1 予備知識

変数がどのような値であっても成立する等式を恒等式と呼びます。例えば、$(x-4)(x+2)=(x-1)^2-9$ や $\dfrac{1}{x^2-1}=\dfrac{1}{2(x-1)}-\dfrac{1}{2(x+1)}$ などが恒等式です。前者は、左辺も右辺も展開すると x^2-2x-8 となりますし、後者は、両辺に x^2-1 を掛けると、左辺はもちろん 1 になり、右辺は

$$\frac{x+1}{2}-\frac{x-1}{2}=\frac{x+1-(x-1)}{2}=\frac{2}{2}=1$$

となることから、容易に証明することが出来ます。

2 恒等式を解く？

次の等式が変数 x に関する恒等式となるように、定数 A、B、C の値を決定してみます[*1]。

$$\frac{x+1}{x(x-2)(x+3)}=\frac{A}{x}+\frac{B}{x-2}+\frac{C}{x+3} \qquad [\star]$$

両辺に $x(x-2)(x+3)$ を掛けて、右辺を変数 x について整理します。

$$\begin{aligned} x+1 &= A(x-2)(x+3)+Bx(x+3)+Cx(x-2) \\ &= A(x^2+x-6)+B(x^2+3x)+C(x^2-2x) \\ &= (A+B+C)x^2+(A+3B-2C)x-6A \end{aligned}$$

係数を比較すると、連立方程式

$$\begin{cases} 0=A+B+C & \cdots① \\ 1=A+3B-2C & \cdots② \\ 1=-6A & \cdots③ \end{cases}$$

を解けばよいことがわかります。③より、$A=-\dfrac{1}{6}$ が求まり、これを①と②に代入し、B と C に関する連立方程式を解くと、$B=\dfrac{3}{10}$、$C=-\dfrac{2}{15}$ が求まります。

上記の手順を実行する関数、すなわち、「未知数を含む恒等式」と「変数」を引数にとり、未知数を求める関数 `solve_identity` を Maxima を使って定義てみましょう。

– solve_identity_1.mac –

```
1 solve_identity(EQ, X) := block(
2     [e, v, linsolvewarn: false],
3     e: fullratsimp(lhs(EQ) - rhs(EQ)),
4     e: expand(e * denom(e)),
5     v: listofvars(e),
```

[*1] Maxima には部分分数分解を求める関数 `partfrac` が用意されています。

```
6       if member(X, v) and length(v) >= 2 then (
7           v: delete(X, v),
8           solve(makelist(coeff(e, X, i), i, 0, hipow(e, X)), v
                )
9       )
10  );
```

あらかじめ、右辺を左辺に移項し（2 行目）、分母を払っておきます（3 行目）。Maxima では、命令 solve(関数, 変数) により、方程式「関数 = 0」を解くことが出来るため、等式「関数 = 0」ではなく、(= 0 を取り除いた)「関数」を扱う方針をとっています。

4 行目の listofvars は、引数に含まれる変数一覧をリストとして出力する関数です。先の恒等式 [★] を引数に与えた場合は、

```
[x, A, B, C]
```

を出力します。

5 行目の if 文は、無意味な恒等式[*2]を扱わないようにする処置です。

7 行目で連立方程式を解いています。ここで、hipowor は、多項式の次数を求める関数です。先の例の場合、命令

```
makelist(coeff(e, X, i), i, 0, hipow(e, X))
```

の実行結果は、

```
[6 A + 1, 2 C - 3 B - A + 1, - C - B - A]
```

となります。この方程式は（運良く？）独立ですが、一般には独立でない方程式が含まれます。その場合に関数 solve によって連立方程式を解くと、Maxima は親切に

```
solve: dependent equations eliminated: (3)
```

のように、独立でない式の番号を出力してくれます。2 行目の linsolvewarn: false は、このメッセージを出力しないようにする設定です。

ここで定義した関数 solve_identity の実行例は次の通りです。

```
(%i1) load("solve_identity_1.mac");
(%o1)                         solve_identity_1.mac
(%i2) solve_identity((x+1)/(x*(x-2)*(x+3)) = A/x + B/(x-2) + C/(x+3), x);
```

[*2] 変数 X が含まれていない場合と、変数 X 以外の変数（未知数）が含まれていない場合が「無意味な恒等式」に相当します。

```
                                  1      3         2
(%o2)                    [[A = - -, B = --, C = - --]]
                                  6      10        15
```

③ 改良（多変数バージョン）

前節で定義した関数 solve_identity は 1 つの変数に対して恒等式となるように未知数を決定する関数でしたが、これを複数変数バージョンに改良してみます。

– solve_identity.mac –

```
1  solve_identity(EQ, X) := block(
2      [e, v, w, linsolvewarn: false],
3      local(L),
4      e: fullratsimp(lhs(EQ) - rhs(EQ)),
5      e: expand(e * denom(e)),
6      v: listofvars(e),
7      if listp(X) then w: X else w: [X],
8      if length(v) >= length(w) + 1 and length(v) = length(
           unique(append(v, w))) then (
9          L[0]: [e],
10         for j: 1 thru length(w) do (
11             v: delete(w[j], v),
12             L[j]: [],
13             for k: 1 thru length(L[j-1]) do
14                 L[j]: append(L[j], makelist(coeff(L[j-1][k],
                       w[j], i), i, 0, hipow(L[j-1][k], w[j]))
                   )
15         ),
16         L[0]: unique(L[length(w)]),
17         solve(L[0], v)
18     )
19 );
```

本質的には一変数バージョンと同じですが、7 行目で、第 2 引数がリストでない（一変数の）場合 solve_identity(式, X) は、変数をリストに代入しています。これは、多変数の場合 solve_identity(式, [X, Y, ...]) だけでなく、一変数の場合 solve_identity(式, X) も平行して扱えるようにする工夫です。

まず、元の恒等式をリスト L[0] に代入し（9 行目）、各変数 w[j] に対して、係数方程式（多項式）リスト L[j-1] に含まれる各方程式の係数比較を行い、リスト L[j] に蓄積します（14

行目)。この操作を帰納的に実行し（10 行目の for 文)、最後に、重複を除去したものをリスト L[0] に代入し（16 行目)、解いています（17 行目)。

この関数の実行例は次の通りです。

```
(%i1) load("solve_identity.mac");
(%o1)                          solve_identity.mac
(%i2) solve_identity(1/(x^2 - 4) = A/(x - 2) + B/(x + 2), x);
                                     1        1
(%o2)                          [[A = -, B = - -]]
                                     4        4
(%i3) eq: (2*x-a*y)^2 - b*x^2 = y*(c*y - d*x);
                                  2      2
(%o3)                  (2 x - a y)  - b x  = y (c y - d x)
(%i4) solve_identity(eq, [x, y]);
                                                          2
(%o4)               [[b = 4, a = %r2, d = 4 %r2, c = %r2 ]]
```