計算機援用証明 I

Computer-Assisted Proofs I

published in Bulletin of the Japan Society for Industrial and Applied Mathematics (Bull. JSIAM), 14(3):2–11, 2004

Siegfried M. Rump
Hamburg-Harburg 工科大学

連絡先: 〒 169-8555 東京都新宿区大久保 3–4–1
早稲田大学大学院 理工学研究科
荻田 武史 (訳者)

Abstract

We will discuss the possibility to 'prove' mathematical theorems with the aid of digital computers and present different methods to approach this goal. Special focus will be on integer and floating point arithmetic, on computer algebra methods as well as on so-called self-validating methods. This note can only cover a very small part of the subject and is intended to stimulate discussions, and possibly reconsideration of one or the other point of view.

Keyword computer-assisted proof, computer algebra, floating point arithmetic, self-validating methods.

キーワード 計算機援用証明，計算機代数学，浮動小数点演算，自己検証的算法.

訳者: 荻田 武史 (早稲田大学大学院 理工学研究科)

0.1 証明と計算機

古来より一般に数学の証明というものは，人間が紙と鉛筆を用いて一歩ずつ理論を展開して行くことにより得られるものである．それゆえ，その証明のあらゆる詳細について追跡できるのである．だがここで一つ考えて欲しい．もし証明の途中に電子計算機を援用して確かめた部分があったとしたら，あなたはそれを許容することができるだろうか．

もちろんそれは様々な場合によるだろう．それでもこの問いかけには実は主要な側面が潜んでいるのである．本稿ではその問題すべてを扱うことは出来ないが[†1]，以下でいくつかについて触れてみたいと思う．それらは，ほんの僅かな側面に対する議論にしか過ぎないが，その問題意識が今後読者に何らかの形で役に立つことを願う．

計算機を援用して実行される数学の『証明』には多くの例がある．たとえば，初等関数の積分に関する R.H. Risch のアルゴリズム [5, 11] である．このアルゴリズムは，基本算術演算・根・べき乗・初等標準関数から構成されるような関数を，初等関数から成る有限項の範囲内で積分可能かどうかを『判定する』．そして，もし判定がイエスなら，積分の閉じた公式を計算する．それは有限ステップで終わるアルゴリズムであり，決定アルゴリズムであり，その最大計算時間は入力の長さで決まる．このアルゴリズムは，たとえば計算機代数学システム Maple [9] で利用できる．一例を挙げてみると，Risch のアルゴリズムは

$$\int e^{x^2} dx \quad \text{は初等関数で表現できない}$$

と断定するが，その結論は自明なものではない．そのアルゴリズムは，導関数が e^{x^2} と等しいような基本算術演算・根・べき乗・初等標準関数の有限の組合せは見つけることができないことを「証明する」のである．実際には，このアルゴリズムはさらに

$$\int e^{x^2} dx = -\frac{1}{2}\sqrt{-\pi} \cdot \mathrm{erf}(\sqrt{-1} \cdot x) + C$$

であることを示すが，この中に含まれる誤差関数は初等関数の有限項の組み合わせでは表現できない．

もう一つの例は，大きなメルセンヌ素数の追求で，現在では主に Lucas-Lehmer テストを用いたプログラムによってしらみつぶしに実行されている．1971 年に，$2^{19937} - 1$ (十進数で約 600 桁) が素数であることが Bryant Tuckerman [16] によって証明された．この事実は記録としてギネスブックに載り，証明に成功した IBM Thomas J. Watson Center (証明が実行されたところ) の人々は，それを誇りに思い郵便封筒に印刷するようになった (図 1).

[†1] 本稿は，本誌用に文献 [12] を大幅に短くまとめたものであるため，より詳細については [12] を参照されたい．

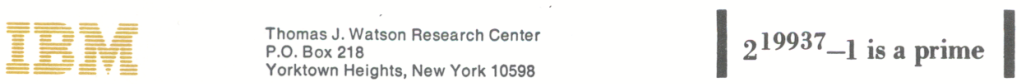

図 1: IBM Thomas J. Watson Center の郵便封筒に印刷されたもの

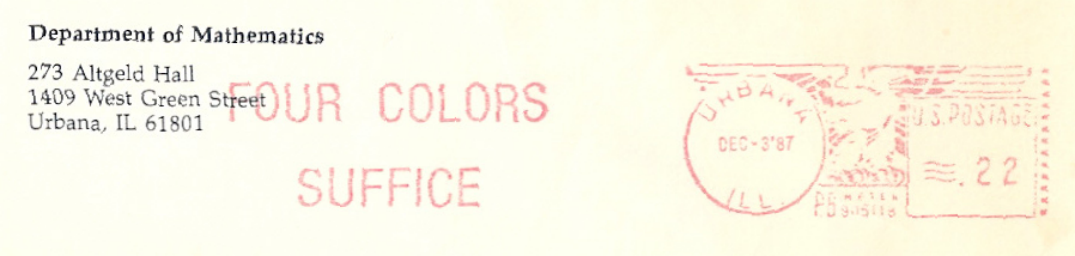

図 2: Illinois 大学の郵便封筒に印刷されたもの

今日知られている最大のメルセンヌ素数 [15] は $2^{6972593}-1$ (十進数で 200 万桁以上) である [†2]．それは Great Internet Mersenne Prime Search [4] の間で見つかり，発見者の Nayn Hajratwala は約 5 万 US ドルを獲得した．

続いて，Champaign-Urbana にある Illinois 大学は郵便封筒に『Four Colors suffice』と印刷して載せたが，これは Kenneth Appel と Wolfgang Haken による名高い功績に帰するものである (図 2)．彼らのアプローチでは何千ものグラフのサンプルをチェックする必要があり，四色問題の解決までには計算機による莫大な処理が必要だった．

もう 1 つの有名な例は Kepler 予想 (Hilbert の第 18 問題) であり，それは立方体の内部に同じ大きさのボールを充填するとき，面心立方の充填が最も高密度な充填になるというものである．この約 400 年来の予想は，1998 年に T.C. Hales [6] によって (肯定的に) 解決された．その証明ではいくつかの大きな非線形最適化問題を厳密に解く必要があった．計算機援用証明に関するより多くの例は，文献 [3] で見ることができる．

上記の例はそれぞれまったく異なる種類のものであるが，共通して大量の計算過程を必要とする．したがって，エラーを引き起こす多数の機会が与えられるわけである．では，どのような場合にそういった証明を信頼することができるのだろうか．言い換えると

証明とは何か？

ということである．

第一に，証明が人間により行われ，また確認されたということが正しさを保証するわけでなければ，反対に証明に計算機を使用したことが正しさを損ねるわけでもない．数学者は人間であるから間違った結論を出すかもしれないし，それに対する有名な例もある．そういった『誤った』アプローチ，つまり元々解

[†2] 一方，2003 年 12 月に 600 万桁以上の大きなメルセンヌ素数 $2^{20996011}-1$ が発見された．

こうとした問題に対しては誤っていたアプローチが，別の意味で新しく，興味深い洞察をもたらすことも少なくない．有名な例として，Fermat の最終定理を解くための Kummer のアプローチがある．それはその定理を証明することはできなかったが，ものごとの，深くかつ違った角度からの理解をもたらした．難解な証明は『受け入れられる』ために，しばしばかなりの時間 (十分な人数の信頼できる数学者によって確かめられるための) を必要とする．

では，証明を『追跡すること』と『受け入れること』とは何を意味するのだろうか．以下の逸話を考えてみよう．1903 年 10 月，New York 市での American Mathematical Society の会議で，Frank N. Cole はとても変わった『講演』をした．物語は，Cole は自分の講演がアナウンスされたとき，席を立って黒板へ進み，以下の 2 つの整数

$$761838257287 \times 193707721$$

を書いて，手でその掛け算を解いていった，と進んでいく．彼は，それから黒板に

$$2^{67} - 1$$

と書き，また手でそれを解いていった．2 つの計算結果は等しかった．雷のような喝采が起こった．そして Cole は席に着いた．彼は言葉を決して話さなかったし，そして質問もなかった．ここまでが，Scientific American の 1983 年 2 月号のレター中にあるこのイベントの報告である．後日，このメルセンヌ数の因数分解を見つけるのにどれくらい掛かったかを尋ねられたとき，Cole はこう説明した:「3 年間，毎週日曜日に」

これが今日起こったと想像してみよう．我々の内の何人が座って手で掛け算を解くだろうか．多くの人が少なくともポケット計算機 (電卓など) を使うか，あるいはすぐに何らかの計算機代数学システムを使用すると思うし，今日では高等なポケット計算機上に MuPAD や Derive のようなシステムも見つけられる．また，証明の一部で，この程度電子機器を使用することに抵抗を感じる人は，現在ではほとんどいないと言ってよいと思う．

信頼には異なるレベルがある．私は，数学の証明を支援する目的で

- ポケット計算機の使用は広く受け入れられる
- 計算機代数学システムの使用は多少受け入れられる
- しかし，浮動小数点演算の使用は多くの数学者に疑わしいようである

と言うことは妥当なことだと思う．しかし，これは公平だろうか．常識というのは，しばしば思いこみで捻じ曲がるものである．たとえば，統計によって，サメに殺されるよりもミツバチに殺される機会のほうが多いことがわかる．同じように，ポケット計算機が信頼できるくらい十分単純に見える (ある意味で，もちろん真実である) のは，エラーの可能性がシステムの複雑さに比例していると

思うからかもしれない．しかし，明らかな丸め誤差は別としても，これはよく目にしてきた普通のポケット計算機にエラーのないことを意味するのだろうか．

これが正しくないことを我々は知っている．内蔵アキュムレータの限られた長さに起因する深刻なエラーは，最も単純なモデル (べき指数を「持たない」標準的な 8 桁の加算・減算・乗算・除算・平方根を計算できるもので，今までにダイレクトメールでよく目にしてきたような今日まだ広く使用中のもの) のどんなポケット計算機にも依然としてある．この種類のどんな計算機でも良いから

$$1\,000\,000 - 999\,999.99$$

を試してみよう．

両方の数は 8 桁の有効数字を持つので，両方とも誤差なしで計算機に保存することができる．単位がドルであれば，結果は明らかに 1 セントであるのにも関わらず，その小さな計算機は結果が 10 セントであると告げるだろう．これは内蔵アキュムレータが 8 桁長であるということの典型的な影響である．最初に，小数点に従って桁が等しくなるように数が調整される．

$$\begin{array}{r|l} 1\,000\,000 & \\ \quad 999\,999.9 & 9 \\ \hline \qquad 0.1 & \end{array}$$

これは，減数する側の 999 999.99 は右へ 1 つシフトされなければならないことを意味する．しかし内蔵アキュムレータは十進 8 桁長しかないので，最後の数字 9 は消える．結果は一種の壊滅的な桁落ちであり，それは算術の不適当な実装によるものである．

それゆえ，そのような単純な機械でさえ 900% も相対誤差を持った結果を引き起こすことがあり得るなら，一体，パソコンやメインフレームは信頼できる結果を生むことができるのか，と尋ねる人がいるかもしれない．この質問に対し，多くの人々は「浮動小数点演算によって得られる結果は『本質的に』信頼できない」という意味で，簡潔に「できない」と答えるのではないだろうか．

ここで，上記のようなことがポケット計算機だけに起こるわけではないことにも触れておきたい．たとえば 80 年代まで，Univac 1108 のようなメインフレームは浮動小数点演算で

$$16777216 - 16777215 = 2$$

と計算した．理由は前と同じで，これを防ぐためのガードビットが欠けていてアキュムレータが作業する精度と同じ長さの桁数しか持たないことにあった．類似した計算違いが 60 年代前半に IBM S/360 というメインフレームで起こったが，同社がこれを改善するためにアーキテクチャを変更したことは，主に Vel Kahan の尽力による．ごく最近まで同じように欠陥のある実装が Cray のスーパーコンピュータに見られた．

本稿の残りの部分では，今日ではすべての計算機上の浮動小数点演算は厳密に定義されているし，信頼して使用できるということを説明する．だがその前

に，おそらくより非常に単純なことだが，整数演算の中にさえ落し穴があることを述べておこう．

整数演算に関する主な問題は，いわゆるラップアラウンド[†3]である．$xmax$ を (与えられた形式で) 表現できる最大の正整数，$xmin$ を表現できる最小の負整数とする．そのとき，$xmax + 1$ はラップアラウンドのために $xmin$ という結果になる．大切なことは，このラップアラウンドが演算例外を引き起こさないということであり，それはユーザに通知することなく起こってしまう．ユーザがこの事実を知らなかったら，これは深刻なトラブルを引き起こすかもしれない．

もっと不思議なことも起こりえる．1 の補数で整数を格納するため，$-xmin$ の結果は $xmin$ となるから，$xmin$ と $-xmin$ は等しい！[†4]

これらの事実は十分に広くは知られていないようである．たとえば，比較 $x \leq y$ を数学的に等価な比較 $x - y \leq 0$ と翻訳するいくつかのコンパイラがある．しかしながら，整数演算ではラップアラウンドのためにこれは必ずしも等しくないので，間違った結果が警告なしで起こるかもしれない．

ここまでのところで引き出すことができる 1 つの結論は，人は作業をしている環境を正確に把握していなければならないということである．あらゆるツールはその限界をもっており，ユーザはそれらを正確に知らなければならない．同じことが整数演算にも当てはまる．正しい方法で使われなければ予想外でしかも間違った結論が生じるだろう．もちろんこれは当たり前の主張である．しかしそれが必ずしも当たり前でないのは，ユーザがシステムの動作を実際とは違うように理解してしまう危険性があるからである．上の例で言えば，整数の加減算は常に正確だというのがその思いこみで，実際にはそうでない場合もありうるのである．

同様に，一般の認識は「浮動小数点演算は通常，不正確である」というものであり，これは一般的に正しい．浮動小数点演算の誤った使用によって罠にはまるのは，使い方によっては『より容易』かもしれない．あるパトリオット防御ミサイルで，制御部分の浮動小数点演算が不正確であったために，28 人の兵士が死亡する悲劇が起きた (文献 [10] や文献 [7] の第 27.11 節を参照のこと)．この出来事を一層悲惨に感じるのは，十進数の 0.1 を単精度浮動小数点数に変換したときの誤差が原因と突きとめられるからである．しかし，非難されなければならないのは演算ではない．

0.2 演算の問題

果たして電子計算機上で正しいあるいは検証された結果を得ることができるかどうか論議するために，以下の例を考えてみよう．実行列 A が与えられたとして，簡単のためにすべての要素 A_{ij} が浮動小数点数であると仮定しよう．言

[†3] wrap-around: ここでは，整数がぐるりと周る形で計算機に実装されていることを意味する．
[†4] もちろん，このようなシステム上では，ということである．

い換えると，その行列は正確に計算機上で表現できる．ここで，行列 A が特異かどうかについて知りたいとする．

いわゆる自己検証的算法 (self-validating method) の特長は，理論的保証と計算機実装のしやすさの両方を兼ね備えていることである．理論上のアプローチとして次の例を考えてみよう．R を任意の実行列とする (R は前処理行列として用いる)．もし，スペクトル半径 $\rho(I - RA)$ が 1 より小さければ，R と A は正則である．なぜなら，そうでないとすると $C := I - RA$ が固有値 1 を持つことになるからである．$|C|$ を要素毎に絶対値を取った値 $|C_{ij}|$ からなる非負行列とすると，Perron-Frobenius の理論から，$\rho(C) \leq \rho(|C|)$ であることがわかる．さらに，よく知られた Collatz の定理から，任意の要素がすべて非負である実ベクトル x に対して

$$|I - RA|x < x \tag{1}$$

は $\rho(|I - RA|) < 1$ であることを意味するので $\rho(I - RA) < 1$ であり，R と A は正則であることがわかる．言い換えれば，要素がすべて正であるようなベクトル (たとえば，要素がすべて 1 のベクトル) に対して式 (1) を証明できれば，A が正則であることを証明できる．行列 R は式 (1) を満たすようなものであれば，それ以外の要件がまったくなくても依然としてこの結論は正しい．もちろん，ひとつの良い選択は A の近似逆行列であるが，そうである必要はない．

今，浮動小数点演算で式 (1) を検証することを目指す．そのためには，浮動小数点演算の性質についてもう少しの情報を必要とする．つまり，当たり前のことだが

使用中の浮動小数点演算がどのように定義されているか

を知る必要がある．我々の多くは，およそ 20 年前までは計算機メーカからこれに関する多くの情報を入手可能ではなかったのを覚えているだろう．これが，浮動小数点演算を標準化しようとする共通の動きの一因となった (1985 年の IEEE 754 二進浮動小数点演算規格 [1])．この規格の中では，すべての浮動小数点演算の最大の相対誤差を `eps` (相対的な丸め誤差の単位) 以下と定める．さらに有向丸め (上向き・下向きの丸め) を定める．

現在，IEEE 754 浮動小数点演算規格は，PC やワークステーションからメインフレームに至るまで，科学技術計算で使用される計算機のうち，相当数で実装されている．丸めモードは頻繁に切り換えられるが，これはプロセッサの丸めモードを設定し直すことで実現される．たとえば，もし演算器が下向き丸めに切り替えられた場合，それ以降の「あらゆる」浮動小数点演算は結果として正確な値以下の唯一の浮動小数点数をもたらす．これはまったく著しい特性であり，頼りにできる「数学的な」特性である．

$\mathrm{fl}(\mathit{expression})$ は，$\mathit{expression}$ のすべての操作を浮動小数点演算で実行したときの値とする．さらに，$\mathrm{fl}_{\nabla}(\mathit{expression})$ と $\mathrm{fl}_{\Delta}(\mathit{expression})$ は，それぞれ下向きと上向きに丸めモードが変更されているものとする (丸め記号が省略されている場合は最近点への丸めと仮定する)．このとき $\mathbb{IF}$ を使用中の浮動小数点数の

集合とすると，数学的なステートメント

$$\forall\, x, y \in \mathbb{F} : \quad \mathrm{fl}_{\nabla}(x+y) = \mathrm{fl}_{\Delta}(x+y) \quad \Leftrightarrow \quad x+y \in \mathbb{F} \tag{2}$$

を得る．これは，実数演算によって得られる (数学的に) 正しい結果が (表現可能な) 浮動小数点数であることと，下向き丸めと上向き丸めによって得られた浮動小数点演算の結果が等しいことは必要十分である，ということを意味している．これはもちろん，減算・乗算・除算についても真である．また，以下のように平方根についても真である:

$$\forall\, x \in \mathbb{F} : \quad \mathrm{fl}_{\nabla}(\sqrt{x}) = \mathrm{fl}_{\Delta}(\sqrt{x}) \quad \Leftrightarrow \quad \sqrt{x} \in \mathbb{F} \tag{3}$$

これらの結果はアンダーフローが起きても正しい．それは数学的に信頼できるステートメントである．ただしこのとき，数学的には上の主張は

- 浮動小数点演算の実装が IEEE 754 規格による定義に従っていて
- オペレーティングシステム，ハードウェアとソフトウェアを含めたすべてが正しく動作していて
- ショートによる停電，放射線，その他の手に負えないような外部の歪みは起こらない

が満たされた場合に正しい，と書くべきだと思う人もいるであろう．それはまったくそのとおりである．その意味で，我々は完全にはどんな機械による結果も決して信頼することができないし，計算機を援用した数学の証明はまったく可能ではない (まあ人間の頭脳にも失敗の危険性はあるわけだが).

だが『ありそうかどうか』より，もっと厳密な議論であることに注意して欲しい．以前，著者は初等標準関数の厳密な実装に取り組んでいたのだが，あるライブラリによって提供される標準関数の精度と信頼性についての徹底的な議論を持った．簡単に言えば，質問は，既存のライブラリを用いるときに 1ulp[†5] より大きい誤差が生じる例が知られていないとき，一体『信頼できる』標準関数が必要なのか，というものであった．そして，既存のライブラリに関しては何億ものテストケースによって『確かめられる』ことができる，と.

しかし，浮動小数点演算の精度と標準関数の精度の間には基本的な違いがある．後者は，非常に巧みな手法 (たとえば，連分数，テーブルベースのアプローチ，CORDIC アルゴリズム [†6] など) で構成されていて，すべては正しい結果に対する精度の良い近似を提供するために実行される．しかし，ほとんどのライブラリでは「すべての」可能な入力引数に対して有効であるような「厳密な」誤差評価がない．そして，結果が誤っていないという何億ものテストは，アルゴリズムが決して失敗しないことを「証明」するわけではない.

[†5] unit in the last place: 浮動小数点数の最下位ビットの大きさを表す単位.
[†6] COordinate Rotation Digital Computing algorithm.

対照的に，IEEE 754 規格に従う浮動小数点演算の実装はよく理解されており，式 (2) や式 (3) のような結論が「すべての」可能な入力引数にあてはまることを解析により示せる．この意味において浮動小数点演算のアルゴリズムは，数学的に厳密に定義されており常に正しい．実装時に定義が守られない危険性はあるかもしれないが．

以下では，浮動小数点演算の実装は IEEE 754 規格の仕様に従い，使用中のソフトウェアとハードウェアは正しく動作していると仮定する．それでは，あらためて電子計算機を援用して結果を検証することが可能かどうか考えてみよう．

式 (1) による行列 A の正則性の検証はとても単純である．必要とする補足的な知識は，f を浮動小数点数とすると $|f|$ も同様に浮動小数点となる，ということである．そのとき，与えられた R, A そして x に対して

$$\begin{aligned} C_1 &:= \mathrm{fl}_\nabla(R * A - I) \\ C_2 &:= \mathrm{fl}_\Delta(R * A - I) \\ C &:= \max(\mathrm{abs}(C_1), \mathrm{abs}(C_2)) \\ y &:= \mathrm{fl}_\Delta(C * x) \end{aligned}$$

を計算する．定義から

$$\mathrm{fl}_\nabla(R_{i\nu} \cdot A_{\nu j}) \le R_{i\nu} \cdot A_{\nu j} \le \mathrm{fl}_\Delta(R_{i\nu} \cdot A_{\nu j}) \quad \text{for all } i, \nu, j$$

であり，それゆえ

$$C_1 \le RA - I \le C_2$$

及び

$$|I - RA| \le \max(|C_1|, |C_2|) = C$$

となる．ここで，最大値・絶対値・比較は要素毎に解釈されるものとする．これと式 (1) を合わせると，

$$y < x \quad \text{は } A \text{ (及び } R\text{) が正則であることを意味する}$$

となる．$RA - I$ を $I - RA$ と入れ替えると，必ずしも不等式が正しくないという点に注意しよう．また，このアプローチは 2 回の丸めモードの切り替えを必要とし，それ以外は浮動小数点演算のみ (特に分岐がない) であることにも注意しよう．これは，行列乗算を BLAS ルーチン [2, 8] によって実行することができるため非常に高速であることを意味する．このようにやり方によっては，丸め計算で速度と厳密さの両方を実現できるのである．

「計算機援用証明 II」では，自己検証的算法についてもう少し詳細に，特に適用性の範囲と計算機代数学的方法との差異について議論したい．

関連図書

[1] ANSI/IEEE 754-1985, Standard for Binary Floating-Point Arithmetic, 1985.

[2] Dongarra, J. J., Du Croz, J. J., Duff, I. S., and Hammarling, S. J., A set of level 3 Basic Linear Algebra Subprograms, ACM Trans. Math. Software, 16 (1990), 1–17.

[3] Frommer, A., Proving conjectures by use of interval arithmetic, Perspectives on Enclosure Methods (edited by Kulisch, U. et al.), Springer, Wien, 2001.

[4] GIMPS - The Great Internet Mersenne Prime Search, `http://www.mersenne.org/prime.htm`.

[5] Grabmeier, J., Kaltofen, E., and Weispfennig, V., Computer Algebra Handbook, Springer, 2003.

[6] Hales, T. C., Cannonballs and honeycombs, Notices of the AMS, 47:4 (2000), 440–449.

[7] Higham, N. J., Accuracy and Stability of Numerical Algorithms, 2nd edition, SIAM Publications, Philadelphia, 2002.

[8] Lawson, C. L., Hanson, R. J., Kincaid, D., and Krogh F. T., Basic Linear Algebra Subprograms for FORTRAN usage, ACM Trans. Math. Soft., 5 (1979), 308–323.

[9] Maple V, Release 7.0, Reference Manual, 2001.

[10] Patriot missile defense: Software problem led to system failure at Dhahran, Saudi Arabia, Report GAO/IMTEC-92-26, Information Management and Technology Division, United States General Accounting Office, Washington, D.C., February 1992, 16 pages.

[11] Risch, R. H., The problem of integration in finite terms, Transactions of the American Mathematical Society, 139 (1969), 167–189.

[12] Rump, S. M., Computer assisted proofs and self-validating methods, Handbook on Accuracy and Reliability in Scientific Computation (edited by Einarsson, B.), to appear, 55 pages.

[13] Rump, S. M., Rigorous and portable standard functions, BIT, 41:3 (2001), 540–562.

[14] Rump, S. M., Self-validating methods, Linear Algebra and its Applications, 324 (2001), 3–13.

[15] The Top Twenty: Mersenne, `http://www.utm.edu/research/primes/lists/top20/Mersenne.html`.

[16] Tuckerman, B., The 24th Mersenne Prime, Proc. Nat. Acad. Sci., volume 68 (1971), 2319–2320.

図 目 次

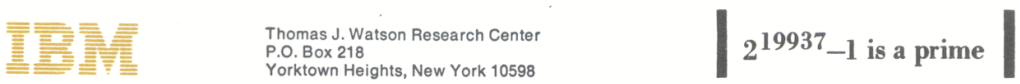

図 1: IBM Thomas J. Watson Center の郵便封筒に印刷されたもの

図 2: Illinois 大学の郵便封筒に印刷されたもの