第３章： 解説「魔方陣研究のための諸技法」： 摂田 寛二

今回は，新発見の報告ではない。今までやって来たことの「おさらい」をする。ついでに立脚地点をもう一度見直して，自明のはずのことを再度意識化して分析し，定義し直す。

第１節：移動変換と解の母集合/子集合

１．具体的な説明の例として「３次の魔方陣」から始める。
最も由緒の古い魔方陣である（以下「三方陣」という）。次のような解が知られている。

```
      1/          2/          3/          4/
 2  7  6     2  9  4     4  3  8     4  9  2
 9  5  1     7  5  3     9  5  1     3  5  7
 4  3  8     6  1  8     2  7  6     8  1  6

      5/          6/          7/          8/
 6  1  8     6  7  2     8  1  6     8  3  4
 7  5  3     1  5  9     3  5  7     1  5  9
 2  9  4     8  3  4     4  9  2     6  7  2
```

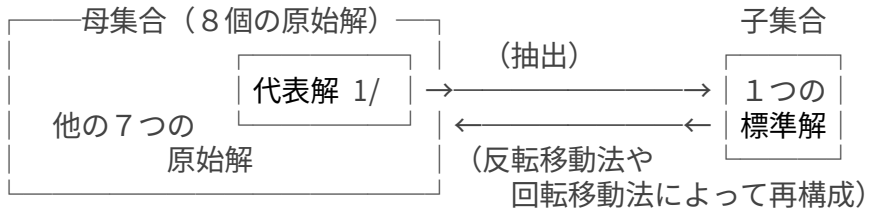
この８つをもって，取りあえず原始解の「母集合」とし，分析を始める。

２．先ず解 1/を原形と仮定し，他の解との比較分析を試みる。
［1］解 1/と 2/をくらべてみると，主対角線の位置にある３数（２，５，８）が同じである。この３数に注目して他の配置を見ると，７←→９，６←→４，１←→３ が鏡に映したみたいに入れ替わっている。
こういう相対的配置は，「軸対称」とも，「鏡映反転形」とも言う。この交換方法を知っていると，１つの原形からもう１つの異形を簡単に作り出すことができる。
［2］似たような関係が解 1/と 6/の間にも見られる。
今度は，縦の中央列の３数（７，５，３）が対称軸である。それに対して
２←→６，９←→１，４←→８ が左右方向に鏡映反転している。
［3］解 1/と 3/の間にも似た関係が見られる。
今度は，横の中央行の３数（９，５，１）が対称軸である。それに対して
２←→４，７←→３，６←→８ が上下方向に鏡映反転している。
［4］解 1/と 7/の間の関係は，最初の［1］と似ている。
今度は，別の主対角線の３数（６，５，４）が対称軸である。それに対して
７←→１，２←→８，９←→３ が鏡映反転している。
この４つの反転法を知っていると，１つの原形から４つの異形を簡単に作り出すことができる。全部の解を覚える必要が無く，１つの原形と４つの反転法を知っていれば良い。

解 1/と解 4/や 5/や 8/との間には，どんな関係があるだろうか。
［5］解 4/は，解 1/が時計回りに 90 度回転した形である。
［6］解 5/は，解 1/が**反**時計回りに 90 度回転した形である。
［7］解 8/は，解 1/が時計回りに 180 度回転した形である。別の見方をすれば，中央の５を中心点にして「点対称」移動した形になっている。
こういう回転移動法によっても，１つの原形から他の異形を作り出すことができる。

つまり，４つの反転移動法と３つの回転移動法を知ってさえいれば，あなたは１つの原形から簡単に他の７つの異形を作り出すことができるのである。

３．こういう反転移動や回転移動は，他の次数のどの魔方陣でも普遍的に成り立つことが知られている。そして，これを秘密にしておく理由はどこにも無いので，この知識と技術は早くから公開され，既に皆に認められている。
そして，ここからが一番大切なことだが，「この８つの解は，本質的に同一のタイプと見なすべきである。ゆえに任意の１つの解をもってこの８解の代表形としてよい」となる。



代表形として選別した８分の１個の解を，今後は「標準解」と言うことにしよう。
標準解は，あなたが自由に選んで良い。覚えやすい特徴を持ったものを選ぶと良い。
標準解を立てる理由は，元の母集合の他の解とお互いにどれも良く似ているし，それらの解を，良く知られた反転移動法や回転移動法によって，簡単に再構成できるからである。
つまり，母集合を簡単に再構成できるのだから，今後は子集合として１つの標準解と，それに母集合構成法を残そうということなのだ。
良く知られた構成法は，えてして殊更に語られず，暗黙の前提にされることが多い。

この構成法のことを，我々は「変換構成法」とも，あるいは単に「変換」とも呼んでいる。変換法は，お互いに異なる複数の解を「同類として」自己同一化するだけでなく，同類化した相手を再構成することができる働きをもつ。

４．この変換法はまた，解の探索・計数の手間ひまを省いてくれる働きを持つ。
三方陣や四方陣はそれほどでもないが，五方陣以上の高次方陣になると，解の総数がめっぽう多い。探索・計数の作業も面倒で，無茶苦茶に時間もかかる。それが，この変換法を効果的に用いると，手間ひまを大幅に減らすことができるのである。
例えば，上の三方陣で想像してほしい。「解を全部求めよ」と言われるより，「標準解を１つだけ求めれば良い」と言われる方が，ずいぶんと気が楽ではないか。あらかじめその解が１つしか無いとわかっていたとしても。‥‥
高次方陣では，パソコン - プログラムによる計数が当世流である。そうでもしなければ，手作業で全部が終わるまでに何百年もかかってしまうような問題が多い。

では，変換構成法をプログラムの中でどういうふうに用いるのか。
それが「不等式」なのである。
例えば，母集合の８個の原始解全部を見つけだすプログラムを組めたとする。
そして，次に原始解の出力を減らしたい場合を想像しよう。そういう時は，最後の段階で，例えば，(n１＜n９）というフルイを用意して，選別すれば良いのである。

三方陣の基本図：

[変数名]

n１	n２	n３
n４	n５	n６
n７	n８	n９

定義の連立方程式群：

```
n１＋n２＋n３＝Ｋ ....(1)
n４＋n５＋n６＝Ｋ ....(2)
n７＋n８＋n９＝Ｋ ....(3)
n１＋n４＋n７＝Ｋ ....(4)
n２＋n５＋n８＝Ｋ ....(5)
n３＋n６＋n９＝Ｋ ....(6)
n１＋n５＋n９＝Ｋ ....(7)
n３＋n５＋n７＝Ｋ ....(8)
```

n１＜n９としてフルイをかけると，

```
     1/          2/          3/          4/
  2  7  6     2  9  4     4  3  8     4  9  2
  9  5  1     7  5  3     9  5  1     3  5  7
  4  3  8     6  1  8     2  7  6     8  1  6    [Count = 4]
```

と出る。半分の４つしか解を出力しない。
プログラムの最後ではなく，途中でこの不等式を組み込むと，実行速度が上がる。という

のは，もう n1＞=n9 の場合を調べなくても良いので，探索時間が半分になるのである。
　では 1＜n9 としたうえで n3＜n7 とすると，どうなるか。

```
      1/          2/
 2  9  4     4  9  2
 7  5  3     3  5  7
 6  1  8     8  1  6    [Count = 2]
```

と，解が２つしか出て来ない。
　対角線上の３数の位置に鏡を置いて反転する作業が禁じられたのと，同じ意味を持つことがおわかりだろうか。一部の回転移動も封じられている。
　では，方陣がコロコロ回転しないようにするには，どう考えれば良いか。
　それには，（n１，n３，n７，n９）の中で n１に最小値が来るようにすれば良い。上部がいつでも軽ければ，達磨の起上がりこぼしのように一定の姿勢を保って安定になる。
　n1＜n9 としたうえで，n1＜n3＜n7 とすると，

```
      1/
 2  9  4
 7  5  3
 6  1  8    [Count = 1]
```

というふうに１つしか解を出力しないようになる。これを標準解とする。
　n1＜n3 を加えたことによって，上のリストの解 2/ が出なくなっている。左右反転が禁じられたからである。
　不等式をプログラムの早い段階から入れて行くと，探索時間が大幅に減ってくれる。

　変換構成法と不等式の各個が必ずしも厳密には一対一対応していないが，総合的に見て同じ結果になるように不等式をうまく組み合わせて立てるのである。
　不等式をプログラムに加えたからといって，間違った解が出て来る心配は無い。不等式は，正しい解を出力するプログラムに選別フルイを追加するだけだから。
　むろん，無駄を極力省こうと不等式を加え過ぎると，出て来るはずの正解を封じてしまう危険性はある。その辺のさじ加減が難しい。ある程度失敗して経験を積まないと，コツがわかりにくい。慣れが必要だ。慣れてくれば，何ということはなくなるはずだ。‥‥
　このようにして，効果的に「標準解」を捜し出して，プリントする。
　我々には，もう標準解しか必要ないから。

５． プログラムの実際を見てもらおう。先ず手順書を作る。そしてコーディングする。

三方陣の解を求めるプログラム手順書：

１． グローバル変数を用意する：
　LSM /* 行列和の 15 を入れる。定数値の代わり */
　cnt /* 解の計数カウンタ */
　配列としては， nm[10]: /* 変数 n１，n２，‥‥‥，n９のため */
　　uflg[10]: /* 使用／未使用を記憶しておくフラグ */

２． 用いるサブルーチン名／種類を定義する：‥‥‥

３．メイン - プログラム：
　タイトルをプリント。
　配列の内容をクリア・リセットしておく。
　初期値を与える：
　　LSM:=15; cnt:=0; /* ":=" の意味は，右辺値を左辺の変数に代入する */
　探索・計数の開始： サブルーチン calc01(); のコール

＊ コールした calc01() の帰還場所
終了プログラム：
　カウンタ値をプリントする。"OK!" を出す。
終了。

```
/*
三方陣の基本図：　　定義の連立方程式群：
　　[変数名]　　n１＋n２＋n３＝LSM ....(1)
　┌--┬--┬--┐　　n４＋n５＋n６＝LSM ....(2)
　｜n１｜n２｜n３｜　　n７＋n８＋n９＝LSM ....(3)
　├--┼--┼--┤　　n１＋n４＋n７＝LSM ....(4)
　｜n４｜n５｜n６｜　　n２＋n５＋n８＝LSM ....(5)
　├--┼--┼--┤　　n３＋n６＋n９＝LSM ....(6)
　｜n７｜n８｜n９｜　　n１＋n５＋n９＝LSM ....(7)
　└--┴--┴--┘　　n３＋n５＋n７＝LSM ....(8)
*/
```

４．探索・計数サブルーチン：

(1) calc01(){ /* 変数 n1 を決定する */
　局所変数 a を用意する。
　　a を制御変数にして for(){} ループを回す。初期値は a＝1;
　＊ a の終了値は 9。それを超えないように値を１ずつ増す。超えたら＊＊＊に進む。
　　a の取る値が今使用中であるかどうかチェックする。
　　　使用中なら，＊＊に進む。
　　　未使用なら，n1:=a; /* a の値を n1 に保存する */
　　　　a の値の使用フラグをセットし，「使用中」と宣言する。
　　　　次のサブルーチン calc02() を呼び出す。
　　　　サブルーチンから帰って来たら，フラグを未使用状態に戻す。
　　　＊＊
　　＊に戻る。
　＊＊＊
終了／帰還 }

(2) calc02(){ /* 変数 n2 を決定する */
　局所変数 a を用意する。
　　a を制御変数にして for(){} ループを回す。初期値は a＝1;
　＊ a の終了値は 9。それを超えないように値を１ずつ増す。超えたら＊＊＊に進む。
　　a の取る値が今使用中であるかどうかチェックする。
　　　使用中なら，＊＊に進む。
　　　未使用なら，n2:=a; /* a の値を n2 に保存する */
　　　　a の値の使用フラグをセットし，「使用中」と宣言する。
　　　　次のサブルーチン calc03() を呼び出す。
　　　　サブルーチンから帰って来たら，フラグを未使用状態に戻す。
　　　＊＊
　　＊に戻る。
　＊＊＊
終了／帰還 }

(3) calc03(){ /* 変数 n3 を決定する（ただし，n1＜n3） */
　局所変数 a を用意する。
　　a:=LSM-n1-n2; /* n1+n2+n3＝LSM という定義式を利用して n3 の値を計算する */
　　a の取る値が n1＜a＜10 の範囲内にあるかどうかチェックする。
　　　No! なら，＊に進む。
　　　Yes! なら，a の取る値が今使用中であるかどうかチェックする。
　　　　使用中なら，＊＊に進む。

```
        未使用なら，n3:=a;
        a の値の使用フラグをセットし，「使用中」と宣言する。
        次のサブルーチン calc04() を呼び出す。
        サブルーチンから帰って来たら，フラグを未使用状態に戻す。
      ＊＊
    ＊
終了／帰還 }

(4) calc04(){ /* 変数 n4 を決定する */
  局所変数 a を用意する。
    a を制御変数にして for(){} ループを回す。初期値は a＝1;
  ＊ a の終了値は 9。それを超えないように値を１ずつ増す。超えたら＊＊＊に進む。
    a の取る値が今使用中であるかどうかチェックする。
      使用中なら，＊＊に進む。
      未使用なら，n4:=a;
        a の値の使用フラグをセットし，「使用中」と宣言する。
        次のサブルーチン calc05() を呼び出す。
        サブルーチンから帰って来たら，フラグを未使用状態に戻す。
      ＊＊
    ＊に戻る。
  ＊＊＊
終了／帰還 }

(5) calc05(){ /* 変数 n7 を決定する（ただし，n3＜n7） */
  局所変数 a を用意する。
    a:=LSM-n1-n4; /* n1+n4+n7＝LSM という定義式を利用して n7 の値を計算する */
    a の取る値が n3＜a＜10 の範囲内にあるかどうかチェックする。
      No! なら，＊に進む。
      Yes! なら，a の取る値が今使用中であるかどうかチェックする。
        使用中なら，＊＊に進む。
        未使用なら，n7:=a;
        a の値の使用フラグをセットし，「使用中」と宣言する。
        次のサブルーチン calc06() を呼び出す。
        サブルーチンから帰って来たら，フラグを未使用状態に戻す。
      ＊＊
    ＊
終了／帰還 }

(6) calc06(){ /* 変数 n5 を決定する */
  局所変数 a を用意する。
    a:=LSM-n3-n7; /* n3+n5+n7＝LSM という定義式を利用して n5 の値を計算する */
    a の取る値が 0＜a＜10 の範囲内にあるかどうかチェックする。
      No! なら，＊に進む。
      Yes! なら，a の取る値が今使用中であるかどうかチェックする。
        使用中なら，＊＊に進む。
        未使用なら，n5:=a;
        a の値の使用フラグをセットし，「使用中」と宣言する。
        次のサブルーチン calc07() を呼び出す。
        サブルーチンから帰って来たら，フラグを未使用状態に戻す。
      ＊＊
    ＊
終了／帰還 }

(7) calc07(){ /* 変数 n6 を決定する */
  局所変数 a を用意する。
```

```
    a:=LSM-n4-n5; /* n4+n5+n6＝LSM という定義式を利用して n6 の値を計算する */
    a の取る値が 0＜a＜10 の範囲内にあるかどうかチェックする。
      No! なら，＊に進む。
      Yes! なら，a の取る値が今使用中であるかどうかチェックする。
        使用中なら，＊＊に進む。
        未使用なら，n6:=a;
        a の値の使用フラグをセットし，「使用中」と宣言する。
        次のサブルーチン calc08() を呼び出す。
        サブルーチンから帰って来たら，フラグを未使用状態に戻す。
      ＊＊
    ＊
終了／帰還 }

/*
三方陣の基本図：        定義の連立方程式群：
      [変数名]          n１＋n２＋n３＝LSM ....(1)
   ┌──┬──┬──┐       n４＋n５＋n６＝LSM ....(2)
   │n１│n２│n３│       n７＋n８＋n９＝LSM ....(3)
   ├──┼──┼──┤       n１＋n４＋n７＝LSM ....(4)
   │n４│n５│n６│       n２＋n５＋n８＝LSM ....(5)
   ├──┼──┼──┤       n３＋n６＋n９＝LSM ....(6)
   │n７│n８│n９│       n１＋n５＋n９＝LSM ....(7)
   └──┴──┴──┘       n３＋n５＋n７＝LSM ....(8)
*/

(8) calc08(){ /* 変数 n8 を決定する */
  局所変数 a を用意する。
    a:=LSM-n2-n5; /* n2+n5+n8＝LSM という定義式を利用して n8 の値を計算する */
    a の取る値が 0＜a＜10 の範囲内にあるかどうかチェックする。
      No! なら，＊に進む。
      Yes! なら，a の取る値が今使用中であるかどうかチェックする。
        使用中なら，＊＊に進む。
        未使用なら，n8:=a;
        a の値の使用フラグをセットし，「使用中」と宣言する。
        次のサブルーチン calc09() を呼び出す。
        サブルーチンから帰って来たら，フラグを未使用状態に戻す。
      ＊＊
    ＊
終了／帰還 }

(9) calc09(){ /* 変数 n9 を決定する（ただし，n1＜n9） */
  局所変数 a を用意する。
    a:=LSM-n7-n8; /* n7+n8+n9＝LSM という定義式を利用して n9 の値を計算する */
    a の取る値が n1＜a＜10 の範囲内にあるかどうかチェックする。
      No! なら，＊に進む。
      Yes! なら，a の取る値が今使用中であるかどうかチェックする。
        使用中なら，＊＊に進む。
        未使用なら，n9:=a;
        a の値の使用フラグをセットし，「使用中」と宣言する。
        次のサブルーチン calc10() を呼び出す。
        サブルーチンから帰って来たら，フラグを未使用状態に戻す。
      ＊＊
    ＊
終了／帰還 }

(10) calc10(){ /* 定義式に合致するかどうかチェックする */
```

```
  局所変数 sm1,sm2 を用意する。
    sm1:=n3+n6+n9;
    sm2:=n1+n5+n9;
    sm1 の取る和が LSM と等しいかどうかチェックする。
    sm2 の取る和が LSM と等しいかどうかチェックする。
    いずれか一方が No! なら，＊に進む。
    両方とも Yes! なら，次のサブルーチン pr1ans() を呼び出す。
  ＊
終了／帰還 }

(11) pr1ans(){ /* 解を１個ずつプリントするルーチン */
  ........ }
```

Ｃ言語による実際のプログラム（ミニ PASCAL みたいな書き方をしてある）

```
/** Magic Squares 3x3: File Name 'MS33S.c' **/
/**  By Kanji Setsuda on December 15, 2002 **/
/**/
/* Basic Form and Basic Conditions */
/*
  .--------.    n1+n2+n3=C ... (1)
  |n1|n2|n3|    n4+n5+n6=C ... (2)
  |--+--+--|    n7+n8+n9=C ... (3)
  |n4|n5|n6|    n1+n4+n7=C ... (4)
  |--+--+--|    n2+n5+n8=C ... (5)
  |n7|n8|n9|    n3+n6+n9=C ... (6)
  '--------'    n1+n5+n9=C ... (7)
                n3+n5+n7=C ... (8)
*/
/**/
#include <stdio.h>
/* Global Variables */
short LSM, cnt, cnt4;
short nm[10], uflg[10];
/* Sub-Routines */
void calc01(void), calc02(void), calc03(void);
void calc04(void), calc05(void), calc06(void);
void calc07(void), calc08(void), calc09(void);
void calc10(void);
void pr1ans(void);
/**/
/** Main Program **/
main(){
 short n;
  printf("\n** Compositions of Magic Squares of Order 3: **\n");
  printf("** The Standard Solution of Magic Square 3x3 **\n");
  for(n=0;n<10;n++){nm[n]=0; uflg[n]=0;}; /* Clear The Tables */
  LSM=15; cnt=0; cnt4=0;
  calc01();           /* Begin the Calculations */
  printf("[Count = %d]\n",cnt);
  printf(" OK!\n");
 return 0;
}
/**/
/* Begin the Calculations */
/* Set n1 */
void calc01(){
 short a;
```

```
  for(a=1;a<10;a++){
    if(uflg[a]==0){
      nm[1]=a; uflg[a]=1;
      calc02();
      uflg[a]=0;}
  }
}
/**/
/* Set n2 */
void calc02(){
 short a;
  for(a=1;a<10;a++){
    if(uflg[a]==0){
      nm[2]=a; uflg[a]=1;
      calc03();
      uflg[a]=0;}
  }
}
/**/
/* Set n3=LSM-n1-n2 & (n1<n3) */
void calc03(){
 short a;
  a=LSM-nm[1]-nm[2];
  if((nm[1]<a)&&(a<10)){
    if(uflg[a]==0){
      nm[3]=a; uflg[a]=1;
      calc04();
      uflg[a]=0;}
  }
}
/**/
/* Set n4 */
void calc04(){
 short a;
  for(a=1;a<10;a++){
    if(uflg[a]==0){
      nm[4]=a; uflg[a]=1;
      calc05();
      uflg[a]=0;}
  }
}
/**/
/* Set n7=LSM-n1-n4 & (n3<n7) */
void calc05(){
 short a;
  a=LSM-nm[1]-nm[4];
  if((nm[3]<a)&&(a<10)){
    if(uflg[a]==0){
      nm[7]=a; uflg[a]=1;
      calc06();
      uflg[a]=0;}
  }
}
/**/
/* Set n5=LSM-n3-n7 */
void calc06(){
 short a;
  a=LSM-nm[3]-nm[7];
  if((0<a)&&(a<10)){
```

```
    if(uflg[a]==0){
      nm[5]=a; uflg[a]=1;
      calc07();
      uflg[a]=0;}
  }
}
/**/
/* Set n6=LSM-n4-n5 */
void calc07(){
 short a;
  a=LSM-nm[4]-nm[5];
  if((0<a)&&(a<10)){
    if(uflg[a]==0){
      nm[6]=a; uflg[a]=1;
      calc08();
      uflg[a]=0;}
  }
}
/**/
/* Set n8=LSM-n2-n5 */
void calc08(){
 short a;
  a=LSM-nm[2]-nm[5];
  if((0<a)&&(a<10)){
    if(uflg[a]==0){
      nm[8]=a; uflg[a]=1;
      calc09();
      uflg[a]=0;}
  }
}
/**/
/* Set n9=LSM-n7-n8 & (n1<n9) */
void calc09(){
 short a;
  a=LSM-nm[7]-nm[8];
  if((nm[1]<a)&&(a<10)){
    if(uflg[a]==0){
      nm[9]=a; uflg[a]=1;
      calc10();
      uflg[a]=0;}
  }
}
/**/
/* Check The Line-Sum & of the Diagonals */
void calc10(){
 short sm1,sm2;
  sm1=nm[3]+nm[6]+nm[9];
  sm2=nm[1]+nm[5]+nm[9];
  if((sm1==LSM)&&(sm2==LSM)){pr1ans();}
}
/**/
/* Print 1 Answer */
void pr1ans(){
 short n;
  cnt++;
  printf("%14d/\n",cnt);
  printf("  .-----------.\n");
  printf("  |%2d |%2d |%2d |\n",nm[1],nm[2],nm[3]);
  printf("  |---+---+---|\n");
```

```
  printf("  |%2d |%2d |%2d |\n",nm[4],nm[5],nm[6]);
  printf("  |---+---+---|\n");
  printf("  |%2d |%2d |%2d |\n",nm[7],nm[8],nm[9]);
  printf("  '-----------'\n");
}
/**/
/*
** Compositions of Magic Squares of Order 3: **
** The Standard Solution of Magic Square 3x3 **
             1/
   .-----------.
   | 2 | 9 | 4 |
   |---+---+---|
   | 7 | 5 | 3 |
   |---+---+---|
   | 6 | 1 | 8 |
   '-----------'
[Count = 1]
 OK!
*/
```

第２節：補数と補数表現

次は，対称型の魔方陣や完全型の魔方陣の研究に良く出て来る「補数」と「補数表現形」について述べよう。

１．補数について

例えば，四方陣を例に取る。

下の図は，対称四方陣の具体例であるが，方陣の幾何学的中心を中心点として対称の位置にある２数の和が一定になるという性質を持たせている。

［対称四方陣］

1	14	15	4
8	11	10	5
12	7	6	9
13	2	3	16

例えば，左図では

１＋16＝17，　14＋３＝17，
15＋２＝17，　４＋13＝17，
８＋９＝17，　11＋６＝17，
10＋７＝17，　５＋12＝17，
12＋５＝17，　７＋10＝17，
　……
３＋14＝17，　16＋１＝17

となっている。

こういうふうに，加えて一定の値になる２数の特徴的なペアのことを，我々は「補数」ペアと呼んでいる。この語の使い方の実例としては，

「１と 16 は，お互いに 17 の補数を成す」とか，「16 は，１の補数である」とか，「ｎ１と n16 は，お互いに 17 の補数を成す」とか，「n16 は，ｎ１の補数である」とか言う。

これらの意味は，すべて

１＋16＝17　あるいは　ｎ１＋n16＝17　ということである。

この定数値 17 のことを「補数和」と言う。この値は，方陣の次数によって決まっていて，四方陣の時は 17，五方陣の時は 26，六方陣の時は 37，……で，一般にＮ方陣の時には，Ｎの２乗に１を加えた値である。

特に断らずただ単に「補数」と言う時は，個々の方陣に特有の定数の補数のことを言っているのだと理解してほしい。

２．補数表現形

一般に魔方陣では，補数の配置が非常に重要な意味を持っている。四方陣であれば，足して 17 になる２数のペアがどこにあるかと言うことが，その方陣の特徴を決定している。点対称型なのか，軸対称型なのか，あるいは汎対角線上に全部の補数ペアが並ぶのか否か。

我々は，補数の配置によって，方陣の種類さえ区別している。たいていの名前もそこに由来している。例えば，上の例は「補数対称型四方陣」(略称して「対称方陣」)だ。

そこで，我々は，補数の配置が直視できるように工夫して，原図のとなりに「補数表現形」も置いて研究する。例えば，下図のように。

基本ポジションと変数名

n1	n2	n3	n4
n5	n6	n7	n8
n9	n10	n11	n12
n13	n14	n15	n16

［補数表現形］

n1	n2	n3	n4
n5	n6	n7	n8
n8	n7	n6	n5
n4	n3	n2	n1

方程式は，
n１＋n16＝17
n２＋n15＝17
n３＋n14＝17
n４＋n13＝17
・・・・・・・・
n７＋n10＝17
n８＋n９＝17
と立てられている。

右図の意味は，n１の補数が n16 の位置にあることを示している。
n１＋n16＝17　と　n１＋n１＝17　との両方を表している。
同様にして　n２＋n２＝17 は，n２＋n15＝17 の意味。
n３＋n３＝17 は，n３＋n14＝17 の意味。
・・・・・・・・・・
n８＋n８＝17 は，n８＋n９＝17 の意味であることが，上図から見て取れる。
こうすると，例えば対角線の和は，
n１＋n６＋n６＋n１＝（n１＋n１)＋(n６＋n６）＝17＋17＝34
n４＋n７＋n７＋n４＝（n４＋n４)＋(n７＋n７）＝17＋17＝34
となっていることが，直視してわかるようになっている。
使い方を覚えると，いろいろ便利なことが多いので，ぜひマスターしてほしい。

補数表現形を用意すると，例えば，
n１＋n４＋n13＋n16＝34　や　n６＋n７＋n10＋n11＝34　などが，簡単に証明できる。
対応する補数表現形では，
n１＋n４＋n４＋n１＝（n１＋n１)＋(n４＋n４）＝17＋17＝34
n６＋n７＋n７＋n６＝（n６＋n６)＋(n７＋n７）＝17＋17＝34
だからである。
［問題］（n１＋n２＋n５＋n６)＋(n11＋n12＋n15＋n16）＝68　を証明して
（n１＋n２＋n５＋n６)＝(n11＋n12＋n15＋n16）＝34　を導け。
［ヒントは，（n１＋n２＋n５＋n６)＋(n３＋n４＋n７＋n８)＝68
および　（n３＋n４＋n７＋n８)＋(n11＋n12＋n15＋n16)＝68　］

第３節：移動変換と拡張方陣空間

１． 次は「汎対角線魔方陣」(略して「汎魔方陣」)を考える。４次を例に取ろう。

1	15	4	14
12	6	9	7
13	3	16	2
8	10	5	11

左図がその具体例である。この種の方陣には，
行列の定和や主対角線の定和ばかりでなく，
例えば，15＋９＋２＋８＝34
４＋７＋13＋10＝34
14＋12＋３＋５＝34
４＋６＋13＋11＝34
15＋12＋２＋５＝34
１＋７＋16＋10＝34
など，主対角線と平行に並んでいる４数の和も，行列の和に等しいという性質がある。この４数の作る斜めのラインのことを「汎対角線」とも「破対角線」とも言う。方陣の外枠に達すると，反対側に回る変な平行対角線だからである。
この種の方陣は，例えば最左の１列を最右列の右側に持って来て新たな方陣を構成しても，

元の条件をすべて満たす。つまり，行列の定和にとどまらず新たな主対角線４数の定和も成り立つ。

1	15	4	14	1
12	6	9	7	12
13	3	16	2	13
8	10	5	11	8

→

1	15	4	14	1
12	6	9	7	12
13	3	16	2	13
8	10	5	11	8

行列は，中の順番が変わっても，その内容までは変わらない。したがって，定和は保存される。変わるのは，主対角線だ。新たに出来た２本の主対角線の和は，この実例では，

15＋９＋２＋８＝34，　１＋７＋16＋10＝34　と定和を守っている。

上図の「破対角線」の内の２本が新たに主対角線になったからである。

　最右の１列を最左列の左側に持って来ても，同じように新対角線の定和が成り立たなければならない。同様に，最上の１行を最下行の下に持って来ても，成り立たなければならない。最下の１行を最上行の上に持って来ても，成り立たなければならない。

1	15	4	14
12	6	9	7
13	3	16	2
8	10	5	11
1	15	4	14

→

1	15	4	14
12	6	9	7
13	3	16	2
8	10	5	11
1	15	4	14

12＋３＋５＋14＝34
７＋16＋10＋１＝34

新対角線の定和が成り立っている。

　こうした行列の移動は，左右上下のいずれの方向にもできなければならない。連続しても良いし，方向を各種組み合わせても良い。

　移動するごとに新しい対角線が出来るが，それについても定和は必ず成り立たなければならない。この次々と出来る新対角線は，個々の四方陣では結局のところ８種類しか無いことがわかっている。「汎対角線」と言う。

２．汎対角線の全てを一望に直視できる工夫をしたのが，「方陣拡張空間」である。

6	9	7	12	6	9	7	12	6	9
3	16	2	13	3	16	2	13	3	16
10	5	11	8	10	5	11	8	10	5
15	4	14	1	15	4	14	1	15	4
6	9	7	12	6	9	7	12	6	9
3	16	2	13	3	16	2	13	3	16
10	5	11	8	10	5	11	8	10	5
15	4	14	1	15	4	14	1	15	4
6	9	7	12	6	9	7	12	6	9
3	16	2	13	3	16	2	13	3	16

［四方陣拡張空間］

　ただ，単純に元の方陣の１行１列を外側に順にコピーして行っただけだが，これがけっこう役に立つ。

　行列を１行１列ずつ移動するという操作が，この拡張空間内では，方陣枠を１筋ずつ移動する操作に対応する。その時に出来る新対角線は，既に元の対角線の隣にある。

　つまり，汎対角線を一望のもとに直視できる。

　汎対角線が８種類しか無いことも直に数えることができる。

　また，行列の移動によって出来る新しい方陣そのものも，すぐに読める。元の方陣の隣に既にあるから。

　見る方向を変えれば，反転形や回転形も直に見て取れる。

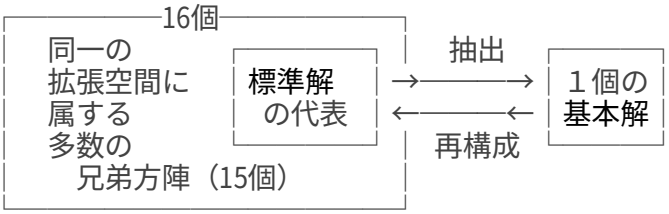
　逆方向から見たものを写して順方向に清書すれば，全く違って見える異形を作り出すこともできる。

例えば，次図を上の拡張空間から読み取ることができる。

1	14	4	15
8	11	5	10
13	2	16	3
12	7	9	6

これも元の条件をすべて満たす。行列の定和に限らず汎対角線の定和も成り立っている。
方向は変わっても，汎対角線の内容が少しも変わっていないことに気付くだろう。
いったいこの拡張空間から，お互いに異なった方陣を何個読み出すことができるだろうか。
単純な反転・回転形を省いて考えても，16 個読める。方陣枠の移動の仕方が 16 通りあるから。
左上の ｎ１の位置に来る数値の違いで数えて良い。

３．この拡張空間から読み取れる 16 個の新方陣を，果たして「異形」とすべきかそれとも「同一解」に属すると見なすか，議論の分かれるところである。
小生は，同じ方陣拡張空間に属する異形方陣をすべて同一化する立場をとる。これらをあえて同一解と見なし，そして「基本解」の概念を提案したい。



何故なら，16 個から１個を選出することも，その選ばれた１個から元の 16 個を再構成することも，方陣拡張空間を作ることによって，簡単に実現できるからである。

そして，覚えやすい特徴を持った代表解である「基本解」には，ｎ１＝１のタイプのものを選びたい。さらに ｎ２＞ｎ４，ｎ５＞n13，ｎ２＞ｎ５ のものを選びたい。
基本解が他に幾つも見つかる場合でも，その全てについて同じ基準で代表解を選びたい。

上の拡張空間からは，ｎ１＝１のタイプのものだけでも，下の４通りを読み出せる。
ここから１つを代表形として選ぶためには，不等式でフルイをかけるしかない。不等号の向きは，反対でもかまわないが，一度決めたら最後まで押し通すこと。他でもそうしたい。

1/

1	15	4	14
12	6	9	7
13	3	16	2
8	10	5	11

2/

1	15	4	14
8	10	5	11
13	3	16	2
12	6	9	7

3/

1	14	4	15
12	7	9	6
13	2	16	3
8	11	5	10

4/

1	14	4	15
8	11	5	10
13	2	16	3
12	7	9	6

基本形から元の 16 個を再構成するには，ただ拡張空間を作るだけで良い。そこからただ読み出せば良いのである。

第４節：基本解だけを探す

０．プログラムで解を見つけ出す時でも，この不等式を積極的に活用する。
早い段階からプログラムに不等式を組み込んでおけば，基本解だけを探し出すことができる。他を探さないで済むので，これは時間と手間の節約になる。一石二鳥の効果がある。
具体的に４次の汎魔方陣の基本解のすべてを求めて見よう。用いる基本の変数名と方程式と不等式は，次の通りにする。

１．基本ポジションと変数名（汎対角線が見やすいように最小限に拡張した）

```
 n2  n3  n4 ┌──┬──┬──┬──┐ n1  n2  n3
            │n1│n2│n3│n4│
 n6  n7  n8 ├──┼──┼──┼──┤ n5  n6  n7
            │n5│n6│n7│n8│
 n10 n11 n12├──┼──┼──┼──┤ n9  n10 n11
            │n9│n10│n11│n12│
 n14 n15 n16├──┼──┼──┼──┤ n13 n14 n15
            │n13│n14│n15│n16│
            └──┴──┴──┴──┘
```

２． 連立方程式群：

n１＋n２＋n３＋n４＝Ｋ ....(1)	n１＋n５＋n９＋n13＝Ｋ ....(5)
n５＋n６＋n７＋n８＝Ｋ ....(2)	n２＋n６＋n10＋n14＝Ｋ ....(6)
n９＋n10＋n11＋n12＝Ｋ ....(3)	n３＋n７＋n11＋n15＝Ｋ ....(7)
n13＋n14＋n15＋n16＝Ｋ ....(4)	n４＋n８＋n12＋n16＝Ｋ ....(8)

これに８本の汎対角線定和の方程式が加わる。

n１＋n６＋n11＋n16＝Ｋ ....(9)	n３＋n８＋n９＋n14＝Ｋ ...(13)
n１＋n８＋n11＋n14＝Ｋ ...(10)	n３＋n６＋n９＋n16＝Ｋ ...(14)
n２＋n７＋n12＋n13＝Ｋ ...(11)	n４＋n５＋n10＋n15＝Ｋ ...(15)
n２＋n５＋n12＋n15＝Ｋ ...(12)	n４＋n７＋n10＋n13＝Ｋ ...(16)

３． 基本解のみを求めるために最初から n１＝１とする。
　不等式は，反転・回転移動を封じるために n２＞n５
　拡張空間内での ４個を１個に絞るために n２＞n４, n５＞n13 とおく。
（別に n４＜n13, n２＜n４, n５＜n13 などとしてもかまわないが，そうしたら最後までそうすること。）

４． 探索・計数の結果を下図に示す。
　これらは，お互いに異なった解である。それぞれ拡張空間を作って調べても，汎対角線の内容が異なる［2/, 3/は，とても良く似ているが，汎対角線の一部に内容の異なるところがある］。全然別の拡張空間に属する方陣であるから，さらに同一化する根拠はもう無い。よって，これら代表解３個の集合をもって「基本解」の最小の集合とする。

1/

1	15	4	14
12	6	9	7
13	3	16	2
8	10	5	11

2/

1	15	6	12
14	4	9	7
11	5	16	2
8	10	3	13

3/

1	15	10	8
14	4	5	11
7	9	16	2
12	6	3	13

５． 基本解との比定方法：
　もしあなたが，たまたま別の形の解を得たとする。それが次のようなものであったとしよ

```
 3  6  9 16        3  6  9 16  3  6  9 16        1 14 11  8        1 15  6 12
13 12  7  2   →   13 12  7  2 13 12  7  2   →   15  4  5 10   →   14  4  9  7
 8  1 14 11        8  1 14 11  8  1 14 11        6  9 16  3        11  5 16  2
10 15  4  5       10 15  4  5 10 15  4  5       12  7  2 13        8 10  3 13
   原形            3  6  9 16  3  6  9 16                            基本解
                  13 12  7  2 13 12  7  2
                   8  1 14 11  8  1 14 11
```

う。それは，上のどの基本解に比定されるだろうか。比定法を実例で示そう。

（1）原形から方陣拡張空間を作る
（2）方陣枠を移動して１を頂点に持つ四方陣を４個見つける。
（3）適当なものを選び，n2＞n5，n2＞n4，n5＞n13 となるように方向を並べ替える。
清書すると，上の基本解のリストの 2/ と同じになる。つまり基本解 2/ に比定される。
　１から左右上下の４つの方向に読み出せるが，１の隣にできるだけ大きい数字の来る方向を選ぶのがコツである。

６．基本解を識別することは，方陣拡張空間そのものの特徴を識別することと同じである。だから，汎魔方陣の研究は，拡張空間そのものの性質を研究することに他ならない。
　汎魔方陣には，特有の行列のローテーションがある。そのつど姿をダイナミックに変える不思議な魔方陣だが，一度作った拡張空間は変化しないので，そちらをトータルに研究する方がずっとわかりやすい。

　さらに高次方陣では，解の総数がベラボーに大きくなる。コンピュータに探索させても，メッポウ時間がかかる。こんな時，基本解だけを求めるのであれば，手間もひまもずいぶんと省ける。
　移動変換と拡張空間と不等式と基本解の知識・技法を組み合わせることで，最近ようやく求まった解の総数もある。工夫は，今後もますます必要になる。

第５節：汎魔方陣と補数

「補数」は，補数対称型魔方陣だけに許された専売品ではない。
実は，汎対角線魔方陣にとっても，非常に重要な存在だ。

１．例えば，汎四方陣の基本解で調べて見よう。どこに 17 の補数ペアがあるか，「補数表現形」を用いて表わして見よう。

1/

1	15	4	14
12	6	9	7
13	3	16	2
8	10	5	11

2/

1	15	6	12
14	4	9	7
11	5	16	2
8	10	3	13

3/

1	15	10	8
14	4	5	11
7	9	16	2
12	6	3	13

9＝17-8＝8
10＝17-7＝7
11＝17-6＝6
12＝17-5＝5
13＝17-4＝4
14＝17-3＝3
15＝17-2＝2
16＝17-1＝1

↓ C1/

1	2	4	3
5	6	8	7
4	3	1	2
8	7	5	6

↓ C2/

1	2	6	5
3	4	8	7
6	5	1	2
8	7	3	4

↓ C3/

1	2	7	8
3	4	5	6
7	8	1	2
5	6	3	4

[補数表現形]

　もう気がついたかも知れないが，ここで方陣空間を最小限に拡張してみよう。汎魔方陣の特徴を直視することができるようになる。例として基本解 C1/を選ぶ。
　下図の「方陣拡張空間」を見てほしい。
　そう，８本の汎対角線全ての上に全ての補数ペアが，１つおきに並んでいる。そして，
　1＋6＋1＋6＝（1＋1）＋(6＋6）＝17＋17＝34
　1＋7＋1＋7＝（1＋1）＋(7＋7)＝17＋17＝34

[方陣拡張空間]

7	5	6	8	7	5	6	8	7	5
2	4	3	1	2	4	3	1	2	4
6	8	7	5	6	8	7	5	6	8
3	1	2	4	3	1	2	4	3	1
7	5	6	8	7	5	6	8	7	5
2	4	3	1	2	4	3	1	2	4

2＋8＋2＋8＝（2＋2）＋(8＋8)＝17＋17＝34
2＋5＋2＋5＝（2＋2）＋(5＋5)＝17＋17＝34
3＋8＋3＋8＝（3＋3）＋(8＋8)＝17＋17＝34
3＋5＋3＋5＝（3＋3）＋(5＋5)＝17＋17＝34
4＋7＋4＋7＝（4＋4）＋(7＋7)＝17＋17＝34
4＋6＋4＋6＝（4＋4）＋(6＋6)＝17＋17＝34
・・・・・・・・

のようにして，汎対角線の定和を実現している。
大変に美しい構造だ。汎四方陣では必ず成り立つ。

また，この方陣に含まれる２×２の内包二方陣を調べてみると，どの場合でも，その４数の和が等しいと言う特質を持っている。

1＋2＋5＋6＝1＋(17-2)＋(17-5)＋6　＝17＋17-1＋1＝34
2＋4＋6＋8＝（17-2)＋4＋6＋(17-8）＝17＋17＋2-2＝34
4＋3＋8＋7＝4＋(17-3)＋(17-8)＋7　＝17＋17＋1-1＝34
5＋6＋4＋3＝（17-5）＋6＋(17-4)＋3＝17＋17＋1-1＝34
6＋8＋3＋1＝6＋（17-8）＋3＋(17-1)＝17＋17-2＋2＝34
8＋7＋1＋2＝（17-8）＋7＋(17-1)＋2＝17＋17-1＋1＝34
4＋3＋8＋7＝（17-4）＋3＋8＋(17-7)＝17＋17-1＋1＝34
3＋1＋7＋5＝3＋（17-1）＋(17-7)＋5＝17＋17＋2-2＝34
1＋2＋5＋6＝（17-1）＋2＋5＋(17-6)＝17＋17＋1-1＝34
・・・・・・・・

つまり，汎四方陣，完全四方陣，そして正方連結四方陣が，全て同じ解集合を持つのである。つまり，一致する。この「奇跡の一致」は，四方陣でしか見られない美しい現象だ。
高次の，例えば八方陣では，この３種類の方陣は，それぞれに異なった解集合を持つことが確かめられている。つまり，お互いに異なった種類の魔方陣なのである。

２．対称型との相違と関連

既に検討した「補数対称型四方陣」と「完全型汎四方陣」の対応する基本解を補数表現形で表して並べると，面白いことに気付かされる。試みに２つに共通の操作を施して見よう。

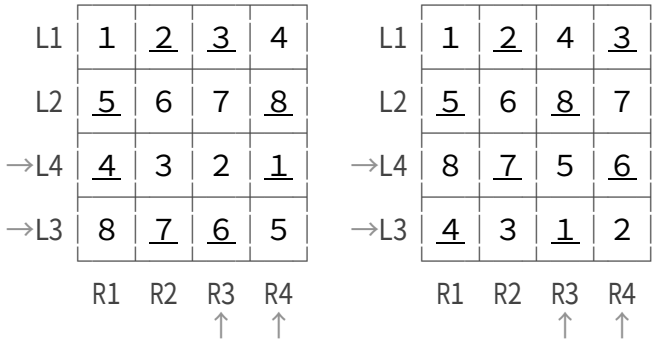
［対称四方陣］

	R1	R2	R3	R4
L1	1	2	3	4
L2	5	6	7	8
→L3	8	7	6	5
→L4	4	3	2	1

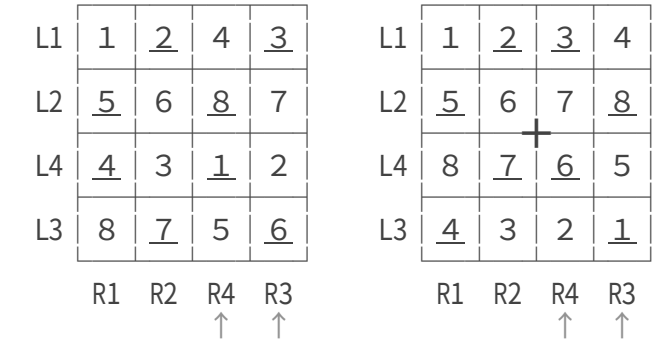
［完全四方陣］

	R1	R2	R3	R4
L1	1	2	4	3
L2	5	6	8	7
→L3	4	3	1	2
→L4	8	7	5	6

（1）両方とも方陣の第３行と第４行をそっくり入れ替える。



（2）さらに両方とも方陣の第３列と第４列をそっくり入れ替える。



結果を元の出発点と比較してほしい。相互に入れ替わっていることにお気付きだろうか。つまり，対称型が完全型に，完全型が対称型に書き変わっているのである。

４次に限らず偶数次の方陣では，同じような変換操作で「補数対称型」と「完全型汎魔方陣」がお互いに入れ替わるという面白い性質がある。

この変換は，逆もまた可なり。同じ手順でいつでも元に戻すことができる。

つまり，この変換は，両方の解を１対１に対応付ける働きを持っている。ゆえに両方の解集合には同数個の解が含まれると推理することができる。

実際に４次では，対称型も完全型も，基本解は同数の３個ずつしかない。
６次では，対称型も完全型も，解は同じ０個ずつしかない。
８次では，対称型も完全型も，解は無数にある。だが，同数個あると推理できる。
何故このようなことが可能なのか。両タイプの補数の並び方に共通性があるからである。

８次では，たとえ汎魔方陣でも，補数ペアの全部が汎対角線上に必ずしも並ぶとは限らない。並ばないものも数多くあるのである。そのタイプは，補数対称型には変換できない。

全ての補数ペアが汎対角線上に並んでいる汎八方陣だけが，対称型に変換することができる。そのタイプの解の総数は，対称型の解の総数と同じだけあると推理される。

これも「奇跡の一致」の１つだ。この一致を持つ汎魔方陣のことを，小生は今後「完全型汎魔方陣(Complete Magic Square)」と呼びたい。古くから使われて来たこの名前をあえて復活させたいと思う。昔通りの意味を新たに厳密に定義した上で。

第６節：内部変換と基本型

０．「方陣拡張空間」や「基本形カウント」の技法は，汎魔方陣の構造解析には有効だが，対称方陣の場合には必ずしも使えない。行列の逐次ローテーションが不可能だからである。

その代わりに「内部変換」という技法を使う。どうやって使うか。

１．例として，対称五方陣を研究する。１～25 の自然数を用いて，５×５の正方形数盤に並べる。行列の定和の他に 12 組の補数定和も要請する。
｛１，25｝，｛２，24｝，｛３，23｝，｛４，22｝，｛５，21｝，｛６，20｝，
｛７，19｝，｛８，18｝，｛９，17｝，{10，16}，{11，15}，{12，14} のペアを必ず点対称に

配置する。n13 には，つねに 13 が入る。13＋13＝26 だからである。
　補数対称型なので，ここでも積極的に「補数表現形」を使おう。

［基本ポジションと変数名］

	R1	R2	R3	R4	R5
L1	n１	n２	n３	n４	n５
L2	n６	n７	n８	n９	n10
L3	n11	n12	n13	n12	n11
L4	n10	n９	n８	n７	n６
L5	n５	n４	n３	n２	n１

　左図の n12 は，この場所（つまり n14）に 26-n12 の値が入っていることを示す。ゆえに n12＋n12＝26 であるとともに
　　n12＋n14＝26　を示す。
同様に，
　n11＋n11＝26，　n11＋n15＝26
　n10＋n10＝26，　n10＋n16＝26
　n９＋n９＝26，　n９＋n17＝26
　　・・・・・・・・・
　n２＋n２＝26，　n２＋n24＝26
　n１＋n１＝26，　n１＋n25＝26
を表わしている。

２．以下，上図を用いて変数名のまま考察を進める。
　方陣の幾何学的中心には n13 がある。ここには既に定数 13 が入っている。その段階から考察を始める。もちろん，行列の和は 65 である。
　上図の主対角線の内容を調べると，
n１＋n７＋n13＋n７＋n１＝(n１＋n１)＋n13＋(n７＋n７)＝26＋13＋26＝65
n５＋n９＋n13＋n９＋n５＝(n５＋n５)＋n13＋(n９＋n９)＝26＋13＋26＝65
となって，補数の配置によって，定和が成り立つことがわかる。

　試みに上図の第２行と第４行をそっくり交換してみる（図２）。そして，その後で第２列と第４列をそっくり交換して見よう（図３）。

	R1	R2	R3	R4	R5
L1：	n１	n２	n３	n４	n５
L4：	n10	n９	n８	n７	n６
L3：	n11	n12	n13	n12	n11
L2：	n６	n７	n８	n９	n10
L5：	n５	n４	n３	n２	n１

（図２）

	R1	R4	R3	R2	R5
L1：	n１	n４	n３	n２	n５
L4：	n10	n７	n８	n９	n６
L3：	n11	n12	n13	n12	n11
L2：	n６	n９	n８	n７	n10
L5：	n５	n２	n３	n４	n１

↑（図３）↑

　図２，図３において次のことをぜひチェックしてほしい。
＊１　各行列の和に変わりは無いか。
＊２　各補数ペアの配置を調べて，いぜん点対称性が成り立つかどうか。
＊３　主対角線の定和に変わりは無いか。
n１＋n９＋n13＋n９＋n１＝(n１＋n１)＋n13＋(n９＋n９)＝26＋13＋26＝65
n５＋n７＋n13＋n７＋n５＝(n５＋n５)＋n13＋(n７＋n７)＝26＋13＋26＝65
n１＋n７＋n13＋n７＋n１＝(n１＋n１)＋n13＋(n７＋n７)＝26＋13＋26＝65
n５＋n９＋n13＋n９＋n５＝(n５＋n５)＋n13＋(n９＋n９)＝26＋13＋26＝65

　不思議なことに，これが全てＯＫなのである。つまり，補数対称型五方陣では，いつでも（2-4)全行交換および（2-4)全列交換が可能なのである。
　そのために，左右(2-4)，上下(2-4)，左右上下(2-4)全行/列交換操作によって，３つの異形を簡単に作り出すことができる。
　ということは，原形も含めた４つの異形の中から「代表形」を１つ選ぶことも可能になる。そういうことを意味している。基本形が１つと（2-4）変換法があれば，いつでも元の４つを再構成することができるからである。

（1-5）変換も可能。同様に（1-2)＆(4-5)ダブル交換も可能だ。実例として上下方

向のみの変換を示す（いずれも原形からの変換）。

```
L5：n５  n４  n３  n２  n１←
L2：n６  n７  n８  n９  n10
L3：n11  n12  n13  n12  n11
L4：n10  n９  n８  n７  n６
L1：n１  n２  n３  n４  n５←
    R1   R2   R3   R4   R5
```
（図４）（１-５)全行交換

```
L5：n５  n４  n３  n２  n１←
L4：n10  n９  n８  n７  n６←
L3：n11  n12  n13  n12  n11
L2：n６  n７  n８  n９  n10←
L1：n１  n２  n３  n４  n５←
    R1   R2   R3   R4   R5
```
（図５）(1-5)，(2-4)全行交換

```
L2：n６  n７  n８  n９  n10←
L1：n１  n２  n３  n４  n５←
L3：n11  n12  n13  n12  n11
L5：n５  n４  n３  n２  n１←
L4：n10  n９  n８  n７  n６←
    R1   R2   R3   R4   R5
```
（図６）(1-2)&(4-5)ダブル交換

```
L5：n１  n２  n３  n４  n５←
L4：n６  n７  n８  n９  n10←
L3：n11  n12  n13  n12  n11
L2：n10  n９  n８  n７  n６←
L1：n５  n４  n３  n２  n１←
    R5   R4   R3   R2   R1
```
（図７）(1-5),(2-4)全行全列交換

では，単純な左右反転や上下反転を，全行/全列交換の組み合わせで実現できるだろうか。
上下反転は，(1-5),(2-4)全行交換でできる。左右反転は，(1-5),(2-4)全列交換でできる。
点対称移動は,できるか。(1-5),(2-4)全列交換の後に(1-5),(2-4)全行交換を行えば良い。
第３行/第３列以外にであれば，たいていの移動は可能だ。

つまり，かなりの数の異形を全行/全列交換の組み合わせで作り出せると，言って良い。
ということは，かなりの数の異形をまとめて，そこから「代表解」を１つ任意に選出することができるということだ。１つの基本解と何種類もの全行/全列交換法があれば，いつでも元の異形たちを簡単に再構成することができるからである。

前節の(３-４)全行全列交換は異種異形を作り出す変換だったが，現節の(２-４)交換や(１-５)や(1-2)&(4-5)の交換は，同種異形を作り出す変換なので，「内部変換」と言う。

３．変換・不等式・基本形

対称型では，基本形をどう考えたら良いか。五方陣・七方陣・九方陣・‥‥など奇数次の方陣では，偶数次の方陣には無い難しさがある。
内部変換の可能性をさらに調べよう。五方陣でできることとできないことを。

	R1	R2	R3	R4	R5
L1	n１	n２	n３	n４	n５
L2	n６	n７	n８	n９	n10
L3	n11	n12	n13	n14	n15
L4	n16	n17	n18	n19	n20
L5	n21	n22	n23	n24	n25

内部変換をどのように組み合わせても，n１にあった数値を n３の位置に持って来ることができない。n11 にも移せない。
n３にあった数値は，簡単に n11 に移せる。原方陣を 90 度回転させても良いし，n１-n25 の対角線を軸に対称移動しても良い。n８を n23 にも移せる。
しかし，n１や n２は，n15 や n23 には方法的に移せない。
つまり，五方陣には，内部変換について相異なる２つの排他的な領域があるのだ。
n13(＝13）は，いま論外とする。

領域Ａ {(n１，n２，n６，n７)，(n４，n５，n９，n10)，
　　(n16，n17，n21，n22)，(n19，n20，n24，n25)} ，
領域Ｂ {(n３，n８，n18，n23)，(n11，n12，n14，n15)}

同じ領域内では，内部変換によって自由に数値が動くのに，違った領域には，変換で移動することができない。しかし，領域Ａ内ならば，数値１がどこにあっても，内部変換によっ

て左肩の ｎ１の位置にいつでも移すことができる。領域Ｂ内ならば，数値１がどこにあっても，内部変換によって中央上の ｎ３の位置にいつでも移すことができる。
　だが，数値１をどの領域におくかは，こちらでいちいち決めてやらなければならない。

　ということは，もし五方陣に「基本形」を考えるならば，最初から２つのタイプが必要ということだ。それに合わせた不等式の立て方と計数プログラムが必要になる。

　では，基本解の「総数」は，求めることができるか。
　タイプＡとタイプＢの総数を別々に計数し単純に足すのは，意味が無い。もともと数え方が違うのだから。
　足し算ができるには，数え方を共通にしなければならない。不等式の立て方もそろえなくてはならない。計数プログラムも，できれば１つにしたい。
　だが，そうすると今度は，基本形をあまり絞れない恨みが残るのである。
　その辺から我々の方の選択が前もって必要になる。
　小生は，２つのタイプに分けて，基本解をできるだけ少なくしたい立場だが，一緒にした場合の数字も，参考値として付することにしている。

［1］タイプを分けて「基本解」を計数する場合，
　Ａ： ｎ１＝１；ｎ５＜n13＜n25；ｎ２＜ｎ４；ｎ６＜n16：　　総数 1948 個
　Ｂ： ｎ３＝１；n11＜n15＜n23；n11＜n12＜n14；ｎ８＜n18：　　総数 1086 個
［2］一緒にして計数する場合，ｎ３＝１ から数え出す。
　不等式は，n3＜n23；n3＜n11＜n15；n3＜n8＜n18；n11＜n12＜n14：　　総数 3034 個

第７節：実践「対称汎五方陣」の攻略

０．偶数次の方陣には，点対称型と完全型の間に「型変換」の技法が見つかって，密接な関係が証明された。だが，奇数次の方陣にはそうした技法は見つかっていない。そのかわりに「補数対称型であってしかも同時に汎対角線型」と言うような奇跡の方陣が存在する。

１．「対称汎五方陣」は，補数対称型と汎対角線型魔方陣の性質を併せもっている。つまり，26 の補数ペアが全て対称の位置にあるだけでなく，かつ汎対角線も存在して定和を守っているのである。
　補数ペアは，全てが汎対角線上にあるわけではなく，わずかに主対角線上に４組あるだけだから，完全型の汎五方陣にはなっていない。分類上は，「非補数型の汎対角線魔方陣で，しかも同時に補数対称型魔方陣」ということになる。
　このような珍しい方陣の解の全数を発見・発表し確定したのはわが鈴木睦教授であるが，我々もこれを追体験して作って見よう。

対称汎五方陣

［基本ポジションと変数名］

n23	n24	n25	n21	n22	n23	n24	n25	n21	n22	n23
n３	ｎ４	ｎ５	ｎ１	ｎ２	ｎ３	ｎ４	ｎ５	ｎ１	ｎ２	ｎ３
ｎ８	ｎ９	n10	ｎ６	ｎ７	ｎ８	ｎ９	n10	ｎ６	ｎ７	ｎ８
n13	n14	n15	n11	n12	n13	n14	n15	n11	n12	n13
n18	n19	n20	n16	n17	n18	n19	n20	n16	n17	n18
n23	n24	n25	n21	n22	n23	n24	n25	n21	n22	n23
ｎ３	ｎ４	ｎ５	ｎ１	ｎ２	ｎ３	ｎ４	ｎ５	ｎ１	ｎ２	ｎ３

［最小限の拡張方陣空間］

２．既に別に詳しい研究報告を書いているので，式や論証の大部分は再録しない。ここでは，補数対称型と汎対角線型という異型方陣の特徴をどう折合いを付けて一緒に攻略するかという点だけを説明する。
　与えられた条件は，五方陣一般の条件式の他に，汎対角線の定和式 10 本，補数定和式 12 本が加わっている。方程式の数は多いので，それら全てを成り立たせる解の総数は極めて少ない。
　その多数の条件式をどう扱うか。
（１）補数対称型で成り立つ式は，全て成り立つ。
（２）汎対角線型で成り立つ式も，全て成り立つ。
（３）だから，両方の条件を積極的に組み合わせて，新しい関係式を導き出してもかまわないのである。
　例えば，汎対角線五方陣では，
　　ｎ１＋ｎ２＋ｎ５＋ｎ６＋n21＝65（ひし形連結５数の定和）
が成り立つことが知られている。
　これと，補数対称型で成り立つ　ｎ５＋n21＝26　を組み合わせると，
　　ｎ１＋ｎ２＋ｎ５＋ｎ６＋n21＝65
　-)　　ｎ５＋n21＝26
　　ｎ１＋ｎ２＋ｎ６＝39
が成り立つ。こういう関係式は，どんどん使って良い。同様に
　ｎ４＋ｎ５＋n10＝39，　n16＋n21＋n22＝39，……　なども使える。

３．不等式はというと，そうは行かない。
（１）補数対称型でしか使えない不等式は，使えない。
（２）汎対角線型でしか使えない不等式は，使えない。
（３）両方に共通の不等式しか使えない。
　不等式は，いくら立てても間違った解が出て来る心配はないが，たくさん立てると，解を絞り過ぎて全然出なくなってしまう心配がある。
　したがって，n1＝１の「標準形」は作らず，全行交換による異形も封じず，ただ方陣の反転・回転だけを封じる。
　つまり，ｎ１＜ｎ５＜n21; ｎ１＜n25 とのみ立てる。

４．プログラムはどう組むか。先ず手順を次のように設計する。

プログラムの手順(抜粋)：

［1］メイン - プログラム
```
タイトルをプリント
変数・配列変数の初期化
n13 に定数値 13 をセット，使用フラグ[13] をセット
リサーチ開始： stp01() をコール
リターン後は，次の終了プログラムを
カウンタ値をプリント
終了
```

［2］リサーチのためのサブルーチン集
```
stp01(); /*  独立変数 Ｎ１ と 従属変数 n25 の値を決める(N1＜n25)  */
  変数ａ，ｂを用意し，Ｎ１，n25 用とする。
  ａ:=1，2，3，...，12；  /*  Ｎ１＜n25 だから 12 までよい  */
  ＊ｂ:=26-ａ；  /*  n25 には Ｎ１ の補数値が入る  */
    ａ，ｂの値が使用中なら，＊＊へ
    ａ，ｂが未使用なら，
     Ｎ１:=ａ；  n25:=ｂ；
```

```
        使用フラグをセット
        stp02() をコール
        リターン後にフラグを未使用に
    ＊＊ａの値を１増して，＊に戻る。
    終了・リターン

  stp02(); /*  独立変数 Ｎ５ と 従属変数 n21を決める(N1＜N5＜n13)  */
    変数ａ，ｂを用意し，Ｎ５，n21 用とする。
    ａ:= n1＋１; ａ＜13; ａ:=ａ＋１ で for ループを作る
    /* こうすることで n1＜a＜13 つまり n1＜a＜n21 を得られる */
    ＊ｂ:=26-ａ；  /*  n21 には Ｎ５ の補数値が入る  */
      ａ，ｂの値が使用中なら，＊＊へ
      ａ，ｂが未使用なら，
        Ｎ５:=ａ；  n21:=ｂ；
        使用フラグをセット
        stp03() をコール
        リターン後にフラグを未使用に
    ＊＊ａの値を１増して，＊に戻る。
    終了・リターン

  stp03(); /*  独立変数 Ｎ２ と n24 を決める  */
    変数ａ，ｂを用意し，Ｎ２，n24 用とする。
    ａ:= １; ａ＜26; ａ:=ａ＋１ で for ループを作る
    ＊ｂ:=26-ａ；  /*  n24 には Ｎ２ の補数値が入る  */
      ａ，ｂの値が使用中なら，＊＊へ
      ａ，ｂが未使用なら，
        Ｎ２:=ａ；  n24:=ｂ；
        使用フラグをセット
        stp04() をコール
        リターン後にフラグを未使用に
    ＊＊ａの値を１増して，＊に戻る。
    終了・リターン

  stp04(); /*  従属変数 ｎ６ と n20 を決める  */
    ａ:=39-ｎ１-ｎ２;  /*  ｎ１＋ｎ２＋ｎ６＝39 だから  */
    もし（０＜ａ＜26）ならば，＊へ。  そうでなければ，＊＊へ。
    ＊ｂ:=26-ａ；  /*  n20 には ｎ６ の補数値が入る  */
      ａ，ｂの値が使用中なら，＊＊へ
      ａ，ｂが未使用なら，
        Ｎ６:=ａ；  n20:=ｂ；
        使用フラグをセット
        stp05() をコール
        リターン後にフラグを未使用に
    ＊＊終了・リターン

/*
   24 25 21 22 23 24 25 21 22
    4  5  1  2  3  4  5  1  2
    9 10  6  7  8  9 10  6  7
   14 15 11 12 13 14 15 11 12
   19 20 16 17 18 19 20 16 17
   24 25 21 22 23 24 25 21 22
    4  5  1  2  3  4  5  1  2
            [基本図]
*/
```

```
  stp05(); /*  独立変数 N３ と n23 を決める  */
    a:= １; ａ＜26; ａ:=ａ＋１ で for ループを作る
    ＊ｂ:=26-ａ；  /*  n23 には N３ の補数値が入る  */
      ａ，ｂの値が使用中なら，＊＊へ
      ａ，ｂが未使用なら，
        N３:=ａ；  n23:=ｂ；
        使用フラグをセット
        stp05() をコール
        リターン後にフラグを未使用に
    ＊＊ａの値を１増して，＊に戻る。
    終了・リターン

  stp06(); /*  従属変数 n４ と n22 を決める  */
    a:=65-n１-n２-n３-n５
    /*  これは  n１＋n２＋n３＋n４＋n５＝65（行和）だから  */
    もし（０＜ａ＜26）ならば，＊へ。  そうでなければ，＊＊へ。
    ＊ｂ:=26-ａ；  /*  n22 には n４ の補数値が入る  */
      ａ，ｂの値が使用中なら，＊＊へ
      ａ，ｂが未使用なら，
        n４:=ａ；  n22:=ｂ；
        使用フラグをセット
        stp06() をコール
        リターン後にフラグを未使用に
    ＊＊終了・リターン

  stp07(); /*  従属変数 n８ と n18 を決める  */
    n８:=65-n２-n６-n21-n23; n18:=26-n８；
        /*  n２＋n６＋n８＋n21＋n23:=65 だから  */
        stp08() をコール
    終了
/*
   24 25 21 22 23 24 25 21 22
    4  5  1  2  3  4  5  1  2
    9 10  6  7  8  9 10  6  7
   14 15 11 12 13 14 15 11 12
   19 20 16 17 18 19 20 16 17
   24 25 21 22 23 24 25 21 22
    4  5  1  2  3  4  5  1  2
            [基本図]
*/
  stp08(); /*  従属変数 n10 と n16 を決める  */
    n10:=65-n４-n８-n23-n25; n16:=26-n10；
        /*  n４＋n８＋n10＋n23＋n25＝65 だから  */
        stp09() をコール
    終了
  stp09(); /*  従属変数 n11 と n15 を決める  */
    n11:=65-n１-n６-n16-n21；  n15:=26-n11；
        /*  n１＋n６＋n11＋n16＋n21:=65(列和)  */
        stp10() をコール
    終了
  stp10(); /*  従属変数 n７ と n19 を決める  */
        n７:=65-n１-n３-n11-n13；  n19:=26-n７；
        /*  n１＋n３＋n７＋n11＋n13＝65 だから  */
        stp11() をコール
    終了
```

```
    ......
  stp13(); stp14(); stp15(); stp16();
  /*  行和・列和・対角線和が同じかどうかをチェックする。
      駄目なら，リターン  */

 [3] プリント - プログラム
  ansrecord(); pr4ans(); /*  見つけた解をプリントする。*/
```

だいぶ省略して書いたが，あと細かいところは実際のプログラムを見ていただこう。

```
/** Simultaneous Magic Squares of Order 5: **/
/* Both Self-complementary and Pan-diagonal */
/** File Name:'MS55Sml.c' by Kanji Setsuda **/
/* on MacOSX and Xcode 1.5; August 26, 2005 */
/**/
#include <stdio.h>
/**/
short int cnt, cnt4;
short LSM, CC;
short nm[26], uflg[26];
short tn[4][26];
/**/
void stp01(void), stp02(void), stp03(void), stp04(void);
void stp05(void), stp06(void), stp07(void), stp08(void);
void stp09(void), stp10(void), stp11(void), stp12(void);
void stp13(void), stp14(void), stp15(void), stp16(void);
void ansrecord(void), pr4ans(void);
/**/
/* Main Program */
int main(){
 short n;
  printf("\n** Simultaneous Magic Squares of Order 5: **\n");
  printf("* Both Self-complementary and Pan-diagonal *\n");
  printf("    ** List of 16 Standard Solutions: **\n");
  for(n=0;n<26;n++){nm[n]=0; uflg[n]=0;}
  LSM=65; CC=26; cnt=0; cnt4=0;
  nm[13]=13; uflg[13]=1;  /* Set n13=13 */
  stp01();               /* Begin the Calculations */
  uflg[13]=0;
  printf(" [Count = %d]\n",cnt);
  printf("  OK!\n");
 return 0;
}
/* Begin The Calculations */
/* Set n1 & n25 & (n1<n25) */
void stp01(){
 short a, b;
  for(a=1;a<13;a++){b=CC-a;
    if((uflg[a]==0)&&(uflg[b]==0)){
      nm[1]=a; nm[25]=b;
      uflg[a]=1; uflg[b]=1;
      stp02();
      uflg[b]=0; uflg[a]=0;}
  }
}
```

```
/* Set n5 & n21 & (n1<n5<n21) */
void stp02(){
 short a, b;
  for(a=nm[1]+1;a<13;a++){b=CC-a;
    if((uflg[a]==0)&&(uflg[b]==0)){
      nm[5]=a; nm[21]=b;
      uflg[a]=1; uflg[b]=1;
      stp03();
      uflg[b]=0; uflg[a]=0;}
  }
}
/* Set n2 & n24 */
void stp03(){
 short a, b;
  for(a=1;a<26;a++){b=CC-a;
    if((uflg[a]==0)&&(uflg[b]==0)){
      nm[2]=a; nm[24]=b;
      uflg[a]=1; uflg[b]=1;
      stp04();
      uflg[b]=0; uflg[a]=0;}
  }
}
/* Set n6=39-n1-n2 & n20 */
void stp04(){
 short a, b;
  a=39-nm[1]-nm[2];
  if((0<a)&&(a<26)){b=CC-a;
    if((uflg[a]==0)&&(uflg[b]==0)){
      nm[6]=a; nm[20]=b;
      uflg[a]=1; uflg[b]=1;
      stp05();
      uflg[b]=0; uflg[a]=0;}}
}
/* Set n3 & n23 */
void stp05(){
 short a, b;
  for(a=1;a<26;a++){b=CC-a;
    if((uflg[a]==0)&&(uflg[b]==0)){
      nm[3]=a; nm[23]=b;
      uflg[a]=1; uflg[b]=1;
      stp06();
      uflg[b]=0; uflg[a]=0;}
  }
}
/* Set n4=LSM-n1-n2-n3-n5 & n22 */
void stp06(){
 short a, b;
  a=LSM-nm[1]-nm[2]-nm[3]-nm[5];
  if((0<a)&&(a<26)){b=CC-a;
    if((uflg[a]==0)&&(uflg[b]==0)){
      nm[4]=a; nm[22]=b;
      uflg[a]=1; uflg[b]=1;
      stp07();
      uflg[b]=0; uflg[a]=0;}}
}
/* Set n8=LSM-n2-n6-n21-n23 & n18 */
void stp07(){
 short a, b;
  a=LSM-nm[2]-nm[6]-nm[21]-nm[23];
```

```
  if((0<a)&&(a<26)){b=CC-a;
    if((uflg[a]==0)&&(uflg[b]==0)){
      nm[8]=a; nm[18]=b;
      uflg[a]=1; uflg[b]=1;
      stp08();
      uflg[b]=0; uflg[a]=0;}}
}
/* Set n10=LSM-n4-n8-n23-n25 & n16 */
void stp08(){
 short a, b;
  a=LSM-nm[4]-nm[8]-nm[23]-nm[25];
  if((0<a)&&(a<26)){b=CC-a;
    if((uflg[a]==0)&&(uflg[b]==0)){
      nm[10]=a; nm[16]=b;
      uflg[a]=1; uflg[b]=1;
      stp09();
      uflg[b]=0; uflg[a]=0;}}
}
/* Set n11=LSM-n1-n6-n16-n21 & n15 */
void stp09(){
 short a, b;
  a=LSM-nm[1]-nm[6]-nm[16]-nm[21];
  if((0<a)&&(a<26)){b=CC-a;
    if((uflg[a]==0)&&(uflg[b]==0)){
      nm[11]=a; nm[15]=b;
      uflg[a]=1; uflg[b]=1;
      stp10();
      uflg[b]=0; uflg[a]=0;}}
}
/* Set n7=LSM-n1-n3-n11-n13 & n19 */
void stp10(){
 short a, b;
  a=LSM-nm[1]-nm[3]-nm[11]-nm[13];
  if((0<a)&&(a<26)){b=CC-a;
    if((uflg[a]==0)&&(uflg[b]==0)){
      nm[7]=a; nm[19]=b;
      uflg[a]=1; uflg[b]=1;
      stp11();
      uflg[b]=0; uflg[a]=0;}}
}
/* Set n9=LSM-n6-n7-n8-n10 & n17 */
void stp11(){
 short a, b;
  a=LSM-nm[6]-nm[7]-nm[8]-nm[10];
  if((0<a)&&(a<26)){b=CC-a;
    if((uflg[a]==0)&&(uflg[b]==0)){
      nm[9]=a; nm[17]=b;
      uflg[a]=1; uflg[b]=1;
      stp12();
      uflg[b]=0; uflg[a]=0;}}
}
/* Set n12=LSM-n2-n7-n17-n22 */
void stp12(){
 short a, b;
  a=LSM-nm[2]-nm[7]-nm[17]-nm[22];
  if((0<a)&&(a<26)){b=CC-a;
    if((uflg[a]==0)&&(uflg[b]==0)){
      nm[12]=a; nm[14]=b;
      uflg[a]=1; uflg[b]=1;
```

```
      stp13();
      uflg[b]=0; uflg[a]=0;}}
}
/* Checks n11+n12+n13+n14+n15=65 & Others */
void stp13(void){
 short sm1, sm2;
  sm1=nm[11]+nm[12]+nm[13]+nm[14]+nm[15];
  sm2=nm[3]+nm[8]+nm[13]+nm[18]+nm[23];
  if((sm1==LSM)&&(sm2==LSM)){stp14();}
}
/* Checks n1+n10+n14+n18+n22=65 & Others */
void stp14(void){
 short sm1, sm2, sm3;
  sm1=nm[1]+nm[10]+nm[14]+nm[18]+nm[22];
  sm2=nm[2]+nm[8]+nm[14]+nm[20]+nm[21];
  sm3=nm[2]+nm[6]+nm[15]+nm[19]+nm[23];
  if((sm1==LSM)&&(sm2==LSM)&&(sm3==LSM)){stp15();}
}
/* Checks n1+n10+n14+n18+n22=65 & Others */
void stp15(void){
 short sm1, sm2, sm3;
  sm1=nm[3]+nm[9]+nm[15]+nm[16]+nm[22];
  sm2=nm[3]+nm[7]+nm[11]+nm[20]+nm[24];
  sm3=nm[4]+nm[10]+nm[11]+nm[17]+nm[23];
  if((sm1==LSM)&&(sm2==LSM)&&(sm3==LSM)){stp16();}
}
/* The Last Checks */
void stp16(void){
 short sm1, sm2;
  sm1=nm[4]+nm[8]+nm[12]+nm[16]+nm[25];
  sm2=nm[5]+nm[6]+nm[12]+nm[18]+nm[24];
  if((sm1==LSM)&&(sm2==LSM)){ansrecord();}
}
/* Record the Answers 4 by 4 */
void ansrecord(){
 short n;
  cnt++;
  tn[cnt4][0]=cnt;
  for(n=1;n<26;n++){tn[cnt4][n]=nm[n];}
  cnt4++;
  if(cnt4==4){pr4ans(); cnt4=0;}
}
/* Print Each 4 Answers */
void pr4ans(){
 short l,l5,m,n;
  for(m=0;m<4;m++){printf("%16d/",tn[m][0]);}; printf("\n");
  for(l=0;l<5;l++){l5=l*5;
    for(m=0;m<4;m++){
      printf("  ");
      for(n=1;n<6;n++){printf("%3d",tn[m][l5+n]);}
    }
    printf("\n");
  }
}
/**/
** Simultaneous Magic Squares of Order 5: **
* Both Self-complementary and Pan-diagonal *
    ** List of 16 Standard Solutions: **
```

```
                1/                  2/                  3/                  4/
 1 15 24 18  7    1 23 20 14  7    1 15 22 18  9    1 23 20 12  9
23 17  6  5 14   15  9  2 21 18   23 19  6  5 12   15  7  4 21 18
10  4 13 22 16   22 16 13 10  4   10  2 13 24 16   24 16 13 10  2
12 21 20  9  3    8  5 24 17 11   14 21 20  7  3    8  5 22 19 11
19  8  2 11 25   19 12  6  3 25   17  8  4 11 25   17 14  6  3 25
                5/                  6/                  7/                  8/
 2 14 25 18  6    2 23 19 15  6    2 14 21 18 10    2 23 19 11 10
23 16  7  4 15   14 10  1 22 18   23 20  7  4 11   14  6  5 22 18
 9  5 13 21 17   21 17 13  9  5    9  1 13 25 17   25 17 13  9  1
11 22 19 10  3    8  4 25 16 12   15 22 19  6  3    8  4 21 20 12
20  8  1 12 24   20 11  7  3 24   16  8  5 12 24   16 15  7  3 24
                9/                 10/                 11/                 12/
 4 12 25 18  6    4 23 17 15  6    4 12 21 18 10    4 23 17 11 10
23 16  9  2 15   12 10  1 24 18   23 20  9  2 11   12  6  5 24 18
 7  5 13 21 19   21 19 13  7  5    7  1 13 25 19   25 19 13  7  1
11 24 17 10  3    8  2 25 16 14   15 24 17  6  3    8  2 21 20 14
20  8  1 14 22   20 11  9  3 22   16  8  5 14 22   16 15  9  3 22
                13/                 14/                 15/                 16/
 5 11 24 18  7    5 23 16 14  7    5 11 22 18  9    5 23 16 12  9
23 17 10  1 14   11  9  2 25 18   23 19 10  1 12   11  7  4 25 18
 6  4 13 22 20   22 20 13  6  4    6  2 13 24 20   24 20 13  6  2
12 25 16  9  3    8  1 24 17 15   14 25 16  7  3    8  1 22 19 15
19  8  2 15 21   19 12 10  3 21   17  8  4 15 21   17 14 10  3 21
[Count = 16]
 OK!
```

６．こういう両立型の珍しい方陣は，５次の場合だけにしか無いのだろうか。

鈴木教授の予想に従って探索したところ，７次でも９次でも多数の解が見つかった。「対称汎七方陣」や「対称汎九方陣」があるのである。しかも，その総数は予想以上に多い。

３次を除けば，およそ全ての奇数次の方陣に存在するものと予想される。しかし，何と言っても，最初の「対称汎五方陣」の解 16 個の稀少価値が特に高い。

およそ 20 年前には，誰もあることすら考えもしなかった方陣だ。

終わりに：中間的総括を

今回は，個々の方陣研究よりも，話題をより縦横に動かして研究技法一般を解説してみた。もう予定の紙数を過ぎたみたいなので，残りは次回に譲ることにして，総括をしたい。

小生の研究報告も，だいぶたまった。が，読み返してみると，ずいぶんと用語や方法適用の不統一が目立つ。自分で開発した技法なら尚のこと，今チェックし，早急に再確認すべき時期だろう，と思った。

人とディスカッションしている時，お互いの前提・定義がわからず，コミュニケーションが頓挫・錯綜してしまうことがよくある。そんな時はイライラせずに，いつも最初に立ち返って，わかり切ったことから一々確かめて行くしかない。

先ず自分から用語の定義，および方法の前提条件を説明してみる。それが，この稿の先ず第一の目的であった。

特に変換，基本形，不等式の関係を説明して，事柄を自分でもハッキリ自覚したかった。

変換によって異形を生み出すことは，同時に異形を同類として自己同一化し，より小さい「基本形の集合」を作る働きを持つ。変換は，母集合と子集合の間に抽出と再構成と言う双方向の作用を可能にする。これが，直接的に基本解の「総数」に絡むのである。

それをプログラミングに結び付けると，不等式が必要になる。不等式は，基本形を絞り，探索・計数の手間ひまを省く特効薬だが，しかし，今のところ変換との対応関係が必ずしも自明とは限らない。しかも，個々の方陣によって事情が異なる。第一印象だけで言えば，個々の方陣は一般化を拒絶するほどに「個性的」なのだ。

各論的に工夫していると，全体的な方法的統一性が無くなってしまうことが多い。

今はまだ，このあたりが暗中模索的状態なのだ。いろいろやってみている段階で，功を焦り過ぎて，とかく無理をしがちだ。妥当と思われる中庸な結論は，まだ出ていない。

よりクリアな標準が出来上がるまでは，取りあえず自分の手の内をさらして，「こういう不等式の時にこういう総数になります」と，方法を一々明示するしかないと思われる。

総数は，数え方によって異なる。この当然のことが，一番扱いに難しいところなのである。

鈴木教授も，小生も，まだこのあたりで苦労している。

今回もまた，鈴木教授にはいろいろと御指導をいただいた。特にこの稿は，教授とのメールでのディスカッションから生まれた，いわゆる賜物である。記して，深く感謝申し上げたいと思う。

（初稿：February 9, 2001; 補筆清書稿：August 27, 2005;
Working on MacOSX and Xcode 1.5 by 摂田　寛二(Kanji Setsuda)）

E-mail Address:<jag12001@nifty.ne.jp>