Professional Maxima

～あなたはまだマセ〇ティカですか～

はしがき

本書ははじめて Maxima を使う人を対象に、Maxima の基本的な使い方を解説しています。全三章で構成されており、各章の内容は次の通りです。

まず第 1 章は、主に Maxima の起動、終了の方法と、本書を読み進める上で欠くことのできない事柄について簡潔にまとめています。たった 6 ページしかありませんが、本書の核とも言えるきわめて重要な章です。

言うまでもなく、場当たり的な知識の詰め込みでは学習効果は上がりません。そこで第 2 章では、Maxima を使いこなす上で必要となる基礎知識を体系的に解説しています。

最後の第 3 章は、具体的な数学の問題を Maxima を使って解いています。「Maxima 実演コーナー」であると同時に、Maxima の機能や性質を把握するための例示にもなっています。

入門書を名乗る多くの書籍を読んだとき、最初の方は内容がスカスカで退屈至極で、最後の方になると今度は中身が濃すぎてほとんど理解できなかった、といった感想を持つことがほとんどです。これは、系統的解説に終始するあまり、知識の量に対しその融合としてもたらされる内容の困難さが指数関数的に増大することを執筆者が十分認識していないことが原因です。そこで本書は、あとで解説する事柄でも無理のない範囲で先んじて取り上げることで、内容の濃淡を少なくし、無理せず退屈せず読み進められるように工夫しました。執筆者の独りよがりとも思える知識のひけらかしや、「実は役に立たない応用例」などは排除していますので、読者を選ばない解説書になっていると思います。

本書には間違いがたくさんあると思いますが、ご利用の際は「利用者責任の原則」でお願いします。

目次

第 1 章

Maxima へようこそ

Maxima は、多項式や有理式、微分や積分、行列など様々な計算を実行することのできるアプリケーションソフトです。高額な値段で販売されている *Mathematica* や Maple と同種のソフトウェアですが、Maxima は GNU General Public License（GPL）を採用しており、「無料である」、「ソースコードが公開されている」、「(一定条件の下で）改変や再配布ができる」といった特徴があります。この章はたった 6 ページしかありませんが、この章を読み終えればすぐに Maxima を使い始めることができます。

1.1 Maxima の歴史

Maxima は、1968 年にマサチューセッツ工科大学（MIT）で開発が始まった Macsyma の子孫にあたります。Macsyma は 1982 年に DOE[*1] 版 と Symbolics 版に袂を分かち、それぞれ独自の進化を遂げていきます。Symbolics 版は高速・高機能な数式処理ソフトとして販売されましたが、後発の強みからか *Mathematica* や Maple に市場を奪われる格好となり、現在は機能拡張も行われず細々と販売が続けられている状況です[*2]。一方、DOE 版は、独自にソースコードを所有していた William Frederick Schelter 氏が GNU Common Lisp 環境へ移植するとともに、DOE と交渉し 1998 年には GPL で公開する許可を得るに至りました。これが Maxima の始まりです。

現在 Maxima は世界中のボランティアによって開発・保守が続けられており、成果物は公式サイト [1] で公開されています。

[*1] U.S. Department of Energy（アメリカエネルギー省）

[*2] 一旦、1992 年に Macsyma Inc. に買収されましたが、1999 年には Macsyma Inc. が Tenedos LLC に買収され、Symbolics 版 は再び Symbolics が販売するようになり、現在に至っています。ちなみに、Windows 版 Macsyma 2.4 の値段は 500 ドルです（2007 年 1 月現在）。なお、Macsyma の更に詳しい歴史については、例えば文献 [2] をご覧ください。

1.2　インストール

Maxima は、Mac OS X を含む（おそらく）全ての UNIX 系 OS と Windows へインストールすることができます。Windows の場合は公式サイトで配布されているバイナリ・パッケージをインストールするのが簡単です。万一インストールに躓いたり、あるいはインストール前に手順を知っておきたいなどの場合はウェブページ [5] をご覧ください。Mac OS X の場合、ソースコードからインストールすることも簡単に出来ますが、Intel Mac であればウェブページ [3] にバイナリ・パッケージを用意してあります。解凍して「アプリケーション」フォルダに入れるだけでインストールが完了します。PowerPC Mac を使っている場合など、ソースコードからインストールしたい場合はウェブページ [4] などを参考にして下さい。

なお、Maxima はプログラミング言語 Lisp（List Processor もしくは List Processing Language）で作成されていますが、ソースコードからインストールする場合は、様々な Lisp の中から好みのものを選ぶことができるかもしれません。無料の Lisp としては、移植性の高さが売りの CLISP、動作速度が売りの CMUCL、SBCL、Maxima と同じ GPL を採用している GCL 等が有名です。また、有料のものでは Franz Inc. の Allegro CL が有名です。

1.3　起動と終了

Maxima を起動するには、Windows の場合は「スタートメニュー」から Maxima を探し出し、「Command line Maxima」クリックします。また、UNIX 系 OS の場合は、ターミナル・アプリケーション[*3]で maxima コマンドを実行してください。

```
Maxima 5.14.0 http://maxima.sourceforge.net
Using Lisp GNU Common Lisp (GCL) GCL 2.6.8 (aka GCL)
Distributed under the GNU Public License. See the file COPYING.
Dedicated to the memory of William Schelter.
This is a development version of Maxima. The function bug_report()
provides bug reporting information.
(%i1)
```

Maxima を起動すると、入力行 `(%i1)` が現れますので、この後に続けて Maxima に実行させたい命令を入力します。試しに、$x^4 - y^4$ を因数分解させてみましょう。

[*3] コマンドプロンプトや kterm、 Terminal.app などのことをターミナル・アプリケーションと呼ぶことにします。

`factor(x^4 - y^4);`[*4] と入力し、Enter キーを押してみてください[*5]。

```
(%i1) factor(x^4 - y^4); Enter

                                        2    2
(%o1)                  - (y - x) (y + x) (y  + x )
```

計算結果 $-(y-x)(y+x)(y^2+x^2)$ が出力されました。この例のように、Maxima は計算結果を一種のアスキーアートで出力します。不定積分 $\int \frac{1}{\sqrt{x^2+1}}\,dx$ の場合は

```
(%i2) 'integrate(1/sqrt(x^2 + 1), x);[*6]

                              /
                              [      1
(%o2)                         I ------------ dx
                              ]       2
                              / sqrt(x  + 1)
```

と表示されます。美しさに欠ける反面、動作が軽快であるとか、テキスト文書にコピー・ペースト出来るなどの実用的なメリットもあります。

一種の GUI フロントエンドである TeXmacs や Emacs Lisp である imaxima を使うと、出力結果を美しく表示することができます。

Maxima Input/Output

```
(%i1) factor(x^4 - y^4);
(%o1)                        -(y - x)(y + x)(y^2 + x^2)
```

以下、本書では（説明のし易さを考慮し、）この「美しい方の出力」で Maxima の出力結果を表現することにします。

Maxima を終了するには `quit();` を実行します。また、無限ループに入ってしまった場合など、Maxima を強制終了したい場合は Ctrl + C キーを押します[*7]。

1.4 初めての Maxima

Maxima はセミコロン（;）またはドル記号（$）までを 1 つの命令と見なします。普通はセミコロン（;）を用いますが、出力結果を表示させたくない場合（出力する必要がない場合）はドル記号（$）を用います。

[*4] factor [fǽktər] 因数、約数、因数分解する。
[*5] 本書では、Enter キーを押すことを強調する場合のみ、記号 Enter を用いることにしています。
[*6] integrate [íntəgrèit] 積分する（⇄ differentiate）。sqrt = square root 平方根。
[*7] 環境によっては Ctrl + G で計算を中断することができるようです。

MAXIMA INPUT/OUTPUT

```
(%i1)  diff(cos(x), x);
(%o1)                                  − sin x
(%i2)  diff(cos(x), x)$
(%i3)
```

入力 (%i2) に対する出力行 (%o2) が表示されていないことに注目してください。

既に見ているように、Maxima では入出力に対し自動的に番号が振られて行きます。その番号を用いて、過去の入出力を参照することができます。

MAXIMA INPUT/OUTPUT

```
(%i3)  factor(x^2 - y^2);
(%o3)                              −(y − x)(y + x)
(%i4)  expand(%o3);
(%o4)                                  x^2 − y^2
```

この例では、因数分解を実行した後、その結果 $-(y-x)(y+x)$ を展開して元の式 x^2-y^2 に戻ることを確認しています。番号を付けずにパーセント記号（%）を用いると、直前の出力結果を参照することができます。従って、先の例は、

MAXIMA INPUT/OUTPUT

```
(%i3)  factor(x^2 - y^2);
(%o3)                              −(y − x)(y + x)
(%i4)  expand(%);
(%o4)                                  x^2 − y^2
```

と実行したことと同値です。

なお、Maxima はフリーフォーマットです。演算子、括弧等の間に半角スペースを入れたり、命令文が長い場合、途中で改行するなどして、見やすい書式で命令を記述することができます。

[*8] diff = differential [dìfərénʃ(ə)l] 微分、差（⇄ integral）。

[*9] expand [ikspǽnd, eks-] 展開する。

メモ

パーセント記号（%）は入出力参照の他、組み込み定数を表す場合にも用いられます。

命令	意　　味
%e	自然対数の底（Napier の数）：$e = \lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n \fallingdotseq 2.71828$
%gamma	Euler の定数：$\gamma = \lim_{n\to\infty}\left(\sum_{k=1}^{n}\frac{1}{k} - \log n\right) \fallingdotseq 0.57721$
%i	虚数単位：i
%phi	黄金比：$\phi = \frac{1+\sqrt{5}}{2} \fallingdotseq 1.61803$
%pi	円周率：$\pi \fallingdotseq 3.14159$

1.5 オンラインヘルプ

Maxima には非常に詳細なリファレンス・マニュアル maxima_toc.html 等が付属していますが、Maxima の画面からもその内容を見ることが出来ます。書式

??␣文字列 Enter

により実行すると、「文字列」を含む関数・変数の一覧が表示されます。ダブル・クエスチョンマーク（??）の後ろに半角スペースを入れることに注意しましょう。また、一般の Maxima への命令と違いセミコロン（;）もドル記号（$）も不要です。試しに、文字列 prime を検索してみましょう。

Maxima Input/Output

```
(%i1) ?? prime Enter
     0: mod_big_prime :(maxima.info)Definitions for zeilberger.
     1: next_prime :Definitions for Number Theory.
     2: prev_prime :Definitions for Number Theory.
     3: primep :Definitions for Number Theory.
     4: primep_number_of_tests :Definitions for Number Theory.
   Enter space-separated numbers, 'all' or 'none':
```

文字列 prime を含む関数・変数が 5 個ヒットしました。最後の行

```
Enter space-separated numbers, 'all' or 'none':
```

に注目してください。つまり、スペースで区切られた複数の番号（例えば「0␣3␣4」）を入力するか、all、もしくは none を入力せよ、と言って Maxima は入力待ち状態になっています。ここで、all を入力すると 0 から 4 までの全ての説明が出力され、none を入力すると何も出力せずにオンラインヘルプが終了します。試しに、3 を入力してみましょう。関数 primep[*10] の解説が表示されるはずです。

MAXIMA INPUT/OUTPUT

```
Enter space-separated numbers, 'all' or 'none': 3 Enter
 - Function: primep (<n>)
     Primality test. If 'primep (n)' returns 'false', <n> is
     a composite number and if it returns 'true', <n> is a
     prime number with very high probability.
（後略）

(%o1)                          false
```

出力の後半部分は省略しましたが、primep が素数テスト関数であること、そして、引数が合成数なら false を返し、高い確率で素数なら true を返すことなど、詳細な解説が出力されます。

以降の章で分からない命令があったら、迷わず「??␣文字列」を実行してください。必ず必要な情報が見つかるはずです。なお、関数や変数の名前が分かっている場合は、「?␣factor」のようにシングル・クエスチョンマーク「?␣名前」で調べることができます。

[*10] primep = prime + probability: prime [práim] 素の。probability [prὰbəbíləti] 確率、見込み。

第 2 章

Maxima の基本

Maxima には千を超える関数や変数が組み込まれています。その全てを使いこなしたいと考えるのはごく自然な発想ですが、実は、専門外の関数というのはあまり使う機会がありません。一生使わないかもしれません。

そこで、この章では、専門によらず利用される関数や変数のうち、Maxima を操作する上での「いろは」とも言える基本的なものについて、実行例とともに体系的に解説していくことにします。

2.1 代入する

Maxima では変数や文字列への代入にコロン（:）を用います。

MAXIMA INPUT/OUTPUT

```
(%i1) x: 5;
(%o1)                                5
(%i2) x^2;
(%o2)                               25
```

逆に、代入されている値を消去するには関数 `remvalue`[*1] を用います[*2]。

MAXIMA INPUT/OUTPUT

```
(%i3) a: x;
(%o3)                                5
(%i4) remvalue(x);
(%o4)                               [x]
(%i5) x^2;
```

[*1] remvalue = remove value

[*2] 変数そのものを削除する関数 `kill` で代用しても良いでしょう。

```
(%o5)                                  x^2
(%i6) a^2;
(%o6)                                   25
```

全ての変数（に代入された値）を消去するには `remvalue(all)` を実行します。ある変数に一時的に値等を代入したい場合は、関数 `ev`[*3] を用いるのが簡単です。次の例は、多項式 x^2+x-1 内の変数 x に $x=5$ を代入しています。変数 `x` や `a` が変化していないことに注目してください。

MAXIMA INPUT/OUTPUT

```
(%i7) a: x^2 + x - 1;
                                        2
(%o7)                                  x  + x - 1
(%i8) ev(a, x: 5);
(%o8)                                   29
(%i9) x;
(%o9)                                   x
(%i10) a;
                                        2
(%o10)                                 x  + x - 1
```

次のように `ev()` は省略することが出来ます。

MAXIMA INPUT/OUTPUT

```
(%i11) a, x: 5;
(%o11)                                  29
(%i12) a, x: 10;
(%o12)                                 109
```

関数 `ev`（もしくはそれを省略した命令）は非常に高機能で、本書でも今後様々な場面で登場するでしょう。これに対し、代入専用の関数 `ratsubst`[*4] も用意されています。「式」内の「変数」に「代入したいもの」を代入するには、

```
ratsubst(代入したいもの, 変数, 式);
```

により実行します。

[*3] ev = evaluate [ivǽljuèit] 評価する。

[*4] ratsubst = rational + substitute: subst = substitute [sʌ́bstət(j)ùːt] 代入する、置換する。

MAXIMA INPUT/OUTPUT

```
(%i13) ratsubst(5, x, a);
(%o13)                                29
```

関数 ratsubst は非常に強力で、例えば、式 $\cos^2 x - \cos x$ の $\cos^2 x$ に $1 - \sin^2 x$ を代入することなども出来ます。

MAXIMA INPUT/OUTPUT

```
(%i14) cos(x)^3 + cos(x)^2;
(%o14)                              cos^3 x + cos^2 x
(%i15) ratsubst(1 - sin(x)^2, cos(x)^2, %);
(%o15)                     - sin^2 x + cos x (1 - sin^2 x) + 1
```

変数に様々な値を代入して式の値を調べたい場合は、式を関数として定義しておくと良いでしょう。Maxima では記号 := で関数を定義します。

MAXIMA INPUT/OUTPUT

```
(%i16) f(x):= 3*x + 1;
(%o16)                              f(x) := 3 x + 1
(%i17) f(1);
(%o17)                                4
(%i18) f(2);
(%o18)                                7
```

多変数関数を定義するには、変数をカンマ（,）で区切って f(x, y, z) のようにします。また、変数の個数が可変の（一定でない）関数を定義する場合は、f([x]) のように引数をリストにします（29 ページおよび 34 ページの例を参照）[*5]。定義した関数 f(x) を消去する場合は関数 remfunction[*6] を用います[*7]。

MAXIMA INPUT/OUTPUT

```
(%i19) f(x);
(%o19)                              3 x + 1
(%i20) remfunction(f);
(%o20)                               [f]
(%i21) f(x);
```

[*5] 変数の個数は length(x) 調べられます。また、各変数へは x[1]、x[2] 等でアクセスします。

[*6] remfunction = remove function: function [fáŋ(k)ʃ(ə)n] 関数。

[*7] 関数 remvalue の場合と同様 kill コマンドで代用しても良いでしょう。

```
(%o21)                                  f(x)
```

ここで、引数が f(x) ではなく f であることに注意しましょう。

何らかの計算結果を関数として定義したいこともあるでしょう。例えば、対数関数 $\log x$ の導関数を $f(x)$ と置く場合を考えてみます。直前の結果を参照するパーセント記号（%）と二重単引用符（''）[*8]を用いて、

MAXIMA INPUT/OUTPUT

```
(%i22) diff(log(x), x);
(%o22)                                  1/x
(%i23) f(x):= ''%;
(%o23)                                f(x) := 1/x
```

とする方法もありますが[*9]、関数 define[*10] を使う方が鮮やかです。

MAXIMA INPUT/OUTPUT

```
(%i24) define(f(x), diff(log(x), x));
(%o24)                                f(x) := 1/x
```

なお、二重単引用符（''）は「評価せよ」という命令で、単引用符（'）は逆に「評価するな」という命令です。

複数の処理を関数として定義する場合、局所変数（ローカル変数）を利用したくなるかもしれません。そのような場合は関数 block を利用すると良いでしょう。これについては、後で実例を挙げます（17、29、34 ページを参照）。

2.2　リスト

Maxima は、四角括弧（[]）内に要素をカンマで区切って並べたものをリストと認識します。

MAXIMA INPUT/OUTPUT

```
(%i1) listp(L);
(%o1)                                  false
(%i2) L: [11, 12, 13, 14, 15];
```

[*8] 二重引用符（"）ではないことに注意。

[*9] 環境によっては二重単引用符（''）は不要なようです。

[*10] define [difáin] 定義する。

```
(%o2)                          [11, 12, 13, 14, 15]
(%i3) listp(L);
(%o3)                                  true
```

ここで、関数 `listp` は、引数がリストか否かによって `true/false` を返す関数です。リストの各要素は 1 番から順に番号が振られ、`L[1]`、`L[2]`、`L[3]` のようにして取り出すことができます。また、リストの長さは関数 `length` で調べることができます。

MAXIMA INPUT/OUTPUT

```
(%i4) L[1]+L[2];
(%o4)                                   23
(%i5) length(L);
(%o5)                                   5
(%i6) L[length(L)];
(%o6)                                   15
```

なお、最後の要素を取り出すためには専用の関数 `last` が用意されています。

MAXIMA INPUT/OUTPUT

```
(%i7) last(L);
(%o7)                                   15
```

リストに要素を追加するには、先頭に追加する関数 cons[*11] もしくは末尾に追加する関数 endcons[*12] を用います。

MAXIMA INPUT/OUTPUT

```
(%i8) cons(a, L);
(%o8)                        [a, 11, 12, 13, 14, 15]
(%i9) endcons(b, %);
(%o9)                        [a, 11, 12, 13, 14, 15, b]
```

逆に要素を削除するには関数 `delete` を用います。関数 `delete` は書式

```
delete(削除する要素, リスト);
```

もしくは

[*11] cons = construct [動 kənstrʌ́kt, 名 kɑ́nstrʌkt] 構成する。

[*12] endcons = end + construct

delete(削除する要素，リスト，削除する個数);

により実行します。「削除する個数」を指定しない場合、該当する全ての要素が削除され、指定した場合は、先頭から指定個数のみが削除されます。

MAXIMA INPUT/OUTPUT

```
(%i10) [a, 1, b, 1, c, 1, d, 1, e, 1];
(%o10)                    [a, 1, b, 1, c, 1, d, 1, e, 1]
(%i11) delete(1, %o10, 1);
(%o11)                     [a, b, 1, c, 1, d, 1, e, 1]
(%i12) delete(1, %o10, 2);
(%o12)                       [a, b, c, 1, d, 1, e, 1]
(%i13) delete(1, %o10);
(%o13)                             [a, b, c, d, e]
```

存在しない要素を削除しようとしてもエラーになることはありませんが、関数 member(要素，リスト) を使えば、「リスト」に「要素」が属しているか否か調べることができます。

MAXIMA INPUT/OUTPUT

```
(%i14) member(1, [1, 2, 3, 4, 5]);
(%o14)                                true
(%i15) member(1.0, [1, 2, 3, 4, 5]);
(%o15)                                false
(%i16) member("1", [1, 2, 3, 4, 5]);
(%o16)                                false
```

リスト [1, 2, 3, 4, 5] を考えるとき、整数 1 は当然このリストに属していますが、小数 1.0 や文字列としての 1 は属していないので、上記のような結果になっています。

リスト同士の連結には関数 append[*13] を用います。関数 append は書式

append(リスト$_1$，リスト$_2$，リスト$_3$，.....);

により実行します[*14]。

[*13] append [əpénd] 付け加える。

[*14] 2 つのリストを交互に連結する関数 join もあります。例えば、(%o10) のリストは join([a,b,c,d,e], [1,1,1,1,1]) で得られます。

MAXIMA INPUT/OUTPUT

```
(%i17) append([a,b,c], [1,2,3], [x,y,z]);
(%o17)                    [a, b, c, 1, 2, 3, x, y, z]
```

あるリストから部分リストを取り出すには関数 rest を用います。関数 rest は書式

rest(リスト, 整数);

で実行します[*15]。rest(リスト, 自然数) を実行した場合は先頭から、rest(リスト, −自然数) を実行した場合は末尾からそれぞれ「自然数」個を取り除いたリストを出力します。

MAXIMA INPUT/OUTPUT

```
(%i18) rest(%o17, 3);
(%o18)                    [1, 2, 3, x, y, z]
(%i19) rest(%o17, -3);
(%o19)                    [a, b, c, 1, 2, 3]
```

2.3 整数と多項式と有理式

整数と多項式は同じような性質を持っているため、Maxima にはどちらにも適用できる関数がたくさんあります。例えば、商と余りを求める関数 quotient、remainder およびその両方を同時に求める関数 divide などです。

MAXIMA INPUT/OUTPUT

```
(%i1) quotient(92, 7);
(%o1)                              13
(%i2) remainder(92, 7);
(%o2)                               1
(%i3) divide(92, 7);
(%o3)                          [13, 1]
(%i4) quotient(x^3 + x^2, x^2 + 1);
(%o4)                            x + 1
(%i5) remainder(x^3 + x^2, x^2 + 1);
(%o5)                           - x - 1
(%i6) divide(x^3 + x^2, x^2 + 1);
```

[*15] 「整数」を省略した場合は 1 が指定されたものと見なされます。

(%o6) $[x+1, -x-1]$

改めて説明するまでもなく、前半は 92 を 7 で割ったときの商 13 と余り 1 を求める計算で、後半は x^3+x^2 を x^2+1 で割ったときの商 $x+1$ と余り $-x-1$ を求める計算です。また、関数 factor は整数、多項式、有理式に共通に用いることができ、素因数分解及び因数分解を出力します[*16]。

MAXIMA INPUT/OUTPUT

```
(%i7) factor(300);
```
(%o7) $2^2 \times 3 \times 5^2$
```
(%i8) factor(x^3 + y^3);
```
(%o8) $(y+x)\left(y^2 - xy + x^2\right)$
```
(%i9) 1/(a^2 - 1) - b^2/(a^2 - 1);
```
(%o9) $\frac{1}{a^2-1} - \frac{b^2}{a^2-1}$
```
(%i10) factor(%);
```
(%o10) $-\frac{(b-1)(b+1)}{(a-1)(a+1)}$

なお、多項式や有理式に関数 factor を適用した場合、整数因数については標準では素因数分解しません。素因数分解したい場合は変数 factorflag に true を代入します。

MAXIMA INPUT/OUTPUT

```
(%i11) factor(9*x^2 + 72*x + 144);
```
(%o11) $9\,(x+4)^2$
```
(%i12) factor(9*x^2 + 72*x + 144), factorflag: true;
```
(%o12) $3^2\,(x+4)^2$

また、変数 dontfactor に変数リストを代入すると、その変数については因数分解しなくなります。つまり、因数分解「しない」変数を指定することができます。上の多項式の場合、

MAXIMA INPUT/OUTPUT

```
(%i13) factor(9*x^2 + 72*x + 144), dontfactor: [x];
```
(%o13) $9\left(x^2 + 8x + 16\right)$
```
(%i14) factor(9*x^2 + 72*x + 144), dontfactor: [x], factorflag: true;
```
(%o14) $3^2\left(x^2 + 8x + 16\right)$

[*16] より正確には「(素) 因数分解を求めようと努力した結果が出力される」と表現すべきでしょう。

のように、ちょうど係数の最大公約数で括った形になります。一般には、多変数の多項式や有理式に適用することが多いでしょう。

MAXIMA INPUT/OUTPUT

```
(%i15) factor(%o9), dontfactor: [a];
```

$$(\%o15) \qquad -\frac{(b-1)(b+1)}{a^2-1}$$

```
(%i16) factor(%o9), dontfactor: [b];
```

$$(\%o16) \qquad -\frac{b^2-1}{(a-1)(a+1)}$$

```
(%i17) factor(%o9), dontfactor: [a, b];
```

$$(\%o17) \qquad -\frac{b^2-1}{a^2-1}$$

```
(%i18) f: expand((x + 1)^2 * (y + 1)^2 * (z + 1)^2);
```

$$(\%o18) \quad x^2y^2z^2+2xy^2z^2+y^2z^2+2x^2yz^2+4xyz^2+2yz^2+x^2z^2+2xz^2+z^2 + 2x^2y^2z+4xy^2z+2y^2z+4x^2yz+8xyz+4yz+2x^2z + 4xz+2z+x^2y^2+2xy^2+y^2+2x^2y+4xy+2y+x^2+2x+1$$

```
(%i19) factor(f), dontfactor: [];
```

$$(\%o19) \qquad (x+1)^2(y+1)^2(z+1)^2$$

```
(%i20) factor(f), dontfactor: [x];
```

$$(\%o20) \qquad \left(x^2+2x+1\right)(y+1)^2(z+1)^2$$

```
(%i21) factor(f), dontfactor: [y];
```

$$(\%o21) \qquad (x+1)^2\left(y^2+2y+1\right)(z+1)^2$$

```
(%i22) factor(f), dontfactor: [z];
```

$$(\%o22) \qquad (x+1)^2(y+1)^2\left(z^2+2z+1\right)$$

```
(%i23) factor(f), dontfactor: [x, y];
```

$$(\%o23) \qquad \left(x^2+2x+1\right)\left(y^2+2y+1\right)(z+1)^2$$

```
(%i24) factor(f), dontfactor: [x, y, z];
```

$$(\%o24) \qquad \left(x^2+2x+1\right)\left(y^2+2y+1\right)\left(z^2+2z+1\right)$$

次に、主に多項式を扱う際に不可欠な関数を解説します。多変数多項式を 1 つの変数で整理したい場合は、関数 `factorout` もしくは `ratsimp` を使います。

MAXIMA INPUT/OUTPUT

```
(%i25) f: expand((a^2 - b^2)*x^2 - 2*(a + b)*x + 1);
```

$$(\%o25) \qquad -b^2x^2+a^2x^2-2bx-2ax+1$$

```
(%i26) factorout(f, a, b);
```

$$(\%o26) \qquad -(b-a)(b+a)x^2-2(b+a)x+1$$

```
(%i27) factorout(f, a, b), dontfactor: [a, b];
(%o27)                    (a^2 - b^2) x^2 - 2 (b + a) x + 1
(%i28) ratsimp(f, x);
(%o28)                    (a^2 - b^2) x^2 + (-2 b - 2 a) x + 1
```

関数 factorout は指定した変数（上例の場合は a と b）を係数とする多項式として整理する関数です。一方、関数 ratsimp は、展開、通分、約分によって、有理式を簡易化する関数です。

係数を取得するには、関数 ratcoef を用います。

MAXIMA INPUT/OUTPUT

```
(%i29) ratcoef(f, x, 0);
(%o29)                                1
(%i30) ratcoef(f, x, 1);
(%o30)                            -2 b - 2 a
(%i31) ratcoef(f, x, 2);
(%o31)                             a^2 - b^2
(%i32) ratcoef(f, x, 3);
(%o32)                                0
```

次数を取得するには関数 hipow を用いますが、少々注意が必要です。例えば、以下の実行結果はいずれも期待した値ではありません。

MAXIMA INPUT/OUTPUT

```
(%i33) hipow((x + 1)^5, x);
(%o33)                                1
(%i34) hipow(1/(x - 1) - x^3/(x - 1), x);
(%o34)                                3
(%i35) hipow(expand((x + 1)^5), x + 1);
(%o35)                                0
```

そこで、関数 hipow の基本書式を関数 rat[17] を合成した形

```
hipow(rat(式), 変数);
```

[17] 関数 rat は有理式を簡易化する関数ですが、四則演算と整数冪の場合しか簡易化を行わない点が関数 ratsimp との機能的な違いです。また、関数 rat は主に Maxima が内部で利用するための関数で、出力結果の内部表現が ratsimp によるものと異なります。

で覚えておくと良いでしょう。第 2 引数は $x+1$ や $2x$ のような「式」ではなく、単独の「変数」とします。

MAXIMA INPUT/OUTPUT

```
(%i36) hipow(rat((x + 1)^5), x);
(%o36)                                5
(%i37) hipow(rat(1/(x - 1) - x^3/(x - 1)), x);
(%o37)                                2
```

失敗例の 3 番目 (%i35) は x の次数を調べればよいので、第 2 引数を $x+1$ などとするのはナンセンスですが、多変数多項式の場合には $x+y$ や xy の次数を取得したくなるかもしれません。このような場合に備えて、独自に次数を取得する関数を定義しておくと良いでしょう。例えば、以下を初期設定ファイル maxima-init.mac に書いておけば、次数を求める関数 degree が使えるようになります。

```
1 degree(f, x):= block([q, i, j],
2     q: f,
3     for i: 0 while (q # 0) do (j: i, q: quotient(q, x)),
4     return(j)
5 );
```

関数 degree の実行例として、$(x+1)^4-(x-1)^4$ における x の次数と x^2+y^2 における $x+y$ の次数を計算してみると次のようになります。

MAXIMA INPUT/OUTPUT

```
(%i38) degree((x + 1)^4 - (x - 1)^4, x);
(%o38)                                3
(%i39) display(expand(part(%i38, 1)));
                      expand((x + 1)^4 - (x - 1)^4) = 8 x^3 + 8 x
(%o39)                               done
(%i40) degree(x^2 + y^2, x + y);
(%o40)                                2
(%i41) display(divide(part(%i40, 1), (x + y)^%));
                        divide(y^2 + x^2, (y + x)^2) = [1, -2 x y]
(%o41)                               done
```

(%i39) と (%i41) は検算です。ここで、part は式の一部を抽出する関数で、display

は「命令 = 結果」の形で出力する関数です。

有理式に関数 ratsimp を適用すると、全ての分数を十把一絡げに通分し、可能なら約分も実行しますが、式が複雑な場合、分母が同じもの同士を個別に考察したいことがあります。これを実現するのが、関数 combine です。

MAXIMA INPUT/OUTPUT

```
(%i42) f: a^2/x - 1/x + a^2/x^2 - 1/x^2;
```

$$\text{(\%o42)} \qquad \frac{a^2}{x} - \frac{1}{x} + \frac{a^2}{x^2} - \frac{1}{x^2}$$

```
(%i43) ratsimp(f);
```

$$\text{(\%o43)} \qquad \frac{\left(a^2-1\right)+a^2-1}{x^2}$$

```
(%i44) combine(f);
```

$$\text{(\%o44)} \qquad \frac{a^2-1}{x} + \frac{a^2-1}{x^2}$$

分数式の分母と分子はそれぞれ関数 ratdenom[*18] および ratnum[*19] で取得できます。名称から想像できるように、内部で関数 rat を実行し、結果として得られた分母と分子を帰す関数です[*20]。

MAXIMA INPUT/OUTPUT

```
(%i45) ratdenom(f);
```

$$\text{(\%o45)} \qquad x^2$$

```
(%i46) ratnum(f);
```

$$\text{(\%o46)} \qquad \left(a^2-1\right)+a^2-1$$

通分された分数式を分解したい場合は関数 ratexpand（もしくは expand）を適用します。

MAXIMA INPUT/OUTPUT

```
(%i47) ratexpand(%o43);
```

$$\text{(\%o47)} \qquad \frac{a^2}{x} - \frac{1}{x} + \frac{a^2}{x^2} - \frac{1}{x^2}$$

関数 partfrac を用いれば、部分分数への分解もできます。

[*18] ratdenom = rational + denominator: denominator [diάmənèitə(r)] 分母（⇄ numerator）。

[*19] ratnum = rational + numerator: numerator [n(j)úːmərèitə(r)] 分子（⇄ denominator）。

[*20] 何らかの事情で約分される前の分母・分子を知りたい場合は、関数名に rat が付かない expand や combine 等を使って上手く整理した後、関数 denom および num を用います。

MAXIMA INPUT/OUTPUT

```
(%i48) 1/(x^2 - 1);
```

$$(\%o48)\qquad \frac{1}{x^2-1}$$

```
(%i49) partfrac(%);
```

$$(\%o49)\qquad \frac{1}{2\,(x-1)}-\frac{1}{2\,(x+1)}$$

```
(%i50) 1/(x^3 + 1);
```

$$(\%o50)\qquad \frac{1}{x^3+1}$$

```
(%i51) partfrac(%);
```

$$(\%o51)\qquad \frac{1}{3\,(x+1)}-\frac{x-2}{3\,(x^2-x+1)}$$

2.4 対数関数（logarithmic function）

微分・積分では自然対数は底 e を省略して $\log x$ と書きますが、Maxima でも `log(x)` は自然対数を表します。

MAXIMA INPUT/OUTPUT

```
(%i1) log(%e);
```

$$(\%o1)\qquad 1$$

底が 2 や 10 の対数を扱いたい場合は、適宜 `log2(x):= log(x)/log(2)` のように関数を定義して用いることになるでしょう[*21]。

対数には縮約形と展開形の 2 種類の標準的な表示方法があります。

縮約形	展開形
$\log xy$	$\log x+\log y$
$\log\dfrac{x}{y}$	$\log x-\log y$
$\log x^a$	$a\log x$

どちらの表示方法が望ましいかは解決すべき問題に依りますから、縮約形と展開形との間の相互変換の方法を知っておく必要があります。

始めに、縮約形を展開形に変換する方法を解説します。縮約→展開変換には関数 `radcan`[*22] を用います。

[*21] 底が 10 の対数 `log10` については、log10.mac を読み込むことで利用できるようになります。

[*22] radcan = radical + canonical form: radical [rǽdikl] 根号。canonical [lənánikl] 標準的な。

MAXIMA INPUT/OUTPUT

```
(%i2) log(x^a);
(%o2)                                  a log x
(%i3) radcan(log(x*y));
(%o3)                              log x + log y
(%i4) radcan(log(x/y));
(%o4)                              log x - log y
```

最初の $\log x^a$ だけ radcan は不要ですが、これはデフォルトで変数 logexpand[*23] に true が代入されているためです。super を代入すると残りの 2 例も radcan が不要になります[*24]。

MAXIMA INPUT/OUTPUT

```
(%i5) logexpand: super;
(%o5)                                  super
(%i6) log(x*y);
(%o6)                              log y + log x
(%i7) log(x/y);
(%o7)                              log x - log y
```

しかし、一般にはこのような変形が自動的に実行されるのは望ましいことではありませんから、むしろ false を代入して、勝手に展開形に変形されないように設定しておくことをお勧めします[*25]。

次に、展開形を縮約形に変換する方法を解説します。展開→縮約変換には関数 logcontract[*26] を用います。

MAXIMA INPUT/OUTPUT

```
(%i8) logcontract(log(x) + log(y));
(%o8)                                  log x y
(%i9) logcontract(log(x) - log(y));
(%o9)                                  log x/y
(%i10) logcontract(2*log(x));
```

[*23] logexpand = logarithm + expand

[*24] 変数 logexpand に super を代入してあっても、例えば $\log(x^2-1)$ のように因数分解が必要な場合は radcan を実行しない限り展開形にはなりません。

[*25] マニュアルには「変数 logexpand に all を代入すると、$\log xy$ は展開されるが $\log x/y$ は展開されない」とありますが、Maxima-5.14.0 では super を代入した場合と同じ結果になってしまうようです。

[*26] logcontract = logarithm + contract: contract [名 kántrækt, 動 kəntrǽkt] 縮小する、縮約する。

```
(%o10)                                log x^2
(%i11) logcontract(n*log(x));
(%o11)                                n log x
(%i12) logcontract(1/2*log(x));
(%o12)                                log x / 2
```

最後の 2 つ、$n \log x$ と $\frac{1}{2} \log x$ は縮約されませんでしたが、これは、係数が「整数、もしくは整数という性質が割り当てられた変数」の場合に限り縮約するように関数 `logcontract` が実装されているためです。試しに、変数 n に「整数という性質」を割り当てて、関数 `logcontract` を適用してみると、

Maxima Input/Output

```
(%i13) declare(n, integer);
(%o13)                                done
(%i14) logcontract(n*log(x));
(%o14)                                log x^n
```

ちゃんと縮約されました。なお、現在割り当てられている性質を確認するには、関数 `properties` を用い、割り当てを解除するには関数 `kill` を用います[*27]。

Maxima Input/Output

```
(%i15) properties(n);
(%o15)                  [database info, kind(n, integer)]
(%i16) kill(n);
(%o16)                                done
(%i17) properties(n);
(%o17)                                []
(%i18) logcontract(n*log(x));
(%o18)                                n log x
```

関数 `logcontract` によって縮約する係数の条件は、変数 `logconcoeffp`[*28] によって変更できます。先ほど縮約されなかった $\frac{1}{2} \log x$ が縮約されるようにするには、まず、「有理数、もしくは有理数という性質が割り当てられた変数」に対して `true` を返す関数を定義します。

[*27] 関数 `kill` は性質だけでなく、変数そのものを削除してしまう命令です。

[*28] logconcoeffp = logarithm + contract + coefficient + probability

MAXIMA INPUT/OUTPUT

```
(%i19) rationalp(r):= ratnump(r) or featurep(r, rational);
(%o19)          rationalp(r) := ratnump(r) or featurep(r, rational)
(%i20) rationalp(2);
(%o20)                                  true
(%i21) rationalp(1/2);
(%o21)                                  true
```

ここで定義した関数 rationalp を変数 logconcoeffp に代入すると、「有理数、もしくは有理数という性質が割り当てられた変数」を係数とする対数が縮約されるようになります。

MAXIMA INPUT/OUTPUT

```
(%i22) logconcoeffp: 'rationalp;
(%o22)                                rationalp
(%i23) logcontract(1/2*log(x));
(%o23)                                log sqrt(x)
(%i24) logcontract(1/3*log(x));
(%o24)                                log x^(1/3)
```

元に戻すには変数 logconcoeffp に false を代入します[*29]。

2.5　三角関数（trigonometric function）

Maxima には三角関数 sin、cos、tan とその逆三角関数 asin、acos、atan が組み込まれており、当然のことながら、$\sin\frac{\pi}{3}$ の値などを知っています。

MAXIMA INPUT/OUTPUT

```
(%i1) sin(%pi/3);
(%o1)                                 sqrt(3)/2
(%i2) asin(1);
(%o2)                                  %pi/2
(%i3)  cos(acos(x));
(%o3)                                   x
(%i4) acos(cos(x));
```

[*29] 「整数、もしくは整数という性質が割り当てられた変数」に対して true を返す関数 integerp(n) or featurep(n, integer) を 変数 logconcoeffp に代入したことと同等です。

```
(%o4)                                  x
```

数学的には、$\arccos(\cos x) = x$ が成り立つのは（主値の取り方によりますが、普通は）$0 \leqq x \leqq \pi$ の場合に限りますから、`(%o4)` は少々注意が必要です。勝手にこのような簡易化が行われては困る場合には、変数 `triginverses`[*30] を `true` または `false` に設定します（デフォルトは `all`）。

MAXIMA INPUT/OUTPUT

```
(%i5) triginverses: true;
(%o5)                                 true
(%i6) cos(acos(x));
(%o6)                                  x
(%i7) acos(cos(x));
(%o7)                              acos(cos x)
```

変数 `triginverses` に `false` を代入すると、`cos(acos(x))` も簡易化されなくなります。

Maxima は、現在ではほとんど使われなくなった割三角関数 `sec`、`csc`、`cot` を出力することがありますので、これらの定義を覚えておきましょう[*31]：

$$\sec\theta = \frac{1}{\cos\theta}, \quad \csc\theta = \frac{1}{\sin\theta}, \quad \cot\theta = \frac{1}{\tan\theta}$$

さて、三角関数の重要な公式を挙げよ、と言われて真っ先に思い浮かべるのは加法定理ではないでしょうか。それを実行する関数が `trigexpand`[*32] です。

MAXIMA INPUT/OUTPUT

```
(%i8) trigexpand(tan(x + y));
                                  tan y + tan x
(%o8)                           ---------------
                                1 - tan x tan y
(%i9) trigexpand(sin(x + %pi/3));
                                  sin x   sqrt(3) cos x
(%o9)                             ----- + -------------
                                    2          2
```

なお、次の例を見れば分かるように、関数 `trigexpand` は（標準では）一段階しか展開しませんので、必要に応じて繰り返し適用します。

[*30] triginverses = trigonometric + inverse: trigonometric [trìg(ə)nəmétrik] function 三角関数。inverse [invə́ː(r)s, ínvəː(r)s] 逆（の）、反対（の）。

[*31] `sec` : secant（正割）、`csc` : cosecant（余割）、`cot` : cotangent（余接）

[*32] trigexpand = trigonometric + expand: trigonometric [trìg(ə)nəmétrik] function 三角関数。expand [ikspǽnd, eks-] 展開する。

MAXIMA INPUT/OUTPUT

```
(%i10) trigexpand(cos(2*x + %pi/3));
```

$$(\%o10) \qquad \frac{\cos 2x}{2} - \frac{\sqrt{3}\,\sin 2x}{2}$$

```
(%i11) trigexpand(%);
```

$$(\%o11) \qquad \frac{\cos^2 x - \sin^2 x}{2} - \sqrt{3}\,\cos x\,\sin x$$

実は、trigexpand は変数としても用いられ、true に設定すると、一気に展開されます。

MAXIMA INPUT/OUTPUT

```
(%i12) trigexpand(cos(2*x + %pi/3)), trigexpand: true;
```

$$(\%o12) \qquad \frac{\cos^2 x - \sin^2 x}{2} - \sqrt{3}\,\cos x\,\sin x$$

この他、加法・減法（$\theta \pm \frac{\pi}{4}$ など）の展開を抑止する変数 trigexpandplus と積（3θ など）の展開を抑止する変数 trigexpandtimes があります。

展開の逆変換に相当する関数が trigreduce[*33] で、次数を下げる方向に変形します。

MAXIMA INPUT/OUTPUT

```
(%i13) trigreduce(sin(x)^3);
```

$$(\%o13) \qquad \frac{3\,\sin x - \sin 3x}{4}$$

また、三角関数の式を簡易化する trigrat[*34] もよく使います。関数 trigrat は、初期位相が 0 になるように（例えば、$\sin(x + \frac{\pi}{6})$ は $\frac{\sqrt{3}\,\sin x + \cos x}{2}$ に）変形し、かつ、次数を下げる方向に変形することで、与えられた式を可能な限り線形和の形に帰着させます。まるで trigexpand と trigreduce の間の子のような関数です。

MAXIMA INPUT/OUTPUT

```
(%i14) trigrat(sin(2*x + %pi/6)^2);
```

$$(\%o14) \qquad \frac{\sqrt{3}\,\sin 4x - \cos 4x + 2}{2}$$

比較のため、同じ式 $\sin^2\left(2x + \frac{\pi}{6}\right)$ に trigexpand と trigreduce を適用してみましょう。

[*33] trigreduce = trigonometric + reduce: reduce [rid(j)úːs] 縮小する。

[*34] trigrat = trigonometric + rational: rational [rǽʃənl] 有理の。

MAXIMA INPUT/OUTPUT

```
(%i15) trigexpand(sin(2*x + %pi/6)^2);
```

$$(\%o15)\qquad \left(\frac{\sqrt{3}\,\sin 2\,x}{2}+\frac{\cos 2\,x}{2}\right)^2$$

```
(%i16) trigexpand(%);
```

$$(\%o16)\qquad \left(\frac{\cos^2 x-\sin^2 x}{2}+\sqrt{3}\,\cos x\,\sin x\right)^2$$

```
(%i17) trigreduce(sin(2*x + %pi/6)^2);
```

$$(\%o17)\qquad \frac{1-\cos\left\{2\left(2\,x+\frac{\pi}{6}\right)\right\}}{2}$$

```
(%i18) trigreduce(%);
```

$$(\%o18)\qquad \frac{1-\cos\left(4\,x+\frac{\pi}{3}\right)}{2}$$

関数 `trigrat` は、$\tan x$ を $\dfrac{\sin x}{\cos x}$ に変形するときにも利用できます。関数 `trigreduce` で元に戻ります。

MAXIMA INPUT/OUTPUT

```
(%i19) trigrat(tan(x));
```

$$(\%o19)\qquad \frac{\sin x}{\cos x}$$

```
(%i20) trigreduce(%);
```

$$(\%o20)\qquad \tan x$$

ただし、$\tan^n x$ 等に `trigrat` を適用してしまうと、すぐには $\tan^n x$ には戻ってこれません（戻る必要はないかもしれませんが)。

MAXIMA INPUT/OUTPUT

```
(%i21) trigrat(tan(x)^3);
```

$$(\%o21)\qquad -\frac{\sin 3\,x-3\,\sin x}{\cos 3\,x+3\,\cos x}$$

```
(%i22) trigreduce(%);
```

$$(\%o22)\qquad \frac{3\,\sin x}{\cos 3\,x+3\,\cos x}-\frac{\sin 3\,x}{\cos 3\,x+3\,\cos x}$$

一端展開して、関数 `trigsimp`[*35] で簡単にしてから、`trigreduce` を実行すると元に戻ります。

[*35] trigsimp = trigonometric + simplify: simplify [símpləfài] 簡単にする。

MAXIMA INPUT/OUTPUT

```
(%i23) trigexpand(%);
```

$$\text{(\%o23)}\quad \frac{3\sin x}{-3\cos x\sin^2 x+\cos^3 x+3\cos x}-\frac{3\cos^2 x\sin x-\sin^3 x}{-3\cos x\sin^2 x+\cos^3 x+3\cos x}$$

```
(%i24) trigsimp(%);
```

$$\text{(\%o24)}\quad \frac{\sin^3 x}{\cos^3 x}$$

```
(%i25) trigreduce(%);
```

$$\text{(\%o25)}\quad \tan^3 x$$

内部でどのような処理が行われるか考えながら計算させなければならない、Maxima はなんと教育的なソフトウェアでしょう。なお、関数 `trigsimp` は、関係式 $\sin^2 x+\cos^2 x=1$ を用いて式を簡単にする関数です。

最後に、半角の公式を利用する方法を解説しておきます。Maxima に半角の公式を使って式変形させるためには、関数 `trigexpand` や `trigreduce` を実行する際に、変数 `halfangles`[*36] を `true` に設定します（デフォルトは `false`）。

MAXIMA INPUT/OUTPUT

```
(%i26) trigexpand(sin(x/2)^2), halfangles: true;
```

$$\text{(\%o26)}\quad \frac{1-\cos x}{2}$$

```
(%i27) trigreduce(cos(x/2)^2), halfangles: true;
```

$$\text{(\%o27)}\quad \frac{\cos x+1}{2}$$

なお、変数 `halfangles` をグローバルに `true` に設定しておけば、以降の計算全てに亘って半角の公式が適用されますが、それはあまりお勧めできません。次の例を見てください。

MAXIMA INPUT/OUTPUT

```
(%i28) trigreduce(sin(x/2)), halfangles: true;
```

$$\text{(\%o28)}\quad \frac{\sqrt{1-\cos x}}{\sqrt{2}}$$

正しくは $\sin\frac{x}{2}=\pm\frac{\sqrt{1-\cos x}}{\sqrt{2}}$ ですね。Maxima は ± を付けてくれません。更に悩ましい次の例はどうでしょう。

[*36] halfangles = half angle: angle [ǽŋgl] 角、角度。

MAXIMA INPUT/OUTPUT

```
(%i29) trigexpand(tan(x/2)), halfangles: true;
(%o29)                          (1 − cos x)/sin x
(%i30) trigreduce(tan(x/2)), halfangles: true;
(%o30)                          (1 − cos x) csc x
```

ここで、$\csc x = \dfrac{1}{\sin x}$ でしたから、`(%o29)` と `(%o30)` は全く同じ式です。つまりMaxima は等式

$$\tan\frac{x}{2} = \frac{1-\cos x}{\sin x} \qquad [2.1]$$

が成立すると主張していますが、この式に $x = 0$ を代入しようとしてみると.....、何となく不自然ですね。式 [2.1] は $\tan\dfrac{x}{2} = \dfrac{\sin\dfrac{x}{2}}{\cos\dfrac{x}{2}}$ の分母・分子に $\sin\dfrac{x}{2}$ を掛けることで導けます。

$$\tan\frac{x}{2} = \frac{\sin\dfrac{x}{2}}{\cos\dfrac{x}{2}} = \frac{\sin^2\dfrac{x}{2}}{\sin\dfrac{x}{2}\cdot\cos\dfrac{x}{2}} = \frac{\dfrac{1-\cos x}{2}}{\dfrac{\sin x}{2}} = \frac{1-\cos x}{\sin x}$$

つまり、0 になるかもしれない $\sin\dfrac{x}{2}$ を掛けたことが問題で、本来は $\cos\dfrac{x}{2}$ を掛けるべきです[*37]。このとき、

$$\tan\frac{x}{2} = \frac{\sin\dfrac{x}{2}}{\cos\dfrac{x}{2}} = \frac{\sin\dfrac{x}{2}\cdot\cos\dfrac{x}{2}}{\cos^2\dfrac{x}{2}} = \frac{\dfrac{\sin x}{2}}{\dfrac{1+\cos x}{2}} = \frac{\sin x}{1+\cos x} \qquad [2.2]$$

となります。

[*37] $\cos x/2 = 0$ のときは $\tan x/2$ が定義されないので、$\cos x/2 \neq 0$ と考えて良い。

メモ

三角関数用に用意されている Maxima の関数（trigexpand、trigreduce、trigsimp 等）や変数（trigexpand、triginverses 等）は双曲線関数にも適用できます。

双曲線関数		Maxima の命令
双曲線正弦関数	$\sinh x = \frac{e^x - e^{-x}}{2}$	sinh(x)
双曲線余弦関数	$\cosh x = \frac{e^x + e^{-x}}{2}$	cosh(x)
双曲線正接関数	$\tanh x = \frac{\sinh x}{\cosh x}$	tanh(x)
双曲線余割関数	$\mathrm{csch}\, x = \frac{1}{\sinh x}$	csch(x)
双曲線正割関数	$\mathrm{sech}\, x = \frac{1}{\cosh x}$	sech(x)
双曲線余接関数	$\coth x = \frac{1}{\tanh x}$	tanh(x)

これらの逆関数は、順に asinh、acosh、atanh、acsch、asech、acoth です。

2.6　繰り返しと条件分岐

ほとんどのプログラミング言語では for 文によって繰り返し処理が実装されていますが、その for 文の説明に使われる定番の例題として、「1 から 10 までの自然数の総和を求めるプログラムを作成せよ」というのがあります。もちろん、公式 $\frac{10 \times (10+1)}{2}$ を使うのは反則ですが、残念ながら（？）Maxima にはそのような計算を実行するための関数 sum[*38] が組み込まれています。関数 sum は

```
sum(式, 変数, 開始値, 終了値);
```

により実行します。今の問題の場合、

MAXIMA INPUT/OUTPUT

```
(%i1)  sum(i, i, 1, 10);
(%o1)                                55
```

[*38] sum = summation [sʌméiʃ(ə)n] 合計、総和。

のように簡単に求まってしまいます。逆数の和 $\frac{1}{1}+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{10}$ だって簡単です。

MAXIMA INPUT/OUTPUT

```
(%i2) sum(1/i, i, 1, 10);
                                  7381
(%o2)                             ----
                                  2520
```

これでは何の練習にもなりませんので、このような関数の存在は忘れてください。

さて、Maxima における for 文の書式は

```
for 変数 in リスト do 処理;
```

もしくは

```
for 変数: 開始値 オプション 終了条件 do 処理;
```

です[*39]。便宜上前者を「リスト書式」、後者を「汎用書式」と呼ぶことにします。なお、複数の処理を実行したい場合は「do (処理$_1$, 処理$_2$, ...)」のように、丸括弧内にカンマで区切って処理を並べます。

リスト書式では、「リスト」の要素が先頭から順に「変数」に代入され、「処理」が実行されます。次の例の場合、変数 i が順に e、v、o、l と変化しながら、処理「s: concat(i, s)」が実行されます。ここで、concat は要素を連結する関数です。

MAXIMA INPUT/OUTPUT

```
(%i3) s: "";
(%o3)
(%i4) for i in ["E", "V", "O", "L"] do s: concat(i, s);
(%o4)                                done
(%i5) s;
(%o5)                               LOVE
```

(%i3) で用意した空の変数 s にリストの要素 E、V、O、L がこの順に連結していきますから、s は内部で E、VE、OVE、LOVE と変化しています。もう一つ簡単な例を挙げてみます。

MAXIMA INPUT/OUTPUT

```
(%i6) f([x]):= block([s], s: 0, for i in x do s: s + i, return(s))$
(%i7) f(7);
```

[*39] 「変数: 開始値」の代わりに「変数 from 開始値」を使うこともできます。

```
(%o7)                                7
(%i8)  f(7, 2);
(%o8)                                9
(%i9)  f(7, 2, 10);
(%o9)                               19
```

これは、引数の総和を求める関数です。引数をリスト x として受け取り、リスト x の各要素が順に変数 i に代入されて、処理「s: s + i」が実行される仕組みです。CSV（Comma Separated Values）形式のファイルを読み込んでデータを利用する場合などにはこのリスト書式が有効でしょう。

一方、汎用書式は（文字通り）様々な用途に使います。まず、もっとも素朴な例として、最初に挙げた「1 から 10 までの総和」を求めるプログラムを作ってみましょう。

MAXIMA INPUT/OUTPUT

```
(%i10)  s: 0;
(%o10)                               0
(%i11)  for i: 1 thru 10 do s: s + i;[*40]
(%o11)                            done
(%i12)  s;
(%o12)                               55
```

「オプション」は省略し、「終了条件」に変数の終了値「thru 10」を設定しました。なお、「オプション」を省略すると変数を 1 ずつ増やす設定「step 1」になります[*41]。すなわち、(%i11) は「for i: 1 step 1 thru 10 do s: s + i;」と同値です。終了条件には終了値を設定する thru の他、while と unless が利用できます。このプログラムの場合、「step 1」を「while i <= 10」や「unless i > 10」に書き代えることができます。

MAXIMA INPUT/OUTPUT

```
(%i13)  s: 0$ for i: 1 while i <= 10 do s: s + i$ s;
(%o15)                               55
(%i16)  s: 0$ for i: 1 unless i > 10 do s: s + i$ s;
(%o18)                               55
```

[*40] thru（米略式）= through [θru:, θrú:]

[*41] 実は変数の開始値も省略可能で、省略した場合、1 が開始値になります。

別のプログラム例を挙げましょう。余りを求める関数 remainder[*42] を利用して、「100! が 5 で何回割れるか」求めてみます。

MAXIMA INPUT/OUTPUT

```
(%i19) for i: 0 while remainder(100!, 5^i) = 0 do ans: i$ ans;
(%o20)                                24
```

このプログラムは、100! が 5^i で割り切れなくなったときに for ループを抜け、そのときの i の値を出力しています。なお、変数 i は for ループを抜けた段階で初期化されてしまうため[*43]、別の（グローバル）変数 ans に代入しておき、最後にその値を出力するという手順を踏んでいます。念のため実行結果 24 が求める値であることを確認しておきましょう。

MAXIMA INPUT/OUTPUT

```
(%i21) remainder(100!, 5^24);
(%o21)                                0
(%i22) remainder(100!, 5^25);
(%o22)                      238418579101562500
```

確かに 100! は 5^{24} では割り切れ、かつ 5^{25} では割り切れませんでした[*44]。

ここで取り上げた 2 例はいずれも変数を 1 ずつ増やせば良かったため「オプション」は不要でしたが、「オプション」には step による増減設定の他、next による漸化式設定も可能です。例えば、変数を 0, 2, 4, 6, ... と偶数のみをとるように変化させるためには、「for i: 0 step 2 .....」あるいは「for i: 0 next i + 2 .....」とすればよいでしょう。もっとも「通なプログラマ」なら「オプション」など使わないかもしれません。

本節のもう一つのテーマである条件分岐に話を移しましょう。条件分岐には if 文を使います。Maxima における if 文の書式は（他の多くのプログラミング言語とほぼ同じですが)

```
if 分岐条件 then 真の場合の処理 else 偽の場合の処理;
```

です。for 文の場合と同様 (処理$_1$, 処理$_2$, ...) のようにして、複数の処理を実行することも可能です。分岐条件には表 2.2 にまとめた真偽を返す関数の他、次の演算子および関数が利用できます。

[*42] remainder [riméində(r)] 余り。

[*43] すなわち、変数 i はローカル変数です。

[*44] このことは、100 までの自然数の中に、5 の倍数が 20 個あり、25 の倍数が 4 個あり、125 の倍数が 0 個あることから暗算でも確認できます。

表 2.1 分岐条件に用いられる演算子等

真となる条件	記号	タイプ
左辺が右辺より小さい	<	中置子
左辺が右辺以下	<=	中置子
左辺と右辺が等しい	=	中置子
左辺と右辺が等しくない	#	中置子
左辺が右辺以上	>=	中置子
左辺が右辺より大きい	>	中置子
左辺と右辺の両方真	and	中置子
左辺と右辺の一方真	or	中置子
偽	not	前置子
2 つの変数が等しい	equal	2 変数関数
2 つの変数が等しくない	notequal	2 変数関数

if 文を用いた例として、小数第 1 位を四捨五入する関数 round を作ってみます[*45]。ガウス記号 $[x]$ に相当する関数 `fix`[*46] を利用します。ガウス記号 $[x]$ とは、実数 x を超えない最大の整数のことです。言い換えると、正の数に対しては切り捨て、負の数に対しては繰り下げる関数です。

MAXIMA INPUT/OUTPUT

```
(%i23)  fix(7.9);
(%o23)                                7
(%i24)  fix(-7.1);
(%o24)                               -8
```

従って、四捨五入する関数は

MAXIMA INPUT/OUTPUT

```
(%i25)  round(x):= if x >= 0 then fix(x + 0.5) else -fix(-x + 0.5)$
```

と定義すればよいはずです[*47]。

[*45] 実は、Lisp には「四捨五入に近い命令」round が組み込まれていて、Maxima からは ?round で実行できます。オンラインヘルプの場合と異なり、? と round の間に半角スペースが入らないことに注目してください。

[*46] ガウス記号に相当する関数 は `fix` の他 `entier` もあります。

[*47] 「0.5 を加えて切り捨てる」のは四捨五入を実現する際の常套手段です。

表 2.2 真偽を返す関数（主なもの）

関数	真となる条件
numberp	数値（%e、%i、%pi、sqrt などを除く）
constantp	定数（%e、%i、%pi、sqrt などを含む）
integerp	整数
ratnump	有理数
floatnump	多倍長でない小数
bfloatp	多倍長小数
evenp	偶数
oddp	奇数
primep	素数（34155071728321 より大きい数については確率的判定）
matrixp	行列
listp	リスト
setp	集合
emptyp	リストや集合が空
member	リストや集合に属する

MAXIMA INPUT/OUTPUT

```
(%i26) round(7.4);
(%o26)                          7
(%i27) round(7.5);
(%o27)                          8
(%i28) round(-7.4);
(%o28)                         -7
(%i29) round(-7.5);
(%o29)                         -8
```

なお、改めて述べるまでもないことですが、ここで作成した関数 round はあくまでサンプルです。「実際に使える関数」にするためには、関数 fix が数値（numberp(x) が true となる x）以外では機能しない点に注意し、実現すべき機能を十分に検討する必要があります。例えば、純粋に（分数と多倍長小数を除く）小数と整数だけに適用したいのであれば、次のように修正することになるでしょう。

MAXIMA INPUT/OUTPUT

```
(%i30) round(x):= if integerp(x) or floatnump(x)
then (if x >= 0 then fix(x + 0.5) else -fix(-x + 0.5)) else false$
```

[*48]

一方、無理数 $\sqrt{2}$ や円周率 π も四捨五入したいのであれば、関数 constantp で定数でない場合を除外した上で、関数 float を併用することになるでしょう。

もう一例 if 文を使ったサンプルを挙げておきます。Maxima は数式処理ソフトですから、有理数（$\frac{1}{3}$ など）も無理数（$\sqrt{2}$ など）も超越数（π など）も正しく処理しますが、時には近似値を知りたくなることもあります。これらを標準の小数（による近似値）に変換するには関数 float を用いますが、より精度の高い多倍長小数（による近似値）に変換するには関数 bfloat を用い、精度は変数 fpprec に設定します（デフォルトは 16）。

MAXIMA INPUT/OUTPUT

```
(%i31) bfloat(%pi);
(%o31)                    3.14159265358979 3b0
(%i32) bfloat(%pi), fpprec:50;
(%o32)   3.1415926535897932384626433832795028841971693993751b0
```

最初の実行例は 16 桁の精度で、一方 2 番目の実行例は 50 桁の精度で円周率 π の近似値を出力しています。if 文を使ったサンプルとして、「数」と「精度」を引数にとる関数 evalf[*49] を定義してみます。

```
1  evalf([x]) := block([fpprec],
2      if length(x) = 1 then
3          bfloat(x[1])
4      else
5          (fpprec:x[2], bfloat(x[1]))
6  );
```

関数 evalf は書式 evalf(数, 精度) で用いますが、精度を省略すると、現在設定されている値（変更していない場合はデフォルトの 16）が使われます。試しに Napier の数 e の近似値を出力してみましょう。

MAXIMA INPUT/OUTPUT

```
(%i33) evalf(%e);
```

[*48] else false は省略可能です。

[*49] evalf = evaluate + floating point: Maple で使われている関数名を拝借しました。

```
(%o33)                          2.7182818284590452b0
(%i34) evalf(%e, 50);
(%o34)      2.7182818284590452353602874713526624977572470937b0
```

最後に、表 2.1 の演算子及び関数について 2 点ほど補足しておきます。まず、表 2.2 の関数と違い、表 2.1 の演算子や関数はそれそのものが `true/false` を返すわけではわけではありません。`if` の条件文として用いる場合は不要ですが、明示的に `true/false` を出力するためには、関数 `maybe`（もしくは `is`）に食わせることになります。

MAXIMA INPUT/OUTPUT

```
(%i35) 3 < %e;
(%o35)                                3 < e
(%i36) maybe(%);
(%o36)                               false
(%i37) 3 >= %pi;
(%o37)                               3 ≧ π
(%i38) maybe(not 3 >= %pi);
(%o38)                                true
```

もう 1 点は、表 2.1 の中の `=/#` と `equal/notequal` の違いです。前者は Maxima の内部表現としての同異であるのに対し、後者は数学的な同異です。以下、いくつか具体例を挙げておきます。まず、等式 $\sqrt{3+2\sqrt{2}} = 1+\sqrt{2}$ が成立するか Maxima に聞いてみると次のような結果になります。

MAXIMA INPUT/OUTPUT

```
(%i39) maybe(sqrt(3 + 2*sqrt(2)) = 1 + sqrt(2));
(%o39)                               false
(%i40) maybe(equal(sqrt(3 + 2*sqrt(2)), 1 + sqrt(2)));
(%o40)                                true
```

また、多項式 $x^2+2x+1=(x+1)^2$ や有理式 $1+\frac{1}{x}=\frac{x+1}{x}$ の場合も同様です。

MAXIMA INPUT/OUTPUT

```
(%i41) maybe(x^2 + 2*x + 1 = (x + 1)^2);
(%o41)                               false
(%i42) maybe(equal(x^2 + 2*x + 1, (x + 1)^2));
(%o42)                                true
```

```
(%i43) maybe(1 + 1/x = (x + 1)/x);
(%o43)                              false
(%i44) maybe(equal(1 + 1/x, (x + 1)/x));
(%o44)                               true
```

関数 assume によって同一視した場合、equal では true になりますが、= では false になります。

MAXIMA INPUT/OUTPUT

```
(%i45) assume(equal(a, b));
(%o45)                           [equal(a, b)]
(%i46) maybe(a = b);
(%o46)                              false
(%i47) maybe(equal(a, b));
(%o47)                               true
```

2.7　ファイル操作

Maxima でファイルを読み込むには、基本的に関数 load もしくは batch のいずれかを使います[*50]。関数 load が黙って読み込むのに対して、関数 batch は記述された内容を逐一実行し、結果を画面に出力していきます。簡単な実験のため、1 行目に 3*4 + 5;、2 行目に 2^10 - 1; と記述したファイル test.mac を作成し、ホームフォルダ C:¥Documents and Settings¥User Name に入れてください[*51]。

```
1  3*4 + 5;
2  2^10 - 1;
3
```

このファイル test.mac を Maxima に読み込んでみます。UNIX 系 OS の場合はもちろん Windows でもフォルダの区切り記号には円（¥）ではなく、半角スラッシュ（/）を使わなければならないことに注意が必要です。

MAXIMA INPUT/OUTPUT

```
(%i1) load("C:/Documents and Settings/User Name/test.mac");
(%o1)              C:/Documents and Settings/User Name/test.mac
```

[*50] この他にもいくつかの関数があります。

[*51] Windows 向けの記述ですが、Mac OS X などの場合もほぼ同じですので、適宜読み替えてください。

```
(%i2) batch("C:/Documents and Settings/User Name/test.mac");
batching #pC:/Documents and Settings/User Name/test.mac
(%i3)                          5 + 3 4*52
(%o3)                             17
(%i4)                          2^10 - 1
(%o4)                            1023
```

関数 batch を実行した場合と違い、関数 load を実行した場合は、内部でいくつ命令を実行しても、それらがカウントされないことにも注目すべきでしょう。

さて、読み込むファイル名をフルパスで入力しなければならないのは苦痛です。実は、カレントフォルダ及び変数 file_search_maxima もしくは maxima_userdir に設定されているフォルダに入っているファイルについてはファイル名のみで読み込むことができます[*53]。カレントフォルダは多くの場合ホームフォルダですが、念のため確認しておきましょう。外部コマンドを実行するための関数 system を使い、UNIX 系 OS の場合は pwd コマンドを、Windows の場合は cd コマンドを実行します。

MAXIMA INPUT/OUTPUT

```
(%i5) system("cd");
C:¥Documents and Settings¥User Name
(%o5)                             0
```

カレントフォルダが（案の定）C:¥Documents and Settings¥User Name と分かりましたから、先ほどの test.mac はファイル名のみで読み込めるはずです。試してみましょう。

MAXIMA INPUT/OUTPUT

```
(%i6) batch("test.mac");
batching #pC:/Documents and Settings/User Name/test.mac
(%i7)                            5 + 3 4
(%o7)                             17
(%i8)                          2^10 - 1
(%o8)                            1023
```

変数 file_search_maxima には公式パッケージをファイル名のみで読み込めるように、はじめからたくさんのフォルダが設定されていますが、そのフォルダリストは 変数 file_search_maxima を実行することで確認できます。

[*52] 3 と 4 の間にかけ算記号 × は付かないようです。

[*53] この他、変数 file_search_lisp に設定されているフォルダに入っている場合もパスは不要です。

MAXIMA INPUT/OUTPUT

```
(%i9)  file_search_maxima;
(%o9)  [C:/Documents and Settings/User Name/maxima/###.{mac,mc},
       C:/PROGRA~1/MAXIMA~1/share/maxima/5.14.0/share/###.{mac,mc},
       C:/PROGRA~1/MAXIMA~1/share/maxima/5.14.0/share/{affine,algebr
       a,algebra/charsets,algebra/solver,calculus,combinatorics,cont
       rib,contrib/boolsimp,contrib/descriptive,contrib/diffequation
       s,contrib/diffequations/tests,contrib/distrib,contrib/dynamic
       s,contrib/ezunits,contrib/format,contrib/gentran,contrib/gent
       ran/test,contrib/Grobner,contrib/lurkmathml,contrib/maximaMat
       hML,contrib/mcclim,contrib/numericalio,contrib/pdiff,contrib/
       prim,contrib/rand,contrib/sarag,contrib/simplex,contrib/simpl
       ex/Tests,contrib/solve_rec,contrib/state,contrib/stats,contri
       b/stringproc,contrib/unit,contrib/Zeilberger,diffequations,lb
       fgs,linearalgebra,integequations,integration,macro,matrix,mis
       c,numeric,orthopoly,physics,simplification,sym,tensor,tensor/
       tests,trigonometry,utils,vector}/###.{mac,mc}]
```

この変数 file_search_maxima に「ファイルの在処」を追加すれば、ファイル名のみで読み込めるようになります。例として、フォルダ C:¥Temp を追加してみましょう。フォルダリストへの追加には関数 append を用います。

MAXIMA INPUT/OUTPUT

```
(%i10)  file_search_maxima: append(file_search_maxima,
["C:/Temp/###.{mac,lisp}"]);
(%o10)  [C:/Documents and Settings/User Name/maxima/###.{mac,mc},
        (中略)
        ,utils,vector}/###.{mac,mc},
        C:/Temp/###.{mac,lisp}]
```

これでフォルダ C:¥Temp に入っているファイルもファイル名のみで読み込めるようになったはずです。なお、上記のように設定すると、パス名だけでなく、拡張子も（それが .mac もしくは .lisp であれば）省略可能になります。先ほどの test.mac を C:¥Temp に移動して、実験してみてください。

MAXIMA INPUT/OUTPUT

```
(%i11)  file_search("test");
```

```
(%o11)                    C:/Temp/test.mac
```

変数 maxima_userdir に設定されたフォルダに入っているファイルもファイル名のみで読み込むことができますが、この変数は Maxima 起動後に変更すべきものではありません。必要なら環境変数 MAXIMA_USERDIR に設定してください。

以上でファイルの読み込みについては Maxima を自在に操れるようになりましたが、常に利用したい関数や設定がある場合、再起動の度に読み込み直さなければならないのは不便でしょう。そういった場合は初期設定ファイルを利用します。Maxima はホームフォルダに置かれた maxima-init.mac という名前のファイルを初期設定ファイルと認識し、起動時に自動的に読み込みます[*54]。

次に、Maxima による計算結果等をファイルに保存する方法について解説します。ファイルへ保存する（保存したくなる）情報は、変数や関数、入力行や出力行といった Maxima が保持している情報（保持しておくべき情報）と標準出力の内容の 2 種類に分けることができます。Maxima が保持している情報をファイルへ保存するには関数 stringout もしくは save を用います。

関数 stringout の書式とその意味を表にまとめると、次のようになります。

書　　式	保存されるもの
stringout("ファイル名", ○, □, △, ...);	○、□、△、.....
stringout("ファイル名", [番号$_1$, 番号$_2$]);	(%i 番号$_1$)〜(%i 番号$_2$)
stringout("ファイル名", input);	全ての入力
stringout("ファイル名", functions);	全ての（ユーザ）関数
stringout("ファイル名", values);	全ての（ユーザ）変数

3 番目 stringout("ファイル名", input) は、とりあえず全ての入力行をファイルに保存しておき、適宜エディタで取捨選択して再利用する、といって用途に使います。出力行を保存したい場合などは 1 番目の書式を使います。

MAXIMA INPUT/OUTPUT

```
(%i12) stringout("test.out", %o5, f(x), a);
(%o12)              C:/Documents and Settings/User Name/test.out
```

この場合、ファイル test.out には 3 つの要素、出力行 (%o5)、関数 f(x)、変数 a が記録されます。なお、関数 f(x) や変数 a に何も代入されていない場合は、文字列「f(x);」および「a;」が記録されます。つまり、記録されるのは画面上で f(x); や a; を実行し

[*54] 何らかの事情で読み込まれない場合は、Maxima の src フォルダに入れておけばよいでしょう。

た結果です[*55]。

なお、標準状態では関数 stringout は上書きモードで実行されますから、同じファイル名を指定すると、先に保存してある内容は消去されてしまいます。追記モードに変更するには、変数 file_output_append に true を代入します。

一方、関数 save は関数 stringout の高機能版とも言える関数で、入力行だけでなく出力行も含めて全てを丸ごとファイルに保存することなどができます。ただし、保存されたデータは（テキスト形式ではありますが）独自フォーマットであり、エディタで開いてもその内容を正確に把握するのは困難です（雰囲気は分かりますが）。従って、関数 save は、作業を一旦中断しあとで続きから実行したい、といった用途に利用することになるでしょう。ファイルを読み込む関数 loadfile と実行内容を振り返る関数 playback を併用します。

MAXIMA INPUT/OUTPUT

```
（延々と続く作業）
(%i13)  save("today.out", all);
(%o13)                                today.out
```

ファイル today.out には (%o13) までの作業全てが保存されます。ここで、一旦 Maxima を終了して、Maxima を再起動します。

MAXIMA INPUT/OUTPUT

```
(%i1)  loadfile("today.out");
(%o1)                                 today.out
(%i14)  playback();

（today.out に保存されていた (%i1) から (%o13) までが、ずら〜っと表示されます）

(%o14)                                  done
```

標準出力の内容をファイルに保存するには、関数 with_stdout を用います。次の例を見てください。

MAXIMA INPUT/OUTPUT

```
(%i15)  load("mactex-utilities.lisp")$[*56]。
(%i16)  tex(diff(log(x), x));
$$\frac{1}{x}$$
```

[*55] フォーマットは異なるかもしれません。

[*56] デフォルトでは plain TEX 流の出力になってしまうため、mactex-utilities.lisp を load しています。

```
(%o16)                                false
```

関数 tex は引数（式やラベル）を LaTeX コマンドで表現したものを標準出力（画面）に、そして戻り値「false」を出力行にそれぞれで出力します。このような場合、先の関数 stringout を用いると、「ファイル」には文字列「false」が保存され、関数 with_stdout を用いると、「$$\frac{1}{x}$$」が保存されます。

命　　令	保存されるもの
stringout("ファイル名", tex(diff(log(x), x)));	false
with_stdout("ファイル名", tex(diff(log(x), x)));	$$\frac{1}{x}$$

引数に入力行番号を与える場合は、with_stdout("test.out", ''%i16); のように頭に二重単引用符を付けます。なお、複数の引数をカンマで繋いで列挙すれば、一度に複数の内容を「ファイル」に保存することも出来ます。

関数 tex の他、関数 display、disp、print などにより出力される文字列をファイルに保存したい場合も関数 with_stdout を用います。

また、関数 writefile と closefile と組み合わせて利用すれば、作業状況を「画面に表示されたそのままの状態で」保存することができます。

MAXIMA INPUT/OUTPUT

```
(%i17)  writefile("work.out");
Starts dribbling to work.out (2317/12/24, 0:0:0).
(%o17)                                false

(いろいろな作業)

(%i18)  closefile();
Finished dribbling to work.out.
(%o18)                                false
```

保存される範囲

このとき、ファイル work.out には「Starts ...」から「closefile();」までが記録されます。上記「いろいろな作業」の部分で関数 playback(); を実行すれば、画面に表示されたものを全て丸ごとそのままの形で普通のテキストデータとして保存することができます。なお、関数 writefile は常にファイルを上書きします。追記する場合は代わりに関数 appendfile を用います。

第 3 章

Maxima を俯瞰する

この章では基本的な数学の問題を Maxima を使って解いてみることによって、Maxima を概観します。どのような問題を「基本的」と呼ぶかは主観に依存しますが、なるべく多くの人にとって「基本的」であることを目指し、センター試験、微分積分、行列の問題を取り上げています。

3.1 センター試験

本節の例題は全てセンター試験からの引用です。ただし、問題の全文ではなく部分的な引用です。また、空欄は［アイ］の代わりに［ア］とするなど簡素化しました。

【例題】 2 次方程式 $x^2 - 3x - 1 = 0$ の解が α、β で、$\alpha > \beta$ とするとき、

$$\alpha = \boxed{\text{ア}}, \quad \beta = \boxed{\text{イ}}$$

である。また、

$$m < \alpha < m + 1 \text{ を満たす整数 } m \text{ の値は } m = \boxed{\text{ウ}}$$
$$n < \beta < n + 1 \text{ を満たす整数 } n \text{ の値は } n = \boxed{\text{エ}}$$

である。次に、$\alpha + \dfrac{1}{\alpha} = \boxed{\text{オ}}$ であり、$\alpha^3 + \dfrac{1}{\alpha^3} = \boxed{\text{カ}}$ である。（平成 18 年度）

【解答】 前半は単純に 2 次方程式を解くだけです。

MAXIMA INPUT/OUTPUT

```
(%i1) x^2 - 3*x - 1 = 0;
(%o1)                          x^2 - 3 x - 1 = 0
```

```
(%i2) s: solve(%, x);
```
[*1]

$$(\%o2)\qquad \left[x = -\frac{\sqrt{13}-3}{2},\ x = \frac{\sqrt{13}+3}{2}\right]$$

```
(%i3) float(s);
```
[*2]

$$(\%o3)\qquad [x = -0.30277563773199,\ x = 3.302775637731995]$$

従って、「ア」は $\frac{\sqrt{13}+3}{2}$、「イ」は $-\frac{\sqrt{13}-3}{2}$、「ウ」は 3、「エ」は -1 と求まりました。

ここで求めた $\alpha = \frac{\sqrt{13}+3}{2}$（リスト s の 2 番目の要素）を用いて後半部分を計算します。

MAXIMA INPUT/OUTPUT

```
(%i4) ratsimp(s[2] + 1/s[2], algebraic: true);
```

$$(\%o4)\qquad \frac{x^2+1}{x} = \sqrt{13}$$

```
(%i5) ratsimp(s[2]^3 + 1/s[2]^3, algebraic: true);
```

$$(\%o5)\qquad \frac{x^6+1}{x^3} = 10\sqrt{13}$$

より、「オ」は $\sqrt{13}$、「カ」は $10\sqrt{13}$ と求まりました。なお、上の計算で、有理式を簡単にする関数 `ratsimp`[*3] を使っていますが、その際、有理化を実行するために変数 `algebraic`[*4] に `true` を代入しています。

【例題】 p、q、r を実数とし、x についての整式 A、B を

$$A = x^3 + p\,x^2 + q\,x + r,\quad B = x^2 - 3\,x + 2$$

とする。

(a) A を B で割ったときの商が $x-1$ であった。このとき、$p =$ [ア] である。

(b) A を B で割ったときの余りが x で割り切れた。このとき、$r =$ [イ] $p +$ [ウ] である。

(c) A を B で割ったとき、その商と余りが等しくなった。このとき、$q + r =$ [エ] である。

（平成 15 年度）

【解答】 (a)　A を B で割ったときの商を求めます。

[*1] solve [sálv, sɔ́ːlv] 解く。
[*2] float [flóut]: floating point 浮動小数点。
[*3] ratsimp = rationally simplify: rational [rǽʃənli] 有理的に。simplify [símpləfài] 簡単にする。
[*4] algebraic [æ̀ldʒəbéiik] 代数的な。

MAXIMA INPUT/OUTPUT

```
(%i1) A: x^3 + p*x^2 + q*x + r;
(%o1)                          x^3 + p x^2 + q x + r
(%i2) B: x^2 - 3*x + 2;
(%o2)                              x^2 - 3 x + 2
(%i3) quotient(A, B);
(%o3)                                x + p + 3
```

これが $x-1$ に一致することから、

MAXIMA INPUT/OUTPUT

```
(%i4) solve(% = x - 1, p);
(%o4)                                [p = -4]
```

より、「ア」は -4 と求まりました。

(b)　A を B で割ったときの余りを求めます。

MAXIMA INPUT/OUTPUT

```
(%i5) remainder(A, B, x);
(%o5)                        (q + 3 p + 7) x + r - 2 p - 6
```

これが x で割り切れるのは定数項が 0 の場合ですから、

MAXIMA INPUT/OUTPUT

```
(%i6) solve(coeff(%, x, 0) = 0, [r, p]);
(%o6)                          [[r = 2 %r4 + 6, p = %r4]]
```

より、「イ」は 2、「ウ」は 6 と求まりました。

(c)　上で求めた商と余りを再利用すれば十分ですが、Maxima には商と余りを同時に求める関数 `divide`[*7] がありますので、改めて A を B で割ったときの商と余りを求め、それらが一致するような p、q、r を求めることにします。

MAXIMA INPUT/OUTPUT

```
(%i7) divide(A, B, x);
(%o7)                  [x + p + 3, (q + 3 p + 7) x + r - 2 p - 6]
```

[*5] quotient [kwóuʃ(ə)nt] 商。

[*6] remainder [riméində(r)] 余り。

[*7] divide [diváid] 割る（⇄ multiply）。

```
(%i8) %[2] - %[1];
(%o8)                    (q + 3 p + 7) x − x + r − 3 p − 9
(%i9) solve([coeff(%, x, 0) = 0, coeff(%, x, 1) = 0], [p, q, r]);
(%o9)              [[p = %r6, q = −3 %r6 − 6, r = 3%r6 + 9]]
(%i10) %[1][2] + %[1][3];
(%o10)                                r + q = 3
```

よって、「エ」は 3 と求まりました。

【例題】 数列 $\{a_n\}$ の初項から第 n 項までの和 $S_n = \sum_{k=1}^{n} a_k$ が

$$S_n = -n^2 + 24n \quad (n = 1, 2, 3, \cdots)$$

で与えられるものとする。このとき $a_1 =$ ア、$a_2 =$ イ である。また、$a_n < 0$ となる自然数 n の範囲は $n \geqq$ ウ であり、

$$\sum_{k=1}^{40} |a_k| = \boxed{\text{エ}}$$

となる。　（平成 17 年度）

【解答】 まず、和 S_n と一般項 a_n の関係

$$a_n = \begin{cases} S_1 & (n = 1) \\ S_n - S_{n-1} & (n > 1) \end{cases}$$

を使って、一般項 a_n を求めてしまいます。

MAXIMA INPUT/OUTPUT

```
(%i1) S[n]:= -n^2 + 24*n;
(%o1)                              S_n := −n^2 + 24 n
(%i2) a[n]:= S[n] - S[n-1];
(%o2)                              a_n := S_n − S_{n−1}
```

和 S_n の式に形式的に $n = 0$ を代入すると、$S_0 = 0$ となりますから、(%o2) は $n = 1$ の場合も成り立ちます。従って、

MAXIMA INPUT/OUTPUT

```
(%i3) a[1];
```

```
(%o3)                                23
(%i4) a[2];
(%o4)                                21
```

より、「ア」は 23、「イ」は 21 と求まりました。また、

MAXIMA INPUT/OUTPUT

```
(%i5) sum(abs(a[n]), n, 1, 40);
(%o5)                                928
```

より、「エ」は 928 と求まりました。なお、「ウ」は「エ」を求めるための布石と思われますから、もはや解く必要はありませんが、例えば次のようにすれば求められます。

MAXIMA INPUT/OUTPUT

```
(%i6) expand(a[n]);
(%o6)                              25 - 2n
(%i7) for n: 1 while(a[n] >= 0) do display(a[n]);
                                   a_1 = 23
                                   a_2 = 21
                                   a_3 = 19
                                  (中　略)
                                   a_11 = 3
                                   a_12 = 1
(%o7)                               done
```

(%o6) より a_n が単調減少と分かりますから、初項から順に a_n が正である限り出力し続けてみました。これにより、$a_n < 0$ となるのが第 13 項以降、つまり、「ウ」が 13 と求まりました。

【例題】 次の資料は 2 科目の小テストに関する 5 人の生徒の得点を記録したものである。2 科目の小テストの得点をそれぞれ変量 x、y とする。

生徒番号	1	2	3	4	5
x	3	4	5	4	4
y	7	9	10	8	6

[*8] sum = summation [sʌméiʃ(ə)n] 合計、総和。abs = absolute [ǽbsəl(j)ùːt] value 絶対値。

(1)　変量 x の分散を小数で求めると、［ア］となる。
(2)　変量 y を使って新しい変量 t を $t = y -$［イ］で定めると、変量 t の平均は 0 になる。
(3)　変量 x と変量 y の相関係数を r とし、その 2 乗を r^2 で表すと、$r^2 =$［ウ］となる。
（平成 18 年度）

【解答】 (1)　まず平均 $\bar{x} = \frac{1}{5}\sum_{i=1}^{5} x_i$ を求め、それを用いて分散 $\sigma_x = \frac{1}{5}\sum_{i=1}^{5}(x_i - \bar{x})^2$ を求めることになります。

MAXIMA INPUT/OUTPUT

```
(%i1) x: [3, 4, 5, 4, 4];
(%o1)                          [3, 4, 5, 4, 4]
(%i2) sum(x[i], i, 1, 5)/5;
(%o2)                                  4
(%i3) sum((x[i] - 4)^2, i, 1, 5)/5;
                                       2
(%o3)                                  -
                                       5
(%i4) float(%);
(%o4)                                 0.4
```

より、「ア」は 0.4 と求まりました。
(2)　変量 t の平均が 0 になるのですから、「イ」に当てはまるのは変量 y の平均です。なお、あとの問題で必要になりそうですから、分散も計算しておきます。

MAXIMA INPUT/OUTPUT

```
(%i5) y: [7, 9, 10, 8, 6];
(%o5)                          [7, 9, 10, 8, 6]
(%i6) sum(y[i], i, 1, 5)/5;
(%o6)                                  8
(%i7) sum((y[i] - 8)^2, i, 1, 5)/5;
(%o7)                                  2
```

より、「イ」は 8 と求まりました。
(3)　求める値は相関係数の 2 乗ですから、共分散の 2 乗を x、y の分散の積 $\frac{2}{5}\times 2$ で割ればよいでしょう。

MAXIMA INPUT/OUTPUT

```
(%i8) sum((x[i] - 4)*(y[i] - 8), i, 1, 5)/5;
                                  3
(%o8)                             -
                                  5
(%i9) %^2/(2/5*2);
                                  9
(%o9)                            --
                                 20
```

従って、「ウ」は $\dfrac{9}{20}$ と求まりました。

【例題】 複素数 z_0、z_1 を $z_0 = (\sqrt{3}+i)(\cos\theta + i\sin\theta)$、$z_1 = \dfrac{4\{(1-\sin\theta)+i\cos\theta\}}{(1-\sin\theta)-i\cos\theta}$ とする。ただし、$0° < \theta < 90°$ とする。また、$\arg z$ は複素数 z の偏角を表すものとし、偏角は $-180°$ 以上 $180°$ 未満とする。

(1) $|z_0| =$ [ア]、$\arg z_0 =$ [イ]$°$ $+\theta$ である。

(2) z_1 の分母と分子に $(1-\sin\theta)+i\cos\theta$ をかけて計算すると $z_1 =$ [ウ]$(-\sin\theta + i\cos\theta)$ となる。よって、$|z_1| =$ [エ]、$\arg z_1 =$ [オ]$°$ $+\theta$ である。

(3) $\left|\dfrac{z_1}{z_0}\right| =$ [カ]、$\arg\dfrac{z_1}{z_0} =$ [キ]$°$ である。 （平成 15 年度）

【解答】 (1)　複素数 $w = \sqrt{3}+i$ の絶対値と偏角を求めます。

MAXIMA INPUT/OUTPUT

```
(%i1) w: sqrt(3) + %i;
(%o1)                          i + sqrt(3)
(%i2) polarform(%);
                                    %i pi
(%o2)                          2 %e
                                       6
```

ここで、`polarform`[*9] は極形式に変形する関数ですが、高校で学ぶ $r(\cos\theta + i\sin\theta)$ ではなく、指数形 $re^{i\theta}$ で出力します：

$$re^{i\theta} = r(\cos\theta + i\sin\theta)$$

従って、`(%o2)` は $2\left(\cos\dfrac{\pi}{6} + i\sin\dfrac{\pi}{6}\right)$ と同値ですから、$|w| = 2$、$\arg w = 30°$ が得られたことになります。よって、「ア」は 2、「イ」は 30 と求まりました。なお、Maxima には絶対値を求める関数 `abs` と偏角を求める関数 `carg` もありますので、これらを利用しても良いでしょう。

[*9] polarform = polar [póulə(r)] form 極形式。

MAXIMA INPUT/OUTPUT

```
(%i3)  abs(w);
(%o3)                                  2
(%i4)  carg(w);
(%o4)                                  π/6
```

(2)　親切に変形手順が記されていますが、Maxima には無用の長物です。

MAXIMA INPUT/OUTPUT

```
(%i5)  z[1]: 4*(1 - sin(x) + %i*cos(x))/(1 - sin(x) - %i*cos(x));
                                  4 (− sin x + i cos x + 1)
(%o5)                             ─────────────────────────
                                  − sin x − i cos x + 1
(%i6)  trigrat(%);*10
(%o6)                          4 i cos x − 4 sin x
```

より、「ウ」は 4 と求まりました。また、$-\sin\theta + i\cos\theta = i(\cos\theta + i\sin\theta)$ と変形できますから、「エ」は 4、「オ」は 90 と求まりました。

(3)　絶対値および偏角の基本的な性質から $\left|\frac{z_1}{z_0}\right| = \frac{|z_1|}{|z_0|} = \frac{4}{2} = 2$、$\arg\frac{z_1}{z_0} = \arg z_1 - \arg z_0 = (90^\circ + \theta) - (30^\circ + \theta) = 60^\circ$ と求める問題ですが、折角ですので Maxima に解かせてみましょう。

MAXIMA INPUT/OUTPUT

```
(%i7)  z[0]: w*(cos(x) + %i*sin(x));
(%o7)                          (i + √3)(i sin x + cos x)
(%i8)  trigsimp(z[1]/z[0]);*11
(%o8)                                √3 i + 1
(%i9)  polarform(%);
(%o9)                                2 e^(i π/3)
```

より、「カ」は 2、「キ」は 60 と求まりました。

3.2　微分積分（極限）

極限（値）を計算するには関数 `limit` を用います。関数 `limit` は書式

*10 trigrat = trigonometric + rational: trigonometric [trìg(ə)nəmétrik] function 三角関数。rational [rǽʃənl] 有理の。

*11 trigsimp = trigonometric + simplify: simplify [símpləfài] 簡単にする。

```
limit(関数，変数，近づける値)
```

または

```
limit(関数，変数，近づける値，近づける方向)
```

により実行します。2 番目の書式は（もちろん）右極限と左極限を求めるためのものですが、右極限の場合は plus を、左極限の場合は minus をそれぞれ「近づける方向」に設定します。この他、Maxima では我々が普段使っている無限大 ∞ 等の記号の代わりに、次のような表記が使われます。

表記	意味
inf	∞
minf	$-\infty$
und	定義されない（もしくは不明）
ind	不定だが有界
infinity	複素無限大

MAXIMA INPUT/OUTPUT

```
(%i1) limit(%e^x, x, inf);
(%o1)                                 inf
(%i2) limit(%e^x, x, minf);
(%o2)                                  0
(%i3) limit(sin(x)/x, x, 0);
(%o3)                                  1
(%i4) limit((-1)^x, x, inf);
(%o4)                                 ind
(%i5) limit((-2)^x, x, inf);
(%o5)                              infinity
(%i6) limit(1/x, x, 0);
(%o6)                                 und
```

上から順に $\lim_{x\to\infty} e^x = \infty$、$\lim_{x\to-\infty} e^x = 0$、$\lim_{x\to 0} \frac{\sin x}{x} = 1$、$\lim_{x\to\infty}(-1)^x$ = 定義されない、$\lim_{x\to\infty}(-2)^x$ = 定義されない、$\lim_{x\to 0} \frac{1}{x}$ = 発散（定義されない）を計算したものです。なお、微分積分（実関数）では負の数を底とする指数関数は定義しないので、`(%i4)` や `(%i5)` のような計算を実行させることには問題があることに注意しましょう。次に、右極限と左極限の例を計算してみましょう。

MAXIMA INPUT/OUTPUT

```
(%i7) limit(1/x, x, 0, plus);
(%o7)                                inf
(%i8) limit(1/x, x, 0, minus);
(%o8)                               minf
```

それぞれ、$\lim_{x\to+0}\frac{1}{x}=\infty$、$\lim_{x\to-0}\frac{1}{x}=-\infty$ を計算したものです。

3.3　微分積分（微分）

微分を計算するには関数 `diff` を用います。関数 `diff` は書式

`diff(`関数`,` 変数`)`

または

`diff(`関数`,` 変数$_1$`,` 回数$_1$`,` 変数$_2$`,` 回数$_2$`, .....)`

により実行します[*12]。手始めに積の微分 $\left(x^2\,2^x\right)'$ と商の微分 $\left(\frac{\log x}{x^2+1}\right)'$ を計算させてみましょう。

MAXIMA INPUT/OUTPUT

```
(%i1) diff(x^2*2^x, x);
(%o1)                        x^2 2^x log 2 + 2 x 2^x
(%i2) diff(log(x)/(x^2 + 1), x);
                                1          2 x log x
(%o2)                       ------- - ---------------
                            x (x^2 + 1)    2
                                          (x^2 + 1)
```

次は、関数 $\sqrt{x}$ の 10 階導関数と 2 変数関数 $\sin(2x+3y)$ を変数 x と y でそれぞれ 1 回ずつ微分する例です。

MAXIMA INPUT/OUTPUT

```
(%i3) diff(sqrt(x), x, 10);
                                   34459425
(%o3)                          - ----------
                                  19/2
                                 x
(%i4) diff(sin(2*x + 3*y), x, 1, y, 1);
(%o4)                        -6 sin(3 y + 2 x)
```

[*12] 変数を省略して `diff(`関数`)` を実行すると、全微分を出力します。

ここで、(%o4) は、関数 $\sin(2x+3y)$ を x で微分し、その結果 $2\cos(2x+3y)$ を y で微分したものになっています。つまり、関数 diff が求めているのは、指定された変数以外を定数と見なして微分する「偏微分」です。この事実はもっと単純な例で実験してみれば明らかになります。

MAXIMA INPUT/OUTPUT

```
(%i5) diff(x^2 + x*y + y^2, x);
```

$$(\%o5) \qquad y + 2x$$

変数 y を x の関数として微分したい場合、すなわち「陰関数微分」を求めたい場合は、変数 y 代わりに明示的に $f(x)$ を使うか、関数 depends を用いて y が x の関数であることを Maxima に教えてあげます[13]。

MAXIMA INPUT/OUTPUT

```
(%i6) diff(x^2 + x*f(x) + f(x)^2, x);
```

$$(\%o6) \qquad 2f(x)\frac{d}{dx}f(x) + x\frac{d}{dx}f(x) + f(x) + 2x$$

```
(%i7) depends(y, x);
```

$$(\%o7) \qquad [y(x)]$$

```
(%i8) diff(x^2 + x*y + y^2, x);
```

$$(\%o8) \qquad 2y\frac{dy}{dx} + x\frac{dy}{dx} + y + 2x$$

試しに、次の例題を解いてみましょう。

【例題】 方程式 $x^y = y^x$ によって定まる陰関数 y の導関数を求めよ。

【解答】 既に depends(y,x) を実行してありますから、微分して、$\dfrac{dy}{dx}$ について解くだけです。

MAXIMA INPUT/OUTPUT

```
(%i9) diff(x^y = y^x, x);
```

$$(\%o9) \qquad x^y\left(\log x\frac{dy}{dx} + \frac{y}{x}\right) = y^x\left(\frac{x\dfrac{dy}{dx}}{y} + \log y\right)$$

```
(%i10) solve(%, diff(y, x));
```

[13] 変数 derivabbrev に true を代入すると、$\dfrac{d}{dx}f(x)$ や $\dfrac{dy}{dx}$ の代わりに $f(x)_x$ や y_x のような出力に切り替えることができます。元に戻すには false を代入します。

(%o10)
$$\frac{dy}{dx} = -\frac{x\,y^{x+1}\,\log y - x^y\,y^2}{x^2\,y^x - x^{y+1}\,y\,\log x}$$

(%o10) が求める答えです。

定義した依存関係を解除するには関数 remove を使います。

MAXIMA INPUT/OUTPUT

```
(%i11) properties(y);
(%o11)                          [dependency]
(%i12) remove(y, dependency);
(%o12)                            done
(%i13) properties(y);
(%o13)                             []
(%i14) diff(x^y = y^x, x);
(%o14)                      x^(y-1) y = y^x log(y)
```

変数 y に割り当てられていた性質 dependency が、関数 remove の実行によって削除されたことが分かります。

Maxima には Taylor 展開（Maclaurin 展開）を求める関数 taylor も存在します。関数 taylor は書式

taylor(関数, 変数, 中心, 階数)

により実行します。言わずもがなですが、「中心」が 0 の場合を Maclaurin 展開と呼ぶのでした。関数 $\dfrac{1}{\sqrt{x}}$ の $x = 1$ を中心とした Taylor 展開と関数 $\arctan x$ の Maclaurin 展開を求めてみましょう。

MAXIMA INPUT/OUTPUT

```
(%i15) taylor(1/sqrt(x), x, 1, 4);
```
(%o15)/T/
$$1 - \frac{x-1}{2} + \frac{3\,(x-1)^2}{8} - \frac{5\,(x-1)^3}{16} + \frac{35\,(x-1)^4}{128} + \cdots$$
```
(%i16) taylor(atan(x), x, 0, 20);
```
(%o16)/T/
$$x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9} - \frac{x^{11}}{11} + \frac{x^{13}}{13} - \frac{x^{15}}{15} + \frac{x^{17}}{17} - \frac{x^{19}}{19} + \cdots$$

3.4 微分積分（積分）

積分を計算するには関数 integrate を用います。関数 integrate は不定積分か定積分かに応じてそれぞれ書式

integrate(関数，変数)　あるいは　integrate(関数，変数，下端，上端)

により実行します。試しに、不定積分 $\int \log^2 x\,dx$ および定積分 $\int_0^1 \arctan x\,dx$ を計算してみましょう。

MAXIMA INPUT/OUTPUT

```
(%i1) integrate(log(x)^2, x);
(%o1)                    x (log^2 x - 2 log x + 2)
(%i2) integrate(atan(x), x, 0, 1);
(%o2)                          - (2 log 2 - %pi)/4
```

不定積分 $\int \log^2 x\,dx$ の結果 (%o1) に積分定数 C が付いていないので、このままコピーペーストすると試験では減点されますので注意しましょう。積分定数が付かないと気が済まない場合は微分方程式として解を求めます。

MAXIMA INPUT/OUTPUT

```
(%i3) diff(f(x), x) = log(x)^2;
                                  d
(%o3)                             -- f(x) = log^2 x
                                  dx
(%i4) ode2(%, f(x), x);
                                                2
(%o4)                f(x) = x (log^2 x - 2 log x + 2) + %c
```

積分定数はさして重要ではありませんが、次の実行結果は明らかに間違いです。

MAXIMA INPUT/OUTPUT

```
(%i5) integrate(1/x, x);
(%o5)                                log x
```

正しくは、$\int \frac{1}{x}\,dx = \log|x| + C$ ですから Maxima は絶対値を忘れていることになります[*14]。絶対値を付けさせるには変数 `logabs`[*15] に `true` を代入しておきます。

MAXIMA INPUT/OUTPUT

```
(%i6) logabs: true;
(%o6)                                  true
(%i7) integrate(1/x, x);
(%o7)                                log abs x
```

ただ、式が複雑になると絶対値が煩わしくなりますので、元に戻しておきます。

MAXIMA INPUT/OUTPUT

```
(%i8) logabs: false;
(%o8)                                  false
```

なお、原始関数を求められなかった場合、Maxima はそのままの式を出力します。

MAXIMA INPUT/OUTPUT

```
(%i9) integrate(sin(x)/x, x);
(%o9)                                 ⌠ sin x
                                      ⎮ ----- dx
                                      ⌡   x
```

広義積分も運が良ければ計算できます。

MAXIMA INPUT/OUTPUT

```
(%i10) integrate(1/x^2, x, 1, inf);
(%o10)                                 1
(%i11) integrate(1/x^3, x, -1, minf);
                                       1
(%o11)                                 -
                                       2
(%i12) integrate(1/sqrt(x), x, 0, 1);
(%o12)                                 2
(%i13) integrate(1/x^2, x, 0, 1);
Integral is divergent
– an error. Quitting. To debug this try debugmode(true);
```

[*14] 絶対値が付かないのはバグではなく仕様です。

[*15] logabs = logarithm + absolute value: logarithm [lɔ́(ː)gərɪ̀ð(ə)m, lá(ː)gərɪ̀ð(ə)m] 対数。absolute value 絶対値。

(%i13) のように発散する場合はエラーになりますが、一応発散することは教えてくれます。

3.5 行列（基本演算）

行列を定義するには関数 matrix[*16] を用います。関数 matrix は書式

matrix([成分$_{11}$，成分$_{12}$，.....]，[成分$_{21}$，成分$_{22}$，.....]，.....);

により実行します。また、行列演算には次の記号を用います。

記号	役　　割
+	足し算
-	引き算
*	スカラー倍（もしくは成分同士のかけ算）
.	行列としてのかけ算
/	成分同士のかけ算
^	成分ごとの累乗
^^	行列としての累乗
^^-1	逆行列（関数 invert でも求まる）

いくつか単純な計算例を実行してみましょう。

MAXIMA INPUT/OUTPUT

```
(%i1) A: matrix([2, -1], [-3, 2]);
```

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

```
(%i2) B: matrix([3, 0], [-2, 1]);
```

(%o2)
$$\begin{pmatrix} 3 & 0 \\ -2 & 1 \end{pmatrix}$$

```
(%i3) A - B;
```

(%o3)
$$\begin{pmatrix} -1 & -1 \\ -1 & 1 \end{pmatrix}$$

```
(%i4) A.B;
```

(%o4)
$$\begin{pmatrix} 8 & -1 \\ -13 & 2 \end{pmatrix}$$

```
(%i5) 2*B;
```

(%o5)
$$\begin{pmatrix} 6 & 0 \\ -4 & 2 \end{pmatrix}$$

```
(%i6) A^^-1;
```

[*16] matrix [méitriks, mǽitriks] 行列。

```
(%o6)
```
$$\begin{pmatrix} 2 & 1 \\ 3 & 2 \end{pmatrix}$$

```
(%i7) A.% = %.A;
(%o7)
```
$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

最後の (%o7) は、逆行列との積が単位行列になることを確認しているところです。
続いて、次の例題を解いてみます。

【例題】 行列 $M = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$ と可換な行列を求めよ。

【解答】 求める行列を X とするとき、具体的に $MX = XM$ を計算してみます。

MAXIMA INPUT/OUTPUT

```
(%i8) M: matrix([0, 1, 0], [0, 0, 1], [0, 0, 0]);
```
$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

```
(%i9) X: matrix([a, b, e], [d, e, f], [g, h, i]);
```
$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$$

```
(%i10) M.X = X.M;
(%o10)
```
$$\begin{pmatrix} d & e & f \\ g & h & i \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & a & b \\ 0 & d & e \\ 0 & g & h \end{pmatrix}$$

(%o10) の成分を比較すると、$d = g = h = 0$ かつ $a = e = i$ かつ $b = f$ が得られますから、求める行列は $\begin{pmatrix} a & b & c \\ 0 & a & b \\ 0 & 0 & a \end{pmatrix}$（$a$、$b$、$c$ は任意の数）となります。

当然のことながら Maxima では行列の基本的な量である行列式や階数、固有値や固有ベクトル、固有多項式も求めることができます。主なものを表 3.1 にまとめました。これらを利用して、代表的な例題を解いてみましょう。

表 3.1 行列に関する関数（主なもの）

関　数	役　割
determinant	行列式
rank	階数
charpoly	固有多項式（特性多項式）
eigenvalues	固有値
eigenvectors	固有ベクトル
transpose	転置行列
adjoint	余因子行列
triangularize	上三角化
echelon	階段行列化
mattrace	トレース（load ("nchrpl"); が必要）

【例題】 行列 $M = \begin{pmatrix} 1 & x & x & x \\ x & 1 & x & x \\ x & x & 1 & x \\ x & x & x & 1 \end{pmatrix}$ が逆行列を持たないような x の値を全て求めよ。

【解答】 行列式が 0 になる x の値を求めればよいですね。

MAXIMA INPUT/OUTPUT

```
(%i11) M: matrix([1,x,x,x], [x,1,x,x], [x,x,1,x], [x,x,x,1]);
```

(%o11) $$\begin{pmatrix} 1 & x & x & x \\ x & 1 & x & x \\ x & x & 1 & x \\ x & x & x & 1 \end{pmatrix}$$

```
(%i12) factor(determinant(M));[17]
```

(%o12) $$-(x-1)^3\,(3\,x+1)$$

```
(%i13) solve(% = 0, x);[18]
```

(%o13) $$\left[x = -\frac{1}{3},\ x = 1\right]$$

[17] determinant [ditə́:(r)mənənt] 行列式。

[18] solve(%, x); のように =0 は省略できます。

本問の場合行列式が単純な式に因数分解できましたから、関数 solve を使うまでもなく、(%o12) からすぐに $x = 1, -\frac{1}{3}$ を導く方が自然かもしれません。

【例題】 行列 $M = \begin{pmatrix} 1 & -3 & 0 \\ 2 & -2 & 1 \\ 1 & 0 & -2 \end{pmatrix}$ を対称行列と交代行列の和に表せ[*19]。

【解答】 なにやら難しそうな問題ですが、実は $S = M + {}^tM$ と $T = M - {}^tM$ がそれぞれ対称行列と交代行列になりますから、$M = \frac{1}{2}S + \frac{1}{2}T$ が答えです。従って、

MAXIMA INPUT/OUTPUT

```
(%i14) M: matrix([1, -3, 0], [2, -2, 1], [1, 0, -2]);
```

$$\text{(\%o14)} \quad \begin{pmatrix} 1 & -3 & 0 \\ 2 & -2 & 1 \\ 1 & 0 & -2 \end{pmatrix}$$

```
(%i15) S: M + transpose(M);
```
[*20]

$$\text{(\%o15)} \quad \begin{pmatrix} 2 & -1 & 1 \\ -1 & -4 & 1 \\ 1 & 1 & -4 \end{pmatrix}$$

```
(%i16) T: M - transpose(M);
```

$$\text{(\%o16)} \quad \begin{pmatrix} 0 & -5 & -1 \\ 5 & 0 & 1 \\ 1 & -1 & 0 \end{pmatrix}$$

```
(%i17) '(M = 1/2 * S + 1/2 * T);
```

$$\text{(\%o17)} \quad M = \frac{T}{2} + \frac{S}{2}$$

```
(%i18) ev(%);
```
[*21]

$$\text{(\%o18)} \quad \begin{pmatrix} 1 & -3 & 0 \\ 2 & -2 & 1 \\ 1 & 0 & -2 \end{pmatrix} = \begin{pmatrix} 1 & -3 & 0 \\ 2 & -2 & 1 \\ 1 & 0 & -2 \end{pmatrix}$$

より、$\begin{pmatrix} 1 & -3 & 0 \\ 2 & -2 & 1 \\ 1 & 0 & -2 \end{pmatrix} = \frac{1}{2}\begin{pmatrix} 2 & -1 & 1 \\ -1 & -4 & 1 \\ 1 & 1 & -4 \end{pmatrix} + \frac{1}{2}\begin{pmatrix} 0 & -5 & -1 \\ 5 & 0 & 1 \\ 1 & -1 & 0 \end{pmatrix}$ と求まりました。なお、最後の (%i17) と (%i18) は本来不要なものですが、まず (%i17) で行列 M、S、T を計算せずに出力しておき、(%i18) で計算を実行させることで、入力ミスがないかを含め検算しました。

[*19] 一般に、${}^tA = A$ および ${}^tA = -A$ を満たす行列をそれぞれ対称行列、交代行列と呼ぶのでした。

[*20] transpose [trænspóuz]: transposed matirx 転置行列。

[*21] ev = evaluate [iváljuèit] 評価する。

【例題】 ベクトル空間 $\mathbb{R}^4$ 内において、$v_1 = (3, 0, -1, 1)$、$v_2 = (1, -2, 3, 2)$、$v_3 = (2, 0, -1, 2)$、$v_4 = (2, -2, 3, 1)$ とするとき、これらのうち 1 次独立なものの最大個数を求めよ。また、そのようなベクトルを 1 組求め、それらの 1 次結合で残りのベクトルを表せ。

【解答】 最大個数については階数を計算するだけです。

MAXIMA INPUT/OUTPUT

```
(%i19) v[1]: [3, 0, -1, 1];
(%o19)                          [3, 0, −1, 1]
(%i20) v[2]: [1, -2, 3, 2];
(%o20)                          [1, −2, 3, 2]
(%i21) v[3]: [2, 0, -1, 2];
(%o21)                          [2, 0, −1, 2]
(%i22) v[4]: [2, -2, 3, 1];
(%o22)                          [2, −2, 3, 1]
(%i23) A: matrix(v[1], v[2], v[3], v[4]);
                                [ 3   0   −1  1 ]
                                [ 1   −2  3   2 ]
(%o23)                          [ 2   0   −1  2 ]
                                [ 2   −2  3   1 ]
(%i24) rank(A);[*22]
(%o24)                                3
```

従って、最大個数は 3 と分かりました。

続いて後半部分ですが、独立なベクトルが 3 個あるはずですから、4 個のベクトルから適当に 3 個選んでそれらの 1 次独立性を調べます。

MAXIMA INPUT/OUTPUT

```
(%i25) rank(matrix(v[1], v[2], v[3]));
(%o25)                                3
```

一発で求まりましたが、実はどの 3 個を選んでも 1 次独立です。そこで、$v_4 = a\,v_1 + b\,v_2 + c\,v_3$ を満たす実数 a、b、c を求めます。

MAXIMA INPUT/OUTPUT

```
(%i26) a*v[1] + b*v[2]+c*v[3]-v[4];
```

[*22] rank [ræŋk] 階数。

```
(%o26)      [2 c + b + 3 a − 2, 2 − 2 b, −c + 3 b − a − 3, 2 c + 2 b + a − 1]
(%i27) solve(%, [a, b, c]);
Dependent equations eliminated: (4)[*23]
(%o27)                          [[a = 1, b = 1, c = −1]]
```

【例題】 行列 $M = \begin{pmatrix} 2 & -1 & -2 \\ -2 & 0 & 4 \\ 1 & 1 & 0 \end{pmatrix}$ の全ての固有値 λ と、固有空間 $V(\lambda)$ の次元を求めよ。

【解答】 固有値を求める関数 `eigenvalues`[*24] を使えば前半は一発です。

MAXIMA INPUT/OUTPUT

```
(%i28) M: matrix([2, -1, -2], [-2, 0, 4], [1, 1, 0]);
                                [ 2   −1   −2 ]
(%o28)                          [ −2  0    4  ]
                                [ 1   1    0  ]
(%i29) eigenvalues(M);
(%o29)                          [[−2, 2], [1, 2]]
```

最初のリスト [−2, 2] が固有値で、2 番目のリスト [1, 2] はそれぞれの固有値に対する重複度です。従って、求める固有値が −2 と 2 であることと、固有空間 $V(-2)$ の次元が 1 であることが分かりました。固有空間 $V(2)$ の次元については、行列の次数から行列 $M - 2E$ の階数を引いたものに一致しますから、

MAXIMA INPUT/OUTPUT

```
(%i30) 3 - rank(M - 2 * ident(3));
(%o30)                                  1
```

より $\dim V(2) = 1$ が得られました。なお、関数 `ident`[*25] は単位行列を生成する関数で、`ident(3)` を実行すると 3 次の単位行列 $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ が生成されます。

[*23] 「a、b、c の値は最初の 3 式だけで求まり、4 番目の式は一次従属なので不要でした。」という意味のメッセージです。このメッセージが煩わしければ変数 `linsolvewarn` に `false` を代入しておきます。

[*24] eigenvalues = eigenvalue [áigənvæljuː] 固有値。

[*25] ident = identity: identity matrix 単位行列。

3.6 行列（生成）

次数の高い行列を定義する場合、成分を一つ一つ入力するのは苦痛です。そこで、関数 ident のように、行列の作成（生成）を容易にしてくれる関数をまとめておきます。

関数 diagmatrix(n, x) を実行すると、対角成分に要素 x を並べた n 次正方行列を生成することができます。従って、diagmatrix(n, x) は x*ident(n) と同じです。

MAXIMA INPUT/OUTPUT

```
(%i1) diagmatrix(4, x) = x * ident(4);
```

$$\text{(\%o1)}\qquad \begin{pmatrix} x & 0 & 0 & 0 \\ 0 & x & 0 & 0 \\ 0 & 0 & x & 0 \\ 0 & 0 & 0 & x \end{pmatrix} = \begin{pmatrix} x & 0 & 0 & 0 \\ 0 & x & 0 & 0 \\ 0 & 0 & x & 0 \\ 0 & 0 & 0 & x \end{pmatrix}$$

関数 zeromatrix(m, n) を実行すると、m 行 n 列の零行列を生成することができます。

MAXIMA INPUT/OUTPUT

```
(%i2) zeromatrix(1, 5);
```

$$\text{(\%o2)}\qquad \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

```
(%i3) zeromatrix(2, 5);
```

$$\text{(\%o3)}\qquad \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

行列の成分がある一定の規則で並んでいる場合は、ラムダ関数（もしくは 2 次元配列）を利用した行列生成関数 genmatrix[*26] が有効です。関数 genmatrix は書式

genmatrix(配列関数, 行$_2$, 列$_2$, 行$_1$, 列$_1$)

により実行します[*27]。このとき、「配列関数」の第 1 成分を「行$_1$」から「行$_2$」まで変化させ、第 2 成分を「列$_1$」から「列$_2$」まで変化させた行列が生成されます。従って、生成される行列の行数は「行$_2$ − 行$_1$ + 1」、列数は「列$_2$ − 列$_1$ + 1」となります。

MAXIMA INPUT/OUTPUT

```
(%i4) a: lambda([i, j], x[i, j]);
```

$$\text{(\%o4)}\qquad \text{lambda}([i,\ j],\ x_{i,j})$$

```
(%i5) genmatrix(a, 4, 4, 1, 1);
```

[*26] genmatrix = generate matrix

[*27] 列$_1$ を省略すると 列$_1$= 行$_1$ と見なされ、列$_1$ と行$_1$ を同時に省略すると、両方とも 1 と見なされます。従って、genmatrix(配列関数, 行, 列, 1, 1) は genmatrix(配列関数, 行, 列) と略記できます。

```
(%o5)
```
$$\begin{pmatrix} x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} \\ x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} \\ x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} \\ x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4} \end{pmatrix}$$

```
(%i6) genmatrix(a, 4, 4, 2, 1);
```
```
(%o6)
```
$$\begin{pmatrix} x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} \\ x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} \\ x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4} \end{pmatrix}$$

```
(%i7) genmatrix(a, 4, 4, 3, 1);
```
```
(%o7)
```
$$\begin{pmatrix} x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} \\ x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4} \end{pmatrix}$$

```
(%i8) genmatrix(a, 4, 4, 4, 1);
```
```
(%o8)
```
$$\begin{pmatrix} x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4} \end{pmatrix}$$

列については、上記 4 通りとも全て 4 − 1 + 1 = 4 列であること、一方、行については、終了番号が常に 4 であるのに対し、開始番号が 1、2、3、4 と変化していることに注目すると、関数 genmatrix の意味がよく分かると思います。この関数 genmatrix を利用して、いくつか有名な関数を生成してみましょう。

MAXIMA INPUT/OUTPUT

```
(%i9) a: lambda([i, j], x[j]^i);
```
```
(%o9)
```
$$\text{lambda}\left([i,\ j],\ {x_j}^i\right)$$

```
(%i10) genmatrix(a, 3, 4, 0, 1);
```
```
(%o10)
```
$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 \\ {x_1}^2 & {x_2}^2 & {x_3}^2 & {x_4}^2 \\ {x_1}^3 & {x_2}^3 & {x_3}^3 & {x_4}^3 \end{pmatrix}$$

```
(%i11) factor(determinant(%));
```
```
(%o11)
```
$$(x_2 - x_1)(x_3 - x_1)(x_3 - x_2)(x_4 - x_1)(x_4 - x_2)(x_4 - x_3)$$

所謂 Vandermonde（ファンデルモンド）[*28] 行列とその行列式です。巡回行列も作ってみましょう。

MAXIMA INPUT/OUTPUT

```
(%i12) kill(all);
(%o12)                                done
(%i13) L:[a, b, c, d, e];
(%o13)                              [a, b, c, d, e]
```

[*28] Alexandre-Théophile Vandermonde (1735.2.28 - 1796.1.1) パリのバイオリニストにして化学者。数学については、1771 年と 1772 年に計 4 本の論文を著していますが、この行列及び行列式についての記述はないそうです。

```
(%i14) x: lambda([i, j], L[mod(j - i, 5) + 1]);
(%o14)                    lambda([i, j], L_{mod(j-i,5)+1})
(%i15) genmatrix(x, 5, 5, 1, 1);
```

$$\text{(\%o15)}\quad \begin{pmatrix} a & b & c & d & e \\ e & a & b & c & d \\ d & e & a & b & c \\ c & d & e & a & b \\ b & c & d & e & a \end{pmatrix}$$

ここで、関数 mod は余りを返す関数ですが、割られる数が負であっても正の余りを返す点が関数 remainder との違いです。対角行列も簡単です。

MAXIMA INPUT/OUTPUT

```
(%i16) a: lambda([i, j], if i=j then [1, 2, 3, 4][i] else 0);
(%o16)          lambda([i, j], if i = j then [1, 2, 3, 4]_i else 0)
(%i17) genmatrix(a, 4, 4, 1, 1);
```

$$\text{(\%o17)}\quad \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 4 \end{pmatrix}$$

少しずらすと冪零行列の代表格とも言える行列を生成することができます。

MAXIMA INPUT/OUTPUT

```
(%i18) genmatrix(a, 5, 4, 2, 1);
```

$$\text{(\%o18)}\quad \begin{pmatrix} 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

疑似乱数を発生させる関数 random を用いれば、全ての成分がランダムな整数であるような行列も簡単に生成できます。

MAXIMA INPUT/OUTPUT

```
(%i19) a: lambda([i, j], random(100));
(%o19)                    lambda([i, j], random(100))
(%i20) genmatrix(a, 3, 3, 1, 1);
```

$$\text{(\%o20)}\quad \begin{pmatrix} 6 & 85 & 15 \\ 25 & 88 & 45 \\ 18 & 38 & 45 \end{pmatrix}$$

```
(%i21) genmatrix(a, 3, 3, 1, 1);
```

```
(%o21)
```
$$\begin{pmatrix} 17 & 91 & 3 \\ 3 & 3 & 51 \\ 91 & 58 & 34 \end{pmatrix}$$

あるいは、各成分に関数を適用する関数 matrixmap を用いる方法でも良いかもしれません。

MAXIMA INPUT/OUTPUT

```
(%i22) matrixmap(random, zeromatrix(3, 3) + 100);
```
```
(%o22)
```
$$\begin{pmatrix} 16 & 27 & 34 \\ 6 & 39 & 61 \\ 26 & 8 & 52 \end{pmatrix}$$
```
(%i23) matrixmap(random, zeromatrix(3, 3) + 100);
```
```
(%o23)
```
$$\begin{pmatrix} 50 & 12 & 52 \\ 1 & 73 & 25 \\ 42 & 9 & 79 \end{pmatrix}$$

すなわち、行列 $\begin{pmatrix} 100 & 100 & 100 \\ 100 & 100 & 100 \\ 100 & 100 & 100 \end{pmatrix}$ を作って、その各成分に関数 random を適用します。なお、random(n) を実行すると 0 以上 n 未満の疑似乱数が出力されますが、引数 n が自然数か小数かに対応して、出力される数も自然数または小数になります。

3.7　行列（抽出と連結）

指定した行列から行および列を抽出するには、それぞれ関数 row および col を用います。

MAXIMA INPUT/OUTPUT

```
(%i1) A: matrix([1,2,3,4], [5,6,7,8], [9,10,11,12], [13,14,15,16]);
```
```
(%o1)
```
$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix}$$
```
(%i2) row(A, 1);
```
```
(%o2)
```
$$\begin{pmatrix} 1 & 2 & 3 & 4 \end{pmatrix}$$
```
(%i3) row(A, 2);
```
```
(%o3)
```
$$\begin{pmatrix} 5 & 6 & 7 & 8 \end{pmatrix}$$
```
(%i4) row(A, 3);
```
```
(%o4)
```
$$\begin{pmatrix} 9 & 10 & 11 & 12 \end{pmatrix}$$
```
(%i5) row(A, 4);
```

```
(%o5)
```
$$\begin{pmatrix}13 & 14 & 15 & 16\end{pmatrix}$$
```
(%i6) col(A, 1);
```
```
(%o6)
```
$$\begin{pmatrix}1\\5\\9\\13\end{pmatrix}$$
```
(%i7) col(A, 2);
```
```
(%o7)
```
$$\begin{pmatrix}2\\6\\10\\14\end{pmatrix}$$
```
(%i8) col(A, 3);
```
```
(%o8)
```
$$\begin{pmatrix}3\\7\\11\\15\end{pmatrix}$$
```
(%i9) col(A, 4);
```
```
(%o9)
```
$$\begin{pmatrix}4\\8\\12\\16\end{pmatrix}$$

部分行列を抽出するには関数 submatrix を用います。関数 submatrix は書式

submatrix(行$_1$, 行$_2$, ..., 行列, 列$_1$, 列$_2$, ...)

により実行します。すなわち、「行列」の左側に削除したい行番号を列挙し、右側に削除したい列番号を列挙します。

MAXIMA INPUT/OUTPUT

```
(%i10) display(A);
```
```
(%o10)
```
$$A = \begin{pmatrix}1 & 2 & 3 & 4\\5 & 6 & 7 & 8\\9 & 10 & 11 & 12\\13 & 14 & 15 & 16\end{pmatrix}$$
```
(%i11) submatrix(1, A);
```
```
(%o11)
```
$$\begin{pmatrix}5 & 6 & 7 & 8\\9 & 10 & 11 & 12\\13 & 14 & 15 & 16\end{pmatrix}$$
```
(%i12) submatrix(A, 4);
```
```
(%o12)
```
$$\begin{pmatrix}1 & 2 & 3\\5 & 6 & 7\\9 & 10 & 11\\13 & 14 & 15\end{pmatrix}$$

```
(%i13) submatrix(1, A, 4);
```

$$\begin{pmatrix} 5 & 6 & 7 \\ 9 & 10 & 11 \\ 13 & 14 & 15 \end{pmatrix}$$

```
(%i14) submatrix(1, 2, A, 4);
```

$$\begin{pmatrix} 9 & 10 & 11 \\ 13 & 14 & 15 \end{pmatrix}$$

```
(%i15) submatrix(1, 2, A, 2, 4);
```

$$\begin{pmatrix} 9 & 11 \\ 13 & 15 \end{pmatrix}$$

行列にリストや行列を連結するには、行として連結する場合は関数 addrow を、列として連結するには関数 addcol を用います。

MAXIMA INPUT/OUTPUT

```
(%i16) A: zeromatrix(2, 2) + 1;
```

$$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

```
(%i17) addrow(A, [a, b]);
```

$$\begin{pmatrix} 1 & 1 \\ 1 & 1 \\ a & b \end{pmatrix}$$

```
(%i18) addcol(A, [a, b]);
```

$$\begin{pmatrix} 1 & 1 & a \\ 1 & 1 & b \end{pmatrix}$$

```
(%i19) B: zeromatrix(2, 2) + x;
```

$$\begin{pmatrix} x & x \\ x & x \end{pmatrix}$$

```
(%i20) addrow(A, B);
```

$$\begin{pmatrix} 1 & 1 \\ 1 & 1 \\ x & x \\ x & x \end{pmatrix}$$

```
(%i21) addcol(A, B);
```

$$\begin{pmatrix} 1 & 1 & x & x \\ 1 & 1 & x & x \end{pmatrix}$$

```
(%i22) zeromatrix(2, 2);
```

$$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

```
(%i23) addrow(addcol(A, %), addcol(%, B));
```

(%o23)
$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & x & x \\ 0 & 0 & x & x \end{pmatrix}$$

参考文献

[1] Maxima 公式サイト：http://maxima.sourceforge.net

[2] Macsyma の歴史 : http://www.answers.com/topic/macsyma

[3] Intel Mac 用 Maxima バイナリ :
http://www.muskmelon.jp/macosx/

[4] Maxima をソースからインストールする手順 :
http://www.cymric.jp/maxima/maxima-leopard.html

[5] Maxima を Windows XP へインストールする手順 :
http://www.cymric.jp/maxima/maxima-winxp.html

索引