数学未解決問題研究

エルデス・シュトラウスの予想を証明する.
「ニコニコしながら和分の積」＆
自動作問研究で考案したプログラミング理論
「万が一理論」からのアプローチ

菅野正人

第１章「和分の積」からのアプローチ

エルデス・シュトラウスの予想
ｎを２以上の任意の自然数とするとき

$$\frac{4}{n}=\frac{1}{x}+\frac{1}{y}+\frac{1}{z} \qquad (1)$$

を満たす自然数ｘ，ｙ，ｚが必ず存在する？

（１）式をよく見ると，これは左辺の分子の４を１にすれば，電気基礎で習う３本の抵抗が並列につながっているときの，合成抵抗を求める式に他ならない.「逆数の和の逆数」.言葉で言えばそう言うことだが、考え方としては容易ではない。しかも，抵抗の場合は３個にとどまらず何個も並列につながるのである．そこで，色々と考えてこんなキャッチフレーズを作ってみた.「ニコニコしながら和分の積」．どんなにたくさんつながった並列回路でも，計算の簡単そうな順から２個取り出して「和分の積」（$R_1\times R_2$／（R_1+R_2））の計算をして１個に置き換え，次にまた，２個を取り出して「和分の積」の計算をする事を繰り返せば、何個並列に抵抗が並んでいようと「和分の積」だけで合成抵抗が求まるという方法である.

$\frac{1}{R}=\frac{1}{R_1}+\frac{1}{R_2}$ 変形して $R=\frac{1}{(\frac{R_1+R_2}{R_1R_2})}$

$$\therefore \quad R=\frac{R1R2}{(R1+R2)}$$

（これはR_1とR_2の和分のR_1とR_2の積なのでこれを称して「和分の積」という）

「ニコニコしながら和分の積」と言うキャッチフレーズは理解しやすく、高校生などに教授するときには大いに役立つと考える.

本題に戻ろう．問題の式（１）をみると，その３本の抵抗の並列合成抵抗を求める式の４倍，つまりｘ，ｙ，ｚの３本の抵抗を使って２回「和分の積」をしてｎ／４を作り出すことが出来れば良いと言うことになる．ｎの条件は２以上という事で最初に作る数は２／４の０．５である.（ｘ，ｙ，ｚの逆数の和で作り出す数が４／ｎなのでｘ，ｙ，ｚの逆数の和の逆数で作る数はその逆数のｎ／４である．念のため）

$$\frac{4}{n}=\frac{1}{x}+\frac{1}{y}+\frac{1}{z}$$

ｘ，ｙ，ｚが自然数で，と言う条件を考えると始まりは１なので，１と１の和分の積をすれば，同じ数同士ではその半分になるので０．５は作れるが，もう一つ残るので、最初に１の２倍の２を二つ使い和分の積を行い半分の１を作れば，残りの１つを１にして目的の０．５は完成する．だから，答えのｘ，ｙ，ｚは２，２，１である。ｎ＝４を作りたければ，求める答えは４／４＝１なので４と４の和分の積で半分の２を作り，残る１つを２にすれば答えの１は求まる．答えのｘ，ｙ，ｚは４，４，２である．こんなふうに考えてみるとｎ／４を作りたければｘ，ｙ，ｚはｎ，ｎ，ｎ／２とすればすべて解決の様に思えるが，条件にあったようにｎが１の時はｎ／２が１／２で自然数にはならない．また，ｎ／２のｎが３，５，７，９・・・の奇数の場合は，２で割ると割り切れずに小数になるので奇数は別に考えなければいけない．しかし，２で割り切れる偶数はすべて自然数になるので、偶数のｎについては式を満たす自然数ｘ，ｙ，ｚは必ず存在すると言える．これで半分（全体の５０％）の数についての一般式が出来た事になる.

第２章　「万が一理論」からのアプローチ

次にｎが奇数の時の解の存在についてだが，その前に面白いことを考えたので紹介しておこう．（１）式を満たすｘ，ｙ，ｚが必ず存在すると仮定すれば「全く適当に自然数の中からｘ，ｙ，ｚを決め計算しｎが自然数になったときだけ完成」というアルゴリズムでプログラムを組んでパソコンで実行してみれば，ある程度の解は得られるのではないか．と言う考え方である．

これは２０年ほど前パソコンが出始めの頃に，電気基礎の授業でキルヒホッフの法則を教えるのに，連立三元１次方程式の問題で，「答えが整数」になり解くのが簡単な問題を大量に作ろうと思って考えた「万が一理論」というプロクラミングの理論である。(「万が一理論」という言葉は以下の様な意味を込めて作った造語である。)

人間にとっての「万が一」は不可能とか滅多にないというイメージであるが、メインクロックが１ＧＨｚ超の昨今のパソコンの世界では「万が一」(1万回に1回の事象）は一秒間に１０万回も起こる．極めて日常的に当たり前に起こる事象という事になる．つまり，人間が一生掛けても実現出来なかったような事が，「万が一」でも出来る可能性があればパソコンの世界ではできると言う事である．これまでのプログラムミングは一行でも短く１秒でも早く情報を処理できるようなアルゴリズムが要求されていた．それに対し「万が一理論」ではあえて遠回りをしながら「万が一」の可能性を模索する．「万が一理論」によるプログラミングは一見邪道のようなプログラミング手法であるが，現在のようにパソコンの性能がどんどん向上していく中で十分実用的な新しいプログラミング理論として面白いのではないかと思う．４年前に成功したナンバープレイスや数独の自動作問プログラムの開発にも大いに役に立った．

ここで，今回のこの問題の証明のためのアプローチとして「万が一理論」を元にして作ったプログラムを紹介する．

＊＊＊＊＊＊プログラムコード＊＊＊＊＊＊

```
Private Sub CommandButton1_Click()
（このリストの以下の部分をコメントは入力不要で End Subの前まで中身だけ入力）
Dim k As Double
      ’使用する変数の宣言
Dim x As Double
Dim y As Double
Dim z As Double
Dim n As Double
Randomize
      ’乱数系列の初期設定
For k = 1 To 1000000000
        ’何回やるか適当に決める
 x = Int(Rnd() * 100 + 1)
     ’適当に乱数の範囲を決める
 y = Int(Rnd() * 1000 + 1)
 z = Int(Rnd() * 10000 + 1)
 ’ ｎの計算
 n = 4 * x * y * z / (x * y + x * z + y * z)
’65536はワークシートの行数は２バイト
  if  n < 65536  then
’ｎが自然数（計算値ｎ＝整数化したｎ）
    If   n = Int(n) Then
   ’ワークシートに記録
 Worksheets("一覧").Cells(1, 1).Value = k
 Worksheets("一覧").Cells(n + 1, 1).Value = n
 Worksheets("一覧").Cells(n + 1, 2).Value = x
 Worksheets("一覧").Cells(n + 1, 3).Value = y
 Worksheets("一覧").Cells(n + 1, 4).Value = z
     End If
```

```
    End If
 Next k
 End Sub
```
＊＊＊＊＊プログラム　以上＊＊＊＊＊＊＊

このプログラムをエクセルの"一覧"と名前をつけたワークシート上にコマンドボタンを作り,コマンドボタンのコードとして貼りつけて実行してみると，どんどん答えの自然数が出てくる．しかも解は１つではないらしくどんどん数字が変わっていく．こうして，ｘ，ｙ，ｚの乱数の範囲設定を適当に変えながら実行してみると，どうしても出来ない数字がある．私の場合，最初は２４１だった．２４時間プログラムを実行しても出来なかった．どうして出来ないのか調べていくと，いくつかの重要な点が見つかった．２４１は素数である．２４１の前に出てきた素数を調べると，どの素数もｘｙｚの３つのうち２つはｎよりもかなり大きな自然数で，ｎで割ってみるとｎの自然数倍であることが分かった.

第３章エルデス・シュトラウスの予想の証明

そこでｎが奇数の時の解に戻って考えると，奇数は偶数のように２回の「和分の積」でｎ／４を作ることは出来ない．ｎ／２が小数になるためだ．しかし，奇数でも最終的にはｎ／４という数が作れなければこの問題は解決しない．そこで，ｎが奇数の時どうやってｎ／４を作ろうかと次のような方法を考えた.

それは、ｘをｎ／４を超えるｎ／４に最も近い自然数　ｉｎｔ（ｎ／４）＋１に設定し（ｉｎｔ関数はｎ／４の計算結果の整数部のみ利用すると言う意味）ここに全く自由な想像上の数として有理数αを考えて，ｘとこの有理数αを使ってｎ／４を作るという方法である．αは有理数なので，ｘに自然数という拘束があってもｘとαの和分の積で分数のｎ／４を作り出すことは可能である．しかも，ｘは自然数でと拘束されていてもｘの自然数上の変化に従って複数のαが存在することは明らかである.

$$\frac{n}{4} = \frac{x\ \alpha}{(x + \alpha)} \qquad \cdots(2)$$

変形して

$$\alpha = \frac{n\ x}{(4\ x - n)} \qquad \cdots(3)$$

ただし４ｘ≠ｎ

次に残りの自然数ｙ，ｚを和分の積を使って複数回出現する有理数αの中の一つを作ることが出来れば問題は解決である.

$$\alpha = \frac{y\ z}{y + z} \qquad \cdots(4)$$

有理数とは整数同士の商であり，正の整数は自然数である。自然数同士の和および積も自然数であるから、（４）のαも有理数である.

また，（４）を次のように変形してながめてみると

$$\frac{1}{\alpha} = \frac{1}{y} + \frac{1}{z} \qquad \cdots(5)$$

エジプト分数によればすべての分数は単位分数の和で表される．解は複数あり１つとは限らないが最低の項数は２である．２項にならない分数も存在するが，ｘをシフトしてαを変化させれば問題なく見つかる．単位分数とは分子が１である分数で分母は自然数である.

さらに，ここで改めて（３）を見ると

$$\alpha = \frac{n\ x}{(4\ x - n)}$$

ｘ＝１，ｎ＝１の時　α＝１／３となりこれは（４）のｙ，ｚの自然数の和分の積では最小値が１／２となって０．５以下の数は自然数の和分の積で作り出す事は出来ない.

しかし，ｎが２以上の自然数では、ｘ＝１で（３）のα＝１でα≧０．５となり（４）のｙ，ｚの自然数の和分の積で作り出す事が可能になる．

つまり，ｎ≧２，（４ｘ－ｎ）＞０の条件のもとで，ｘ，ｙ，ｚが自然数の範囲で自由に設定できるとすれば、（３）のｎとｘの変化によって作られるすべてのαは有理数しかも（４）と同様に自然数同士の商によって作られた有理数だから，仮に，同じ関係式（４）を満たす自然数ｙ，ｚが存在しないと仮定すれば（３）で作られるαの中には有理数以外の物が含まれていなければならない。自然数同士の商で有理数以外の数を作り出す事が出来るとすると，これは「有理数とは整数ａ，ｂを用いて（ａ／ｂ）で表されるすべての数」であるとする有理数の定義に反する．

∴　ｎ≧２，（４ｘ－ｎ）＞０の条件のもとで，有理数αを仲介して次のような等式が成立した時

$$\frac{n\ x}{4\ x\ -\ n}=\frac{y\ z}{y\ +\ z}\quad \cdots(6)$$

２以上の自然数ｎに対して前述の様に想像上の有理数αを仲介として考えてみると（１）を満たす自然数ｘ，ｙ，ｚは必ず存在すると言える。

証明終わり．

また，（４ｘ－ｎ）＞０よりｘ，ｙ，ｚの３つの自然数の中で最小の自然数Ａは必ず

Ａ＞ｎ／４

つまり　Ａ≧ｉｎｔ（ｎ／４）＋１である．

第４章　エルデス・シュトラウスの予想の証明から発見した単位分数に関する定理

この問題を考えていたら，この問題から発展して新しい定理を思い付いた．想像上の有理数を仲介役に使うとこんな定理も成立する．蛇足だが記しておく．

$$\frac{4}{n}=\frac{1}{A_1}+\frac{1}{A_2}+\frac{1}{A_3}+\frac{1}{A_4}$$

ｎを自然数とするとき，任意のｎに対し上式を満たす自然数A_1～A_4は必ず存在する．

$$\frac{5}{n}=\frac{1}{A_1}+\frac{1}{A_2}+\frac{1}{A_3}+\frac{1}{A_4}+\frac{1}{A_5}$$

ｎを自然数とするとき，任意のｎに対し上式を満たす自然数A_1～A_5は必ず存在する．

$$\frac{6}{n}=\frac{1}{A_1}+\frac{1}{A_2}+\frac{1}{A_3}+\frac{1}{A_4}+\frac{1}{A_5}+\frac{1}{A_6}$$

ｎを自然数とするとき，任意のｎに対し上式を満たす自然数A_1～A_6は必ず存在する．

となり次の様な定理まで証明できる．

エルデス・シュトラウスの予想から単位分数に関する定理

$$\frac{m}{n}=\frac{1}{A_1}+\frac{1}{A_2}+\cdots+\frac{1}{A_m}\quad \cdots(7)$$

ｍ≧４の任意の自然数ｍにおいて，任意の自然数ｎに対し上式を満たす自然数A_1～A_mは必ず存在する．

第５章　解を求める「万が一理論」によるプログラムの流れ

次に，この論理に従ってｎ≧２の自然数ｎに対して複数存在すると思われる解の一つを探していこうと思う．先ずは論理に従って解を探す手順を考える．

ここで，先に述べた「万が一理論」のプログラムで得たデータから解の在処について一つの仮説を立ててみた．

仮説

奇数のｎについての解は ｘを（ｎ／４）を超える（ｎ／４）に最も近い自然数　ｉｎｔ

（ｎ／４）＋１から探し始めて，ｎを自然数倍した自然数ｙ，ｚの組み合わせの中に必ず存在する.

この仮説からアルゴリズムを作る.
プログラムの流れ
与えられたｎに対し
１．ｘをｎ／４を超えるｎ／４に最も近い自然数　ｉｎｔ（ｎ／４）＋１に設定する
２．ｙ１＝１に設定
３．ｙの値をｙ１のｎ倍に設定する.
４．５の式の分母
（ｙ１＊(４＊ｘ－ｎ)－ｘ)≠０を確認
５．ｎ、ｘ、ｙ１を次式に代入し計算

$$z = \frac{n y1 x}{y1(4x - n) - x}$$

（上式は（６）の　ｎｘ／（４ｘ－ｎ）＝ｙ＊ｚ／（ｙ＋ｚ）　を仮説よりｎの自然数倍上に必ず存在するとしてｙ＝ｎ・ｙ１とおき，ｚ＝に変形した式で（６）が成立しているときのｚの値を求める.）
６．もしｚが自然数なら完成　９．へ
７．ｙ１を１刻みで変化させる。
８．ｙ１の設定範囲を超えても見つからなければｘを１刻みで増加し再検索
９．結果のｎ，ｘ，ｙ，ｚをワークシートに記録してｙ１をリセットしてループ終了
次のｎの検索に入る.

コード化すれば，たったの３０行程度のプログラムであるが，設定をすれば論理に沿って解を見つけ出せるアルゴリズムである。スペースの関係でサンプルプログラムは割愛するが，この論文が認知されれば次回にすべてを発表したい．このプログラムでｎ＝１００万まで作成したサンプルデータではｎが５０万以上１００万までの間で検証結果に３点ほどエラーがでた．これはエクセルの有効桁数が１５桁程度で丸められてしまうためと考えられる．エクセルでは１５桁程度なので信頼できる結果はｎ＝５０万程度までである．ところで，パソコンで無限大まで解を求める事はこの時代でもまだ出来るわけもなく，最大限可能な桁数まで解を求めたとしてもそれは数学的にはあまり意義がない．そこで，最後に上記のアルゴリズムによって作ったプログラムで求めた解のデータを分析して判明した一般式（恒等式）を記載して報告を終わりたい．　全体の約９８％が一般式という形で表現できた．解は複数存在していてこの一般式で求めたデータが唯一解ではないしもちろんｘ，ｙ，ｚは入れ替えても成立する．１００％一般式を求める事が理想だが調べていくうち，その存在自体が疑問になってきた．残りの２％の数に関しては解の在処についてグラフで考察してみたが，こちらも今回はスペースが足りないので次の機会に発表できればと思う.

第６章　解から一般式を求める（９８％）

偶数のデータ（全体の５０パーセント）
ｘ＝ｎ
ｙ＝ｎ
ｚ＝ｎ／２

奇数のデータ（４で割って１余る数と４で割って３余る数　　全体の５０％）

＊４で割って３余る数　　25%
ｘ＝　ｉｎｔ（ｎ／４）＋１
ｙ＝（ｎ＋４）ｎ
ｚ＝ ｘ＊ｙ

＊４で割って１余る数の内　６で割って３余る数　　8.3%
ｘ＝ｉｎｔ（ｎ／４）＋１
ｙ＝（ｉｎｔ（ｎ／12）＋2)ｎ
ｚ＝（ｉｎｔ（ｎ／12）＋2)ｙ
＊４で割って１余る数の内６で割って５余

る数　8.3%
x=ｉｎｔ（ｎ／４）＋１
y=（ｉｎｔ（ｎ／12）＋1)ｎ
z=（３（ｉｎｔ（ｎ／12）＋1）＾２－（ｉｎｔ（ｎ／12）＋1））ｎ

最後の残りが４で割って１余る数の内６で割って1余る数で全体の8.3%だがこれが一つの一般式では表現できずに，多くの集合に分かれている．
＊この４で割って１余る数の内６で割って1余る数
４で割って１余る数の内６で割って1余る数の集合は，１２で割って１余る数と言うことでＭＯＤ１２－１と呼ぶことにすると，これを１２０で割って剰余を見ると,全部で１０の集合が存在する．その中で調べてみると
⑴ ＭＯＤ１２－１の中で
＊１２０で割って１３余る数
ｘ＝ｉｎｔ（ｎ／４）＋１
ｙ＝（ｉｎｔ（ｎ／120）＊10＋2)ｎ
ｚ＝０．５ｘｙ

⑵ ＭＯＤ１２－１の中で
＊１２０で割って２５余る数
ｘ＝ｉｎｔ（ｎ／４）＋１
ｙ＝（ｉｎｔ（ｎ／120）＊10＋４)ｎ
ｚ＝０．２ｘｙ

⑶ ＭＯＤ１２－１の中で
＊１２０で割って３７余る数
ｘ＝ｉｎｔ（ｎ／４）＋１
ｙ＝（ｉｎｔ（ｎ／120）＊10＋４)ｎ
ｚ＝０．５ｘｙ

⑷ ＭＯＤ１２－１の中で
＊１２０で割って６１余る数
ｘ＝ｉｎｔ（ｎ／４）＋１
ｙ＝（ｉｎｔ（ｎ／120）＊10＋６)ｎ
ｚ＝０．５ｘｙ

⑸ ＭＯＤ１２－１の中で
＊１２０で割って８５余る数
ｘ＝ｉｎｔ（ｎ／４）＋１
ｙ＝（ｉｎｔ（ｎ／120）＊10＋８)ｎ
ｚ＝０．５ｘｙ

⑹ ＭＯＤ１２－１の中で
＊１２０で割って９７余る数
ｘ＝ｉｎｔ（ｎ／４）＋１
ｙ＝（ｉｎｔ（ｎ／120）＊10＋10)ｎ
ｚ＝０．２ｘｙ

⑺ ＭＯＤ１２－１の中で
＊１２０で割って１０９余る数
ｘ＝ｉｎｔ（ｎ／４）＋１
ｙ＝（ｉｎｔ（ｎ／120）＊10＋10)ｎ
ｚ＝０．５ｘｙ

以上の７つの集合に一般式が発見できた．これらはそれぞれの初期値から成り立つ一般式である．　そして，ＭＯＤ１２－１の中で１２０で割って1余る数　４９余る数　７３余る数の３つの集合が残った．これらの集合をＭＯＤ１２０－１－４９－７３と呼ぶことにする．　この３つの集合は３／１２０で全体の２．５％である．

ＭＯＤ１２０－１－４９－７３を１３２０で割って剰余を見ると
◎ＭＯＤ１２０－１－４９－７３の内
＊１３２０で割って１２１余る数
ｘ＝ｉｎｔ（ｎ／４）＋１
ｙ＝(ｉｎｔ（ｎ／1320）＊110＋14)ｎ
ｚ＝(ｉｎｔ（ｎ／1320）＊30＋2+81/99)ｙ

◎ ＭＯＤ１２０－１－４９－７３の内
＊１３２０で割って４８１余る数
ｘ＝ｉｎｔ（ｎ／４）＋１
ｙ＝(ｉｎｔ（ｎ／1320）＊110＋４４)ｎ

ｚ＝(ｉｎｔ（ｎ／1320）＊30＋１１)y

◎ ＭＯＤ１２０－１－４９－７３の内
　＊１３２０で割って１００９余る数
ｘ＝ｉｎｔ（ｎ／４）＋１
ｙ＝(ｉｎｔ（ｎ／1320）＊110＋８８)ｎ
ｚ＝(ｉｎｔ（ｎ／1320）＊30＋２３)y

◎ ＭＯＤ１２０－１－４９－７３の内
　＊１３２０で割って１２７３余る数
ｘ＝ｉｎｔ（ｎ／４）＋１
ｙ＝(ｉｎｔ（ｎ／1320）＊110＋１１０)ｎ
ｚ＝(ｉｎｔ（ｎ／1320）＊30＋２９)y

以上４つの集合について一般式が発見できた．これらの式もそれぞれの初期値から成り立つ一般式である．

しかし、
◎ ＭＯＤ１２０－１－４９－７３の内
＊１３２０で割って１，４９，７３，１６９，１９３，２４１，２８９，３１３，３６１，４０９，４３３，５２９，５５３，６０１，６４９，６７３，７２１，７６９，７９３，８４１，８８９，９１３，９６１，１０３３，１０８１，１１２９，１１５３，１２０１，１２４９余る数の２９の集合が一般式で表すことが出来ずに残った。全体の約２％にあたる．

第７章　終わりに

これまでに導き出された一般式はわずか３０行足らずのプログラムによって導き出されたデータから求めた．しかも，現行のプログラムは解の存在を証明できれば足りるので，最初の解が見つかれば時間短縮のためにそこで検索をやめているが，続ければ複数の解も導き出せる．とすれば，理にかなったアルゴリズムで組まれたプログラムはたった３０行足らずですべての解を導き出せると考えられる．このアルゴリズム自体がこの問題を実証するための新しい一般式（公式）の形と言えるのではないか。コンピュータの時代を迎えて数学の難問に対する新しい実証の形として，これも面白いのではないかと考えている．

２００８．５．２０　菅野正人

添付資料１
Original programming method
A programming method with “One times ten thousands Theory”　　（英語版）

添付資料２
パズルの作問について
独創のプログラミング　「万が一理論」のプログラミング手法について　（日本語版）

参考文献
数学の有名な未解決問題集
http://www.alpha-net.ne.jp/users2/eijitkn/mondai.html
(仮称)十進BASICのホームページ
http://hp.vector.co.jp/authors/VA008683/index.htm
私の備忘録
http://www004.upp.so-net.ne.jp/s_honma/reminder.htm
数学切り抜き帳　古代エジプトの分数
http://www.shinko-keirin.co.jp/kosu/mathematics/kirinuki/kirinuki53.html
art32m-k　ギャラリー
http://hw001.gate01.com/art32m-k/

この論文は日本数学協会が２００８年１２月２５日に発行した数学文化０１１の別冊数学文化日本数学協会論文集第４号に掲載されました。論文集は会員と主要大学に配布されました。

２００９．１．２０

添付資料１

Original programming method

A programming method with “One times ten th ousands Theory”

This puzzle is created to use the Original p rogramming method named “One times ten thou sands Theory”

Recently we are able to use a high ability P C (Personal Computer).

Main clock speed is more than a thousand mil lion times par second.

A phenomenon of one times ten thousands is a very rare phenomenon for human being

But it’s happen a hundred thousands times i n PC. It’s a frequent occurrence.

However it is impossible to do for human bei ng. If there is a little possibility to do, it is possible for PC. It is able to do in P C.

I created this method about twenty years ag o.

When I teach a subject of electricity in hig h school, many students stumble on Kirchhoff ’s Theory. This theory is easy to understand but it is very difficult to calculation.

I thought if there are some questions what e asy to calculation (For example all solution s are integer), many students are able to un derstand to Kirchhoff’s Theory

So I develop a method to make many questions

to easy to calculations automatically what named “One times ten thousands Theory”

I’d like to explain this idea concretely.

For example

Make simultaneous equations what example all solutions are integer.

First time.

Define a coefficient of Ｘ, Ｙ, Ｚ to reason ably integer.

$$３Ｘ＋４Ｙ＋５Ｚ＝a$$

$$４Ｘ－２Ｙ＋６Ｚ＝b$$

$$－２Ｘ＋３Ｙ＋２Ｚ＝c$$

Next is

Define the solution to Integer.

$$Ｘ＝１,\ Ｙ＝２,\ Ｚ＝３$$

And calculate ａ, ｂ, ｃ

$$a＝\ ３×１＋４×２＋５×３＝２６$$

$$b＝\ ４×１－２×２＋６×３＝１８$$

$$c＝－２×１＋３×２＋２×３＝１０$$

So simultaneous equation is

$$３Ｘ＋４Ｙ＋５Ｚ＝２６$$

$$４Ｘ－２Ｙ＋６Ｚ＝１８$$

$$－２Ｘ＋３Ｙ＋２Ｚ＝１０$$

So first time It’s define a coefficient of X, Y, Z reasonably integer and next time define solution to Integer that we can get a one question of simultaneous equations what solution is integer .

This method is very popular to make questions.

But my Original programming method is a little different from them.

I pay attention to a fact to exist a one question of simultaneous equations what all solution is integer and build algorithm of program.

The algorithm of program is All coefficient of X, Y, Z defined at random and calculate to use determinant and check the solutions. If when all of the solutions are integer then completion.

I don’t know when it’s completion

But so there is a fact to exist a one question of simultaneous equations what all solutions are integer that it’s complete someday.

This is the Original programming method of “One times ten thousands Theory”

A programming method until now is required more quickly response and more shorter code.

But my method is take a roundabout venture and seek the possibility of one times ten thousands to find solutions.

So recently ability of PC is very high that a time of reach the solution is gradually shorter.

So the time is gradually come up our time of life that it became to practical programming method enough.

2006.2.3 Masato.Kanno

添付資料２

パズルの作問について

独創のプログラミング

「万が一理論」のプログラミング手法について

ここで、今回公開したこれらのパズルの作問の手法について少しだけ解説する。

近年パソコンが普及し性能の高い物が自由に使えるようになり、これまでは出来なかったような事がプログラム次第で色々と可能になってきた。

「万が一理論」はパソコンの出始めの頃から、私が２０年来温めているプログラミング理論である。

メインクロックが１ＧＨｚ超の昨今のパソコンの世界では「万が一」（１万回に１回）の事が毎秒１０万回も起こっている。人間にとっての「万が一」は不可能とか滅多にないというイメージであるが、パソコンの世界では「万が一」は一秒間に１０万回も起こる。極めて日常的に当たり前に起こる事象という事になる。

つまり、人間が一生掛けても実現出来なかったような事が、「万が一」でも出来る可能性があればパソコンの世界ではできると言う事である。

私が最初にこのプログラミング手法を考えたのは今から２０年程前の事である。

高校で電気基礎という教科を教えているときに、生徒が最初につまずく「キルヒホッフの法則」という単元がある。生徒は法則自体はすぐに理解して３元１次連立方程式を作ることが出来るようになるのだがその計算が出来ない。基本的には中学の数学で解ける方程式だが、半数以上の生徒が計算途中の小数や分数の計算で間違えたりして、お手上げの状態になってしまう状況だった。本当は計算が出来ないだけなのに、結果的に「キルヒホッフの法則」は難しくて理解出来なかったという印象を生徒が持ってしまうのが非常に残念だったので、出来るだけ途中の計算が簡単な問題を作問しようと考えて、答えが簡単な整数になる問題を自動的にたくさん作れないかと考えたときに思い付いたのがこのプログラミング手法である。

具体的な例を挙げて説明すると

例えば未知数Ｘ，Ｙ，Ｚの３元１次連立方程式の問題を作問するとしよう。

易しい問題にして途中の計算が簡単になるように答えが１，２，３などの単純な整数になるような問題を作りたい。

Ｘ，Ｙ，Ｚの係数をあらかじめ適当に決めて

３Ｘ＋４Ｙ＋５Ｚ＝ａ

４Ｘ－２Ｙ＋６Ｚ＝ｂ

－２Ｘ＋３Ｙ＋２Ｚ＝ｃ

とする。

次に、答えをＸ＝１，Ｙ＝２，Ｚ＝３と決めてａ，ｂ，ｃを求めると

ａ＝　３×１＋４×２＋５×３＝２６

ｂ＝　４×１－２×２＋６×３＝１８

ｃ＝－２×１＋３×２＋２×３＝１０

となるので

３Ｘ＋４Ｙ＋５Ｚ＝２６

４Ｘ－２Ｙ＋６Ｚ＝１８

－２Ｘ＋３Ｙ＋２Ｚ＝１０

このように、はじめに答えを特定の整数の数値に決めておいてから式を作ればとりあえず当初の目的を満足する問題が一問出来るので、「世の中に目的とする形の問題は存在している」と言える。

このような作問法は一般的なプログラミングの手法であるが、「万が一理論」のプログラミングでは「世の中に目的とする形の問題は存在している」と言う点に着目してプログラムのアルゴリズムを考える。

Ｘ，Ｙ，Ｚにかかるすべての係数を乱数で全く無作為に設定し行列式で計算し答えをチェックする。そして、その答えが「３つとも整数になった時に完成」と言うアルゴリズムを組む。いつ出来上がるかは分からないが、前述の式のように「世の中に目的とする形の問題は存在している」ので何万回、何億回試行したとしてもいつかは必ず出来る。

このプログラムで１００問程問題を作り実際に授業で使ってみたところ途中の計算が簡単なのでほとんどの生徒が自分の力で解けるようになった。自分の力で答えを出すことが出来たという正

解のよろこびはその後の学習にも非常に大きく良い影響を与える。現在でも使っている最初の実用的な「万が一理論」のプログラムである。そのプログラムは当時の東京都工業高等学校電気教育研究会で発表した。

このようなプログラムでも、パソコンの性能がどんどん向上しているので結果が出るまでの時間もどんどん私達の生活時間に近づいてきた。これまでのプログラムミングは一行でも短く１秒でも早く情報を処理できるようなアルゴリズムが要求されていた。それに対し「万が一理論」ではあえて遠回りをしながら「万が一」の可能性を模索する。予め作為的に色々な要素を設定しないので出来上がった問題が非常に新鮮である。「万が一理論」によるプログラミングは一見邪道のようなプログラミング手法であるが、現在のようにパソコンの性能がどんどん向上していく中で十分実用的な新しいプログラミング理論として面白いのではないかと思う。

「万が一理論」では、ルールに合った問題を作ると言う以外の作為的な指示は一切与えていないので、どんな問題が出来上がるかは（またはルールによっては出来ないかも含めて）プログラム制作者にとってもまったくの未知数なのだ。

今回の５種類の新作パズルも「万が一理論」で作成した。これからも「万が一理論」のプログラミング手法から実用的な物を生み出す「ものづくり」の可能性を追究していきたい。

２００６．２．３　菅野正人