Instant Maxima*

Stephen Ellner, Ecology and Evolutionary Biology, Cornell

Last compile: August 24, 2015

1 Introduction

These notes were written using the wxMaxima front-end, versions 0.7.6 and 0.85, and Maxima 5.16.3 and higher, both obtained from `maxima.sourceforge.net`. It is assumed that you have these installed on a Windows computer; some things may be different in OS X and Linux versions.

Each input or output line in Maxima has a label such `%i1` and can be referred to by label for the rest of the session. i labels denote your commands, and o labels denote output from Maxima. Never use variable names like %i1 or %o5, as these will be confused with the lines so labeled. Depending on your version of Maxima, it may automatically start with a line labeled `%i1` displayed, or you can create one by entering a command like `a:2;`, followed by pressing the "Enter" key. **NOTE:** in some versions of wxMaxima you need to first "turn on" the Enter key as follows: in the menus at the top of the wxMaxima window, go to Edit/Configure and in the Options tab check the box "Enter evaluates cells".

Maxima distinguishes lower and upper case. All built-in functions have names which are completely lower case (sin, cos, save, load, etc). Built-in constants have lowercase names (%e, %pi, inf, etc). If you type SIN(x) or Sin(x), Maxima assumes you mean something other than the built-in sin function. User-defined functions and variables can have names which are lower or upper case or both. foo(XY), Foo(Xy), FOO(xy) are all different.

2 Special Keys and Symbols

1. To tell Maxima that you have finished a command, use the semicolon (;) followed by a return (pressing the "Enter" key). A return by itself isn't enough.

2. To assign a value to a variable, Maxima uses the colon (:), not the equal sign. For example,

   ```
   (%i5) f: (x+4)^2;
   (%o5) (x+4)^2
   ```

   The equal sign is used for representing equations (see below) and defining functions.

3. An alternative end for a command is the dollar sign ($), which supresses displaying the output of the computation. This is useful for long results that you don't want displayed on the screen. For example:

   ```
   (%i6) f: expand((x+4)^25) $
   (%i7)
   ```

*Based on "Introduction to Maxima" by Richard H. Rand, which was adapted from *Perturbation Methods, Bifurcation Theory and Computer Algebra* by Rand and Armbruster (Springer, 1987).

4. To end a Maxima session, control-Q or use the File menu. To stop a computation without leaving Maxima, type control-C.

5. If you want to refer to the immediately preceding result computed my Maxima, you can either use its o label, or you can use the percent symbol(%).

6. The constants e (natural log base), i (square root of -1) and π (3.14159...) are respectively referred to as %e, %i, and %pi. Note that the use of % here as a prefix is completely unrelated to the use of % to refer to the preceding result computed.

Exercise 2.1 Get Maxima to confirm for you that $e^{i\pi} = -1$. As your next command, type in
% ^2 ;
The result should be "1". Why?

3 Getting help

There are (at least) four ways of asking Maxima for help in the wxMaxima environment.

1. `? topic` will bring up a window with the entry for `topic` in the manual. Note that there has to be a space between the question mark and the topic.

2. `describe(topic)` gets you the manual entry (or something like it) written out as text in the Maxima console window.

3. `example(topic)` gets you some examples of using `topic`, written out as text in the Maxima console window.

4. `apropos("string")` finds functions or commands whose name includes `string`, for example `apropos("exp")`.

Exercise 3.1 Use all four of these methods on `taylor`, the function that computes taylor series. Compare the reuslts of `apropos("taylor")` and `apropos("tay")`.

4 Arithmetic

Symbols for the common arithmetic operations are

+, -	(addition and subtraction)
*	(scalar multiplication)
/	(division)
^ or **	(exponentiation)
.	(matrix multiplication)
sqrt(x)	(square root of x)

Maximas output is characterized by exact (rational) arithmetic. For example

```
(%i7) 1/100+1/101;
(%o7) 201/10100
```

If irrational numbers are involved in a computation, they are kept in symbolic form:

```
(%i2) (1 + sqrt(2))^5;
(%o2) (sqrt(2) + 1)^5
(%i3) expand (%);
(%o3) 29 sqrt(2) + 41
```

To express a result in decimal notation, follow the expression you want expanded by ,numer:

```
(%i4) %, numer;
(%o4) 82.01219330881976
```

5 Programming in Maxima

For extended computations it is useful to write and execute script files. Script files are written outside of Maxima with a text editor, and can then be loaded into Maxima with the `batch` command. In wxMaxima, control-B brings up a menu that can be used to search for Maxima script files (text files names `*.mac`; but be aware that OS X may have its own ideas about what a `.mac` file is and what to open it with).

Exercise 5.1 : To try out Maxima scripting, type the commands

```
f: x*(1-x-y) $
g: b*(x-a)*y $
Jfg: jacobian([f,g],[x,y]) $
J00: subst([x=0,y=0],Jfg);
J10: subst([x=1,y=0],Jfg);
J11: subst([x=a,y=1-a],Jfg);
```

into a text file named `VVJac.mac`. Note the use of the `subst` function to create the Jacobians at the three equilibria. Below we will see a "shortcut" way to do substitutions, but using `subst` helps to make clear exactly what you're doing. Also note that this exercise gives you a role model for computing Jacobian matrices, which you should refer back to when you work on some of the exercises below.

Before running the script, enter 'by hand' in the Maxima console the commands

```
remvalue(all);
load(linearalgebra);
```

The first line removes the values from all previously defined variables, so that things you've done before can't mess up what you're trying to do now. The second loads the package of linear algebra routines. For unclear reasons, some versions of Maxima are *much, much* happier if these statements are done first at the command line, rather than putting them into a script.

Now (finally) use Maxima to run the script, either by (a) copy-paste into the Maxima console window, or by using control-B to load the script file.

Maxima scripts can have comments. The syntax is that anything between /∗ and ∗/ is a comment. For example, here is a commented version of the script above:

```
/* --- Define the vector field --- */
f: x*(1-x-y) $
g: b*(x-a)*y $

/* --- Compute the jacobians ---  */
Jfg: jacobian([f,g],[x,y]) $ /* general Jacobian */
```

```
J00: subst([x=0,y=0],Jfg);   /* Jacobian at (0,0) */
J10: subst([x=1,y=0],Jfg);   /* Jacobian at (1,0) */
J11: subst([x=a,y=1-a],Jfg); /* Jacobian at (a,1-a) */
```

6 Algebra

Maxima's value becomes apparent when we see how it can do algebra. **Try the following:**

```
f: (x + 3*y + x^2*y)^3;
expand(f);
q: expand(f), x=5/z;
ratsimp(q);
```

The first line creates f as a symbolic expression involving x and y. The second line expands the polynomial. The third does a *substitution*: inside f, we replace x by 5/z, and we call the result q. This syntax is an alternative to using the `subst` function. The last line puts q over a common denominator. The wxMaxima window has buttons or menus (depending on the version) for various functions that can expand expressions and simplify them in various ways.

Maxima can also try to solve systems of nonlinear equations. In this example, we solve three equations in the three unknowns a, b, c:

```
eq1: A + B*C = 1;
eq2: B - A*C = 0;
eq3: A + B = 5;
out: solve([eq1,eq2,eq3], [A, B, C]);
```

Note that the three equations are created as symbolic expressions, with ":" as the assignment operator and "=" as an assertion within the equations. The last line invokes the `solve` function, which in this case finds the solutions:

$$\left[A = \frac{25\sqrt{79}\,i + 25}{6\sqrt{79}\,i - 34}, B = \frac{5\sqrt{79}\,i + 5}{\sqrt{79}\,i + 11}, C = \frac{\sqrt{79}\,i + 1}{10}\right],$$

$$\left[A = \frac{25\sqrt{79}\,i - 25}{6\sqrt{79}\,i + 34}, B = \frac{5\sqrt{79}\,i - 5}{\sqrt{79}\,i - 11}, C = -\frac{\sqrt{79}\,i - 1}{10}\right]$$

(Aside to TeX users: the `tex` command makes it easy to put complicated Maxima output into a document. Try `tex(out)` to see, after `out` has been created.)

The object `out` created above is a list of lists. To see how to access the solutions for future use, try the following

Exercise 6.1 Enter the following commands one at a time, and figure out what they do for you (note: `rhs` stands for "right hand side"):

```
out[1];
out[1][1];
A: rhs(out[1][1]);
```

Then, figure out how to extract the corresponding value of B and verify that $A + B = 5$. Remember that Maxima doesn't simplify expressions unless you ask it to! Other useful functions for extracting parts of a complicated expression are `num`, `denom`, and `part`.

Exercise 6.2 Try the following:

```
out[1][1];
out[1][2];
z: out[1][1]+out[1][2]
```

What kind of an object is z? Why is it *not true* that z=5?

Exercise 6.3 : Consider the differential equation model

$$\begin{aligned} \frac{dN}{dt} &= rN - \frac{cNP}{a+P} \\ \frac{dP}{dt} &= \frac{bNP}{a+P} - mP \end{aligned} \tag{1}$$

Write a script to have Maxima to solve for the point $N^* > 0, P^* > 0$ at which the nullclines for this equation intersect, and define variables `nstar` and `pstar` that are the relevant solutions as functions of the parameters.

Exercise 6.4 : The model above can be rescaled to

$$\begin{aligned} \dot{x} &= rx - \frac{bxy}{1+y} \\ \dot{y} &= \frac{xy}{1+y} - y \end{aligned} \tag{2}$$

Do for this model what you did for the last one, defining variables `xstar` and `ystar` that are the relevant solutions as functions of the parameters. Contrasting this with your solutions to the last exercise should put to rest any questions about why we rescale models. Save your script as a `*.mac` text file for future use.

Exercise 6.5 : By looking at the expressions for `xstar` and `ystar`, state the conditions on parameters for the model to have an interior equilibrium where the predator and prey coexist.

7 Calculus

Maxima can compute derivatives and integrals, expand in Taylor series, take limits, and obtain exact solutions to ordinary differential equations. We begin by defining the symbol f to be the following function of x:

```
(%i1) f: x^3 * %e^(k*x) * sin(w*x);
(%o1) x^3*%e^(k*x)*sin(w*x)
```

Maxima can compute the derivative and indefinite integral (antiderivative) of f:

```
diff(f,x);
integrate(f,x);
```

A slight change in syntax gives definite integrals:

```
(%i4) integrate (1/x^2, x, 1, inf);
```

Take a look at the Algebra, Calculus, and Simplify menus to get an idea of what's available.

As we've seen above, Maxima's `jacobian` function can calculate the Jacobian matrix for the set of functions defining a system of differential equations or difference equations. Maxima can also solve some differential equations. The first step is to define the differential equation as a Maxima expression. Consider the differential equation

$$\frac{d^2y}{dx^2} + \frac{dy}{dx} + y = 0.$$

We define this equation in Maxima using the *unevaluated* form of the `diff` function, which is done using the quote (') operator:

```
(%i1) eq1: 'diff (y, x, 2) + 'diff (y, x) + y;
```

Maxima's ode2 function can solve first and second order ODEs:

```
(%i12) ode2(eq1, y, x);
```

Try the last two commands and see what happens.

Maxima can also solve linear systems of differential equations using Laplace transforms (the function name is desolve). Differential equations are not one of Maxima's strong points; more recently developed computer algebra systems (Maple, Mathematica) can deal with a wider range of differential equations, and can help you identify if an equation at hand falls into one of the solvable categories.

Exercise 7.1: Continuing your study of the rescaled predator-prey model, write a script to have Maxima compute the Jacobian matrix for this model at the equilibria $(x, y) = (0, 0)$ and $(x, y) = (x^*, y^*)$.

8 Matrix calculations

Maxima can compute the determinant, trace, inverse and eigenvalues and eigenvectors of matrices which have symbolic elements (i.e., elements which involve algebraic variables.)

First the matrix must be created. In some situations Maxima creates it for us, for example as a Jacobian matrix:

```
f: x*(1-x-y) $
g: b*(x-a)*y $
Jfg: jacobian([f,g],[x,y]) $
J: Jfg,x=a,y=1-a;
```

which should give you

$$\begin{pmatrix} -a & -a \\ (1-a)\ b & 0 \end{pmatrix}$$

Now try the following: `determinant(J); invert(J); invert(J),detout; eigenvalues(J);`

The function to compute the trace of a matrix is `mattrace`, which for some reaason is not in the linear algebra package. You need to `load("nchrpl")` and then you can use `mattrace(J);` to get the trace.

The eigenvalues function gives you a list as its output:

$$\left[\left[-\frac{\sqrt{(4\,a^2-4\,a)\ b+a^2}+a}{2}, \frac{\sqrt{(4\,a^2-4\,a)\ b+a^2}-a}{2}\right], [1,1]\right]$$

The first [,] has the two eigenvalues; the second has their multiplicities (as roots of the characteristic polynomial). Here we see that the eigenvalues are

$$-\frac{\sqrt{(4\,a^2-4\,a)\ b+a^2}+a}{2} \quad \text{and} \quad \frac{\sqrt{(4\,a^2-4\,a)\ b+a^2}-a}{2}$$

each with multiplicity 1, which we recognize as $\frac{-a\pm\sqrt{(4\,a^2-4\,a)\,b+a^2}}{2}$ even though Maxima did not (as always, Maxima has tremendous power but absolutely no insight).

To create a matrix "by hand", you can use the menus in wxMaxima: Algebra/Enter Matrix brings up an interactive display that can be used to enter a matrix. *Try this now* to create the matrix

$$\begin{pmatrix} d & 1 & 2 \\ 0 & 3 & 0 \\ 0 & 0 & 5 \end{pmatrix}$$

and compute its inverse.

Exercise 8.1 : Have Maxima compute the determinant and trace of the Jacobian matrix at (x^*, y^*) from the last exercise. Using these results, analyze the stability of the equilibrium.

9 Learning more

1. A lot of documentation, including the full Maxima manual, are available online at `maxima.sourceforge.net/documentation.html`.
2. The Wikipedia page on Maxima, `en.wikipedia.org/wiki/Maxima_(software)`, has several useful links at the bottom.
3. Richard Rand's "Introduction to Maxima", from which a lot of this document was lifted, is available at `maxima.sourceforge.net/docs/intromax/intromax.html`.

10 A selective list of Maxima functions

`allroots(a)` Finds all the (generally complex) roots of the polynomial equation `A`, and lists them in numerical format (i.e. to 16 significant figures).

`append(a,b)` Appends the list `b` to the list `a`, resulting in a single list.

`batch(a)` Loads and runs a program with filename `a`.

`coeff(a,b,c)` Gives the coefficient of `b` raised to the power `c` in expression `a`.

`desolve(a,b)` Attempts to solve a linear system `a` of ODE's for unknowns `b` using Laplace transforms.

`determinant(a)` Returns the determinant of the square matrix `a`.

`diff(a,b1,c1,b2,c2,...,bn,cn)` Gives the mixed partial derivative of `a` with respect to each `bi`, `ci` times. For brevity, `diff(a,b,1)` may be represented by `diff(a,b)`. `'diff(...)` represents the unevaluated derivative, useful in specifying a differential equation.

`eigenvalues(a)` Returns two lists, the first being the eigenvalues of the square matrix `a`, and the second being their respective multiplicities.

`eigenvectors(a)`Does everything that `eigenvalues` does, and adds a list of the eigenvectors of `a`.

`entermatrix(a,b)` Cues the user to enter an `a` × `b` matrix, element by element.

`ev(a,b1,b2,...,bn)` Evaluates `a` subject to the conditions `bi`. In particular the `bi` may be equations, lists of equations (such as that returned by `solve`), or assignments, in which cases `ev` "plugs" the `bi` into `a`. The `bi` may also be words such as `numer` (in which case the result is returned in numerical format), `detout` (in which case any matrix inverses in `a` are performed with the determinant factored out), or `diff` (in which case all differentiations in `a` are evaluated, i.e., `'diff` in `a` is replaced by `diff`). For brevity in a manual command (i.e., not inside a user-defined function), the `ev` may be dropped, shortening the syntax to `a,b1,b2,...,bn`.

`expand(a)` Algebraically expands `a`. In particular multiplication is distributed over addition.

`factor(a)` Factors `a`.

`grind(a)` Displays a variable or function `a` in a compact format.

`ident(a)` Returns an `a` × `a` identity matrix.

`imagpart(a)` Returns the imaginary part of `a`.

`integrate(a,b)` Attempts to find the indefinite integral of `a` with respect to `b`.

`integrate(a,b,c,d)` Attempts to find the integral of `a` with respect to `b`, taken from `b=c` to `b=d`. The limits of integration `c` and `d` may be taken is `inf` (positive infinity) of `minf` (negative infinity).

`invert(a)` Computes the inverse of the square matrix `a`.

`kill(a)` Removes the variable `a` with all its assignments and properties from the current Maxima environment.

`limit(a,b,c)` Gives the limit of expression `a` as variable `b` approaches the value `c`. The latter may be taken as `inf` of `minf` as in `integrate`.

`lhs(a)` Gives the left-hand side of the equation `a`.

`matrix(a1,a2,...,an)` Creates a matrix consisting of the rows `ai`, where each row `ai` is a list of `m` elements, `[b1, b2, ..., bm]`.

`num(a)` Gives the numerator of `a`.

`ode2(a,b,c)` Attempts to solve the first- or second-order ordinary differential equation `a` for `b` as a function of `c`.

`playback(a)` Displays the last `a` (an integer) labels and their associated expressions.

`ratsimp(a)` Simplifies `a` and returns a quotient of two polynomials.

`realpart(a)` Returns the real part of `a`.

`rhs(a)` Gives the right-hand side of the equation `a`.

`solve(a,b)` Attempts to solve the algebraic equation `a` for the unknown `b`. A list of solution equations is returned. For brevity, if `a` is an equation of the form `c = 0`, it may be abbreviated simply by the expression `c`.

`subst(a,b,c)` Substitutes `a` for `b` in `c`.

`taylor(a,b,c,d)` Expands `a` in a Taylor series in `b` about `b=c`, up to and including terms of order `d`. Maxima also supports Taylor expansions in more than one independent variable; see the Manual for details.

`transpose(a)` Gives the transpose of the matrix `a`.

`trigexpand(a)` Trig simplification function which uses the sum-of-angles formulas to simplify the arguments of individual `sin`'s or `cos`'s. For example, `trigexpand(sin(x+y))` gives `cos(x) sin(y) + sin(x) cos(y)`.

`trigreduce(a)` Trig simplification function which uses trig identities to convert products and powers of `sin` and `cos` into a sum of terms, each of which contains only a single `sin` or `cos`. For example, `trigreduce(sin(x)^2)` gives `(1 - cos(2x))/2`.

`trigsimp(a)` Trig simplification function which replaces `tan`, `sec`, etc., by their `sin` and `cos` equivalents.