社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2006－HPC－107（27）
2006／8／1

任意精度数値微分法に基づくJacobi行列計算について

幸谷智紀
静岡理工科大学

数値微分に基づくJacobi行列計算は，自動微分が使用できない場合に用いられるものであるが，一般には低精度な値しか求められないと思われている。本稿では，補外を用いた数値微分法に基づく任意精度Jacobi行列計算法を提案し，その有効性をMPFR[11]を用いた数値実験によって示す。

On Jacobian Matrix Computation based on Arbitrary Precision Numerical Differentiation Method

Tomonori Kouya
Shizuoka Institute of Science and Technology

Jacobian matrix computation based on numerical differentiation should be selected if automatic differentiation cannot be applied, which seem to be generally low precision. But we can obtain arbitrary precision Jacobian matrices by using extrapolation and multiple-precision floating-point arithmetic. In this paper, we propose the arbitrary precision numerical computation method of Jacobian matrix based on numerical differentiation using extrapolation and demonstrate its efficiency through our numerical experiments with MPFR[11].

1. 初めに

n 変数関数 $\mathbf{F}(\mathbf{Y})=[F_1(\mathbf{Y})\ ...F_n(\mathbf{Y})]^T$, $\mathbf{Y}\in\mathbb{R}^n$ のJacobi行列 $\partial\mathbf{F}/\partial\mathbf{Y}\in\mathbb{R}^{n\times n}$

$$\frac{\partial\mathbf{F}}{\partial\mathbf{Y}}=\begin{bmatrix}\frac{\partial F_1}{\partial Y_1} & \frac{\partial F_1}{\partial Y_2} & \cdots & \frac{\partial F_1}{\partial Y_n}\\ \frac{\partial F_2}{\partial Y_1} & \frac{\partial F_2}{\partial Y_2} & \cdots & \frac{\partial F_2}{\partial Y_n}\\ \vdots & \vdots & & \vdots\\ \frac{\partial F_n}{\partial Y_1} & \frac{\partial F_n}{\partial Y_2} & \cdots & \frac{\partial F_n}{\partial Y_n}\end{bmatrix}=\left[\frac{\partial F_i}{\partial Y_j}\right]_{i,j=1,2,...,n} \quad (1)$$

の近似行列 $J(\mathbf{Y})\approx\partial\mathbf{F}/\partial\mathbf{Y}$ を，中心差分と補外法を用いて高精度に計算する方法について，本稿では議論する。

一般に，数値微分は真の導関数をソフトウェアを用いて代数的に計算する自動微分 (Automatic Differentiation, AD)[7] が使用できない場合に利用されることが多い。例えば，陰的公式を用いた常微分方程式ソルバや，非線型方程式向けのNewton法のライブラリ関数には数値微分を用いて求めた $J(\mathbf{Y})$ を用いるオプションが用意されている。しかしこれは，一階差分商を用いたものであり，低精度な近似値しか得られないとされている。

今回提案する手法は，中心差分商と補外法を用いて高精度な近似Jacobi行列を求めるものである。永坂・福井の結果 [1, 2] から，補外法を用いても，初期系列に用いた近似値の丸め誤差が2倍未満に抑えられることが判明している。従って，補外法の段数を増やして打ち切り誤差を削っていけば，$J(\mathbf{Y})$ の各成分も高精度に求められることになる。しかし，実際の計算では計算量の削減と，収束判定の精密さを両方実現する工夫が必要である。

本稿ではまず，永坂・福井による一変数関数 $f(x)$ に対する数値微分の誤差解析の結果を紹介する。次に，これに基づいて，列単位で $J(\mathbf{Y})$ を計算し，要素ごとに収束判定を行う任意精度Jacobi行列計算法のアルゴリズムを述べる。最後に，このアルゴリズムの応用例として，大規模な常微分方程式の漸近安定性解析を紹介する。

2. 中心差分商と補外法を用いた任意精度数値微分

無限回微分可能な一変数関数 $f(x)$ の m 階微係数 $f^{(m)}(x)$ を，N 点差分商で近似すると，その近似式は一般に

$$f^{(m)}(x)=\frac{1}{h^m}\sum_{s=1}^{N}b_s f(x_s)+E_T(h) \quad (2)$$

と表現できる。ここで b_s は定数，$E_T(h)$ は打ち切り誤差である。我々が使用する1階微係数 $f'(x)$ を3点中心差分商を用いて近似すると，(2) 式は

$$f'(x)=\frac{1}{h}\left\{\frac{1}{2}f(x+h)-\frac{1}{2}f(x-h)\right\}+E_T(h) \quad (3)$$

である。この時 $E_T(h)$ は

$$E_T(h)=\sum_{w=0}^{\infty}\frac{f^{(2w+3)}(x)}{(2w+3)!}h^{2w+2}$$

と h の2のべき乗和として表現できる。

(2) を p 進 q 桁の浮動小数点数を用いて数値計算する時に発生する丸め誤差の限界を $E_R(h)$ とすると，

総和の計算，h^m の計算，及びそれらを乗じる際に発生する丸め誤差を総合して

$$E_R(h) = \frac{N-1}{h^m} \max_s |b_s f(x_s)| \cdot c \cdot p^{-q} \qquad (4)$$

と評価できる [1]。ここで c は丸めの方式によって決まる定数で，

$$c = \begin{cases} p & \text{(切り捨て)} \\ \frac{p}{2} & (p/2-1 \text{ 捨 } p/2 \text{ 入}) \end{cases}$$

である。

中心差分商を用いて求めた近似値 (2) を，Romberg 数列 $2^0, 2^1, \ldots$ を用いた補外法の初期系列 $f^{l,1}$ として使用する。この時

$$f^{l,1} = \frac{\sum_{s=1}^{N} b_s f(x_s)}{h^m / 2^{l-1}}$$

である。

補外計算は

$$\begin{array}{ccc} f^{l-1,k-1} & & \\ & \searrow & \\ f^{l,k-1} & \rightarrow & f^{lk} = f^{l,k-1} + \frac{f^{l,k-1} - f^{l-1,k-1}}{4^{l-1}-1} \end{array} \qquad (5)$$

となる。この時 $f^{l,k}$ に含まれる打ち切り誤差は $O(h^{2l})$ で減少していく。この計算を，以下の表の初期系列から右の下三角成分を計算していくことになる。

初期系列				
$f^{1,1}$				
$f^{2,1}$	$f^{2,2}$			
$\vdots$	$\vdots$	$\ddots$		
$f^{L-1,1}$	$f^{L-1,2}$	$\cdots$	$f^{L-1,L-1}$	
$f^{L,1}$	$f^{L,2}$	$\cdots$	$f^{L,L-1}$	$f^{L,L}$

この補外計算で混入する丸め誤差限界 $E_R^{l,k}$ は

$$E_R^{l,k} = \frac{1+2^{-m}}{4^{l-1}-1} \cdot \frac{25}{14} E_R(h/2^{l-1})$$

であることが福井によって証明されている [2]。これは p 進 q 桁計算における近似値の絶対誤差の限界値に近い値であるため，実際の数値計算ではこの値になる以前に収束したと判定することが望ましい。よって，収束判定は，ユーザが与えた相対許容度 $\varepsilon_r > 0$ と絶対許容度 $\varepsilon_a \geq 0$ も用い，補外計算 (5) において

$$\left| \frac{f^{l,k-1} - f^{l-1,k-1}}{4^{l-1}-1} \right| \leq \max\left(\varepsilon_r |f^{l,k-1}| + \varepsilon_a, E_R^{l,k}\right) \qquad (6)$$

を満足した時に収束したと判定する。与えられた ε_r や ε_a が小さすぎる時には，自動的に求められた丸め誤差限界値 $E_R^{l,k}$ が歯止めになる仕組みである。

3. 任意精度 Jacobi 行列計算法

前節で述べた一変数関数の数値微分のアルゴリズムを用いると，任意精度 Jacobi 行列計算が実現できる。この際，n 変数関数 $\mathbf{F}(\mathbf{Y})$ の呼び出し回数を少なく抑えるため，$J(\mathbf{Y}) = [\mathbf{J}_1\ \mathbf{J}_2\ \ldots\ \mathbf{J}_n]^T$ を列ベクトル単位で計算する必要がある。しかし，各成分は全て異なる偏導関数であるため，収束判定は各成分ごとに個別に行う必要が出てくる。以下，そのアルゴリズムと収束判定法を述べる。

3.1 アルゴリズム

Jacobi 行列の各要素 $\partial F_i / \partial Y_j$ の近似計算には，前述の 3 点中点公式 (3 点公式) を初期系列とする補外法 [1, 2] を使用し，ベクトル単位で全ての計算を行う。ここでは第 j 列目の近似ベクトル $\mathbf{J}_j \approx \partial \mathbf{F} / \partial Y_j$ の計算を対象として解説する。

初期系列 $J_j^{l,1}$ の計算は，まず $\mathbf{Y} = [Y_1\ \cdots\ Y_N]^T$ の第 j 成分を基準に刻み幅 h_j を Romberg 数列に乗じて

$$\mathbf{Y}_c^{j\pm} = \left[\cdots\ Y_{j-1}\ Y_j \pm 2^{-l} h_j\ Y_{j+1}\ \cdots\right]^T$$

とし，中心差分を用いた近似式

$$\mathbf{J}_j^{l,1} = (2^l h_j)^{-1} \left(2^{-1}\mathbf{F}(\mathbf{Y}^{j+}) - 2^{-1}\mathbf{F}(\mathbf{Y}^{j-})\right)$$

で計算する。最適な h_j は各要素ごとに異なるが，列単位で計算する本手法の場合は最も大きな値に揃える必要がある。今回は任意の j に対して $h_j = 1$ と設定した。

補外計算では以下の表の初期系列より右の下三角成分を求める。

初期系列				
$\mathbf{J}_j^{1,1}$				
$\mathbf{J}_j^{2,1}$	$\mathbf{J}_j^{2,2}$			
$\vdots$	$\vdots$	$\ddots$		
$\mathbf{J}_j^{L-1,1}$	$\mathbf{J}_j^{L-1,2}$	$\cdots$	$\mathbf{J}_j^{L-1,L-1}$	
$\mathbf{J}_j^{L,1}$	$\mathbf{J}_j^{L,2}$	$\cdots$	$\mathbf{J}_j^{L,L-1}$	$\mathbf{J}_j^{L,L}$

ここで各段の計算は，

$$\mathbf{J}_j^{l,k} := \mathbf{J}_j^{l,k-1} + (4^{l-1}-1)^{-1} \left(\mathbf{J}_j^{l,k-1} - \mathbf{J}_j^{l-1,k-1}\right)$$

として行う。

この際，行列の列ごとに補外計算を行うため，収束判定については $\mathbf{J}_j^{l,k} = [\cdots J_{ij}^{l,k} \cdots]$ の各要素 $J_{ij}^{l,k}$ ごとに行う必要がある。

3.2 収束判定

収束判定式には (6) を使用する。この際，$J_{ij}^{l,k}$ の収束判定に使用する補外法の丸め誤差限界 $E_{R,ij}^{l,k}$ を，p 進 q 桁計算においては

$$E_{R,ij}^{l,k} = \frac{\max\left(\left|F_j(\mathbf{Y}^{j+})\right|, \left|F_j(\mathbf{Y}^{j-})\right|\right) \cdot c \cdot p^{-q}}{h_j / 2^l} \qquad (7)$$

とする。

よってこの $E^{l,k}_{R,ij}$ と，ユーザが与えた ε_r, ε_a を用いて，

$$\left|J^{l,k}_{ij} - J^{l,k-1}_{ij}\right| \leq \max\left(\varepsilon_r \left|J^{l,k-1}_{ij}\right| + \varepsilon_a, E^{l,k}_{R,ij}\right) \quad (8)$$

を満足していれば，その i 番目の要素は収束したと判断する。もしこの基準を満足したときには，その i 番目の要素の計算を飛ばして列単位の補外計算を続ける。従って，補外計算が停止するのは全ての列要素が収束した時となる。この収束過程を視覚的に表現すると図 1 のようになる。

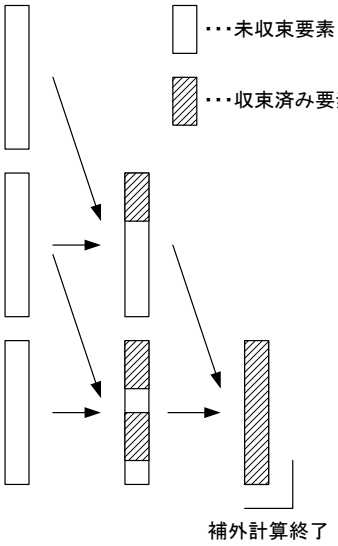

図 1: 収束過程

この収束判定により，あくまで列単位の計算を行いつつ，成分ごとの決め細やかな収束判定を行うことができるため，最大 L 段までの補外を実施したとしても，全ての要素を計算するためには最大 $2nL$ 回の関数 $\mathbf{F}(\mathbf{Y})$ 呼び出しで済む。もしこれを要素単位で行おうとすると，関数評価回数は $2n^2L$ に激増する。計算時間の多くは関数計算で占められているため，これによって全体の計算時間の大幅な縮減が期待できる。

4. 数値実験

数値実験には次のハードウェア・ソフトウェアを使用した。

ハードウェア AMD Athlon64 X2 3800+, 4GB RAM

ソフトウェア Fedora Core 4 x86_64, MPFR 2.1.2/GMP 4.1.4, GCC 4.0.2

ここで使用する $\mathbf{F}(\mathbf{Y})$ は

$$F_i(\mathbf{Y}) = \begin{cases} \sin\left(\sum_{i=1}^n Y_i\right) & (i \mod 3 = 0) \\ \cos\left(\sum_{i=1}^n Y_i\right) & (i \mod 3 = 1) \\ \prod_{i=1}^n Y_i & (i \mod 3 = 2) \end{cases} \quad (9)$$

である。この時，真の Jacobi 行列は

$$\frac{\partial F_i}{\partial Y_j} = \begin{cases} \cos\left(\sum_{i=1}^n Y_i\right) & (i \mod 3 = 0) \\ -\sin\left(\sum_{i=1}^n Y_i\right) & (i \mod 3 = 1) \\ \prod_{i=1, i\neq j}^n Y_i & (i \mod 3 = 2) \end{cases}$$

となる。

今回は $n = 30$ とし，$\mathbf{Y}= [1\ 2\ \cdots 30]^T$ における Jacobi 行列を評価した。また，あえて $\varepsilon_r = \varepsilon_a = 0$ と設定し，丸め誤差限界のみによる収束判定を行った。その結果を表 1 に示す。

表 1: 数値実験結果

計算 bit 数 (10 進桁数)	最大相対誤差	計算秒数	最大段数
128(38.5)	7.65E − 37	0.15	9
256(77.1)	2.80E − 74	0.35	13
512(154.1)	2.57E − 149	0.92	19
1024(308.3)	1.28E − 300	2.9	28
2048(616.5)	5.30E − 606	12.4	40
4096(1233)	1.76E − 1216	65.3	58
8192(2466)	2.06E − 2441	358.6	84

この結果，全ての計算桁数でそれに応じた精度が得られていることが分かる。従って，ユーザが小さすぎる $\varepsilon_r, \varepsilon_a$ を指定したとしても，丸め誤差限界値によって計算桁数に応じた精度の Jacobi 行列が得られることが示された。

次に $\varepsilon_a = 0$ とし，十分な計算桁数が確保されている時に相対許容度 ε_r によってどの程度の精度桁が得られるかを調べてみた。その結果を表 2 に示す。

表 2: 8192bit 計算の数値実験結果

$\log_{10}\varepsilon_r$	最大相対誤差	計算秒数	最大段数
50	2.11E − 51	26.5	10
100	8.90E − 102	41.6	15
200	9.12E − 201	62.6	21
500	7.34E − 506	127.5	36
1000	3.16E − 1005	215.0	52
2000	6.56E − 2001	368.0	75

ε_r の値と，相対誤差がほぼ一致していることがわかる。これにより，十分な計算桁数が確保できている場合は，相対許容度によって相対誤差の調整を行うことができる事が示された。

4.1 Testset 問題

常微分方程式の初期値問題を集めた Testset[8] 問題から，Hires 問題 (8 次元, 10 式) と Medakzo 問題 (400 次元, 11 式) に適用する。関数は以下の通り。

$$\mathbf{F}(\mathbf{Y}) = \begin{bmatrix} -1.71Y_1 + 0.43Y_2 + 8.32Y_3 + 0.0007 \\ 1.71Y_1 - 8.75Y_2 \\ -10.03Y_3 + 0.43Y_4 + 0.035Y_5 \\ 8.32Y_2 + 1.71Y_3 - 1.12Y_4 \\ -1.745Y_5 + 0.43Y_6 + 0.43Y_7 \\ -280Y_6Y_8 + 0.69Y_4 + 1.71Y_5 - 0.43Y_6 + 0.69Y_7 \\ 280Y_6Y_8 - 1.81Y_7 \\ -280Y_6Y_8 + 1.81Y_7 \end{bmatrix} \tag{10}$$

$$\begin{aligned} F_{2j-1}(t,\mathbf{Y}) &= \alpha_j \frac{Y_{2j+1} - Y_{2j-3}}{2\Delta\zeta} \\ &\quad + \beta_j \frac{Y_{2j-3} - 2Y_{2j-1} + Y_{2j+1}}{(\Delta\zeta)^2} - kY_{2j-1}Y_{2j} \\ F_{2j}(t,\mathbf{Y}) &= -kY_{2j}Y_{2j-1} \quad (j = 1, 2, ..., 199) \end{aligned}$$

ここで

$$\begin{aligned} k &= 100,\ c = 4,\ \Delta\zeta = 1/200 \\ \alpha_j &= 2(j\Delta\zeta - 1)^3/c^2, \\ \beta_j &= (j\Delta\zeta - 1)^4/c^2 \quad (j = 1, 2, ..., 200) \\ Y_{-1}(t) &= \begin{cases} 2 & (t \in (0, 5]), \\ 0 & (t \in (5, 20]) \end{cases} \\ Y_{400}(t) &= Y_{399}(t) \end{aligned} \tag{11}$$

使用した $\mathbf{Y} = [Y_1\ ...\ Y_N]^T$ は，いずれも $Y_i = i$ とした。また Medakzo 問題においては $t = 0$ とした。

計算結果を表 3, 4 に示す。いずれも最初のテスト問題同様，ほぼ計算 bit 数に近い相対誤差になっていることが分かる。非線型性が弱いためか，段数も最小で済んでいる。

5. 大規模常微分方程式の漸近安定性並列化解析

以上の Jacobi 行列計算法を，大規模な常微分方程式系の漸近安定性解析に適用してみる。テスト問題としては，既に我々が安定点とその周囲の漸近安定性を確認している (12) 式を使用する。

表 3: Hires 問題

計算 bit 数	最大相対誤差	計算秒数	最大段数
128	7.65E − 37	0.0	2
256	3.17E − 73	0.0	2
512	4.11E − 150	0.01	2
1024	2.33E − 304	0.01	2
2048	4.87E − 613	0.03	2
4096	3.51E − 1229	0.04	2
8192	5.05E − 2462	0.13	2

表 4: Medakzo 問題

計算 bit 数	最大相対誤差	計算秒数	最大段数
128	1.61E − 31	2.0	2
256	5.19E − 70	3.1	2
512	3.90E − 147	5.1	2
1024	3.27E − 301	7.4	2
2048	1.87E − 609	18.2	2
4096	4.58E − 1226	52.6	2
8192	5.98E − 2459	166.6	2

ここで，$f_j^{(i)}(t), g_j^{(i)}(t), h_j^{(i)}(t)$ は

$$\begin{aligned} f_j^{(i)}(t) &= x_{j-2}^{(i-1)}(t) + x_{j+2}^{(i-1)}(t) + x_{j-2}^{(i+1)}(t) + x_{j+2}^{(i+1)}(t) \\ g_j^{(i)}(t) &= \sum_{k=1,k\neq j}^{n} y_k^{(i)}(t) \\ h_j^{(i)}(t) &= y_j^{(i-1)}(t) + y_j^{(i+1)}(t) \end{aligned}$$

である。但し，1 番目と n 番目の細胞が繋がって円環状になっているため，この添字は

$$\begin{aligned} i - 1 < 1 \quad &\text{の時は} \quad (i - 1) + n \\ i + 1 > n \quad &\text{の時は} \quad (i + 1) - n \\ j - 2 < 1 \quad &\text{の時は} \quad (j - 2) + n \\ j + 2 > n \quad &\text{の時は} \quad (j + 2) - n \end{aligned}$$

となる。今回もパラメータとしては大槇ら [3] が使用した

$$\begin{aligned} b_x &= 1000, a_x = 10, p_x = 3 \\ b_y &= 10^{-9}, a_y = 10^{-1}, p_y = 1 \\ c_f &= 10^{-1}, p_f = 3, c_g = 1, p_g = 3 \\ c_h &= 10^{-9}, p_h = 2 \end{aligned}$$

を用いる。

$$\frac{d}{dt}\begin{bmatrix} \vdots \\ \mathbf{x}_i(t) \\ \\ \mathbf{y}_i(t) \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \dfrac{\left(h_j^{(i)}(t)\right)^{p_x}}{b_x + \left(h_j^{(i)}(t)\right)^{p_x}} - a_x x_j^{(i)}(t) \\ \vdots \\ \dfrac{\left(y_j^{(i)}(t)\right)^{p_y} + c_f\left(f_j^{(i)}(t)\right)^{p_f}}{b_y + \left(y_j^{(i)}(t)\right)^{p_y} + c_f\left(f_j^{(i)}(t)\right)^{p_f} + c_g\left(g_j^{(i)}(t)\right)^{p_g} + c_h\left(h_j^{(i)}(t)\right)^{p_h}} - a_y y_j^{(i)}(t) \\ \vdots \end{bmatrix} \tag{12}$$

(正定数：$a_x, a_y, b_x, b_y, c_f, c_g, c_h \in \mathbb{R},\ p_x, p_y, p_f, p_g, p_h \in \mathbb{N}$)

$N = 50, 200 (n = 5, 10)$ 次元まで漸近安定性は既に確認済みである [5] が，今回並列化を行うことによって，$N = 800, 1800 (n = 20, 30)$ 次元の漸近安定性を確認することが出来た。なお，(b)〜(d) までの全ての計算は 10 進 50 桁相当 (2 進 167bits) の多倍長計算で行っている。

使用した並列計算環境は以下の通りである。

使用ハードウェア

PentiumD Pentium D 820 (2.8GHz), Fedora Core 4 x86_64, 2 nodes

cs-pccluster2(Pentium4) Pentium IV 2.8GHz, Vine Linux 2.6, 11 nodes

VTPCC(Xeon) Dual Xeon 3.0GHz, Redhat 8.0, 8 nodes(max 16PEs)

使用ソフトウェア

- GMP 4.1.4, MPFR 2.1.1, BNCpack[9]
- LAM 7.1.1(PentiumD), MPICH2 1.0.1(cs-pccluster2), MPICH 1.2.5(VTPCC)
- gcc-4.0.2(PentiumD), gcc-3.4.3(cs-pccluster2), gcc-3.2(VTPCC)

5.1 漸近安定性の確認

前述した常微分方程式系の場合，今回調査した 200, 800, 1800, 3200 次元全てにおいて，平衡点における Jacobi 行列は対角優位非対称行列となり，対角成分は −10 もしくは −0.0990 のどちらである。全ての次元数において，6 回の Newton 反復で収束することが確認できた。また，これらの平衡点のユークリッドノルム値 $\|\mathbf{Y}^*\|_2$ と，$t = 100$ における常微分方程式の数値解のノルム値 $\|\mathbf{Y}(100)\|_2$ はかなり近いものであることが確認できた (表 5)。

表 5: $\mathbf{Y}^*$ と $\mathbf{Y}(100)$(上)，Gerschgorin disc(中, 下)

N	$\|\mathbf{Y}^*\|_2$	$\|\mathbf{Y}(100)\|_2$
200	3.16236E + 1	3.16242E + 1
800	4.47224E + 1	4.47230E + 1
1800	5.47736E + 1	5.47732E + 1
3200	6.32448E + 1	6.32471E + 1

N	max radius for −10
200	$\lvert\lambda - (-1.00\text{E}+1)\rvert \le 1.50\text{E}-1$
800	$\lvert\lambda - (-1.00\text{E}+1)\rvert \le 1.50\text{E}-1$
1800	$\lvert\lambda - (-1.00\text{E}+1)\rvert \le 1.50\text{E}-1$
3200	$\lvert\lambda - (-1.00\text{E}+1)\rvert \le 1.50\text{E}-1$

N	max radius for −0.0990
200	$\lvert\lambda - (-9.90\text{E}-2)\rvert \le 1.23\text{E}-5$
800	$\lvert\lambda - (-9.90\text{E}-2)\rvert \le 1.26\text{E}-5$
1800	$\lvert\lambda - (-9.90\text{E}-2)\rvert \le 1.29\text{E}-5$
3200	$\lvert\lambda - (-9.90\text{E}-2)\rvert \le 1.32\text{E}-5$

更に，各対角成分ごとに Gerschgorin disc の半径を計算してみると，最大でも 0.15 もしくは 1.23×10^{-5} 〜1.32×10^{-5} であり，固有値を計算するまでもなく，その実数部は負になることが判明した。

なお，$N = 3200$ の結果は PentiumD クラスタのみで実行できたものである。これは後述するように，x86_64 Dual-core CPU を用いたことによる性能向上と，メモリ容量が 3 種のクラスタの中で一番大きい (4GB) ことによって初めて計算が可能となった。

5.2 並列化による計算時間の縮減効果

1(b)〜(d) の総計算時間は，表 6 のようになった。×印は実行が不可能であったことを示す。

表 6: (b)〜(d) 全体の最短計算時間 (単位:秒)

N	VTPCC	cs-pccluster2	PentiumD
200	78.0 (5PEs)	187.7 (2PEs)	76.1 (2PEs)
800	672.6 (20PEs)	2002.4 (10PEs)	978.1 (4PEs)
3200	×	×	13453.1 (4PEs)

以下，計算時間の内訳を，$N = 800(n = 20)$ の場合に限って，詳細に見ることにする。まず，Jacobi 行列の並列化の効果を 2 に示す。

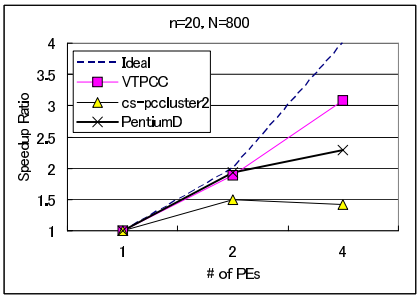

図 2: Jacobi 行列の並列化の効率

理想的には PE 数に比例して性能向上があるところが，最も効率の良い VTPCC でも 4PE では 3 倍程度に留まっており，最も性能向上が見られる 20PE でも 4 倍程度までしか上がらないことが実験から判明している。これは各 PE で分散して $\mathbf{F}(\mathbf{Y})$ の評価をしている所で必ず一度集団通信 Allgather を実行していることが響いていると思われる。逆に言えば，現状の実装ではこの集団通信の実行回数を減らす必要があると言える。

最後に，3 に，cs-pccluster2 を 1 とした時の性能向上比を示す。

1PE，2PE では PentiumD クラスタが VTPCC よりも高性能であるが，通信性能の悪さ [6] が足を引っ張った結果，4PE では逆に PentiumD の方が劣る結果となっている。それでも全体的には現状の cs-pccluster2 に比べて 2 倍以上の性能向上が見られることから，より大次元の問題に対してはこの程度の時間短縮が期待できると言える。

6. 結論と今後の課題

多倍長計算を用いた数値実験により，本手法の有効性が示された。一般的には AD が使用できる関数

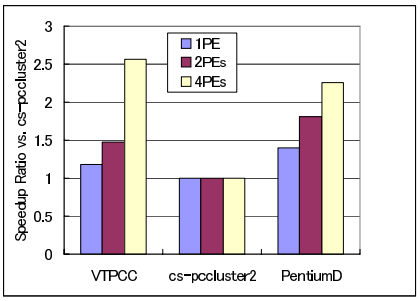

図 3: 対 cs-pccluster2 性能向上比

に対しては，計算量の点で不利であると思われるが，AD が使用できない関数や計算環境において，高精度な Jacobi 行列が必要な時には一つの手段として使用できるものである。

今後の課題としては

- より大規模な常微分方程式系の漸近安定性解析を行う
- AD を効率的に使用する方法の考案
- 並列化した際の計算時間予測

ということが挙げられる。

参考文献

[1] 永坂秀子・福井義成, “数値微分の誤差”, 情報処理学会論文誌, vol.22, No.5, pp.411-416.

[2] 福井義成, “数値微分における補外法の誤差”, 日本応用数理学会論文誌 Vol.15, No.4, pp.521-535, 2005.

[3] 大椙弘順・他，単純化した相互作用ルールによる擬似細胞のパターン形成と再生, 静岡理工科大学紀要 Vol.11, 2003.

[4] 幸谷智紀・大椙弘順, “多倍長計算を用いた仮想化細胞間モデルの漸近安定性解析”, To appear.

[5] 幸谷智紀・大椙弘順, 仮想化した細胞間モデルの安定性解析, HOKKE2005, 2005.

[6] 幸谷智紀, x86_64 Dual-core PC cluster の性能評価, HOKKE2006, 2006.

[7] Automatic Differentiation, `http://www.autodiff.org/`

[8] Test Set for Initial Value Problem Solvers, `http://pitagora.dm.uniba.it/~testset/`

[9] BNCpack, `http://na-inet.jp/na/bnc/`

[10] GMP, `http://swox.com/gmp/`

[11] MPFR, `http://www.mpfr.org/`

[12] LAM/MPI, `http://www.lam-mpi.org/`