Quick BASIC によるプログラミングについて

吉村　卓

On Programming in Quick BASIC

Takashi YOSHIMURA

（平成2年10月15日受理）

1．はしがき

従来の BASIC の大きな欠点は構造化プログラミングが行ないにくいことである。たとえば，ブロック構造がない，流れ制御構造が完備されていない，データ構造が豊富でない，モジュール化ができないなどの点である。

しかし，BASIC の持つ使いやすさ，親しみやすさなどの良い環境を残しながら BASIC に構造化という概念を導入すれば，そこに新しい BASIC の可能性が開ける。このような観点から，Quick BASIC という処理系が Microsoft 社から発表された。

以下に，具体的に数学の問題を例にして，いかに分かりやすいプログラムが書けるか試みてみよう。

2．手続きと再帰呼出し

従来，GOSUB で呼出すサブルーチンはメインルーチンに従属するものとして構成され，サブルーチン間のデータ授受は共通変数を介して行なうため，各ルーチン間の独立性が保てないという欠点を持っていた。Quick BASIC では従来型 BASIC の欠点を改良し，プロシージャ（手続き）という概念を導入した。手続には，SUB～END SUB（サブルーチンプロシージャ）と FUNCTION～END FUNCTION（関数プロシージャ）の2種類ある。

また，Quick BASIC では再帰呼出（recirsive call）が可能となった。これについては，従来の BASICでも不十分ではあるが書くことは書けた。というのは，GOSUB 文が現われると戻り先アドレスをシステムスタックに積んでサブルーチンへ飛ぶので，サブルーチンでの処理が済めば，呼出し元に正しく戻れるからである。FORTRAN では FORTRAN 77 でも再帰呼出しが許されないが，何故こんな簡単なことが FORTRAN ではできないのか，筆者には分らない。

ところで，従来型の BASIC のサブルーチンでは，引数という概念を持たないため，サブルーチン内で使用している変数にメイン側でデータを代入してから，サブルーチンをコールするという煩雑な処理になってしまう。

図1－1は，フラクタル図形として代表的なコッホ曲線を描かせるプログラムを従来の BASIC で書いたものである。サブルーチンをコールする前に変数 ORD の現時点の内容をユーザスタックに積んでから，サブルーチンを呼出している。

```
100 ' save "KOCH
300 *RECURSE
310 IF ORD=0 THEN L=SIZE:GOSUB *MOVE:RETURN
320 GOSUB *PUSH:ORD=ORD-1:GOSUB *RECURSE
325 GOSUB *POP:TH=60: GOSUB *LEFT
330 GOSUB *PUSH:ORD=ORD-1:GOSUB *RECURSE
335 GOSUB *POP:TH=120:GOSUB *RIGHT
340 GOSUB *PUSH:ORD=ORD-1:GOSUB *RECURSE
345 GOSUB *POP:TH=60: GOSUB *LEFT
350 GOSUB *PUSH:ORD=ORD-1:GOSUB *RECURSE
355 GOSUB *POP
360 RETURN
400 *MOVE
420 X=X+L*COS(THETA*PI/180)
430 Y=Y-L*SIN(THETA*PI/180)
440 LINE -(X,Y),COL
450 RETURN
500 *LEFT
510 THETA=THETA+TH
520 RETURN
550 *RIGHT
560 THETA=THETA-TH
570 RETURN
600 *PUSH
610 STAK(SP)=ORD:SP=SP+1
620 RETURN
650 *POP
660 SP=SP-1:ORD=STAK(SP)
670 RETURN
```

図　1－1

図1－2は，同じ問題を Quick BASIC でプログラミングしたものである。Quick BASIC では，プロシージャがプログラムとして独立しているだけでなく，変数もまたプロシージャ間で独立して使用できるので，他のプロシージャで同じ変数名を使用しても，他のプロシージャには全く影響を及ばさない。そのため，図1－2は図1－1にくらべ，すっきりとしたプログラムになっている。

```
SUB KOCH (N)
IF N = 0 THEN CALL MOVE: EXIT SUB
CALL KOCH(N - 1)
CALL LEFT(60): CALL KOCH(N - 1)
CALL RIGHT(120): CALL KOCH(N - 1)
CALL LEFT(60): CALL KOCH(N - 1)
END SUB

SUB MOVE
SHARED X, Y, LENG, ANGLE, RAD
X = X + LENG * COS(RAD * ANGLE)
Y = Y - LENG * SIN(RAD * ANGLE)
LINE -(X, Y)
END SUB

SUB LEFT (TH)
SHARED ANGLE
ANGLE = ANGLE + TH
END SUB

SUB RIGHT (TH)
SHARED ANGLE
ANGLE = ANGLE - TH
END SUB
```

図　1−2

図２は図２－１に示す一筆描きの曲線を描かせるプログラムである。図２－２，図２－３はそれぞれ，従来型 BASIC, Quick BASIC で書いたものである。図１のプログラムよりもさらにQuick BASIC によるプログラミングのし易すさが実感できる。

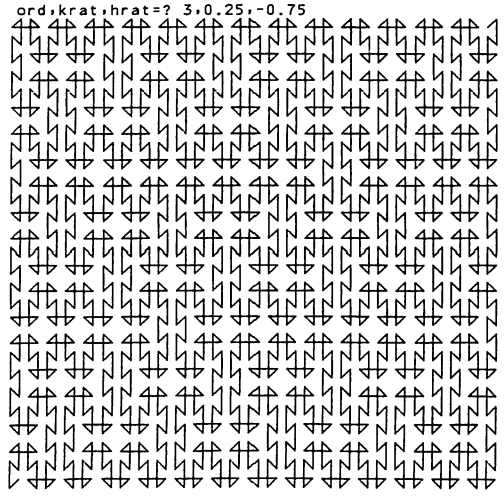

図　2−1

```
100 ' save "RIEUL
300 *RECURSE
310 IF ORD=0 THEN MAG=KRAT:C=A:D=B:GOSUB *MOVE:RETURN
320 GOSUB *PUSH:C=0:D= B          :
325 GOSUB *PROC:GOSUB *POP
330 GOSUB *PUSH:C=0:D= B:A=-A     :
335 GOSUB *PROC:GOSUB *POP
340 GOSUB *PUSH:C=A:D= 0:         :
345 GOSUB *PROC:GOSUB *POP
350 GOSUB *PUSH:C=0:D=-B:     B=-B:
355 GOSUB *PROC:GOSUB *POP
360 GOSUB *PUSH:C=0:D=-B:A=-A:B=-B:
365 GOSUB *PROC:GOSUB *POP
370 GOSUB *PUSH:C=A:D= 0:    :B=-B:
375 GOSUB *PROC:GOSUB *POP
380 GOSUB *PUSH:C=0:D= B          :
385 GOSUB *PROC:GOSUB *POP
390 GOSUB *PUSH:C=0:D= B:A=-A     :
395 GOSUB *PROC:GOSUB *POP
400 GOSUB *PUSH:ORD=ORD-1:GOSUB *RECURSE
405 GOSUB *POP:RETURN
410 *MOVE
420 X=X+MAG*C
430 Y=Y-MAG*D
440 LINE -(X,Y),COL
450 RETURN
500 *PROC
510 GOSUB *PUSH:ORD=ORD-1:GOSUB *RECURSE
515 GOSUB *POP
520 GOSUB *PUSH:MAG=HRAT:GOSUB *MOVE
525 GOSUB *POP
540 RETURN
600 *PUSH
610 STAK(SP)=ORD:SP=SP+1:STAK(SP)=MAG:SP=SP+1
615 STAK(SP)=A:SP=SP+1:STAK(SP)=B:SP=SP+1
620 STAK(SP)=C:SP=SP+1:STAK(SP)=D:SP=SP+1
630 RETURN
650 *POP
660 SP=SP-1:D=STAK(SP):SP=SP-1:C=STAK(SP)
670 SP=SP-1:B=STAK(SP):SP=SP-1:A=STAK(SP)
675 SP=SP-1:MAG=STAK(SP):SP=SP-1:ORD=STAK(SP)
680 RETURN
```

図　2−2

```
SUB RIEUL (ORD%, A%, B%)
SHARED KRAT
IF ORD% = 0 THEN
  CALL MOVE(0, A%, B%, KRAT)
  EXIT SUB
END IF
CALL PROC(ORD%, A%, B%, 0, B%)
CALL PROC(ORD%, -A%, B%, 0, B%)
CALL PROC(ORD%, A%, B%, A%, 0)
CALL PROC(ORD%, A%, -B%, 0, -B%)
CALL PROC(ORD%, -A%, -B%, 0, -B%)
CALL PROC(ORD%, A%, -B%, A%, 0)
CALL PROC(ORD%, A%, B%, 0, B%)
CALL PROC(ORD%, -A%, B%, 0, B%)
CALL RIEUL(ORD% - 1, A%, B%)
END SUB

SUB MOVE (ORD%, A%, B%, MAG#)
SHARED COL%, X, Y
X = X + MAG * A%: Y = Y - MAG * B%
LINE -(X, Y), COL%
END SUB

SUB PROC (ORD%, A%, B%, C%, D%)
SHARED HRAT
CALL RIEUL(ORD% - 1, A%, B%)
CALL MOVE(ORD%, C%, D%, HRAT)
END SUB
```

図　2−3

正方整行列の行列式（明らかに整数値である）を計算するには，展開定理を用いる方法と，与えられた行列を上三角化する方法が考えられる。三角行列の行列式は対角要素の積として直ちに求まる。この第２の方法によるプログラムは後で考えることにして，ここでは，展開定理を用いる方法について考える。これによるプログラムは，明らかに再帰呼出しを用いることになる。図３に示すプログラムがそれである。

$$|A| = \sum_{j=1}^{n} (-1)^{i+j} a_{ij} |A_{ij}|$$

ここに，A_{ij} は行列Ａの第ｉ行と第ｊ列をとり除いて得られるｎ－１次の行列である。プログラムはこの定理をそのままコーディングして得られる。

行列式の値を算出できれば，整数係数の連立一次方程式 $Ax = b$ の解はクラメールの公式

$$x_j = \frac{1}{|A|}(A^1, \cdots A^{j-1}, b, A^{j+1}, \cdots A^n)$$

（ここに，A^j はAの第j列ベクトルである）
を用いて容易に求められる。

```
FUNCTION DET (N%, A())
DEFINT I-N
DIM C(N - 1, N - 1)
IF N = 1 THEN DET = A(1, 1): EXIT FUNCTION
S = 0
FOR K = 1 TO N
  FOR I = 2 TO N
    FOR J = 1 TO K - 1
      C(I - 1, J) = A(I, J)
    NEXT J
    FOR J = K + 1 TO N
      C(I - 1, J - 1) = A(I, J)
    NEXT J
  NEXT I
  D = DET(N - 1, C())
  IF K MOD 2 THEN
    S = S + A(1, K) * D
    ELSE S = S - A(1, K) * D
  END IF
NEXT K
DET = S
END FUNCTION
```

図 3

```
SUB CRAMER (N%, A(), B(), X())
DEFINT I-N
DIM C(N, N)
BUNBO = DET(N, A())
IF BUNBO = 0 THEN PRINT "det(A)=0": EXIT SUB
FOR K = 1 TO N
  FOR I = 1 TO N
    FOR J = 1 TO N: C(I, J) = A(I, J): NEXT J
  NEXT I
  FOR I = 1 TO N: C(I, K) = B(I): NEXT I
  D = DET(N, C())
  X(K) = D / BUNBO
NEXT K
END SUB
```

図 4

図4がそのプログラムである。プログラム中 X(K)=D/BUNBO の割算を実行せず，分子Dと分母 BUNBO をそれぞれ出力するならば，整係数連立方程式の解を有理数の形で得ることができる。解を実数形式でなく有理数の形で求めようとするのは，浮動小数計算に伴う誤差をなくすためである。

3．プログラムの制御構造

プログラムの構造化のための制御構造は，順次処理，選択，繰返しの3つの要素で構成される。すべてのプログラムは，この3つの制御構造を使って作成することができる。

Quick BASIC では，行番号を使わないが，プログラムは上から下へと書かれた順番に実行されるので，行番号がなくても差支えない。

選択は，イエスかノーの2方向分岐の場合と，条件によっていくつかあるケースに分かれる多方向分岐の場合とがある。従来の BASIC には，前者の構文として IF～THEN～ELSE～が,後者には，ON～GOTOやON GOSUB があったが，Quick BASIC ではさらに，ブロックIF文 IF～THEN～END IF や複数条件判断の SELECT CASE～END SELECT が追加された（FORTRAN77 でもブロックIF文は書ける)。いままでのBASICが1行しか使えないIF文のために，GOTO 文を使わざるを得ず，そのために分りにくいプログラムになっていたのに比べ，ブロックIF文の使用により，きわめて読みやすいプログラムができるようになった。

繰返し構造としては，従来型 BASIC には所定回数反復の FOR～NEXT ならびに，条件判定反復のWHILE～WEND があったが，Quick BASIC にはさらに，前判定反復の DO WHILE～LOOP, DO UNTIL～LOOP や，後判定反復の DO～LOOP WHILE ならびに DO～LOOP UNTIL が加わって，スマートで読み易く，理解しやすいプログラムを書けるようになった。

```
' GAUS'S INTEGRAL
DECLARE FUNCTION F! (X!)
DEFINT I-N
INPUT "積分区間 [a,b]   a, b  ="; A, B
INPUT "標本点の個数 n(2<=n<=7)="; N
DIM X(1 TO N), W(1 TO N)
SELECT CASE N
CASE 2
  X(1) = -.5773503: X(2) = -X(1)
  W(1) = 1: W(2) = 1
CASE 3
  X(1) = -.7745967: X(2) = 0: X(3) = -X(1)
  W(1) = .5555556: W(2) = .8888889: W(3) = W(1)
CASE 4
  X(1) = -.8611363: X(2) = -.339981
  X(3) = -X(2): X(4) = -X(1)
  W(1) = .3478548: W(2) = .6521452
  W(3) = W(2): W(4) = W(1)
CASE 5
  X(1) = -.9061798: X(2) = -.5384693: X(3) = 0
  X(4) = -X(2): X(5) = -X(1)
  W(1) = .2369269: W(2) = .4786287: W(3) = .5688889
  W(4) = W(2): W(5) = W(1)
CASE 6
  X(1) = -.9324695: X(2) = -.6612094: X(3) = -.2386192
  X(4) = -X(3): X(5) = -X(2): X(6) = -X(1)
  W(1) = .1713245: W(2) = .3607616: W(3) = .4679139
  W(4) = W(3): W(5) = W(2): W(6) = W(1)
CASE 7
  X(1) = -.9491079: X(2) = -.7415312: X(3) = -.4058452
  X(4) = 0: X(5) = -X(3): X(6) = -X(2): X(7) = -X(1)
  W(1) = .1294849: W(2) = .2797054: W(3) = .3818301
  W(4) = .4179592: W(5) = W(3): W(6) = W(2): W(7) = W(1)
END SELECT
Z = 0
FOR I = 1 TO N
  Y = ((A + B) + (B - A) * X(I)) / 2
  Z = Z + W(I) * F(Y)
NEXT I
Z = .5 * (B - A) * Z
PRINT Z
END
```

図 5

図5はガウスの積分公式をプログラムしたものである。ガウスの数値積分公式とは，定積分

$$\int_a^b f(x)\,dx \quad を \quad \sum_{i=1}^{n} f(x_i)\,w_i$$

の形で近似するものである。ここにx_iはn次の直交多項式の零点であり，w_iはクリストッフェル数と呼ばれる標本点x_iにおける重みである。図5は標本点数2～7の場合のプログラムである。SELECT CASE 文を用いるとこのように，GOTO 文なしのプログラムが書ける。

さて，先に Quick BASIC では，プロシージャ間のデータ授受に引数を用いることができると述べたが，Quick BASIC の引数渡しでは参照による呼出し（call by reference あるいは call by address）と，値による呼出し（call by value）という2つの方法が使える。

Quick BASIC のプロシージャ間の引数渡しは原則的には参照による呼出しで行われる。すなわち，実引数のアドレスが仮引数に渡され，呼ばれた側ではこのアドレスをもとに，呼出し側の実引数を参照する。したがって，呼ばれた側で引数の値を変更すると，呼出し側の引数の値も変ってしまう。つまり，仮引数と実引数は連動しているのである。

呼出し側の実引数の値を変えられないようにするには，実引数を式として与えれば，値による呼出しと同じ機能を実現できる。それには，実引数に括弧（ ）をつければよい。実引数を式にすることで，その値が一時的なアドレスに記憶され，そのアドレスが仮引数に渡されることになる。

なお，配列データ全体をプロシージャに渡すには，引数として配列名の後に添え字のない（ ）を付ける。これは実引数の先頭アドレスを渡しているのであって，呼出された側では渡された配列の先頭アドレスしか分っていないのである。

2つの整数 a,b の最大公約数を求めるユークリッドの互除法のプログラムは，図6－1のようにするのが最もアルゴリズムに忠実な書き方であろう。このプログラムでは，メインルーチンで入力したデータa,bを関数プロシージャ EUCLID から戻ってきたとき計算結果の GCD とともに書き出すために，値による呼出しを使っている。

```
DECLARE FUNCTION EUCLID% (W1%, W2%)
INPUT "a,b"; A%, B%
GCD% = EUCLID%((A%), (B%))
PRINT "("; A%; ","; B%; ")="; GCD%
END

FUNCTION EUCLID% (W1%, W2%)
DO
  W3% = W1% MOD W2%
  IF W3% = 0 THEN EXIT DO
  W1% = W2%: W2% = W3%
LOOP
EUCLID% = W2%
END FUNCTION
```

図 6－1

このプログラムでは，操返しに DO～LOOP を用いているが，WHILE ～ WEND を使って図6－2のようにプログラムしてもよいであろう。

```
FUNCTION EUCLID% (W1%, W2%)
WHILE W2% <> 0
  W3% = W1% MOD W2%
  SWAP W1%, W2%
WEND
EUCLID% = W1%
END FUNCTION
```

図 6－2

また，3つ以上の0でない整数$a_1, a_2, \cdots, a_n$の最大公約数を求めるには，初めの2つの最大公約数を$d_2 = GCD(a_1, a_2)$とし，以下すでに求めたものと次の数とのGCDを順に

$d_j = GCD(d_{j-1}, a_j)$

とすると，$d_n = GCD(a_1, a_2, \cdots, a_n)$となる。これをプログラミングしたのが図7である。

```
SUB GCD (N%, ZC() AS LONG, ZG&)
DEFINT I-N: DEFLNG Z
FOR I = 1 TO N
  ZG = ABS(ZC(I))
  IF ZG <> 0 THEN EXIT FOR
NEXT I
IF ZG = 0 THEN EXIT SUB
FOR J = I + 1 TO N
  IF ZC(J) <> 0 THEN
    ZA = ZG: ZB = ZC(J)
    ZG = EUCLID(ZA, ZB)
  END IF
NEXT J
END SUB

FUNCTION EUCLID (ZA&, ZB&)
DEFINT I-N: DEFLNG Z
WHILE ZB <> 0
  ZA = ZA MOD ZB
  SWAP ZA, ZB
WEND
EUCLID = ABS(ZA)
END FUNCTION
```

図 7

方程式 f(x)＝0 の近似解を求めるのに，f（a）・f（b）＜0なる2点a，bからスタートし，

（1）a，bの中点c＝（a＋b）／2を計算する。f（c）を求め，

（2）f（a）・f（c）＜0ならば解はa，cの間にあるのでb←cとし，f（b）・f（c）＜0なら解はc，bの間にあるのでa←cとして（1）に戻る。｜b－a｜＜εとなるまで（1），（2）を繰返す。これを2分法という。このとき，cが解の近似値であるが，その誤差は｜b－a｜／2内である。

図8－1は後判定ループを用いたプログラムであるが，サブルーチン NIBUN は図8－2，図8－3，図8－4のように書くこともできる。

```
SUB NIBUN (A, B, C)
FA = F(A)
DO
  C = (A + B) * .5: FC = F(C)
  IF FC = 0 THEN EXIT DO
  PRINT "  c="; C
  IF FA * FC > 0 THEN A = C
  IF FA * FC < 0 THEN B = C
LOOP UNTIL B - A < EPS
C = (A + B) * .5
END SUB

FUNCTION F (X)
F = ((X + 4) * X + 3) * X - 2
END FUNCTION
```

図 8−1

```
SUB NIBUN (A, B, C)
FA = F(A)
DO
  C = (A + B) * .5: FC = F(C)
  IF FC = 0 THEN EXIT DO
  PRINT "  c="; C
  IF FA * FC > 0 THEN A = C
  IF FA * FC < 0 THEN B = C
LOOP WHILE B - A > EPS
C = (A + B) * .5
END SUB
```

図 8−2

```
SUB NIBUN (A, B, C)
FA = F(A)
DO WHILE B - A > EPS
  C = (A + B) * .5: FC = F(C)
  IF FC = 0 THEN EXIT DO
  PRINT "  c="; C
  IF FA * FC > 0 THEN A = C
  IF FA * FC < 0 THEN B = C
LOOP
C = (A + B) * .5
END SUB
```

図 8−3

```
SUB NIBUN (A, B, C)
FA = F(A)
DO UNTIL B - A < EPS
  C = (A + B) * .5: FC = F(C)
  IF FC = 0 THEN EXIT DO
  PRINT "  c="; C
  IF FA * FC > 0 THEN A = C
  IF FA * FC < 0 THEN B = C
LOOP
C = (A + B) * .5
END SUB
```

図 8−4

4．変数の通用範囲

方程式 f（x）＝0 を解くのに，初期近似値 x_0 からスタートし，漸化式

$x_{k+1}=x_k-f(x_k)/f'(x_k)$

により逐次解を計算し，$|x_{k+1}-x_k|<\varepsilon$ となったら反復を止める。これをニュートン法という。f が下に凸のとき，大きめの近似値からスタートすると，逐次解は単調減少的に真値に近ずき，f が上に凸のとき，小さめの近似値からスタートすると単調増加的に真値に近ずく。それ故，ニュートン法は 2 分法に比し収束は速いが，単調収束なので誤差評価ができない。この欠点を解決するのには，解を挟んで x_k とは反対側から解に近ずく別の近似解の列 y_k を計算してやればよい。その方法として，はさみ打ち法を用いるとすれば，その漸化式は

$y_{k+1}=\{x_k f(y_k)-y_k f(x_k)\}/\{f(y_k)-f(x_k)\}$

となる。$|y_k-x_k|<\varepsilon$ となったら反復を止める。このとき $(x_k+y_k)/2$ を解の近似値とすると誤差は $|y_k-x_k|/2$ 内である。この方法はダンデリン法と呼ばれる。

```
' DANDELIN
DIM SHARED N
CLS
INPUT "多項式の次数 n="; N
DIM SHARED A(N), B(N), C(N - 1)
FOR I = 0 TO N
  PRINT "a("; I; ")="; : INPUT A(I)
NEXT I
INPUT "初期近似値 x = "; X
INPUT "初期近似値 y = "; Y
INPUT "      epsilon = "; EPS
PRINT "出発値"; "  x="; X, " y="; Y
PRINT "中間値"
WHILE ABS(X - Y) > EPS
  CALL HORNER(X): BN = B(N): CN1 = C(N - 1)
  X1 = X - BN / CN1
  CALL HORNER(Y)
  Y1 = (X * B(N) - Y * BN) / (B(N) - BN)
  PRINT "  x="; X1, "  y="; Y1
  X = X1: Y = Y1
WEND
PRINT "結果"
PRINT "root ="; (X1 + Y1) * .5
PRINT "error="; ABS(X1 - Y1) * .5
PRINT "eps  ="; EPS
END

SUB HORNER (X) STATIC
SHARED N, A(), B(), C()
B(0) = A(0): C(0) = B(0)
FOR J = 1 TO N - 1
  B(J) = B(J - 1) * X + A(J)
  C(J) = C(J - 1) * X + B(J)
NEXT J
B(N) = B(N - 1) * X + A(N)
END SUB
```

図 9

f（x）＝0 が n 次の代数方程式の場合，f（x）ならびに f'（x）の値を計算するにはホルナー法を用いるのが速い。

n 次の多項式

$f(x)=a_0x^n+a_1x^{n-1}+\cdots+a_{n-1}x+a_n$

を $x-\alpha$ で割った商を

$g(x)=b_0x^{n-1}+\cdots+b_{n-2}x+b_{n-1}$

余りを $r_1=b_n$ とする。

$f(x)=(x-\alpha)g(x)+r_1$

から

$$\begin{cases} b_0=a_0 \\ b_j=a_j+\alpha b_{j-1} \quad (j=1,2\cdots,n) \end{cases}$$

なる漸化式が得られる。

$f'(x)=g(x)+(x-\alpha)g'(x)$

である。そこでさらに g（x）を $x-\alpha$ で割った商を

$h(x)=c_0x^{n-2}+\cdots c_{n-3}x+c_{n-2}$

余りを $r_2=c_{n-1}$ とする。

$g(x)=(x-\alpha)h(x)+r_2$

から漸化式

$$\begin{cases} c_0 = b_0 \\ c_j = b_j + \alpha\, c_{j-1} \quad (j = 1, 2 \cdots, n-1) \end{cases}$$

を得る。したがって，$f(\alpha)$ ならびに $f'(\alpha) = g(\alpha)$ を求めるには，上の漸化式を用いて b_n と c_{n-1} を計算すればよい。（ホルナー法）

図9はダンデリン法をプログラミングしたものである。

ここで，プログラム中の DIM SHARED について説明を加えよう。

変数の内容を参照できる範囲に，グローバル（大域的）とローカル（局所的）の2種類ある。従来型 BASIC の変数はすべてグローバル変数であったが，Quick BASIC ではグローバル変数とローカル変数の2種類が指定できるようになった。そのため，プロシージャ間の独立性が高められ，プログラムが書き易くなったのである。

グローバル変数はプログラムの中のどんな部分からでも参照できるが，ローカルな変数はそれが宣言（使用）されているプロシージャの中だけで参照することができる。そこで，プログラムの全域で参照したい変数は，メインルーチンにおいて，DIM SHARED により，グローバル変数であることを宣言しておくのである。

```
CONST PI = 3.14159
X0 = 320: Y0 = 200: R = 110: C = 1: K = 0
CLS : INPUT "ord (1<=ord<=6) = "; ORD
SIZE = (3 ^ ORD - 1) / 2 * 3
DIM SHARED X(SIZE), Y(SIZE)
CALL HEXAGON(C, X0, Y0, R, K)
CALL RECURSE(ORD - 1, R * .5, C)
END

SUB HEXAGON (C, X0, Y0, R, K)
SHARED X(), Y()
FOR I = 0 TO 6
  A = I * 2 * PI / 6
  X = X0 + R * SIN(A): Y = Y0 + R * COS(A)
  IF C MOD 2 = 1 THEN
    IF I = 0 THEN X(K + 1) = X: Y(K + 1) = Y
    IF I = 2 THEN X(K + 2) = X: Y(K + 2) = Y
    IF I = 4 THEN X(K + 3) = X: Y(K + 3) = Y
  END IF
  IF C MOD 2 = 0 THEN
    IF I = 1 THEN X(K + 1) = X: Y(K + 1) = Y
    IF I = 3 THEN X(K + 2) = X: Y(K + 2) = Y
    IF I = 5 THEN X(K + 3) = X: Y(K + 3) = Y
  END IF
  IF I = 0 THEN PSET (X, Y), C ELSE LINE -(X, Y), C
NEXT I
END SUB

SUB RECURSE (ORD, R, C)
SHARED X(), Y()
IF ORD = 0 THEN EXIT SUB
I1 = 3 * (3 ^ (C - 1) - 1) / 2 + 1
I2 = 3 * (3 ^ C - 1) / 2
FOR I = I1 TO I2
    X0 = X(I): Y0 = Y(I): K = 3 * I
    CALL HEXAGON(C + 1, X0, Y0, R, K)
NEXT I
CALL RECURSE(ORD - 1, R * .5, C + 1)
END SUB
```

図　10—1

なお，Quick BASIC では CONST 文により記号定数を宣言することができる。記号定数とは，定数に名前をつけて，その名前を変数ではなく，定数として扱えるようにしたものである。

記号定数も，メインルーチンで宣言されたものはグローバルであり，SUB あるいは FUNDTION の中で宣言されたものはローカルである。

図10－1は図10－2に示す図形を描かせるプログラムである。メインルーチンで円周率の値を記号定数PIで定義している。

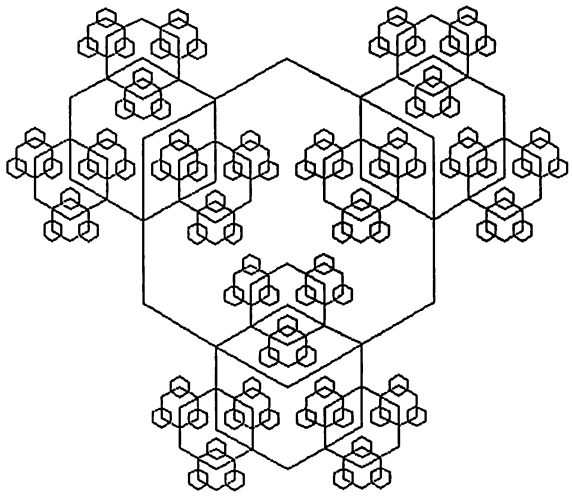

図　10—2

5．応　　用

整正方行列Aを対角化する問題を考えよう。それには，

1）行列の固有多項式の係数を求める（CHAR.POL）。

2）固有値を計算する（EIGENVAL）。

3）固有ベクトルを求める（EIGENVEC）。

4）固有ベクトルを列とする行列の逆行列を計算し，Aを対角化する（REGULAR）。特に，Aが対称行列の場合は直交行列を用いて対角化する（ORTHO）。

まず，n次の行列Aの固有多項式は

$$\phi_A(t) = \det(tE_n - A)$$
$$= t^n + C_1 t^{n-1} + \cdots + C_{n-1} t + C_n$$

ここに，E_n はn次の単位行列であり，

$C_k = (-1)^k \Sigma$（Aのk次主行列式）$(k = 1, 2, \cdots, n)$

特に，$C_1 = -(a_{11} + a_{22} + \cdots + a_{nn})$

$C_n = (-1)^n \det A$

である。図11のプログラムでは係数 C_k を配列ZP(K)に求めている。

```
SUB CHAR.POL (N%, Z() AS LONG, ZP() AS LONG)
DEFINT A-X: DEFLNG Y-Z
DIM C(7), Y(7, 14)
SHARED ZG()
ZP(0) = 1
FOR I = 1 TO N: ZP(I) = 0: NEXT I
FOR I = 1 TO N: ZP(1) = ZP(1) + Z(I, I): NEXT I
ZP(1) = -ZP(1)
FOR I = 1 TO N: FOR J = I + 1 TO N
  ZP(2) = ZP(2) + Z(I, I) * Z(J, J) - Z(I, J) * Z(J, I)
NEXT J, I
IF N = 2 THEN EXIT SUB
FOR I = 1 TO N - 2: FOR J = I + 1 TO N - 1
  FOR K = J + 1 TO N
    ZP(3) = ZP(3) + Z(I, I) * Z(J, J) * Z(K, K)
    ZP(3) = ZP(3) + Z(I, J) * Z(J, K) * Z(K, I)
    ZP(3) = ZP(3) + Z(I, K) * Z(K, J) * Z(J, I)
    ZP(3) = ZP(3) - Z(I, I) * Z(J, K) * Z(K, J)
    ZP(3) = ZP(3) - Z(I, K) * Z(K, I) * Z(J, J)
    ZP(3) = ZP(3) - Z(I, J) * Z(J, I) * Z(K, K)
  NEXT K
NEXT J, I: ZP(3) = -ZP(3)
IF N = 3 THEN EXIT SUB
FOR I1 = 1 TO N - 3: FOR I2 = I1 + 1 TO N - 2
  FOR I3 = I2 + 1 TO N - 1: FOR I4 = I3 + 1 TO N
    C(1) = I1: C(2) = I2: C(3) = I3: C(4) = I4
    FOR II = 1 TO 4: FOR IJ = 1 TO 4
      Y(II, IJ) = Z(C(II), C(IJ))
    NEXT IJ, II
    N1 = 4: N2 = 4: CALL SWEEP(N1, N2, Y(), R)
    IF R = 4 THEN
      ZD = 1: FOR I = 1 TO 4
                ZD = ZD * ZG(I) * Y(I, I)
              NEXT I
      ZD = ZD / ZT: ZP(4) = ZP(4) + ZD
    END IF
NEXT I4, I3, I2, I1
END SUB
```

図　11

行列の固有多項式は最高次の係数が１の多項式である。ｎ次方程式

$f(x) = x^n + \cdots + a_{n-1}x + a_n = 0 \quad (a_n \neq 0)$

の整数解を求めるには固有多項式ｆ(ｘ)を$(x-\alpha)$で割って，余りb_nをホルナー法を使って求めればよい。$b_n = 0$ならｆ（α）＝０である。固有値の候補αとしては$a_n \neq 0$のときその約数だけを対象にすればよい。図12のプログラムでは固有値α_kを配列Ｖ(k)に求めている。

```
SUB EIGENVAL (N%, ZP() AS LONG, S%, V() AS INTEGER)
DEFINT A-Z: DEFLNG Y-Z
DIM ZQ(7, 7)
S = 0: XX = 0
FOR I = 0 TO N: ZQ(0, I) = ZP(I): NEXT I
DO
  IF XX MOD 2 = 0 THEN X = -XX / 2 ELSE X = (XX + 1) / 2
  IF ABS(X) > ABS(ZQ(S, N - S)) THEN EXIT DO
  IF X <> 0 THEN Z = ZQ(S, N - S) / X
  IF Z = INT(Z) THEN
    DO
      ZQ(S + 1, 0) = 1
      FOR I = 1 TO N - S
        ZQ(S + 1, I) = ZQ(S, I) + X * ZQ(S + 1, I - 1)
      NEXT I
      IF ZQ(S + 1, N - S) <> 0 THEN EXIT DO
      IF ZQ(S + 1, N - S) = 0 THEN S = S + 1: V(S) = X
    LOOP
  END IF
  IF S = N - 1 THEN EXIT DO
  XX = XX + 1
LOOP
IF S = N - 1 THEN S = N: V(S) = -ZQ(N - 1, 1)
END SUB
```

図　12

固有ベクトルを計算するには，各整数固有値ｔに対し，斉次連立１次方程式（$tE_n - A$）ｘ＝０を解いて１組の基本解を求める。このとき，$tE_n - A$も整行列であるから，解ベクトルは必要なら成分の公分母をかけて，整ベクトルになるようにする。プログラムは長くなるので省略するが，こうして得られる一次独立なｎ個の固有ベクトルの全体を$W_1, W_2, \cdots, W_n$とし，これらを列ベクトルとして並べたｎ次整行列をＰとすれば（プログラムでは２次元配列ZX(,)に求めている），行列Ａは$P^{-1}AP$と積を計算することによって対角化される。

```
SUB REGULAR (N%, A() AS INTEGER, ZS&, ZX() AS LONG, Z() AS LONG)
DEFINT A-X: DEFLNG Y-Z
DIM ZY(7, 14)
SHARED ZG()
FOR I = 1 TO N: FOR J = 1 TO N
  Z(I, J) = ZX(I, J)
  IF I = J THEN Z(I, N + J) = 1 ELSE Z(I, N + J) = 0
NEXT J, I
N1 = N: N2 = 2 * N: CALL SWEEP(N1, N2, ZS, Z(), R)
FOR I = 1 TO N: FOR J = 1 TO 2 * N
  ZY(I, J) = Z(I, J)
NEXT J, I
ZD = 1: FOR I = 1 TO N
          ZD = ZD * ZG(I) * ZY(I, I)
        NEXT I: ZD = ZD / ZS
FOR I = 1 TO N: FOR J = 1 TO N
  ZY(I, N + J) = ZY(I, N + J) * ZD / (ZG(I) * ZY(I, I))
NEXT J, I
ZE = ABS(ZD)
FOR I = 1 TO N: FOR J = 1 TO N
  ZY(I, N + J) = ZY(I, N + J) * SGN(ZD)
NEXTT J, I
FOR I = 1 TO N: FOR L = 1 TO N
  Z(I, L) = 0
  FOR J = 1 TO N: FOR K = 1 TO N
    Z(I, L) = Z(I, L) + ZY(I, N + J) * A(J, K) * ZX(K, L)
  NEXT K, J
  Z(I, L) = Z(I, L) / ZE
NEXT L, I
END SUB
```

図　13

図13のプログラムでは対角化された行列を配列Z(,)に求めている。

```
SUB ORTHO (N%, A() AS INTEGER, ZX() AS LONG, Z() AS LONG)
DEFINT A-X: DEFLNG Y-Z
DIM ZY(7, 14), ZZ(7, 7), ZA(7), ZB(7), ZC(7)
DIM L(7), LL(7), YS(7), YR(7), ZS(7), ZR(7)
ZLCM = 1
FOR K = 1 TO N
  IF K = 1 THEN
    FOR I = 1 TO N: ZY(I, K) = ZX(I, K): NEXT I
  END IF
  IF K > 1 THEN
    FOR J = 1 TO K - 1
      ZB(J) = 0
      FOR I = 1 TO N
        ZB(J) = ZB(J) + ZY(I, J) * ZX(I, K)
      NEXT I
    NEXT J
    FOR I = 1 TO N
      ZY(I, K) = ZX(I, K) * ZLCM: Z = 0
      FOR J = 1 TO K - 1
        Z1 = ZLCM / ZA(J)
        Z = Z + ZB(J) * ZY(I, J) * Z1
      NEXT J
      ZY(I, K) = ZY(I, K) - Z
    NEXT I
  END IF
  FOR I = 1 TO N: ZC(I) = ZY(I, K): NEXT I
  CALL GCD(N, ZC(), ZG)
  FOR I = 1 TO N: ZY(I, K) = ZY(I, K) / ZG: NEXT I
  ZA(K) = 0
  FOR I = 1 TO N
    ZA(K) = ZA(K) + ZY(I, K) * ZY(I, K)
  NEXT I
  ZA = ZA(K): CALL SQDV(ZA, ZS, ZR)
  ZS(K) = ZS: ZR(K) = ZR
  ZA = ZLCM: ZB = ZA(K)
  ZLCM = ZLCM / EUCLID(ZA, ZB) * ZA(K)
NEXT K
ZA = ZLCM: CALL SQDV(ZA, ZS, ZR)
ZS(0) = ZS: ZR(0) = ZR
FOR I = 1 TO N: FOR L = 1 TO N
  Z(I, L) = 0
  FOR J = 1 TO N: FOR K = 1 TO N
    Z(I, L) = Z(I, L) + ZY(J, I) * A(J, K) * ZY(K, L)
  NEXT K, J
  IF I = L THEN Z(I, L) = Z(I, L) / ZA(I)
NEXT L, I
END SUB
```

図　14−1

行列Aが対称の場合には，固有ベクトルを列にもつ行列を Gram-Schmidt の直交化法を用いて直交行列とすることができる。直交行列の逆行列はその転置行列に等しいので，逆行列を求める計算は必要でない。ところで，正規直交化されたベクトルは次のような形になる。

$u_j = t(C_{1j}, C_{2j}, \cdots, C_{nj}) / sqr(\alpha_j)$

ただし，C_{ij} はすべて整数であり，

$\alpha_j = C_{ij}^2 + C_{2j}^2 + \cdots + C_{nj}^2$

である。グラムーシュミットの直交化は図14－１のプログラムで実施している。

```
SUB SQDV (ZA&, ZS&, ZR&)
DEFINT A-Y: DEFLNG Z
SHARED PM()
ZS = 1: ZR = ZA
FOR I = 1 TO 46
  ZP = PM(I) * PM(I): IF ZR < ZP THEN EXIT SUB
  WHILE (ZR / ZP) = INT(ZR / ZP)
    ZS = ZS * PM(I): ZR = ZR / ZP
  WEND
NEXT I
END SUB
```

図　14－2

```
SUB SWEEP (N1%, N2%, ZS&, Z() AS LONG, R%) STATIC
DEFINT A-X: DEFLNG Y-Z
DIM ZC(7): SHARED ZG(), P()
R = 0: ZS = 1
FOR I = 1 TO N1: P(I) = 0: NEXT I
FOR J = 1 TO N1
  FOR I = R + 1 TO N1
    IF Z(I, J) <> 0 THEN
      R = R + 1: P(J) = I
      FOR K = I + 1 TO N1
        WHILE Z(K, J) <> 0
          CALL ELM.UND(N2, I, J, K, ZS, Z())
        WEND
      NEXT K
    END IF
  NEXT I
  IF P(J) > 0 THEN
    IF R <> P(J) THEN EXCH N2, R, P(J), ZS, Z()
    FOR K = 1 TO R - 1
      IF Z(K, J) <> 0 THEN ELM.OVR N2, R, J, K, ZS, Z()
    NEXT K
  END IF
NEXT J
IF R = 0 THEN EXIT SUB
FOR K = 1 TO R
  FOR J = 1 TO N1: ZC(J) = Z(K, J): NEXT J
  CALL GCD(N1, ZC(), ZG): ZG(K) = ZG
NEXT K
P(0) = 0: P(R + 1) = N1 + 1
FOR I = 1 TO R
  J = P(I - 1) + 1
  WHILE Z(I, J) = 0
    J = J + 1
  WEND
  P(I) = J
NEXT I
FOR I = 1 TO R
  SN = SGN(Z(I, P(I))): Z = SN * ZG(I)
  FOR J = 1 TO N1: Z(I, J) = Z(I, J) / Z: NEXT J
  ZS = SN * ZS
  IF SN < 0 THEN
    FOR J = 1 TO N1
      Z(I, N1 + J) = -Z(I, N1 + J)
    NEXT J
  END IF
NEXT I
END SUB
```

図　15－1

なお，根号内から平方因子を外に出す。これは図14－２の SUB SQDV で行なっている。

CHAR.POL では行列式の値を求め，EIGENVEC では斉次連立１次方程式を解いて固有ベクトルを求め，REGULAR では逆行列を算出する。そのためガウスの消去法を用いる。行列式の値を算出するには，上三角化のために対角要素より下の要素を消去するだけでよいのであるが(ELM.UND)，CHAR.POL，EIGENVEC ならびに REGULAR で共通のサブルーチン SWEEP を使用するため，CHAR.POL でも，対角要素の下だけでなく上のほうも掃き出す(ELM.OVR)ことにする。そのために，整行列に対し次の４つの整数型行基本変形を用いる。

（１）Aの２つの行を入れかえる。行列式の符号が変る（SUB EXCH)。

（２）Aのある行に別の行の整数倍を加える。行列式の値は変らない（SUB ADD)。

（３）Aのある行に０でない整数αを乗ずる。行列式の値はα倍される(SUB MUL)。

（４）Aのある行をその行の成分の最大公約数βで割る。行列式の値は$1/\beta$倍される。

図15－１，図15－２に消去法のプログラムを示す。

```
SUB ELM.UND (N2%, I%, J%, K%, ZS&, Z() AS LONG)
DEFINT A-X: DEFLNG Y-Z
IF ABS(Z(I, J)) > ABS(Z(K, J)) THEN EXCH N2, I, K, ZS, Z()
ZC = -INT(Z(K, J) / Z(I, J))
CALL ADD(N2, I, K, ZC, Z())
END SUB

SUB ELM.OVR (N2%, R%, J%, K%, ZS&, Z() AS LONG)
DEFINT A-X: DEFLNG Y-Z
IF ABS(Z(R, J)) <= ABS(Z(K, J)) THEN
  ZC = -INT(Z(K, J) / Z(R, J))
  CALL ADD(N2, R, K, ZC, Z())
END IF
IF Z(K, J) = 0 THEN EXIT SUB
ZA = Z(R, J): ZB = Z(K, J): ZQ = Z(R, J) / EUCLID(ZA, ZB)
CALL MUL(N2, K, ZQ, Z()): ZS = ZS * ZQ
ZC = -Z(K, J) / Z(R, J): CALL ADD(N2, R, K, ZC, Z())
END SUB

SUB ADD (N%, I1%, I2%, ZQ&, Z() AS LONG)
DEFINT A-X: DEFLNG Y-Z
FOR J = 1 TO N
Z(I2, J) = Z(I2, J) + ZQ * Z(I1, J)
NEXT J
END SUB

SUB MUL (N%, K%, ZQ&, Z() AS LONG)
DEFINT A-X: DEFLNG Y-Z
FOR J = 1 TO N: Z(K, J) = ZQ * Z(K, J): NEXT J
END SUB

SUB EXCH (N%, I1%, I2%, ZS&, Z() AS LONG)
DEFINT A-X: DEFLNG Y-Z
FOR J = 1 TO N: SWAP Z(I1, J), Z(I2, J): NEXT J
ZS = -ZS
END SUB
```

図　15－2

参考文献

広森勝久　パソコンによる線型代数学　現代数学社　1987年

安居院猛，中嶋正之，永江孝規　やさしいフラクタル　工学社　1990年

河西朝雄　Ｑｕｉｃｋ　ＢＡＳＩＣ　初級プログラミング入門　技術評論社　1990年