DonnishJournals 2041-1189

Donnish Journal of Educational Research and Reviews.
Vol 1(2) pp. 030-035 December, 2014.
http://www.donnishjournals.org/djerr

Original Research Paper

Equations, Polynomials and Differential Equations with MAXIMA

Niyazi Ari and Savaş Tuylu*

Computer Science, Nigerian Turkish Nile University, Nigeria.

Accepted 3rd December, 2014.

The methods of calculus lie at the heart of the physical sciences and engineering. Maxima can help you make faster progress, if you are just learning calculus. The examples in this research paper will offer an opportunity to see some Maxima tools in the context of simple examples, but you will likely be thinking about much harder problems you want to solve as you see these tools used here. This research paper includes Equations, Polynomials and Differential Equations.

Keywords: Equations, Polynomials, Differential Equations, Algebraic Equations, Linear Equations, Laplace Transform, Boundary Value Problems, Maxima.

1. INTRODUCTION

Maxima is a system for the manipulation of symbolic and numerical expressions, including differentiation, integration, Taylor series, Laplace transforms, ordinary differential equations, systems of linear equations, polynomials, sets, lists, vectors, matrices, tensors, and more. Maxima yields high precision numeric results by using exact fractions, arbitrary precision integers, and variable precision floating point numbers. Maxima can plot functions and data in two and three dimensions.

Maxima source code can be compiled on many computer operating systems, including Windows, Linux, and MacOS X. The source code for all systems and precompiled binaries for Windows and Linux are available at the SourceForge file manager. Maxima is a descendant of Macsyma, the legendary computer algebra system developed in the late 1960s at the Massachusetts Institute of Technology. It is the only system based on that effort still publicly available and with an active user community, thanks to its open source nature. Macsyma was revolutionary in its day, and many later systems, such as Maple and Mathematica, were inspired by it.

The Maxima branch of Macsyma was maintained by William Schelter from 1982 until he passed away in 2001. In 1998 he obtained permission to release the source code under the GNU General Public License (GPL). It was his efforts and skills that made the survival of Maxima possible. MAXIMA has been constantly updated and used by researcher and engineers as well as by students.

2. EQUATIONS

The solution of simultaneous equations occurs frequently in applications of engineering mathematics. Nonlinear equations arise in a number of instances, such as optimization, eigenvalue calculations, simulations, etc.

Corresponding Author: savastyludf@hotmail.com

Table 1: MAXIMA uses for solving of equations the following functions:

Function	Description
solve *(expr, x)* **solve** *(expr)* **solve** *([eqn_1, …, eqn_n], [x_1, …, x_n])*	Solves the algebraic equation *expr* for the variable *x* and returns a list of solution equations in *x*. If *expr* is not an equation, the equation *expr* = 0 is assumed in its place. *x* may be a function (e.g. f(x)), or other nonatomic expression except a sum or product. *x* may be omitted if *expr* contains only one variable. *expr* may be a rational expression, and may contain trigonometric functions, exponentials, etc.
linsolve *([expr_1, …, expr_m], [x_1, …, x_n])*	Solves the list of simultaneous linear equations for the list of variables. The expressions must each be polynomials in the variables and may be equations.
funcsolve *(eqn, g(t))*	Returns [*g*(*t*) = ...] or [], depending on whether or not there exists a rational function *g*(*t*) satisfying *eqn*, which must be a first order, linear polynomial in (in this case) *g*(*t*) and *g*(*t*+1)

2.1. Examples

Example 2.1.1.

```
(%i1) eq1:x+y+z=a$
      eq2:x-y+z=b$
      eq3:x-y-z=c$
      solve([eq1,eq2,eq3],[x,y,z]);
      linsolve([eq1,eq2,eq3],[x,y,z]);
```

$$(\%o4)\quad \left[\left[x=\frac{c+a}{2},y=\frac{a-b}{2},z=\frac{b-c}{2}\right]\right]$$

$$(\%o5)\quad \left[x=\frac{c+a}{2},y=\frac{a-b}{2},z=\frac{b-c}{2}\right]$$

Example 2.1.2.

```
(%i6) eq:a*cos(x)-x/b;
      solve(eq,x);
```

$$(\%o6)\quad a\cos(x)-\frac{x}{b}$$

$$(\%o7)\quad [x=a\,b\cos(x)]$$

Example 2.1.3.

```
(%i8) eq:(n-1)*f(n)-(n+1)*f(n-1)=(n+1)/(n-1);
      funcsolve(eq,f(n));
```

$$(\%o8)\quad (n-1)f(n)-(n+1)f(n-1)=\frac{n+1}{n-1}$$

$$(\%o9)\quad f(n)=\frac{\%0\,n^2+\%1\,n+\%2}{(n-1)^2}$$

Example 2.1.4.

```
(%i10) eq:asin(sin(x)*(f(x)+1));
       solve(eq,x);
```

$$(\%o10)\quad \mathrm{asin}((f(x)+1)\sin(x))$$

solve: using arc-trig functions to get a solution
Some solutions will be lost.

$$(\%o11)\quad [x=0,f(x)=-1]$$

Example 2.1.5.

```
(%i12) eq:x^4-1;
       solve(eq,x);
```

$$(\%o12)\quad x^4-1$$

$$(\%o13)\quad [x=\%i,x=-1,x=-\%i,x=1]$$

Example 2.1.6.

```
(%i14) solve([x+y=1,5*x+3*y=2],[x,y]);
```

$$(\%o14)\quad \left[\left[x=-\frac{1}{2},y=\frac{3}{2}\right]\right]$$

3. POLYNOMIALS

3.1. Examples

Example 3.1.1.

```
(%i1) p:a^4*x^4+a^3*x^3+a^2*x^2+a^1*x+a^0;
      coeff(p,x^4);
```

$$(\%o1)\quad a^4x^4+a^3x^3+a^2x^2+ax+1$$

$$(\%o2)\quad a^4$$

Example 3.1.2.

```
(%i3) eq:(x^2-1);
      factor(eq);
```

$$(\%o3)\quad x^2-1$$

$$(\%o4)\quad (x-1)(x+1)$$

Example 3.1.3.

```
(%i5) eq:sin(x^2/(x^2+x))=exp((log(x)+1)^3-log(x)^2);
      ratsimp(eq);
```

$$(\%o5)\quad \sin\left(\frac{x^2}{x^2+x}\right)=\%e^{(\log(x)+1)^3-\log(x)^2}$$

$$(\%o6)\quad \sin\left(\frac{x}{x+1}\right)=x^3\,\%e^{\log(x)^3+2\log(x)^2+1}$$

Example 3.1.4.

```
(%i7) eq:(y^(n/2)+1)^3*(y^(n/2)-1)^3/(y^n-1);
      ratsimp(eq);
```

$$(\%o7)\quad \frac{(y^{n/2}-1)^3\,(y^{n/2}+1)^3}{y^n-1}$$

$$(\%o8)\quad \frac{y^{3n}-3y^{2n}+3y^n-1}{y^n-1}$$

```
(%i9) fullratsimp(%);
```

$$(\%o9)\quad y^{2n}-2y^n+1$$

Table 1: MAXIMA uses for polynomials the following selected functions:

Function	Description
coeff *(expr, x, n)* **coeff** *(expr, x)*	Returns the coefficient of *x^n* in *expr*, where *expr* is a polynomial or a monomial term in *x*. coeff(*expr*, *x^n*) is equivalent to coeff(*expr*, *x*, *n*). coeff(*expr*, *x*, 0) returns the remainder of *expr* which is free of *x*. If omitted, *n* is assumed to be 1. *x* may be a simple variable or a subscripted variable, or a subexpression of *expr* which comprises an operator and all of its arguments.
factor *(expr)* **factor** *(expr, p)*	Factors the expression *expr*, containing any number of variables or functions, into factors irreducible over the integers. factor (*expr*, *p*) factors *expr* over the field of rationals with an element adjoined whose minimum polynomial is *p*.
rat *(expr)* **rat** *(expr, x_1, …, x_n)*	Converts *expr* to canonical rational expression (CRE) form by expanding and combining all terms over a common denominator and cancelling out the greatest common divisor of the numerator and denominator, as well as converting floating point numbers to rational numbers within a tolerance of ratepsilon. The variables are ordered according to the *x_1*, …, *x_n*, if specified, as in ratvars.
fullratsimp *(expr)*	fullratsimp repeatedly applies ratsimp followed by nonrational simplification to an expression until no further change occurs, and returns the result.
algsys *([expr_1, …, expr_m], [x_1, …, x_n])* **algsys** *([eqn_1, …, eqn_m], [x_1, …, x_n])*	Solves the simultaneous polynomials *expr_1*, …, *expr_m* or polynomial equations *eqn_1*, …, *eqn_m* for the variables *x_1*, …, *x_n*. An expression *expr* is equivalent to an equation *expr* = 0. There may be more equations than variables or vice versa. algsys returns a list of solutions, with each solution given as a list of equations stating values of the variables *x_1*, …, *x_n* which satisfy the system of equations. If algsys cannot find a solution, an empty list [] is returned. The symbols %r1, %r2, …, are introduced as needed to represent arbitrary parameters in the solution; these variables are also appended to the list %rnum_list.
allroots *(expr)* **allroots** *(eqn)*	Computes numerical approximations of the real and complex roots of the polynomial *expr* or polynomial equation *eqn* of one variable.
realroots *(expr, bound)* **realroots** *(eqn, bound)* **realroots** *(expr)* **realroots** *(eqn)*	Computes rational approximations of the real roots of the polynomial *expr* or polynomial equation *eqn* of one variable, to within a tolerance of *bound*. Coefficients of *expr* or *eqn* must be literal numbers; symbol constants such as %pi are rejected
funcsolve *(eqn, g(t))*	Returns [*g*(*t*) = ...] or [], depending on whether or not there exists a rational function *g*(*t*) satisfying *eqn*, which must be a first order, linear polynomial in (for this case) *g*(*t*) and *g*(*t*+1)
horner *(expr, x)* **horner** *(expr)*	Returns a rearranged representation of *expr* as in Horner's rule, using *x* as the main variable if it is specified. x may be omitted in which case the main variable of the canonical rational expression form of *expr* is used.

Example 3.1.5.

```
(%i10) eq1:2*x*a-2*x*b^3;
       eq2:a*x-2*b^3*x;
       eq3:a*(-y-x^2+1);
       eq4:b^3*(y-x^2);
       algsys([eq1,eq2,eq3,eq4],[x,y,a,b]);
(%o10) 2 a x-2 b^3 x
(%o11) a x-2 b^3 x
(%o12) a(-y-x^2+1)
(%o13) b^3(y-x^2)
(%o14) [[x=%r1,y=%r2,a=0,b=0],[x=0,y=1,a=%r3,b=0],[x=0,
=0,a=0,b=%r4]]
```

Example 3.1.6.

```
(%i15) eq:(2*x^2-1);
       allroots(eq);
(%o15) 2 x^2-1
(%o16) [x=0.70710678118655,x=-0.70710678118655
```

Example 3.1.7.

```
(%i17) eq:3*x+2*x^3-3;
       realroots(eq);
(%o17) 2 x^3+3 x-3
(%o18) [x=24667181/33554432]
```

Example 3.1.8.

```
(%i19) eq:(2-x)^7*(4-x)^3*(3-x);
       realroots(eq,1e-6);
(%o19) (2-x)^7 (3-x)(4-x)^3
(%o20) [x=2,x=4,x=3]
```

Example 3.1.9.

```
(%i21) eq:a*y^5+b*y^2+c*y+d;
       horner(eq,y);
```

(%o21) $a\,y^5+b\,y^2+c\,y+d$

(%o22) $y\left(y\left(a\,y^3+b\right)+c\right)+d$

4. Differential Equations

It is available in Maxima to obtain analytical solutions for some specific types of first and second order equations.

4.1. Examples

Example 4.1.1.

```
(%i1) eq:'diff(f,x)+f/x^2=0;
      ode2(%,f,x);
```

(%o1) $\frac{f}{x^2}+\frac{d}{d\,x}\,f=0$

(%o2) $f=\%c\,\%e^{1/x}$

Example 4.1.2.

```
(%i3) eq:'diff(f,x)+f/x^2=1;
      ode2(%,f,x);
      ic1(%o2,x=1,f=0);
```

(%o3) $\frac{f}{x^2}+\frac{d}{d\,x}\,f=1$

(%o4) $f=\left(\%c+\mathrm{gamma\_incomplete}\left(-1,\frac{1}{x}\right)\right)\%e^{1/}$

(%o5) $f=0$

Example 4.1.3.

```
(%i6) 'diff(f,x,2)+f*'diff(f,x)^2=0;
      ode2(%,f,x);
      ic2(%,x=1,f=1,'diff(f,x)=1);
```

(%o6) $\frac{d^2}{d\,x^2}\,f+f\left(\frac{d}{d\,x}\,f\right)^2=0$

(%o7) $-\frac{\sqrt{\pi}\,\%i\,\mathrm{erf}\left(\frac{\%i\,f}{\sqrt{2}}\right)}{\sqrt{2}\,\%k1}=x+\%k2$

(%o8) $-\frac{\sqrt{\pi}\,\%i\,\mathrm{erf}\left(\frac{\%i\,f}{\sqrt{2}}\right)}{\sqrt{2}\,\sqrt{\%e}}=x-\frac{\%e^{-1}\left(\sqrt{2}\,\sqrt{\%e}\,\sqrt{\pi}\,\%i\,\mathrm{erf}\left(\frac{\%i}{\sqrt{2}}\right)+2\,\%\epsilon\right.}{2}$

Example 4.1.4.

```
(%i9) eq1:'diff(f(x),x)=cos(x);
      desolve(eq1,f(x));
```

(%o9) $\frac{d}{d\,x}\,f(x)=\cos(x)$

(%o10) $f(x)=\sin(x)+f(0)$

Example 4.1.5.

```
(%i11) eq1:'diff(f(x),x,2)=cos(x);
       eq2:'diff(g(x),x,2)=cos(x);
       desolve([eq1,eq2],[f(x),g(x)]);
```

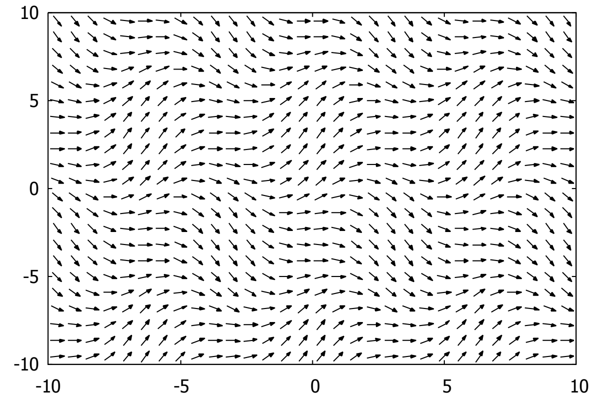
(%o11) $\frac{d^2}{d\,x^2}\,f(x)=\cos(x)$

(%o12) $\frac{d^2}{d\,x^2}\,g(x)=\cos(x)$

(%o13) $[f(x)=x\left(\frac{d}{d\,x}\,f(x)\Big|_{x=0}\right)-\cos(x)+f(0)+1,\,g(x)=x\left(\frac{d}{d\,x}\,g(x)\Big|_{x=0}\right)-\cos($
$+g(0)+1]$

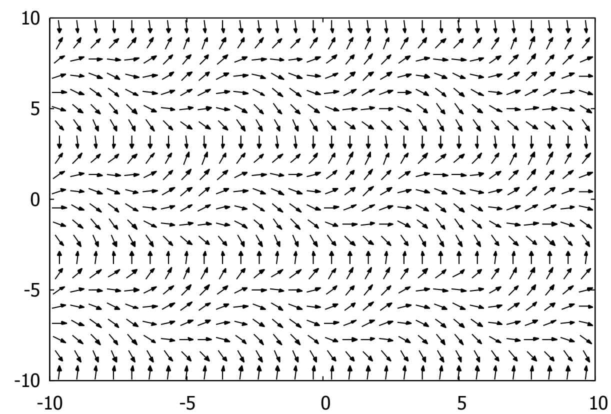
Example 4.1.6.

```
(%i14) load(drawdf)$
       eq1:cos(x)+sin(y/2)$
       drawdf(eq1);
(%o16) 0
```



Example 4.1.7.

```
(%i20) load(drawdf)$
       eq1:sin(x)+tan(y/2)$
       drawdf(eq1,[x,y]);
(%o22) 0
```



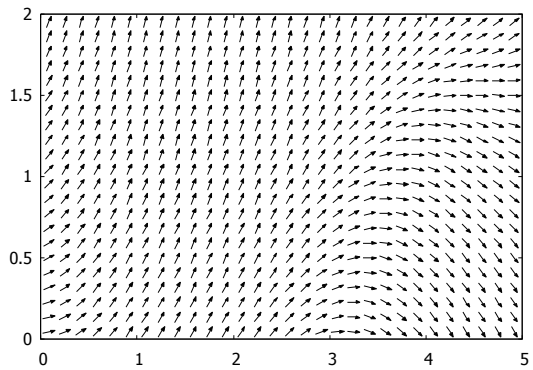
5. Table 1: MAXIMA uses for differential equations the following the used functions:

Function	Description
desolve *(eqn, x)* **desolve** *([eqn_1, ..., eqn_n], [x_1, ...,x_n])*	The function desolve solves systems of linear ordinary differential equations using Laplace transform. Here the *eqn*'s are differential equations in the dependent variables *x_1*, ..., *x_n*. The functional dependence of *x_1*, ..., *x_n* on an independent variable, for instance *x*, must be explicitly indicated in the variables and its derivatives.
bc2 *(solution, xval1, yval1, xval2, yval2)*	Solves a boundary value problem for a second order differential equation. Here: *solution* is a general solution to the equation, as found by ode2; *xval1* specifies the value of the independent variable in a first point, in the form *x = x1*, and *yval1* gives the value of the dependent variable at that point, in the form *y = y1*. The expressions *xval2* and *yval2* give the values for these variables at a second point, using the same form.
ic1 *(solution, xval, yval)*	Solves initial value problems for first order differential equations. Here, the *solution* is a general solution to the equation, as found by ode2, *xval* gives an initial value for the independent variable in the form *x = x0*, and *yval* gives the initial value for the dependent variable in the form *y = y0*.
ic2 *(solution, xval, yval, dval)*	Solves initial value problems for second-order differential equations. Here, the *solution* is a general solution to the equation, as found by ode2, *xval* gives the initial value for the independent variable in the form *x = x0*, *yval* gives the initial value of the dependent variable in the form *y = y0*, and *dval* gives the initial value for the first derivative of the dependent variable with respect to the independent variable, in the form diff(*y*,*x*) = *dy0* (diff does not have to be quoted).
ode2 *(eqn, dvar, ivar)*	The function ode2 solves an ordinary differential equation (ODE) of first or second order. It takes three arguments: an ODE given by *eqn*, the dependent variable *dvar*, and the independent variable *ivar*. When successful, it returns either an explicit or implicit solution for the dependent variable. %c is used to represent the integration constant in the case of first order equations, and %k1 and %k2 the constants for second-order equations. The dependence of the dependent variable on the independent variable does not have to be written explicitly, as in the case of desolve, but the independent variable must always be given as the third argument.
drawdf *(dydx,...options and objects...)* **drawdf** *(dvdu, [u,v], ...options and objects...)* **drawdf** *(dvdu, [u,umin,umax], [v,vmin,vmax] ,...options and objects...)* **drawdf** *([dxdt,dydt],...options and objects...)* **drawdf** *([dudt,dvdt], [u,v], ...options and objects...)* **drawdf** *([dudt,dvdt], [u,umin,umax], [v,vmin,vmax],...options and objects...)*	Function drawdf draws a 2D direction field with optional solution curves and other graphics using the draw package. The first argument specifies the derivative(s), and must be either an expression or a list of two expressions. *dydx*, *dxdt* and *dydt* are expressions that depend on *x* and *y*. *dvdu*, *dudt* and *dvdt* are expressions that depend on *u* and *v*. To make use of this function, write first load(drawdf).

For the expanded applications use the package „contrib_ode“. This is demonstrated in example 10.

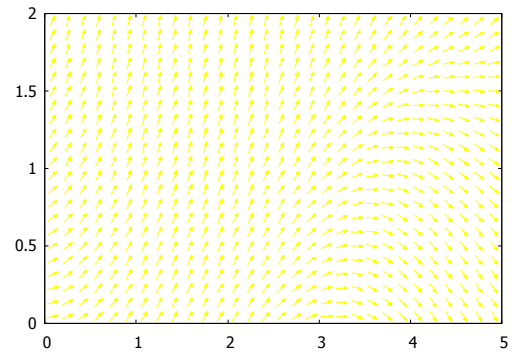
Example 4.1.8.

```
(%i23) load(drawdf)$
       eq1:sin(x)+tan(y/2)$
       drawdf(eq1,[x,y],[x,0,5],[y,0,2]);
(%o25) 0
```



Example 4.1.9.

```
(%i26) load(drawdf)$
       eq1:sin(x)+tan(y/2)$
       drawdf(eq1,[x,y],[x,0,5],[y,0,2],
       field_color=yellow);
(%o28) 0
```



Example 4.1.10.

```
(%i29) load('contrib_ode)$
       eq:'diff(f,t)=(f-t)^3;
       contrib_ode(eq,f,t);
```

$$(\%o30)\quad \frac{d}{dt}f=(f-t)^3$$

$$(\%o31)\quad [[t=(-\sqrt{3}\,\%i-1)\left(-\frac{\log(\%t^{2/3}+\%t^{1/3}+1)}{12}-\frac{\operatorname{atan}\left(\frac{2\,\%t^{1/3}+1}{\sqrt{3}}\right)}{2\sqrt{3}}+\frac{\log(\%t^{1/3}-1)}{6}\right)+\%c,f=t+\left(-\frac{\sqrt{3}\,\%i}{2}-\frac{1}{2}\right)\%t^{1/3}],[t=(\sqrt{3}\,\%i-1)\left(-\frac{\log(\%t^{2/3}+\%t^{1/3}+1)}{12}-\frac{\operatorname{atan}\left(\frac{2\,\%t^{1/3}+1}{\sqrt{3}}\right)}{2\sqrt{3}}+\frac{\log(\%t^{1/3}-1)}{6}\right)+\%c,f=t+\left(\frac{\sqrt{3}\,\%i}{2}-\frac{1}{2}\right)\%t^{1/3}]]$$

6. CONCLUSION

The research paper can apply each and every part of Equations, Polynomials and Differential Equations, help application of the physical sciences and engineering, make faster progress, and help to understand Equations, Polynomials and Differential Equations faster. The paper particularity helps to understand parts of Calculus and is going to extend to other parts of the Calculus.

7. ACKNOWLEDGEMENTS

I would like to thank the Maxima developers, Nigerian Turkish Nile University, and Prof. Niyazi ARI for their friendly help.

REFERENCES

[1] Niyazi ARI, Lecture notes, University of Technology, Zurich, Switzerland.
[2] Niyazi ARI, Symbolic computation of electromagnetics with Maxima (2013).
[3] http://maxima.sourceforge.net/
[4] R. H. Rand, Introduction to Maxima,
[5] R. Dodier, Minimal Maxima, 2005
[6] https://www.ma.utexas.edu/maxima/maxima_19.html.
N. Ari, G. Apaydin, Symbolic Computation Techniques for Electromagnetics
[7] MAXIMA and MAPLE, Lambert Academic Publishing, 2009.