1

A Maxima Guide for Calculus Students

Moses Glasner

e-mail: glasner_AT_math_DOT_psu_DOT_edu

Address:Department of Mathematics

PSU

University Park, PA 16802

Additional documentation is available at the Maxima URL:
`http://maxima.sourceforge.net/docs.shtml` Also see:
`http://www.math.psu.edu/glasner/Max_doc`

2

Scientific Calculator

By default Maxima comes with its own graphical user interface, Xmaxima. People familiar with Emacs may be much more comfortable using Maxima under the Emacs maxima-mode provides for a very powerful environment for combining computations together with the LaTeXcomposition of a document. For very brief usage not requiring extensive interaction, Maxima can be run from a console, such as Xterm under Linux, or DOS prompt under Windows.

Maxima can be run in two other more elaborate environments: TeXmacs and wxMaxima. The former gives a typeset appearance to Maxima's output and provides automatic generation of LaTeXsource whereas the latter provides a menu driven environment for using Maxima. While this is a very friendly way to start using Maxima, it puts a severely limits the usefullness of Maxima.

We assume that you have loaded Maxima onto your computer and can run it from your chosen GUI. If it is Xmaxima, then you should be aware that the font can be adjusted somewhat by selecting Fonts in the Options menu. Note that the Maxima Primer that appears on the bottom half of the window. Any text appearing there can be copied and pasted into the top half as an instruction for Maxima to execute. Also the entire Maxima manual is available in this buffer by clicking on help. Maxima also provides help from the command line in response to the commands describe(something) or example(something). Maxima may ask additional questions before responding to these help requests and the response should be ended with a semicolon ; and then pressing the ENTER key.

When Maxima is ready to process an instruction it displays a %i followed by an integer on the screen: for example, (%i1). The instruction given to Maxima is typed in at this point. It must end with a semicolon or a dollar sign followed by a press on the Enter key. Maxima automatically assigns the label (%o1) to its output. Each subsequent inputs and outputs are assigned labels (%i2),(%i3),... and (%o2), (%o3),... . Versions of Maxima before 5.9.1 used C D instead of i o . Even though you are probably using the current version of Maxima, you still need to be aware of this because a lot of the available documentation and archives of user groups refer to the old labeling system.

Maxima instructions may be spread over multiple lines and pressing the Enter key without a semicolon or a dollar sign is the way to accomplish this. If Maxima needs to break a line of output, for example when evaluating $10^{100} + 1$, then it will inform the user that the continuation follows on the next line by inserting the symbol
.

If an input line is ended with a semicolon and Enter, then Maxima prints the output to the screen but if a dollar sign is used, then the result of the calculation is stored internally but not to be displayed on the screen but it is assigned the appropriate label. In any case both the input and output can be accessed by means of the label assigned by Maxima.

Although previous versions of Maxima were not case sensitive, versions since 5.9.1 are. Therefore, Maxima

will evaluate *sin (%pi/6)* as 1/2 but will not evaluate *Sin (%pi/6)* and *sin (%pI/6)*.

```
(%i1) a: 1;


(%o1)                                  1
(%i2) b: 2$



(%i3) a + b;


(%o3)                                  3
```

Note that ending the input line with $ suppresses the output.

It is very important to remember that Maxima interprets the symbol = in an entirely different way. The following dialogue demonstrates how = is understood by Maxima.

```
(%i1) 2*x + 1 = 7;


(%o1)                            2 x + 1 = 7
(%i2) solve([%],[x]);


(%o2)                              [x = 3]
```

So you see when we do math we use the symbol = in two distinct meanings but Maxima uses two different symbols for these two meanings. Also note that the symbol

A very simple use of Maxima is an infinite precision scientific calculator. To illustrate this use consider the following:

```
(%i1) fpprec: 60;


(%o1)                                 60
(%i1) q : sqrt(2);


(%i2) 1 + q;
```

```
(%o3)                              sqrt(2) + 1
(%i2) bfloat(q);


(%i4) bfloat(1+q);


(%o5)   2.41421356237309504880168872420969807856967187537694807317668b0
```

The Maxima variable *fpprec* controls the number of digits Maxima prints to the screen and the command *bfloat(x)* asks it to print out a decimal approximation to the infinite precision value stored in x.

When Maxima completes a calculation all labels remain in use until either Maxima is instructed to free them or the user quits Maxima. If the user is not aware of this when starting a new calculation unexpected results can occur. The instruction *kill(all)* destroys all information, including loaded packages. Freeing labels with this instruction may be wasteful if some of the objects currently attached or some of the loaded packages will be needed in subsequent calculations. The instructions *values* and *functions* ask Maxima to display all labels currently attached to expressions or functions. The the following dialogue indicates a procedure for freeing specific labels while keeping the rest intact.

```
(%i10) kill(all)$


(%i1) a:1$


(%i2) b:2$


(%i3) f(x) := x^2$


(%i4) g(x):=x^3 $


(%i5) values;


(%o5)                                [a, b]
(%i6) functions;
```

```
(%o6)                              [f(x), g(x)]
(%i7) kill(a,f);

(%o7)                                  done
(%i8) values;

(%o8)                                   [b]
(%i9) functions;

(%o9)                                 [g(x)]
```

3

Finding Limits with Maxima

Maxima can be very helpful in checking limits of functions. But first one needs to know how to define functions for Maxima. It is done with a slight addition; specifically, the command is a colon followed immediately by an equal sign, as you see in the following dialogue:

```
(%i1) kill(all)$


(%i1) f(x) := sin(x)/x;


                                        sin(x)
(%o1)                           f(x) := ------
                                          x
(%i2) [f(1), f(2), f(1 + h), f(x + h)];


                            sin(2)  sin(h + 1)  sin(x + h)
(%o2)              [sin(1), ------, ----------, ----------]
                              2       h + 1       x + h
```

and finding limits is easy for Maxima:

```
(%i3) limit(f(x), x, 0);


(%o3)                                  1
(%i4) limit((sin(x+h)-sin(x))/h,h,0);


Is  sin(x)  positive, negative, or zero?
```

```
positive;
Is  cos(x)  positive, negative, or zero?

positive;
(%o4)                                cos(x)
```

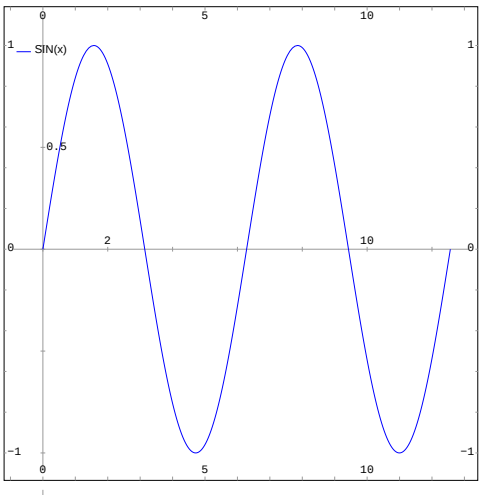
Note that in the course of the above dialogue Maxima required responses to two questions. After the reponse, positive, negative. or zero, one needs to type a semicolon and press the ENTER key.

Maxima can even deal with functions defined by several formulas:

```
(%i5) f(x) := if x > 0 then 1 else -1;


(%o5)                  f(x) := if x > 0 then 1 else - 1
(%i6) [f(-%pi/3), f(%pi/3)];


(%o6)                               [- 1, 1]
```

Maxima can also be used as a graphing calculator. Try the following instruction:

```
(%i7) plot2d(sin(x),[x,0,4*%pi]);


(%o7)
```



4

Limit of Difference Quotient

Finding the limit of Difference Quotients is a calculation where Maxima can really be helpful. For example, here is a Maxima session finding the limit of the difference quotient of $\sqrt{x-1}$.

```
(%i7) f(x) := sqrt(x - 1);


(%o7)                          f(x) := sqrt(x - 1)
(%i8) algebraic : true;


(%o8)                                  true
(%i9) diff_quot :(f(x+h) - f(x))/h;


                         sqrt(x + h - 1) - sqrt(x - 1)
(%o9)                    -----------------------------
                                       h
(%i10) simplified_diff_quot : 1/ratsimp(1/diff_quot);


                                       1
(%o10)                   -----------------------------
                         sqrt(x + h - 1) + sqrt(x - 1)
(%i11) limit(simplified_diff_quot,h,0);


                                    sqrt(x - 1)
(%o11)                              -----------
                                      2 x - 2
(%i12) diff(f(x),x);
```

```
                                    1
(%o12)                      -------------
                            2 sqrt(x - 1)
```

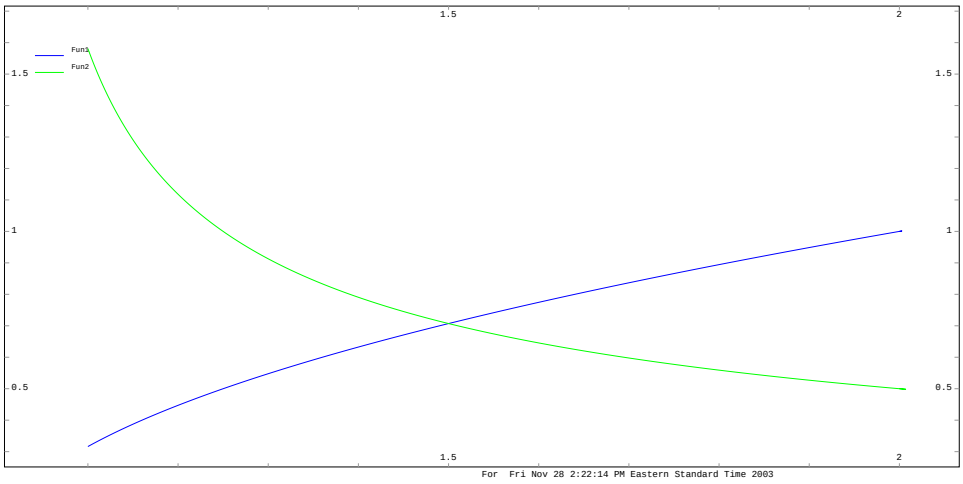
In the above dialogue you see several new commands. The last instruction labeled (%i12) tells Maxima to just compute the derivative of $f(x)$ as rapidly as it knows how. It is included only to confirm that the output labeled (%i11) is correct.

The command *ratsimp* tells maxima to simplify. However, Maxima by default is set to simplify rational functions and has been told to ignore algebraic (radical) simplifications in order to optimize the other types of simplifications it attempts. However, when we would like it to also perform algebraic simplifications we add the instruction *algebraic : true;*.

If you frequently work with algebraic expressions you may wish to set up an initialization file which maxima reads everytime it (re)starts. To do this you create a textfile called maxima-init.mac consisting of the line *algebraic : true;* and of course any other instructions you would like Maxima to execute at startup. (It is advisable to use a text editor, not a word processor for editing files maxima will read. GNU Emacs is an excellent possibility.) You may place this file in your Maxima work directory if you have one or any other directory you choose. However, Maxima must be told to start in this directory. If you are using Windows, then this can be accomplished setting the "Start in" parameter under the Properties of the Maxima icon. (You gain access to its Properties by right-clicking on it.) If Maxima is used under Linux, then the environment variable HOME should be set to the path of directory containing the file maxima-init.mac.

Maxima can plot two or more graphs simultaneously. The only modification required to the plot2d instruction is enclosing the funtions to be plotting in square brackets. Thus if you would like to see the two graphs in Figure 5 on page 138 plotted on the same axes, then give Maxima the following instruction:

```
(%i13) plot2d([f(x),diff(f(x),x)],[x,1.1,2]);

(%o13)
```


Fun3
Fun2
1.5
2
1.5
1
0.5
1.5
2
1.5
1
0.5
For Fri Nov 28 2:22:14 PM Eastern Standard Time 2003

5

Differentiation Formulas

You may be wondering whether you can get Maxima to show you differentiation rules without giving it specific functions to differentiate. The following dialogue indicates how this is done.

```
(%i1) kill(all)$


(%i1) depends(f,x);

(%o1)                                [f(x)]
(%i2) depends(g,x);

(%o2)                                [g(x)]
(%i3) derivative_of_quotient:diff(f/g,x);

                                   df     dg
                                   --   f --
                                   dx     dx
(%o3)                              -- - ----
                                   g       2
                                          g
(%i4) ratsimp(derivative_of_quotient);

                                     dg   df
                                   f -- - -- g
                                     dx   dx
(%o4)                            - -----------
                                         2
                                        g
```

You see Maxima has the quotient rule memorized in a somewhat different form than the one whichis optimal for humans. Also note that the command *depends(f,x)* tells maxima that f is a function of x. It can then deal with its derivative in a "symbolic" way. If we had not done this, then instruction *diff(f,x)* would have evoked the response 0 from Maxima because it would have thought that f and x are simply two independent variables.

A variety of instructions control the final form of the answer which Maxima returns. If we wish the answer to be over a common denominator then the instruction is $factor$.

```
(%i5) f(x):= x^(1/2);

                                      1/2
(%o5)                          f(x) := x
(%i6) g(x):= 1 + x^2;

                                        2
(%o6)                          g(x) := 1 + x
(%i7) answer:diff(f(x)/g(x),x);

                                             3/2
                          1             2 x
(%o7)                -------------- - ---------
                              2          2     2
                     2 sqrt(x) (x  + 1)   (x  + 1)
(%i8) factor(answer);

                                   2
                                3 x  - 1
(%o8)                     - --------------
                                 2     2
                            2 sqrt(x) (x  + 1)
```

If you need to access just parts of an answer provided by Maxima, then you can request that Maxima give you labels with which you can access the individual parts. The instruction for this is *pickapart*. For example, if you wish to find where the derivative in Example 10 is equal to 0, then you need a label with which to access the numerator. This is achieved with the following. Note that the additional variable 2 tells Maxima to what "depth" to the expression should broken into parts. Obviously, you want to use the smallest value that gets the desired label.

```
(%i9) pickapart(factor(answer),2);

                                   2
(%t9)                           3 x  - 1
```

```
                                          2     2
(%t10)                        2 sqrt(x) (x  + 1)

                                        %t9
(%o10)                                - ----
                                        %t10
```

You see that Maxima has attached labels (%t9) and (%t10) to the numerator and denominator of the factored form of the answer. You may wish to see what happens if the 2 is replaces by 1, 3, etc. Now in order to find the zeros of the numerator you do as follows:

```
(%i10) soln: solve(%t9,x);

                                    1             1
(%o10)                  [x = - -------, x = -------]
                             sqrt(3)       sqrt(3)
```

Note that the variable *soln* is a list with two elements. (Actually everything in Maxima is a list because it is written in the computer language LISP that is based on list processing.) You can access the elements which are in the form of equations $x =$something with the instruction **first** and then value of this solution can be retrieved with the instruction *rhs*

```
(%i11) the_first_soln: first(soln);

                                      1
(%o11)                          x = - -------
                                    sqrt(3)
(%i12) value_of_soln:rhs(the_first_soln);

                                     1
(%o12)                             - -------
                                     sqrt(3)
```

6

Derivatives of Trig Functions

Maxima can be quite helpful in differentiating trig functions. However, a couple of commands specific to trig functions are required in order to instruct Maxima to apply trig identities in simplifications. For example consider the following dialogue:

```
(%i7) kill(all);


(%o0)                                done
(%i1) diff( sin(x)/(1 + cos(x)),x);

                              2
                           sin (x)         cos(x)
(%o1)                    ------------- + ----------
                                        2  cos(x) + 1
                         (cos(x) + 1)
(%i2) factor(%);

                            2         2
                         sin (x) + cos (x) + cos(x)
(%o2)                    --------------------------
                                                2
                               (cos(x) + 1)
(%i3) trigsimp(%);

                                    1
(%o3)                             ----------
                                  cos(x) + 1
```

You see that the instruction *trigsimp* was required to get Maxima to make the obvious simplification using the Pythagorean identity. Also note that Maxima does not recognize the shortcut of leaving out parentheses when using trig functions, eg, $\sin x, \cos x, \tan x$; neither does it recognize the common abuse of notation that puts an exponent

directly onto the trig function; i.e., $\sin^2(x), \tan^{-1}(x)$ are not recognized; the correct inputs are $\sin(x)^2, \tan(x)^{-1}$. Also observe that $\tan(x)^{-1}$ means the reciprocal of tangent and not arctangent which is denoted by $\text{atan}(x)$. The other Maxima instruction is *trigreduce* which allows using the multiple angle formulas to reduce powers, eg:

```
(%i4) factor(cos(x)^2 + 2*sin(x)^2);

                                    2         2
(%o4)                          2 sin (x) + cos (x)
(%i5) trigsimp(cos(x)^2 + 2*sin(x)^2);

                                      2
(%o5)                               sin (x) + 1
(%i6) trigreduce(cos(x)^2 + 2*sin(x)^2);

                         cos(2 x) + 1      1   cos(2 x)
(%o6)                    ------------ + 2 (- - --------)
                              2            2      2
```

Of course, one could argue that the latter expression is not simpler than the one we started with. However, it is invaluable for integration, the inverse process of differentiation.

7

The Chain Rule

You get Maxima to show how it applies the chain rule by telling it which variables are dependent on which.

```
(%i10) f(x):= x^3;
                                          3
(%o10)                           f(x) := x
(%i11) depends(x,u)$

(%i12) diff(f(x),u);
                                     2 dx
(%o12)                             3 x  --
                                        du
```

In the above dialogue we used the functional notation to define f and used the instruction *depends* to inform Maxima that x is a function of u but did not provide a specific formula for it. However, we can also specify the dependence of x on u as follows:

```
(%i13) remove([x,u],dependency);

(%o13)                              done
(%i14) x: sin(u);

(%o14)                             sin(u)
(%i15) diff(f(x),u);
```

```
                                          2
(%o15)                          3 cos(u) sin (u)
```

Or, we can use functional notation for both functions and get Maxima to differentiate their composition.

```
(%i16) kill(x);


(%o16)                                done
(%i17) g(x):= sin(x);


(%o17)                           g(x) := sin(x)
(%i18) diff(f(g(u)),u);


                                          2
(%o18)                          3 cos(u) sin (u)
```

Note that the instruction *kill* was needed to remove the relationships set in each of the previous dialogues with Maxima. If a variable has multiple dependencies and only one of them is to be removed, then the instruction *remove([u,x],dependency)* can be used.

8

Implicit Differentiation; Higher Derivatives

Maxima can easily compute derivatives of implicit functions. Consider the following dialog that instructs Maxima to find dy/dx given the equation $x^2 + y^2 = 25$.

```
(%i1) eqn: x^2 + y^2 = 25;

                                  2    2
(%o1)                            y  + x  = 25
(%i2) depends(y,x);

(%o2)                               [y(x)]
(%i3) deriv_of_eqn:diff(eqn,x);

                                   dy
(%o3)                         2 y -- + 2 x = 0
                                   dx
(%i4) solve(deriv_of_eqn,'diff(y,x));

                                   dy     x
(%o4)                             [-- = - -]
                                   dx     y
```

Note the new symbol appearing in the *solve* instruction above. Normally the first argument of *solve* is an equation or list of equations and the second is a variable and a list of variables. Here we see dy/dx as the second argument. On the other hand there is also a single quote in front. A single quote in front of a symbol tells Maxima to attempt to evaluate the symbol but to treat it as an unknown quantity. For example,

```
(%i5) a : 3;
```

```
(%o5)                                3
(%i6) b : 4;


(%o6)                                4
(%i7) a + b;


(%o7)                                7
(%i8) 'a + b;


(%o8)                              a + 4
(%i9) 'a + 'b;


(%o9)                              b + a
(%i10) a + 'b;


(%o10)                             b + 3
```

So we see that the instruction *solve(deriv_of_eqn,'diff(y,x))* tells Maxima not try to evaluate the derivative of y with respect to x directly (which it really cannot do) but to regard *diff(y,x)* as an unknown quantity and solve for it from the differentiated equation, *deriv_of_eqn*.

The Maxima instruction to find higher order derivatives is the same as that for finding the first derivative except for a third argument indicating the order:

```
(%i11) diff(x^n,x,2);


                                       n - 2
(%o11)                      (n - 1) n x
```

9

Related Rates

Maxima can be helpful in solving related rates problems. For example consider the problme of finding the rate of change of the area of a circle given that the rate of rate of change of the radius is $dr/dt = 60$ when $t = 2$ and $r = 120$:

```
(%i1) area: a = %pi*r^2;
```

$$(\%o1) \qquad a = \%pi\ r^2$$

```
(%i2) depends([a,r],t)$

(%i3) deriv_of_area: diff(area,t);
```

$$(\%o3) \qquad \frac{da}{dt} = 2\ \%pi\ r\ \frac{dr}{dt}$$

```
(%i4) subst([diff(r,t)=60,r=120],deriv_of_area);
```

$$(\%o4) \qquad \frac{da}{dt} = 14400\ \%pi$$

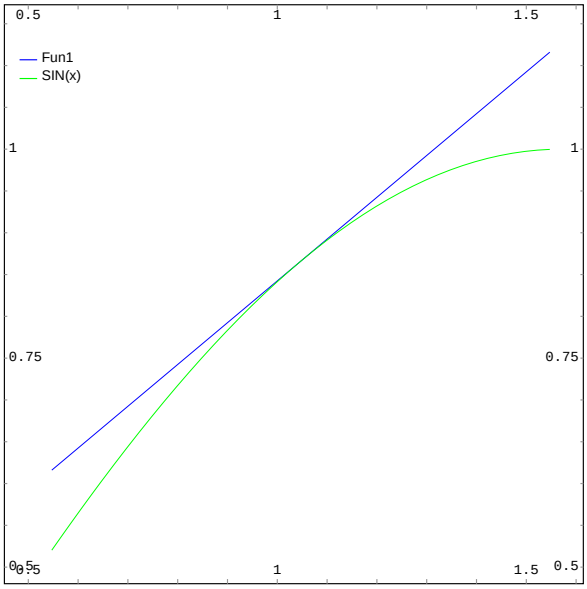
Note that the expression *%PI* is the Maxima's label for π. You might want to see how many digits of *pi* Maxima can find by giving the instruction *fpprec= 1000, bfloat(%PI)*; Also note that Maxima must be told which variables are time-dependent with the instruction *depends*. The instruction *subst* tells maxima to substitute a list of equations appearing as the first argument of *subst*, enclosed in square brackets, into the expression appearing as its second arguement.

10

Linear Approximations and Differentials

Maxima has a very powerful built-in tool for finding linear approximations to functions called the Taylor expansion of order 1.

```
(%i1) L: Taylor(sin(x),x,%pi/3,1);


                                          %pi
(%o1)                  Taylor(sin(x), x, ---, 1)
                                           3
(%i2) error1 :float(subst([x=1.1],L)) - float(subst([x=1.1],sin(x)));


(%o2) Taylor(0.89120736006144, 1.1, 1.047197551196598, 1.0) - 0.89120736006144
(%i3) error2 :float(subst([x=1.4],L)) - float(subst([x=1.4],sin(x)));


(%o3) Taylor(0.98544972998846, 1.4, 1.047197551196598, 1.0) - 0.98544972998846
(%i3) plot2d([L,sin(x)],[x,%pi/3-.5,%pi/3+.5]);


sh: gnuplot: command not found
```



In the dialogue above the instruction (C1) tells maxima to approximate $\sin(x)$ by a Taylor polynomial of degree 1, at the point $x = \pi/3$ This is simply another name for a linear approximation. On the other hand, Taylor polynomials come in all degrees, and you will have an opportunity to work with them in Math 141 or have a look at the Laboratory Project on page 212. The last argument in the Maxima instruction *Taylor* specifies the degree. By comparing *error1* and *error2* in the above dialogue you can verify that indeed the linear approximation is much more accurate close to the point of tangency than far away. You can also see this from the graph of the function and its linear approximation produced by Maxima.

You may be wondering about the extra characters $/T/$ in the output line (D1). This is only Maxima's reminder that the formula it has produced has been truncated and hence cannot be used when great accuracy is needed. If you are really irritated by the choice Maxima makes for its automatic labelling of input and output you can enter something like following instructions:

$$inchar : MY\_INPUT \qquad\qquad outchar : MAXIMA\_OUT$$

Or, you can put the following line into your maxima-init.mac file. tt(inchar : IN_ ,outchar : OUT_);

11

Maximum and Minimum Values

Maxima can be a great help in checking the process of locating critical points of functions as can see from the following dialogue:

```
(%i5) f(x):=x^(3/5)*(4-x);

                                      3/5
(%o5)                        f(x) := x    (4 - x)
(%i6) deriv_of_f: diff(f(x),x);

(%i7) pickapart(factor(deriv_of_f),2);

(%i7) solve([E3=0],x);

(%t8)                                5 x

                                       %t7
(%o8)                                - ---
                                       %t8
```

Remember that the instruction *pickapart* labels each of the expressions in order that you can refer to it in subsequent instructions. If you are not interested in seeing the intermediate steps, then the instruction *solve(diff(f(x),x);* would also give you the answer that $x = 3/2$ is one a critical number. However, you would still need to supply the information that the derivative fails to exist at $x = 0$.

12

Using Maxima's Lambda definition

Maxima can be helpful in seeing that indeed the c of the Mean Value Theorem depends on the given functions. For example consider the following dialogue which calculates the c for the functions x^2 and x^3 on the interval $[0, 1]$

```
(%i10) f : lambda([x])$


(%i11) MVT(f,a,b):=block([deriv, aver, c], aver:(f(b) - f(a))/(b-a),
                                    deriv : diff(f(x),x),
                                  c : solve(deriv=aver,x),
                                  print(c))$


(%i11) g : lambda([x],x^2)$

(%i11) MVT(g,0,1)$



(%i12) h : lambda([x],x^3)$


(%i12)
(%i13) MVT(h,0,1)$
```

So you see that for g Maxima found that c equals $1/2$ whereas for h Maxima found different solution with $1/\sqrt{3}$ being in the interval $[0, 1]$.

You also see several new useful instructions. First the declaration involving *lambda* tells maxima that g is a function of x given by the formula x^2. How is this different from the declaration $g(x) := x^2$ which we have used before? The new declaration requests that Maxima recognize the symbol g by itself in the definition and/or invocation of

another function, where as with the old declaration Maxima would not have recognized anything other than $g(x)$.

The other new instructions are *block* and *print* in the definition of a function. The first instructs Maxima to make the variables listed in square brackets local to the function being defined by the block and the *print* statement asks Maxima to do just that with all the current values of the variable or quoted strings that are passed as arguments to it.

13

Maxima helps sketch graphs

Of course Maxima's plotting ability helps verify graphs which are sketched using calculus techniques. But Maxima can also help implement these techniques. For example, Maxima can find on which intervals a derivative (and second derivative) is positive and on which it is negative. For example, suppose that we are to sketch the graph of the function $\frac{1}{3}x^3 - 4x = 4$. Then we find that its derivative is $x^2 - 4$, which is zero at $x = 2$ and $x = 1$. We can instruct Maxima to evaluate the derivative at $x = -3$, $x = 0$ and $x = 3$ to find where the derivative is positive and where it is negative. We may try this:

```
(%i16) expr : x^2- 4$


(%i17) subst(-3, x,expr);


(%o17)                                5
(%i18) subst(0, x,expr);


(%o18)                               - 4
(%i19) subst(3, x,expr);


(%o19)                                5
```

If however, you would want to substitute an algebraic expression for x, eg, $u = x - 2$ then Maxima would issue an error message. To inform Maxima that it is supposed to execute the algebra involved in the substitution, the command *ratsubst* needs to be used:

```
(%i20) ratsubst(u, x - 2,expr);


                                  2
(%o20)                           u  + 4 u
```

14

Limits at Infinity and One-sided Limits

Maxima is very helpful in checking limits at infinity. Obviously, it can check the extensive algebra that may need to be performed. But in addition it can evaluate the limits directly.

```
(%i13) f: (3*x^2 - x - 2)/(5*x^2 + 4*x +1 );
```

$$(\%o13)\qquad \frac{3\,x^2 - x - 2}{5\,x^2 + 4\,x + 1}$$

```
(%i14) limit(f,x,inf);
```

$$(\%o14)\qquad \frac{3}{5}$$

You see that Maxima recognizes the concept of infinity via the label *inf*. On the other hand Maxima does not compute one-sided limits directly. For this purpose the substitution $x = 1/t$ is very useful as you can see from the following dialogue computing $\lim_{x\to 0^-} \frac{|x|}{x}$.

```
(%i15) f : abs(x)/x$

(%i16) limit(ratsubst(1/t,x,f),t,-inf);
```

$$(\%o16)\qquad -1$$

And Maxima knows about infinite limits as well:

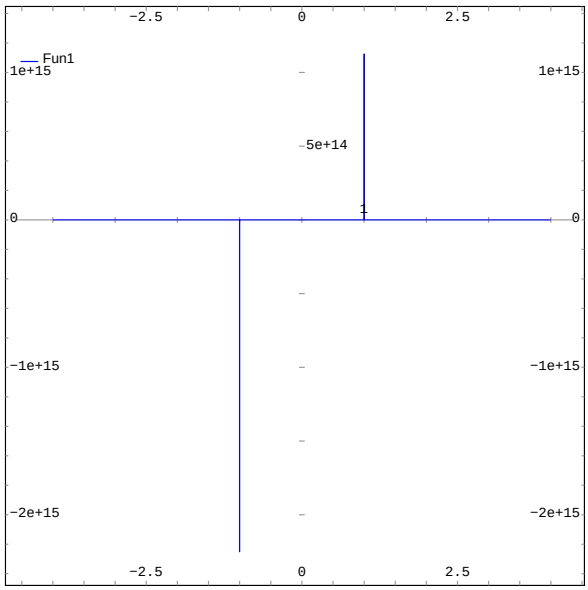
```
(%i17) f : 1/x$


(%i18) limit(ratsubst(1/t,x,f),t,-inf);

(%o18)                                - inf
```

15

Plotting Inspite of Vertical Asymptotes

Using Maxima to check your sketches of graphs of functions requires a little bit of care. For example, if you simply instruct Maxima to plot the function $f(x) = \dfrac{x^2}{x^2-1}$ by issuing the following instruction you get a nearly useless plot.

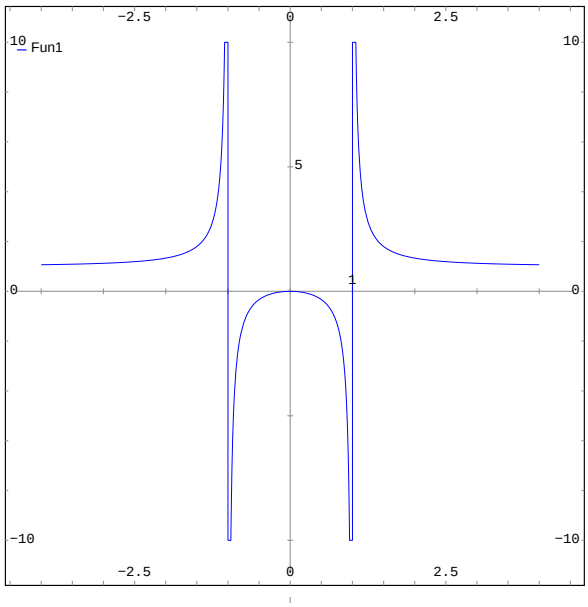
```
(%i27) plot2d(x^2/(x^2-1),[x,-4,4])$
```



It does show you the vertical asymptotes; so it is not a complete waste. The problem Maxima encounters is a lack of information about what you want to do with the function values that are too large or too negative to fit on any

computer screen. You can instruct Maxima to discard values greater than 10 or less than -10 as indicated in the following dialogue:

```
(%i28) plot2d(max(min(x^2/(x^2-1),10),-10),[x,-4,4])$
```



Of course the nearly vertical lines are to be interpreted as vertical asymptotes and not portions of the graph.

From this illustration you see that even the most advanced technology cannot replace a human being with good mathematical skills. Although Maxima can help check your execution of the eight steps in sketching the graph of a function, it cannot execute them without your input; it is especially important that you executed Step A and form some understanding of the range of the function and the interval on which you wish to view the graph. However, Maxima can be enormously helpful in checking the accuracy of your work in steps B through H.

16

Optimization; Computing Derivatives of Functions Subject to Constraint

Let us illustrate the use of Maxima to solve the following. A window is in the form of a rectangle surmounted by a semicircle. The rectangle is of clear glass while the semicircle is of tinted glass that transmits only half as much light per unit area as clear glass does. The total perimeter is fixed. Find the proportions of the window that will admit the most light. Neglect the thickness of the frame.

Let r be the radius of the semicircle. Then $2r$ is the base of the rectangle. Denote the height of the rectangle by h. Let P be the total perimeter of the window. Then P is a fixed constant. We have the following equations connecting r and h and expressing the amount of light L in terms of r and h, assuming that L is one for each unit of area in the rectangular part and L is $1/2$ for each unit of area in the semicircular part of the window:

$$P = 2r + 2h + \pi r \qquad\qquad L = 2rh + \frac{\pi r^2}{4}$$

```
(%i53) Eqn: P = 2*r +2*h + %pi*r$


(%i54) L : 2*r*h + %pi*r^2/4$


(%i54)
(%i55) Soln_h : solve(Eqn,h);


                                    3    P + (- %pi - 2) r
(%o55)               [lambda([x], x ) = -----------------]
                                                2
(%i56) h_is : rhs(first(Soln_h));


                                P + (- %pi - 2) r
(%o56)                          -----------------
                                        2
```

```
(%i56) L_fcn_r : ratsubst(h_is,h,L);

                                          2
                              (3 %pi + 8) r  - 4 r P
(%o57)                      - ----------------------
                                        4
(%i58) deriv_L :diff(L_fcn_r,r);

(%i59) Soln_r:solve(deriv_L = 0,r);

                                       2 P
(%o59)                          [r = ---------]
                                     3 %pi + 8
```

Since this is the only critical point found for L on the interval $[0, P(\pi+2)^{-1}]$ this must give the absolute maximum for L. Maxima can now be instructed to find the ratio of r to h:

```
(%i60) r_is : rhs(first(Soln_r));

                                    2 P
(%o60)                             ---------
                                   3 %pi + 8
(%i61) h_is_now : ratsubst(r_is,r,h_is);

                                  (%pi + 4) P
(%o61)                            -----------
                                  6 %pi + 16
(%i62) Ratio_finally :ratsimp(r_is/h_is_now);

                                     4
(%o62)                              -------
                                    %pi + 4
```

A somewhat different approach to solving problems in this section is to differentiate the constraint equation. This is significantly simpler than solving the constraint equation for one of the variables in case where the constraint equation is complicated or perhaps not even solveable for either of the variables. So in the above problem one would differentiate the equation for the perimeter with respect to r. This would yield a linear equation in the unknown derivative $\frac{dh}{dr}$ which can easily be solved. Then one would differentiate the formula for L with respect to r and this would also involve the unknown derivative $\frac{dh}{dr}$ which now can be eliminated. And then one could determine

the relationship between r and h that yields a critical point. The following dialogue indicates how Maxima can be instructed to carry out this computation.

```
(%i1) depends(h,r);


(%o1)                                [h(r)]
(%i2) Eqn: P = 2*r +2*h + %pi*r$



(%i3) L : 2*r*h + %pi*r^2/4$



(%i4) Deriv_eqn : diff(Eqn,r);


                                      dh
(%o4)                          0 = 2 -- + %pi + 2
                                      dr
(%i5) Deriv_h : rhs(first(solve(Deriv_eqn,'diff(h,r))));


                                    %pi + 2
(%o5)                             - -------
                                        2
(%i6) Deriv_L : diff(L,r);


                                dh     %pi r
(%o6)                         2 -- r + ----- + 2 h
                                dr       2
(%i7) Deriv_L_is : ratsubst(Deriv_h,'diff(h,r),Deriv_L);


                                (%pi + 4) r - 4 h
(%o7)                         - ----------------
                                        2
(%i8) solve(Deriv_L_is=0,r);


                                      4 h
(%o8)                            [r = -------]
                                     %pi + 4
```

Note that apostrophe in *'diff(h,r)* must be used of instructed to treat the derivative as an unknown quantity.

17

Newton's Method; Loading Library Programs; Recursive Functions; Initial Conditions

---

Newton's method can be implemented very easily in Maxima. Actually, there is a Maxima library function which has to be loaded and then all one has to do is give an initial guess. For example the following finds the root of $x^2 - 2$ closest to 1.5:

```
(%i8) load(newton)$


(%i9) f: x^2 - 2;


                                     2
(%o9)                               x  - 2
(%i10) sqr_rt_two:newton(f,3/2)$


(%i10) sqr_rt_two;


(%o11)                         1.414213562373469b0
```

The command *newton* is not built into Maxima and must be loaded from the library. Of course, if you wish you can automatically load it each time it is started by placing *load(newton);* in your maxima-init.mac file. Observe that although Maxima is not case sensitive when it comes to its own commands, it is case sensitive with user labeled and *newton* is regraded as one of the latter. Note that *newton* converts Maxima's output to bigfloat which can be given arbitrary precision. Thus setting *fpprec :100;* would give you square root of 2 to 100 significant digits. Also, the warning about converting float to bigfloat is inocuous. However, it can be avoided by passing 1.5b0 instead of 1.5 to *NEWTON*.

It may be good practice to define your own Maxima function which implements Newton's method.

```
(%i12) kill(all)$
```

```
(%i12)
(%i12) my_newt(f,guess,prec):= block([f_x,der_x,x_new],
                              f_x : f(guess),
                              der_x : subst(guess,x, diff(f(x),x)),
                              x_new : guess - f_x/der_x,
                              if abs(x_new - guess) < prec then return(guess)
                                 else my_newt(f,x_new,prec));


(%i12) g: lambda([x],x^2 -2);


(%i12) my_newt(g,1.5,.0001);
```

This definition of *my_newt* is recursive; i.e. the definition of the function calls the function itself until the desire precision is attained. From the viewpoint of program clarity this is the method of choice. When calling *my_newt* lambda definition of function must be used here because the name of a function f is being passed to *my_newt*.

Maxima is very helpful in checking your work on checking antiderivatives that are subject to an initial condition. For example, let us find the antiderivative v of $a = 6t + 4$ subject to the initial condition $v(0) = 9$

```
(%i2) kill(all)$


                                        2
(%o2)                          lambda([x], x  - 2)
(%i3) a: 6*t+4;


(%o3)                           1.41421568627451
(%i4) v: integrate(a,t);


(%i4)
(%i1) c_val:solve(subst(0,t,v+c)=-6,c);


(%o1)                                6 t + 4
(%i2) v: v+rhs(first(c_val));


                                      2
(%o2)                               3 t  + 4 t
(%i3) s: integrate(v,t);
```

```
(%o3)                                [c = - 6]
(%i4) c_val:solve(subst(0,t,s+c)=9,c);


                                       2
(%o4)                           3 t  + 4 t - 6
(%i5) s+rhs(first(c_val));


                                    3      2
(%o5)                           t  + 2 t  - 6 t
```

You see that Maxima chronically omits the constant of integration when it performs the instruction *integrate*; so it is up to the user to provide it.

18

Limits of Riemann sums; The Definite Integral

The actual evaluation of the limit of the Riemann sums associated to a function on an interval $[a, b]$ is absolutely tedious. So we are very pleased that Maxima can help. Maxima's instruction for summing a series is *sum* and it takes four arguments: the expression, the index, the beginning value of the index, and the ending value of the index. Maxima's default is not to carry out the evaluation of the sum of a series. And the instruction to change this is: *simpsum : true$* The Maxima dialogue for doing is as follows:

```
(%i29) simpsum:true$


(%i30) R_sum: ratsimp((3/n)*sum((2+3*i/n)^4,i,1,n));

                            4         3         2
                       6186 n  + 9135 n  + 3510 n  - 81
(%o30)                 --------------------------------
                                     4
                                  10 n
(%i31) answr:limit(R_sum,n,\protect \mathop {\relax \kern \z@ \mathgroup \symoperators inf}\nmlimits@ )$


(%i32) answr;

                                    3093
(%o32)                              ----
                                     5
(%i33) the_same: answr - integrate(x^4,x,2,5)$


(%i34) the_same;


(%o34)                               0
```

19

Fundamental Theorem of Calculus

You may be curious whether Maxima knows the Fundamental Theorem. The following dialogue provides the answer.

```
(%i8) assume(x>0)$


(%i9) S: lambda( [x], integrate(cos(t),t,0,x));


(%o9)               lambda([x], integrate(cos(t), t, 0, x))
(%i10) answr:diff(S(x),x)$


(%i11) answr;


(%o11)                              cos(x)
(%i12) answr:diff(S(x^3),x)$


(%i13) answr;


                                     2      3
(%o13)                            3 x  cos(x )
```

Note that without the command *assume(x) ¿ 0* Maxima will ask the user to declare the sign of x. The response to Maxima's query should be ended with a dollar sign or semicolon. Also, note that the name of the "dummy" variable in the definition of S cannot be chosen to be x even though that is perfectly acceptable in mathematical communications.

20

Indefinite Integrals; Using ode2

The Maxima instruction *integrate* serves two purposes depending on the number of arguments that are passed to it. It is used for telling Maxima to find either the indefinite or definite integral. If two arguments are passed then Maxima finds the indefinite integral of the function in the first argument with respect to the variable in the second. If four arguments are passed, then maxima finds the definite integral over the interval starting at the third and ending at the fourth.

Another instruction that can be used to achieve the same purpose is ode2. It can be used to instruct Maxima to solve the equation $\frac{dy}{dx} = f(x)$, for y. This is a very powerful instruction but in this case using a sledge hammer to drive a tack does no harm. The following indicates how this is done.

```
(%i3) eq: 'diff(y,x) = sqrt(1/x^2-1/x^3);

                                dy        1    1
(%o3)                           -- = sqrt(-- - --)
                                dx         2    3
                                          x    x
(%i4) ode2(eq,y,x);

                                                          2
                              2                    2 sqrt(x  - x)
(%o4)       y = log(2 sqrt(x  - x) + 2 x - 1) - -------------- + %c
                                                       x
```

Whenever, Maxima is handed an expression it automatically tries to evaluate it. Therefore the quote before the *diff* tells Maxima not to waste time trying to evaluate $\frac{dy}{dx}$. Maxima is being told what this derivative is and the next instruction will ask it to solve for y. Note that when you use *ode2* Maxima includes the arbitrary constant *%c* in its respnose.

21

Areas between Curves, Finding Intersections

Maxima can be very helpful in finding the points of intersection of curves. The following dialogue demonstrates how:

```
(%i1) eq1 :  y = x - 1$


(%i2) eq2 : y^2  = 2*x + 6$


(%i2) solve([eq1,eq2],[x,y]);

(%o3)                [[x = - 1, y = - 2], [x = 5, y = 4]]
```

Note that Maxima does not integrate absolute value. Thus to get Maxima to do a problem where neither graph remains above the other on the entire interval, you first need to find the points of intersection and then instruct Maxima to sum the integrals of the upper function minus the lower function over the intervals between.

22

Volumes by slicing; Using Trigsimp

Maxima cannot help with finding the formula for the area of the cross-section of a typical slice $A(x)$ but it is certainly capable of checking the ensuing integration.

For example, finding the volume generated by revolving the area below $y = \sec(x)$, above $y = 1$ and between $x = 1$ and $x = -1$ requires evaluating $V = \int_{-1}^{1} \sec^2(x) - 1^2\, dx$

```
(%i4) A : (sec(x))^2 - 1$


(%i5) B : trigsimp(A);

                                      2
                                   sin (x)
(%o5)                              -------
                                      2
                                   cos (x)
(%i5) integrate(B,x,-1,1);


(%o6)                          2 tan(1) - 2
```

Of course, Maxima could have found the integral without issuing the instruction *trigsimp*. This instruction causes Maxima to use the Pythagorean identities to simplify an expression. An instruction like *ratsimp* would have no effect here.

23

Volumes by Cylindrical Shells; Using Trigreduce and Trigexpand

---

Let find the volume of the solid generated by revolving the region between $y = \cos x$, $y = 0$ and the lines $x = 0$, $x = \pi/2$ about the x-axis. Using the shell method, the integral to be calculated is

$$2\pi \int_0^1 y \cos^{-1} y \, dy$$

which is another difficult integral requiring integration by-parts. However, the disk method yields an integral which can easily be found:

$$\pi \int_0^{\pi/2} \cos^2 x \, dx = \frac{\pi}{2} \int_0^{\pi/2} 1 + \cos 2x \, dx = \frac{\pi}{2} \left[x + \frac{1}{2} \sin 2x \right]_0^{\pi/2} = \frac{\pi^2}{4}$$

If you forget some of the double/half angle formulas, then you ask Maxima to remind you;

```
(%i1) halfangles: true$


(%i2) A : cos(x)^2$


(%i3) B: trigreduce(A);

                                cos(2 x) + 1
(%o3)                           ------------
                                     2
(%i4) C: trigexpand(B);

                                    2         2
                               - sin (x) + cos (x) + 1
(%o4)                          -----------------------
                                          2
```

From this dialogue you see that *trigreduce* tells Maxima to use the half-angle formulas to simplify an expression and *trigexpand* tells Maxima to use the double-angle formulas. Note that the commonly accepted mathematical abuse of notation of using $\cos^2(x)$ to stand for $cos(x)^2$ is not recognized by Maxima.