maxima による行列計算超入門

足立健朗

Contents

Date: 2005/10/5.

1. 基本的な操作

1.1. **行列の入力.** まずは、**maxima** に行列を入力しないといけません。行列を入力するには、関数 **matrix** を使います。具体例で見てみましょう。

```
(%i1) matrix([1,2,3],[4,5,6],[7,8,9]);
```

$$\text{(\%o1)} \quad \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

```
(%i2) matrix([a,b,c,d],[e,f,g,h]);
```

$$\text{(\%o2)} \quad \begin{pmatrix} a & b & c & d \\ e & f & g & h \end{pmatrix}$$

関数 **matrix** の引数には、入力したい行列の成分を行ごとに”[“と “]“ でくくって、第 1 行から順に並べたものを使います。

1.2. **行列に名前をつける.** さて、**matrix** だけを使っても色々な行列の計算を **maxima** にさせることは可能なのですが、計算のたびに一々成分を入力するのでは全く疲れてしまいます。そこで、一度入力した行列に適当な名前を付けて、名前で色々な操作ができれば便利です。**maxima** で、何かの対象に名前を付けるは記号 **:** を使います。例を見てみましょう。

```
(%i3) A : matrix([1,2,3],[4,5,6],[7,8,9]);
```

$$\text{(\%o3)} \quad \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

```
(%i4) B : matrix([a,b,c,d],[e,f,g,h]);
```

$$\text{(\%o4)} \quad \begin{pmatrix} a & b & c & d \\ e & f & g & h \end{pmatrix}$$

これで、A, B といった名前で行列を使うことができるようになりました。この A や B を**変数**と言います。本当に A、B に入力した行列のデータが入っているか確認しましょう。

```
(%i5) A;
```

$$\text{(\%o5)} \quad \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

```
(%i6) B;
```

$$\text{(\%o6)} \quad \begin{pmatrix} a & b & c & d \\ e & f & g & h \end{pmatrix}$$

1.3. **名前のついた行列に別名をつける (エイリアスをつくる).** 上の操作で名前 A を与えた行列に、さらに別の名前を付けることもできます。A に別名 A1 を付けるときには

```
(%i7) A1 : A;
```

$$\text{(\%o7)} \quad \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

```
(%i8) A1;
```

$$\text{(\%o8)} \quad \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

とします。ここで注意することは、A と A1 は実体は同じものなので、後で説明する行列の操作で A の成分を変えてしまうと A1 の成分も同じように変わってしまうと言うことです。逆に、A1 の成分を変えると A の成分も同じように変わります。

1.4. **行列のコピーを作る.** それでは、行列の操作をしたいのだが、もとの行列のデータも残して置きたいときにはどうすればよいのでしょうか？そのような場合には、行列のコピーを作って使えばいいのです。そうすれば、もとの行列のデータは保存されます。A という行列と同じ成分をもつ、新たな変数 C (つまり A のコピー) を作るには、**maxima** の関数 **copymatrix** と記号 **:** を使って次の様にします。

```
(%i9) C :copymatrix(A);
```

$$(\%o9)\quad \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

```
(%i10) C;
```

$$(\%o10)\quad \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

これで、A と同じ成分を持つ行列 C が得られましたが、今度は A の成分を変化させても C の成分には影響がありませんし、逆もまたしかりです。

1.5. **行列の足し算.** 次に、**maxima** に行列の演算をさせる方法を見ていきましょう。まずは、同じ型の行列の足し算をします。これには、演算子 + を使います。ここで、上で使った行列の名前 A, B, A1, C とその内容の関係をキャンセルして、もう一度、A, B などを定義しなおすことにします。そのために、**maxima** の命令 **kill** を使います。

```
(%i11) kill(A, B, A1, C);
(%o11) DONE
```

これで、準備ができました。足し算の例を見て見ましょう。

```
(%i12) A : matrix([1,2],[3,4]);
```

$$(\%o12)\quad \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

```
(%i13) B : matrix([a,b],[c,d]);
```

$$(\%o13)\quad \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

```
(%i14) C : A + B;
```

$$(\%o14)\quad \begin{pmatrix} a+1 & b+2 \\ c+3 & d+4 \end{pmatrix}$$

この例では、A と B という 2×2-行列を与えて、A + B を計算し、その結果に C という名前を付けています。結果に名前を付ける必要がなければ、単に

```
(%i15) A + B;
```

$$(\%o15)\quad \begin{pmatrix} a+1 & b+2 \\ c+3 & d+4 \end{pmatrix}$$

としても結構です。

1.6. **行列のかけ算.** 次に行列と行列のかけ算をしてみましょう。線形代数学で学んだように、行列の積では行列の型が重要ですが、ここではそれについては触れません。行列の積を計算するには、演算子 . を使います。早速、例を見てみましょう。

```
(%i16) A . B;
```

$$(\%o16)\quad \begin{pmatrix} 2c+a & 2d+b \\ 4c+3a & 4d+3b \end{pmatrix}$$

```
(%i17) D : matrix([a,b,c],[d,e,f]);
```

$$(\%o17)\quad \begin{pmatrix} a & b & c \\ d & e & f \end{pmatrix}$$

```
(%i18) E : matrix([p,q],[r,s],[t,u]);
```

$$(\%o18)\quad \begin{pmatrix} p & q \\ r & s \\ t & u \end{pmatrix}$$

```
(%i19) F : D . E;
```

$$(\%o19)\quad \begin{pmatrix} ct+br+ap & cu+bs+aq \\ ft+er+dp & fu+es+dq \end{pmatrix}$$

この例では、まず上で定義した 2 つの 2×2-行列 A, B の積 AB を計算しています。次に、2×3-行列 D と 3×2-行列 E を定義してそれらの積 DE を計算し、それに名前 F を付けています。

積が定義できないような行列の組み合わせを入力すると、次のようなエラーメッセージがでます。

```
(%i20) D . A;
incompatible dimensions - cannot multiply
-- an error.  Quitting.  To debug this try DEBUGMODE(TRUE);
```

1.7. **行列のスカラー倍.** 行列のスカラー倍を計算するには、演算子 * を使います。ここで、例を見る前に今まで使った変数 (名前) をすべてキャンセルしましょう。そのための命令は **kill(ALL)** です。

```
(%i21) kill(ALL);
(%o0) DONE
```

それでは、例を始めます。

```
(%i1) A : matrix([a,b,c],[d,e,f]);
```

$$(\%o1)\quad \begin{pmatrix} a & b & c \\ d & e & f \end{pmatrix}$$

```
(%i2) B : 3 * A;
```

$$(\%o2)\quad \begin{pmatrix} 3a & 3b & 3c \\ 3d & 3e & 3f \end{pmatrix}$$

1.8. **行列の冪乗.** ある行列 A が正方行列ならば、A の冪乗を考えることができます。**maxima** で正方行列の冪乗を計算するには、演算子 ^^ を使います。例を見ましょう。

```
(%i3) kill(ALL);
(%o0) DONE
(%i1) A : matrix([2,3,4],[-1,0,3],[2,1,-2]);
```

$$(\%o1)\quad \begin{pmatrix} 2 & 3 & 4 \\ -1 & 0 & 3 \\ 2 & 1 & -2 \end{pmatrix}$$

```
(%i2) A^^3;
```

$$(\%o2)\quad \begin{pmatrix} 26 & 36 & 48 \\ -12 & 2 & 36 \\ 24 & 12 & -22 \end{pmatrix}$$

この例では A^3 を計算しています。本当に正しいかどうか、先ほど扱った行列の積を使って確認してみましょう。

```
(%i3) A . A . A;
```

$$(\%o3)\quad \begin{pmatrix} 26 & 36 & 48 \\ -12 & 2 & 36 \\ 24 & 12 & -22 \end{pmatrix}$$

確かに正しいようです。冪乗は使うのに注意が必要です。成分が文字式だったり、冪乗の指数が大きかったりすると計算時間や画面表示が大変な事になるかもしれません。使う前に、他に良い方法 (例えば対角化など) がないかどうか考えて見る方が良いでしょう。

1.9. **行列の転置.** ここからは、行列に関する操作とか、行列に関係した諸量の計算のうち、**maxima** で関数が用意されているものの中で基本的なものを取り上げていくことにしましょう。まずは、行列の転置から始めます。**maxima** で行列の転置を求めるには、関数 **transpose** を使います。

```
(%i4) A;
(%o4) matrix([2,3,4],[-1,0,3],[2,1,-2])
(%i5) transpose(A);
(%o5) matrix([2,-1,2],[3,0,1],[4,3,-2])
(%i6) C : matrix([a,b,c],[d,e,f]);
(%o6) matrix([a,b,c],[d,e,f])
(%i7) transpose(C);
(%o7) matrix([a,d],[b,e],[c,f])
```

1.10. **階段行列.** 与えられた行列 A に対して、それに対し行基本変形を施すことによって得られる階段行列を求めるには、**maxima** の関数 **echelon** を使います。ただし、階段行列の定義はテキストごとに少し違うかも知れませんので注意が必要です。

```
(%i8) kill(ALL);
(%o0) DONE
(%i1) A : matrix([1,2,3],[4,5,6],[7,8,9]);
(%o1) matrix([1,2,3],[4,5,6],[7,8,9])
(%i2) echelon(A);
(%o2) matrix([1,2,3],[0,1,2],[0,0,0])
(%i3) B : matrix([1,2,3],[4,5,6],[7,8,0]);
(%o3) matrix([1,2,3],[4,5,6],[7,8,0])
(%i4) C : echelon(B);
(%o4) matrix([1,8/7,0],[0,1,14],[0,0,1])
```

しかし、**maxima** で求めた階段行列からテキストの定義に沿った階段行列を求めるのは容易でしょう。

1.11. **行列の階数.** 与えられた行列 A に対して、対応する階段行列 echelon(A) の階段の数を、A の階数 (rank) と言いましたが、**maxima** では直接階数を求める関数 **rank** が用意されています。

```
(%i5) rank(A);
(%o5) 2
(%i6) rank(B);
```

```
(%o6) 3
```

ここでは、上の行列 A, B の階数を求めています。

1.12. **逆行列.** 与えられた n 次正方行列 A の階数が n ならば、A は逆行列 A^{-1} をもつのですが、**maxima** で逆行列を求めるには、関数 **invert** を用います。

```
(%i7) invert(B);
```

$$(\%o7)\quad \begin{pmatrix} -\frac{16}{9} & \frac{8}{9} & -\frac{1}{9} \\ \frac{14}{9} & -\frac{7}{9} & \frac{2}{9} \\ -\frac{1}{9} & \frac{2}{9} & -\frac{1}{9} \end{pmatrix}$$

しかし、階数が次数より小さいとき、正則でない行列に対して **invert** を適用すると次のようなエラーメッセージがでます。

```
(%i8) invert(A);
Division by 0
-- an error.  Quitting.  To debug this try DEBUGMODE(TRUE);
```

これを避けるには、事前に **rank** を使って考えている正方行列が正則であるかどうか調べておいた方が良いでしょう。

正則行列を求めるには、別の方法もあります。すでに説明した行列の冪乗を使って、例えば **A^^**(-1) のようにすれば、A の逆行列が求まります。

```
(%i9) B^^(-1);
```

$$(\%o9)\quad \begin{pmatrix} -\frac{16}{9} & \frac{8}{9} & -\frac{1}{9} \\ \frac{14}{9} & -\frac{7}{9} & \frac{2}{9} \\ -\frac{1}{9} & \frac{2}{9} & -\frac{1}{9} \end{pmatrix}$$

1.13. **行列式.** 与えられた正方行列 A の行列式を求めるには、**maxima** の関数 **determinant** を使います。

```
(%i10) determinant(A);
(%o10) 0
(%i11) determinant(B);
(%o11) 27
(%i12) determinant(B^^(-1));
```

$$(\%o12)\quad \frac{1}{27}$$

2. 少しだけ発展した操作

2.1. **行列の行を操作する.** 与えられた行列から、ある行だけ取り出す、例えば第 i 行を取り出すには、**A[i]** とタイプするか、または **maxima** の関数 **row** を使って、**row(A,i)** とタイプします。

```
(%i13) kill(ALL);
(%o0) DONE
(%i1) A : matrix([a,b,c],[d,e,f],[g,h,i]);
```

$$(\%o1)\quad \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$$

```
(%i2) A[1];
```

$$(\%o2)\quad [a,b,c]$$

```
(%i3) A[2];
```

$$(\%o3)\quad [d,e,f]$$

```
(%i4) A[3];
```

$$(\%o4)\quad [g,h,i]$$

```
(%i5) row(A,1);
```

$$(\%o5)\quad \begin{pmatrix} a & b & c \end{pmatrix}$$

```
(%i6) row(A,2);
```

$$(\%o6)\quad \begin{pmatrix} d & e & f \end{pmatrix}$$

```
(%i7) row(A,3);
```

$$(\%o7)\quad \begin{pmatrix} g & h & i \end{pmatrix}$$

2 通りの操作 A[i] と row(A,i) によって得られる結果は似ていますが微妙に意味が異なります。例えば、上の例で A[1] の値である [a,b,c] というのは、a, b, c を並べて得られる**リスト**と呼ばれる型を持ちます。一方、row(A,1) の値である (a b c) は 1×3-行列の型を持ちます。

ここで、行列の行基本変形を扱ってみましょう。例えば、A の第 2 行を 2 倍したいときには次のようにします。

```
(%i8) A[2] : 2 * A[2];
```

$$(\%o8)\quad [2d, 2e, 2f]$$

```
(%i9) A;
```

$$(\%o9)\quad \begin{pmatrix} a & b & c \\ 2d & 2e & 2f \\ g & h & i \end{pmatrix}$$

確かに A の第 2 行が 2 倍されていることがわかりますね。A 自身を変えたくないときには、A のコピーに対して変形操作を適用すればよいでしょう。一方、row の方は上のような方法では使えません。A をもとに戻すつもりで、row(A,2) : (1/2) * row(A,2) とタイプしても、次のようなエラーメッセージを受けとるだけです。

```
(%i10) row(A,2) : (1/2) * row(A,2);
Improper value assignment:
ROW (A, 2) FALSEfalse
-- an error.  Quitting.  To debug this try DEBUGMODE(TRUE);
```

もとに戻すには

```
(%i11) A[2] : (1/2) * A[2];
```

$$(\%o11)\quad [d, e, f]$$

```
(%i12) A;
```

```
(%o12)
```
$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$$

とします。つぎに、A の第 1 行に第 2 行の 3 倍を加えてみましょう。そのためには、次のようにします。

```
(%i13) A[1] : A[1] + 3 * A[2];
(%o13) [3d + a, 3e + b, 3f + c]
(%i14) A;
(%o14)
```
$$\begin{pmatrix} 3d+a & 3e+b & 3f+c \\ d & e & f \\ g & h & i \end{pmatrix}$$

もとに戻すには、

```
(%i15) A[1] : A[1] - 3 * A[2];
(%o15) [a, b, c]
(%i16) A;
(%o16)
```
$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$$

とします。今度は、A の第 1 行と第 3 行を入れ換えてみます。この操作は、1 度ではできないので、リスト型のダミーの変数 B を使って、次のようにします。

```
(%i17) B : A[1];
(%o17) [a, b, c]
(%i18) A[1] : A[3];
(%o18) [g, h, i]
(%i19) A[3] : B;
(%o19) [a, b, c]
(%i20) A;
(%o20)
```
$$\begin{pmatrix} g & h & i \\ d & e & f \\ a & b & c \end{pmatrix}$$

2.2. **行列の列を操作する.** 与えられた行列から、ある行を取り出すには、例えば行列 A の第 i 行を取り出すには、transpose(transpose(A)[i]) または col(A[i]) とタイプします。

```
(%i21) kill(ALL);
(%o0) DONE
(%i1) A : matrix([r,s,t],[u,v,w],[x,y,z]);
(%o1)
```
$$\begin{pmatrix} r & s & t \\ u & v & w \\ x & y & z \end{pmatrix}$$

```
(%i2) transpose(transpose(A)[1]);
(%o2)
```
$$\begin{pmatrix} r \\ u \\ x \end{pmatrix}$$

```
(%i3) col(A,1);
(%o3)
```
$$\begin{pmatrix} r \\ u \\ x \end{pmatrix}$$

ここで、行の操作の所での話しと違い、transpose(transpose(A)[1]) と col(A,1) はともに 3×1-行列の型を持ちます。列ベクトルの基本変形も行ベクトルの操作と transpose を組み合わせることで実現可能なのですが、少しややこしくなりますので、ここでは割愛させていただきます。

2.3. **行列の成分の操作.** 与えられた行列 A の第 (i,j)-成分を取り出すには、A[i][j] とか、A[i,j] とタイプします。

```
(%i4)  kill(ALL);
(%o0)  DONE
(%i1)  A : matrix([1,2,3,4],[5,6,7,8],[9,10,11,12]);
(%o1)  ⎛ 1  2   3   4  ⎞
       ⎜ 5  6   7   8  ⎟
       ⎝ 9  10  11  12 ⎠
(%i2)  A[2][3];
(%o2)  7
(%i3)  A[2,3];
(%o3)  7
```

この例では、A の第 (2,3) 成分を取り出しています。逆に、A の第 (2,3) 成分に新しい値、例えば x を入れるにはどうしたらいいでしょうか? それには、A[2][3] : x とタイプすれば OK です。

```
(%i4)  A[2,3] : x;
(%o4)  x
(%i5)  A;
(%o5)  ⎛ 1  2   3   4  ⎞
       ⎜ 5  6   x   8  ⎟
       ⎝ 9  10  11  12 ⎠
```

今度は、A の第 (3,2)-成分を y に変えてみましょう。A[3,2] : y とタイプして下さい。

```
(%i6)  A[3,2] : y;
(%o6)  y
(%i7)  A;
(%o7)  ⎛ 1  2   3   4  ⎞
       ⎜ 5  6   x   8  ⎟
       ⎝ 9  y   11  12 ⎠
```

この方法でもうまく行くことがわかりました。

2.4. **連立一次方程式の係数行列.** **maxima** では、連立一次方程式を扱うこともできます。例えば、次の連立一次方程式

$$\begin{cases} x+2y+3z & =4, \\ 5x+6y+7z & =8, \\ 9x+10y+11z & =0 \end{cases}$$

を **maxima** に認識させて、eq1 と言う名前をつけるには、

```
(%i8)  kill(ALL);
(%o0)  DONE
(%i1)  eq1 : [x+2*y+3*z=4, 5*x+6*y+7*z=8, 9*x+10*y+11*z=12];
(%o1)  [3z+2y+x=4,7z+6y+5x=8,11z+10y+9x=12]
```

のように、方程式のリストとして入力します。ここで、eq1 の未知数 (x,y,z) に関する係数行列 A を取り出して見ます。係数行列を求める **maxima** の関数は、**coefmatrix** です。これは、次の例の様にして使いま

す。

```
(%i2) xx : [x,y,z];
(%o2) [x, y, z]
(%i3) A : coefmatrix(eq1, xx);
(%o3) ⎛ 1  2  3 ⎞
      ⎜ 5  6  7 ⎟
      ⎝ 9 10 11 ⎠
```

ここで xx は [x,y,z] のことです。直接

```
(%i3) A : coefmatrix([x+2*y+3*z=4, 5*x+6*y+7*z=8, 9*x+10*y+11*z=12],[x,y,x]);
```

と書いても同じことですが、タイプ量の節約のためにも、またタイプミスを減らすためにもうまく名前を使うことは有益です。行列で表現された連立一次方程式 $A\mathbf{x} = \mathbf{b}$ に対して、行列 $(A|\mathbf{b})$ のことを、拡大係数行列と言いましたが、**maxima** でも関数 **augcoefmatrix** によって、拡大係数行列を求めることができます。ただし、$(A|\mathbf{b})$ ではなく、$(A| - \mathbf{b})$ を求めるので、注意する必要があります。

```
(%i4) Ab : augcoefmatrix(eq1, xx);
(%o4) ⎛ 1  2  3  -4 ⎞
      ⎜ 5  6  7  -8 ⎟
      ⎝ 9 10 11 -12 ⎠
```

さて、連立一次方程式の解の様子を知るには、拡大係数行列を行基本変形で階段行列に変形して係数行列の階数 rank(A) と拡大係数行列の階数 rank(Ab) を比較してみればよかったですね。**maxima** でそれをする方法についてはすでに説明しました。早速やってみます。

```
(%i5) echelon(Ab);
(%o5) ⎛ 1  2  3  -4 ⎞
      ⎜ 0  1  2  -3 ⎟
      ⎝ 0  0  0   0 ⎠
(%i6) rank(A);
(%o6) 2
(%i7) rank(Ab);
(%o7) 2
```

得られた階段行列の形からもわかりますが、rank(A) = rank(Ab) = 2 なので、この連立一次方程式 eq1 は、未知数個数 n = 3 から rank(A) = 2 を引いただけの、すなわち 3 - 2 = 1 個の任意定数を含む無限個の解を持つことがわかります。

2.5. **連立一次方程式を解く.** 連立一次方程式が解を持つことがわかり、その解の様子が予想できたなら、今度は具体的に解を求めたくなりますね。解を求めることも、**maxima** なら簡単に実行できます。上の連立一次方程式 eq1 の解を、**maxima** の関数 **solve** で求めて見ましょう。

```
(%i8) solve(eq1, xx);
Dependent equations eliminated:  (3)
(%o8) [[x = %R1 - 2, y = 3 - 2%R1, z = %R1]]
```

ここで、%R1 と表示されているものが任意定数です。次の例の様に、解を持たない方程式をいきなり solve で解こうとすると、エラーメッセージを貰うことになるので、事前に解の様子を調べておいた方が良いでしょう。

```
(%i9) kill(ALL);
(%o0) DONE
```

```
(%i1) eq2 : [x + 2*y = 3, 4*x + 5*y = 6, 7*x + 8*y = 0];
(%o1) [2y + x = 3, 5y + 4x = 6, 8y + 7x = 0]
(%i2) xx : [x,y];
(%o2) [x, y]
(%i3) solve(eq2, xx);
Inconsistent equations:  (1)
-- an error.  Quitting.  To debug this try DEBUGMODE(TRUE);
```

実際、

```
(%i4) A : coefmatrix(eq2, xx);
(%o4) ⎛ 1  2 ⎞
      ⎜ 4  5 ⎟
      ⎝ 7  8 ⎠
(%i5) Ab : augcoefmatrix(eq2, xx);
      ⎛ 1  2  -3 ⎞
(%o5) ⎜ 4  5  -6 ⎟
      ⎝ 7  8   0 ⎠
(%i6) echelon(Ab);
      ⎛ 1  8/7   0  ⎞
(%o6) ⎜ 0   1   -14 ⎟
      ⎝ 0   0    1  ⎠
(%i7) rank(A);
(%o7) 2
(%i8) rank(Ab);
(%o8) 3
```

より、rank(A)≠rank(Ab) ですから、この連立一次方程式 eq2 は解をもちません。

3. もっと高度な機能について

maxima では、これまで述べてきたことに加えて、さらに色々なことができます。講義ですでに習ったことに関連する機能としては、例えば余因子行列を求めるとか、三角化するとかいったものがあります。また、ベクトルについても内積を求めたりすることなどができます。一次独立なベクトルの正規直交化といったことや、行列の固有値、固有ベクトルを求めるといったこともできます。この超入門編では, そう言った進んだ話題については取り上げることができませんでしたが、興味のあるかたは、**maxima** のマニュアルを読んだり、このページにに置いてある「行列計算における数式処理ソフト **maxima** の利用について」などを参考にして、より高度な利用にチャレンジしてみて下さい。

E-mail address: kenrou@yo.rim.or.jp