数値計算と誤差の話

~ 浮動小数点演算はどれくらい信用できるか ~

渡部 善隆 *

この解説の主旨は、次の文章に要約されています。

> Significant discrepancies (between the computed and the true result) are very rare, too rare to worry about all the time, yet not rare enough to ignore.
>
> by W. M. Kahan

数値計算をとりまく計算機の発展はものすごいものがあります。今の時点でもっとも高性能な計算機を *supercomputer* [1] といいます。現在の supercomputer の主流は、ベクトル計算機といわれるものです。さらに、次世代の supercomputer の座を狙う (超) 並列計算機や並列ベクトル計算機の研究・開発を各メーカーが (不況の中) 膨大な予算を割り当てられながら進めています。

これらの supercomputer の進化すべき方向は「高速化」と「大規模化」です。各メーカーが発表するカタログを見ると、“GFLOPS[2]” とか “GBYTE[3]” とかいった、高速かつ大容量を宣伝する文句が踊っています。

しかしながら、数値計算を行なう人なら誰もが心配する、計算の「精度」については、カタログにはほとんど書いてありません。また、数値計算と精度との関係が、現在どのような状況にあるかご存知ない方も多いと思われます。

何を隠そう、私もよく知りません。ただし、次のことは言えます。一般に、皆さんが数値計算に使われている浮動小数点演算ソフトウェア (Fortran, C, C++, Pascal, etc.) では、計算の際に発生する誤差の厳密な把握は (現在のところ) できません。しかし、誤差の範囲を厳密に評価する数値計算の品質保証付きのソフトウェアが既に開発・利用され、数値シミュレーションだけでなく、微分方程式の解の存在を証明するような解析学的な分野においても、成果が次々に報告されています。

日本で数値計算の品質保証の大切さが研究者の間で意識されだしたのは、最近になってからです。数値計算の品質保証に関する研究は、ヨーロッパ、特にドイツで精力的に行なわれています。もし、読者のどなたかがヨーロッパの学会で、例えばある非線形方程式の数値計算などの研究発表をされる場合、精度保証付のソフトウェアを使って誤差をきちんと評価した結果を出してないと、(ましてや単精度の計算などだったら) ブイブイ突っ込みが来る可能性のある分科会もあるので、特にドイツに行かれる方はご注意下さい。

---

*九州大学大型計算機センター・研究開発部

[1] JIS 規格に準拠した某 F じつう社では、これを「スーパコンピュータ」とカタカナにしています。一方、新聞では「スーパーコンピューター」とよんでいます。ここでは、面倒なのでそのまま英語を書きます。

[2] FLOPS は一秒間に何回浮動小数点演算ができるかを示す指標。G(ギガ) なので、例えば 5GFLPOS は、浮動小数点演算が一秒間に 50 億回できるという意味。ただし絶対的な計算機の優秀さを示すものではありません。

[3] この場合、メモリーやディスクの容量の大きさのこと。1GBYTE=1024MBYTE=1024^3BYTE

1　計算機で普段何をやっているのか

まず、初心に返って、「計算とはなんぞや」を考えます。『計算』の意味を国語辞典で調べてみましょう。

> けいさん【計算】
> （1）はかりかぞえること。勘定。また、見積り。考慮。
> （2）演算をして結果を求め出すこと。
>
> 『広辞苑』より

二番めの『演算』の意味を調べてみます。

> えんざん【演算】
> 数式の示す通りに所要の数値を計算すること。運算。
>
> 『広辞苑』より

となっています。ものごとを厳密に追究する人 ($\simeq$ あげ足とりのうまい人) でしたら、広辞苑に向かって

> 『計算』の説明で『演算』を使い、『演算』の説明で『計算』を使うなんて、どないなっとんねん

と文句をつけるかもしれません。しかし、このような “どうどうめぐり” は、社会生活における、例えば長い会議[4]や煮詰まった男女の会話[5]や飲み屋での酔っ払い同士の愚痴や口喧嘩[6]などによく出てきます。従って、気にせずに先に進みます。

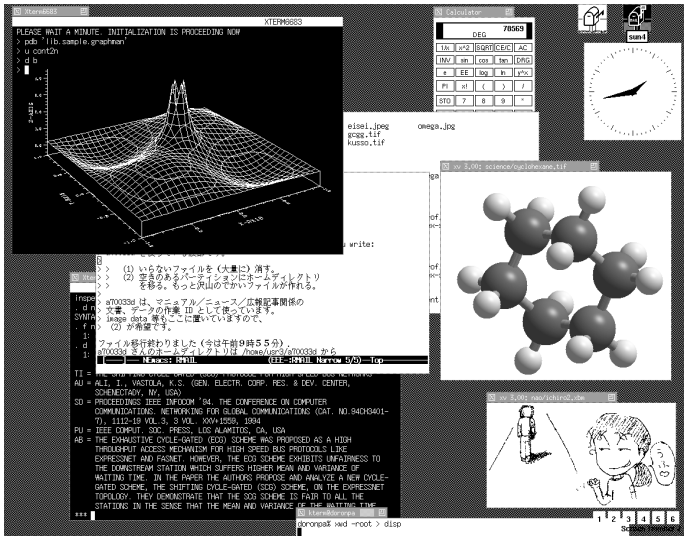
本稿では、計算する (compute) とは “数をいじくること” だと勝手に定義します。いわゆる数値計算のことです。数値計算とは、四則演算 ($+, -, \times, \div$) を基本とした数の結合をごちゃごちゃやることです。従って、計算機 (computer) は “数をいじくる機械” となります。

この決めつけは、情報処理の世界から見れば極めて狭い定義です。なぜなら、近くにある computer で普段みなさんが何をやっているか、先輩 (後輩) や先生 (学生) の様子を見て下さい。数値計算以外に、おおよそ次のことに大量の時間を費やしているはずです。

- AIR などを使ったデータベースの検索。
- TEX や一太郎などを用いた論文作成、文章作成。お絵書き。
- ゲーム。特に Graphics が綺麗なもの。
- イメージデータ、特に動画を画面に出して喜ぶ。かつ、そのコレクションの整備。
- 実際は使いもしないのに make ばっかりやっている。
- 電子メールで他人の悪口をバンバン書く。

これらをまとめると「データの処理を行なっている」とでもいえるでしょう。

[4] 特に午後のねむい会議。
[5] 特に深夜の電話。
[6] 特にセンター職員。



よく見られる端末の風景

もちろん supercomputer の世界では、四則演算を基本とした数値計算が圧倒的な主役です。理由は極めて単純で、 supercomputer はもともと大規模科学技術計算を高速におこなうために設計された機械だからです。

文字を打つことを専門にしたワープロは、電卓機能やファイルの管理が可能なものが多く、computer といえないこともないのですが、これを計算機とは呼べないでしょう。また、任天堂を世界へ飛翔させたファミコンも、 computer と名がつきながら完全なゲーム専門機であり、とても計算機とは呼べません。どうやら「計算機 =computer」という訳語の対応関係自体が実情に合わなくなっているみたいです。

それでは以下、数値計算に関する話題を、「精度保証」というみなさんにはあまりおなじみでない切り口から見ていくことにしましょう。

2 Computation

誤差の取り扱いを中心に考えると、数値計算は次のように分類できます。

Computation on the computer

Algebraic Computation
- computing without errors
- ring of integers or field of rational numbers
- algebraic number fields

Mathematica
CALC

Numerical Computation
- approximation of the exact result
- floating-point arithmetic

Fortran
C++
Pascal

Verified Inclusions
- approximation with a guaranteed error bound
- interval arithmetic
- fixed point theorem

ACRITH
Pascal/XSC
PROFIL

以下、この 3 つの Computation について、簡単な説明を加えます。

Algebraic Computation

algebraic computation とは、計算を 誤差なし で行なう手法です。誤差なしという意味を強調するため、 *Error-Free computation* とも呼ばれています。

数値計算における Algebraic computation の代表的なものは、有理数演算です。これは、すべての計算を有理数でやってしまおうというものです。 3 節以降で述べるように、浮動小数点演算の最大の欠点は、実数を近似してしまうことです。それに対して、有理数演算はこの欠点がなく、厳密な誤差なし計算が可能です。

簡単な例として、 10 進数表現の計算機[7]の中をのぞいてみます。 10 進数では

$$\frac{1}{3} \sim 0.3333333$$

といったように、適当な桁で打ち切って近似しないと表現できない実数がたくさんあります。有理数演算は、これを有理数のまま

$$\frac{1}{3} + \frac{1}{9} = \frac{4}{9}$$

などと計算します。これなら、入力のデータをすべて有理数で与えておけば、 exact な計算が可能になります。

ただし、有理数演算には難点があります。たとえば、 $\pi, \sin\frac{1}{3}, \log\frac{11}{13}$ など、無理数の有理数表現はできません[8]。また、演算の途中で

$$\frac{9689}{11213} + \frac{9941}{19937} = \frac{304638026}{223553581}$$

のように、分子 / 分母がむやみに大きくなって、最後にはメモリーの許容量を越えたり、計算時間が膨大になったりする場合があります。

この対策として、増大した有理数を分母 / 分子が小さくなるように区間として包み込みながら演算をすすめるシステム [11] なども構築されていますが、これは次の次に出てくる Verified Computation になってしまいます。

また、 $a, b \geq 0$ を未知数としたまま

$$\int_0^\infty (e^{-\frac{a^2}{x^2}} - e^{-\frac{b^2}{x^2}})dx = (b-a)\sqrt{\pi}$$

などと積分計算を行なうことが可能な数式処理システムも、台形公式などの近似公式を用いず、正確に数学的変形ができるという意味で、誤差なしの Algebraic Computation と言えます。もっとも、数式処理がちゃんと解けないと役にたちません。

Numerical Computation

Numerical Computation とは、適当な演算を用いて exact な結果の 近似 を行なう手法です。手法は浮動小数点演算が一般的です。実は浮動小数点演算に対して、固定小数点演算という手法もありますが、これはあまり使われていません。

[7] 実際の計算機は 2 進数 (つまり 0 と 1 の世界) が主流です。

[8] なぜなら、有理数と無理数の共通部分は空集合だからです。

浮動小数点演算の大きな特徴は、exact な計算をあきらめて近似計算に徹した分だけ演算処理が極めて高速であり、さらに変数が必要とする記憶領域も少なくてすむことにあります。

さて、数値計算をやる方がたまにされている勘違いは、

> 【誤解その一】
> 計算させようというモデルがもともと連続量の近似なんだから、
> 浮動小数点演算の多少の誤差なんて、まあ、どうでもいいや。

とたかをくくられていることです。中には、極端な人になると、次のような思いきり間違った認識をされている場合もあります。

> 【誤解その二】
> 浮動小数点演算は、どんなときでも単精度演算で小数点以下 7 桁、
> 倍精度演算は小数点以下 15 桁までの数値精度を厳密に保証する。

実は Numerical Computation で生じる誤差は、厳密な精度の保証どころか、場合によってはもとのモデルをたてた意味すら失わせるほど増大する可能性があるのです。この解説は、上の「まあ、どうでもいいや」と思っている人に「どうでもよくない」ことを警告することが主眼です。

Verified Inclusions

最後の *Verified Inclusions* (Verified computation) は、「誤差なし」の Algebraic Computation と「近似」の Numerical Computation の中間に位置する手法で、区間演算 (interval arithmetic) をその基礎においています。

区間演算とは、数値をある幅をもった実数の集合としてとらえ、区間の四則演算を定義することで Numerical Computation によって発生する 誤差の厳密な把握 を行なう手法です。

例えば $\sqrt{2}$ は無理数で、計算機で厳密に表現することはできません。区間演算ではこれを次のように表現します。

$$\sqrt{2} = 1.4142\cdots \in [1.414, 1.415] \equiv \{x \in \mathbb{R}; 1.414 \leq x \leq 1.415\}$$

Verified computation が可能なソフトウェアでの計算は、出力が exact な値を包み込んだ区間の形で出力されます。下は、1.0 を包み込んだ出力例です。

```
x=[0.9999999999999993, 1.0000000000000001]
```

区間は、上のように実数の集合として表現されますが、その区間に必ず真の値が存在するという (数学的に厳密な) 保証を与えてくれます。従って、正確な数値結果、即ち計算の品質保証が欲しい人にとっては、大変役に立ちます。その他、Verified Inclusions についての詳細は、[3] を御覧ください。

Verified Inclusions の課題は、この区間がどれくらい 狭く 実現できるかです。また、高い精度を保証しつつ浮動小数点演算に劣らぬ高速演算ができるのかという、コストの問題もあります。

区間演算は誤差の最悪の結果を想定して包み込みを行ないますので、問題によっては包み込みの区間が増大し、実用的な値が得られないケースも生じます。

品質保証用数値計算ソフトは、既にいくつかが開発・市販され高い評価を受けています。また [11] では、有理数演算と区間演算とをあわせた精度保証付きシステムを実現しています。

3　浮動小数点を数える

浮動小数点演算は有限の世界です。浮動小数点の集合は、実数のように連続でもないし、有理数のように無限集合でもありません。実数と有理数の間には、次の関係が成り立ちます。

【定理】
実数 α に収束する有理数列が存在する。さらに、 α は次のように表現可能である。

$$\alpha = a_0 + \frac{c_1}{10} + \frac{c_2}{10^2} + \cdots + \frac{c_n}{10^n} + \cdots$$

ここで、 a_0, c_n は整数、 $0 \leq \alpha - a_0 < 1, 0 \leq c_n < 10$ をみたす。

これを 「10 進数としての α の表現」といいます。定理の証明は、名著 [1] をお読み下さい。 10 の代わりに 1 よりも大きい任意の自然数 t を使った、実数 α の t 進展開も可能です。

$$\alpha = a_0 + \frac{c_1}{t} + \frac{c_2}{t^2} + \cdots, \qquad (0 \leq c_n < t) \tag{1}$$

整数 m を適当に定め (1) を t^m で括り出すと、 0 以外の実数は次のように表現可能です。

$$\alpha = \pm(\frac{x_1}{t} + \frac{x_2}{t^2} + \frac{x_3}{t^3} + \cdots)t^m \tag{2}$$

各 x_i は整数であり、 $0 < x_1 < t, 0 \leq x_i < t \ (i = 2, 3, \cdots)$ です。

計算機の内部では、もちろんこのような表現はできません。展開は一般に無限に続く一方で、各 x_i の情報を格納する記憶領域にも制限があります。

浮動小数点数 (floating point number) とは、 (2) の表現を有限桁で打ち切った実数の近似のことです。例えば n 桁で展開を打ち切る場合、実数 α の浮動小数点表現 $\bar{\alpha}$ は下の様になります。

$$\bar{\alpha} = \pm(\frac{x_1}{t} + \frac{x_2}{t^2} + \cdots \frac{x_n}{t^n})t^m \tag{3}$$

また、 m の値も無限に大きくできません。そこで、自然数 m_L, m_U を上限・下限として、 $-m_L \leq m \leq m_U$ という制限がつきます。浮動小数点数は、この 4 つの自然数 n, t, m_L, m_U を使って正確に数えることができます。つまり正の場合、負の場合がそれぞれ $(t-1)t^{n-1}(m_L + m_U + 1)$ 個、これにゼロを加えた合計 $2(t-1)t^{n-1}(m_L + m_U + 1) + 1$ 個が、正確な浮動小数点の数になります。

計算機のシステムによって n, t, m_L, m_U の値は変わります。例えば、 Fujitsu M1800/20 の UXP/Fortran77 EX システムの倍精度実数型データは

$$\underbrace{16}_{t} \text{ 進数 } \underbrace{14}_{n} \text{ 桁、 } \underbrace{-64}_{m_L} \leq m \leq \underbrace{63}_{m_U}$$

です。従って浮動小数点の数は $2 \times 15 \times 16^{13} \times (64+63+1)+1 = 1729382256910270464 1$個となります。

M1800/20

4　精度の損失

計算機の内部で、数値を一定の桁で近似することを丸め (round-off error) と呼びます。丸めを n 桁で行なえば、当然その先の $n+1$ 桁以降の情報が失われ、誤差が混入します。この誤差を丸め誤差 (round-off error) といいます。 10 進数展開では、次のようになります。

$$\underbrace{\sum_{i=0}^{\infty}\frac{c_i}{10^i}}_{\text{実数値}} = \underbrace{\sum_{i=0}^{n}\frac{c_i}{10^i}}_{n\text{ 桁の丸め}} + \underbrace{\sum_{i=n+1}^{\infty}\frac{c_i}{10^i}}_{\text{丸め誤差}}$$

浮動小数点演算では、この有限への近似の影響から、まれにプログラムを組んだ人が期待する計算値と異なる結果を出力することがあります。ただし、その異なり具合 (誤差の範囲) が、プログラムを組んだ人の許容する範囲に収まっているかという問題があります。そして、許容範囲は、計算の目的によって変わります[9]。

丸め誤差による計算上のトラブルを防止するには、演算桁数、すなわち丸めの数 n を大きくとるのがもっとも簡単、かつ有効な方法です。一部の計算機やソフトウェアでは、単精度でも充分な桁数を確保するものもありますが、特に九州大学大型計算機センターの Fortran システムを利用する方は

数値計算に倍精度演算を使用するのは常識

と思って下さい。

ただし、ここで慌てて、対偶[10]は必ずしも真でないことを、いい添えておきます。誤差の許容量、およびメモリーや演算速度の関係等、様々な事情があって、単精度を用いていらっしゃる場合もありますので...

とかいいながら、 [2] から引用しておきます。

> 以前は多倍精度演算を用いると「何倍もの計算時間がかかる、何倍もの記憶場所をとる、だからなるべく使うな」といわれたこともあったが、今日ではコンピュータの性能に充分な余裕があるので、計算時間や記憶容量などを心配せずに多倍精度演算を用いることができる。大型コンピュータの場合には倍精度計算も単精度計算のせいぜい 2 倍くらいである。計算時間を節約したために誤った結論を出して技術者の信用を失墜することのないよう、演算桁数は充分長くとっておくことを勧めたい。
>
> 戸川 隼人『数値計算』より

次に気をつけるべきことは、数学的に同値な公式だからといって、計算して得られる結果が同じとは限らないという恐ろしい事実です。もちろん、浮動小数点が実数の「近似」に過ぎない有限の表現であるということが原因です。

ということは、連立一次方程式の解法や、固有値問題、高速フーリエ変換などの、数値計算の手法がある程度確立された問題については、できるだけ既成のサブルーチンを利用した方が無難だということです。

例えば、当センターで提供している SSL II や NUMPAC は、数値計算の専門家が丸め誤差の影響を最小にするように工夫をこらして作成したライブラリが揃っています。自分で意気

[9] 小数点以下 3 桁くらいがあっていれば良い場合や、完璧な精度で結果を要求する場合など様々です。

[10] 『常識がない奴が単精度で数値計算をする』

込んでプログラミングするのも結構ですが、論文の出来を左右するような大事なプログラムを組まれる場合は、なるだけ評価の確立したライブラリを使用した方がいいでしょう。

以下、ひとつだけ例をあげます。 2 次方程式

$$ax^2 + bx + c = 0, \quad a \neq 0$$

の解は、高校時代暗記したように、公式

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

で与えられます。ところが、この公式を直接使ったプログラムを組むと、 b と $\sqrt{b^2 - 4ac}$ の関係によっては、精度が極端に悪くなる場合があります。 [4] に、かなりしつこい解説がありますので、興味のある方はそちらを御覧いただくとして、ひとつだけ精度が悪くなる例をあげます。

任意の自然数 n について

$$a = 1, \quad b = -(10^n + 10^{-n}), \quad c = 1$$

の場合、解は正確に

$$\begin{aligned} x_1 &= 10^n \\ x_2 &= 10^{-n} \end{aligned}$$

です。 $n = 5$ の場合は

$$\begin{aligned} x_1 &= 100000 \\ x_2 &= 0.00001 \end{aligned}$$

になります。そこで、先の公式をそのまま用いたプログラムと、精度の損失を考慮した SSL II のサブルーチン RQDR とを単精度で比べてみます。コンパイラは Fujitsu M1800/20 の Fortran77 EX です。

解の公式 VS. SSL II サブルーチンのプログラムは以下の通りです。

```
      program niji
       real    a/1.0/,b,c/1.0/,x1,x2
       complex z(2)
       b  = -1.0E5-1.0E-5                          <--- n=5
       x1 = ( -b + sqrt(b**2 - 4*a*c) )/(2.0*a)    <--- 解の公式
       x2 = ( -b - sqrt(b**2 - 4*a*c) )/(2.0*a)
       print*,x1,x2
       call rqdr(a,b,c,z,icon)                     <--- SSL II の使用
       print*,z
      end
```

結果は次のようになりました。

```
      x1=100000.0    x2=0.0                  <--- 解の公式
      x1=100000.0    x2=0.0000099999997      <--- SSL II
```

解の公式をそのまま適用した場合、 x_2 の値が 0 に落ちてしまうのに対して、 SSL II を使用した場合、精度の損失を防ぐ手法をとり入れているため $(\mathrm{cf.}[5])$、真の値に近い数値を得ています。

単精度での計算の場合、 2 次方程式などの簡単な公式についても精度の欠落が待ち構えています。特に、得られた結果を再度利用して計算を進める反復法をプログラミングされている方は、精度の悪化に加えて、 IF 文判定の間違えや、ゼロ割りを起こしたりなどの、致命的な誤りをおかしている かも しれません。

上のプログラムでは、少なくとも先ほどの忠告 $([2])$ を受け入れて、倍精度演算以上に修正すれば、結果がゼロに落ちるという危険は防止できます。参考までに倍精度、 4 倍精度演算で行なった結果をあげます。

```
x1= 99999.999999999985                                  倍精度
x2= 0.000013385357559

x1= 99999.99999999999999999999999980          4 倍精度
x2= 0.00001000000000000000000000000626847810
```

$\longrightarrow$ time

論文提出後、計算ミスを発見した場合の様相の変化
(左から驚愕之図 / 危惧之踊 / 嘆息抱膝之図)

5　筆算 vs. computer

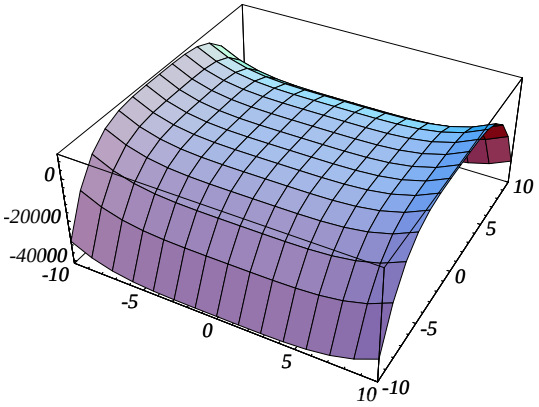
続いては、カタログ上では世界屈指の高性能を誇る汎用計算機に搭載された Fortran コンパイラに、紙と鉛筆で敢然と挑戦してみます。

【問題】

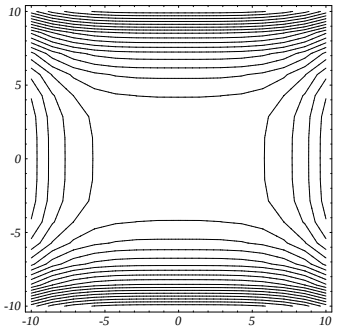
次の値を計算せよ

$$x^4 - 4y^2 - 4y^4 \quad \text{for} \quad x = 665857.0 \quad \text{and} \quad y = 470832.0$$

ちなみに $z = x^4 - 4y^2 - 4y^4$ は次のような馬の鞍の様な形になっています。



$z = x^4 - 4y^2 - 4y^4$



等高線

Fortran での計算

問題の通りに Fortran プログラムを組んでみます。念のため、単精度と倍精度で値を出力させます。

```
program ex1
 real    xr/665857.0E0/,yr/470832.0E0/
 real*8  xd/665857.0D0/,yd/470832.0D0/
 print*,'single precision : ',xr**4 - 4*yr**2 - 4*yr**4
 print*,'double precision : ',xd**4 - 4*yd**2 - 4*yd**4
end
```

わずか 5 行のプログラムです。実行結果は...

```
single precision : 7.2057594e+16
double precision : -16777216.000000000
```

となりました。単精度と倍精度で全く値が違います。ためしに、二番目と三番目の項を入れ換えてみます。

```
program ex2
 real    xr/665857.0E0/,yr/470832.0E0/
 real*8  xd/665857.0D0/,yd/470832.0D0/
 print*,'single precision : ',xr**4 - 4*yr**4 - 4*yr**2
 print*,'double precision : ',xd**4 - 4*yd**4 - 4*yd**2
end
```

もちろん、値は先ほどと同じになる筈です。ところが...

```
single precision : 7.2056709e+16
double precision : -4891648.0000000000
```

単精度・倍精度ともに、入れ換える前と異なる結果が出てしまいました。これは、演算の結果が単精度変数および倍精度変数の持つ情報量ではカバーできなくなったために起きたものです。もう少し具体的に、 x^4 がどのように表現されているか出力させます。

単精度	:	19657297 0000000000000000	=	1.9657297×10^{23}
倍精度	:	19657300600455819 0000000	=	$1.9657300600455819 \times 10^{23}$
本当の値	:	19657300600455819 4713601		

本当の値と比較しておわかりのように、下線部以外の必要な情報が落ちています。従って、これらの情報が欠落した数同士の引き算や足し算を繰り返すと、全く異なった結果が出てくることになります。

CALC で本当の値を求める

情報を欠落させないためには、あくまで近似計算である浮動小数点演算を使わずに、 exact な計算 (algebraic computation) をやってしまえばいいわけです。 exact な計算として、ここでは二つの方法を用いて Fortran コンパイラが失敗した計算に再度チャレンジしてみます。一つは有理数演算、そしてもう一つは筆算です。

まず、有理数演算ソフトを使って計算を行ないます。ここでは、 CALC というソフトに登場してもらいます。以下、©の紹介です。

> CALC – C-style arbitrary precision calculator.
> calc version: 1.25.0 - k1.0
> Copyright (c) 1992 David I. Bell
> Modified 1993 by Masahide Kashiwagi

CALC は有理数演算を基本とし、様々な数学関数を備え、任意の精度計算が可能な電卓プログラムです。現在のところワークステーションでの動作で、 UXP, MSP は未サポートです。

早速計算してみます。

```
01> 665857**4 - 4*470832**4 - 4*470832**2    ↵
        1                                          <--- 答え
02> 665857**4    ↵                                 <--- 一応個別に計算してみる
        196573006004558194713601
03> 4*470832**4    ↵
        196573006003671463624704
04> 4*470832**2    ↵
        886731088896
05> 196573006004558194713601 -196573006003671463624704 -886731088896    ↵
        1                                           <--- 答え
```

御覧のように計算結果は 1 になりました。これで正解です。

手計算で本当の値を求める

次は筆算です。上の計算は、基本的に日本の小学校で必ず暗誦させられる「九九 (くく)」さえ知っていれば解ける問題です。そこで、センター内の暇そうにしていた女の子を捕まえて計算させてみました。ただし、親切のため問題を変えています。

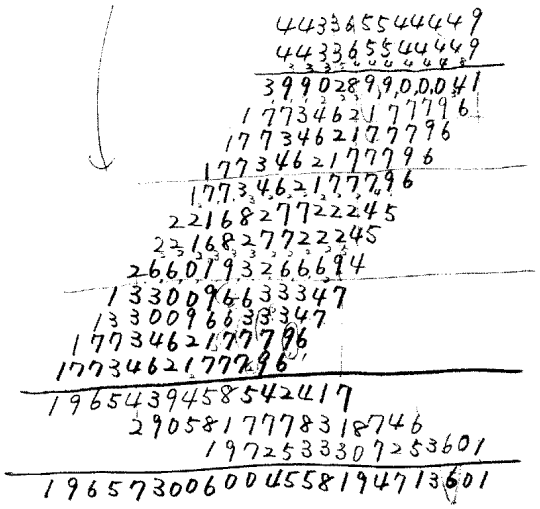
【問題 (修正版)】

次の値が 1 になることを示せ。

$$x^4 - 4y^2 - 4y^4 \quad \text{for} \quad x = 665857 \quad \text{and} \quad y = 470832$$

計算を頼んだ人は、まず $x^4,\ y^2,\ y^4$ を個別に計算しようとして見事に粉砕されました。原因は集中力が途切れたため、足し算のときに「くりあげ」を忘れてしまったためです。

以下は、しかたがないので、他の人にやってもらった x^4 の計算の一部です。



$x^2 \times x^2$ の計算の模様

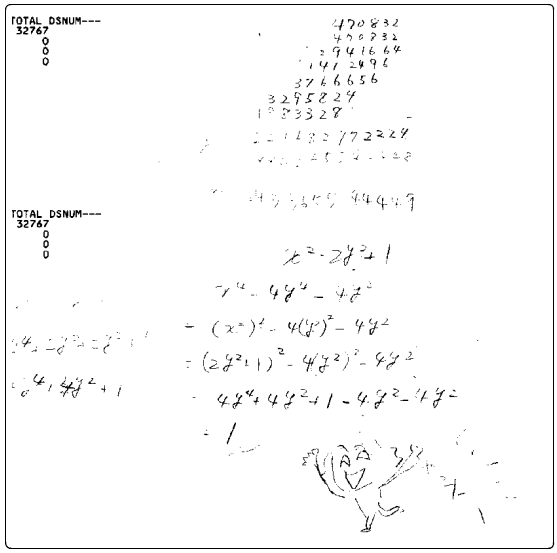
実際のところ、 $x^4 - 4y^2 - 4y^4$ が 1 になることがわかっているので、因数分解したくなるのが人情です。やってみると、

$$x^4 - 4y^2 - 4y^4 - 1 = (x^2 + 2y^2 + 1)(x^2 - 2y^2 - 1)$$

となります。 x^2, y^2 は非負より $x^2 + 2y^2 + 1 > 0$ です。従って、紙の上で $x^2 - 2y^2 = 1$ を示せば解答になります。これだと 2 乗 $(x^2,\ y^2\ )$ の計算で事足りるので、何とか出来そうです。

「うーうー」うなっている解答者の後ろで、いち早くこれに気付いた新婚の助教授は「ふふふふ、できた~」と喜んで、解答者にいやがられていました。

ヒントを助教授からもらった解答者も、ようやく解答を得ました。めでたいことです。



解答の一部と歓喜の図

閑話

ところで、手計算にも出てきた x^4 の計算結果である

$$1965730060045581947136 01$$

を「読め」と言われた場合、みなさんはちゃんと読めるでしょうか。つまり、桁を数えればおわかりのように、“兆”より上の単位をご存知か、ということです。

漢字で書くと、 $1965730060045581947136 01$ は

> 一千九百六十五垓七千三百京六千四兆五千五百八十一億九千四百七十一万三千六百一

です。ひらがなで書いてみます。ぜひ、息つぎなしで一気に読んでみて下さい。

> いっせんきゅうひゃくろくじゅうごがいななせんさんびゃくけいろくせんよんちょうごせんごひゃくはちじゅういちおくきゅうせんよんひゃくななじゅういちまんさんぜんろっぴゃくいち

「兆」の上は「京」、その上は「垓」になります。何かの話のたねになるかもしれませんので、その他の単位の一覧を紹介します。出典は平凡社の『世界大百科事典』によっています。単位は全て仏教用語からきています。単位が大きく (小さく) なるに従って、浮世から段々はなれていく模様を感じとられて下さい。

もちろん、すでに悟られている方は感じないと思います。

浄	じょう	10^{-23}	沙	しゃ	10^{-8}	京	けい	10^{16}
清	せい	10^{-22}	繊	せん	10^{-7}	垓	がい	10^{20}
空	くう	10^{-21}	微	び	10^{-6}	杼	じょ	10^{24}
虚	きょ	10^{-20}	忽	こつ	10^{-5}	穣	じょう	10^{28}
六徳	りっとく	10^{-19}	糸	し	10^{-4}	溝	こう	10^{32}
殺那	せつな	10^{-18}	毛	もう	10^{-3}	澗	かん	10^{36}
弾指	だんし	10^{-17}	厘	りん	10^{-2}	正	せい	10^{40}
瞬息	しゅんそく	10^{-16}	分	ぶ	10^{-1}	載	さい	10^{44}
須臾	しゅゆ	10^{-15}	一	いち	1	極	ごく	10^{48}
逡巡	しゅんじゅん	10^{-14}	十	じゅう	10^{1}	恒河沙	ごうがしゃ	10^{56}
模糊	もこ	10^{-13}	百	ひゃく	10^{2}	阿僧祇	あそうぎ	10^{64}
漠	ばく	10^{-12}	千	せん	10^{3}	那由他	なゆた	10^{72}
渺	びょう	10^{-11}	万	まん	10^{4}	不可思議	ふかしぎ	10^{80}
埃	あい	10^{-10}	億	おく	10^{8}	無量大数	むりょうたいすう	10^{88}
塵	じん	10^{-9}	兆	ちょう	10^{12}			

数の単位一覧

浮動小数点演算の名誉挽回のために

今までの計算結果を見る限り、必ずしも浮動小数点演算が万能といえないことがよくわかります。その理由は、浮動小数点演算が「近似」であるということにつきます。

もっとも、与えられた問題の桁数ならば、精度を 4 倍精度にまで拡張することで、必要な情報が欠損することなく正しい答が得られます。プログラムを変更してみます。

```
program ex3
 real    xr/665857.0E0/,yr/470832.0E0/
 real*8  xd/665857.0D0/,yd/470832.0D0/
 real*16 xq/665857.0Q0/,yq/470832.0Q0/
 print*,'single precision     : ',xr**4 - 4*yr**2 - 4*yr**4
 print*,'double precision     : ',xd**4 - 4*yd**2 - 4*yd**4
 print*,'quadruple precision  : ',xq**4 - 4*yq**2 - 4*yq**4
end
```

実行結果は次のようになります。

```
single precision     : 7.2057594e+16
double precision     : -16777216.000000000
quadruple precision  : 1.0000000000000000000000000000000000
```

4 倍精度になって、ようやく正しい答が得られました。ただし、 4 倍精度にすれば全て計算は大丈夫というのは幻想で、全然正しい答えが出てこない恐ろしい例も作ることができます。次の節ではそれを紹介します。

6　もう少し複雑な計算

【問題】

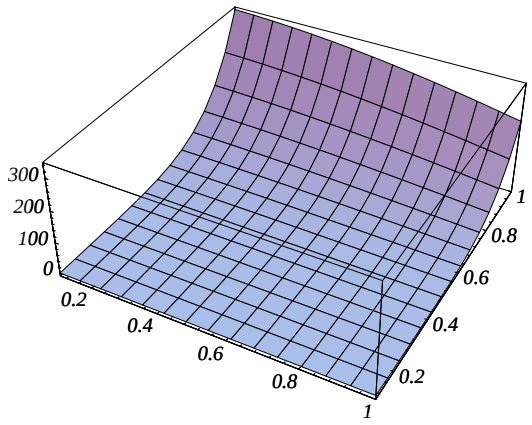
次の値を小数点以下 6 桁まで計算せよ

$$333.75b^6 + a^2(11a^2b^2 - b^6 - 121b^4 - 2) + 5.5b^8 + \frac{a}{2b} \qquad (4)$$

$$\text{for} \quad a = 77617.0 \quad \text{and} \quad b = 33096.0$$

これくらい複雑になると、誰も紙と鉛筆で計算してくれません。b^8 や $a/(2b)$ の手計算には、非常な困難がつきまといます。こうなると計算機の出番です。

ちなみに、$z = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$ を $x > 0,\ y > 0$ の範囲で図にすると、以下のような “べちゃっ” とした曲面になります。



$$z = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$$

Fortran プログラムに細工をする

以下の細工は、1994 年 10 月現在の Fortran コンパイラのバージョン、および計算機 Fujitsu M1800/20 の浮動点形式に依存していますので、全ての計算機で同じ結果が得られるわけではないことをあらかじめご注意下さい。

(4) を変形して、各精度用の関数を作ります。結果的には細工になるのですが、数式変形の上では何の問題もありません。

$$f_1(a,b) = 333.75b^6 + 11a^4b^2 - b^6a^2 - 121b^4a^2 - 2a^2 + 5.5b^8 + a/(2b) \qquad \text{単精度用}$$

$$f_2(a,b) = (333.75 + 5.5b^2)b^6 + a^2(11a^2b^2) + a^2(-b^6 - 121b^4 - 2) + a/(2b) \qquad \text{倍精度用}$$

$$f_3(a,b) = 333.75b^6 + a^2(11a^2b^2 - b^6 - 121b^4 - 2) + 5.5b^8 + a/(2b) \qquad 4\text{ 倍精度用}$$

$$f_1 = f_2 = f_3$$

以下は関数 $f_1,\ f_2,\ f_3$ を用いて作成した、問題の解答用プログラムです。

```
      program ex3
       real    a1/77617.0E0/,b1/33096.0E0/,f1
       real*8  a2/77617.0D0/,b2/33096.0D0/,f2
       real*16 a3/77617.0Q0/,b3/33096.0Q0/,f3
       external f1,f2,f3
       print*,'single precision     : ',f1(a1,b1)
       print*,'double precision     : ',f2(a2,b2)
       print*,'quadruple precision  : ',f3(a3,b3)
C
       function f1(a,b)
       real a,b
        f1=333.75E0*b**6+11.0E0*a**4*b**2 -b**6*a**2-121.0E0*b**4*a**2
     &    -2.0E0*a**2 +5.5E0*b**8 + a/(2.0E0*b)
       return
       end
C
       real*8 function f2(a,b)
       real*8 a,b
        f2=(333.75D0 + 5.5E0*b**2)*b**6 + a**2*(11.0D0*a**2*b**2)
     &  + a**2*( -b**6 -121.0D0*b**4 -2.0D0) + a/(2.0D0*b)
       return
       end
C
       real*16 function f3(a,b)
       real*16 a,b
       f3=333.75Q0*b**6+a**2*(11.0Q0*a**2*b**2-b**6-121.0Q0*b**4-2.0Q0)
     &    +5.5Q0*b**8 + a/(2.0Q0*b)
       return
       end
```

各関数の式は、数式としては同等なので、(変形がわざとらしいのが多少気がかりですが)計算結果も同じ値が一応期待されます。

実行してみます。

```
single precision     : 1.1726036
double precision     : 1.1726039400531785
quadruple precision  : 1.172603940053178631858834904520183
```

アンダーラインの部分の 7 桁は同じ値です。これから、問題の答えは 1.172603 になりそうな気がします。が...

Mathematica での有理数計算

ところが、この浮動小数点演算にも情報の欠損が生じています。真の値は実は マイナスの値 をとります。では、 algebraic computation で exact な計算をしてみましょう。式をみておわかりのように、この計算は有理数でデータが入力できます[11]。従って、有理数演算を用いることで、正しい値を (有理数の形で) 計算することが可能です。

計算ソフトは、さきほどは CALC を使いましたので、こんどは Mathematica [12]で計算します。 Mathematica は、四則演算を有理数で行なうことが可能であり、その結果を小数点形式で任意の桁まで表示できます。ただし、残念ながら Mathematica の汎用計算機版も CALC と同様ありません。

```
In[1]:= a=77617  ↵

Out[1]= 77617

In[2]:= b=33096  ↵

Out[2]= 33096

In[3]:= 33375/100 b^6 + a^2(11 a^2 b^2 - b^6 -121 b^4 -2) +   ↵
        55/10 b^8 + a/(2 b)   ↵

          54767
Out[3]= -(-----)              <--- exact value
          66192

In[4]:= N[%,16]  ↵            <--- 小数点表示

Out[4]= -0.827396059946821    <--- 値
```

問題は「小数点以下 6 桁まで求めよ」でした。 Mathematica によれば、答は -0.827396 となり、これで正解です。

これは極端な例だとしても、例えばこの値の正負によって計算プロセスが分岐するなどの何か大事な目安とした数値計算をしている場合は、その後の処理によっては、とんでもない結果になる可能性もあります。ぜひ、お手持ちのプログラムをチェックされては如何でしょう。

この例は、浮動小数点演算が全く信用できないことを示しているのではなくて、浮動小数点演算は 必ずしも信用できない ことの意地悪な例です。数値計算における精度保証の難しさと大事さが多少ともおわかりいただければ幸いです。

---

[11] $333.75 = 33375/100$, $5.5 = 55/10$ などと有理数表現できます。

[12] Stephen Wolfram により開発された汎用数学処理システム。高度な計算機能、数式処理機能、 2 次元・ 3 次元グラフィックス機能を有する。また、数式は Fortran, C, TEX ソースとして出力可能。グラフィックス出力は PostScript コードで出力される。

7　いくつかの実験

この節では、数値計算においてよく行なわれる演算

1. LU 分解
2. 連立一次方程式の解法
3. 固有値問題

について、各種専用のサブルーチンライブラリで計算を行ない、浮動小数点演算の信頼度をチェックしてみます。

もちろん、結果は例題や解法に依存しますので、「適当にやったテスト」として、あまり真剣にならないようにお読み下さい。

逆行列の作成

数値計算の中で、逆行列自体を求めることはそれほど多くありません。理由は、もとの行列が band matrix であっても、逆行列は一般に band matrix ではなく full matrix になり、大規模計算にまともに使ってしまうと、メモリーがすぐに不足するためと、LU 分解をした行列 (これは band matrix になります) さえ領域に確保できれば、連立一次方程式の解法のためにわざわざ逆行列を求める必要がないためです。

ここでは Fortran の組み込み関数の精度と、LU 分解 をもとにした逆行列作成サブルーチンライブラリの有効性をチェックするため、次の例題を [7] から引いてみました。

【問題】

次の $n \times n$ 行列 $A = (a_{ij})$ の逆行列を求めよ。

$$a_{ij} = (\frac{2}{n+1})^{\frac{1}{2}} \sin(\frac{ij\pi}{n+1})$$

このとき A は orthogonal symmetric matrix で、逆行列は $A^{-1} = A$ となります。したがって、もとの行列と、計算して得られた逆行列の差がそのまま計算誤差となります。

計算に用いたコンパイラとサブルーチンは次の通りです。すべて計算は倍精度です。

1. Fortran77 EX + SSL II
   SSL II [5] のサブルーチンの DALU と DLUIV を連続して呼び出しています。 DALU はクラウト法による LU 分解、 DLUIV は LU 分解された行列の逆行列を求めるサブルーチンです。

2. Fortran77 EX + NUMPAC
   NUMPAC [6] のサブルーチンは、 MINVD を使用しています。方法は LU 分解法です。

3. gcc + utility
   Sun Sparc 10 のワークステーションの gcc(C++) に、 PROFIL [8] というライブラリにあった逆行列を求める utility を使います。

下表は、 $n = 10$ と $n = 150$ について、もとの行列 $A = (a_{ij})$ と、 numerical computation で得られた行列 $B = (b_{ij})$ (数学的には $A = B$ のはずです) の誤差 $error$ を

$$error \equiv \max_{1 \leq i,j \leq n} |a_{ij} - b_{ij}|$$

で計算した結果[13]です。

	$n = 10$	$n = 150$
Fortran + SSL II	3.2058×10^{-15}	5.4484×10^{-14}
Fortran + NUMPAC	3.1503×10^{-15}	1.2779×10^{-14}
gcc + utility	7.2165×10^{-16}	9.4386×10^{-15}

表を見る限り、 gcc + utility が若干精度がよいみたいですが、どれもまあまあの精度が得られています。 Fortran の組み込み関数と、逆行列を求めるサブルーチンの優秀さが伝わってきます。

連立一次方程式の解法

連立一次方程式の解法として、 [7] から少し性質 (たち) の悪い問題を選んでみました。

【問題】

次の連立一次方程式を $n = 2, 8, 10$ に対して解け。

$$\begin{pmatrix} 1 - 10^{-n} & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & 0 & 0 \\ 1 & 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} -3 \\ -2 \\ -1 \\ -3 \end{pmatrix} \tag{5}$$

(5) の解は、線形代数を大学時代に習った人ならば、 n を未知数としたまま簡単に紙と鉛筆で求めることができると思います。

Algebraic Computation

Mathematica を使っても、この方程式は n をそのままにして解けます。手順は、以下のように極めて簡単です。

```
doronpa% math    ↵                              <--- Mathematica の起動
Mathematica 2.0 for SPARC
Copyright 1988-91 Wolfram Research, Inc.
 -- Terminal graphics initialized --

In[1]:= m={{1-10^(-n),-1,1,-1},{1,-1,1,-1},{1,-1,0,0},{1,0,0,-1}}    ↵
```

[13] 厳密に言えば、 $error$ の計算にも誤差が混入しています。

```
                 -n
Out[1]= {{1 - 10  , -1, 1, -1}, {1, -1, 1, -1}, {1, -1, 0, 0}, {1, 0, 0, -1}}

In[2]:= LinearSolve[m,{-3,-2,-1,-3}]   ↵      <--- 連立一次方程式を解く

           n        n          n              n         n
Out[2]= {10 , 1 + 10 , 4 + 3 10  + 2 (-1 - 10 ), 3 + 10 }

In[3]:= Factor[%]   ↵                         <--- 結果を整理する

           n        n        n        n
Out[3]= {10 , 1 + 10 , 2 + 10 , 3 + 10 }

In[4]:= Quit   ↵                              <--- Mathematica の終了
```

答えはこの結果で正しく、(5) の解は正確に

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 10^n \\ 10^n + 1 \\ 10^n + 2 \\ 10^n + 3 \end{pmatrix}$$

です。パチパチパチ。

Numerical Computation

続いては、 numerical computation を Fortran で行ないます。当然、 n の値は計算の過程で具体的に与える必要があります。サブルーチンは SSL II の DLAX を用いて、倍精度で計算しました。

n=2

計算結果	真の値
100.00000000000005	100
101.00000000000005	101
102.00000000000005	102
103.00000000000005	103

$n = 2$ の場合、ほぼ真の値と近い結果が得られました。それでは n を大きくしてみます。

n=8

計算結果	真の値
100000000.05263561	100000000
100000001.05263561	100000001
100000002.05263561	100000002
100000003.05263561	100000003

計算誤差が小数点部分をかなり汚しています。もっと n を大きくしてみます。

```
n=10
        計算結果                    真の値
 10000000560.375102          10000000000
 10000000561.375102          10000000001
 10000000562.375102          10000000002
 10000000563.375102          10000000003
```

とうとう整数部分にまで誤差が侵食してしまいました。浮動小数点演算ではこのあたりが限界のようです。

Verified Computation

次は、この連立一次方程式の計算を、精度保証付きのソフトウェアで行ないます。使用したものは Hamburg-Harburg 工科大学の Olaf Knüppel さんが C++ 上に構築した PROFIL ([8]) というライブラリです。

PROFIL には、与えられた連立一次方程式の解を、区間の集合として包み込む solver が組み込まれています。原理は S. M. Rump さんの次の定理 (cf.[9]) を、区間作用素の形で実現したものです。定理だけを英語で紹介します。記号の説明はパスさせていただきます。

THEOREM
Let $\mathcal{A} \in \mathbb{IPIR}^{n\times n}$, $\mathcal{B} \in \mathbb{IPIR}^{n}$ be given and let $\tilde{x} \in \mathbb{IR}^n$, $R \in \mathbb{IR}^{n\times n}$, $\emptyset \neq X \in \mathbb{IPIR}^n$, X being compact. Define

$$\mathcal{Z} = R \cdot (\mathcal{B} - \mathcal{A}\tilde{x}) \quad \text{and} \quad \mathcal{C} = I - R \cdot \mathcal{A},$$

$$L(X) = \mathcal{Z} + \mathcal{C} \cdot X,$$

all operations being power set operations. If

$$L(X) \subseteq \mathrm{int}(X)$$

then R and every $A \in \mathbb{IR}^{n\times n}$, $A \in \mathcal{A}$ is nonsingular and for every $b \in \mathcal{B}$ the unique solution $\hat{x} = A^{-1}b$ satisfies

$$\hat{x} = \tilde{x} + L(X).$$

この定理によって何が言えるのかというと、浮動小数点演算で得られた $Ax = b$ の近似解 $\tilde{x}$ と真の解 $\hat{x}$ との誤差評価が、行列 A の可逆性を含め、計算機の中で検証できます。

計算結果は区間の形で表現され、$L(X) \subseteq \mathrm{int}(X)$ という検証条件が満たされた場合、区間 $\tilde{x} + L(X)$ の中に連立一次方程式の解が必ず存在することを、上のややこしい定理が保証しています。

以下は、各 n に対し PROFIL で行なった verified computation の出力結果です。

```
n=2
 [ 99.99999999999959, 100.000000000001]
 [100.9999999999996 , 101.000000000001]
 [101.9999999999996 , 102.000000000001]
 [102.9999999999996 , 103.000000000001]
```

```
n=8
 [ 99999999.11763082, 100000002.0978633]
 [100000000.1176306 , 100000003.0978635]
 [100000001.1176306 , 100000004.0978635]
 [100000002.1176308 , 100000005.0978633]

n=10
 [9999991201.131073, 10000029348.54311]
 [9999991201.971003, 10000029349.70318]
 [9999991202.971003, 10000029350.70318]
 [9999991204.131073, 10000029351.54311]
```

区間解析に詳しい方ならご存知の様に、区間解析は"最悪"の場合を想定した演算の定義がされています。そのため、求める解の存在が保証された区間を提示されても、区間幅が、上の $n = 10$ のように広がり過ぎて、あまり意味がなくなる場合もあります。

御覧のように、 verified computation は、あくまで「近似」に過ぎない浮動小数点演算に比べ、連立一次方程式の場合は解析的な定理に基づく精度保証を与えている分だけ、 algebraic computation に近いといえます。

固有値問題

最後に、これも数値計算の王道ともいえる固有値解析の例題で Numerical Computation のアラを探します。問題は [10] から拝借しました。

【問題】

次の $n \times n$ 行列 A の固有値を求めよ。

$$A = \begin{pmatrix} 1 & & & & -1 & -1 \\ 1 & 1 & & & -1 & 0 \\ & \ddots & \ddots & & \vdots & \vdots \\ & & 1 & 1 & -1 & 0 \\ & & & 1 & 0 & 0 \\ & & & & 1 & 2 \end{pmatrix} \tag{6}$$

(6) を見る限り、それほど誤差が混入するようには見えません。固有値解析のサブルーチンは NUMPAC の HEQRVD を使用しました。 $n = 17$ の場合の計算結果を見てみます。

```
Eigenvalue
   1.1322859860
   1.1229217217 ± 0.0484689972i
   1.0963368694 ± 0.0897980776i
   1.0566992036 ± 0.1181569137i
```

```
1.0099364473 ± 0.1299194454i
0.9626498727 ± 0.1240285995i
0.9210696786 ± 0.1018650121i
0.8902963972 ± 0.0667226311i
0.8739468165 ± 0.0232084483i
```

出力結果を見るだけでは、固有値は複素数で、複素平面上 $z = 1$ を中心にぐるりと分布しているようです。数値結果も極端に大きな値ではなく、計算も倍精度で行なっていますので、これで真の値に相当近い値が得られたと期待されます。

ところが exact な A の固有値は、多重度 n で 1 になることがすぐに確かめられます。この場合 $n = 17$ ですので、 A の固有値は 1 が 17 個ずらっと出てくるのが正解です。

Mathematica で計算してみます。(6) をファイルに作成しておき、読み込ませます。

```
In[1]:= ReadList[``matrix.data'', Number, RecordLists -> True]  ↵

Out[1]= {{1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, -1},
                :
          (行列の表示)
                :
In[2]:= Eigenvalues[%]  ↵        <-- 固有値の計算

Out[2]= {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}
```

見事に正しい結果が得られました。

この、浮動小数点演算の信頼性を大いに失墜させる例題をこしらえた Rump さんは [10] の中で、「このようなエラーはまれである。しかし W. M. Kahan の言葉に耳を傾けるべきである」と、本稿の冒頭にあげたセリフを引用しています。

日本語にするとこんな意味です。

> 計算結果と真の値が有効性を失うほどの不一致を起こすことは極めてまれである。あまりにまれであり、始終そのことを心配する必要はない。
> ただし、 無視できるほどまれでもない。

8 おわりに — 究極の計算とは —

本稿では、数値計算に関わるいくつかの例を、“精度保証” という視点から見てきました。計算機ハードと数値計算ソフトの飛躍的な発展は、ひと昔前は手も足も出なかった大規模計算を次々に可能にしています。さらに、近年は従来の浮動小数点演算にプラスして、区間演算や有理数演算を取り入れたライブラリも登場しており、数値計算は大規模 + 品質保証という時代に既に突入しています。

では最後に、これまであげた計算の分類にまだ入っていない “計算” があることを紹介して、あとがきにかえたいと思います。以下の双子の行なう “計算” の仕組みがわかれば、みなさんが冴えた時に行使される “直観” の正体も解明されるのでは、と私は個人的に考えています。が、もしかしたらこの仕事は哲学者の仕事かもしれません。

素数のお話

素数 (prime number) とは、ご存知のように、 1 とその数自身でのみ割り切れる自然数です。小さい順にならべると

$$2 \quad 3 \quad 5 \quad 7 \quad 11 \quad 13 \quad 17 \quad 19 \quad 23 \cdots$$

といったものです。

現在、ある自然数が素数であるか否かを一発で判定する方法は知られていません。

素数を見つける唯一地道な方法は、与えられた数以下の (たくさんある) 素数で、その数が割り切れないことを順番に確かめていくというものです。

例えば、 697 は 17×41 と素数どうしに分解できるので、素数ではありません。これくらいなら何とか判定できますが、『19937 は素数か？』と聞かれると、うなりながら数表をひっくり返すしか方法がなくなるでしょう。

Mathematica には素数判定機能がありますが、値が大きくなると、段々レスポンスが遅くなってきます。まして、紙と鉛筆でもって、たとえば 10 桁くらいの自然数が素数かどうかを判定することは、仮に出来たとしても膨大な時間がかかります。

ところが、心理学者オリヴァー・ザックスさんの報告によると、世の中には、何らかの方法で (しかも暗算で) 素数の判定をおこなう人が存在するのです。

マイケルとジョンという双子がアメリカのある州の施設に収容されています。双子についての医師の診断結果は、自閉症・精神異常 etc. です。

ある日ザックスさんは、二人が部屋の隅で満足そうに気味の悪い笑みを浮かべながら六桁の数字を紙に書いて興じているのを見つけました。ザックスさんはそのいくつかを写し、後で数学の本で調べてみると、すべて素数ということがわかりました。で、

> 翌日、二人のゲームの仲間入りした。前日数表から写し取っておいた八桁の素数を一ついきなり二人に突きつけた。二人はびっくりしたように彼を眺め、三十秒ばかりその数を見ていた。それから顔を笑いでいっぱいにした。次に二人は九桁の素数の交換を始めた。ザックスは、けしかけて十桁にも挑ませた。ザックスの驚愕は続いた。しばらく沈黙があって、ジョンは彼に十二桁の数を一つ示した。確認の手段はなかった。数表には十桁までしか出ていない。しかし、彼はそれが素数であることを疑わなかった。それから一時間後、兄弟は二十桁の数の交換を始めた。
>
> Colin Wilson & Damon Wilson
> “The Encyclopedia of Unsolved Mysteries” 関口篤訳

Colin Wilson さんは、「双子はザックスが八桁の素数を突きつけた時の三十秒の沈黙で何をしていたのか？」という問いを投げかけています。

はたして、双子はこの三十秒間、何を“計算”していたのでしょうか。

参考文献

[1] 高木貞治 : 解析概論, 岩波書店 (1939).

[2] 戸川隼人 : 数値計算, 情報処理入門コース 7, 岩波書店 (1991).

[3] 中尾充宏 ： 精度保証付き数値計算の現状と動向, 情報処理, Vol.31, No.9, 1177–1190 (1990).

[4] Forsythe, G.E., Malcolm, M. A. and Moler, C. B. : *Computer methods for mathematical computations*, Prentice-Hall, Inc. (1977).

[5] SSL II 使用手引書 (科学用サブルーチンライブラリ) 99SP–0050, 富士通株式会社 (1980).

[6] ライブラリー・プログラム利用の手引 (数値計算編 : NUMPAC), 名古屋大学大型計算機センター (1989).

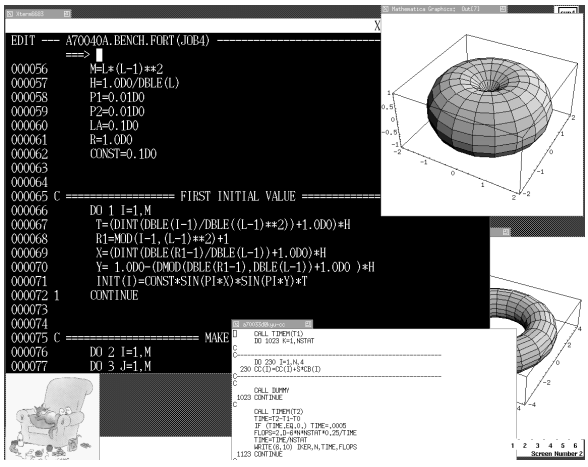
[7] Gregory, R. T. and Karney, D. L. : *A collection of matrices for testing computational algorithms*, John Wiley & Sons, New York (1969).

[8] Knüppel, O. : *PROFIL — Programmer's runtime optimized fast interval library*, Berichte des Forschungsschwerpunktes Informations- und Kommunikationstechnik, TUHH (1993).

[9] Rump, S. M. : Solving algebraic problems with high accuracy, in Kulish, U. W., and Miranker, W. L., editors, *A New Approach to Scientific Computation*, Academic Press, New York, 51–120 (1983).

[10] Rump, S. M. : *Verification methods for dense and sparse system of equations*, Berichte des Forschungsschwerpunktes Informations- und Kommunikationstechnik, TUHH (1993).

[11] 柏木雅英, 大石進一 : 区間解析と有理数演算による非線形方程式の近似解の精度保証, 数理解析研究所講究録 831, 京都大学数理解析研究所, 53–72 (1993).



一見、数値計算をやっていそうな端末の画面