A wxMaxima Guide for Calculus Students

This guide follows *A Maxima Guide for Calculus Students* very closely. Details are at the end of this document. I thank Professsor Glasner for allowing me to use the material that he had developed. I am responsible for errors or shortcomings where the material below deviates from Professor Glasner's guide. Please contact me (wmixon at berry dot edu) with corrections or suggestions for improvement.

1 Scientific Calculator

By default *Maxima* comes with its own graphical user interface, *Xmaxima*. People familiar with *Emacs* may be more comfortable using *Maxima* under the *Emacs* `maxima`-mode, which provides for a powerful environment for combining computations with the LaTeX composition 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 LaTeX source whereas the latter provides a menu driven environment for using *Maxima*. This tutorial employs *wxMaxima*, which can be used to export both html and LaTeX files (though the exported files will likely require editing before being ready for publication).

We assume that you have loaded *Maxima* onto your computer and can run it from *wxMaxima*. Be aware that the `F1` key or the `Help` menu can be used to open the entire *Maxima* manual. *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. If you are not comfortable with *wxMaxima*, see the `Help` option at wxmaxima.sourceforge.net/. In particular, work through "10 Minute (wx)Maxima tutorial" (expect to spend more than 10 minutes on this).

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 is case sensitive. The cell below contains three *Maxima* commands, each ending with a semicolon. The first command yields the expected result. In the other two an improper capitalization precludes computation. The second command contains an unknown command `Sin`. The third command contains the command to return the sine of an unknown quantity (`%Pi/2`). When *Maxima* cannot interpret and command, it simply repeats that command as output.

Note that `%pi` is Maxima's way of rendering the transcendental number with that name. Entering `pi` in *wxMaxima* results in the Greek letter of that name. Enter the command `sin(pi/2);` and confirm that *Maxima* does not evaluate it.

```
(%i1) sin(%pi/2); Sin(%pi/2); sin(%Pi/2);
```

(%o1) 1

(%o2) $\mathrm{Sin}\left(\frac{\pi}{2}\right)$

(%o3) $\sin\left(\frac{\Pi}{2}\right)$

Ending a command with a dollar sign results in the command being carried out without printing the results. In the cell below, the values 1 and 2 are bound to the names a and `b`. Both commands end with the dollar sign so the results are not printed. The third command, which spreads over two lines, ends with a semicolon, so its result is printed. *Maxima* ignores spaces and line breaks.

```
(%i4) a:1$ b:2$
      a +
      b;
```

(%o6) 3

One might interpret the two expressions above as equivalent to `a = 1` and `b = 2`. *Maxima* does not. The equal sign is reserved for expressions that are temporarily true, but the statement of equality is not retained indefinitely. Consider the two expressions below.

```
(%i7) c = 5; c;
```

(%o7) $c=5$

(%o8) c

The equality below is expressed with the equal sign. The solve command is applied to the expression, using the `%` symbol to refer to the most recently generated output. Maxima no longer retains any memory of the expression *2*x + 1 = 7*.

```
(%i9) 2*x + 1 = 7; solve(%, x);
```

(%o9) $2x+1=7$

(%o10) $[x=3]$

We can bind the expression to a name, such as `expr1`. and then use `expr1` in place of the expression itself. The command below binds this expression to a name, then binds the solution of the expression to another name, and finally evaluates the expression given the solution.

```
(%i11) expr1: 2*x + 1 = 7$ solnx: solve(expr1, x); ev(expr1, solnx);
```

(%o12) $[x=3]$

(%o13) $7=7$

A very simple use of *Maxima* is as an infinite precision scientific calculator. To illustrate this use, consider the following, where `bfloat` invokes *Maxima*'s `bigfloat` option, and `fpprintprec:60` sets the floating point print precision.

```
(%i14) fpprec:60$ q : sqrt(2); 1 + q;
       bfloat(q); bfloat(1+q);
```

(%o15) $\sqrt{2}$

(%o16) $\sqrt{2}+1$

(%o17) 1.41421356237309504880168872420969807856967187537694807317668b0

(%o18) 2.41421356237309504880168872420969807856967187537694807317668b0

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.

Note the use of `f(x) :=` as a third way to relate an expression to a name. In this case a functional relationship is established.

```
(%i19) kill(all);
       values; functions;
```

(%o0) *done*

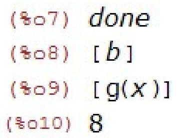
(%o1) []

(%o2) []

In the next cell two values and two functions are defined. One value and one function are killed, leaving the value b and the function `g(x)`.

```
(%i3) a:1$ b:2$ f(x) := x^2$ g(x):=x^3 $
      kill(a,f);
```

```
values; functions; g(b);
```

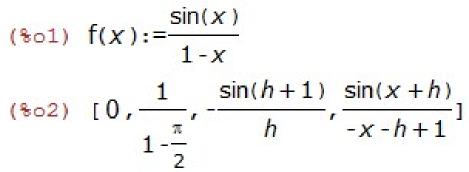


```
(%o7) done
(%o8) [b]
(%o9) [g(x)]
(%o10) 8
```

2 Finding Limits with Maxima

Maxima can be very helpful in checking limits of functions. Before we consider the limits, however, we examine the functional notation more closely. The function *f(x) = sin(x)/x* is specified and then evaluated for four values of *x*. Note the use of a list of commands and the corresponding list of output values.

```
(%i11) kill(all)$
        f(x) := sin(x)/(1-x);
        [f(0), f(%pi/2), f(1 + h), f(x + h)];
```

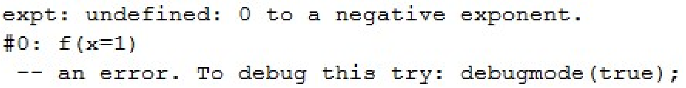


(%o1) $f(x):=\frac{\sin(x)}{1-x}$

(%o2) $[0,\frac{1}{1-\frac{\pi}{2}},-\frac{\sin(h+1)}{h},\frac{\sin(x+h)}{-x-h+1}]$

Inspection of *f(x)* indicates that *f(0)* is not defined. We may wish, however, to determine the value toward which *f(x)* tends as *x* approaches zero. The cell below confirms that *f(0)* is not defined, and it reports that the limit of *f(x)* as *x* approaches 0 is "infinity". This is an unsatisfactory reply in that this value is *Maxima*'s way of indicating a complex infinity. *Maxima* indicates a positive infinity with `inf` and a negative infinity with `minf`. *wxMaxima* converts these to the standard symbols as we see below.

```
(%i3) f(1);
```



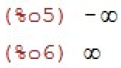
```
expt: undefined: 0 to a negative exponent.
#0: f(x=1)
 -- an error. To debug this try: debugmode(true);
```

```
(%i4) limit(f(x), x, 1);
```

(%o4) *infinity*

The difficulty that results in the imprecise "infinity" response above is removed when we tell *Maxima* whether *x* approaches 0 from above ("plus") or below ("minus").

```
(%i5) limit(f(x), x, 1, plus); limit(f(x), x, 1, minus);
```



(%o5) $-\infty$

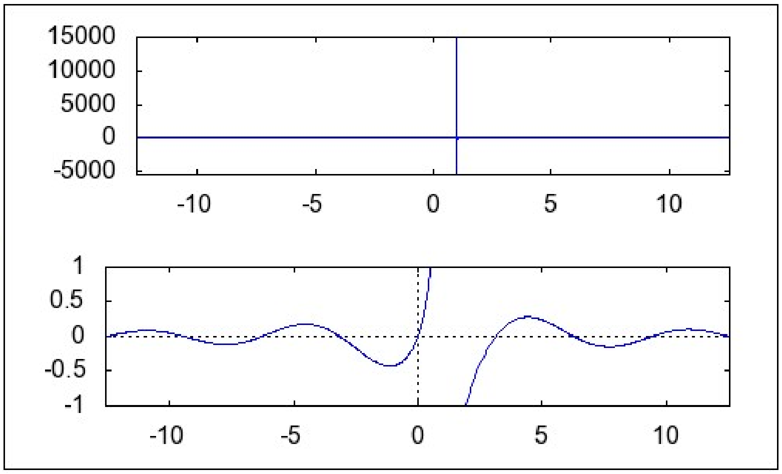
(%o6) ∞

2.1 Plotting Despite Asymptotes

Graphing this function presents a challenge, because *f(x)* grows without limit as x approaches 1. The use of draw's scenario creation and graphing feature below shows how to circumvent this difficulty. The first graph places no limits on the *f(x)* range, so the values being graphed over the relevant range cannot be discerned. Forcing `draw` to consider only *f(x)* values between -1 and 1 allows for the drawing of the graph that we require. The *x* and *y* axes are optional.

```
(%i7) graph1: gr2d( explicit(f(x), x, -4*%pi, 4*%pi) )$
      graph2: gr2d( xaxis=true, yaxis = true, yrange= [-1, 1], explicit(f(x), x, -4*%pi, 4*%pi) )$
      wxdraw( graph1, graph2 );
```

(%t9)



(%o9)

Maxima can deal with functions that involve logical expressions. The `transpose( matrix( ) )` command is used to generate a table and is not pertinent to the analysis.

```
(%i10) g(x) := if x < 2 then x^2 else sqrt(x);
       transpose( matrix([g(0), g(2.5), g(4.999), g(5), g(5.0)]) );
```

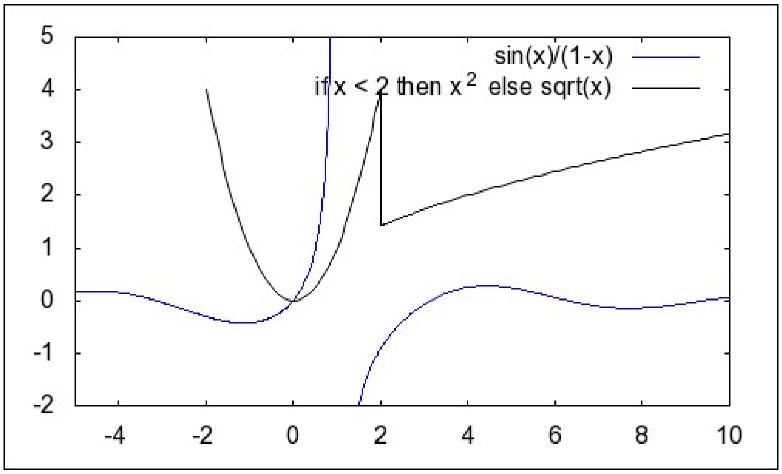
(%o10) $g(x):=\text{if } x<2 \text{ then } x^2 \text{ else } \sqrt{x}$

(%o11)
$$\begin{bmatrix} 0 \\ 1.58113883008419 \\ 2.235844359520582 \\ \sqrt{5} \\ 2.23606797749979 \end{bmatrix}$$

Maxima can also be used as a graphing calculator, and it can calculate more than one function. Observe that the two graphs are not plotted over the same range. Note the use of string to make the functional expression the key entry.

```
(%i12) wxdraw2d( yrange = [-2, 5], key = string(f(x)), explicit( f(x), x, -5, 10),
       color = black, key = string(g(x)) , explicit(g(x), x, -2, 10)
         )$
```

(%t12)



2.2 The Limit of the Difference Quotient

Being able to determine the limit of a diifference quotient can be quite helpful. The expression that characterizes the change in *f(x)* per unit change in *x* is defined below (h is the size of the change in *x*). The limit of this expression is determined. It is a quite long expression, but one that can be simplified, as shown. The `ratsimp` command results in simplfication of a rational expression.

```
(%i13) diffratio: (f(x+h) - f(x) )/h;
       dr_limit: limit(diffratio, h, 0); ratsimp(%);
```

(%o13) $$\frac{\frac{\sin(x+h)}{-x-h+1}-\frac{\sin(x)}{1-x}}{h}$$

(%o14)
$$\frac{\frac{2(x-1)\cos(x)\sin(x)}{(2x-2)\sin(x)+(-2x^2+4x-2)\cos(x)}-\frac{2(x-1)^2\cos(x)^2}{(2x-2)\sin(x)+(-2x^2+4x-2)\cos(x)}-\frac{2\sin(x)^2+(2-2x)\cos(x)\sin(x)}{(2x-2)\sin(x)+(-2x^2+4x-2)\cos(x)}}{1-x}$$

(%o15) $$\frac{\sin(x)+(1-x)\cos(x)}{x^2-2x+1}$$

The limit of the difference quotient is the derivative of the function. This equality is confirmed for the example below. The next section discusses the notation for the `diff` (derivative) command.

```
(%i16) diff(f(x), x); ratsimp(%);
```

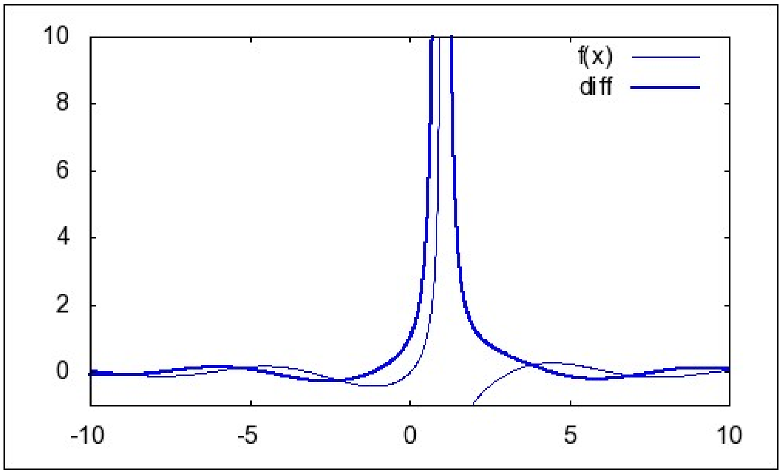
(%o16) $$\frac{\sin(x)}{(1-x)^2}+\frac{\cos(x)}{1-x}$$

(%o17) $$\frac{\sin(x)+(1-x)\cos(x)}{x^2-2x+1}$$

The graph below shows the function f(x) and its derivative for *x* from -10 to 10. In general, care should be taken when graphing *f(x)* and its derivatives, because often the *y* units are different for the two.

```
(%i18) wxdraw2d( yrange=[-1,10], key = "f(x)",
       explicit( f(x), x, -10, 10), line_width = 2, key = "diff",
       explicit( diff(f(x), x), x, -10, 10 ) )$
```

(%t18)



3 Differentiation Rules

The example below shows that *Maxima* can provide an expression for the derivative of a particular function. It can do more: *Maxima* can show the rules for differentiation, independent of the specific expression. We ensure that *Maxima* understands the dependencies in *f(x)* and *g(x)* but do not specify the functional forms.

```
(%i19) kill(all) $ depends(f, x, g, x)$
       diff(f + g, x);
       diff(f*g, x); diff(f/g, x); ratsimp(%);
```

(%o2) $\frac{d}{d\,x}\,g+\frac{d}{d\,x}\,f$

(%o3) $f\left(\frac{d}{d\,x}\,g\right)+\left(\frac{d}{d\,x}\,f\right)g$

(%o4) $\frac{\frac{d}{d\,x}\,f}{g}-\frac{f\left(\frac{d}{d\,x}\,g\right)}{g^2}$

(%o5) $-\frac{f\left(\frac{d}{d\,x}\,g\right)-\left(\frac{d}{d\,x}\,f\right)g}{g^2}$

Maxima knows the sum, product, and quotient rules (and many others).1 The quotient rule is stored in a somewhat different form than the one that is optimal for humans. The command `depends(f,x)` is required, to tell *Maxima* that *f* is a function of *x*. It can then deal with its derivative in a symbolic way. If we had not specified these dependencies, 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.

The cell below contains the same commands as above, but the `depends` command has been "commented out." Not knowing about these dependencies, *Maxima* returns 0's.

```
(%i6) kill(all) $ /* depends(f, x, g, x)$ */
      diff(f + g, x);
      diff(f*g, x); diff(f/g, x);
```

(%o1) 0

(%o2) 0

(%o3) 0

3.1 Exracting and Manipulating Results

A variety of instructions control the final form of the answer that *Maxima* returns. If we wish the answer to be over a common denominator, for example, then the instruction is `factor`.

```
(%i4) kill(all)$
      f(x) := x^(1/2); g(x) := 1 + x^2;
      answer: diff(f(x)/g(x), x);
      factor(answer);
```

(%o1) $\mathrm{f}(x):=x^{1/2}$

(%o2) $\mathrm{g}(x):=1+x^2$

(%o3) $\frac{1}{2\sqrt{x}\,(x^2+1)}-\frac{2\,x^{3/2}}{(x^2+1)^2}$

(%o4) $-\frac{3\,x^2-1}{2\sqrt{x}\,(x^2+1)^2}$

To access just parts of an answer provided by *Maxima*, 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 the example above 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. Use the smallest value that gets the desired label.

```
(%i5) pickapart( factor(answer),2);
```

(%t5) $3\,x^2-1$

(%t6) $2\sqrt{x}\,(x^2+1)^2$

(%o6) $-\frac{\%t5}{\%t6}$

Maxima has attached labels (`%t5`) and (`%t6`) to the numerator and denominator of the factored form of the answer. (These numbers vary, depending on whether *Maxima* has produced any such labels before this `pickapart` command.) To see what happens if the level 2 is replaces by 1, 3, and so forth.

To find the zeros of the numerator, do as follows:

```
(%i7) soln: solve(%t5,x);
```

$$\text{(\%o7)}\quad [x=-\frac{1}{\sqrt{3}}, x=\frac{1}{\sqrt{3}}]$$

The resulting output, named `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 that 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, or with the appropriate subscript, indicated by square brackets.

```
(%i8) the_first_soln: soln[1]; /* or */ first(soln);
```

$$\text{(\%o8)}\quad x=-\frac{1}{\sqrt{3}}$$

$$\text{(\%o9)}\quad x=-\frac{1}{\sqrt{3}}$$

Each item in `soln` is an expression. We can use the command `rhs` (right-hand side) to extract the value. It can be bound to a name.

```
(%i10) the_first_soln_value : rhs(soln[1]);
```

$$\text{(\%o10)}\quad -\frac{1}{\sqrt{3}}$$

3.2 Derivatives of Trigonometric 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:

```
(%i11) kill(all)$ diff( sin(x)/(1 + cos(x)),x);
        factor(%); trigsimp(%);
```

$$\text{(\%o1)}\quad \frac{\sin(x)^2}{(\cos(x)+1)^2}+\frac{\cos(x)}{\cos(x)+1}$$

$$\text{(\%o2)}\quad \frac{\sin(x)^2+\cos(x)^2+\cos(x)}{(\cos(x)+1)^2}$$

$$\text{(\%o3)}\quad \frac{1}{\cos(x)+1}$$

The instruction `trigsimp` instructs *Maxima* to make the obvious simplification using the Pythagorean identity. The other *Maxima* instruction is `trigreduce` which allows using the multiple angle formulas to reduce powers, e. g:

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

$$\text{(\%o4)}\quad 2\sin(x)^2+\cos(x)^2$$

$$\text{(\%o5)}\quad \sin(x)^2+1$$

$$\text{(\%o6)}\quad \frac{\cos(2x)+1}{2}+2\left(\frac{1}{2}-\frac{\cos(2x)}{2}\right)$$

The latter expression might not appear to be simpler than the one we started with. It is invaluable, however, for integration, the inverse process of differentiation.

3.3 The Chain Rule

Before *Maxima* can apply the chain rule, it must know that the relevant dependencis exist. In the example below, `f(x):=` defines an explicit relationship between *x* and *f(x)*. The existence of relationship between *x* and *u* is asserted, but the relationshhip is not defined.

```
(%i7) f(x):= x^3;
      depends(x,u)$
      diff(f(x),u);
```

(%o7) $f(x):=x^3$

(%o9) $3\,x^2\left(\frac{d}{d\,u}x\right)$

The dialogue above uses the functional notation to define *f* and uses the instruction depends to inform Maxima that *x* is a function of *u*. It did not, however, provide a specific formula for this dependency. We can, however, specify the dependency of *x* on *u* as follows:

```
(%i10) remove([x,u],dependency)$
        x: sin(u); diff(f(x),u);
```

(%o11) $\sin(u)$

(%o12) $3\cos(u)\sin(u)^2$

Alternatively, we can use functional notation for both functions and get *Maxima* to differentiate their composition. Note that in this case *g(u*) and not *x* must be entered as the expression to be differentiated.

Note that `diff(f(x), u)` results in 0, because *Maxima* no longer remembers the dependency of *x* on *u*.

```
(%i13) kill(x)$ g(x):= sin(x); diff(f(g(u)),u); diff(f(x), u);
```

(%o14) $g(x):=\sin(x)$

(%o15) $3\cos(u)\sin(u)^2$

(%o16) 0

The instruction `kill` was needed to remove the relationships that had been set. 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.

3.4 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 $x^2 + y^2 = 25$.

```
(%i17) eqn: x^2 + y^2 = 25;
       depends(y,x)$
       deriv_of_eqn : diff(eqn,x);
       solve(deriv_of_eqn,'diff(y,x));
```

(%o17) $y^2+x^2=25$

(%o19) $2\,y\left(\frac{d}{d\,x}y\right)+2\,x=0$

(%o20) $[\frac{d}{d\,x}y=-\frac{x}{y}]$

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 *dy/dx* is the second argument. Also, note the single quote in front of `diff(y,x)`. A single quote in front of a symbol tells *Maxima* not to evaluate the symbol but to treat it as an unknown quantity. For example,

```
(%i21) [a, b] : [4, 3]$
       [a + b, 'a + b, a + 'b, 'a + 'b, '(a + b) ];
```

(%o22) $[7,a+3,b+4,b+a,b+a]$

Likewise, then, 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 anyway) but to regard `diff(y,x)` as an unknown quantity and solve for it from the differentiated expression, named `deriv_of_eqn`.

Higher Derivatives. 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. The first command below is equivalent to `diff(x^n, x, 1`). (Confirm this.) The second in the list of commands calls for the fourth derivative. The third command calls for the *n*-th derivative. Until *n* is specified, this expression cannot be evaluated.

Exercise: Confirm that for the expression x^8 the 8-th derivative equals 8!. Does this generalize to any value of *n*? Explain.

```
(%i23) kill(n)$
       [diff(x^n, x), diff(x^n, x, 4), diff(x^n, x, n)];
       n:8$
       [diff(x^n, x), diff(x^n, x, 4), diff(x^n, x, n)];
```

(%o24) $[n\,x^{n-1},(n-3)(n-2)(n-1)\,n\,x^{n-4},\frac{d^{n}}{d\,x^{n}}x^{n}]$

(%o26) $[8\,x^{7},1680\,x^{4},40320]$

3.5 Related Rates

Maxima can help in solving related rates problems. For example consider the problem of finding the rate of change of the area of a circle given that the rate of rate of change of the circle's radius is *dr/dt = 60* when *t = 2* and *r = 120*:

```
(%i27) kill(a, r)$
       area: a = %pi*r^2;
       depends([a,r],t)$
       deriv_of_area: diff(area,t);
       subst([diff(r,t)=60, r=120],deriv_of_area);
       float(rhs(%));
```

(%o28) $a=\pi\,r^{2}$

(%o30) $\frac{d}{d\,t}a=2\,\pi\,r\left(\frac{d}{d\,t}r\right)$

(%o31) $\frac{d}{d\,t}a=14400\,\pi$

(%o32) 45238.93421169302

Note that *Maxima* must be told which variables are time-dependent with the instruction `depends([a list of variables], t)`. The instruction `subst` instructs *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 argument.

Maxima's default is to produce exact calculations, not numeric approximations. Thus, *da/dt = 14400π*. The `float(%)` command instructs *Maxima* to provide the floating-point approximation to the exact value. You might want to see how many digits of `%pi` (`Maxima`'s way of denoting the number as opposed to the Greek letter) Maxima can find by giving the instructions `fpprec= 1000$ fpprintprec:1000$ bfloat(%pi);`. The commands `fpprec` and `fpprintprec` set the floating-point precision and the floating-point print precision. These need not have the same value.

3.6 Linear Approximations and Differentials

We can use the diff operation to express the differential of an equation as a linear function of its arguments. The next cell defines *f(x)* and a differential equation based on that function. The command `diff` does not specify a variable with which to take a derivative, so *Maxima* returns *dy = (dy/dx) dx*, where *dx* is denoted `del(x)`.

The expression is picked apart for use below.

```
(%i33) kill(all)$ f(x) := sin(x);
       diff(f(x) );
       pickapart(%, 1)$
```

(%o1) $\mathrm{f}(x):=\sin(x)$

(%o2) $\cos(x)\,\mathrm{del}(x)$

(%t3) $\cos(x)$

(%t4) $\mathrm{del}(x)$

To create an equation that produces a line tangent to *f(x)* at the value *x* = *π/3*, we evaluate the term *dy/dx* at that value. Then we create a linear function with that slope that passes through the point *(π/3, f(π/3))*.

```
(%i5) ev(%t3, x = %pi/3);
      f(%pi/3) + %*(x - %pi/3);
      float(%); expand(%);
```

(%o5) $\frac{1}{2}$

(%o6) $\frac{x-\frac{\pi}{3}}{2}+\frac{\sqrt{3}}{2}$

(%o7) $0.5\,(x-1.047197551196598)+0.86602540378444$

(%o8) $0.5\,x+0.34242662818614$

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

```
(%i9) L: taylor(f(x), x, %pi/3, 1);
```

(%o9)/T/ $\frac{\sqrt{3}}{2}+\frac{x-\frac{\pi}{3}}{2}+\ldots$

The ellipsis (...) after the output denotes a residual, and the `/T/` before the output indicates that the terms reported constitute a truncated representation of a polynomial of higher degree. We use `taytorat` to convert the Taylor expansion to a rational expression. The second and third commands below are used to express the result in a manner that makes it comparable to the result above. The two are equivalent.

```
(%i10) L: taytorat(L);
       expand(L); float(%);
```

(%o10)/R/ $\frac{x-\frac{\pi}{3}+\sqrt{3}}{2}$

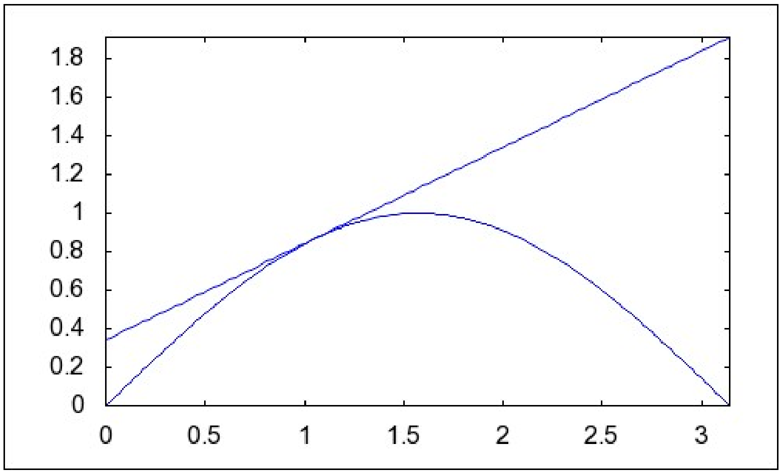
(%o11) $\frac{x}{2}-\frac{\pi}{6}+\frac{\sqrt{3}}{2}$

(%o12) $0.5\,x+0.34242662818614$

The graph below shows the linear approximation to the function, along with the function itself. The graph makes obvious one important aspect of the linear approximation: It is best near the value for which the Taylor expansion is computed.

```
(%i13) wxdraw2d(
       explicit(f(x), x, 0, %pi), explicit(L, x, 0, %pi)
       )$
```

(%t13)



Taylor polynomials come in all degrees. You will work with them in a systematic way later. The last argument in the *Maxima* instruction Taylor specifies the degree.

3.7 Locating Maxima and Minima

Maxima can be a great help in checking the process of locating critical points of functions, as the following dialogue shows.

```
(%i14) kill(all)$
       f(x):=x^(3/5)*(4-x);
       deriv_of_f: diff(f(x),x);
        soln: solve(deriv_of_f, x); x_crit: rhs(soln[1]);
       f(x_crit) ; float(%);
```

(%o1) $f(x):=x^{3/5}(4-x)$

(%o2) $\frac{3(4-x)}{5x^{2/5}}-x^{3/5}$

(%o3) $[x=\frac{3}{2}]$

(%o4) $\frac{3}{2}$

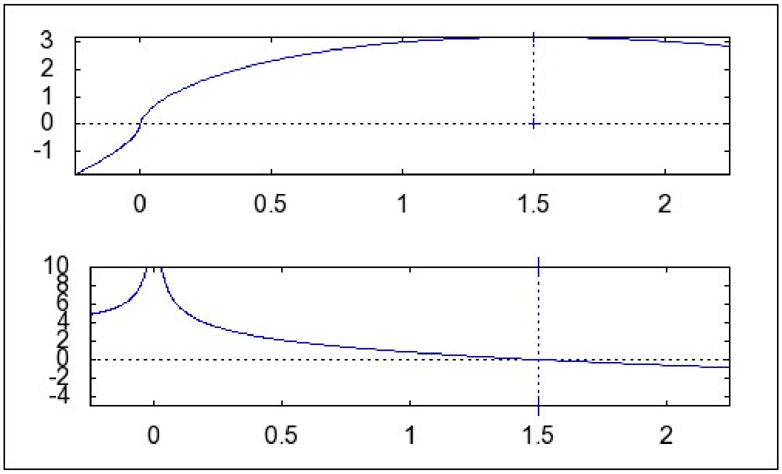
(%o5) $\frac{5\,3^{3/5}}{2^{8/5}}$

(%o6) 3.188561251564477

Examination of output %o2, the derivative function, shows that *df/dx* is not defined when *x = 0*. The graphs below show the behavior of *f* and *df/dx*. The derivative grows without bound as *x* approaches 0 (use the limit command to confirm this), but *f* is continuous.

```
(%i7) fgraph: gr2d( xaxis = true, explicit(f(x), x, -0.25 , 1.5*x_crit),
      line_type = dots,
      points_joined=true, points( [ [x_crit, 0], [x_crit, f(x_crit) ] ]) )$
      diffgraph: gr2d(xaxis = true, yrange=[-5, 10],
      explicit(deriv_of_f, x, -.25, 1.5*x_crit),
      point_size = 2, points_joined = true, line_type=dots,
      points( [[x_crit,-5],[x_crit, 10] ])
      ) $
      wxdraw(fgraph, diffgraph)$
```

(%t9)



Have we found a maximum or a minimum? The condition that *df/dx = 0* is a necessary condition for a value of *f(x)* to be a (local) maximum. It is also a necessary conditon for the value of *f(x)* to be a (local) minimum. To determine which of the two has been discovered requires that we examine the second derivative of *f* with respect to *x*. If this second derivative is negative, then the critical value of *x* yields a (local) maximum value of *f(x)*; if it is positive, a (local) minimum has been found. In our case, the result is a local (and global, the graph suggests) maximum value of *f(x)*.

```
(%i10) second_deriv: diff(f(x), x, 2);
```

(%o10) $-\frac{6}{5x^{2/5}}-\frac{6(4-x)}{25x^{7/5}}$

Inspection of the expression above shows that the second derivative is negative. We can determine its exact value by using `subst` or `ev`. Note the alternative ways of converting the exact answer to a floating point value.

```
(%i11) ev(second_deriv, x=x_crit)$ float(%);
       float( subst(x_crit, x, second_deriv) );
```

(%o12) -1.36045280066751

(%o13) -1.36045280066751

3.8 Optimization: Finding Maximum or Minimum Values of Functions Subject to a Constraint

We use *Maxima* to solve the following. A rectangular window is 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. We denote the fixed value of *P* as *P0*.

```
(%i14) kill(all)$
       P : 2*r +2*h + %pi*r$ L : 2*r*h + %pi*r^2/4$
       print("Perimeter = ", P, ", and Light = ", L)$
       soln_h : solve(P = P0,h);
```

$$\mathit{Perimeter} = \pi r+2r+2h,\ \mathit{and\ Light} = \frac{\pi r^2}{4}+2hr$$

(%o4) $[h=\frac{P0+(-\pi-2)r}{2}]$

We extract the righthand side of the solution and assign it the name `h_is`. This expression is substiuted into the *L* expression, yielding *L* as a function of *r* (recall that *P* has a constant value *P0*).

```
(%i5) h_is : rhs(first(soln_h));
      L_fcn_r : ratsubst(h_is,h,L);
```

(%o5) $\frac{P0+(-\pi-2)r}{2}$

(%o6) $-\frac{(3\pi+8)r^2-4\,r\,P0}{4}$

Taking the derivative of `L_fcn_r` and setting the derivative equal to zero yelds an expression for *r*: *r* is proportional to *P0*.

```
(%i7) deriv_L :diff(L_fcn_r,r);
      soln_r:solve(deriv_L = 0,r);
```

(%o7) $-\frac{2(3\pi+8)r-4\,P0}{4}$

(%o8) $[r=\frac{2\,P0}{3\pi+8}]$

We assign the right hand side of the solution the name `r_is` and we substitute `r_is` into the expression `h_is`, yielding the expression `h_is_now`, which shows that *h*, like *r*, is proportional to *P*.

That that the critical values of both *h* and *r* are proportional to *P0* implies that they are proportional to each other. The third command shows the ratio of height to the radius.

```
(%i9) r_is: rhs(soln_r[1]);
      h_is_now: subst(r_is, r, h_is), ratsimp;
      h_is_now / r_is, ratsimp;
```

(%o9) $\frac{2\,P0}{3\pi+8}$

(%o10) $\frac{(\pi+4)P0}{6\pi+16}$

(%o11) $\frac{\pi+4}{4}$

Finally, we can determine the maximum light, given the window's perimeter.

```
(%i12) max_L : subst([r=r_is, h=h_is_now], L)$
       ratsimp(%);
```

(%o13) $\frac{P0^2}{3\pi+8}$

An alternative approach to solving problem like this one is to differentiate the constraint equation. This is significantly simpler than solving the constraint equation for one of the variables in cases where the constraint equation is complicated or perhaps not even solveable for either of the variables.

In the above problem we differentiate the equation for the perimeter with respect to *r*. This yields a linear equation in the unknown derivative *dh/dr* which can easily be solved. Then we differentiate the formula for *L* with respect to *r* and this would also involve the unknown derivative *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.

```
(%i14) kill(all)$ depends(h,r)$
       P : 2*r +2*h + %pi*r$
       L : 2*r*h + %pi*r^2/4$
       deriv_P : diff(P-P0,r);
```

(%o4) $2\left(\frac{d}{d\,r}h\right)+\pi+2$

```
(%i5) solve(deriv_P, diff(h, r));
      deriv_h : rhs(%[1]);
```

(%o5) $[\frac{d}{d\,r}h=-\frac{\pi+2}{2}]$

(%o6) $-\frac{\pi+2}{2}$

```
(%i7) deriv_L : diff(L,r);
```

(%o7) $2\left(\frac{d}{d\,r}h\right)r+\frac{\pi\,r}{2}+2\,h$

```
(%i8) deriv_L_is: ratsubst(deriv_h, 'diff(h,r), deriv_L);
```

(%o8) $-\frac{(\pi+4)\,r-4\,h}{2}$

```
(%i9) solve(deriv_L_is = 0, r);
```

(%o9) $[r=\frac{4\,h}{\pi+4}]$

3.9 Optimization with More than One Variable

Suppose that we wish to construct a box that has a given surface area in a configuration that yields the maximum volume. The Lagrangian approach to solving a problem like this is to create an augmented expression that incorporates the constraint. Expression A in the cell below adds an undetermined multiplier mu times the constraint to the expression for volume. The constraint is expressed so that the value by which mu (μ) is being multiplied is zero.

```
(%i10) kill(all)$
       V: L*W*H; S: 2*(L*W + L*H + W*H);
       A: V + mu*(S0 - S);
```

(%o1) $H\,L\,W$

(%o2) $2\,(L\,W+H\,W+H\,L)$

(%o3) $\mu\,(S0-2\,(L\,W+H\,W+H\,L))+H\,L\,W$

We now take all possible derivatives of the augmented expression, including a derivative with respect to μ, which is the constraint. The last line is added only to make the output easier to read.

We solve the system of equations for the derivatives with respect to *L*, *W*, and *H*. The value of μ will be implied, as we see below.

```
(%i4) A_L : diff(A, L);
      A_W : diff(A, W);
      A_H : diff(A, H);
      A_mu : diff(A, mu);
      soln: solve([A_L, A_W, A_H], [L, W, H]);
      transpose( matrix(soln) );
```

```
(%o4) H W-2 μ(W+H)
(%o5) H L-2 μ(L+H)
(%o6) L W-2 μ(W+L)
(%o7) S0-2(L W+H W+H L)
(%o8) [[L=0,W=0,H=0],[L=4 μ,W=4 μ,H=4 μ]]
      ⎡    [L=0,W=0,H=0]    ⎤
(%o9) ⎢                     ⎥
      ⎣[L=4 μ,W=4 μ,H=4 μ]  ⎦
```

Maxima returns two solutions. The first, $L = W = H = 0$, yields a minimum. In this case, μ must equal zero in order for $S0 - S = 0$ to hold ($S0 > 0$ and $S = 0$). The interpretation is the adding to S does not further decrease V; a minimum has been attained.

The second solution provides the conditons for a maximum V. Here, $L = W = H = 4*\mu$. The example below illustrates this result. Suppose that $S = 60$ meters2. Then $2*(3*L^2) = 60$, so L = √(10) meters.

```
(%i10) solnL0: solve(60 - 6*L^2, L);
```

```
(%o10) [L=-√10,L=√10]
```

The resulting volume is $10^{(3/2)}$ meters^3, approximately 31.6228 meters^3.

```
(%i11) L0: rhs(solnL0[2]);
       V0: ev(V, L = L0, W = L0, H = L0);
       float(%);
```

```
(%o11) √10
(%o12) 10^(3/2)
(%o13) 31.6227766016838
```

The value of μ is sqrt(10)/4, approximately 0.7906. Therefore a change in S from 60 to 61 meters^2 should increase V by approximately 0.7906 meters^3. The results below show that the actual increase is approximately 0.7939 meters^3. Confirm that the discrepancy between $L0/4 = \mu$ and the actual value decreases as h decreases by using smaller values of *h0*.

We can confirm that μ has the same units as L, W, and H: μ is the ratio of the change in volume (meters^3) to change in surface (meters^2). Thus, the unit for μ is meters, the same as for L, W, and H.

```
(%i14) h0: 1$ S1: 60 + h0$
       solnL1 : solve(S1 - 6*L^2, L)$ L1: rhs(solnL1[2])$
       V1: ev( V, L = L1, W = L1, H = L1)$ float(%);
       float( [(V1 - V0)/h0, L0/4]);
```

```
(%o19) 32.41663096256246
(%o20) [0.79385436087866,0.79056941504209]
```

The Lagrangian technique can be applied to multiple constraints. Suppose that the height of the box is required to equal one-half the length. We define a new augmented function that adds this constraint to augmented function A. We then generate the set of derivatives and solve for *L, W, H*, and kappa (κ), the new multiplier.

```
(%i21) A2 : A + kappa*(.50 - H/L);
       A2_L : diff(A2, L);
       A2_W : diff(A2, W);
       A2_H : diff(A2, H);
       A2_mu : diff(A2, mu);
       A2_kappa: diff(A2, kappa);

       soln2: solve([A2_L, A2_W, A2_H, A2_kappa], [L, W, H, kappa])$
       transpose( matrix(soln2) );
```

(%o21) $\mu(S0-2(L\,W+H\,W+H\,L))+H\,L\,W+\kappa\left(0.5-\frac{H}{L}\right)$

(%o22) $-2\,\mu(W+H)+H\,W+\frac{\kappa\,H}{L^2}$

(%o23) $H\,L-2\,\mu(L+H)$

(%o24) $-2\,\mu(W+L)+L\,W-\frac{\kappa}{L}$

(%o25) $S0-2(L\,W+H\,W+H\,L)$

(%o26) $0.5-\frac{H}{L}$

```
rat: replaced 0.5 by 1/2 = 0.5
```

(%o28) $\begin{bmatrix} [L=0,W=\%r1,H=0,\kappa=0] \\ [L=6\,\mu,W=4\,\mu,H=3\,\mu,\kappa=24\,\mu^3] \end{bmatrix}$

Again, Maxima produces two solutions. The first shows the conditions for minimizing the volume, and the second for maximizing it. Now the values of *L, W*, and *H* are all different from each other.

The condition that κ = *24*μ^2 warrants attention. The value of κ is proportional to the box's volume (recall that μ is in the same units as *L, W*, and *H*).

```
(%i29) [L2 : rhs(soln2[2][1]), W2 : rhs(soln2[2][2]), H2 : rhs(soln2[2][3]),
       kappa2 : rhs(soln2[2][4])];
       soln_mu2 : solve(2*(L2*W2 + L2*H2 + W2*H2)- 60, mu)$ float(%);
```

(%o29) $[6\,\mu,4\,\mu,3\,\mu,24\,\mu^3]$

(%o31) $[\mu=-0.74535599249993,\mu=0.74535599249993]$

The value of mu falls from approximately 0.7906 to approximately 0.7454, so less volume is added per unit change in the surface area. We reevaluate *L2, W2, H2*, and κ*2* given the positive value of μ -- `soln_mu2[2]` -- and then compute the new volume.

The value of κ estimates that a 0.01 unit increase in the ratio (from 0.5 to 0.51) will cause volume to rise by appoximately 0.0994 meters^3. Replace 0.5 with 0.51 above and confirm that the actual increase is approximately 0.0973.

```
(%i32) [L2, W2, H2, kappa2] : ev([L2, W2, H2, kappa2], soln_mu2[2])$ float(%);
       V2 : L2 * W2 * H2$ float(%);
```

(%o33) $[4.47213595499958,2.98142396999972,2.23606797749979,9.938079899999067]$

(%o35) 29.8142396999972

For *S = 60*, the maximum area is now approximately 29.8142 meters^3, down from approximately 32.4166 meters^3 when the relationships among *L*, *W*, and *H* are not constrained.

4 Integration

The syntax of indefinite integration resembles that of differentiation, with integrate replacing diff, except that multiple integration requires multiple calls to integrate. The general expressions below show that *Maxima* does not report a constant of integration. It is up to the analyst to keep track.

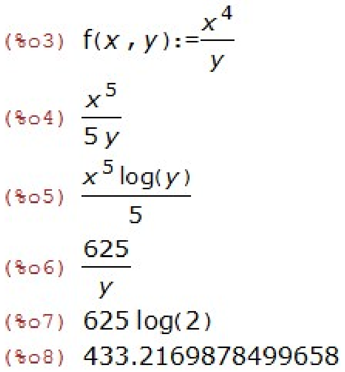
```
(%i36) kill(all)$
       /* Indefinite integrals */
       ["Single integral: ", integrate(f(x), x),
          " Double integral: ", integrate(integrate(f(x,y), y), x ) ];
       /* A definite integral */
       ["Single definite integral: ", integrate(f(x), x, a, b) ];
```

(%o1) $[\textit{Single integral: },\int \mathrm{f}(x)dx,\quad \textit{Double integral: },\int\int \mathrm{f}(x,y)dy\,dx]$

(%o2) $[\textit{Single definite integral: },\int_a^b \mathrm{f}(x)dx]$

With the same function *f (x, y) = x*^{4}*/y*, we find the indefinite and definite integrals of the expression with respect to *x*. The definite integral is over the range *x = 0* to *x = 5*. Then the corresponding integrals of the resulting expression with respect to *y*, with the definite integral taken over the range *y = 1* to *y = 2*.

```
(%i3) f(x, y) := x^4 / y;
      integrate(f(x, y),x ); integrate(%, y);
      integrate(f(x, y), x, 0, 5); integrate(%, y, 1, 2); float(%);
```



(%o3) $f(x,y):=\frac{x^4}{y}$

(%o4) $\frac{x^5}{5y}$

(%o5) $\frac{x^5\log(y)}{5}$

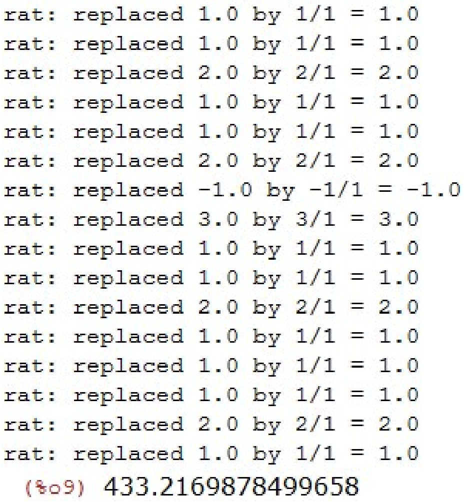
(%o6) $\frac{625}{y}$

(%o7) $625\log(2)$

(%o8) 433.2169878499658

The output can also be generated in a single nested command. Setting the limits as floating point numbers invokes a set of statements specifying the substitutions that *Maxima* made while carrying out the operation. Adding the command `ratprint: false` as the first line would suppress this output, which is often not useful.

```
(%i9) integrate(integrate(f(x, y), x, 0, 5), y, 1.0, 2.0 );
```

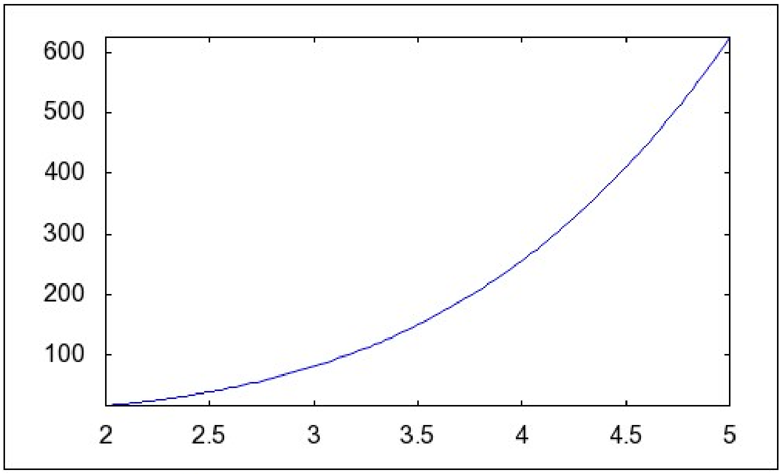


```
rat: replaced 1.0 by 1/1 = 1.0
rat: replaced 1.0 by 1/1 = 1.0
rat: replaced 2.0 by 2/1 = 2.0
rat: replaced 1.0 by 1/1 = 1.0
rat: replaced 1.0 by 1/1 = 1.0
rat: replaced 2.0 by 2/1 = 2.0
rat: replaced -1.0 by -1/1 = -1.0
rat: replaced 3.0 by 3/1 = 3.0
rat: replaced 1.0 by 1/1 = 1.0
rat: replaced 1.0 by 1/1 = 1.0
rat: replaced 2.0 by 2/1 = 2.0
rat: replaced 1.0 by 1/1 = 1.0
rat: replaced 1.0 by 1/1 = 1.0
rat: replaced 1.0 by 1/1 = 1.0
rat: replaced 2.0 by 2/1 = 2.0
rat: replaced 1.0 by 1/1 = 1.0
 (%o9) 433.2169878499658
```

The limit of a Riemann sum, if it exists, defines the definite integral of a function on [*a, b*]. The actual evaluation of the limit of the Riemann sums associated to a function on an interval is tedious. *Maxima* can help. *Maxima*'s instruction for summing a series is sum. The sum command takes four arguments: the expression, the index, the beginning value of the index, and the ending value of the index.

As before, the `print` command is not required, but it makes the result easier to read.

```
(%i10) kill(R_sum)$
       wxdraw2d( explicit(x^4, x, 2, 5) )$
       print( R_sum: (3/n)*sum((2+3*i/n)^4,i,1,n), " = ", ratsimp(R_sum) )$
```

(%t11) 

$$\frac{3\sum_{i=1}^{n}\left(\frac{3\,i}{n}+2\right)^4}{n}=\frac{3\sum_{i=1}^{n}16\,n^4+96\,i\,n^3+216\,i^2\,n^2+216\,i^3\,n+81\,i^4}{n^5}$$

Maxima's default is not to carry out the evaluation of the sum of a series. The instruction to change this is `simpsum : true`.

```
(%i13) simpsum:true$ print( "The Riemann sum of x^4 is ",
          R_sum: ratsimp( (3/n)*sum((2+3*i/n)^4,i,1,n) ),
          ", whose limit is ",
          float( limit(R_sum, n, infinity) ) )$
```

The Riemann sum of x^4 is $\frac{6186\,n^4+9135\,n^3+3510\,n^2-81}{10\,n^4}$*, whose limit is* 618.6

The result of the corresponding definite integral confirms that the integral does equal the limit of the Riemann sum.

```
(%i15) float( integrate(x^4, x, 2, 5) );
```

(%o15) 618.6

4.1 The Fundamental Theorem of Calculus

That *Maxima* knows the Fundamental Theorem is illustrated below. The extensions, labeled "Chain rule illustrations," show the flexibility of the `lambda` function. Note the use of the single-quote operator to suppress evaluation.

```
(%i16) kill(all)$
       assume(x>0)$
       S : lambda( [x], integrate(a*log(t),t,0,x));
       answr:diff(S(x),x);
       ["Chain rule illustrations: ", 'diff(S(x^3),x), " = ", diff(S(x^3),x) ] ;
       [ 'diff(S((1/x)),x) , " = ", diff(S((1/x)),x)];
```

(%o2) $\mathrm{lambda}\left([x],\int_0^x a\log(t)\,dt\right)$

(%o3) $a\log(x)$

(%o4) $[\textit{Chain rule illustrations: },\frac{d}{dx}\left(a\,(3\,x^3\log(x)-x^3)\right),\ =\ ,9\,a\,x^2\log(x)]$

(%o5) $[\frac{d}{dx}\frac{a\,(\log(x)-1)}{x},\ =\ ,\frac{a}{x^2}-\frac{a\,(\log(x)-1)}{x^2}]$

Without the command `assume(x > 0)` *Maxima* will ask the user to declare the sign of *x*.

The name of the dummy variable *t* in the definition of *S* cannot be chosen to `be` *x* even though that is perfectly acceptable in mathematical communications.

4.2 The Commands desolve and ode2

The *Maxima* instruction `integrate` serves either of 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 *dy/dx = f (x)*, or *dy = f(x) dx* 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. The `ode2` command follows the familiar integrate command.

```
(%i6) eqn: 'diff(y,x) = sqrt(1/x^2-1/x^3);
      integrate(rhs(eqn), x);
      ode2(eqn,y,x);
```

(%o6) $\frac{d}{d\,x}\,y=\sqrt{\frac{1}{x^2}-\frac{1}{x^3}}$

(%o7) $-\frac{2\sqrt{x-1}}{\sqrt{x}}+\log\left(\frac{\sqrt{x-1}}{\sqrt{x}}+1\right)-\log\left(\frac{\sqrt{x-1}}{\sqrt{x}}-1\right)$

(%o8) $y=-\frac{2\sqrt{x-1}}{\sqrt{x}}+\log\left(\frac{\sqrt{x-1}}{\sqrt{x}}+1\right)-\log\left(\frac{\sqrt{x-1}}{\sqrt{x}}-1\right)+\%c$

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 *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 response.

Here `ode` stands for *ordinary differential equation*. From the *Maxima Manual*: "The function `ode2` solves an ordinary differential equation (ODE) of first or second order." Another Maxima command, `desolve`, can be used to solve systems of one or more ordinary differential equations.

4.3 Intersections and Areas Between Curves

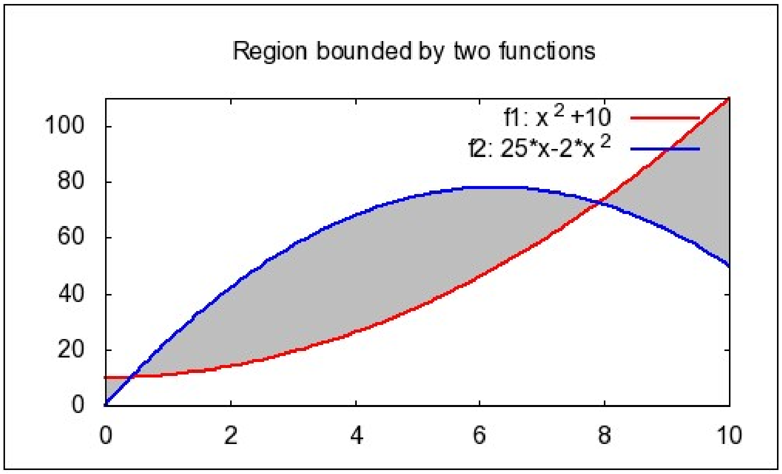
`Maxima` can be very helpful in finding the points of intersection of curves. The cell below demonstrate how.

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, we first find the points of intersection and then instruct *Maxima* to sum the integrals of the upper function minus the lower function over the relevant interval(s).

The `draw2d` command below has much more detail than is required to solve this problem, but such detail can be useful. The *Maxima Manual* (Chapter 48) contains more information regarding options for the `draw` command.

```
(%i9) kill(all)$
      [f1, f2] : [x^2 + 10, -2*x^2 + 25*x]$
      wxdraw2d(title = "Region bounded by two functions",
             fill_color = gray, filled_func = f2,
             explicit(f1,x,0,10), filled_func = false,
                   line_width = 2, key = concat("f1: ", string(f1) ),
             color = red, explicit(f1,x,0,10),
             key = concat("f2: ", string(f2) ), color = blue,
             explicit(f2,x,0,10) );
```

(%t2)



(%o2)

Use solve to determine the range for which *f2 > f1*.

```
(%i3) soln : solve(f1=f2, x)$ float(%);
```

(%o4) $[x = 0.42129915762596, x = 7.912034175707371]$

The area to be determined is the integral of *f2 - f1* over this interval.

```
(%i5) x1 : rhs(soln[1])$ x2: rhs(soln[2])$
      float( integrate(f2 - f1, x, x1, x2) );
```

(%o7) 210.1567324517284

4.4 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 does help in 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 the integral of the expression *V = sec*^{2}*(x) - 1* over the indicated range.

The command `trigsimp` (trigonometric simplification) is used for trigonometric expressions as `ratsimp` is used for rational expressions.

```
(%i8) A : (sec(x))^2 - 1;
      B : trigsimp(A);
      integrate(A, x, -1, 1);
      integrate(B, x, -1, 1); float(%);
```

(%o8) $\sec(x)^2 - 1$

(%o9) $\dfrac{\sin(x)^2}{\cos(x)^2}$

```
Principal Value
```

(%o10) $2\tan(1) - 2$

```
Principal Value
```

(%o11) $2\tan(1) - 2$

(%o12) 1.114815449309805

As the output indicates, *Maxima* could have found the integral without the instruction `trigsimp`, but one should be aware that it is available. This instruction causes *Maxima* to use the Pythagorean identities to simplify an expression. An instruction like `ratsimp` would have no effect here. The units in *Maxima* are radians, so *tan(1)* is the tangent when *x = 1* radian, or π/180 degrees.

The note "Principal Value" indicates how Maxima approaches this problem. "Principal Value" refers to the Cauchy principal value, which is a finite integral of a function about a point *c*, within the range of integration. It is the limit of the function shown below as epsilon approaches zero from above.

```
(%i13) kill(f)$
       limit('integrate(f(x), x, a, c - %epsilon)) + 'integrate(f(x), x, c + %epsilon, b);
```

(%o14) $\displaystyle\int_{c+\varepsilon}^{b} f(x)\,dx + \int_{a}^{c-\varepsilon} f(x)\,dx$

4.5 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 = π/2* about the *x*-axis. Using the shell method, the integral to be calculated is as below:

```
(%i15) A : x*cos(x)^-1;
       B: trigreduce(z);
       integrate(z, x, 0, %pi/2);
       integrate(x*cos(x)^-1, x, 0, %pi/2);
```

(%o15) $\frac{x}{\cos(x)}$

(%o16) z

(%o17) $\frac{\pi z}{2}$

(%o18) $\int_{0}^{\frac{\pi}{2}} \frac{x}{\cos(x)} dx$

This is a difficult integral that requires integration by parts. The disk method yields an integral that can be evaluated:

```
(%i19) [%pi*'integrate(cos(x)^2, x, 0, %pi/2), " = ",
                     %pi*integrate(cos(x)^2, x, 0, %pi/2)] ;
```

(%o19) $[\pi \int_{0}^{\frac{\pi}{2}} \cos(x)^2 dx , = , \frac{\pi^2}{4}]$

If you are working problems like this as exercise, you will need to recall double/half angle formulas. *Maxima* can help.

```
(%i20) halfangles: true$
       A: cos(x)^2;
       B: trigreduce(A);
       C: trigexpand(B);
       D: trigsimp(C);
```

(%o21) $\cos(x)^2$

(%o22) $\frac{\cos(2x)+1}{2}$

(%o23) $\frac{-\sin(x)^2+\cos(x)^2+1}{2}$

(%o24) $\cos(x)^2$

This dialogue shows that `trigreduce` tells *Maxima* to use the half-angle formulas to simplify an expression and `trigexpand` tells *Maxima* to use the double-angle formulas.

5 Some Numerical Methods in Maxima

In many cases we cannot find an analytical solution to a problem (one might not exist). *Maxima* offers a range of numerical methods for finding approximate values in such cases.

5.1 Solving Equations

Finding solutions for equations or systems of equations is an important application of numerical analysis. Consider the equation below, for which `solve` does not return a useful analytical solution.

```
(%i25) f(x, a, b) := x^a - b*log(x);
       [a0, b0] : [0.8, 5]$
       solve(f(x, a0, b0), x);
```

(%o25) $f(x,a,b):=x^a-b\log(x)$

```
rat: replaced 0.8 by 4/5 = 0.8
```

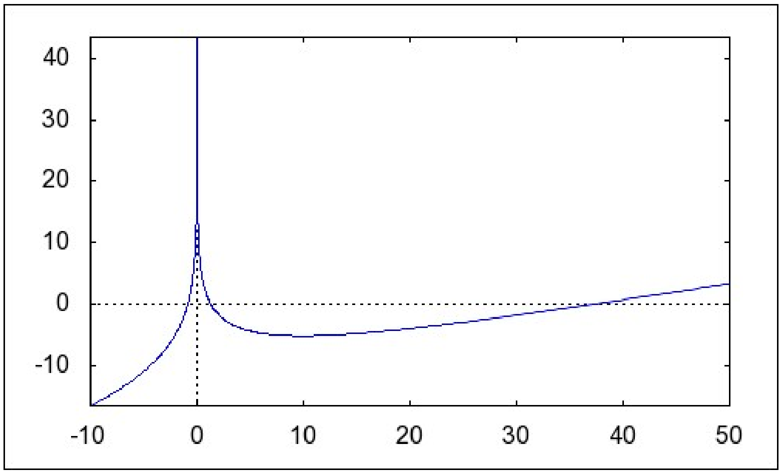
(%o27) $[\log(x)=\frac{x^{4/5}}{5}]$

When using numerical methods to find roots, one should graph the expression(s) to be solved. The graph can show whether multiple roots are to be expected over the relevant range. It can also guide the selection of guesses that serve as initial values for iterative search methods.

The graph below indicates that a roots occurs around x = 0 and for around x = 40. The function is not defined for x = 0. Why?

```
(%i28) wxdraw2d( xaxis = true, yaxis = true,
           explicit(f(x, a0, b0), x, -10, 50))$
```

(%t28)



The `find_root` command is used first. This command requires the expression to be evaluated, the expression's argument, and the interval endpoints. If the sign of *f(x)* is the same at both endpoints, an error message results.

```
(%i29) find_root(f(x, a0, b0), x, 1, 50);
```

```
find_root: function has same sign at endpoints: mequal(f(1.0),1.0),
mequal(f(50.0),3.305137569225593)
 -- an error. To debug this try: debugmode(true);
```

Shorten the interval to [.0001, 10]. (Why not [0, 10]?) The first root is x = 1.2750, approximately. The resulting *f(x, a0, b0)* is nearly zero. The discrepancy reflects error in the search process.

```
(%i30) find_root(f(x, a0, b0), x, 0.001, 10);
       f(%, a0, b0);
```

(%o30) 1.274936330738291

(%o31) $-2.220446049250313\ 10^{-16}$

The second root -- *x = 1.2749*, approximately -- is found the same way. Again, the resulting *f(x, a0, b0*) is approximately 0; 0.0 is a floating-point approximation of 0.

```
(%i32) find_root(f(x, a0, b0), x, 30, 50);
       f(%, a0, b0);
```

(%o32) 37.3312905308498

(%o33) 0.0

An alternative approach is to use the Newton search process. Doing so requires loading the module `newton1`. This is the module for applying the Newton process to a single equation. The `newton` command differs slightly from `find_root`. It requires the expression, the expression's argument, an initial guess of the value, and a tolerance level. The process works better if the guess is near to the root, and this is where the graph provides guidance.

```
(%i34) load(newton1)$
       newton(f(x, a0, b0), x, 1, .0000000001);
       newton(f(x, a0, b0), x, 40, .0000000001);
```

(%o35) 1.274936330738274

(%o36) 37.33129053084981

Newton's method can be extended to multiple equations. As long as the number of equations (and unknowns) is less than 4, graphing can still be used to guide the search process. Consider the two equations below. To avoid three dimensional graphs, we use the `implicit` command within `draw2d`. The graph shows *x, y* pairs for which *f(x, y) = 0* and for which *g(x, y) = 0*. Because *f(0, y)* is not defined, the implicit function is discontinuous at *x = 0*.

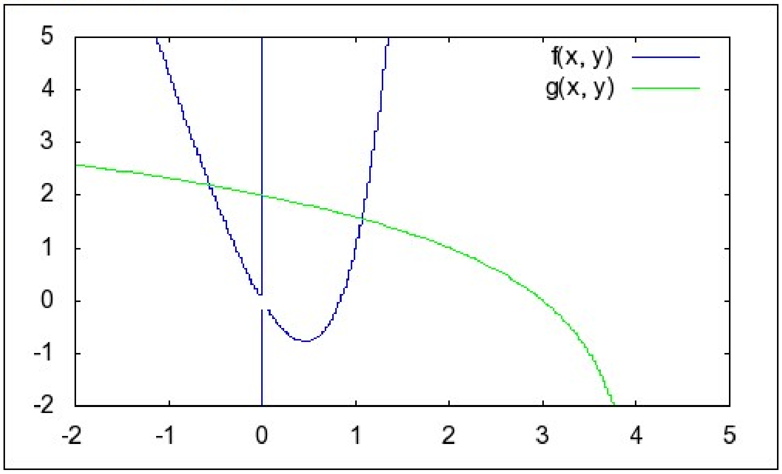
The graph shows that *f(x, y) = g(x, y) = 0* at two values of *x*: just over -1.0 , and just over +1.0.

```
(%i37) kill(all)$
       f(x,y) := 2*(3^x) - y/x - 5; g(x,y) := x + 2^y - 4;
       wxdraw2d(
       key = "f(x, y)", implicit(f(x,y), x, -2, 5, y, -2, 5),
       color = green, key = "g(x, y)",
       implicit(g(x,y), x, -2, 5, y, -2, 5) )$
```

(%o1) $f(x,y):=2\,3^{x}-\frac{y}{x}-5$

(%o2) $g(x,y):=x+2^{y}-4$

(%t3)



To determine the roots for multlple expressions, we load the `mnewton` module and enter the `mnewton` command. This command's arguments consist of three lists: a list of expressions, a list of variables, and a list of guesses. The tolerance level must be set explicitly if one wishes to use a value other than the default.

Because the graph identifies two *x, y* pairs that are roots for both equations, we execute `mnewton` twice.

```
(%i4) load(mnewton)$
      soln1: mnewton( [f(x, y), g(x, y)], [x, y], [-1, 2] );
      soln2: mnewton( [f(x, y), g(x, y)], [x, y], [1, 1.5] );
```

(%o5) $[[x=-0.55869137049845,y=2.1886197401254]]$

(%o6) $[[x=1.066618389595407,y=1.552564766841786]]$

We extract the critical values above and check the function values. The solutions are expressed as embedded lists, so extracting these values requires double indices (even though the top-level lists contains only one item).

```
(%i7) [x1, y1] : [rhs(soln1[1][1]), rhs( soln1[1][2]) ];
      [x2, y2]: [rhs(soln2[1][1]), rhs( soln2[1][2]) ];
      [ f(x1, y1), g(x1, y1) ];
      [f(x2, y2), g(x2, y2)] ;
```

(%o7) $[-0.55869137049845,2.1886197401254]$

(%o8) $[1.066618389595407,1.552564766841786]$

(%o9) $[0.0,8.881784197001252\,10^{-16}]$

(%o10) $[0.0,-4.440892098500626\,10^{-16}]$

5.2 Recursive Functions

It may be a good exercise to define your own *Maxima* function which implements Newton's method. Doing so shows a way to deal with recursive functions in *Maxima*.

```
(%i11) kill(all)$
       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))$
       g: lambda([x],x^2 -2);
       my_newt(g,1.5,.0001);
```

(%o2) $\mathrm{lambda}([x],x^{2}-2)$

(%o3) 1.41421568627451

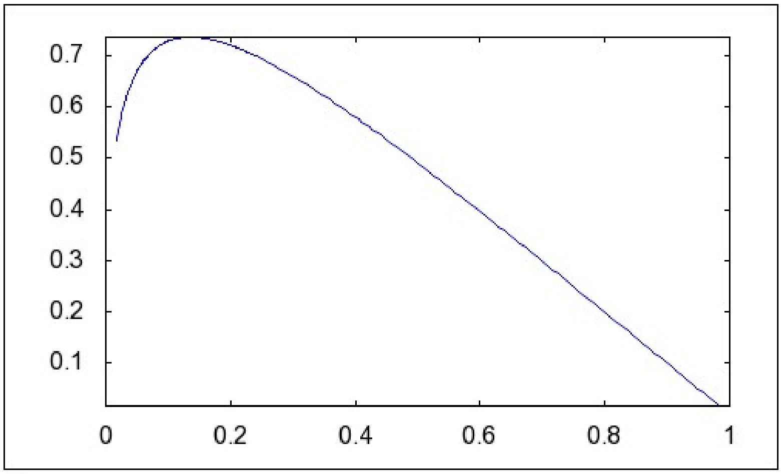
This definition of `my_newt` is recursive. That is, the definition of the function calls the function itself until the desired precision is attained. From the viewpoint of program clarity this is the method of choice. When calling `my_newt`, a lambda definition of function must be used because the name of a function f is being passed to `my_newt`.

5.3 Numerical Integration

Maxima provides a set of options for conducting numerical integration. To illustrate two of these, we use the function below. It can be integrated, so the numerical approach need not be taken, but we use it as a basis for comparison.

```
(%i4) kill(all)$ f(x) := x^(1/2)*log(1/x)$
      wxdraw2d( explicit(f(x), x, 0, 1) )$
      integrate(f(x), x, 0, 1);
      float(%);
```

(%t2)



```
      4
(%o3) -
      9
(%o4) 0.44444444444444
```

A relatively simple method to implement is that of Romberg. *Maxima*'s `romberg` command has the same syntax as the `integrate` command. For the function above, `romberg` cannot evaluate the integral over the entire range because it cannot deal with *f(0)*. We use *f(1e-100)* instead. The `romberg` command can handle values as small as `f(1e-320)` or so. For this function the `romberg` command yields a solution with errors after the fourth decimal place. Confirm that moving the beginning value closer to zero by using 1e-300 does not appreciably change the result.

```
(%i5) romberg(f(x), x, 1e-100 , 1 );
```

```
(%o5) 0.4444361409298
```

A more sophisticated method (set of methods, actually) is available in QUADPACK. We consider one example, using the command `quad_qag`. This command contains at least five arguments. (The *Maxima Manual* describes other, optional arguments.) The first four are the same as those of `integrate` and `romberg`. The fifth argument is a key that determines details of the search method and depends on the user's knowledge of the function's characteristics.

In this example `quad_qag` provides a closer approximation to the correct value than does `romberg`. The other three pieces of output are these: the estimated error of the method (quite small here), the number of iterations taken, and an error code (0 means no errors).

```
(%i6) quad_qag (f(x), x, 0, 1, 3);
```

```
(%o6) [0.44444444444921,3.1700968483768995 10^-9,961,0]
```

The Romberg technique can be applied to multidimensional integration, as in this example. The `assume(x > 0)` command is entered to avoid dealing with a dialog. Remove this line and determine the values once more.

```
(%i7) g(x, y) := x*y / (x + y);
      estimate : romberg (romberg (g(x, y), y, 0, x/2), x, 1, 3);
      assume(x > 0)$
      integrate (integrate (g(x, y), y, 0, x/2), x, 1, 3); float(%);
```

(%o7) $g(x,y):=\frac{x\,y}{x+y}$

(%o8) 0.81930228643245

(%o10) $-9\log\left(\frac{9}{2}\right)+9\log(3)+\frac{2\log\left(\frac{3}{2}\right)-1}{6}+\frac{9}{2}$

(%o11) 0.81930239639591

5.4 Numerical Solution of ODEs

Some ordinary differential equations (ODEs), or systems of ODEs, cannot be solved analytically. Numerical solutions of these systems can provide insights that are otherwise not attainable. A classic example is the Lorenz equations that were used to represent the convective motion of a fluid cell that is being warmed from below and cooled from above. This same set of equations can be applied to the analysis of dynamos and laser light. Numerious books analyze this system's behavior, most famously James Gleck's *Chaos*.

The equations that comprise this system are named `dxdt` (for *dx/dt*), `dydt`, and `dzdt`. The variable t is time. The parameters *a, b*, and *c* are all positive: *a* is called the Prandtl number, *b* is called the Rayleigh number, and *c* is a factor of proportionality.

The values of *x, y*, and *z* are not coordinates in physical space. Rather the value of *x* is proportional to the intensity of convective motion, and *y* is proportional to the temperature difference between ascending and descrend currents. Finally, *z* is propotional to the distortion of the vertical temperature profile from linearity. Of course, *x, y*, and *z* have quite different interpretations in other applications of this set of equations.

```
(%i12) kill(all)$ a: 10$ b: 28$ c: 2.667$
       dxdt : a*(y - x); dydt: x*(b - z) - y; dzdt : x*y - c*z;
```

(%o4) $10\,(y-x)$

(%o5) $x\,(28-z)-y$

(%o6) $x\,y-2.667\,z$

The cell below shows how *Maxima* uses the Runge-Kutte (`rk`) method to provide numerical solutions for the values of x, y, and z at points in time. The results of 1000 rk solutions are placed into a list named data. The `rk` command has these arguments: the name(s) of the expression(s), the name(s) or the variable(s), the initial value(s) or the variable(s), and the domain over which the solution is to occur. The domain itself has four elements: the name of the independent variable (time, here), that variable's initial value, its final value, and the increment. In the example below time ranges from *t = 0* to *t = 100*, with increments of *0.01*, so Maxima produces a list of 10,000 sets of values.

The first, second, 9999th, and 10000th lists of values appear as output. Each of these four items shows the time, the *x* value, the *y* value, and the *z* value.

```
(%i7) fpprintprec:5$
      data: rk([dxdt, dydt, dzdt],[x,y,z],[-15,20,-5],[t,0,100,0.01])$
      matrix( ["time", "x", "y", "z"], data[1], data[2], ["...", "...","...","..."],
      data[length(data) - 1], data[length(data)] );
```

(%o9) $$\begin{bmatrix} \textit{time} & x & y & z \\ 0 & -15 & 20 & -5 \\ 0.01 & -11.905 & 15.252 & -7.1975 \\ \ldots & \ldots & \ldots & \ldots \\ 99.99 & -12.141 & -17.165 & 25.244 \\ 100.0 & -12.625 & -17.245 & 26.685 \end{bmatrix}$$

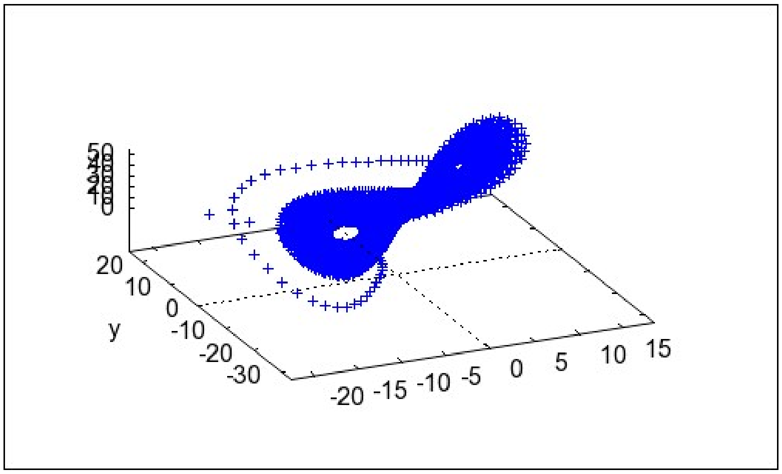
Below, we create a list of the 10,000 values of variables *x, y*, and *z* and show their interactions. The graph shows that the system does not tend toward any single point, rather it oscillates around two points. This points are called *strange attractors*. The *y* axis is labeled; the positions of the *x* and *z* labels can be deduced. The graph is produced with the `points` option.

Suggested exerecise: change one of the initial values in the `rk` command and trace the effects of that change. Make the change small. For example, let the new initial value of *x* be 15.1 or 14.9 rather than 15.

It can be instructive to remove the `wx` from the `draw` command and execute the command again. This produces a *gnuplot* window that lets you rotate the graph. This window must be closed before *wxMaxima* can proceed to the next command.

```
(%i10) uL: makelist([data[i][2],data[i][3],data[i][4]],i,1,length(data))$
       wxdraw3d(view = [36, 336], ylabel = "y", xaxis=true, yaxis=true, points(uL))$
```

(%t11)



Based on:
A Maxima Guide for Calculus Students
c January 2, 2007 by Moses Glasner

e-mail: glasner AT math DOT psu DOT edu
Address:Department of Mathematics
Pennsylvania State University
University Park, PA 16802

Created with wxMaxima.
Edited with Kompozer.
Converted to PDF with Winnovative HTML to PDF Converter.