寄　　書

Runge-Kutta-Gill 法について*

伊　理　正　夫**　　松　谷　泰　行***

古典的 Runge-Kutta 法に対して Runge-Kutta-Gill 法のもつ利点の一つに**丸め誤差の自動修正**ということがあげられているが，現在わが国に流布されている数値計算関係の教科書やプログラム例のほとんどはこの点を正しくとらえていない．これに関する問題点を指摘し，Runge-Kutta-Gill 法の利点を正しく実現することのできるプログラム例を示すことが本小論の目的である．

a, η^1, ……, η^m を与えられた定数，f^1, ……, f^m を与えられた（$m+1$ 変数の）函数として

$$\left.\begin{aligned}&\frac{dy^i}{dx}=f^i(x,\ y^1,\ \cdots\cdots,\ y^m),\\&y^i(a)=\eta^i\qquad(i=1,\ \cdots\cdots,\ m)\end{aligned}\right\}\qquad(1)$$

という連立m元常微分方程式の初期値問題を数値的に解く際に，独立変数xの離散的な値（分点）x_n（$n=0,\ 1,\ 2,\ \cdots\cdots$）を

$$a=x_0<x_1<x_2<\cdots\cdots$$

であるように適当に選び，第n分点x_nにおける（未知）函数$y^i(x_n)$（$i=1,\ \cdots\cdots,\ m$）の値を近似する数値解y_n^iを

$$\left.\begin{aligned}&y_0^i=\eta^i,\\&h_n=x_{n+1}-x_n,\\&k_{n,1}^i=h_nf^i(x_n,\ y_n^j),\\&k_{n,2}^i=h_nf^i\left(x_n+\frac{h_n}{2},\ y_n^j+\frac{k_{n,1}^j}{2}\right),\\&k_{n,3}^i=h_nf^i\left(x_n+\frac{h_n}{2},\ y_n^j+\frac{k_{n,2}^j}{2}\right),\\&k_{n,4}^i=h_nf^i(x_n+h_n,\ y_n^j+k_{n,3}^j),\\&y_{n+1}^i=y_n^i+\frac{1}{6}(k_{n,1}^i+2k_{n,2}^i+2k_{n,3}^i+k_{n,4}^i)\end{aligned}\right\}\qquad(2)$$

という式によって $y_0^i(=\eta^i)$, y_1^i, y_2^i, …… と次々に定めて行くというのが **Runge-Kutta 法**として古くからよく知られ用いられて来ている方法である．

普通のやり方で（2）をプログラムすると，必要な作業用記憶装置のうち方程式の元数mと共に増大する分の数は $4m$ になる．これに対し 1951 年に S. Gill は（2）と打切誤差のオーダが等しくしかも上述の記憶装置の数が $3m$ ですむ次の方法を提案した〔1〕．

（数がmに比例する記憶装置としては k^i, y^i, q^i があり，rはただ一つのそれで十分である．）

$$y_0^i=\eta^i,\quad q_{0,1}^i=0,\qquad(3.0)$$

$$h_n=x_{n+1}-x_n,\qquad(3.n)$$

$$k_{n,1}^i=h_nf^i(x_n,\ y_n^j),\qquad(3.n.1.1)$$

$$r_{n,1}^i=\frac{1}{2}k_{n,1}^i-q_{n,1}^i,\qquad(3.n.1.2)$$

$$y_{n,1}^i=y_n^i+r_{n,1}^i,\qquad(3.n.1.3)$$

$$q_{n,2}^i=q_{n,1}^i+3r_{n,1}^i-\frac{1}{2}k_{n,1}^i,\qquad(3.n.1.4)$$

$$k_{n,2}^i=h_nf^i\left(x_n+\frac{h_n}{2},\ y_{n,1}^j\right),\qquad(3.n.2.1)$$

$$r_{n,2}^i=\left(1-\frac{1}{\sqrt{2}}\right)(k_{n,2}^i-q_{n,2}^i),\qquad(3.n.2.2)$$

$$y_{n,2}^i=y_{n,1}^i+r_{n,2}^i,\qquad(3.n.2.3)$$

$$q_{n,3}^i=q_{n,2}^i+3r_{n,2}^i-\left(1-\frac{1}{\sqrt{2}}\right)k_{n,2}^i,\qquad(3.n.2.4)$$

$$k_{n,3}^i=h_nf^i\left(x_n+\frac{h_n}{2},\ y_{n,2}^j\right),\qquad(3.n.3.1)$$

$$r_{n,3}^i=\left(1+\frac{1}{\sqrt{2}}\right)(k_{n,3}^i-q_{n,3}^i)\qquad(3.n.3.2)$$

$$y_{n,3}^i=y_{n,2}^i+r_{n,3}^i,\qquad(3.n.3.3)$$

$$q_{n,4}^i=q_{n,3}^i+3r_{n,3}^i-\left(1+\frac{1}{\sqrt{2}}\right)k_{n,3}^i,\qquad(3.n.3.4)$$

$$k_{n,4}^i=h_nf^i(x_n+h_n,\ y_{n,3}^j),\qquad(3.n.4.1)$$

$$r_{n,4}^i=\frac{1}{6}(k_{n,4}^i-2q_{n,4}^i),\qquad(3.n.4.2)$$

$$y_{n+1}^i=y_{n,3}^i+r_{n,4}^i,\qquad(3.n.4.3)$$

$$q_{n+1,1}^i=q_{n,4}^i+3r_{n,4}^i-\frac{1}{2}k_{n,4}^i.\qquad(3.n.4.4)$$

上の（3）式に含まれている Gill の工夫は，記憶装置の数の減少という点だけでなく，（3）によると**丸め誤差の累積が自動的に避けられる**ようになっている．（2）の改良版としての（3）は **Runge-Kutta-Gill 法**と呼ばれ，わが国では大抵の数値計算法の書物に紹介されているほど有名である．（〔3〕～〔11〕など

* A remark on the floating-point arithmetic programmes for the Runge-Kutta-Gill method

** 東京大学工学部計数工学科
*** 八幡製鉄株式会社経理部機械計算課

に皆紹介されている.)

さて，Gill が（3）式を提案した頃は，電子計算機といっても高速記憶装置が全部で数百語といった程度，しかも金物でできる演算は固定小数点というようなものがはばをきかしていた．そこで記憶装置の必要数に神経質になり，また，丸めにも重大な関心が払われた．（3）式の場合，r, k, q は一般に h の程度の大きさの小さな数，それに対して x, y は1程度の大きな数であるので，(3.n.l.1), (3.n.l.2), (3.n.l.4) の計算は，r, k, q の有効桁数をなるべく多くとるようにして行ない，(3.n.l.3) で r を y に加えるときに始めて r を y の桁と揃えてやるとよい（l=1, 2, 3, 4)．このとき r の下の方の桁が丸められる．

しかし，数万語の高速記憶装置をもち，浮動小数点演算の金物とプログラミング言語の完備した機械が常識となった今日，記憶装置の必要数は大して問題ではなく，また，大きさの程度の異る数の間の加減算のための桁揃えの問題も，ALGOL や FORTRAN でプログラムを書き，r, k, q, y, x などをすべて“real”と宣言しておくだけで，万事自動的に解決されてしまう．だから，（3）をそのまま ALGOL や FORTRAN で書けば立派な Runge-Kutta-Gill 法のプログラムが完成するように思える．実際，そのようなプログラムの例が〔6〕,〔10〕,〔11〕などにある（一例として〔11〕の295ページのプログラムを**第1図**に掲げる.).

```
C   MAIN PROGRAM
      COMMON Y(10), F(10), M
      DIMENSION Q(11)
      REAL K
      READ (5, 102) H, NN
  102 FORMAT (F12. 0, I8)
      WRITE (6, 103) H
  103 FORMAT (1H0, 2HH=E15. 7//),
      READ (5, 104) M, (Y(I), I=1, M)
  104 FORMAT (I8/(6F12. 0))
      DO 10 I=1, M
   10 Q (I)=0.0
      F (1)=1.0
      CALL UHEN
      CALL INSATU
C   SINKO
      DO 20 N=1, NN
      DO 30 I=1, M
      K=H*F(I)
      QI=Q(I)
      R=0.5*K-QI
      Q(I)=QI+3.0*R-0.5*K
      Y(I)=Y(I)+R
   30 CONTINUE
      CALL UHEN
      DO 40 I=1, M
      K=H*F(I)
      QI=Q(I)
      R=0.2928932*(K-QI)
      Q(I)=QI+3.0*R-0.2928932*K
      Y(I)=Y(I)+R
   40 CONTINUE
      CALL UHEN
      DO 50 I=1, M
      K=H*F(I)
      QI=Q(I)
      R=1.707107*(K-QI)
      Q(I)=QI+3.0*R-1.707107*K
      Y(I)=Y(I)+R
   50 CONTINUE
      CALL UHEN
      DO 60 I=1, M
      K=H*F(I)
      QI=Q(I)
      R=(K-2.0*QI)/6.0
      Q(I)=QI+3.0*R-0.5*K
      Y(I)=Y(I)+R
   60 CONTINUE
      CALL UHEN
      CALL INSATU
   20 CONTINUE
      STOP
      END
```

第1図 Runge-Kutta-Gill 法のプログラム（流布版）

上掲の文献のうち〔3〕,〔5〕,〔7〕,〔8〕,〔9〕にも――プログラム例は明示されていないものの――上に述べたような考え方以外の可能性は示されていない．固定小数点演算の場合について論じている部分のある〔4〕にも上のような考え方を示唆するものがあり，特に〔4〕では浮動小数点演算の場合のプログラ

ム例として〔6〕のそれをあげている.

しかし，本当に（3）式を無心に計算していけば丸め誤差の累積が自動的に避けられるであろうか．実は，そうではなくて，（3）式には隠されたトリックがある(そのトリックの理論的根拠については Gill の原論文〔1〕を参照して頂くことにして省略する.). それは，(3.n.l.3) 式を計算するとき右シフトで下の桁を丸められた r を y に加えた後，(3.n.l.4) で再び用いる r が，もとの r ではなく下の桁を丸められた後の r である——必要なら q, k などと桁を揃えるため再び左シフトする——ということである．つまり，(3.n.l.4) で用いられる r は (3.n.l.3) 式を計算するとき生じた丸め誤差を一緒に背負いこんでいるものでなければならない．また，(3.n.l.3) と (3.n.l.4) とは（第1図では実際そうしているように）一見順序を逆にしていいようにもみえるが，上のような仕掛けが含まれているので両者の順序は重要である.

上のようなトリックを，浮動小数点演算で実行することを工夫してみよう．それには (3.n.l.3) と (3.n.l.4) (l=1, 2, 3, 4) の代わりに

$$s := y, \tag{4.1}$$

$$y := y + r, \tag{4.2}$$

$$r' := y - s, \tag{4.3}$$

$$q := q + 3r' - c_l k \tag{4.4}$$

```
C   MAIN PROGRAM
      COMMON Y(10), F(10), M
      DIMENSION Q(11)
      REAL K
      READ (5, 102) H, NN
  102 FORMAT (F12.0, I8)
      WRITE (6, 103) H
  103 FORMAT (1H0, 2HH=E15.7//)
      READ (5, 104) M, (Y(I), I=1, M)
  104 FORMAT (I8/(6F12.0))
      DO 10 I=1, M
   10 Q(I)=0.0
      F(1)=1.0
      CALL UHEN
      CALL INSATU
C   SINKO
      DO 20 N=1, NN
      DO 30 I=1, M
      K=H*F(I)
      QI=Q(I)
      R=0.5*K-QI
      S=Y(I)
      Y(I)=S+R
      R=Y(I)-S
      Q(I)=QI+3.0*R-0.5*K
   30 CONTINUE
      CALL UHEN
      DO 40 I=1, M
      K=H*F(I)
      QI=Q(I)
      R=0.2928932*(K-QI)
      S=Y(I)
      Y(I)=S+R
      R=Y(I)-S
      Q(I)=QI+3.0*R-0.2928932*K
   40 CONTINUE
      CALL UHEN
      DO 50 I=1, M
      K=H*F(I)
      QI=Q(I)
      R=1.707107*(K-QI)
      S=Y(I)
      Y(I)=S+R
      R=Y(I)-S
      Q(I)=QI+3.0*R-1.707107*K
   50 CONTINUE
      CALL UHEN
      DO 60 I=1, M
      K=H*F(I)
      QI=Q(I)
      R=(K-2.0*QI)/6.0
      S=Y(I)
      Y(I)=S+R
      R=Y(I)-S
      Q(I)=QI+3.0*R-0.5*K
   60 CONTINUE
      CALL UHEN
      CALL INSATU
   20 CONTINUE
      STOP
      END
```

第2図 Runge-Kutta-Gill 法のプログラム(真正版)

のような形の一群の式を使ってやればよいことは容易に理解されよう．(4.2) で生じた丸めの分だけの値の変化をうけた r が (4.3) で作られ，それが (4.4) で使われることになるからである．そのような変更を第1図のプログラムに加えると**第2図**のようになる．

この変更により作業用の場所は s 一つだけしか増えていない．以上の注意と実質的に同内容の注意が，実は，〔2〕において既になされているが，その後何故かこのことは忘れられがちであったようにみえる．〔2〕では上の注意を印象深く裏づけるような数値実験が十分でなかったからかもしれないが．

机上の議論はこれまでとして，第1図と第2図のプログラムの相異点を検証する計算を実際に行なってみた結果を示そう．計算には東京大学大型計算機センターの HITAC 5020E を FORTRAN Ⅳ で用いた．すなわち，すべての浮動小数点演算は有効数字2進23ビット（10進換算約7桁）で行なわれている．解いた問題は

$$\frac{dy}{dx}=1,\quad y(0)=1 \qquad (5)$$

および

$$\frac{dy}{dx}=1.00001,\quad y(0)=1 \qquad (6)$$

の2題で，いずれも刻み幅を $h_n=0.001$ に固定した（すなわち $x_n=0.001\times n$）．結果の一部を**第1表**に示す．

第 1 表

	第1図のプログラムによる		第2図のプログラムによる	
n	(5) の解	(6) の解	(5) の解	(6) の解
0	1.000 000 0	1.000 000 0	1.000 000 0	1.000 000 0
100	1.099 992 8	1.099 992 8	1.099 999 9	1.100 001 1
200	1.199 985 5	1.199 985 5	1.200 000 0	1.200 002 0
300	1.299 978 3	1.299 978 3	1.300 003 0	1.300 003 1
400	1.399 971 0	1.399 971 0	1.400 000 1	1.400 003 9
500	1.499 963 8	1.499 963 8	1.500 000 0	1.500 005 0
600	1.599 956 5	1.599 956 5	1.599 999 9	1.600 006 1
700	1.699 949 3	1.699 949 3	1.700 000 0	1.700 007 0
800	1.799 942 0	1.799 942 0	1.800 000 0	1.800 008 1

丸め誤差がなければ，いずれも，それぞれ正確な解

$$y=1+x=1+0.001n, \qquad (5')$$

$$y=1+1.00001x=1+0.00100001n \qquad (6')$$

を与えてくれるはずである．しかし，実際には第1図のプログラムで解いた結果では丸め誤差の影響が相当ひどく認められるし，特に (5') に相当する値と (6') に相当する値との間に全然差が認められない．これに対して，第2図のプログラムによる結果はほとんど7桁一杯正しい値が得られている．僅かのプログラムの変更ではあるが差は歴然たるものである．

なお，第1図，第2図のプログラムで呼んでいるサブプログラム UHEN および INSATU の内容は**第3図**のとおりである（〔11〕の 295 ページのそれに倣っている）．

```
      SUBROUTINE UHEN
      COMMON Y(10), F(10)
      F(2)=1.0
      F(3)=1.00001
      RETURN
      END

      SUBROUTINE INSATU
C     PRINTS SOLUTION AND RIGHT-HAND
      COMMON Y(10), F(10), M
      WRITE(6, 101) Y(1), (Y(I), F(I), I=2, M)
  101 FORMAT (1H, 2HX=E16.8, 5(5X, 2E16.8))
      RETURN
      END
```

第 3 図

もっとも，上記の数値実験は実験のための実験で，第1図と第2図のプログラムの相異を極端に強調するように計画されているので，普通の問題を普通に解くときには一般にはこれほどの差は現われないであろうと思われるから，(3) をそのまま ALGOL や FORTRAN で書いたプログラムを持っておられる方々および使っておられる方々にとって，本論の注意は実際問題としてあまり影響ないかも知れない．しかし，それにしても，「Runge-Kutta-Gill 法には丸め誤差に対する配慮がなされている」という説明と第1図と同内容のプログラム例（あるいはそれを示唆するような (3) 式に類似の式）とが共存しているのは好ましくない．

数値計算法は近年急速に進歩しつつある分野であり，その若さの故にまだ吟味不十分のまま残されている箇所も少なくないかと思われるので（たとえば〔12〕なども参照されたい），数値計算の理論については多くの人々が機会あるごとに反省してみる必要があるのではなかろうか．

因みに，Gill の工夫――すなわち，方程式の元数 m

に比例して必要になる記憶装置の数を $3m$ に抑えること，および，丸め誤差の累積を自動的に避けること——は，任意のオーダ4の Runge-Kutta 的方法に対して（もちろん（2）も含めて）実行可能であることを示した論文が発表されている〔13〕．これらの場合にも本論における注意が適用されることは勿論である．

なお，本論における数値実験にさいしては東京大学工学部計数工学科小林光夫君および本多房子嬢の御協力を得たことを記して謝意を表わす次第です．

参考文献

〔1〕 S. Gill, A Process for the Step-by-Step Integration of Differential Equations in an Automatic Digital Computing Machine. Proceedings of the Cambridge Philosophical Society, Vol. 47 (1951), pp. 96-108.

〔2〕 松谷泰行，TAC における常微分方程式数値解法．東京大学大学院数物系研究科応用物理専門課程提出修士論文，1959年2月．

〔3〕 高田勝，微分方程式の数値解法．第2回 CP セミナーテキスト，1960年11月，日本科学技術連盟．

〔4〕 吉田恪郎，常微分方程式 — Runge-Kutta-Gill 法．第4回 CP セミナーテキスト，1962年12月，日本科学技術連盟．

〔5〕 伊理正夫，常微分方程式（1），（2），第6回 CP セミナーテキスト，1964年7月，日本科学技術連盟．

〔6〕 森口繁一（編），ALGOL 入門．日本科学技術連盟，昭和37年．

〔7〕 山内二郎，森口繁一，一松 信（編），電子計算機のための数値計算法 I．培風館，昭和40年．

〔8〕 宇野利雄，計算機のための数値計算．朝倉書店，昭和38年．

〔9〕 一松 信，数値計算．至文堂，昭和38年．

〔10〕 杉山昌平，高橋磐郎，数値解析．広川書店，昭和40年．

〔11〕 森口繁一，FORTRAN IV 入門．東京大学出版会，1965年．

〔12〕 伊理正夫，予測子・修正子法により常微分方程式の初期値問題を解く際の局所打切誤差の評価をするためのいわゆる Milne の方法について，昭和39年度情報処理学会全国大会講演予稿集，59頁～60頁．

〔13〕 D. J. Fyfe, Economical Evaluation of Runge-Kutta Formulae. Mathematics of Computation, Vol. 20 (1966), pp. 392-398.

（昭和41年11月17日受付）