数理解析研究所講究録
第643巻 1988年 129-153

ROW法について

名古屋大学工学部情報工学科　森山貴志(Takashi Moriyama)
名古屋大学工学部情報工学科　三井斌友(Taketomo Mitsui)

1. Stiffな問題とROW法

常微分方程式の離散変数法を，ステップ幅 h を固定して $y'=\lambda y$ $(Re\ \lambda<0)$ の問題に適用して，その解 y_n が $n\to\infty$ で 0 に収束するとき絶対安定という．絶対安定になる $h\lambda$ の領域を，絶対安定領域という．陽的な公式では，絶対安定領域は，複素左半平面で左側有界となる．ところが，stiffな問題では，$\partial f/\partial y$ の固有値を $\lambda_1\cdots\lambda_d$ とすると

$$Re\ \lambda_i<0 \qquad (i=1\cdots d) \quad \text{で}$$

$$\frac{\max_i|Re\ \lambda_i|}{\min_i|Re\ \lambda_i|} \qquad \text{(硬度比)}$$

が非常に大きい値になる．そしてこの問題を安定に解くためには，すべての固有値に対して $h\lambda$ が絶対安定領域に入らなくては，ならない．実数部分の絶対値最大の固有値に対して，小さくステップ幅 h を選び，さらに実数部分絶対値最小の固有値の成分が十分減衰するまでの長い区間を積分することになり，数値的な困難が生じる．したがってステップ幅 h に対する制限を取り除いて，安定に解くた

めには，絶対安定領域が左半平面で左側有界でないことが要求される．そしてこのような方法は，必ず陰的な方法になることが示されている．　陰的な方法では，積分の各段階で連立の非線型方程式を解かなくてはならない．例えば，P 段の陰的 Runge-Kutta 法では，$y(x_0+h)$を求めるのに次式を解くことになる．

（問題は，$y'=f(x,y)$，　$y(x_0)=y_0$　）

$$k_1=f(x_0+\alpha_1 h, y_0+h\sum_{l=1}^{p}\beta_{1l}k_l)$$

$$k_2=f(x_0+\alpha_2 h, y_0+h\sum_{l=1}^{p}\beta_{2l}k_l)$$

$$\vdots$$

$$k_p=f(x_0+\alpha_p h, y_0+h\sum_{l=1}^{p}\beta_{pl}k_l)$$

この方程式を，逐次代入法で解くとする．つまり，

$$k_j^{[t+1]}=f(x_0+\alpha_j h, y_0+h\sum_{l=1}^{j-1}\beta_{jl}k_l^{[t+1]}+h\sum_{l=j}^{p}\beta_{jl}k_l^{[t]})$$

$f(x,y)$は，次のリプシッツ連続性が成り立っていて，リプシッツ定数をLとする．

$$\|f(x,y)-f(x,z)\|\leq L\|y-z\|$$

さらに

$$b_L=\max(|\beta_{21}|,|\beta_{31}|+|\beta_{32}|,\cdots,|\beta_{p1}|+\cdots+|\beta_{pp-1}|)$$
$$b_M=\max(|\beta_{11}|+|\beta_{12}|+\cdots+|\beta_{1p}|,|\beta_{22}|+\cdots+|\beta_{2p}|,\cdots,|\beta_{pp}|)$$

を定義すると，逐次代入法が，収束するためには，h に対して次の条件が必要になる．[1]

$$h < \frac{1}{(b_L + b_M)L} \qquad (1)$$

また陰的な線型多段階法や半陰的Runge-Kutta法でも，同様な条件が必要になる．

ところが stiff な問題では，L が硬度比程度大きいので，結局hを小さく選ばなくてはならない制限が課せられる．そこでこの制限をなくすために，非線型方程式を解くのにニュートン法を使うことになる．ニュートン法を使えば，逐次代入法より収束性がよくなりhに対する (1) 式のような厳しい条件は，なくなる．ところがニュートン反復を計算するには，ヤコビアン行列$\partial f/\partial y$が必要になる．そこで Runge-Kutta 公式の中にヤコビアン行列を取り入れた方法が考えられた．

それがROW法 (Rosenbrock-Wannner 法) である．

簡単のために問題を，$y' = f(y)$, $y(x_0) = y_0$とするとROW法は，

$$E = I - h\gamma \frac{\partial f}{\partial y}(y_0)$$

$$Ek_1 = hf(y_0)$$

$$Ek_j = hf(y_0 + \sum_{l=1}^{j-1} a_{jl}k_l) + \sum_{l=1}^{j-1} c_{jl}k_l \qquad (j = 2 \cdots s)$$

$$y_{new} = y_0 + \sum_{i=1}^{s} b_i k_i$$

KapsとRentropは，4 段($s=4$)，4 次のROW法で，誤差評価のための3次公式を埋め込んだA－安定（絶対安定領域が左半平面を含む）なものを作った[2]．この公式では，$a_{41}=a_{31}$, $a_{42}=a_{32}$, $a_{43}=0$だから，4 段公式であるにもかかわらず1 ステップでの関数計算が3回だけでよい．

2．　ヤコビアン行列の誤差

問題が非線型で連立数が小さくないときは，解析的なヤコビアン行列を求めるのは，実際的でないと考えられてきた．そこで一般的には，数値微分が使われている．ニュートン反復でヤコビアン行列を使う場合には，ヤコビアン行列に含まれる誤差は，収束までの反復の回数に影響を与えるが，解の精度には影響を与えないと考えられる．

ところがROW法では，公式の中にヤコビアン行列が含まれていて，公式を作るときには，ヤコビアン行列は，正確であると考えられているので，実際の計算上はヤコビアン行列の誤差の結果に与える影響は重要である．そこでヤコビアン行列の誤差が精度，安定性に与える影響を考察した．

精度への影響

Kaps-Rentropの公式は，次のように書くことができる．[3]

1. $g_1 = f(y_0)$
2. $f' = \dfrac{\partial f}{\partial y}(y_0)$
3. $E = I - h\gamma f'$
4. $Ek_1 = hg_1$

5. $g_2 = f(y_0 + a_{21}k_1)$
6. $Ek_2 = hg_2 + c_{21}k_1$
7. $g_3 = f(y_0 + a_{31}k_1 + a_{32}k_2)$
8. $Ek_3 = hg_3 + c_{31}k_1 + c_{32}k_2$
9. $Ek_4 = hg_3 + c_{41}k_1 + c_{42}k_2 + c_{43}k_3$
10. $y_{new} = y_0 + b_1k_1 + b_2k_2 + b_3k_3 + b_4k_4$
11. $error = e_1k_1 + e_2k_2 + e_3k_3 + e_4k_4$

（局所的離散化誤差の評価）

係数は，付録に与えた．

このアルゴリズムでヤコビアン行列に誤差が含まれたとする．正確なヤコビアン行列をJ，誤差の行列をΔとして誤差の含まれている量に~をつける．

1. $g_1 = f(y_0)$
2. $f' = J \qquad \tilde{f}' = J + \Delta$
3. $E = I - h\gamma f' = I - h\gamma J$

   $\tilde{E} = I - h\gamma \tilde{f}' = I - h\gamma J - h\gamma\Delta = E - h\gamma\Delta$
4. $Ek_1 = hg_1$

   $\tilde{E}\tilde{k}_1 = hg_1$

   $(E - h\gamma\Delta)\tilde{k}_1 = Ek_1$

   $\tilde{k}_1 = (E - h\gamma\Delta)^{-1}Ek_1$

   ここで $(E - h\gamma\Delta)^{-1} = [E(I - h\gamma E^{-1}\Delta)]^{-1}$

   $= (I - h\gamma E^{-1}\Delta)^{-1}E^{-1}$

   Δ は，小さいのでΔの2乗以上は無視すると

   $$(E - h\gamma\Delta)^{-1} \simeq (I + h\gamma E^{-1}\Delta)E^{-1}$$

したがって

$$\tilde{E}^{-1} \simeq (I + h\gamma E^{-1}\Delta)E^{-1}$$
$$\tilde{k}_1 \simeq k_1 + h\gamma E^{-1}\Delta k_1$$

5. $$g_2 = f(y_0 + a_{21}k_1)$$
$$\tilde{g}_2 = f(y_0 + a_{21}\tilde{k}_1)$$
$$= f(y_0 + a_{21}k_1 + a_{21}h\gamma E^{-1}\Delta k_1)$$
$$\simeq f(y_0 + a_{21}k_1) + \frac{\partial f}{\partial y}(y_0 + a_{21}k_1)[a_{21}h\gamma E^{-1}\Delta k_1]$$
$$\simeq g_2 + a_{21}h\gamma f' E^{-1}\Delta k_1$$
$$\left(\frac{\partial f}{\partial y}(y_0 + a_{21}k_1) \simeq \frac{\partial f}{\partial y}(y_0)\right)$$

6. $$Ek_2 = hg_2 + c_{21}k_1$$
$$\tilde{E}\tilde{k}_2 = h\tilde{g}_2 + c_{21}\tilde{k}_1$$
$$= hg_2 + a_{21}h^2\gamma f' E^{-1}\Delta k_1 + c_{21}k_1 + c_{21}h\gamma E^{-1}\Delta k_1$$
$$\simeq Ek_2 + c_{21}h\gamma E^{-1}\Delta k_1 \qquad (h^2\text{の項を無視する})$$
$$\tilde{k}_2 = (I + h\gamma E^{-1}\Delta)k_2 + c_{21}h\gamma(I + h\gamma E^{-1}\Delta)E^{-2}\Delta k_1$$
$$\simeq k_2 + h\gamma E^{-1}\Delta k_2 + c_{21}h\gamma E^{-2}\Delta k_1$$

7. $$g_3 = f(y_0 + a_{31}k_1 + a_{32}k_2)$$
$$\tilde{g}_3 = f(y_0 + a_{31}\tilde{k}_1 + a_{32}\tilde{k}_2)$$
$$= f(y_0 + a_{31}k_1 + a_{32}k_2 + a_{31}h\gamma E^{-1}\Delta k_1$$
$$+ a_{32}h\gamma E^{-1}\Delta k_2 + a_{32}c_{21}h\gamma E^{-2}\Delta k_1)$$
$$\simeq f(y_0 + a_{31}k_1 + a_{32}k_2)$$
$$+ f'(a_{31}h\gamma E^{-1}\Delta k_1 + a_{32}h\gamma E^{-1}\Delta k_2 + a_{32}c_{21}h\gamma E^{-2}\Delta k_1)$$
$$= g_3 + f'(a_{31}h\gamma E^{-1}\Delta k_1 + a_{32}h\gamma E^{-1}\Delta k_2 + a_{32}c_{21}h\gamma E^{-2}\Delta k_1)$$

8. $$Ek_3 = hg_3 + c_{31}k_1 + c_{32}k_2$$
$$\tilde{E}\tilde{k}_3 = h\tilde{g}_3 + c_{31}\tilde{k}_1 + c_{32}\tilde{k}_2$$
$$\simeq hg_3 + c_{31}k_1 + c_{31}h\gamma E^{-1}\Delta k_1 + c_{32}k_2$$
$$+ c_{32}h\gamma E^{-1}\Delta k_2 + c_{32}c_{21}h\gamma E^{-2}\Delta k_1$$
$$= Ek_3 + c_{31}h\gamma E^{-1}\Delta k_1 + c_{32}h\gamma E^{-1}\Delta k_2 + c_{32}c_{21}h\gamma E^{-2}\Delta k_1$$
$$\tilde{k}_3 = k_3 + h\gamma E^{-1}\Delta k_3 + c_{31}h\gamma E^{-2}\Delta k_1$$
$$+ c_{32}h\gamma E^{-2}\Delta k_2 + c_{32}c_{21}h\gamma E^{-3}\Delta k_1$$

9. $$Ek_4 = hg_3 + c_{41}k_1 + c_{42}k_2 + c_{43}k_3$$
$$\tilde{E}\tilde{k}_4 = h\tilde{g}_3 + c_{41}\tilde{k}_1 + c_{42}\tilde{k}_2 + c_{43}\tilde{k}_3$$
$$= Ek_4 + c_{41}h\gamma E^{-1}\Delta k_1 + c_{42}h\gamma E^{-1}\Delta k_2 + c_{42}c_{21}h\gamma E^{-2}\Delta k_1$$
$$+ c_{43}h\gamma E^{-1}\Delta k_3 + c_{43}c_{31}h\gamma E^{-2}\Delta k_1 + c_{43}c_{32}h\gamma E^{-2}\Delta k_2$$
$$+ c_{43}c_{32}c_{21}h\gamma E^{-3}\Delta k_1$$
$$\tilde{k}_4 = k_4 + h\gamma E^{-1}\Delta k_4 + c_{41}h\gamma E^{-2}\Delta k_1 + c_{42}h\gamma E^{-2}\Delta k_2$$
$$+ c_{42}c_{21}h\gamma E^{-3}\Delta k_1 + c_{43}h\gamma E^{-2}\Delta k_3 + c_{43}c_{31}h\gamma E^{-3}\Delta k_1$$
$$+ c_{43}c_{32}h\gamma E^{-3}\Delta k_2 + c_{43}c_{32}c_{21}h\gamma E^{-4}\Delta k_1$$

10. $$y_{new} = y_0 + b_1k_1 + b_2k_2 + b_3k_3 + b_4k_4$$
$$\tilde{y}_{new} = y_0 + b_1\tilde{k}_1 + b_2\tilde{k}_2 + b_3\tilde{k}_3 + b_4\tilde{k}_4$$
$$= y_{new} + h\gamma[E^{-1}(b_1\Delta k_1 + b_2\Delta k_2 + b_3\Delta k_3 + b_4\Delta k_4)$$
$$+ E^{-2}(b_2c_{21}\Delta k_1 + b_3c_{31}\Delta k_1 + b_3c_{32}\Delta k_2 + b_4c_{41}\Delta k_1$$
$$+ b_4c_{42}\Delta k_2 + b_4c_{43}\Delta k_3)$$
$$+ E^{-3}(b_3c_{32}c_{21}\Delta k_1 + b_4c_{43}c_{31}\Delta k_1 + b_4c_{43}c_{32}\Delta k_2$$
$$+ b_4c_{42}c_{21}\Delta k_1)$$
$$+ E^{-4}(b_4c_{43}c_{32}c_{21}\Delta k_1)]$$

$\tilde{y}_{new}$ から誤差を推定することは，難しいが $\Delta k_1 = \Delta k_2 = \Delta k_3 = \Delta k_4 := \Delta k$ と仮定して係数の関係式（ [2],[3] ）を使うと，

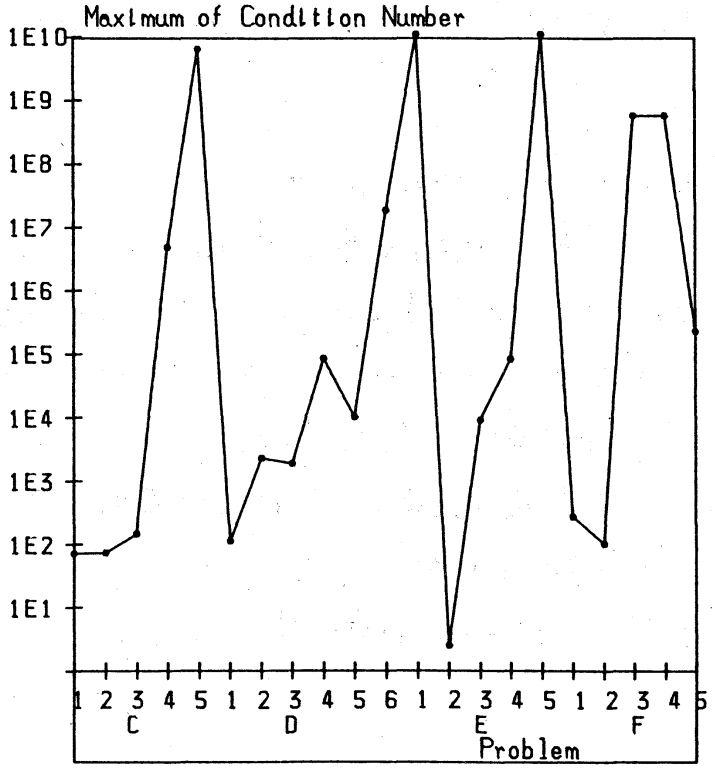
$$\tilde{y}_{new} = y_{new} + h\gamma E^{-4}\Delta k + O(h^2)$$

$$E = I - h\gamma J, \quad \gamma = 0.395$$

h の値は，ステップ幅の自動調整のために，どのような値になるかわからない．そのため E は，どのような性質の行列かわからないが，悪条件の行列ならば，ヤコビアン行列の誤差 Δ の影響は，大きくなると考えられる．

[3],[4] にある **stiff** な問題で非線型なものについて行列 $\mathbf{E}$ の条件数の最大値を計算した．

（誤差の許容量 10^{-4}，条件数の計算は [5] のプログラムを使用．
横軸は問題の種類，縦軸は条件数の最大値）



安定性への影響

[2] でKapsとRentropの4次公式は，A－安定であることが示されている．しかし，そこではヤコビアン行列は正確であることが仮定されているので，ヤコビアン行列に誤差が含まれたとき，絶対安定領域がどのように変化するかを調べた．

問題 $y'=\lambda y$ を適用してアルゴリズムの3のEの計算でλを$\lambda+d$として

$$E=1-h\gamma f'=1-h\gamma(\lambda+d)$$

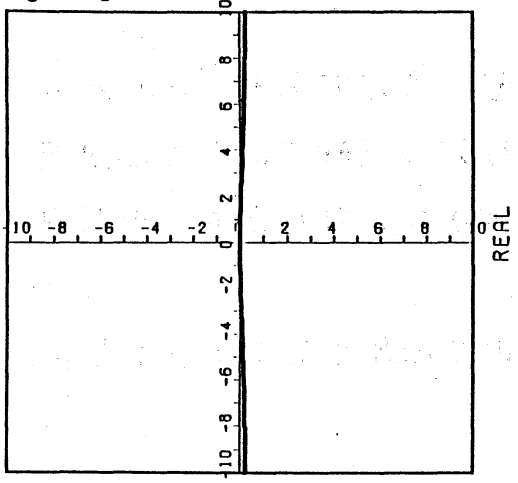
として計算してみると

$$y_{new}=R(z,\delta)y_0,\quad z=h\lambda,\quad \delta=hd$$

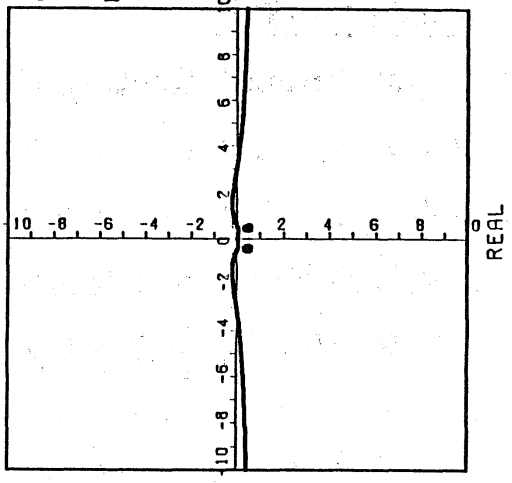
$R(z,\delta)$の具体的な式は省略するが，これを安定性関数といって$|R(z,\delta)|<1$となるzの領域を絶対安定領域という．[2] によると$|R(z,0)|<1$となる領域は左半平面を含むことが示されている．問題が$y'=(\lambda+d)y$でヤコビアン行列の計算が正確ならば，安定性関数は，$R(z+\delta,0)$となるが，$\delta\neq 0$のときは，$R(z,\delta)\neq R(z+\delta,0)$である．つまり$\delta$の値によって絶対安定領域は位置がずれるだけでなく形も変わる．

数値計算により，$\delta=0,\pm1,i,1+i,-1+i,\pm2,2i,2+2i,$
$-2+2i$ での絶対安定領域を調べた．すべての図で境界の左側が $|R(z,\delta)|<1$ つまり絶対安定領域である．

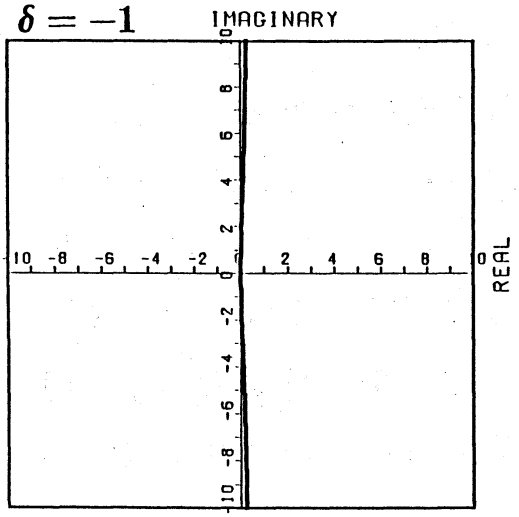
$\delta = 0$



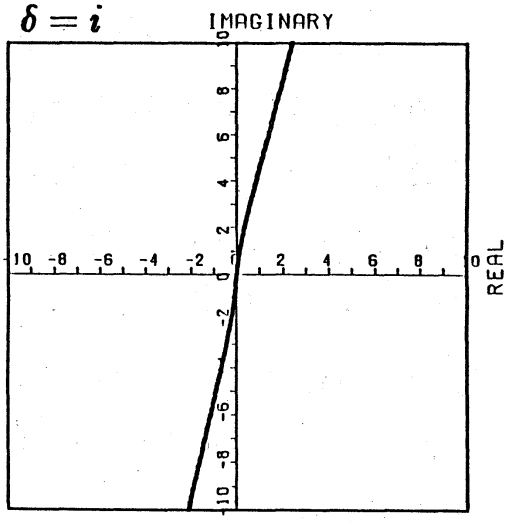
$\delta = 1$



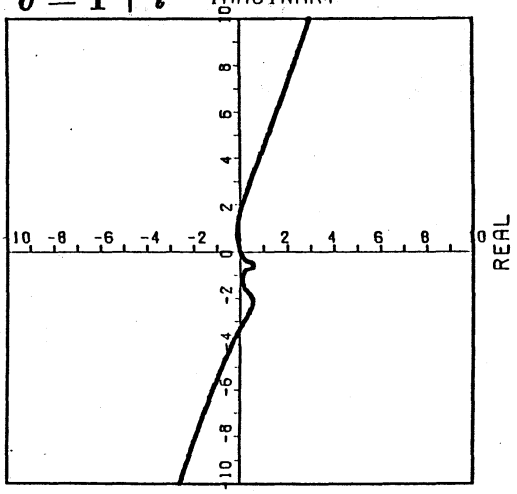
$\delta = -1$



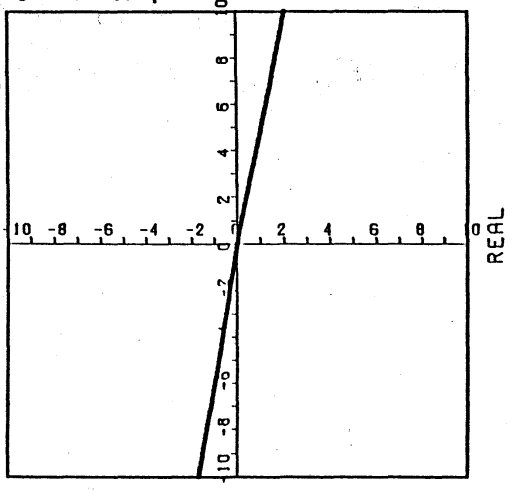
$\delta = i$



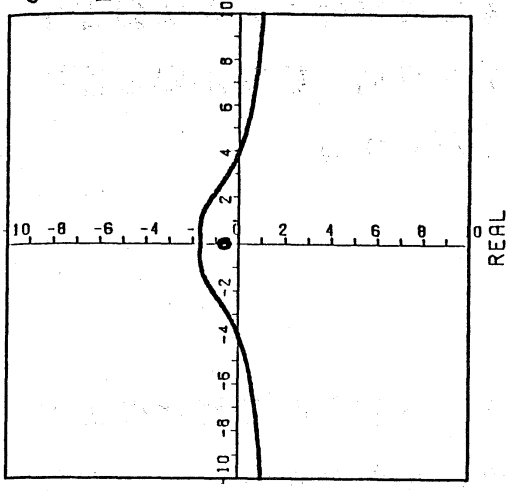
$\delta = 1 + i$



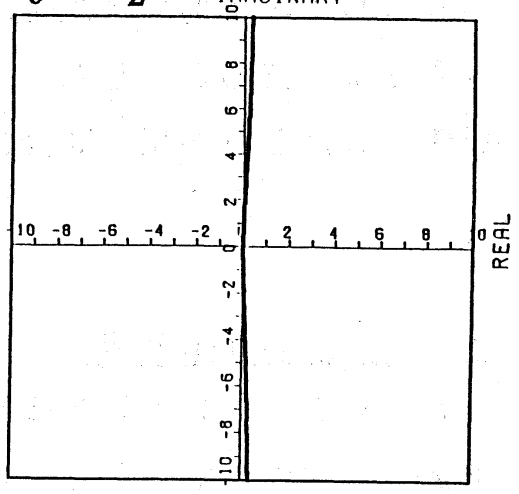
$\delta = -1 + i$



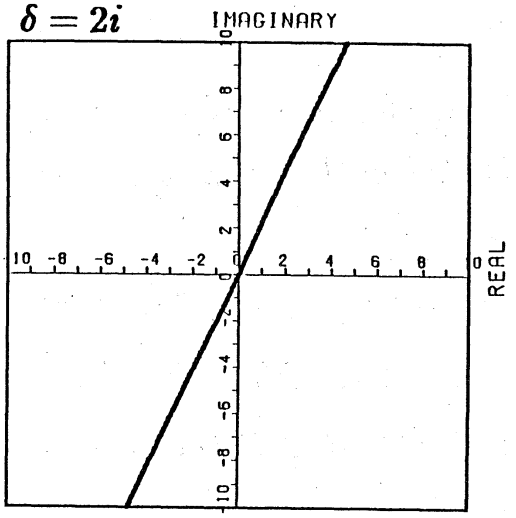
$\delta = 2$



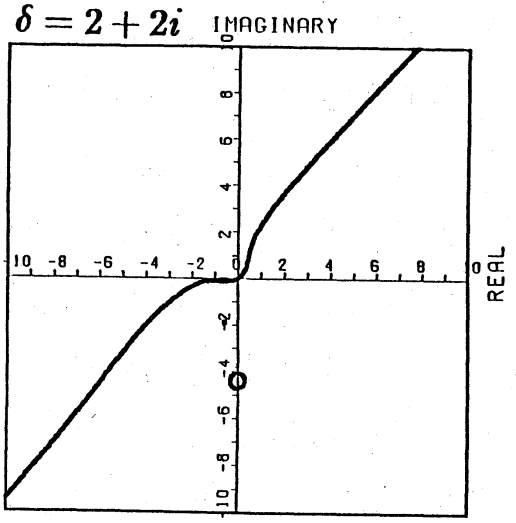
$\delta = -2$



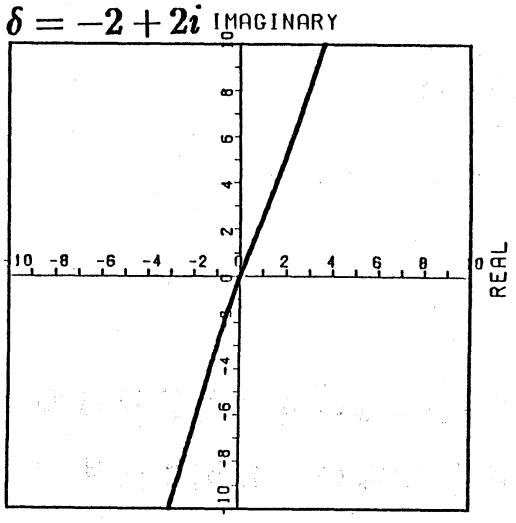
$\delta = 2i$



$\delta = 2 + 2i$



$\delta = -2 + 2i$



//

計算してみた例では，$Im\ \delta \neq 0$ のときに安定性に大きな影響を与えるようである．したがって ヤコビアン行列の誤差が安定性に影響するかどうかは，問題に依存することがわかる．

3． 高速微分法

ヤコビアン行列の誤差の影響は，少なくとも数学的な解析の上では生じうることが分かってきたから，解析的なヤコビアン行列の計算を結合することを考え，実行した．その際，数式処理的な側面を持つ伊理の高速微分法 [6],[7] を利用した．

高速微分法は，四則演算や初等関数（ exp, log, sin など）の組み合わせによってできている関数をその関数値を計算する途中の中間値を利用して，偏導関数値を計算する方法である．

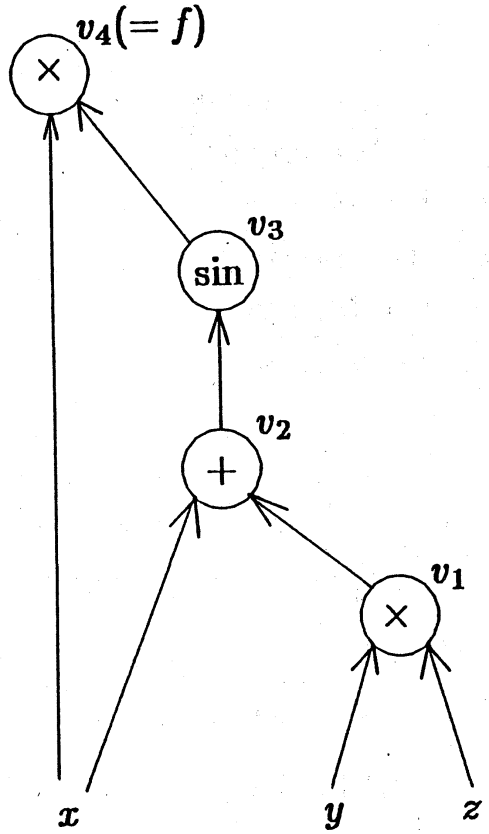
例を使って簡単に説明する． $f = x\sin(x+yz)$ という関数を考える．この関数の関数値と偏導関数値を，解析的に計算するには，普通は偏導関数を入力するか，または数式処理により次のような式を与えて，数値計算することになる．

$$
\begin{aligned}
f &= x\sin(x+yz) \\
\frac{\partial f}{\partial x} &= \sin(x+yz) + x\cos(x+yz) \\
\frac{\partial f}{\partial y} &= xz\cos(x+yz) \\
\frac{\partial f}{\partial z} &= xy\cos(x+yz)
\end{aligned}
$$

ところが，ここで yz, $\sin(x+yz)$, $x\cos(x+yz)$ などの計算は同じ計算を何回か実行することになり，能率的でない．高速微分法で

は，これらの値を再利用して偏導関数を計算する．

まず計算する過程を表現する計算グラフ (Kantorovichグラフ) を考える．関数計算を四則演算や初等関数に分割して，各演算結果を中間変数とする．そして入力変数，定数，中間変数を頂点とするグラフを考えて，計算される順番にしたがって，有向枝を与える．上の例の $f = x\sin(x+yz)$ では次の図のようになる．



各中間変数（演算結果）に図のような v_1, v_2, v_3, v_4 と名前をつける．この図にしたがって関数 f を計算して $v_1 \cdots v_4$ を記憶しておく．そして各中間変数に対して，その演算の入力変数に関する偏導関数（要素的偏導関数）を計算する．つまり

$$\frac{\partial v_3}{\partial v_2} \quad \frac{\partial v_2}{\partial v_1} \quad \frac{\partial v_2}{\partial x}$$

などである．要素的偏導関数は，中間変数から計算できる．

そうすると，関数 f の x に関する偏導関数は，計算グラフ上で x から f へ枝をたどり，途中の要素的偏導関数の積を計算して，x から f へ至るすべてのパスについて足し合わせたものになる．

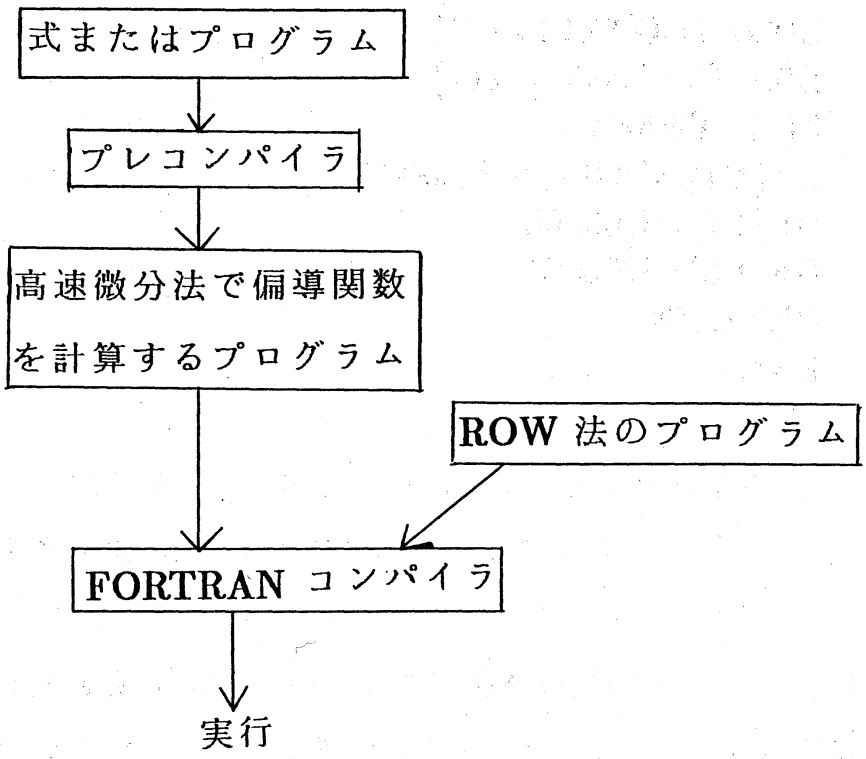
例では，次のようになる．

$$\frac{\partial f}{\partial x} = \frac{\partial v_4}{\partial x} + \frac{\partial v_4}{\partial v_3}\frac{\partial v_3}{\partial v_2}\frac{\partial v_2}{\partial x}$$
$$\frac{\partial f}{\partial y} = \frac{\partial v_4}{\partial v_3}\frac{\partial v_3}{\partial v_2}\frac{\partial v_2}{\partial v_1}\frac{\partial v_1}{\partial y}$$
$$\frac{\partial f}{\partial z} = \frac{\partial v_4}{\partial v_3}\frac{\partial v_3}{\partial v_2}\frac{\partial v_2}{\partial v_1}\frac{\partial v_1}{\partial z}$$

このようにして，偏導関数を計算すれば，n 変数の関数を n によらず関数計算の定数倍の手間で行うことができ，数値微分や偏導関数を与えた場合より高速化することができる．高速微分法を実際に使うためには，関数式または，関数計算のサブルーチンを入力すると，高速微分法で関数と偏導関数を計算するプログラムを出力するプレコンパイラが必要である．

4．　高速微分法のためのプレコンパイラ

高速微分法のためのプレコンパイラ




```
-input-
Y(1)*SIN(Y(1)+Y(2)*Y(3))
-output-
      SUBROUTINE FJAC(Y, F, DF)
      DIMENSION Y(50), F(50), DF(50,50)
      V10001=Y(2)*Y(3)
      D00000=Y(3)
      D00001=Y(2)
      V10002=Y(1)+V10001
      D00002=1.0
      V10003=SIN(V10002)
      D00000=D00000*COS(V10002)
      D00001=D00001*COS(V10002)
      D00002=D00002*COS(V10002)
```

```
V10004=Y(1)*V10003
D00003=V10003
D00000=D00000*Y(1)
D00001=D00001*Y(1)
D00002=D00002*Y(1)
F(1)=V10004
DF(1,1)=D00002+D00003
DF(1,2)=D00000
DF(1,3)=D00001
RETURN
END
```

このような動作をするプレコンパイラをつくるには，コンパイラ生成系を使う方法と，すべてのプログラムを手書きする方法が，考えられる．

[12]では，小型の言語PLANのコンパイラの作成を例に，2 つの方法を比較している．コンパイラ生成系としてUNIXの lex とyacc を，取り上げているが，この生成系はCのプログラムを生成するので，条件を同じにするために手書きの方は，C 言語を使っている．

	ソースプログラムの大きさ（行）	字句解析 s	構文解析 s	計 s
lex+yacc	1045	4.4	3.0	7.4
C	1159	1.7	2.1	3.8

その結果，生成系で作ったコンパイラの方が2 倍程度の時間がかかる．しかしこのプレコンパイラの場合は，出力したプログラムを

再びFORTRANコンパイラに入力するので，全体の処理時間の大部分はFORTRANコンパイラでかかると考えられる．したがってプレコンパイラでの2倍程度の違いはほとんど全体の処理時間に影響しないことになる．このことと，信頼性，保守，修正，作成，の容易さは，コンパイラ生成系の方が優れていることを考えて，コンパイラ生成系を使うことにした．

lexと yacc は，UNIX上のCのプログラムジェネレータである．lex は正規表現を使ったソースを与えると，それをもとに字句解析をするCのプログラムを生成する．字句解析とは，与えられた文字列から文法上意味のある文字列（トークン，終端記号）を分離する．

lex

字句解析 ← 正規表現

lexical analyzer　　lexical rules

Y(1)*SIN(Y(1)+Y(2)*Y(3))

トークン

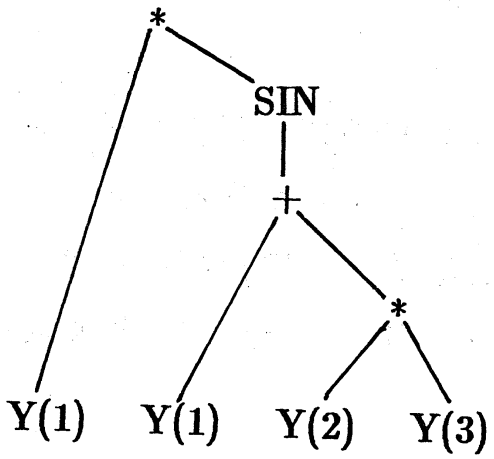
終端記号

またyaccは，構文規則を書いたソースを与えると構文解析をするCのプログラムを生成する．これは，トークンから構文木を作るものである．

yacc (yet another compiler compiler)

構文解析 ← 構文規則

parser grammar rules



lex source

```
%%
[ \ t \ n]      {}
";"       return(EOL);
[0−9]+       { yylval=atoi(yytext);
               return(number); }
[0−9]*\ .[0−9] *       { yylvalf=atof(yytext);
                         return(numberf); }
[0−9]*\ .[0−9] * e[+−] * [0−9] *       {
                    yylvalf=atof(yytext);
                    return(numberf); }
"**"       return(PWR);
"y("[0−9]+")"       { yylval=atoi(yytext+2);
                    return(VAR); }
"sin"       { yylval=7;    return(FUNC1); }
"cos"       { yylval=8;    return(FUNC1); }
"tan"       { yylval=9;    return(FUNC1); }
```

```
"exp"       { yylval=10;    return(FUNC1); }
"log"       { yylval=11;    return(FUNC1); }
.       return(*yytext);
%%
```

yacc source

```
/*---declarations---*/
%{
#include <stdio.h>
#include <cytpe.h>
#include <math.h>
static float yylvalf;
%}
%token number numberf PWR EOL VAR FUNC1
%left '+' '-'
%left '*' '/'
%left MINUS
%right PWR
/*---rules---*/
%%
line    : /* empty */
        |   line EOL
        |   line expr EOL
                { form($2);}
        |   line error EOL
        ;
expr    : '(' expr ')' { $$=$2;}
        |       expr '+' expr { $$=sub(1,$1,$3); }
        |       expr '-' expr { $$=sub(2,$1,$3); }
        |       expr '*' expr { $$=sub(3,$1,$3); }
        |       expr '/' expr { $$=sub(4,$1,$3); }
        |       '-' expr %prec MINUS { $$=sub(5,$2,$2); }
        |       expr PWR expr { $$=sub(6,$1,$3); }
        |       VAR { $$=$1; }
```

```
        |       FUNC1 '(' expr ')' { $$=sub($1,$3,$3); }
        |       number { $$=stcns0($1); }
        |       numberf { $$=stcns1(yylvalf); }
        ;
%%
#include "lex.yy.c"
```

5. プレコンパイラの評価

[3][4] にある問題を例に，次のことをテストした．

（A） プレコンパイラとFORTRANコンパイラの処理時間の比較

（B） 数値微分と高速微分法での，微分計算の時間の比較

（C） 数値微分と高速微分法を使った場合のROW 法の計算時間の比較

（D） 数値微分と高速微分法を使った場合のROW 法の計算精度の比較

プレコンパイラの速度

	pre-compiler(s)	fortran77 compiler(s)	
n=2	0.007	0.25	2.8%
n=3	0.008	0.50	1.6%
n=4	0.009	0.62	1.4%
n=8	0.013	1.27	1.0%

FUJITSU A-600 UTS

微分の計算時間の比較　　(1000回 msec)

	数値微分	高速微分法	比
n=2　4 間	208	68	3.06
n=3　7 間	625	145	4.31
n=4　9 間	1920	531	3.62
n=8　1 間	389	42	9.26

ROW法の計算時間の比較　　(tolerance=10^{-4},　msec)

	数値微分	高速微分法	
n=2　4 間	11.24	10.44	7.1%
n=3　7 間	106.33	96.18	9.5%
n=4　9 間	126.45	110.43	12.7%
n=8　1 間	13.80	11.60	15.9%

ROW法での計算精度の比較　(tolerance=10^{-10})

相対誤差の最大値

	数値微分	高速微分法
D−4	3.84×10^{-9}	2.76×10^{-9}
D−5	1.55×10^{-10}	8.81×10^{-11}
E−1	1.29×10^{-11}	2.45×10^{-12}
E−2	2.25×10^{-11}	1.59×10^{-11}
E−4	9.11×10^{-10}	7.34×10^{-10}

FACOM　M-780/20

結果として，次のことが言える．

（A） プレコンパイラの処理時間は，FORTRAN コンパイラの処理時間に比べて，無視できる程小さい，また問題の次元数が大きくなれば，その比率は小さくなる．

（B） 次元数をnとすると，高速微分法は数値微分の 1/n 程度の時間で計算できる．

（C） 高速微分法を使えば，n が大きくなれば大きくなるほど計算時間が短縮できる．

（D） 21 問解いた中で 5 問だけで，精度が改善された．その理由は今のところ不明である．

6． 問題点

・ ROW法

解析的なヤコビアン行列を計算することが，どのような問題に対して有効なのか，わからない．

・高速微分法のプレコンパイラ

文字定数，全ての基本関数，中間変数，ループ，ユーザ定義関数を使えるようにする．エラーメッセージを出力させる．

参考文献

[1] 三井斌友 ： 数値解析入門 －常微分方程式を中心に－ 朝倉書店 東京 (1985)

[2] Kaps, P., Rentrop, P. ： Generalized Runge-Kutta Methods of Order Four with Stepsize Control for Stiff Ordinary Differential Equations. Numer. Math. 33. 55-68 (1979)

[3] Gottwald, B.A., Wanner, G. : A Reliable Rosenbrock Integrator for Stiff Differential Equations.
Computing 26, 355-360 (1981)

[4] Enright, W.H., Hull, T.E., Lindberg, B. : Comparing Numerical Methods for Stiff Systems of O.D.E.
BIT 15, 10-48 (1975)

[5] 森正武（訳） : 計算機のための数値計算法
科学技術出版 (1978)

[6] Iri, M. : Simultaneous Computing of Functions, Partial Derivatives and Estimates of Rounding Errors -Complexity and Practicality-. Japan J.Appl.Math.
1, 223-252 (1984)

[7] 伊理正夫，久保田光一 : 高速自動微分法 －グラフ，丸め誤差，ノルム－ 数理科学 第 25 巻 第 3 号
No.285, 41-48 (1987)

[8] Gottwald, B.A., Wanner, G. : Comparison of Numerical Methods for Stiff Differential Equations in Biology and Chemistry.
SIMULATION 1982 February 61-65

[9] 三井斌友 : 数式処理と数値処理との界面 情報処理
Vol.27 No.4 422-430 (1986)

[10] 佐々政孝 : コンパイラの自動生成 情報処理
Vol.28 No.10 1359-1367 (1987)

[11] 佐々政孝 : コンパイラ生成系 情報処理
Vol.23 No.9 802-817 (1982)

[12] 木村友則，武市正人 : コンパイラ構成法の比較
情報処理学会第27回大会 6E-9 (1983)

[13] 近藤嘉雪，森公一郎　：YACCに挑戦　日経バイト
June　1987　203-212

[14] Kernighan, B., Pike, R.　:　The UNIX Programing Environment.　Prentice-Hall　1984
邦訳　石田晴久：UNIXプログラミング環境　アスキー出版局
1985

[15] 中田育男　:　コンパイラ　産業図書　(1981)

[16] 石田晴久　：UNIXシステム入門14　UNIXのCプログラムジェネレータ　bit　Vol.14　No.13　68-73　(1982)

[17] Kernighan, B., Ritchie, D.　:　The C Programing Language.　Prentice-Hall　(1978)

[18] UTS Support Tool Guide V10L20　FUJITSU Manual (1986)

付録

KapsとRentropの公式 [2][3] の係数

$$\gamma = 0.395$$
$$a_{21} = 0.438$$
$$a_{31} = 0.938948678483428$$
$$a_{32} = 0.0730795420615381$$
$$c_{21} = -1.94347441894707$$
$$c_{31} = 0.416957530989189$$
$$c_{32} = 1.32396782072923$$
$$c_{41} = 1.51951325778448$$
$$c_{42} = 1.35370815030093$$
$$c_{43} = -0.854151495257539$$

$$b_1 = 0.729044879960308$$
$$b_2 = 0.0541069773272405$$
$$b_3 = 0.281599362440017$$
$$b_4 = 0.25$$
$$e_1 = -0.0190858871999474$$
$$e_2 = 0.255608791716455$$
$$e_3 = -0.0863816280897592$$
$$e_4 = 0.25$$