数理解析研究所講究録
1038巻 1998年 205-209

Javaによる数式処理

電子技術総合研究所 元吉文男 (Fumio Motoyoshi)

1. 動機

筆者はこれまで数式処理システムをLispで開発してきたが、Javaという言語が発表されそれを使用してみたところ、数式処理システムの記述用の言語としても十分に使用可能であると認識したので、現在の開発状況を報告する。

2. LispとJavaの比較

数式処理システムの記述用言語として見た場合のLispとJavaの特徴を比較してみると以下のようになる。

	Lisp	Java
freeな処理系	有	有
実行速度	△	△
自動ゴミ集め	○	○
対話的か	○	×
関数引数	○	△
型宣言	不要	必要
Object指向	△	○
module化	△	○
必要メモリ	大	小
分散計算	×	○
GUI	△	○
platform	少	多
移植性	○	◎

処理できるデータ型に関してはどちらの言語でも大きな差はないと判断した。また、この表のうちJavaにとって問題であると考えられる対話的に使用ができないという欠点はあるものの、コンパイルにそれほど時間がかからない、Lispからの移植が主になるためにアルゴリズムは確定しているなどの点でそれほど負担にはならないと考えてJavaでシステムを作成することにした。

3. 現状

これまでのところで作成・移植したプログラムは以下のものである。

- 多倍長整数ルーチン
- 多倍長固定小数点演算
- Lisp 関数の実現
- Lisp から Java への自動変換
- 1 変数多項式の因数分解

多倍長整数ルーチンに関しては、JDK 1.1 以降では標準でサポートされており、しかも、native code で記述されているために筆者のルーチンよりも一桁近く高速であったのは残念である。ただし、現在の多くのブラウザでは Java が多倍長整数ルーチンをサポートしていないため、Web から使用できるようにするためには当分の間は必要となると思われる。計算量の大きい乗算に関しても、特別な方法は使用していないため、n 桁同士の乗算では n^2 に比例した計算時間がかかっている。今回は、余分なメモリを必要とする、プログラムが複雑になるという理由で Karatsuba のアルゴリズム、あるいは FFT を利用する計算法は採用しなかった。

多倍長固定小数演算は、多倍長整数演算ルーチンのデバッグの意味もあり作成した。四則以外に、指数、対数、三角関数の計算を Brent の方法を使用して、通常の級数を使用する方法よりは計算量のオーダは小さい。二進 n 桁の精度で計算するとして、二進 n 桁の数同士の掛け算の時間を $M(n)$ とする。このとき、たとえば指数関数の計算では通常の級数を使用する方法では、計算時間の主要項は $M(n)n$ であるが、Brent の方法では $13M(n)\log_2 n$ となる。Brent の超越関数計算では、逆関数を計算するときに Newton 法による逐次近似を行なっている。このときに導関数の計算が困難な場合には差分によって微係数の近似値としているが、このときの誤差の評価について次節で述べる。

このようにして作成した多倍長固定小数演算ルーチンを利用して 多倍長電卓を Java の applet として作成し、Web を通して利用できることを確かめた。

また、Lisp で作成してあったアルゴリズムを Java に移植するに際して、手間を少なくするために、実行時間がそれほど問題にならないようなプログラムに関しては自動で Java への変換を行ないたいと考え、Lisp から Java への自動変換プログラムを作成した。これは、すべての Lisp プログラムが変換できるわけではなく、「行儀の良い」プログラムについて変換を行なうものである。変換された Java プログラムから呼ばれることになる関数についても Lisp と同じ働きをする Java プログラムも合せて作成した。Lisp のデータ型は (関数オブジェクトは除いて) 容易に Java のクラスとして実現できるため (一部の高機能関数を除いて) 移植は簡単であった。この自動変換プログラムを用いて数式の構文解析プログラムを変換して、実際に使用して動作を確認した。ただ、自動変換された結果のプログラムは読み難く、非効率的なコードであるために、時間に関して厳しいルーチンなどに関しては、手動で変換するか、変換プログラムを賢くする等の手段が必要になる。

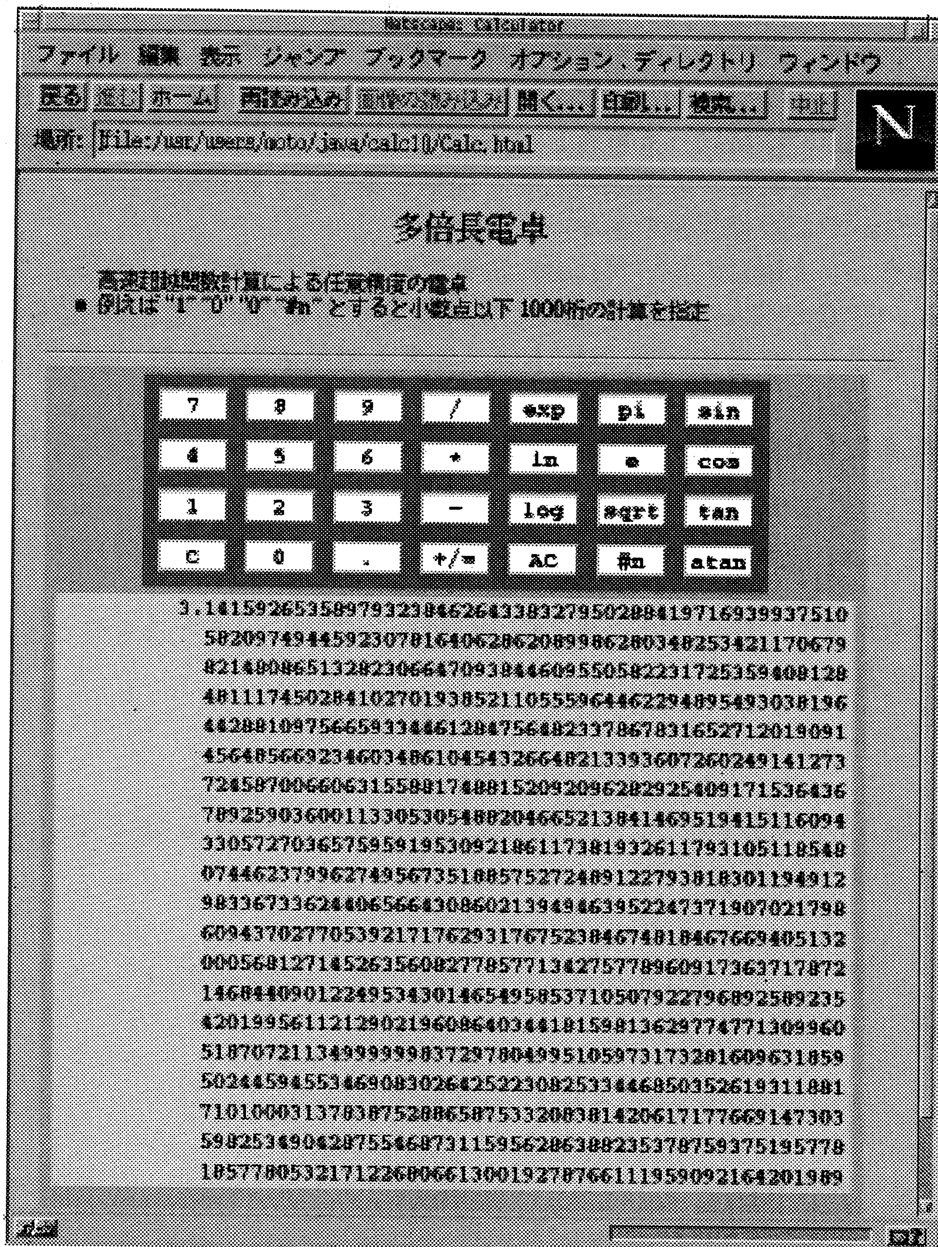

Fig. 1. Java applet による多倍長電卓

4. Newton 法の有効桁

Newton 法による逆関数の計算において計算に必要な桁を正確に見積もった結果を以下に示す。

多倍長で $f(x)=0$ の解を Newton 法により求めるときには、(このアルゴリズムは安定であるので) 必要な桁だけの計算を行なえばよい。このようにすると、逆関数の計算時間は元の関数を計算する時間と同じオーダにすることができる。しかし、必要な精度に達したかどうかを、普通は、前の回との差分を求めて判断しているが、これでは、求める精度の計算を 2 回行なうことになって、ほぼ 2 倍の計算時間がかかってしまう。ただし、関数が固定されている場合には前以って解析しておくことにより、誤差を正確に見積ることができる。ここでは、元の関数の導関数の計算が困難な場合に、数値微分によって微係数を代用する方法についての誤差評価を示す。

$f(x)=0$ の解を α、i 回目のループにおける近似解を $\alpha+\alpha_i$ とする。さらにこのときの誤差を δ とする。(すなわち $|\alpha_i|<\delta$)。また、逆関数を求めようとする区間 $[a,b]$ において $f(x)$ は C^1 級であるとし、その区間での $|f'(x)|$ の最小値を F ($\gg\delta$)、$|f''(x)|$ の最大値を G ($\gg\delta$) とする。

$\alpha+\alpha_i$ での微係数を計算するときの x の値は $\alpha+\alpha_i+\delta'$ とする。ただし、

$$|\delta'|=\delta \tag{1}$$

$$|\alpha_i+\delta'|\leq\delta \tag{2}$$

とする。すなわち、区間の幅は δ で、真の解 α に近づく方向に x をとるものとする。すると、Newton 法により、

$$\alpha+\alpha_{i+1}=\alpha+\alpha_i-\frac{\delta' f(\alpha+\alpha_i)}{f(\alpha+\alpha_i+\delta')-f(\alpha+\alpha_i)} \tag{3}$$

であるが、$f(\alpha+\alpha_i)$ の計算誤差を $A_1\delta^2$、$f(\alpha+\alpha_i+\delta')$ のを $A_2\delta^2$ とする ($A\geq|A_1|,|A_2|\gg\delta$) と、実際に計算される値 α'_{i+1} は、

$$\alpha'_{i+1}=\alpha_i-\frac{\delta'(f(\alpha+\alpha_i)+A_1\delta^2)}{f(\alpha+\alpha_i+\delta')+A_2\delta^2-f(\alpha+\alpha_i)-A_1\delta^2} \tag{4}$$

である。ここで

$$f(\alpha+\alpha_i)=f(\alpha)+\alpha_i f'(\alpha)+\alpha_i^2 f''(\beta_1)/2 \tag{5}$$

$$\beta_1 \text{ は } \alpha \text{ と } \alpha+\alpha_i \text{ の間の値,}$$

$$f(\alpha+\alpha_i+\delta')=f(\alpha)+(\alpha_i+\delta')f'(\alpha)+(\alpha_i+\delta')^2 f''(\beta_2)/2 \tag{6}$$

$$\beta_2 \text{ は } \alpha \text{ と } \alpha+\alpha_i+\delta' \text{ の間の値}$$

を代入すると ($f(\alpha)=0$ を考慮に入れて)、

$$\alpha_{i+1}=\alpha_i-\frac{\delta'(\alpha_i f'(\alpha)+\alpha_i^2 f''(\beta_1)/2+A_1\delta^2)}{\delta' f'(\alpha)+(\alpha_i+\delta')^2 f''(\beta_2)/2+A_2\delta^2-\alpha_i^2 f''(\beta_1)/2-A_1\delta^2} \tag{7}$$

$$=\frac{\alpha_i(\alpha_i+\delta')^2 f''(\beta_2)/2+A_2\alpha_i\delta^2-\alpha_i^2(\alpha_i+\delta')f''(\beta_1)/2-A_1(\alpha_i+\delta')\delta^2}{\delta' f'(\alpha)+(\alpha_i+\delta')^2 f''(\beta_2)+A_2\delta^2-\alpha_i^2 f''(\beta_1)-A_1\delta^2} \tag{8}$$

よって、

$$|\alpha_{i+1}| \le \frac{\delta^3|f''(\beta_2)|/2 + A\delta^3 + \delta^3|f''(\beta_1)|/2 + A\delta^3}{\delta(|f'(\alpha)| - \delta|f''(\beta_2)|/2 - A\delta - \delta|f''(\beta_1)|/2 - A\delta)} \quad (9)$$

$$\le \frac{\delta^2(G/2 + A + G/2 + A)}{F - \delta G/2 - A\delta - \delta G/2 - A\delta} \quad (10)$$

$$\le \frac{\delta^2(G + 2A)}{F - \delta(G + 2A)} \quad (11)$$

$$\le \frac{2\delta^2(G + 2A)}{F} \quad (12)$$

となる。ここで、fが具体的に与えられれば、F、Gの値が求まり、Aの値を調整することによって計算した近似値の精度の上限が決定できるので、求める精度までに必要な繰り返し回数が決定できることになる。

5. 今後の予定

今後の Java による数式処理システムの作成については以下のものを考えている。

- 各モジュールの充実
  - 基本演算
  - 高等演算
  - 簡易入出力
  - 2D 入出力
  - TeX 用プリプロセッサ
- 単独でのシステム
- 分散処理への応用
  - プロトコルの設計

基本演算には、四則、微分、関数適用などがあり、内部表現までも含めて設計中である。高等演算は、因数分解、代数拡大体の計算、積分などがあり、因数分解については一変数のものは既に作成した。また、単独の数式処理システムとして利用するために、簡単な(一次元の)入出力プログラムや制御プログラムを加えてシステム化する必要がある。Java の特性を生かして Web から使用できるシステムにすることも予定している。

TeX 用のプリプロセッサとは、グラフィカルに数式を入力するプログラムで TeX と同じアルゴリズムで数式を入力と同時に表示するプログラムである。この出力は数式処理システムへの入力として使用できるだけでなく、TeX 文書への貼り込み用にも利用できるように考えている。ただ、現在のところでは、Java のフォントの扱いが十分ではないために、数式用の文字はフォントとしてではなく、画像として処理しようと予定している。

プロトコルの設計では Mathematica における MathLink のようなものを想定しており、ネットワークを介して離れている計算機上で計算を実行させるようなメカニズムを開発していと考えている。