッセンベルグ変換後の小行列の特性方程式が，1 つのグループに複数の根をもつ難しさがある．dvsrch1.cpp では近似固有値を interval 型の変数で挟み込み，この区間を縮小することで，根と係数の関係を使用できるところまで精度改良すれば，因子多項式を見つけられることを確かめた．これは有理算術演算で無理数を扱う第 1 歩である．dvsrch2.cpp ではさらに次の項目を試みる．

- 精度改良のための包含では，関数値を使うだけでは分離が難しく，導関数も使用して近接根を分離する．
- 「有理数計算プログラミング環境」が，倍精度浮動小数点演算以外に，4 倍精度，8 倍精度といった多倍精度演算をサポートすべきかどうかを評価する．

dvsrch2.cpp では，CT2D が生成した倍精度浮動小数点行列の解析を行う．

第 1 節で，浮動小数点演算による倍精度の行列をテキストファイル経由で読み込む方法を説明する．この行列のヘッセンベルグ変換とフロベニウス変換は計算時間がかかるので，第 2 節で，チェックポイントとリスタートについて触れる．第 3 節で，1 グループに最大で 3 つの根をもつ特性多項式の包含方法を説明する．第 4 節で計算例を示すが，浮動小数点行列が，コンパイルオプションの違いによって固有値の多重度に影響を与える現象を解明する．

5.1 倍精度浮動小数点数行列の読込み

浮動小数点計算の途中の行列を抽出して有理数変換して正確な計算を実行することが「有理数計算プログラミング環境」の目的である．抽出された行列要素の下位の桁は，浮動小数点計算のさまざまな条件の組合せで変化する．これらの条件による摂動を分析することで，誤差の分布の原因を探す．浮動小数点計算プログラムとして，構造解析プログラム CT2D（C program for Truss 2 Dimensional analysis）を使用する．CT2D は C 言語で書かれた単純なプログラムで，入手も可能（ダウンロード可能）である [7, p. 68]．

剛性行列 K と質量行列 M を CT2D で作成する．ただし質量行列は，CT2D プログラムに機能追加して，固有振動モードの解析を可能にして作成した．solveSTATIC.c を変更する形で NormalMode.c プログラムを作成し，このプログラムの NormalMode 関数が massmat 関数（NormalMode.c 中）を呼び出し，そこで質量行列を生成する．質量行列 M は，トラス要素の質量（断面積，密度，部材長の積）を両節点に等配分する集中質量行列（対角行列）とした．したがって 1 次元配列に格納される．さらに，NormalMode は，eigv.c プログラムの genamat 関数を呼び出す．genamat は n と 行列 K と M と $A = S^{-1}KS^{-1}$ を，非零要素だけを，行番号，列番号，行列要素の組でファイルに出力する．その後，jacobev 関数で固有値と固有ベクトルを求め，固有値はソートする[68]．

調和振動（定常解）を表す 2 階の微分方程式 $M\ddot{\boldsymbol{u}}(t) + K\boldsymbol{u}(t) = 0$ は一般化固有値問題

$$\omega^2 M\boldsymbol{u} = K\boldsymbol{u}, \tag{43}$$

となる[69]．M は対角行列なので，M の対角項の平方根を対角項とする対角行列 S によって

$$A = S^{-1}KS^{-1}, \quad \boldsymbol{x} = S\boldsymbol{u}, \tag{44}$$

[68] `jacobev` は dvsrch.cpp などに入れた `jacob2` に固有ベクトルの計算を加えた．ヤコビ法の説明を付録に記した．

[69] 調和振動では $\boldsymbol{u}(t) = \boldsymbol{u}\cos\omega t$ のようにおくことができる（$\boldsymbol{u}$ は固有振動モードの振幅）．

と対称性を保存したまま固有方程式 (7) に変形できる[70]．

なお dvsrch2.cpp では行列は次の 3 通りを解析できる（後述するコマンド行入力データの mattype に 0，1，2 のいずれかを指定する）．

- $A = S^{-1}KS^{-1}$ を倍精度で計算してから有理数変換する（$\texttt{mattype} = 0$）．
- $A = K$ として有理数変換する（$\texttt{mattype} = 1$）．
- K と M を有理数変換してから，$A = M^{-1}K$ を有理数計算する（$\texttt{mattype} = 2$）．

5.1.1 浮動小数点行列のファイルへの出力

例題集の CT2Dnorm ディレクトリに，トラスデータ作成プログラム trusgenorm.f があり[71]，gfortran でコンパイルして nx と ny に 2 を与えると，最小の 13 節点のデータが作成される．これを S13Free.dat とする．2 つのコマンド "gcc -c *.c" と "gcc -lm *.o" で作成した a.exe を実行すれば，S13free.txt が作成される．**ファイル名は CT2D のメインプログラム Clientmain.c に埋め込んだ**ので，コンパイルオプションを変更する場合は，その都度，Clientmain.c を変更して使用する．

IEEE 標準の倍精度浮動小数点数は，書式 "%25.17le" を指定したテキストファイル経由で渡す．10 進数で 17 桁の有効桁数は，IEEE 倍精度浮動小数点数を正確に復元できる [10]．CT2D の eigv.c の genamat 関数を示す．

```
void genamat(double * mvec, double *b, double *a, int n, int nb, FILE *fp_mat){
   int i, j, j1;
   double scal;
   for(j = 0; j < n ; j++){
      for(i = 0; i < n; i++) a[j*n+i]=0;
   }
   for(j = 1; j <= n ; j++){
      j1=MAX(j-nb+1,1);
      for(i=j1; i<=j; i++) a[(j-1)*n+i-1]=b[j*nb-j+i-1];
   }
/*** fill lower portion by transpose   ***/
   for(j = 0; j < n ; j++){
      for(i=j+1; i<n; i++) a[j*n+i] = a[i*n+j];
   }
/*** file output n and K matrix  ***/
   fprintf(fp_mat,"%d\n",n);
   for(i=0; i<n; i++){
      for (j=0; j<n; j++){
         if(a[n*j+i] != 0){;
            fprintf(fp_mat," %d",i);
            fprintf(fp_mat," %d",j);
            fprintf(fp_mat,"%25.17le\n",a[n*j+i]);
   }  }  }
/*** file output M matrix  ***/
   for (j=0; j<n; j++) fprintf(fp_mat,"%25.17le",mvec[j]);
   fprintf(fp_mat,"\n");
   for(j = 0; j < n ; j++){
      scal = 1.0 / sqrt(mvec[j]);
      for(i=0; i<n; i++){
         a[j*n+i] = scal * a[j*n+i]; a[i*n+j] = scal * a[i*n+j];
   }  }
/*** file output A matrix  ***/
   for(i=0; i<n; i++){
      for (j=0; j<n; j++){
```

[70] $M = S^2$ とおき，$K\boldsymbol{u} = \lambda SS\boldsymbol{u}$ の両辺に S^{-1} を掛けて $\underbrace{S^{-1}KS^{-1}}_{A}(\underbrace{S\boldsymbol{u}}_{\boldsymbol{x}}) = \lambda(\underbrace{S\boldsymbol{u}}_{\boldsymbol{x}})$ と変形し $\boldsymbol{x} = S\boldsymbol{u}$ とおくことで固有方程式 $A\boldsymbol{x} = \lambda\boldsymbol{x}$ が得られる．

[71] このデータ生成プログラムは，外周にない垂直部材（節点 2 と 7，7 と 12）と水平部材（節点 6 と 7，7 と 8）は 2 重になっている．拘束条件がないので，3 つの剛体モードと，上方向と横方向の対称性のために，複数組の重根をもつ．

```
        if(a[n*j+i] != 0){;
           fprintf(fp_mat," %d",i);
           fprintf(fp_mat," %d",j);
           fprintf(fp_mat,"%25.17le\n",a[n*j+i]);
  } } } }
```

ファイル名は，例題集の中では S13Free.txt とした.

5.1.2　浮動小数点行列のファイルからの入力

対応する C++ ルーチンでの読込みは，まず自由度 n を読み込む.

dvsrch2.cpp の CT2D ファイルの n の読込み

```
   strcpy(matfile,matname);  // required #include <string.h>
   strcat(matfile, ".txt");
   cout << "Reading CT2D output matfile =" << matfile << endl;
   fin.open(matfile);
   if(!fin){
         cout << "cannot open input file\n";
         exit(EXIT_FAILURE);
   }
   fin >> n;
   std::cout << "dvsrch2 n=" << n << std::endl;
```

$n \times n$ の 2 次元配列を定義する.

dvsrch2.cpp の読込み領域の定義

```
   matrix<double> a(n,n);
```

1 次元配列 ka，ma，aa に C の 1 次元配列の倍精度データを読み込む．非零要素のみを読み込むので，配列を零クリアして，行と列の番号に従って行列要素を読み込む.

```
/***************  read matrix from file made by CT2D program *************/
   matrix<double> ak(n,n);
   rational_matrix kar(n+1,n+1);
   ka = new double[n*n];
   ma = new double[n];
   aa = new double[n*n];
   for (j=0; j<n; j++){
      for (i=0; i<n; i++){ aa[j*n+i]  = 0; ka[j*n+i]  = 0; }
      ma[j]  = 0;
   }
   std::cout << " Now read file=" << matfile << std::endl;
   for (;;){
      fin >> i; fin >> j; fin >> tmp;
      ka[n*j+i]=tmp;
      if(i==n-1 && j==n-1) break;
   }
   for (j=0; j<n; j++){
      fin >> tmp;
      ma[j]=tmp;
   }
   for (;;){
      fin >> i; fin >> j; fin >> tmp;
      aa[n*j+i]=tmp;
      if(i==n-1 && j==n-1) break;
   }
   fin.close();
   std::cout << "dvsrch2 read file end" << " n*n=" << n*n << std::endl;
```

1 次元配列から 2 次元配列に移す.

dvsrch2.cpp の行列の形式の変換

```
/******** format conversion from C type one dimensional array to matrix format *********/
   for (j=0; j < n; ++j) {
      for (i=0; i < n; ++i){ a[i][j] = aa[j*n+i]; }
   }
```

倍精度から有理数に変換する．dvsrch2 の実行時の指定で，A か K か $M^{-1}K$ かを選択する．

dvsrch2.cpp の行列の型変換

```
/***********  convert matrix from double to rational ****************/
   cout << "matk=" << matk << " mattype=" << mattype  << endl;
   printf("mattype=%x\n",mattype);
   printf("matk=%s\n",matk);
   if(mattype == 2){          //MinvK
      cout << " MinvK selected for A matrix" << endl;
      for (j=0; j < n; ++j) {
         for (i=0; i < n; ++i){ ak[i][j] = ka[j*n+i]; }
      }
      cnvmat(ak,kar,n,n);    // kar(n,n) <-- K matrix rational
      for (i=0; i < n; ++i) {
         rational mforinv = Rdset(&ma[i]);
         for (j=0; j < n; ++j){  h[i][j] = kar[i][j] / mforinv; }
      }                        // h(n,n) <-- M^{-1}K matrix rational
   }else if(mattype == 1){
      cout << " K selected for A matrix" << endl;
      for (j=0; j < n; ++j) {
         for (i=0; i < n; ++i){ a[i][j] = ka[j*n+i]; }
      }
      cnvmat(a,h,n,n);     // h(n,n) <-- K matrix rational
   }else{
      cout << " SinvKSinv selected for A matrix" << endl;
      cnvmat(a,h,n,n);    // h(n,n) <-- S^{-1}KS^{-1} matrix rational
   }
   cout << n << "x" << n << " A matrix" << endl;
```

有理数の行列になれば，固有値の計算は熱伝導行列やグラフ・ラプラシアン行列のように計算できるはずである．

5.2　チェックポイントとリスタート

CT2D の構造解析行列は，dvsrch1.cpp の行列よりも，ヘッセンベルグ変換とフロベニウス変換に計算時間がかかる．これはヘッセンベルグ行列の `MatDgtCount` から得られる平均の桁数が多い（234 に及ぶ，グラフ・ラプラシアン “G 9” は 7.3 “G 10” は 9.2 ）ことからも分かる．そこで変換された行列のチェックポイントをとり，リスタート可能とした．チェックポイント・ファイル名は，入力行列のファイル名が S13free.txt の場合は，$S^{-1}KS^{-1}$ の場合は S13freeA.chk に，K の場合は S13freeK.chk に，$M^{-1}K$ の場合は S13freeMinvK.chk になる．

```
 if(chkres==0){
    rblas::elmhes(ar,n);  /***   ELMHES  *******************/
    // ----- checkpoint H
    std::stringstream ofn;
    if(mattype==0){
       ofn << matname << "A" << ".chk";
    }else if(mattype==1){
       ofn << matname << "K" << ".chk";
    }else{
       ofn << matname << "MinvK" << ".chk";
    }
    ofs.open(ofn.str().c_str(),std::fstream::trunc);
    if (!ofs) {
       throw std::runtime_error("Write open failed for " + ofn.str());
    }
    oa << ar;
 }else{                          //   read H from checkpoint file
```

```
    std::stringstream ifn;
    if(mattype==0){
       ifn << matname << "A" << ".chk";
    }else if(mattype==1){
       ifn << matname << "K" << ".chk";
    }else{
       ifn << matname << "MinvK" << ".chk";
    }
    ifs.open(ifn.str().c_str(),std::ofstream::in);
    if (!ifs) {
       throw std::runtime_error("Read open failed for " + ifn.str());
    }
    ia >> ar;
    std::cout << "Read Hessenberg matrix from checkpoint file" << std::endl;
 }
```

フロベニウス行列についてのプログラムの説明は，ヘッセンベルグ行列と同様なので省略する．チェックポイント・ファイル名は，入力行列のファイル名が S13free.txt の場合は，$S^{-1}KS^{-1}$ の場合は S13freeA2.chk に，K の場合は S13freeK2.chk に，$M^{-1}K$ の場合は S13freeMinvK2.chk になる．**フロベニウス行列の実体は特性多項式なので，ファイルサイズがヘッセンベルグ行列に比較してはるかに小さい**．したがってダウンロードする tar ファイルには S13freeA2.chk，S13freeO3A2.chk などを含めた．

変数 chkres に 0 か 1 か 2 を与える仕様にした．1 の場合は H を読込んで計算するので，ヘッセンベルグ変換はスキップし，フロベニウス変換から計算する．2 の場合は F を読込んで計算するので，ヘッセンベルグ変換もフロベニウス変換もスキップして，因子探索を計算する．

5.3 近接根の分離

1 つのグループに複数の近似固有値が存在し，ヘッセンベルグ変換後の小行列でも近接固有値の状態で存在する場合は，分離が困難になる．グループ化は相対誤差 $\varepsilon = 10^{-9}$ で行っている（selectev）．区間 (a, b) に 2 根が存在すると，$f(a)f(b) > 0$ になる．これを分離するには接線の傾き $f'(a)$ と $f'(b)$ を使用して，極値（導関数 $f'(x)$ の零点）を含む区間を求める．極値を 2 分法で探すことで，近接する 2 根の間に存在する点 c を求めれば，区間 (a, b) を $f(a)f(c) < 0$ と $f(c)f(b) < 0$ によって区間 (a, c) と (c, b) に分解できる．3 根が存在する場合は曲率 $f''(a)$ と $f''(b)$ を判定に使用して，変曲点（2 階導関数 $f''(x)$ の零点）を含む区間を探すことで区間を分解する．分解された区間で 2 根が存在する区間は，2 根を求める問題に帰着する．

なお，ここで説明するアルゴリズムは，対称行列の固有値問題であるから，実数解が n 個存在することが保証されている問題用である．また，ヘッセンベルグ変換で重根の問題を解決しているので，重根が存在しないことも保証されている．虚数解がある場合や，多重度の高い根がある場合は，特性多項式の関数や導関数の符号だけでは，本節のアルゴリズムは機能しない．

5.3.1 近接する固有値の分離（2 根の場合）

1 つのグループに最大で 3 根が存在する場合までを考慮した．各グループごとに chkinc 関数と countinc 関数によって包含を成立させる．chkinc 関数は，倍精度で得られた近似固有値を有理数変換して，グループの上限と下限を設定する．引数 mult[i] は，27 ページの 3.1 節の「処理の概要」に示したグループ化を行う selectev 関数で得た，各グループに属する根の数マイナス 1 である（引数の am は固有値の平均の値で，dvsrch2.cpp では selectev 関数の引数に加えた）．countinc 関数は，近似固有値 eigvl[i] と eigvh[i] から示唆された数の根を包含して，それらの区間を，それぞれ上限用の btemp と下限用の atemp 配列に返し，成立した包含の数を返す．呼び側は制御変数 nexist をインクリメントして，包含の成立した数を得る．なお，零近傍の固有値に対しては，近似固有値に対し，特別な処理を必要とする（後述）

dvsrch2.cpp の上限と下限のセットと包含カウント

```
nexist=0;
   for(i=1; i<=ngrp1;i++){
      rational tempa,tempb;
      int idummy, nmult, maxit = 53;
      double eigvli=eigvl[i], eigvhi=eigvh[i];
      if(fabs((eigvli+eigvhi)/am) < eps){  // eigenvalue close to zero
         if(eigvli*eigvhi <= 0){           // 上限と下限で符号が異なる場合
            eigvli=eigvli*2; eigvhi=eigvhi*2;     それぞれを 2 倍し
         }else{                            //  同符号の場合
            eigvli=eigvli-fabs(eigvli)*2;         下限から下限の絶対値の 2 倍を引き
            eigvhi=eigvhi+fabs(eigvhi)*2;         上限に上限の絶対値の 2 倍を加える
         }
      }
      idummy = chkinc(nsize,p,eigvli,eigvhi,mult[i],am,tempa,tempb);
      nmult = countinc(nsize,p,tempa,tempb,mult[i],epsint,maxit,abtemp);
      for(j=0; j<nmult; j++){
         nexist ++; eigint[nexist]=abtemp[j]; included[nexist]=1;
         eigvr[nexist]=(abtemp[j].lower()+abtemp[j].upper())/rational::TWO;
      }
}
std::cout << std::endl << "%FOUND " << nexist << " intervals.  nsize=" << nsize << std::endl;
```

countinc 関数は，大きく複雑な関数であり，全体の構成は後述する．区間に最大 2 根が存在する mult==1 の場合，まず a と b での関数値 $f(a)$ と $f(b)$，傾き $f'(a)$ と $f'(b)$，曲率 $f''(a)$ と $f''(b)$ を Horner3 関数で求める．図 6 の左に，区間 (a, b) に 2 根が存在する場合を示した[72]．図には極値の位置と導関数を点線で示した．

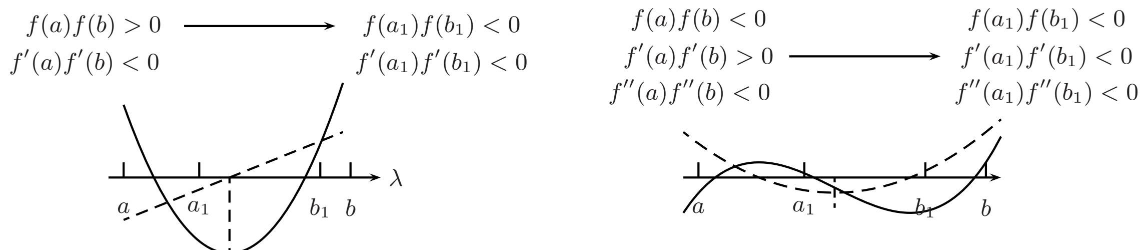


図 6: bisect2 による (a, b) から (a_1, b_1) の絞り込み（左）と bisect3 による (a, b) から (a_1, b_1) の絞り込み（右）

プログラムは mult=1 によって「2 根あるかもしれない」と認識している．「$f(a)f(b) > 0$ かつ $f'(a)f'(b) < 0$」が 2 根ある場合で，このとき極値（$f'(x)$ の零点）を含む区間を bisect2 関数で，反復回数を十分大きくとって探す．

countinc 関数の導関数の零点を含み両端で関数値が異符号の区間探索

```
      if(fa*fb > rational::ZERO && fda*fdb <= rational::ZERO){
         rtnval = bisect2(p,n,maxit+100,a,fa,fda,b,fb,fdb);
         if(rtnval<=0) {
            cout << "countinc return From bisect2 rtnval=" << rtnval << " return rtnval" << endl;
            return rtnval;
         }
      }
```

bisect2 関数は，(a, b) 間に 2 根ある場合，$f'(x)$ は (a, b) 間に 1 つの零点をもつので 2 分法でこれを探す．**区間両端での導関数の値が異符号の状態を保ち，関数値が異符号になると収束**する．図示したように，極値を含み，$f'(a_1)f'(b_1) \leq 0$ でかつ $f(a_1)f(b_1) < 0$ を満たす区間が bisect2 によって得られる．

[72] 左図のグラフは PostScript のスタック型（後置記法）の言語で “x x mul x sub” で記述している．関数は $x^2 - x$ である．右図のグラフは “x x x mul mul 3 div x x mul 2 div sub 0.05 add” で記述している．関数は $\frac{1}{3}x^3 - \frac{1}{2}x^2 + 0.05$ である．

極値を含み両端で関数値が異符号の区間が見つかったら，区間端点の値と関数値を保存してから，軽く（数回）`bisect` 関数で反復して包含を確認する（1 次式の因子探しに必要な精度での精度改良は後に行う）．この区間の外側と，初めに与えられた区間端との間（$[a, a_1]$ か $[b_1, b]$）にもう 1 つの根があるので，2 個めの包含を探す．`bisect2` 関数の役割は，図 6 の左の上の 2 つの不等式の間の矢印で示したように，**両端で関数値が同符号の状態から異符号の状態に遷移**させるところにある．

5.3.2　近接する固有値の分離（3 根の場合）

図 6 の右に，区間 (a, b) に 3 根が存在する場合を示した．図には変曲点の位置と導関数を点線で示した．プログラムは `mult=2` によって「3 根あるかもしれない」と認識している．$f'(a)f'(b) > 0$ で，$f''(a)f''(b) < 0$ の場合は `bisect3` を呼び，変曲点（$f''(x)$ の零点）を探す．`bisect3` は，両端で曲率が異符号の状態を保ち，**関数の変曲点を含み，両端で傾きが異符号になると収束**する．

`countinc` 関数が `bisect3` によって (a, b) の内側に，変曲点を含み両端で傾きが異符号の区間 (a_1, b_1) を得たとき，関数値 $f(a_1)$ と $f(b_1)$ の**符号の組合せは** $(+, +)$**，**$(+, -)$**，**$(-, +)$**，**$(-, -)$ **の 4 通り**ある．ここでは作画の都合上 $f(a) > 0$ として説明するが，$f(a) < 0$ の場合は符号を反転して考えればよいので，根を含む区間を絞り込むアルゴリズムとしては $f(a) < 0$ でも同じであり，一般性は失われない．以下，$f(a)$，$f(a_1)$，$f(b_1)$，$f(b)$ の符号に着目して，3 根を含む区間から 1 根のみを含む 3 つの区間を分離する方法を述べる．

$(+, -, +, -)$ **の場合**　図 7 に $\mathrm{sign}(f(a), f(a_1), f(b_1), f(b)) = (+, -, +, -)$ の場合を示し，変曲点の位置を点線で入れた．

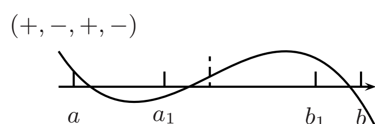


図 7: bisect3 による (a_1, b_1) と符号 $\mathrm{sign}(f(a), f(a_1), f(b_1), f(b)) = (+, -, +, -)$

変曲点を含む区間 (a_1, b_1) によって，符号の異なる 3 つの区間 (a, a_1)，(a_1, b_1)，(b_1, b) で包含が成立している．したがってこれを処理するプログラムは単純である．

$(+, -, -, -)$ **の場合**　変曲点の座標値が負のとき（$f''(c) = 0$ のとき $f(c) < 0$），変曲点を含み両端で傾きが異符号の区間 (a_1, b_1) が根を含まない場合がある（図 8 の左）．一方，図 8 の右は，区間 (a_1, b_1) が 2 根を含む．

図 8: bisect3 による (a_1, b_1) と符号 $\mathrm{sign}(f(a), f(a_1), f(b_1), f(b)) = (+, -, -, -)$ の 2 ケース

この 2 ケースのどちらかは $f'(b_1)$ の符号（または $f'(a_1)$ の符号）で判定できる．$f'(b_1) > 0$ の場合は，`bisect2`(b_1, b) によって 1 根を含む区間 (a_2, b_2) を分離するが，(a_2, b_2) の選び方は 2 通りある．図では x 軸の下に (a_2, b_2) の範囲を矢印で示した．$f'(b_1) < 0$ の場合は，`bisect2`(a_1, b_1) によって 1 根を含む区間 (a_2, b_2) を分離するが，この選び方も 2 通りある．(a_2, b_2) 間の他に，これらの場合に応じた包含を成立させる．もう 1

つの根は (a, a_1) にある．プログラム処理は，`bisect3` 関数が返す区間 (a_1, b_1) によって `bisect2` の使い方が 2 通りに分れ，それぞれの場合に，(a_2, b_2) が 2 通りあるので，if ブロックの構造は 2 階層になる．

$(+, +, +, -)$ の場合　$(+, -, -, -)$ の場合を，変曲点回りに 180° 回転すると $(+, +, +, -)$ の場合になる．したがって，変曲点の座標値が正のとき，変曲点を含み両端で傾きが異符号の区間 (a_1, b_1) が根を含まない場合がある（図 9 の左）．一方，図 9 の右は，区間 (a_1, b_1) が 2 根を含む．

図 9: bisect3 による (a_1, b_1) と符号 $\text{sign}(f(a), f(a_1), f(b_1), f(b)) = (+, +, +, -)$ の 2 ケース

この 2 ケースを $f'(b_1)$ の符号で判定し，`bisect2`(a, a_1) または (a_1, b_1) によって 1 根を含む区間を分離する．プログラム処理は，$(+, -, -, -)$ の場合と同様に，if ブロックの構造は 2 階層になる．

$(+, +, -, -)$ の場合　図 10 の左に示したように，(a_1, b_1) 間に 1 根存在し，残る 2 根は (b_1, b) 間に存在する．

図 10: bisect3 による (a_1, b_1) と符号 $\text{sign}(f(a), f(a_1), f(b_1), f(b)) = (+, +, -, -)$ と導関数

したがって bisect2(b_1, b) で分離する．このケースで重要な点は，b_1 が右側の極値よりも右側には選ばれないことにある．もしそのように選ばれると，符号による状態が遷移しない（ (a_1, b_1) 間に依然として 3 根存在する）ので，プログラムが終わらなくなる危険性がある．「$f(x)$ の a_1 での傾きは負なので，図 10 の右に点線で導関数を示したように $f'(a_1) < 0$ で,（変曲点を含み傾きが異符号の区間 (a_1, b_1) の）b_1 は $f'(x) > 0$ の区間から選ばれなくてはならない」．つまり b_1 は,「変曲点と右側の極値の間」に存在しなくてはならない．プログラム処理は，$(+, -, -, -)$ の場合のように `bisect3` 関数が返す区間 (a_1, b_1) によって 2 通りに分岐させる必要がないのでやや単純になる．

countinc の if ブロック構造　ここでは図解の都合上 $f(a) > 0$ の場合について説明したが，$f(a) < 0$ の場合は正負の条件がすべて逆になる．したがって,「$(+, -, +, -)$ の場合」と書かれた条件は「$(+, -, +, -)$ または $(-, +, -, +)$ の場合」と読み替える．この読み替えられた文章に対して素朴に if 文でブロック構造を記述する．次のプログラムでは，変数 `a0` は上記の説明の a，`a1` は a_1，`fa1` は $f(a_1)$ などに対応する．また，if ブロックの処理の順番は，上記の説明の順番とは異なる．

countinc 関数の 3 根を分離するための if ブロック構造

```
if((fa0>rational::ZERO && fa1<rational::ZERO && fb1>rational::ZERO) ||
   (fa0<rational::ZERO && fa1>rational::ZERO && fb1<rational::ZERO)){
      // sign(fa0,fa1,fb1,fb0)=(+ - + -) || sign(fa0,fa1,fb1,fb0)=(- + - +) の場合の処理
}else if((fa0>rational::ZERO && fa1>rational::ZERO && fb1<rational::ZERO) ||
         (fa0<rational::ZERO && fa1<rational::ZERO && fb1>rational::ZERO)){
      // sign(fa0,fa1,fb1,fb0)=(+ + - -) || sign(fa0,fa1,fb1,fb0)=(- - + +) の場合の処理
}else if((fa0>rational::ZERO && fa1<rational::ZERO && fb1<rational::ZERO) ||
         (fa0<rational::ZERO && fa1>rational::ZERO && fb1>rational::ZERO)){
      // sign(fa0,fa1,fb1,fb0)=(+ - - -) || sign(fa0,fa1,fb1,fb0)=(- + + +) の場合の処理
}else{
      // sign(fa0,fa1,fb1,fb0)=(+ + + -) || sign(fa0,fa1,fb1,fb0)=(- - - +) の場合の処理
}
```

`countinc` 関数に mult $=2$ によって示唆されるのは「3 根があるかもしれない」可能性で，実際にはヘッセンベルグ変換とフロベニウス変換によって得られた小行列から特性多項式を得ているので，区間 (a,b) に存在する根の数は 3，2，1，0 個の場合がある．これらを含めて全体の if ブロック構造は次のようになる．

countinc 関数の `mult==2` の場合の if ブロック構造

```
if(fa*fb > rational::ZERO){  // three roots do not exist, but two may exist.
   if(fda*fdb < rational::ZERO && fa*fda < rational::ZERO){ //  two roots may exist.
     // two roots
   }
}else{
   if( fdda*fddb > rational::ZERO){  // cannot call bisect3
     // one root
   }else{
     // three roots
   }
}
```

変数 `fa` は $f(a)$，`fda` は $f'(a)$，`fdda` は $f''(a)$ が，最初の `Horner3` 呼出しの後に格納される．上記の「3 根を分離するための if ブロック構造」は，「// three roots」の部分に入る．

この構造は `mult==2` の場合の if ブロック構造の中にあり，この上に `mult==1` と `mult==0` の if ブロックがある．

5.3.3 零を含むグループに対する包含の数値例

浮動小数点演算で得られる零近傍の固有値は，精度が 1 桁もない場合がある（符号も異なる場合がある）．1 グループに 3 根が存在するケースの例は，2 次元トラス構造解析プログラム CT2D（32 ビット環境で “–O3” の最適化コンパイルオプションを指定して作成）を 13 節点と 41 節点の 2 通りのデータに対して実行して抽出した行列において現れる．この行列データを，32 ビットの Cygwin 環境，64 ビットの Cygwin 環境，64 ビットの Linux 環境で実行すると，数値計算ライブラリとコンパイラの生成するコードの計算順序の違いから，倍精度演算によるヤコビ法によって得られる近似固有値は三者三様の結果を与える．行列は同じなので，どの環境で実行しても真の固有値を包含することに変わりはないが，出発値が異なるので，本節で述べた関数値の符号の組合せは異なる状態を遷移して包含に至る．

図 11 の左に，13 節点データに対する，32 ビットの Cygwin 環境での最初の区間 (a,b) と，3 根を示した．図の右に，41 節点データに対する，32 ビットの Cygwin 環境での最初の区間 (a,b) と，3 根を示した．13 節点データによる行列は 26×26 の行列 $A=S^{-1}KS^{-1}$ を生成し，これを dvsrch2 プログラムが読込み，26 次の特性多項式を作る．最大固有値は $\lambda_{26} \fallingdotseq 5.8\times 10^{16}$ である．41 節点データによる行列は 82×82 の行列に対して，82 次の特性多項式を作る．最大固有値は $\lambda_{82} \fallingdotseq 6.0\times 10^{16}$ である．どちらのデータの場合も，特性多項式の次数は 3 次よりもかなり高い．しかし特性多項式の零点の存在する範囲は 10^{16} のオーダーであり，これに対し区間幅は 1 桁か 2 桁しかない．3 根が 1 つの区間に存在する場合のアルゴリズムの説明を，3 次式の図を用いて行ってきたが，**区間幅がこのように狭い場合には，局所的に 3 次式と見なす**ことができる．

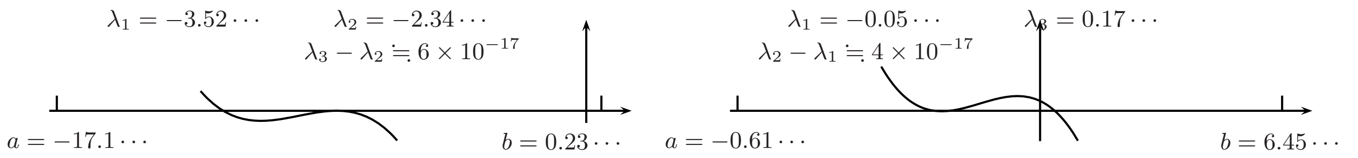


図 11: 13 節点データ（左）と 41 節点データ（右）の 3 根の分布

表 7 に，13 節点データに対する，ヤコビ法で得た小さいほうから 3 つの近似固有値，設定された区間 (a, b)，関数の符号 $\mathrm{sign}(f(a), f(a_1), f(b_1), f(b))$ と導関数の符号 $\mathrm{sign}(f'(a), f'(a_1), f'(b_1), f'(b))$ を示した．表 8 に，41 節点データに対する同様の値を示した．

倍精度算術演算の固有値は安定に求められて 10 数桁の精度があるが，固有値が零近傍の場合，丸め誤差の影響で事実上精度がなくなっている．例えば表 8 の 41 節点データに対する 32 ビット Cygwin 環境では，3 根が負であるにもかかわらず，区間は正になる．このため区間の上限と下限の設定は，全固有値の平均値と比較して「零近傍」を判断して，特別な設定が必要になる．設定方法は，61 ページに示した「上限と下限のセットと包含カウント」プログラムによって,「下限から下限の絶対値の 2 倍を引く」あるいは「上限に上限の絶対値の 2 倍を加える」などとした．

しかしこの設定だけでは不十分で，`chkinc` 関数は，固有値の平均 `am` と相対誤差判定値 `eps` から，固有値が零近傍であると判断し，包含が成立していない場合は，さらに上限と下限を 2 倍して再度関数値を求めて包含を再トライする．

表 7: 13 節点データの近似固有値の上下限, $((a, b))$，関数と導関数の符号

	32 ビット Cygwin	64 ビット Cygwin	Linux
近似固有値	$-4.1 \sim 0.57$	$-3.2 \sim 3.4$	$-5.7 \sim -0.23$
$a \sim b$	$-8.2 \sim 1.14$	$-6.4 \sim 6.8$	$-17.1 \sim 0.23$
$\mathrm{sign}(f(a), f(a_1), f(b_1), f(b))$	$+, +, -, -$	$+, -, -, -$	$+, -, -, -$
$\mathrm{sign}(f'(a), f'(a_1), f'(b_1), f'(b))$	$-, -, +, -$	$-, -, +, -$	$-, -, +, -$

表 8: 41 節点データの近似固有値の上下限, $((a, b))$, 関数と導関数の符号

	32 ビット Cygwin	64 ビット Cygwin	Linux
近似固有値	$0.61 \sim 2.15$	$-7.1 \sim 3.04$	$-2.67 \sim 3.3$
$a \sim b$	$-0.61 \sim 6.45$	$-14.25 \sim 6.08$	$-5.35 \sim 6.6$
$\mathrm{sign}(f(a), f(a_1), f(b_1), f(b))$	$+, +, +, -$	$+, +, +, -$	$+, +, +, -$
$\mathrm{sign}(f'(a), f'(a_1), f'(b_1), f'(b))$	$-, -, +, -$	$-, -, +, -$	$-, -, +, -$

chkinc 関数

```
int chkinc(const int n, const rational_vector& p, const double e, const double h, const int mult,
     const double am, rational& a, rational& b) {
   double ta,tb;
   int rtnval=0;
   rational fa,fb;

   if(fabs(e/am) > eps){
      ta=e-fabs(e)*eps; tb=h+fabs(h)*eps;
      if(mult==0) tb=e+fabs(e)*eps;
      a = Rdset(&ta); fa=Horner(n,p,a);
      b = Rdset(&tb); fb=Horner(n,p,b);
      if(fa*fb<rational::ZERO) rtnval=1;
   }else{                                          ゼロ近傍の場合で
      ta=e-eps; tb=h+eps;   // consider eigenvalues close to zero
      if(mult==0) tb=e+eps;
      a = Rdset(&ta); fa=Horner(n,p,a);
      b = Rdset(&tb); fb=Horner(n,p,b);
      if(fa*fb<rational::ZERO){
         rtnval=1;
      }else{                                       包含が不成立の場合
         if(mult>0){
            ta=2*ta; tb=2*tb;                      区間を拡張して再トライする
            a = Rdset(&ta); fa=Horner(n,p,a);
            b = Rdset(&tb); fb=Horner(n,p,b);
            std::cout << "chkink interval inflation ta=" << ta << " fa=" << (double)fa << " tb=" << tb << " fb=" << (doub
            if(fa*fb<rational::ZERO) rtnval=1;
         }
      }
   }
   return rtnval;
}
```

2 倍すれば包含が成立するという保証はないが，前述したように，3 通りの環境で，この応急処置で包含が成立した．正しくは，多項式の次数と最高次の係数，および定数項の符号などを利用して，零近傍の根を包含する頑強（robuat）なアルゴリズムを考案すべきであろう

真の解は 1 つであるが，包含の出発値を倍精度浮動小数点演算で得ており，これが言語ライブラリやコンパイラのコード生成の違いによって相対的にかなり大きな差異を生じさせる．この差異を吸収して，包含を成立させれば，その後の精度改良は単純なアルゴリズムで実現され，真の解に向かって解の存在する区間は縮小される．**浮動小数点演算と有理算術演算の接点となる部分**なので，プログラミングの好例と考え，零近傍の固有値の包含の方法を紹介した．

5.3.4　挟み撃ち法

2 分法 bisect 関数を，挟み撃ち法 bisectrf 関数に置換えて実行する（#define BISECT を指定しないでコンパイルする）と，反復回数が少なくなる．

図 12: 乗算の精度

挟み撃ち法を浮動小数点演算のように，2 分法の中点の代わりに，$d = \dfrac{af(b) - bf(a)}{f(b) - f(a)}$ で得られる x 切片の座標値 d を用いると，分母 2 べきが保存されない．そこで `bisectrf` 関数ではこれを `roundrat` 関数で分母 2 べきにしている．詳しくは，LongintRational.pdf を参照されたい．

5.3.5 精度改良の dvsrch1.cpp に対する変更点

dvsrch1 では，判定式 (38) が使用する近似固有値は，そのグループに属する固有値のおよその値を使用した．

dvsrch1 で用いた参照固有値

```
log2ev=log2(eigvl[i])+1;                  // その近似固有値の小数点よりも上のビット数
int reqndeig=((int)log2ev+2)*d+accdouble; // 必要桁（ビット）数
int niteration=reqndeig-mxitr[i];
 略
double dzero=0;
bisect(p,nsize,dzero,niteration,eiglow[i],faeig,eigup[i],fbeig,ceig);
```

n 桁の精度がある数同士の積は，約 n 桁の精度がある．図 12 の左図は，被乗数も乗数も真ん中に小数点があり，積の真ん中に小数点がくる場合を示した．被乗数と乗数がともに同じ桁数の精度がある場合，積もその精度まではほぼ正しく，下半分は誤差が混入している．したがって，精度が n 桁あれば（小数点の上の $n \div 2$ 桁と下に $n \div 2$ 桁が正しければ），積の小数点よりも上の n 桁はぎりぎりである．整数性判定は小数点以下の数桁を見るので，この場合は精度の改良が必要になる．

もし一方の数値の精度が悪いと，積の精度はどうなるだろうか．図 12 の右図は，乗数の真ん中の数字が怪しい（精度が $n \div 2$ 桁しかない）場合である．**積の精度は短いほうの桁数**しか保証されない．**倍精度浮動小数点演算で零固有値を求めても**，行列そのものに丸め誤差が混入していて，零にならず，**絶対値の小さな固有値が得られるが，この固有値は精度がない**．零固有値の小数点よりも上の桁数を参照して必要精度を決め，さらに精度のない零固有値に対する近似解を，この必要精度から得られた反復回数だけ反復する dvsrch1 の方法では，整数性判定ができない．そこで dvsrch2 では，要求精度の参照固有値は，包含を作成のループ反復で最大固有値を変数 `evlarge` に保存して，この桁数（ビット数）を判定式 (38) に代入し必要精度を決定した．

dvsrch2 で用いた参照固有値と bisect 用区間幅

```
log2ev=log2(evlarge)+1;                 // 最大近似固有値の小数点よりも上のビット数
reqndeig=((int)log2ev+2)*d+accdouble;   // 必要桁（ビット）数
double stmp=pow(2,log2ev*d);            // 最大近似固有値の小数点よりも上のビット数を数値に変換
double epstmp=1/stmp;                   // その逆数で区間幅に変換
 略
int irtn=bisect(p,nsize,epstmp,reqndeig,eigint[i],faeig,fbeig,eigvr[i]);
```

さらに反復回数 k を，区間幅 2^{-k} に変換することで，2 分法以外の区間縮小アルゴリズムでも使用できるように改良した．

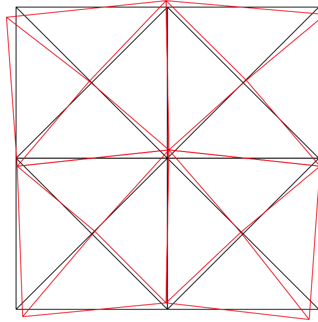


図 13: 13 節点トラス構造の振動モード

5.3.6 その他の dvsrch1.cpp に対する変更点

固有値が零近傍の小さな値をとるため，上記の変更点のほかに，`selectev` 関数に，相対誤差を使用する目的で，近似固有値の中間的な値を参照した．この値は変数 `am` に保存して `chkinc` 関数でも使用した．dvsrch1 では特性多項式が 1 次式の場合は因子探しを行わなかったが，dvsrch2 では，1 次式でも対応する近似固有値があるかどうかを見極める目的で，因子探しを行う．

因子多項式の符号は，dvsrch1 ではこだわらなかったが，dvsrch2 では問題の種類が増え，計算結果を diff コマンドでチェックするために，因子多項式の最高次の項は 1 とすることで符号を統一した．

5.4 固有ベクトルの計算

行列の全要素の分母の LCM 倍を求めてかけることで整数行列とすると，元の行列の有理数固有値は整数固有値になる．この固有値は 1 次式で分離される．この固有値に対応する固有ベクトルの成分も整数で得られる．λ が行列 A の正確な固有値のときは，$A-\lambda I$ は特異であり，このとき線形同次方程式 $(A-\lambda I)\boldsymbol{x}=0$ の非自明解が固有ベクトルである．これを ratutil クラスの関数 solhmg によって得る．

5.5 計算例

はじめに解析内容を指定する**入力データの与え方**について述べる．このとき “0 S13Free 0” のように，“数字，ファイル名，数字” を指定する．最初の数字はチェックポイント・ランかリスタート・ランかを指定する．0 のときは初期ラン，1 のときはリスタートで，ヘッセンベルグ行列から読み込み，2 のときはフロベニウス行列（特性多項式）をチェックポイント・ファイルから読み込む（5.2 項）．ファイル名は CT2D の行列を格納したファイル名から拡張子 “.txt” を除いた文字列で，3 つ目の数字が解析する行列のタイプである．0，1，2 で，$S^{-1}KS^{-1}$，K，$M^{-1}K$ のいずれかを指定する．初期ランでは常にチェックポイントをとる（ファイル名は 5.2 項）．

13 節点の剛体モードをもつ構造モデル S13Free.txt を例に説明する．図 13 に，最低次の非零モードを示した．赤が振動モードで，同じ振動数のモードが存在し $\lambda_4=\lambda_5=1.5542\ldots$ である．上辺と下辺が伸び縮みするモードで，これを 90 度回転したモードが同じ固有振動数で存在する．このような多重度 2 の非零固有値が 6 ペア存在する．零固有値になる並進モードと剛体回転モードは，力学的には 3 重根であるが，誤差の影響で並進モードは重根，回転モードは異なる固有値をとる．表 9 にこれを示した．3 つの理論的には零になってほしい固有値は，CT2D では零としては得られない．グラフ・ラプラシアン行列の零固有値は，特性多項式の定数項が零になるので，浮動小数点演算で得た近似固有値が非零であっても解けた．しかし，浮動小数点演算で作成された行列の特性多項式を正確に求めても，定数項は零にならない．これは行列そのものがすでに丸め誤差で汚染されているからである．さらにこの**絶対値の小さな近似固有値の有効桁数がない**（正確に計算してみると，符号すらもあっていない）うえに，システムによって異なる値を与える．これが問題を厄介にする．これが，精度改良の参照固

表 9: S13 モデルの固有値

group	i	λ_i	下位 3 桁の差	
1	1,2,3	0.	$-0.00\ldots007,\ -0.00\ldots006,\ 0.00\ldots003$	$-1.6150E-16,\ -1.5117E-16$
2	4,5	$1.554\ldots$	$1.55\ldots331,\ 1.55\ldots335$	1.55421534213563386560
3	6	$1.776\ldots$		1.77640027632889171915
4	7	$2.175\ldots$		2.17495713431908715053
5	8	$2.955\ldots$		2.95518450014503430978
6	9,10	$3.380\ldots$	$3.38\ldots410,\ 3.38\ldots446$	3.38026893907544201597
7	11	$3.456\ldots$		3.45573401676622660333
8	12,13	$4.488\ldots$	$4.48\ldots162,\ 4.48\ldots189$	4.48755519945811977008
9	14	$4.753\ldots$		4.75318573732433953243
10	15	$5.045\ldots$		5.04481549985496435795
11	16	$6.139\ldots$		6.13854338429956082551
12	17,18	$8.161\ldots$	$8.16\ldots960,\ 8.16\ldots978$	8.16130634735060237338
13	19,20	$10.71\ldots$	$10.7\ldots541,\ 10.7\ldots648$	10.70846185912325619640
14	21	$10.87\ldots$		10.86985836553587454288
15	22	$11.25\ldots$		11.24681426267565644372
16	23	$11.68\ldots$		11.67871248279866691460
17	24	$12.26\ldots$		12.26431833879140533122
18	25,26	$12.93\ldots$	$12.9\ldots590,\ 12.9\ldots767$	12.93226981213176417701

有値を最大の値（evlarge）に変更しなくてはならなくなった理由である．本章の冒頭に目標として掲げた改良項目の 1 つである，多倍精度演算のサポートについては，この零近傍の解に対する浮動小数点演算の弱点が理由で，4 倍精度，8 倍精度と多重精度算術演算を有理数計算プログラミング環境がサポートしても意味がないと判断した．

5.5.1 最適化オプションなしで CT2D を作成した場合

熱伝導行列の場合のように，各小行列に属する固有値を表 10 にまとめる．LCM は 4,503,599,627,370,496= 2^{52} で，これによって係数行列を整数行列とする．H_1 から得られる固有値は有理数で $\lambda_{15} = 22719829205299720$ であった．つまり H_1 から得られる特性多項式は $-\lambda + \lambda_{15}$ である．

”Sub Matrix 2” の表示後に包含のループが回り，H_9 に対して “FOUND 9 intervals. nsize=9” が表示される．1 次の因子を見つけるために近似固有値の精度改良を行うが，零固有値に対しては，24 回の反復によって $(-0.72695829, -0.72695820)$ の区間に狭めるが，それ以外の近似固有値には，単根には反復 1 回，2 根のグループには 10 回程度の反復を行う．1 次の因子はみつからない．

2 次の因子を見つけるための精度改良は，90 回ほどの反復によって区間幅を 4 から 7×10^{-35} に狭める．零固有値に対しては，次のように表示される．

```
eig=[        -0.72695826351001392823598353775519177966215988907840:
             -0.72695826351001392823598353775519170679139841389762]
                                                  |
```

なおこの表示は formatprn を呼ぶことで得られる．

表 10: 小行列に属する固有値と因子多項式

λ_i			value	LCM 倍での区間（最適化なし）	H_9	H_{16}
λ_2, λ_3	λ_1		$\pm 0.00\cdots$	$(-1.513466502033021,\ 4.344940969805278)$	7	7,3
λ_4	λ_5		$1.554\cdots$	(6999563635695548, 6999563635695552)	7	7
λ_6			$1.776\cdots$	8000195622535650		*3*
	λ_7		$2.175\cdots$	9795136139666244	2	
λ_8			$2.955\cdots$	1.330896781366425e+16		1
λ_9	λ_{10}		$3.380\cdots$	(1.52233779344322e+16, 1.522337793443223e+16)	7	7
λ_{11}			$3.456\cdots$	1.556324243019992e+16		2
λ_{12}	λ_{13}		$4.488\cdots$	(2.02101519240841e+16, 2.021015192408413e+16)	7	7
λ_{14}			$4.753\cdots$	2.140644551543668e+16		3
		λ_{15}	$5.045\cdots$	2.271982920529966e+16		
λ_{16}			$6.139\cdots$	2.764554169812914e+16		*3*
λ_{17}	λ_{18}		$8.161\cdots$	(3.675525622478457e+16, 3.675525622478465e+16)	7	7
λ_{19}	λ_{20}		$10.71\cdots$	(4.822662483845869e+16, 4.822662483845872e+16)	7	7
	λ_{21}		$10.87\cdots$	4.895349008459746e+16	2	
λ_{22}			$11.25\cdots$	5.065114852249124e+16		3
λ_{23}			$11.68\cdots$	5.25962451856991e+16		2
λ_{24}			$12.26\cdots$	5.523357950053409e+16		*3*
λ_{25}	λ_{26}		$12.93\cdots$	(5.824176550697125e+16, 5.824176550697141e+16)	7	7
H_{16}	H_9	H_1				

```
std::cout << "eig="; formatprn(eigint[i]);
```

特性多項式は 2 次式 $\lambda^2 - 58748626224263676\lambda + 479506099890433391927153378920472$ で割り切れて，7 次式になる．この 2 次式は表 10 の λ_7 と λ_{21} の単根から構成される．

3 次式のための精度改良は，区間幅を 10^{-52} 程度にする．3 次の因子は見つからない（因子探しの時間は 0.1 秒以下である）．

この後，4 次，5 次，6 次の因子探しを行うが，この探索はスキップできる（チューニングのアイテムとして残した）．7 次式のための精度改良は，区間幅を 10^{-120} 程度にする．7 次の因子

```
-                          -1.*x^7
+           185656740064426520.*x^6
-13355185 (20 digits) 9311196.*x^5
+47021772 (35 digits) 58533088.*x^4
-84245215 (52 digits) 1060672.*x^3
+71844297 (67 digits) 97565184.*x^2
-22232711 (84 digits) 2703744.*x^1
-16162253 (84 digits) 6976128.*x^0
```

は 9 つの包含された区間のうちの残りの近似固有値から構成されることを確かめる[73]．この 7 つの固有値は，**7 つの重根の片方**である．定数項と 1 次の項の桁数が 99 桁あるので，近似固有値の区間幅も 10^{-120} 程度の精度になっている．H_9 の特性多項式は次のように分解された．

$$(\lambda^2 + a_1\lambda + a_0)(\lambda^7 + b_6\lambda^6 + \cdots + b_0) \tag{45}$$

a_1 と a_0 は λ_7 と λ_{21} の単根から構成され，b_0 から b_6 は λ_1，λ_5，λ_{10}，λ_{13}，λ_{18}，λ_{20}，λ_{26} の重根から構成される．

H_{16} から得られる特性多項式は，式 (45) の 7 次式 $(\lambda^7 + b_6\lambda^6 + \cdots + b_0)$ で割り切れて 9 次式になる[74]．なお，H_9 から得られた 7 次式で割り切れる理由は，この 7 つの固有値が重根の残った片方なので，もとの行列の特性多項式が平方の形 $(-\lambda^7 + \cdots)^2$ になるからである．9 次式は 1 次式 $= 13308967813664242 - \lambda$ で割り切れる（λ_8）．つまり λ_8 は有理数である．残りの 8 次式は 2 次式と 2 つの 3 次式に分解される．つまり，H_{16} の特性多項式は次のように分解された．

$$(\lambda^7 + b_6\lambda^6 + \cdots + b_0)(\lambda - \lambda_8)(\lambda^2 + c_1\lambda + c_0)(\lambda^3 + d_2\lambda^2 + d_1\lambda + d_0)(\lambda^3 + e_2\lambda^2 + e_1\lambda + e_0) \tag{46}$$

c_1 と c_0 は λ_{11} と λ_{23} の単根から構成され，d_0 から d_2 は λ_3，λ_{14}，λ_{22} の単根から構成され，e_0 から e_2 は λ_6，λ_{16}，λ_{24} の単根から構成される．

H_{16} の特性多項式に関係する 2 つの零固有値は，重根のペアで H_9 と H_{16} の 7 次式に含まれ，負根である（2 つの並進モード）．残ったひとつは λ_{14} と λ_{22} と組み合わされた 3 次の因子になる．これは剛体の回転モードで，固有値は正である．表 10 の最後の 2 列に，因子多項式の次数を示した．H_{16} は 2 つの 3 次の因子を持つが，3 とイタリック体の *3* で識別した．

整数性判定のために精度改良される固有値の区間幅は 10^{-120} と狭く，小数点の上に 17 桁の数字があるので 140 桁近い精度が要求される．この精度は，多重精度算術演算では 8 倍精度でも不足する．

5.5.2　$A = K$ を選択した場合

質量行列を考慮しないので，全節点に質量 1 を想定した場合の振動と考えられる．質量が変わるので固有値の値に変化があり，重根値と単根の順序関係が変わる．各小行列に属する固有値を表 11 にまとめる．ヘッセンベルグ

[73] x^7 の係数が -1 で，定数項が負なので,「デカルトの符号法則」をもちださずとも，負の根をもつことが分かる．
[74] この 9 次式の定数項は負で，最高次の係数は 1 なので，負根はもたない．

表 11: $A = K$ の場合の小行列に属する固有値と因子多項式

λ_i		H_{10}	H_{16}
λ_2, λ_3	λ_1	7	7,3
λ_4			*3*
λ_5	λ_6	7	7
λ_7			1
	λ_8	2	
λ_9			2
λ_{10}	λ_{11}	7	7
λ_{12}			3
λ_{13}	λ_{14}	7	7
λ_{15}			*3*
	λ_{16}	1	
λ_{17}			2
λ_{18}	λ_{19}	7	7
λ_{20}			*3*
λ_{21}	λ_{22}	7	7
	λ_{23}	2	
λ_{24}			3
λ_{25}	λ_{26}	7	7

行列は H_{16} と H_{10} に分かれる．零になる 3 つの固有値は，3 つとも小さな正の値をとった（$\lambda_1 = \lambda_2 = 0.61\cdots$，$\lambda_3 = 1.14\cdots$）．

H_{10} の特性多項式は次のように分解された．

$$(\lambda - \lambda_{16})(\lambda^2 + a_1\lambda + a_0)(\lambda^7 + b_6\lambda^6 + \cdots + b_0) \tag{47}$$

a_1 と a_0 は λ_8 と λ_{23} の単根から構成され，b_0 から b_6 は λ_1，λ_6，λ_{11}，λ_{14}，λ_{19}，λ_{22}，λ_{26} の重根から構成される．

H_{16} の特性多項式は次のように分解された．

$$(\lambda - \lambda_7)(\lambda^7 + b_6\lambda^6 + \cdots + b_0)(\lambda^2 + c_1\lambda + c_0)(\lambda^3 + d_2\lambda^2 + d_1\lambda + d_0)(\lambda^3 + e_2\lambda^2 + e_1\lambda + e_0) \tag{48}$$

c_1 と c_0 は λ_9 と λ_{17} の単根から構成され，d_0 から d_2 は λ_3，λ_{12}，λ_{24} の単根から構成され，e_0 から e_2 は λ_4，λ_{15}，λ_{20} の単根から構成される．

5.5.3 $A = M^{-1}K$ を選択した場合

$S^{-1}KS^{-1}$ と $M^{-1}K$ は，演算の丸め誤差がなければ同じ物理モデルになるので，固有値の順序は $S^{-1}KS^{-1}$ の場合と同じになる．ヘッセンベルグ行列は H_{16} と H_{10} に分かれる．各小行列に属する固有値を表 11 の形式でまとめるならば，表 10 の第 1，2，3 列の λ の並びのうち λ_{15} を第 2 列に移動する（表は省略する）．

質量行列の対角項で正確に割り算を計算するので，行列の桁数は増えて，LCM は 6.76×10^{46} となり，もとは零固有値になる 3 つの近似固有値も，倍精度浮動小数点演算では -4.69×10^{31} から 2.10×10^{31} に分布する．最大の近似固有値は 8.74×10^{47} である．

H_{10} の特性多項式は次のように分解された．

$$(\lambda - \lambda_{15})(\lambda^2 + a_1\lambda + a_0)(\lambda^7 + b_6\lambda^6 + \cdots + b_0) \tag{49}$$

a_1 と a_0 は λ_7 と λ_{21} の単根から構成され，b_0 から b_6 は λ_1，λ_5，λ_{10}，λ_{13}，λ_{18}，λ_{20}，λ_{26} の重根から構成される（$S^{-1}KS^{-1}$ の場合と同じ）．

H_{16} の特性多項式は次のように分解された．

$$(\lambda^7 + b_6\lambda^6 + \cdots + b_0)(\lambda - \lambda_8)(\lambda^2 + c_1\lambda + c_0)(\lambda^3 + d_2\lambda^2 + d_1\lambda + d_0)(\lambda^3 + e_2\lambda^2 + e_1\lambda + e_0) \tag{50}$$

c_1 と c_0 は λ_{11} と λ_{23} の単根から構成され，d_0 から d_2 は λ_3，λ_{14}，λ_{22} の単根から構成され，e_0 から e_2 は λ_6，λ_{16}，λ_{24} の単根から構成される（$S^{-1}KS^{-1}$ の場合と同じ）．

最後の 7 次の因子を 7 つの近似固有値から構成するときの，近似固有値の精度は，500 桁を超える．

```
 Refined accuracy of eigval eigvr=8.795420292271488e+30 Width of interval=0
eig=[  8795420292271488564653283220829.969762110309793004 (468 digits) 190808880683222488:
       8795420292271488564653283220829.969762110309793004 (468 digits) 190808880683262954]
 eiglow_n=61 eiglow_d=58 eigup_n=61 eigup_d=58
```

dvsrch1.cpp ですでに，整数性判定に必要な精度まで近似固有値の精度を改良すれば，因子多項式を見つけられることを確かめていたが，この程度のデータでも，必要な精度は，多倍精度算術演算では 32 倍精度を超える長さに達している．

5.5.4 Cygwin 環境で最適化オプション -O3 で CT2D を作成し $A = S^{-1}KS^{-1}$ とした場合

ヘッセンベルグ行列の副対角項はすべて非零であり，26 次の特性多項式が得られ（定数項は 379 桁），多重度の高い固有値はなくなる．この問題では，零固有値のグループに 3 根が存在するので，包含が困難になる．浮動小数点演算で近似固有値を求めるので，システムソフトや計算機の命令セットの違いで，得られる固有値に違いが生ずる．ここでは 64 ビットアーキテクチャの Linux システムの結果で説明する．3 根を含む区間 $(a_0, b_0) = (-5.7,\ -0.22)$ に対して `bisect3` で 2 階導関数（曲率）がゼロを含み両端で傾きが異なる区間 $(a_1, b_1) = (-2.96,\ -0.22)$ が見つかる．この区間に対して `bisect2` で導関数（傾き）がゼロを含み両端で関数値が異なる区間 $(a_2, b_2) = (-2.340375387850572,\ -2.340375387850571)$ が見つかる．この区間を `bisect` で絞り込み，区間 (a, b) が

```
a=       -2.34037538785057119945927902383957475866260145381082
b=       -2.34037538785057119945927902383943958258973208643083
                                                 |
```

に固有値が 1 つ存在することが分かる．図 6 の右で，区間に 3 根が存在するグラフを示したが，ここで解析している問題は，本来は重根であったものが，倍精度浮動小数点演算の丸め誤差のために分離した 2 根なので，図示すると図 14 のように描ける．

2 つめの包含をプログラムのコメントの図

```
//            ------|----|----|---------|-----|------|------------
//                 a0   a1   a2        b2    b1     b0
//                            f''(x)=0
//  setinit               1               2              second root
//  setinit          1                          2        third  root
```

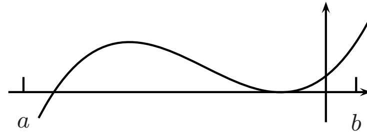


図 14: $[-4, 1]$ に 3 根が存在する場合（2 根は重根に近い状態）

に従って説明する．$(a_1,\ b_1)$ 間には 2 根存在するはずなので，次に (a_1, a_2) か $(b_2,\ b_1)$ を探す（変数 setinit が 1 か 2）．ここでは setinit $= 2$ が選択されて，bisect で，区間 (a, b) が

```
a= -2.34037538785057115846081188823544830251686213817924:
b= -2.34037538785057068919606547918916441250341339773654]
                         |
```

に固有値が 1 つ存在することが分かる．

3 つめの包含は，$(a_0,\ a_1)$ か $(b_1,\ b_0)$ を探す．ここでは setinit $= 1$ が選択されて，bisect で，区間 (a, b) が

```
 a= -3.65411544897164871498196347943121509160846471786499
 b= -2.96869132933425497467005982343835057690739631652832
```

に固有値が 1 つ存在することが分かる．

他のグループについては，グループに 2 つ以下の近似固有値なので，bisect2 の使用で，FOUND 26 intervals. となって包含が成立する．

有理数の固有値 λ_8 は 1 次式として分解される．

2 次の因子探しでは近似固有値の精度は，最小固有値に対しては，

```
 Refined accuracy of eigval eigvr=-2.340375387850571 Width of interval=6.600394183074579e-35
eig=[         -2.34037538785057119945927902383953132806887682308034:
              -2.34037538785057119945927902383953126206493499233455]
 eiglow_n=6 eiglow_d=6 eigup_n=6 eigup_d=6
```

の精度で多項式係数の整数性判定を行う．最後の行は，区間の下限の有理数の分子が 2^{32} 進数で 6 桁，分母が 6 桁，分母の有理数の分子が 6 桁，分母が 6 桁を表示している．

3 次から 11 次まで因子は見つからない．12 次の因子探しには 200 桁を超える精度が要求される．

```
 Refined accuracy of eigval eigvr=-2.340375387850571 Width of interval=1.366351164922065e-205
eig=[         -2.340375387850571199 (173 digits) 626977603588361562:
              -2.340375387850571199 (173 digits) 626977603588347899]
 eiglow_n=24 eiglow_d=23 eigup_n=24 eigup_d=24
```

25 次の特性多項式は 12 次式で割り切れる．

13 次の因子探しには 230 桁程度の精度が要求される．12 次の因子と値が近いほうの零固有値は次の区間に絞り込まれた．

```
 Refined accuracy of eigval eigvr=-2.340375387850571 Width of interval=9.480965768874258e-223
eig=[         -2.340375387850571199 (191 digits) 060678401700349592:
              -2.340375387850571199 (191 digits) 060678401700254782]
 eiglow_n=25 eiglow_d=25 eigup_n=25 eigup_d=25
```

もう１つの理論的に零になる固有値は次の区間に絞り込まれた．

```
  Refined accuracy of eigval eigvr=-3.457446882946081 Width of interval=9.480965768874258e-223
eig=[         -3.457446882946080503 (191 digits) 951572206194082491:
              -3.457446882946080503 (191 digits) 951572206193987682]
 eiglow_n=25 eiglow_d=25 eigup_n=25 eigup_d=25
```

13 次多項式が，2 つの零固有値を含む 13 個の近似固有値から構成された．

構造の形状対称性から**重根になるペアは 12 次の因子多項式と 13 次の因子多項式に分かれた**．また 3 つの理論的に零固有値になる 3 根は，最適化なしの場合と同様に，2 つの並進モードが 12 次と 13 次に分かれ，剛体の回転モードは 13 次の因子に含まれた．これは最適化なしの場合の H_{16} に剛体の回転モードが回ったことと関係があるかもしれない（理由はわからない）．

H_{26} の特性多項式は次のように分解された．

$$(\lambda - \lambda_8)(\lambda^{12} + a_{11}\lambda^{11} + \cdots + a_0)(\lambda^{13} + b_{12}\lambda^{12} + \cdots + b_0) \tag{51}$$

λ_8 は，最適化なしでは H_1 で分離していた有理数の固有値である．最適化した場合の行列では，ヘッセンベルグ変換で H_1 として分離されなくなった．

a_{11} から a_0 は，7 つの重根のペアと λ_7，λ_{11}，λ_{15}，λ_{21}，λ_{23} から構成される．表 10 では， λ_7 と λ_{21} は H_9 の特性多項式の 2 次の因子，λ_{11} と λ_{23} は H_{16} の特性多項式の 2 次の因子，λ_{15} は有理数で H_1 に分離していた．

b_{12} から b_0 は，7 つの重根のペアと H_{16} で 2 つの 3 次多項式の因子を構成した固有値から構成される．このように，最適化なしでの因数分解の因子多項式を構成する固有値が，最適化のための誤差に影響されても因子多項式のペアリングを崩さないことは，もとの行列の係数の作られ方と，丸め誤差の載り方に起因していると考えられるが，細かいことは分からない[75]．

5.5.5 Cygwin 環境で最適化オプション -O3 で CT2D を作成し $A = K$ とした場合

ヘッセンベルグ行列は H_{16} と H_{10} に分かれる．H_{10} と H_{16} の特性多項式は，最適化なしの場合の式 (47) および式 (48) と同じであった．

5.5.6 Cygwin 環境で最適化オプション -O3 で CT2D を作成し $A = M^{-1}K$ とした場合

ヘッセンベルグ行列は H_{16} と H_{10} に分かれる．H_{10} と H_{16} の特性多項式は，最適化なしの場合の式 (49) および式 (50) と同じであった．

5.5.7 多重精度算術演算のサポートに関して

浮動小数点演算のヤコビ法などで求める固有値の精度は，通常は 10 数桁あるが，零固有値の近傍では，全く精度がない場合がある．今回の計算では，符号すら違っていた．これを 4 倍精度にしても，改善の保証はない．したがって，**精度を出す必要がある場合は，多重精度算術演算よりも有理算術演算を用いるべき**である．さらにそれを区間算術演算の形で使用するのがよい．

[75]固有ベクトルと合わせて考えるとヒントが得られるかもしれない．

5.5.8 デバッグ例

有理算術演算は**数値の 1 ビットの違いにも敏感に結果が反応する**．浮動小数点演算では，有効桁の下位のビットが違っても，結果に大差ないのが普通なので，著しい特長である．CT2D の行列で最適化ありとなしでは，行列要素の違いは浮動小数点データの下位のわずかな（数値を目で眺めても分からない）ビットにしか存在しないが，この差に敏感に反応して，ヘッセンベルグ変換で副対角項が零にならず，計算時間は数倍になる．プログラムのバグで変換が正しく行われない場合も同様の結果になることが多い．このような問題を解決するために，ヘッセンベルグ変換では，軸選択，副対角項の値（倍精度変換），その桁数の情報を表示している[76]．行列 K を選択した場合次のように表示される．

剛性行列のヘッセンベルグ変換中の表示

```
elmhes  start n=26
m=2 i=3 (double)x=-9.0072e+15 3
m=3 i=5 (double)x=-9.0072e+15 3
m=4 i=13 (double)x=4.72215e+16 10
m=5 i=7 (double)x=-2.11837e+16 13
m=6 i=12 (double)x=8.85753e+15 30
m=7 i=26 (double)x=-1.06297e+16 53
       中略
m=15 i=19 (double)x=-2.89904e+15 340
m=16 i=20 (double)x=4216.64 364
m=17 i=17 (double)x=0 1
m=18 i=25 (double)x=-2.36378e+16 30
m=19 i=23 (double)x=-2.91652e+16 59
       中略
m=24 i=26 (double)x=5.33917e+15 117
m=25 i=25 (double)x=1.46586e+16 137
```

入力行列を間違えたり，プログラムの下位の階層のバグに触れたりすると，副対角項 k_{17} が零にならず，計算時間が極端に伸びる（1 分程度の分解の時間が 8 時間近くかかる）．

剛性行列のヘッセンベルグ変換中の表示

```
 * elmhes  start n=26
m=2 i=3 x=-9.0072e+15 3
m=3 i=5 x=-9.0072e+15 3
m=4 i=13 x=4.72215e+16 10
m=5 i=7 x=-2.11837e+16 13
m=6 i=12 x=8.85753e+15 30
m=7 i=26 x=-1.06297e+16 70    <=== ここで違っている
m=8 i=8 x=2.02643e+16 120
       中略
m=24 i=25 x=-4.50491e+15 55238
m=25 i=26 x=-2.51654e+15 59828
```

この例では正解と比較できるので,（計算は 7 列目で停止するようにして）更新された行列要素をすべて出力するプログラムを 2 通り実行して，正しい結果と比較する（diff をとる）ことで，**問題が最初に現れた演算**を捜した．次に示す問題解決用のコードは，ヘッセンベルグ変換の関数 elmhes にはコメントで残した．

[76] ここでのデバッグ例は，プログラムの更新のミスでバグがある longint.cpp を使用したことで発生したので，正しい計算（正解）と比較することで問題解決を行うことができた．このような場合での問題追及の方法を紹介する．

elmhes に加えたデバッグ用プリント

```
for(int j=m; j<=n; ++j) {
   rational saved=a[i-1][j-1];
   a[i-1][j-1] = a[i-1][j-1] - y * a[m-1][j-1];
   if(saved != a[i-1][j-1]) std::cout << "change a[i-1][j-1]=" << a[i-1][j-1] << " i=" << i << " j=" << j
   << " a[m-1][j-1]=" << a[m-1][j-1]  << " m-1=" << m-1 << " saved=" << saved << " y=" << y << std::endl;
}
```

乗加算 “saved - y * a[m-1][j-1]” に着目する．次に示すペアの 2 行は，上が正しくない計算の表示，下が正しい計算の表示である．被乗数 `y` も `a[m-1][j-1]` も 被加数 `seved` も同じであるが結果 `a[i-1][j-1]` が異なるのであるから，乗加算 “saved - y * a[m-1][j-1]” が正しく計算されていない．

elmhes に加えたデバッグ用プリント

```
change a[i-1][j-1]=766654595212970732358095103346411/47221509289895611 i=6 j=6
change a[i-1][j-1]=766654595212996819993745768917245/47221509289895611 i=6 j=6

a[m-1][j-1]=258359429617980736260547150317435671248383889011/40564819207303340847894502572032 m-1=3
a[m-1][j-1]=258359429617980736260547150317435671248383889011/40564819207303340847894502572032 m-1=3

saved=15376250927266764/1
saved=15376250927266764/1

y=-40564819207303340847894502572032/300756232722000874105869297291081
y=-40564819207303340847894502572032/300756232722000874105869297291081
```

この部分をテキストエディタのコピー&ペースト機能で，デバッグ用の mpyad.cpp プログラムに埋め込んで確かめた．

デバッグ用 mpyad.cpp プログラム

```
int main(int argc, char** argv) {
longint an("15376250927266764");
rational a=RRset(an,longint::ONE);
longint bn("258359429617980736260547150317435671248383889011");
longint bd("40564819207303340847894502572032");
rational b=RRset(bn,bd);
longint cn("40564819207303340847894502572032");
longint cd("300756232722000874105869297291081");
rational c=RRset(cn,cd);
rational d=a+b*c;
std::cout << "d=" << d << std::endl;
}
```

これによって下位の階層（rational クラスか longint クラス）に問題があることがはっきりした．“g++ mpyad.cpp rational.o longint.o mempool.o” でコンパイルでき，稼働するので，デバッグ作業は軽くなる．

付　録

A　ヤコビ法

A.1　ヤコビ法のアルゴリズム

ヤコビ法（1846 年）は対称行列の固有値と固有ベクトルを，平面回転を繰返し適用することで求める．n 次元空間（座標軸を $x_1, x_2, \ldots, x_n$ とする）内の特定の平面（$x_i x_j$ 平面）を選び，その面内で θ だけ回転を与える行列は $c = \cos\theta$ および $s = \sin\theta$ とすると次のように表わすことができる．

$$H^{-1} = R^T = \begin{pmatrix} \ddots & & & & \\ & c & & s & \\ & & \ddots & & \\ & -s & & c & \\ & & & & \ddots \end{pmatrix} \tag{52}$$

この行列で c や s などを記入した以外の成分は，対角項は 1 で非対角項はゼロである．第 (i,i) 成分と (j,j) 成分は c，(i,j) 成分は s，(j,i) 成分は $-s$ である．

平面回転 $R^T AR$ を繰り返すことによって，A の非対角項をすこしずつ小さくしてゆき，対角行列に導く．$A^{(0)} = A$ として，回転操作を $A^{(k)} = R_k^T A^{(k-1)} R_k$ のように記述する．最初の相似変換 $A^{(1)} = R_1^T AR_1$ を書き下す．

$$R_1^T AR_1 = \begin{pmatrix} & \vdots & & \vdots & \\ \cdots & a_{ii}^{(1)} & \cdots & a_{ij}^{(1)} & \cdots \\ & \vdots & & \vdots & \\ \cdots & a_{ji}^{(1)} & \cdots & a_{jj}^{(1)} & \cdots \\ & \vdots & & \vdots & \end{pmatrix} \tag{53}$$

更新を受ける行列要素は，第 i 行，第 j 行， 第 i 列，第 j 列に限られるので，更新される要素の位置が分かるように示した．$R_1^T AR_1$ で更新される行列要素の計算式を示す．

$$\begin{aligned} a_{ii}^{(1)} &= c^2 a_{ii} + 2cs a_{ij} + s^2 a_{jj} \\ a_{ij}^{(1)} &= (c^2 - s^2) a_{ij} - cs(a_{ii} - a_{jj}) = a_{ji}^{(1)} \\ a_{jj}^{(1)} &= c^2 a_{ii} - 2cs a_{ij} + s^2 a_{jj} \\ a_{im}^{(1)} &= c a_{im} + s a_{jm}, \quad a_{jm}^{(1)} = c a_{im} - s a_{jm} \end{aligned} \tag{54}$$

m には i，j 以外の 1 から n が入る．元祖ヤコビ法では，R_k 行列として $A^{(k-1)}$ の非対角成分の絶対値が最大の $a_{ij}^{(k-1)}$ を探し $x_i x_j$ 平面を選ぶ．$a_{ij}^{(k-1)}$ をゼロにするためには，計算式 (54) より，

$$a_{ij}^{(1)} = (c^2 - s^2) a_{ij} - cs(a_{ii} - a_{jj}) = 0 \tag{55}$$

を満たせばよいことになる．回転角 θ は

$$\phi = \cot 2\theta = \frac{c^2 - s^2}{2cs} = \frac{a_{ii} - a_{jj}}{2a_{ij}} \tag{56}$$

より得られる．$t \equiv \dfrac{s}{c}$ と置くと，この ϕ の定義式は次の 2 次方程式の根として得られる．

$$t^2 + 2\phi t - 1 = 0 \tag{57}$$

この 2 次方程式の小さいほうの根は，絶対値が $\frac{\pi}{4}$ より小さい回転角に対応している．この根は次式で与えられる．

$$t = \frac{\mathrm{sgn}\phi}{|\phi| + \sqrt{\phi^2 + 1}} \tag{58}$$

この t から次のように c と s が決まる．

$$c = \frac{1}{\sqrt{t^2+1}}, \quad s = tc$$

このようにして決めた $x_i x_j$ 平面内での θ の回転を行列 A に繰り返し掛ける（$A^{(k)} = R_k^T A^{(k-1)} R_k$）．連立 1 次方程式のガウスの消去法とは異なって，一度ゼロになった要素が後続の変換で非ゼロに戻るので，収束することを証明しておく．

Proof. S_k を $A^{(k)}$ の非対角項の平方和とする．最初の反復では

$$S_0 - S_1 = \sum_{r,s,r\neq s} a_{rs}^2 - \sum_{r,s,r\neq s} (a_{rs}^{(1)})^2 = 2a_{ij}^2 \tag{59}$$

なので，R_1 による平面回転で行列 $A^{(1)}$ は $A^{(0)}$ よりも対角行列に近い．一方 R_k^T を左から掛ける操作も R_k を右から掛ける操作も直交変換であるから，対角項の平方和は $2a_{ij}^2$ だけ増加する．このように，1 回の反復で非対角項の平方和 S_k は必ず小さくなるので，反復回数を大きくすると対角行列に収束する． □

m 回の変換で非対角項が十分小さな値になり[77]，対角化が完了したとすると，アルゴリズムは次のように記述される．

$$\underbrace{R_m^T R_{m-1}^T \cdots R_3^T R_2^T R_1^T}_{H^{-1}} A \underbrace{R_1 R_2 R_3 \cdots R_{m-1} R_m}_{H} = \Lambda \tag{60}$$

式 (60) の左から H を掛ければ $AH = H\Lambda$ が得られるので H **は固有ベクトルである**．そこで相似変換と同時に I を初期値として $(\cdots(((IR_1)R_2)R_3)\cdots R_{m-1})R_m$ によって固有ベクトル H を求めることができる．

A.2 ヤコビ法の数値例

次数 5 のフランク行列の固有値と固有ベクトルをヤコビ法で求めてみよう．

$$A = \begin{pmatrix} 5 & 4 & 3 & 2 & 1 \\ 4 & 4 & 3 & 2 & 1 \\ 3 & 3 & 3 & 2 & 1 \\ 2 & 2 & 2 & 2 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} \tag{61}$$

元祖ヤコビ法は非対角要素の中から絶対値最大の要素を探したが，計算機で計算する場合は探す手間を省いて，非対角項を 1 行 2 列目の要素から順番にゼロにしてゆく巡回ヤコビ法（cyclic Jacobi method）を用いる．最初の平面回転は $x_1 x_2$ 平面内で $\phi = \frac{a_{11} - a_{22}}{2a_{12}} = -\frac{1}{8}$ である．この演算によって $a_{12} = 0$ となる．値が更新されるのは 1,2 行と 1,2 列である．

$$\begin{pmatrix} 8.531 & 0.000 & 4.234 & 2.823 & 1.411 \\ 0.000 & 0.469 & 0.264 & 0.176 & 0.088 \\ 4.234 & 0.264 & 3.000 & 2.000 & 1.000 \\ 2.823 & 0.176 & 2.000 & 2.000 & 1.000 \\ 1.411 & 0.088 & 1.000 & 1.000 & 1.000 \end{pmatrix} \tag{62}$$

[77] 計算機では実数は浮動小数点演算という有限のビット数で近似される．したがって数学的な意味でのゼロとはすこし異なり，計算機が与えられたビット数で表現可能な値よりもすべての非対角項の絶対値が小さくなったところで「ゼロとなった」と判定して反復を終える．

しかし次の a_{13} をゼロにする演算で a_{12} は非ゼロ 0.125 になる．

$$\begin{pmatrix} 10.823 & 0.125 & 0.000 & 3.435 & 1.717 \\ 0.125 & 0.469 & 0.232 & 0.176 & 0.088 \\ 0.000 & 0.232 & 0.708 & 0.415 & 0.208 \\ 3.435 & 0.176 & 0.415 & 2.000 & 1.000 \\ 1.717 & 0.088 & 0.208 & 1.000 & 1.000 \end{pmatrix} \tag{63}$$

しかし大局的に見ると，a_{12} から a_{45} までをゼロにする 9 回の演算が完了した時点で行列 $A^{(1)}$ は次のようになる．

$$A = \begin{pmatrix} 12.338 & 0.084 & 0.187 & -0.033 & -0.158 \\ 0.084 & 0.307 & 0.021 & -0.037 & -0.052 \\ 0.187 & 0.021 & 1.447 & 0.046 & 0.045 \\ -0.033 & -0.037 & 0.046 & 0.330 & 0.000 \\ -0.158 & -0.052 & 0.045 & 0.000 & 0.578 \end{pmatrix} \tag{64}$$

非対角項の平方和は最初は $S_0 = 100$ であったが，$S_1 = 0.153$ になる．反対に対角項の平方和は 55 から 154.847 に増加する．

この操作を繰り返すと非対角項の平方和は $S_2 = 0.00226$，$S_3 = 6.38 \times 10^{-9}$，$S_4 = 1.15 \times 10^{-24}$ と急速に小さくなり，6 回の反復で計算機の 32 ビットの単精度浮動小数点数のゼロ認識の精度内で対角行列になる[78]．1 回の走査（sweep）は 10 項の非対角項に対して演算するが，ゼロになった項はスキップするので，実際には 60 回ではなく 52 回の平面回転を行った．

$$\Lambda \approx A^{(6)} = \begin{pmatrix} 12.344 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.000 & 0.272 & 0.000 & 0.000 & 0.000 \\ 0.000 & 0.000 & 1.449 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.000 & 0.353 & 0.000 \\ 0.000 & 0.000 & 0.000 & 0.000 & 0.583 \end{pmatrix} \tag{65}$$

固有値は大きい順には並ばないので，対角項を大きい順に並べて $\lambda_1 = 12.344$，$\lambda_2 = 1.449$，$\lambda_3 = 0.583$，$\lambda_4 = 0.353$，$\lambda_5 = 0.272$ を得る．

$X^TAX = \Lambda$ の両辺の逆（inverse）をとると $(X^TAX)^{-1} = \Lambda^{-1}$ であり，左辺は $X^TA^{-1}X$ だから，逆行列の固有ベクトルは元の行列の固有ベクトルに一致する．実践的ではないが，固有値と固有ベクトルが分かっていれば，A の逆行列を求めるのに式 (65) の逆行列数を求めて（対角項の逆数を求めて）$A^{-1} = X\Lambda^{-1}X^T$ と計算してもよい．

ヤコビ法によるフランク行列の固有値を求める symeig.BAS プログラムを BASIC プログラム例に含めた．また，c++ では，ratutil.cpp に，倍精度で固有値と固有ベクトルを求める関数 `jacobev` を含めた．

[78] ヤコビ法は固有値に重なりがない場合には 2 次収束する．

参考文献

[1] D. Knuth. *The Art of Computer Programming, Volume 2, Third Edition.* Addison-Wesley, 1998. 有澤　誠，和田英一監訳：The Art of Computer Programming, Third Edition，株式会社アスキー，2004.

[2] *http://hp.vector.co.jp/authors/VA008683/.*

[3] 飯高 茂. **パソコンで開く数の不思議世界**. 岩波ジュニア新書, 2004.

[4] 寒川 光. 有理数線形代数計算における有理数 blas の提案. In *HPCS2014* **論文集**, pages 57–64, 2014.

[5] W. H. Press, Teukolsky S. A., Vetterling W. T., and B. P. Flannery. *Numerical Recipes in Fortran, second edition.* Cambridge University Press, 1992.

[6] J. H. Wilkinson. *Algebraic Eigenvalue Problem.* Oxford University Press, 1965.

[7] 寒川 光, 藤野清次, 長嶋利夫, and 高橋大介. *HPC* **プログラミング**—*ITText* **シリーズ**. オーム社, 2009.

[8] 宮西正宣（他）. **高等学校–数学** *B.* 新興出版社啓林館, 2007.

[9] D. Patterson and J. Hennessy. *Computer Organization & Design: The Hardware/Software Interface, fourth edition.* Morgan Kaufmann Publishers, Inc., 2011. 成田光彰訳: コンピュータの構成と設計–ハードウェアとソフトウェアのインターフェース–第 4 版,（上）（下），日経 BP 社 (2011).

[10] D. Goldberg. What every computer scientist should know about floating–point arithmetic. *ACM Computing Survey*, 23(1), 1991.

索 引

2 次以上の因子探し, 40
2 次以上の因子探索ループ, 49

askinty, 34

bisect, 46
bisectrf, 66
Blue Bene/Q, 7

chkinc, 36, 45
chkres, 60
Clientmain.c, 57
comb, 32
Combination.BAS, 31
Cray-1, 7
CT2D, 56

derogatory, 10
dfcand, 48
do while, 39
dvsrch.cpp, 25
dvsrch1.cpp, 25
dvsrch2.cpp, 25

eigv.c, 56
elementary symmetric function, 29
ELMHES, 10
epsint, 33, 45

FACTORIZEONLY, 54
formatprn, 45

GCD で割る, 22
getpolynomial, 35

HanoiStackMR.BAS, 51
Horner, 36, 38
Horner3, 38

inty, 21
INTYDOUBLE, 49

jacobev, 81

Knuth, 4

LCM, 8
longint, 23
`longint` 型, 6
LUdecomp, 5
LU 分解, 5

mattype, 57
mulvieta, 51

NewtonSDFig.BAS, 38
non-trivial solution, 15
normalization, 23
NormalMode.c, 56
nxtcmb, 31, 33

polydivchk, 35
polytexform, 35
polytexformint, 35
polytexformr, 35
putpolynomial, 34

rational, 23
`rational` 型, 6
roundrat, 67

S13Free.dat, 57
S13Free.txt, 57
S13freeA2.chk, 60
S13freeK2.chk, 60
S13freeMinvK2.chk, 60
setecand, 39
setecanint, 48
solhmg, 68
solveSTATIC.c, 56
squarefree, 12
symeig.BAS, 81
synthetic division, 37

trusgenorm.f, 57

Vieta's formula, 29
vietaterm, 33
vietatermschk, 52

while, 39
Wilkinson, 10

ガウスの消去法, 5

基数, 6
基本対称式, 29
局所的なモード, 19

区間算術演算, 24
組立除法, 37
グラフ・ラプラシアン行列, 19

結合則, 6

5 点差分, 12
根と係数の関係公式, 9, 21

再帰呼出し, 50
最小公倍数, 8, 22

縮重, 10
縮小反復, 24
十進 BASIC, 4
循環小数, 22
乗算の計算時間, 53
冗長性, 22

正規化, 23
整数性, 9
整数性判定, 21, 43
精度改良, 48
零副対角項, 10

相似変換, 9

多桁数, 6
多重度, 14

直交, 15

デカルトの符号法則, 14

特性多項式, 8
特性方程式, 8

2 分法, 9, 46

熱伝導問題, 12

必要桁の抽出, 43

符号変移, 14
フランク行列, 5
プロシージャフレーム, 51
フロベニウス標準形, 11
フロベニウス変換, 11
分母 2 べき, 22
分母 2 べきに丸める, 23
分母 2 べきは保存, 23

ヘッセンベルグ変換, 9

包含, 9, 24

無平方, 8

モニック多項式, 29

有意桁, 22
有理数, 6
有理数モード, 4
有理標準系, 11

ループ変換技術, 31