Einführung in Maxima

Robert Glöckner © 2006, 2007

Inhaltsverzeichnis

$Header: /home/cvs/lisp/maxima-einfuehrung.html,v 1.44 2007/06/02 13:59:21 robert Exp $

1. Hinweise

Ich möchte nur kurz betonen, dass ich selbst Maxima-Anfänger bin und der Text nur auf die Grundlagen der Maxima-Benutzung eingehen kann. Die Beispiele haben keinen tieferen Sinn. Sie dienen lediglich der Darstellung der Möglichkeiten von Maxima.
Für weitere Anregungen bin ich immer dankbar.
email (entferne alle Ziffern): 2R4o5b66e7r8t9.G0l5oe55ck8ne5r@6w0eb4.d3e

Verbesserungsvorschläge durch: Volker van Nek, Robert Figura, Dieter Schuster, Angelika Wirsing, Robert Dodier

2. Einführung

2.1. Starten von Maxima

Maxima ist ein in Lisp geschriebenes freies Computer-Algebra System (http://maxima.sourceforge.net/index.shtml). Es ist auf verschiedenen Betriebssystemen lauffähig. Es gibt mehrere Möglichkeiten das Programm zu verwenden:

- auf der Konsole (hierzu maxima, bzw. maxima.bat starten)
- eine rudimentäre grafische Oberfläche bietet xmaxima (mitgeliefert)
- eine grafische Formelausgabe bietet wxmaxima
- für Leute die LaTex benutzen ist texmax und emaxima interessant
- für Emacs-verrückte gibt es einen mitgelieferten maxima und emaxima Modus (Start im Emacs mit M-x maxima oder Öffnen einer .max Datei)

Startet man Maxima (auf der Konsole) so erhält man folgende Meldung:

```
Maxima 5.10.0 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)
```

Es erscheint eine Meldung über die freie Lizenz, die Widmung an Prof. W. Schelter (ihm haben wir die freie Version von Maxima zu verdanken) und ein sog. Label `(%i1)`. Jede Eingabe wird mit einer Marke (Label) gekennzeichnet. Marken, welche mit einem i beginnen kennzeichnen Benutzereingaben, o-Markierungen kennzeichnen Ausgaben des Programms. Der Benutzer sollte dies bei der Namensgebung eigener Variablen oder Funktionen berücksichtigen, um Verwechslungen zu vermeiden.

2.2. Kommandoeingabe

Kommandos werden entweder mit einem Semikolon `;` oder einem Dollarzeichen `$` abgeschlossen. Es reicht nicht Return oder Enter zu drücken. Maxima wartet auf eines der beiden o.g. Zeichen, sont beginnt Maxima nicht mit der Auswertung der Eingabe. Ist das letzte Zeichen ein Semikolon, so wird das Ergebnis der Verarbeitung angezeigt, im Falle eines Dollarzeichens wird die Anzeige unterdrückt. Dies kann bei sehr langen Ergebnissen sinnvoll sein, um die Wartezeit zu reduzieren und die Übersicht zu wahren.
Maxima unterscheidet Groß- und Kleinschreibung. Alle eingebauten Funktionen und Konstanten sind klein geschrieben (`simp, solve, ode2, sin, cos, %e, %pi, inf etc`). sImP oder SIMP werden von Maxima nicht den eingebauten Funktionen

zugeordnet. Benutzerfunktionen und -variablen können klein und/oder groß geschrieben werden.
Auf vorherige Ergebnisse und Ausdrücke kann mittels `%` zugegriffen werden. `%` bezeichnet das letzte Ergebnis, `%i13` die 13. Eingabe und `%o27` das 27. Ergebnis, `''%i42` wiederholt die Berechnung der 42. Eingabe, `%th(2)` ist das vorletzte Ergebnis.

2.3. Zuweisungen

Ausdrücke werden mit `:` einem Symbol zugewiesen. Funktionen werden mit `:=` einem Symbol zugewiesen. Makro(expansions)funktionen werden mit `::=` definiert.

```
(%i151) value : 3;

(%o151)                                3


(%i152) equation : a + 2 = b;

(%o152)                            a + 2 = b


(%i153) function(x) := x + 3;

(%o153)                     function(x) := x + 3


(%i154) function(3);

(%o154)                                6


(%i155) function(b);

(%o155)                              b + 3


(%i156) printq1 (x) ::= block (print ("(1) x is equal to", x),
                                  '(print ("(2) x is equal to", x)));

(%156) printq1(x) ::= block(print("(1) x is equal to", x),
                               '(print("(2) x is equal to", x)))
```

Zuweisungen werden mit `kill` einzeln oder auch insgesamt gelöscht.

```
(%i150) kill(equation);

(%o150)                                done


(%i151) equation;

(%o151)                            equation
```

```
(%i152) function(3);

(%o152)                                6


(%i153) kill(all);

(%o154)                              done


(%i155) function(3);

(%o155)                          function(3)
```

2.4. Beenden von Maxima

Zum Abbrechen eines Kommandos drückt man die Tastenkombination `Strg-C` oder `Strg-G`. Meldet sich der Debugger, so beendet man diesen durch Eingabe von `Q`. Zum Beenden von Maxima gibt man `quit();` ein (Bemerkung: unter xmaxima das Menue benutzen).

2.5. Hilfefunktionen

Neben den Hilfefunktionen der Benutzerumgebung enthält Maxima eigene Hilfsfunktionen. Mit `apropos` kann nach Befehlen bezüglich eines Stichwortes gesucht werden, mit `describe` können detaillierte Befehlsbeschreibungen angezeigt werden:

```
(%i3) apropos('plot);

(%o3) [plot, plot2d, plot2dopen, plot2d_ps, plot3d, plotheight, plotmode,
                                          plotting, plot_format, plot_options]


(%i4) describe("plot");
```

Manchmal fragt `describe` auch nach, welcher Teilbereich beschrieben werden soll (hier nur ein kleiner Ausschnitt der angezeigten Informationen):

```
 0: (maxima.info)Plotting.
 1: Definitions for Plotting.
 2: OPENPLOT_CURVES :Definitions for Plotting.
 3: PLOT2D :Definitions for Plotting.
 4: PLOT2D_PS :Definitions for Plotting.
 5: PLOT3D :Definitions for Plotting.
 6: PLOT_OPTIONS :Definitions for Plotting.
 7: SET_PLOT_OPTION :Definitions for Plotting.
Enter n, all, none, or multiple choices eg 1 3 : 5
Info from file /usr/share/info/maxima.info:PLOT3D
```

```
(expr,xrange,yrange,...,options,..)
 -- Function: PLOT3D ([expr1,expr2,expr3],xrange,yrange,...,options,..)
          plot3d(2^(-u^2+v^2),[u,-5,5],[v,-7,7]);
     would plot z = 2^(-u^2+v^2) with u and v varying in [-5,5] and
     [-7,7] respectively, and with u on the x axis, and v on the y axis.

     An example of the second pattern of arguments is
          plot3d([cos(x)*(3+y*cos(x/2)),sin(x)*(3+y*cos(x/2)),y*sin(x/2)],
             [x,-%pi,%pi],[y,-1,1],['grid,50,15])

     which will plot a moebius band, parametrized by the 3 expressions
     given as the first argument to plot3d.  An additional optional
     argument [grid,50,15] gives the grid number of rectangles in the x
     direction and y direction.
....
```

Mit der Funktion `example` können Beispiele zu einigen Funktionen von Maxima angezeigt werden (hier gekürzt):

```
(%i3) example(integrate);


(%i4) test(f):=block([u],u:integrate(f,x),ratsimp(f-diff(u,x)))

(%o4) test(f) := block([u], u : integrate(f, x), ratsimp(f - diff(u, x)))


(%i5) test(sin(x))

(%o5)                                0


(%i6) test(1/(x+1))

(%o6)                                0


(%i7) test(1/(x^2+1))

(%o7)                                0


(%i8) integrate(sin(x)^3,x)
                                   3
                              cos (x)
(%o8)                         ------- - cos(x)
                                   3
...
```

2.6. Darstellung der Ergebnisse

Die Darstellung der Ergebnisse von Maxima, ist im Wesentlichen von der verwendeten Oberfläche abhängig. Während die Ausgabe auf der Konsole und im einfachen Emacs-

Modus auf die Darstellung von ASCII Zeichen begrenzt ist, zeigen der erweiterete Emacs-Modus, Imaxima, TexMacs und WxMaxima die Ergebnisse in grafischer Form an. D. h. es werden entsprechende Symbole für Pi, Integral, Summe usw. verwendet.
Allgemein zeichnen sich die Ausgaben von Maxima durch exakte (rationale) Arithmetik aus:

```
(%i38) 1/11 + 9/11;

                                      10
(%o38)                                --
                                      11
```

Irrationale Zahlen werden in ihrer symbolischen Form beibehalten (mit `%` wurde auf das Ergebnis der letzten Berechnung zugegriffen):

```
(%i39) (sqrt(3) - 1)^4;

                                        4
(%o39)                          (sqrt(3) - 1)


(%i40) expand(%);

(%o40)                          28 - 16 sqrt(3)
```

Mit `ev(Ausdruck, numer);` oder kurz: `Ausdruck, numer;` oder `float(Ausdruck)` kann eine Dezimaldarstellung erzwungen werden (beachten Sie hier die Referenz auf das vorangegangene Ergebnis Nr. 40 via `%o40`):

```
(%i41) %o40, numer;

(%o41)                          0.28718707889796



(%i5) float(%e);

(%o5)                           2.718281828459045
```

Die Voreinstellung der Genauigkeit bei Fließkommazahlen beträgt 16 Stellen, wobei die letzte Stelle unsicher ist. Die Genauigkeit kann beliebig eingestellt werden, wenn der Zahlentyp `bfloat` verwendet wird. Die Anzahl der angezeigten Stellen wird mit `fpprec` gesteuert. Man kann dazu `fpprec` nur für die Auswertung einer Zeile setzen, wie dies in Zeile `%i46` geschieht, oder für alle folgenden Berechnungen setzen, wie dies in Zeile `%i48` geschieht:

```
(%i45) bfloat(%o40);

(%o45)                       2.871870788979631B-1


(%i46) bfloat(%o40), fpprec=100;
```

```
(%o46) 2.8718707889796330356085853590604212891511593903390995110708832876907 17#
29459199406701661011884022789 1B-1


(%i47) fpprec;

(%o47)                                16


(%i48) fpprec : 20;

(%o48)                                20


(%i49) bfloat(%o40);

(%o49)                    2.8718707889796330358B-1


(%i50) bfloat(%o40), fpprec=100;

(%o50) 2.8718707889796330356085853590604212891511593903390995110708832876907 17#
29459199406701661011884022789 1B-1
```

Die Eingabe von bestimmten Konstanten (e, i, pi, ...) erfolgt mit vorangestelltem `%` (`%e, %i, %pi...`).
Die Darstellung von bestimmten Konstanten (e, i, pi, ...), Operatoren (Summe, Integral, Ableitungen, ...) und anderen Symbolen (Klammern, Brüche, ...) ist abhängig von der gewählten Oberfläche. Im Textmodus, der von der Konsole, dem einfachen Emacs-Modus und der mitgelieferten xmaxima Oberfläche geboten wird, werden Konstanten mit einem `%` vorangestellt (`%e, %i, %pi, ...`), Operatoren werden in ASCII-Grafik dargestellt, Klammern werden nicht in der Größe expandiert und Brüche mit Hilfe von - dargestellt:

```
(%i51) sqrt(-3);

(%o51)                            sqrt(3) %i


(%i52) exp(5 * a);

                                        5 a
(%o52)                                 %e



(%i54) integrate( f(x) , x, 0, inf);

                                    inf
                                   /
                                   [
(%o54)                             I    f(x) dx
                                   ]
                                   /
                                    0
```

Die Darstellung im erweiterten Emacs-Modus, in Imaxima und in TexMax ist grafisch, d. h. für `%pi, %e`, Integrale, Summen, ... werden entsprechende Symbole verwendet.
Maxima kann natürlich auch Funktionen plotten. Die Funktion
`plot2d([Funktionsliste], [X-Var, Min, Max], [Y-Var, Min, Max]);`
kann eine Gruppe von Funktionen plotten. Hierzu gibt man die Liste von Funktionen in eckigen Klammern und durch Kommas getrennt als ersten Parameter an. Es folgt eine Liste, welche die abhängige Variable und den Plotbereich (x-Achse) angibt. Es gibt noch viele andere Plotmöglichkeiten (`apropos('plot), describe(plot)`)

```
(%i55)  plot2d([sin(x), cos(x)], [x, 0, 5]);


(%i58) apropos('plot);

(%o58) [plot, plot2d, plot2dopen, plot2d_ps, plot3d, plotheight, plotmode,
                                          plotting, plot_format, plot_options]


(%i59) describe(plot);

 0: (maxima.info)Plotting.
 1: Definitions for Plotting.
 2: openplot_curves :Definitions for Plotting.
 3: plot2d :Definitions for Plotting.
 4: plot2d_ps :Definitions for Plotting.
 5: plot3d :Definitions for Plotting.
 6: plot_options :Definitions for Plotting.
 7: set_plot_option :Definitions for Plotting.
Enter space-separated numbers, `all' or `none': Enter space-separated numbers,
`all' or `none': none
```

3. Rechnen mit Maxima

3.1. Operatoren

Die üblichen arithmetischen Operatoren stehen zur Verfügung:

- `+` Addition,
- `-` Subtraktion,
- `*` skalare Multiplikation,
- `.` Skalarmultiplikation von Vektoren und Matrix-Multiplikation,
- `/` Division,
- `**` oder `^` Potenzfunktion,
- `sqrt()` Wurzelfunktion,
- `exp()` Exponentialfunktion,
- `log()` natürliche Logarithmusfunktion
- `...`

3.2. Algebra

Niemand ist vor Flüchtigkeitsfehlern bei der Umformung, Transformation usw. von algebraischen Ausdrucken gefeit. Hier eignet sich Maxima hervorragend bei der Untersützung analytischer Berechnungen.
Hier ein einfaches Beispiel für die Behandlung von Polynomen. Zunächst wird via `expand` expandiert, anschließend eine Ersetzung vorgenommen, danach mittels `ratsimp` ein gemeinsamer Nenner gesucht und anschließend via `factor` faktorisiert:

```
(%i72) (5*a + 3*a*b )^3;

                                        3
(%o72)                          (3 a b + 5 a)


(%i73) expand(%);

                       3  3         3  2         3           3
(%o73)             27 a  b  + 135 a  b  + 225 a  b + 125 a


(%i74) %, a=1/x;

                             3         2
                          27 b    135 b    225 b   125
(%o74)                    ----- + ------ + ----- + ---
                            3         3        3      3
                           x         x        x      x
```

```
(%i75) ratsimp(%);

                            3         2
                         27 b  + 135 b  + 225 b + 125
(%o75)                   ----------------------------
                                      3
                                     x


(%i76) factor(%);

                                            3
                                     (3 b + 5)
(%o76)                               ---------
                                          3
                                         x
```

Funktionen zur Vereinfachung von Ausdrücken

Es gibt keinen globalgalaktischen simplify Befehl. Für verschiedene Gebiete stehen spezifische Vereinfachungsfunktionen zur Verfügung. Hier eine Auswahl. Im Abschnitt "Wichtige Maxima Funktionen" sind Kurzbeschreibungen dieser Befehle verfügbar.

- `fullratsimp`, `ratsimp`, `radcan`,
- `grind`,
- `trigsimp`, `trigreduce`, `trigexpand`, `foursimp`,
- `atensimp`, `vectorsimp`.

Es gibt zusätzlich Befehle zur benutzerdefinierten Vereinfachung.

3.3. Lösen von Gleichungssystemen

Maxima ist in der Lage, exakte Lösungen auch von nichtlinearen algebraischen Gleichungssystemen zu berechnen. Im folgenden Beispiel werden 3 Gleichungen nach drei Unbekannten aufgelöst:

```
(%i77) eq1: a + b + c = 6;

(%o77)                           c + b + a = 6


(%i78) eq2: a * b + c = 5;

(%o78)                            c + a b = 5


(%i79) eq3: a + b * c = 7;

(%o79)                            b c + a = 7
```

```
(%i93) s: solve([eq1, eq2, eq3], [a, b, c]);

(%o93)          [[a = 1, b = 3, c = 2], [a = 1, b = 2, c = 3]]
```

Die Lösung wird in Form einer Liste, welche mit eckigen Klammern umschlossen ist, dargestellt. Im Folgenden wird gezeigt, wie auf die Elemente einer Liste zugegriffen wird und wie man Lösungen in Gleichungen einsetzt.

```
(%i94) s[1];

(%o94)                       [a = 1, b = 3, c = 2]


(%i95) s[2];

(%o95)                       [a = 1, b = 2, c = 3]


(%i98) eq4: a * a + 2 * b * b + c * c;

                                 2      2    2
(%o98)                          c  + 2 b  + a


(%i99) eq4, s[1];

(%o99)                                 23


(%i100) eq4, s[2];

(%o100)                                18
```

3.4. Trigonometrische Funktionen

Es stehen u.a. `tan, sin, cos, tanh, sinh, cosh` und deren Umkehrfunktionen zur Verfügung.

```
(%i102) example(trig);


(%i103) tan(%pi/6)+sin(%pi/12)
                                    %pi       1
(%o103)                        sin(---) + -------
                                    12     sqrt(3)


(%i104) ev(%,numer)
(%o104)                        0.83616931429214658


(%i105) sin(1)
(%o105)                              sin(1)
```

```
(%i106) ev(sin(1),numer)
(%o106)                        0.8414709848078965


(%i107) beta(1/2,2/5)
                                      1  2
(%o107)                            beta(-, -)
                                      2  5


(%i108) ev(%,numer)
(%o108)                        3.6790939804058804


(%i109) diff(atanh(sqrt(x)),x)
                                       1
(%o109)                          -----------------
                                 2 (1 - x) sqrt(x)


(%i110) fpprec:25


(%i111) sin(5.0B-1)
(%o111)                  4.7942553860420300027328793B-1


(%i112) cos(x)^2-sin(x)^2
                                   2         2
(%o112)                          cos (x) - sin (x)


(%i113) ev(%,x:%pi/3)
                                     1
(%o113)                            - -
                                     2


(%i114) diff(%th(2),x)
(%o114)                          - 4 cos(x) sin(x)


(%i115) integrate(%th(3),x)
                           sin(2 x)            sin(2 x)
                           -------- + x   x - --------
                              2                   2
(%o115)                    ------------ - ------------
                                2               2
(%i116) expand(%)
                                sin(2 x)
(%o116)                         --------
                                   2
```

Trigonometrische Ausdrücke lassen sich in Maxima leicht manipulieren. Die Funktion `trigexpand` benutzt die Summe-der-Winkel-Funktion, um Argumente innerhalb jeder trigonometrischen Funktion so stark wie möglich zu vereinfachen.

```
                                  sin(2 x)
(%o116)                           --------
                                     2
```

```
(%i117) trigexpand(%)
(%o117)                        cos(x) sin(x)
```

Die Funktion `trigreduce` konvertiert einen Ausdruck in eine Summe von Einzeltermen, bestehend aus jeweils einer sin- oder cos- Funktion.

```
(%o117)                        cos(x) sin(x)
```

```
(%i118) trigreduce(%)
                                  sin(2 x)
(%o118)                           --------
                                     2
```

```
(%i119) sech(x)^2*tanh(x)/coth(x)^2+cosh(x)^2*sech(x)^2*tanh(x)/coth(x)^2
                                  +sech(x)^2*sinh(x)*tanh(x)/coth(x)^2

           2                          2        2                  2
        sech (x) sinh(x) tanh(x)   cosh (x) sech (x) tanh(x)   sech (x) tanh(x)
(%o119) ------------------------ + ------------------------- + ----------------
                    2                          2                       2
                 coth (x)                   coth (x)                coth (x)
```

`sech()` ist eine hyperbolische Sekantenfunktion. `trigsimp` ist eine Vereinfachungsroutine, welche verschiedene trigonometrische Funktionen in sin und cos Equivalente umwandelt.

```
(%i120) trigsimp(%)

                          5          4             3
                       sinh (x) + sinh (x) + 2 sinh (x)
(%o120)                ---------------------------------
                                    5
                                cosh (x)
```

`exponentialize()` transformiert trigonometrische Funktionen in ihre komplexen Exponentialfunktioen.

```
(%i121) ev(sin(x),exponentialize)
                                  %i x     - %i x
                            %i (%e     - %e      )
(%o121)                   - ----------------------
                                     2
```

`taylor(Funktion, Variable, Entwicklungspunkt, Grad);` Generiert eine Taylorreihenentwicklung der angegebenen Funktion nach einer Variable um den Entwicklungspunkt bis einschließlich des angegebenen Grades.

```
(%i122) taylor(sin(x)/x,x,0,4)

                                  2    4
                                 x    x
(%o122)/T/                  1 - -- + --- + . . .
                                 6    120
```

```
(%i123) ev(cos(x)^2-sin(x)^2,sin(x)^2 = 1-cos(x)^2)

                                     2
(%o123)                         2 cos (x) - 1

(%o123)                             done
```

3.5. Komplexe Zahlen

Die Funktionen `realpart` und `imagpart` geben den Real- bzw. Imaginärteil eines komplexen Ausdruckes zurück.

```
(%i144) z: a + b * %i;

(%o144)                              %i b + a
```

```
(%i145) z^2;

                                           2
(%o145)                           (%i b + a)
```

```
(%i146) exp(z);

                                     %i b + a
(%o146)                             %e
```

`trigrat()` transformiert (komplexe) Exponentialfunktionen in entsprechende `sin()` und `cos()` Funktionen um.

```
(%i148) trigrat(exp(z));

                                 a            a
(%o148)                   %i %e  sin(b) + %e  cos(b)
```

Komplexe Zahlen lassen sich mit `imagpart()` und `realpart()` in die entsprechenden Real- und Imaginärteile aufspalten:

```
(%i149) imagpart(%);
```

```
                                    a
(%o149)                          %e  sin(b)


(%i150) realpart(%th(2));
```

3.6. Ableitungen, Grenzwerte, Integrale

Mit Maxima lassen sich u. a. Ableitungen, Integrale, Taylorentwicklungen, Grenzwerte, exakte Lösungen gewöhnlicher Differentialgleichungen berechnen.
Zunächst definieren wir ein Symbol f als Funktion von x auf 2 Arten. Beachten Sie die Unterschiede bei der Auswertung der Ableitung.

```
(%i129) f : x^3;

                                     3
(%o129)                             x


(%i130) diff(f,x);

                                    2
(%o130)                          3 x


(%i131) kill(f);

(%o131)                          done


(%i132) f(x) := x^3;

                                        3
(%o132)                          f(x) := x


(%i133) diff(f,x);

(%o133)                             0


(%i134) diff(f(x),x);

                                    2
(%o134)                          3 x
```

Ein Beispiel für eine Taylorreihenentwicklung und eine Grenzwertberechnung:

```
(%i140) f(x) := sin(x) / x;

                                      sin(x)
(%o140)                       f(x) := ------
                                        x
```

```
(%i141) taylor(f(x),x,0,5);

                                    2    4
                                   x    x
(%o141)/T/                   1 - -- + --- + . . .
                                   6    120


(%i142) limit(f(x),x,0);

(%o142)                                1
```

Integrale lassen sich, sofern möglich, bestimmt und unbestimmt berechnen:

```
(%i109) integrate(%e^x/(2+%e^x),x)
                                        x
(%o109)                          log(%e  + 2)


(%i116) integrate(x^(5/4)/(1+x)^(5/2),x,0,inf)

                                        9  1
(%o116)                           beta(-, -)
                                        4  4
```

3.7. Lösen von Differentialgleichungen

Ableitungen bzw. Differentiale können so eingegeben werden, dass sie ausgewertet oder nicht ausgewertet werden. Das Apostroph wirkt als Maskierung und verhindert die Auswertung durch Maxima:

```
(%i9) 'diff (y, x);
                                 dy
(%o9)                            --
                                 dx


(%i10) diff (y, x);
(%o10)                            0
```

Zum Lösen von gewöhnlichen Differentialgleichungen stehen folgende Funktionen zur Verfügung: `ode2, ic1, ic2, bc1, bc2.`
`ic1, ic2` sind auf Anfangswertaufgaben 1. bzw. 2. Ordnung spezialisiert.
`bc1, bc2` sind auf Randwertaufgaben 1. bzw. 2. Ordnung spezialisiert.

```
(%i23) dgl1: -'diff(y,x) * sin(x) + y * cos(x) = 1;

                                          dy
(%o23)                 cos(x) y - sin(x) -- = 1
                                          dx
```

```
(%i24) ode2( dgl1, y, x);

                                        1
(%o24)                  y = sin(x) (------ + %c)
                                     tan(x)


(%i25) trigsimp(%);

(%o25)                   y = %c sin(x) + cos(x)
```

Anfangswertaufgabe: Harmonische Schwingungen z.B. eines Pendels werden durch folgende Differentialgleichung beschrieben und mittels `ode2` und `ic2` gelöst:

```
(%i38) dgl2: 'diff(y,x,2) + y = 0;

                                  2
                                 d y
(%o38)                           --- + y = 0
                                   2
                                 dx


(%i39) ode2(dgl2, y, x);

(%o39)                  y = %k1 sin(x) + %k2 cos(x)


(%i40) ic2(%, x=0, y=y0, 'diff(y,x)=0);

(%o40)                          y = cos(x) y0
```

Randwertaufgabe: Bei gleichmäßiger Belastung lässt sich die Biegelinie eines auf 2 Sützen ruhenden Balkens unter bestimmten Umständen durch folgende Differentialgleichung beschreiben und mittels `ode2` und `bc2` lösen:

```
(%i41) dgl3: 'diff(y,x,2) = x -  x^2;

                                  2
                                 d y        2
(%o41)                           --- = x - x
                                   2
                                 dx


(%i42) ode2(dgl3, y, x);

                                4      3
                               x  - 2 x
(%o42)                 y = - --------- + %k2 x + %k1
                                 12


(%i43) bc2(%, x=0, y=0, x=1, y=0);
```

```
                                        4      3
                                       x  - 2 x    x
(%o43)                          y = - --------- - --
                                         12       12
```

```
(%i45) expand(%);

                                        4    3
                                       x    x    x
(%o45)                           y = - -- + -- - --
                                       12   6    12
```

3.8. Matrizenrechnung

Mit Maxima lassen sich allgemeine Matrizenoperationen durchführen.

```
(%i79) m:matrix([a,0],[b,1])
                                      [ a  0 ]
(%o79)                                [      ]
                                      [ b  1 ]
```

```
(%i80) m^2
                                      [  2     ]
                                      [ a   0  ]
(%o80)                                [        ]
                                      [  2     ]
                                      [ b   1  ]
```

```
(%i81) m . m
                                    [    2       ]
(%o81)                              [   a      0 ]
                                    [             ]
                                    [ a b + b  1  ]
```

```
(%i82) m[1,1]*m
                                     [  2     ]
(%o82)                               [ a    0 ]
                                     [        ]
                                     [ a b  a ]
```

```
(%i83) 1-%th(2)+%
                                    [   1     1 ]
(%o83)                              [           ]
                                    [ 1 - b   a ]
```

```
(%i84) m^^(-1)
```

```
                                  [  1     ]
                                  [  -   0 ]
                                  [  a     ]
(%o84)                            [        ]
                                  [   b    ]
                                  [ - -  1 ]
                                  [   a    ]
```

```
(%i85) [x,y] . m

                           [ b y + a x  y ]
```

```
(%i86) matrix([a,b,c],[d,e,f],[g,h,i])

                                  [ a  b  c ]
                                  [         ]
(%o86)                            [ d  e  f ]
                                  [         ]
                                  [ g  h  i ]
```

```
(%i87) %^^2
             [                2                                  ]
             [ c g + b d + a    c h + b e + a b  c i + b f + a c ]
             [                                                  ]
(%o87)       [                         2                       ]
             [ f g + d e + a d  f h + e  + b d   f i + e f + c d ]
             [                                                  ]
             [                                      2           ]
             [ g i + d h + a g  h i + e h + b g  i  + f h + c g  ]
(%o87)                           done
```

Außerdem lassen sich u. a. die Determinante, die Inverse, die Eigenwerte und -vektoren einer Matrix berechnen. Die Matrix darf dabei auch symbolische Ausdrücke enthalten. `eigenvalues(m)` ergibt als Ergebnis eine Liste, bestehend aus 2 Unterlisten. Die erste Unterliste enthält die Eigenwerte, die zweite Unterliste die entsprechenden Multipliktaroren.

```
(%i60) m : matrix( [1, 0, 0], [0, 2, 0], [0, 0, 3]);

                                  [ 1  0  0 ]
                                  [         ]
(%o60)                            [ 0  2  0 ]
                                  [         ]
                                  [ 0  0  3 ]
```

```
(%i61) eigenvalues(m);

(%o61)                     [[1, 2, 3], [1, 1, 1]]
```

Die Funktion `eigenvectors` berechnet Eigenwerte, deren Multiplikatoren sowie die Eigenvektoren der gegebenen Matrix. Die Ergebnisse werden in Listen bzw. Unterlisten zusammengefasst. Es gibt verschiedene Möglichkeiten die Auswertung zu beeinflussen `nondiagonalizable, hermitianmatrix, knowneigvals`. Diese werden mit `describe(eigenvectors);` beschrieben.

```
(%i62) eigenvectors(m);

(%o62)     [[[1, 2, 3], [1, 1, 1]], [1, 0, 0], [0, 1, 0], [0, 0, 1]]


(%i71) part( %, 2);

(%o71)                              [1, 0, 0]
```

Weiterhin gibt es Funktionen zur Transponierung (`transpose`), Berechnung der Determinante (`determinant`), Berechnung des charakteristischen Polynomes `charpoly(Matrix, Variable)`, Berechnung der Inversen (`invert`), etc. Das Schlüsselwort `detout` faktorisiert dabei die Determinante aus der Inversen.

4. Programmieren in Maxima

Bis jetzt haben Sie gesehen, wie man Maxima im interaktiven Modus wie einen Taschenrechner benutzt. Für Berechnungen, welche wiederholt Kommandosequenzen durchlaufen müssen, sind Programme geeigneter.
Programme werden gewöhnlich in einem Texteditor (Emacs) geschrieben und dann mittels `batch` in Maxima geladen.
Ein kleines Statistik-Münzwurfprogramm. Mit `block( [Lokale_Variablen], Kommando1, Kommando2...)` wird ein Block von Kommandos definiert. Die lokalen Variablen können auch mit Startwerten versehen werden, wie im Folgenden gezeigt wird:

```
(%i13)  muenze_werfen(n) := block( [muenze, statistik:[0,0] ],
          /* Kommentare wie in C/C++ */
          print("Ich werde die Münze ", n, "mal werfen. 1 = Zahl, 2 = Kopf"),
          for i: 1 thru n do (
            /* eine for schleife, für genauere informationen siehe DO */
            muenze : random(2) + 1,
            /* 1 + Zufallsgenerator von 0 bis 1  */
            /* Array-Indizes beginnen mit EINS! */
          statistik[muenze] : statistik[muenze] + 1 ),
          print("Zahl wurde ", statistik[1], "mal geworfen"),
          print("Kopf wurde ", statistik[2], "mal geworfen"),
         n );

(%o13) muenze_werfen(n) := block([muenze, statistik : [0, 0]],

       print("Ich werde die Münze ", n, "mal werfen. 1 = Zahl, 2 = Kopf"),

       for i thru n do (muenze : random(2) + 1,

          statistik       : statistik       + 1),
                   muenze            muenze

          print("Zahl wurde ", statistik , "mal geworfen"),
                                       1

          print("Kopf wurde ", statistik , "mal geworfen"), n)
                                       2



(%i14) muenze_werfen(1000);

       Ich werde die Münze  1000 mal werfen. 1 = Zahl, 2 = Kopf
       Zahl wurde  485 mal geworfen
       Kopf wurde  515 mal geworfen

(%o14)                               1000
```

Ausgaben in die Datei `Daten.txt` umlenken (s.a. Anhang):

```
muenze_werfen(n) := block( [muenze, statistik:[0,0] ], /* Kommentare wie in
```

```
C/C++ */
  with_stdout( "Daten.txt",
    print("Ich werde die Münze ", n, "mal werfen. 1 = Zahl, 2 = Kopf"),
    for i: 1 thru n do (
      muenze : random(2) + 1,
      statistik[muenze] : statistik[muenze] + 1 ),
    print("Zahl wurde ", statistik[1], "mal geworfen"),
    print("Kopf wurde ", statistik[2], "mal geworfen")),n );
```

5. Weitere Hilfen im Internet

Es gibt eine sehr aktive englischsprachige Mailingliste. Archive dieser Liste sind im Netz verfuegbar und durchsuchbar. Dies ist die offizielle Seite des Archivs eine andere (bedienerfreundlichere) Moeglichkeit bietet Gmane hier .
Im IRC habe ich vor einiger Zeit einen Channel eingerichtet, dort treiben sich immer haeufiger einige Benutzer herum: server irc.freenode.net, port 6667, channel #maxima.
Weitere Dokumentationen befinden sich hier .

6. Verschiedenes, Tipps

6.1. Fehlerhafte Auswertung einer Funktion ignorieren

Folgende Funktion wertet eine Funktion der Form `f(x):=...` aus. Wenn ein Fehler auftritt, wird ein Fragezeichen zurückgegeben:

```
try1D(func1D, x) := block([result],
  result : ev( errcatch( func1D(x)), numer),
  if( length(result) < 1) then( "?") else (first(result)));
```

Dies ist nützlich, wenn Funktionen mittels Programmstrukturen definiert wurden (z. B. `if`) und diese Funktionen in Plots, Integrationen usw. verwendet werden sollen.

6.2. Dateiausgabe mit variablem Dateinamen

Robert Dodier gab auf der Maxima-Mailingliste eine Lösung für dieses Problem:

```
(%i1) my_with_stdout (fn, [L]) ::= buildq ([fn:ev(fn), L], with_stdout(fn,
splice (L)));

(%o1) my_with_stdout(fn, [L]) ::= buildq([fn : ev(fn), L],
                                          with_stdout(fn, splice(L)))

(%i2)  F (namebase, n) := my_with_stdout (concat (namebase, n), print (foo, bar,
baz));

(%o2) F(namebase, n) := my_with_stdout(concat(namebase, n),
                                       print(foo, bar, baz))

(%i3)  F ("/tmp/foo", 3);

(%o3)                                baz
```

Die Datei `/tmp/foo3` enthält nun `foo bar baz .`

6.3. Warnung "unbekanntes Symbol" für globale Variablen vermeiden

Setzt man Variablen nur mit `var : 42;` so kann es zu Warnungen kommen, dass diese Variable nicht bekannt sei (z. B. bei numerischen Integrationen). Dies kann vermieden werden, wenn man bei der Defintion `define_variable( variable, (Wert)), Typ)` verwendet:

```
(%i4) define_variable (C_MAX,   (0.31),               float);
```

```
(%o4)                                0.31
```

6.4. Formelexport nach OpenOffice.org

Dieter Schuster hat eine modifizierte Version der Datei mactex.lisp (`mactex2ooo.lisp`) geschrieben: Openoffice.org-Export. Die Lisp-Datei wird mittels `load` geladen, danach steht der Befehl `ooo(...)` zur Verfügung. Diese Funktion ersetzt die `tex(...)` Funktion. Die Ausgabe kann in OpenOffice.org in einem Formelobjekt verwendet werden.

```
(%i205) load("mactex2ooo.lisp");

(%o205)                          mactex2ooo.lisp


(%i212) %ny^2 = A_i * %pi + f(x) / f( x + %my);

                                2       f(x)
(%o212)                       %ny  = ---------- + %pi A_i
                                      f(x + %my)


(%i213) ooo(%);
%ny^2 ={alignc {f left( x right) } over {f left( x + %my right) }} + %pi cdot
A_i
```

In OpenOffice.org sieht die Formel dann so aus:

$$\nu^2 = \frac{f\left(x\right)}{f\left(x+\mu\right)} + \pi \cdot A_i$$

6.5. Emacs

Emacs bietet per Tastenkombination sehr hilfreiche Funktionen. Es ist möglich, eine Datei zu editieren und deren Inhalt Portionsweise an einen Maximaprozess zu senden. Es kann die Hilfe zu einem Symbol eingeblendet werden.

Maxima Modus

Der Hauptmodus zum Editieren von Maxima-Code.
`C-M-a`: Gehe zum Begin eines Bereichs.
`C-M-e`: Gehe zum Ende eines Bereichs.
`C-M-b`: Gehe zum Beginn einer sexp.
`C-M-f`: Gehe zum Ende einer sexp.

`M-TAB`: Vervollständige ein Maxima-Symbol soweit möglich. Bei Mehrdeutigkeit werden

die Möglichkeiten aufgelistet.
Teile einer Datei können zu einem Maxima-Prozess gesendet werden. Existiert noch kein Prozess, so wird ein neuer Prozess gestartet. Wird ein Zusatzargument angegeben, so wird auf die Überprüfung der Klammern verzichtet.
`C-c C-r`: Sende die Markierung (Region) zu Maxima.
`C-c C-b`: Sende den Buffer (Dateiinhalt) zu Maxima.
`C-c C-c`: Sende die Zeile zu Maxima.
`M-x maxima-send-form`: Sende den Bereich zu Maxima.
`C-return`: Sende den kleinsten auswertbaren Teilbereich zu Maxima und gehe zum nächsten Bereich.
`M-return`: Wie oben, aber mit der gesamten Region.
`C-c C-l`: Fragt nach einem Dateinamen und lädt die Datei zu Maxima.

`C-c C-h`: Hilfe zu einem Maxima-Gebiet.
`C-c RET`: Öffnet die Maxima-Hilfe.
`M-x maxima-completion-help`: Sucht interaktiv die passende Hilfeseite.
`M-x maxima-apropos-help.`: Für Hilfe passend zur Cursorposition.

Infix-Prefix Konversion

Da ich mittlerweile Lisp-infiziert bin, benutze ich Maxima auch zur Infix-Prefix Konversion von Formeln. Ein einfacher (noch nicht vollständiger) Weg ist der folgende:

```
with_stdout( "lisp-translation-temp.max",
  print(sconcat( cylcase2_47a, ";")));

translate_file( "lisp-translation-temp.max");
```

In der Datei "lisp-translation-temp.max" steht:

```
 c1 = -C*R1/A+D-B*C/A-dxx*C/A+dzz;
```

In der Datei "lisp-translation-temp.LISP" steht (u. a.):

```
[...]
  (SIMPLIFY
      (LIST '(MEQUAL) (TRD-MSYMEVAL $C1 '$C1)
            (ADD* (DIV (MUL* (*MMINUS (TRD-MSYMEVAL |$c| '|$c|))
                             (TRD-MSYMEVAL |$r1| '|$r1|))
                       (TRD-MSYMEVAL |$a| '|$a|))
                  (TRD-MSYMEVAL |$d| '|$d|)
                  (DIV (MUL* (*MMINUS (TRD-MSYMEVAL |$b| '|$b|))
                             (TRD-MSYMEVAL |$c| '|$c|))
                       (TRD-MSYMEVAL |$a| '|$a|))
                  (DIV (MUL* (*MMINUS (TRD-MSYMEVAL $DXX '$DXX))
                             (TRD-MSYMEVAL |$c| '|$c|))
                       (TRD-MSYMEVAL |$a| '|$a|))
                  (TRD-MSYMEVAL $DZZ '$DZZ))))
```

Ziel ist es, diesen Teil so in Lisp zu übersetzen, dass man die Formel in eigenen Lisp-Programmen verwenden kann. Dazu führt man einige Ersetzungen mittels regulärer Ausdrücke durch. Hier ein Beispielprogramm in Emacs-Lisp welches diese Aufgabe übernimmt.

```
  (defun robert-maxima-translation-converter ()
    "do replace-regexp so that a translated file will (hopefully) in (normal)
lisp
     infix -> prefix converter
   usage:  (1) write formulas to file (with_stdout, sconcat )
           (2) translate_file (filename) -> filename.LISP
           (3) open filename.LISP, run robert-maxima-translation-converter"
    (interactive)
    (save-excursion
      (beginning-of-buffer)
      (dolist (regexp (list
                       (list "(trd-msym.*? $\\(.*?\\) '$\\(.*?\\))" "\\1")
                       (list "(trd-msym.*? |$\\(.*?\\)| '|$\\(.*?\\)|)" "\\1")
                       (list "(add\\*" "(+")
                       (list "(mul\\*" "(*")
                       (list "(div" "(/")
                       (list "(\\*mminus" "(-")))
        (save-excursion
          (replace-regexp (first regexp) (second regexp) nil)))
      (dotimes  (i 10)
        (save-excursion
          (replace-regexp "  " " ")))
      (indent-region (point-min) (point-max))))
```

Nachdem man `robert-maxima-translation-converter` in "lisp-translate-temp.LISP" aufgerufen hat sieht die Formel so aus:

```
[...]
  (SIMPLIFY
   (LIST '(MEQUAL) C1
         (+ (/ (* (- c)
                  r1)
               a)
          d
          (/ (* (- b)
                c)
             a)
          (/ (* (- DXX)
                c)
             a)
          DZZ)))
```

Jetzt sind nur noch geringfügige Anpassungen notwendig, um diesen Ausdruck in einem Lisp-Programm zu verwenden.

7. Wichtige Maxima-Funktionen

Die Referenz befindet sich unter `doc/html/maxima_toc.html` (im Installationspfad von Maxima). In Maxima können Sie Hilfe durch die Kommandos `describe`, `apropos` und `example` erhalten. `apropos(Stichwort)` gibt eine Liste von evtl. geeigneten Kommandos. `describe(Befehl)` beschreibt die einzelnen Kommandos genauer (evtl. müssen Sie eine Auswahl treffen, welcher Aspekt genauer beschrieben werden soll) und `example(Befehl)` gibt Beispiele für den jeweiligen Befehl aus (soweit vorhanden).

`allroots(a)` Findet alle (allgemein komplexen) Wurzeln einer Polynomialgleichung.
`append(a,b)` Fügt Liste b an a an.
`apropos(a);` Liefert zu einem Stichwort mögliche Befehle/Funktionen.
`atensimp(a>)` Vereinfacht algebraische Tensorausdrücke.
`batch(a)` Lädt und startet Programm/File a.
`bc1, bc2 ( DGL, x=x0, y=y0, x=x1, y=y1)` Lösung einer Randwertaufgabe einer DGL nach Behandlung mit ode2.
`charpoly(Matrix, Variable)` Berechnet das charakteristische Polynom einer Matrix bzgl. der gegebenen Variable.
`coeff(a,b,c)` Koeffizienten von b der Potenz C in Ausdruck a.
`concat(a,b)` Generiert ein Symbol ab.
`cons(a,b)` Fügt a in Liste b als erstes Element ein.
`demoivre(a)` Transformiert alle komplexen Exponentialterme in trigonometrische.
`denom(a)` Nenner von a.
`depends(a,b)` Erklärt a als Funktion von b (nützlich für Differentialgleichungen).
`desolve(a,b)` Versucht ein lineares System a von gew. DGLs nach unbekannten b mittels Laplace-Transformation zu lösen.
`describe(a)` Beschreibt einen Befehl oder eine Funktion näher. Evtl. wird nachgefragt, welcher Aspekt eines Befehls oder einer Befehlsgruppe näher beschrieben werden soll.
`determinant(a)` Determinante
`diff(a,b1,c1,b2,c2,...,bn,cn)` Gemischte partielle Ableitung von a nach bi der Stufe ci.
`eigenvalues(a)` Berechnet die Eigenwerte und ihre Multiplikatoren.
`eigenvectors(a)` Berechnet Eigenvektoren, Eigenwerte und Multiplikatoren.
`entermatrix(a,b)` Matrixeingabe
`ev(a,b1,b2,...,bn)` Berechnet Ausdruck a unter Annahmen bi (Gleichungen, Zuweisungen, Schlüsselwörter (numer - Zahlenwerte, detout - Matrixinverse ohne Determinante, diff - alle Ableitungen werden ausgeführt). NUR bei direkter Eingabe kann ev weggelassen werden.
`example(a)` Zeigt Beispiele für die Verwendung eines Befehls oder einer Funktion an. Nicht für jede Funktion sind Beispiele vorhanden.
`expand(a)` Algebraische Expansion (Distribution).
`exponentialize(a)` Transformiert trigonometrische Funktionen in ihre komplexen Exponentialfunktioen.

`factor(a)` Faktorisiert a.
`fortran(a)` Konvertiert einen Ausdruck in einen Fortran-Ausdruck, soweit möglich.
`foursimp(a)` Vereinfacht trigonometrische Funktionen, welche (ganzzahlige) Vielfache von Pi enthalten in Abhängigkeit verschiedener Flags.
`freeof(a,b)` Ergibt wahr, wenn b nicht a enthält.
`fullratsimp(a)` Wiederholte Ausführung von ratsimp, gefolgt von nicht-rationalen Vereinfachungen bis keine weitere Veränderung auftritt.
`grind(a)` Darstellung einer Variable oder Funktion in einer kompakten Form.
`ic1, ic2 ( dgl, x=x0, y=y0, dy0/dx = y1)` Lösung einer Anfangswertaufgabe einer DGL (nach Behandlung mit ode2).
`ident(a)` Einheitsmatrix a x a.
`imagpart(a)` Imaginärteil von a.
`integrate(a,b)` Berechnungsversuch des unbestimmten Integrals a nach b.
`integrate(a,b,c,d)` Berechnung des Integrals a nach b in den Grenzen b=c und b=d.
`invert(a)` Inverse der Matrix a.
`kill(a)` Vernichtet Variable/Symbol a oder alle Variablen/Symbole (all).
`limit(a,b,c)` Grenzwertbestimmung des Ausdrucks a für b gegen c.
`lhs(a)` Linke Seite eines Ausdrucks.
`loadfile(a)` Lädt eine Datei a und führt sie aus.
`makelist(a,b,c,d)` Generiert eine Liste von a(b) mit b=c bis b=d.
`map(a,b)` Wendet a auf b an.
`matrix(a1,a2,...,an)` Generiert eine Matrix aus Zeilenvektoren.
`num(a)` Zähler von a.
`ode2(a,b,c)` Löst gewöhnliche Differentialgleichungen 1. und 2. Ordnung a für b als Funktion von c.
`part(a,b1,..,bn)` Extrahiert aus a die Teile bi.
`playback(a)` Zeigt die a letzten Labels an, wird a weggelassen, so werden alle Zeilen zurückgespielt.
`print( a1, a2, a3, ... )` Zeigt die Auswertung der Ausdrücke an.
`rat(a)` Umwandlung von a in eine kanonische rationale Form.
`ratsimp(a)` Vereinfacht a und gibt einen Quotienten zweier Polynome zurück.
`radcan(a)` Vereinfacht Ausdrücke für log-, exp, Radikale in eine kanonische oder eine reguläre Form. Die Differenz gleicher Ausdrücke verschieden Aussehens kann so zu NULL vereinfacht werden.
`realpart(a)` Realteil von a
`rhs(a)` Rechte Seite einer Gleichung a.
`save(a,b1,..., bn)` Generiert eine Datei a (im Standardverzeichnis), welche Variablen, Funktionen oder Arrays bi enthält. So generierte Dateien lassen sich mit loadfile zurückspielen. Wenn b1 all ist, wird alles bis auf die Labels gespeichert.
`solve(a,b)` Algebraischer Lösungsversuch für ein Gleichungssystem oder eine Gleichung a für eine Variable oder eine Liste von Variablen b. Gleichungen können =0 abkürzen.
`string(a)` Konvertiert einen Ausdruck in Maximas lineare Notation.
`stringout(a,b1,..bn)` Generiert eine Datei a im Standardverzeichnis, bestehend aus

Symbolen bi. Die Datei ist im Textformat und nicht dazu geeignet von Maxima geladen zu werden. Die Ausdrücke können aber genutzt werden, um sie in Fortran, Basic oder C-Programmen zu verwenden.
`subst(a,b,c)` Ersetzt a für b in c.
`taylor(a,b,c,d)` Taylorreihenentwicklung von a nach b in Punkt c bis zur Ordnung d.
`translate_file(filename)` Übersetzt ein Maxima-File in ein LISP-File (Abschnitt Infix-Prefix Konversion).
`transpose(a)` Transponiert Matrix a.
`trigexpand(a)` Eine Vereinfachungsroutine, welche trigonometrische Winkelsummen nutzt, um einen Ausdruck zu vereinfachen.
`trigreduce(a)` Eine Vereinfachungsroutine für trigonometrische Produkte und Potenzen.
`trigsimp(a)` Eine Vereinfachungsroutine, welche verschiedene trigonometrische Funktionen in sin und cos Equivalente umwandelt.
`vectorsimp(a)` Wendet Vereinfachungen und Expansionen bzgl. Vektoroperationen in Abhängigkeit verschiedener globaler Flags an.
`with_stdout( Datei, Ausdrücke);` Leitet die Ausgabe der Ausdrücke in die angegebene Datei um. Hiebei kann kein variabler Dateinamen angegeben werden. Im Abschnit "Verschiedenes" wird ein verbessertes Makro vorgestellt.