Maxima by Example: Ch.6: Differential Calculus *

Edwin L. Woollett

October 21, 2010

Contents

*This version uses Maxima 5.21.1 This is a live document. Check `http://www.csulb.edu/~woollett/` for the latest version of these notes. Send comments and suggestions to `woollett@charter.net`

Preface

```
COPYING AND DISTRIBUTION POLICY
This document is part of a series of notes titled
"Maxima by Example" and is made available
via the author's webpage http://www.csulb.edu/~woollett/
to aid new users of the Maxima computer algebra system.

NON-PROFIT PRINTING AND DISTRIBUTION IS PERMITTED.

You may make copies of this document and distribute them
to others as long as you charge no more than the costs of printing.

These notes (with some modifications) will be published in book form
eventually via Lulu.com in an arrangement which will continue
to allow unlimited free download of the pdf files as well as the option
of ordering a low cost paperbound version of these notes.
```

Feedback from readers is the best way for this series of notes to become more helpful to new users of Maxima. *All* comments and suggestions for improvements will be appreciated and carefully considered.

```
LOADING FILES
The defaults allow you to use the brief version load(fft) to load in the
Maxima file fft.lisp.
To load in your own file, such as qxxx.mac
using the brief version load(qxxx), you either need to place
qxxx.mac in one of the folders Maxima searches by default, or
else put a line like:

file_search_maxima : append(["c:/work2/###.{mac,mc}"],file_search_maxima )$

in your personal startup file maxima-init.mac (see later in this chapter
for more information about this).

Otherwise you need to provide a complete path in double quotes,
as in load("c:/work2/qxxx.mac"),

We always use the brief load version in our examples, which are generated
using the XMaxima graphics interface on a Windows XP computer, and copied
into a fancy verbatim environment in a latex file which uses the fancyvrb
and color packages.
```

```
Maxima.sourceforge.net. Maxima, a Computer Algebra System. Version 5.21.1
 (2010). http://maxima.sourceforge.net/
```

The homemade function **fll(x)** (first, last, length) is used to return the first and last elements of lists (as well as the length), and is automatically loaded in with **mbe1util.mac** from Ch. 1. We will include a reference to this definition when working with lists.

This function has the definitions

```
fll(x) := [first(x),last(x),length(x)]$
declare(fll,evfun)$
```

Some of the examples used in these notes are from the Maxima html help manual or the Maxima mailing list: **http://maxima.sourceforge.net/maximalist.html**.

The author would like to thank the Maxima developers for their friendly help via the Maxima mailing list.

6 Differential Calculus

The methods of calculus lie at the heart of the physical sciences and engineering.

The student of calculus needs to take charge of his or her own understanding, taking the subject apart and putting it back together again in a way that makes logical sense. The best way to learn any subject is to work through a large collection of problems. The more problems you work on your own, the more you "own" the subject. Maxima can help you make faster progress, if you are just learning calculus.

For those who are already calculus savvy, the examples in this chapter will offer an opportunity to see some Maxima tools in the context of very simple examples, but you will likely be thinking about much harder problems you want to solve as you see these tools used here.

After examples of using **diff** and **gradef**, we present examples of finding the critical and inflection points of plane curves defined by an explicit function.

We then present examples of calculating and plotting the tangent and normal to a point on a curve, first for explicit functions and then for implicit functions.

We also have two examples of finding the maximum or minimum of a function of two variables.

We then present several examples of using Maxima's powerful **limit** function, followed by several examples of using **taylor**.

The next section shows examples of using the vector calculus functions (as well as cross product) in the package **vcalc.mac**, developed by the author and available on his webpage with this chapter, to calculate the gradient, divergence, curl, and Laplacian in cartesian, cylindrical, and spherical polar coordinate systems. An example of using this package would be `curl([r*cos(theta),0,0]);` (if the current coordinate system has already been changed to spherical polar) or `curl([r*cos(theta),0,0], s(r,theta,phi) );` if the coordinate system needs to be shifted to spherical polar from either cartesian (the starting default) or cylindrical.

The order of list vector components corresponds to the order of the arguments in `s(r,theta,phi)`. The Maxima output is the list of the vector curl components in the current coordinate system, in this case `[0, 0, sin(theta)]` plus a reminder to the user of what the current coordinate system is and what symbols are currently being used for the independent variables.

Thus the syntax is based on lists and is similar (although better!) than Mathematica's syntax.

There is a separate function to change the current coordinate system. To set the use of cylindrical coordinates `(rho,phi,z)`:

```
setcoord( cy(rho,phi,z) );
```

and to set cylindrical coordinates `(r,t,z)`:

```
setcoord( cy(r,t,z) );
```

The package **vcalc.mac** also contains the plotting function **plotderiv** which is useful for "automating" the plotting of a function and its first **n** derivatives.

The next two sections dicuss the use of the batch file mode of problem solving by presenting Maxima based derivations of the form of the gradient, divergence, curl, and Laplacian by starting with the cartesian forms and changing variables to (separately) cylindrical and spherical polar coordinates. (The batch files used are **cylinder.mac** and **sphere.mac**.) These two sections show Maxima's implementation of the calculus chain rule at work with use of both **depends** and **gradef**.

6.1 Differentiation of Explicit Functions

We begin with explicit functions of a single variable. After giving a few examples of the use of Maxima's **diff** function, we will discuss critical and inflection points of curves defined by explicit functions, and the construction and plotting of the tangent and normal of a point of such curves.

6.1.1 All About diff

The command **diff(expr,var,num)** will differentiate the expresssion in slot one with respect to the variable entered in slot two a number of times determined by a positive integer in slot three. Unless a dependency has been established, all parameters and "variables" in the expression are treated as constants when taking the derivative.

Thus **diff(expr,x,2)** will yield the second derivative of **expr** with respect to the variable **x**.

The simple form **diff(expr, var)** is equivalent to **diff(expr,var,1)**.

If the expression depends on more than one variable, we can use commands such as **diff(expr,x,2,y,1)** to find the result of taking the second derivative with respect to **x** (holding **y** fixed) followed by the first derivative with respect to **y** (holding **x** fixed).

Here are some simple examples.

We first calculate the derivative of x^n.

```
(%i1) diff(x^n,x);
                                      n - 1
(%o1)                               n x
```

Next we calculate the third derivative of x^n.

```
(%i2) diff(x^n,x,3);
                                                 n - 3
(%o2)                     (n - 2) (n - 1) n x
```

Here we take the derivative (with respect to x) of an expression depending on both x and y.

```
(%i3) diff(x^2 + y^2,x);
(%o3)                                 2 x
```

You can differentiate with respect to any expression that does not involve explicit mathematical operations.

```
(%i4) diff( x[1]^2 + x[2]^2, x[1] );
(%o4)                                2 x
                                       1
```

Note that x_1 is Maxima's way of "pretty printing" **x[1]**. We can use the **grind** function to display the output **%o4** in the "non-pretty print mode" (what would have been returned if we had set the **display2d** switch to **false**).

```
(%i5) grind(%)$
2*x[1]$
(%i6) display2d$
```

Note the dollar sign **grind** adds to the end of its output.

Finally, an example of using one invocation of **diff** to differentiate with respect to more than one variable:

```
(%i7) diff(x^2*y^3,x,1,y,2);
(%o7)                              12 x y
```

6.1.2 The Total Differential

If you use the **diff** function without a symbol in slot two, Maxima returns the "total differential" of the expression in slot one, and by default assumes every parameter is a variable.

If an expression contains a single parameter, say **x**, then **diff(expr)** will generate

$$\mathrm{d}\,\mathrm{f}(\mathrm{x}) = \left(\frac{\mathrm{d}\,\mathrm{f}(\mathrm{x})}{\mathrm{d}\,\mathrm{x}}\right)\,\mathrm{d}\,\mathrm{x} \tag{6.1}$$

In the first example below, we calculate the differential of the expression x^2, that is, the derivative of the expression (with respect to the variable x) multiplied by "the differential of the independent variable x", $\mathrm{d}\,\mathrm{x}$. Maxima uses **del(x)** for the differential of x, a small increment of x.

```
(%i1) diff(x^2);
(%o1)                              2 x del(x)
```

If the expression contains two parameters **x** and **y**, then the total differential is equivalent to the Maxima expression

diff(expr,x)*del(x) + diff(expr,y)*del(y) ,

which generates (using conventional notation):

$$\mathrm{d}\,\mathrm{f}(\mathrm{x},\mathrm{y}) = \left(\frac{\partial \mathrm{f}(\mathrm{x},\mathrm{y})}{\partial \mathrm{x}}\right)_{\mathrm{y}}\,\mathrm{d}\,\mathrm{x} + \left(\frac{\partial \mathrm{f}(\mathrm{x},\mathrm{y})}{\partial \mathrm{y}}\right)_{\mathrm{x}}\,\mathrm{d}\,\mathrm{y}. \tag{6.2}$$

Each additional parameter induces an additional term of the same form.

```
(%i2) diff(x^2*y^3);
                          2  2               3
(%o2)                   3 x  y  del(y) + 2 x y  del(x)
(%i3) diff(a*x^2*y^3);
                       2  2                 3           2  3
(%o3)          3 a x  y  del(y) + 2 a x y  del(x) + x  y  del(a)
```

We can use the **subst** function to replace, say **del(x)**, by anything else:

```
(%i4) subst(del(x) = dx, %o1);
(%o4)                                2 dx x
(%i5) subst([del(x)= dx, del(y) = dy, del(a) = da],%o3 );
                          2  3             3           2  2
(%o5)                  da x  y  + 2 a dx x y  + 3 a dy x  y
```

You can use the **declare** function to prevent some of the symbols from being treated as variables when calculating the total differential:

```
(%i6) declare (a, constant)$
(%i7) diff(a*x^2*y^3);
                          2  2                   3
(%o7)                  3 a x  y  del(y) + 2 a x y  del(x)
(%i8) declare([b,c],constant)$
(%i9) diff( a*x^3 + b*x^2 + c*x );
                                2
(%o9)                      (3 a x  + 2 b x + c) del(x)
(%i10) properties(a);
(%o10)                 [database info, kind(a, constant)]
(%i11) propvars(constant);
(%o11)                             [a, b, c]
```

```
(%i12) kill(a,b,c)$
(%i13) propvars(constant);
(%o13)                                []
(%i14) diff(a*x);
(%o14)                        a del(x) + x del(a)
```

We can "map" **diff** on to a list of functions of **x**, say, and divide by **del(x)**, to generate a list of derivatives, as in

```
(%i15) map('diff,[sin(x),cos(x),tan(x) ] )/del(x);
                                                        2
(%o15)                      [cos(x), - sin(x), sec (x)]
(%i16) map('diff,%)/del(x);
                                                    2
(%o16)               [- sin(x), - cos(x), 2 sec (x) tan(x)]
```

6.1.3 Controlling the Form of a Derivative with gradef

We can use **gradef** to select one from among a number of alternative ways of writing the result of differentiation. The large number of "trigonometric identities" means that any given expression containing trig functions can be written in terms of a different set of trig functions.

As an example, consider trig identities which express $\mathbf{sin}(\mathbf{x})$, $\mathbf{cos}(\mathbf{x})$, and $\mathbf{sec^2}(\mathbf{x})$ in terms of $\mathbf{tan}(\mathbf{x})$ and $\mathbf{tan}(\mathbf{x/2})$:

$$\mathbf{sec}^2(x) = 1 + \mathbf{tan}^2(x), \tag{6.3}$$

$$\mathbf{sin}\, x = 2\,\mathbf{tan}(x/2)/(1 + \mathbf{tan}^2(x/2)), \tag{6.4}$$

$$\mathbf{cos}\, x = (1 - \mathbf{tan}^2(x/2))/(1 + \mathbf{tan}^2(x/2)). \tag{6.5}$$

The default Maxima result for the first derivatives of $\mathbf{sin}\, x$, $\mathbf{cos}\, x$, and $\mathbf{tan}\, x$ is:

```
(%i1) map('diff,[sin(x),cos(x),tan(x)] )/del(x);
                                                    2
(%o1)                     [cos(x), - sin(x), sec (x)]
```

Before using **gradef** to alter the return value of **diff** for these three functions, let's check that Maxima agrees with the trig "identities" displayed above. Variable amounts of expression simplification are needed to get what we want.

```
(%i2) trigsimp( 1 + tan(x)^2 );
                                        1
(%o2)                                -------
                                        2
                                     cos (x)
(%i3) ( trigsimp( 2*tan(x/2)/(1 + tan(x/2)^2) ), trigreduce(%%) );
(%o3)                                sin(x)
(%i4) ( trigsimp( (1-tan(x/2)^2)/(1 + tan(x/2)^2) ),
          trigreduce(%%), expand(%%) );
(%o4)                                cos(x)
```

Recall that **trigsimp** will convert **tan**, **sec**, etc to **sin** and **cos** of the same argument. Recall also that **trigreduce** will convert an expression containing **cos(x/2)** and **sin(x/2)** into an expression containing **cos(x)** and **sin(x)**. Finally, remembering that $\mathbf{sec}\, x = 1/\mathbf{cos}\, x$, we see that Maxima has "passed the test".

Now let's use **gradef** to express the derivatives in terms of $\tan x$ and $\tan(x/2)$:

```
(%i5) gradef(tan(x), 1 + tan(x)^2 );
(%o5)                                tan(x)
(%i6) gradef(sin(x),(1-tan(x/2)^2 )/(1 + tan(x/2)^2 ) );
(%o6)                                sin(x)
(%i7) gradef(cos(x), -2*tan(x/2)/(1 + tan(x/2)^2 ) );
(%o7)                                cos(x)
```

Here we check the behavior:

```
(%i8) map('diff,[sin(x),cos(x),tan(x)] )/del(x);
                           2 x             x
                     1 - tan (-)     2 tan(-)
                              2             2         2
(%o8)              [-----------, - -----------, tan (x) + 1]
                         2 x             2 x
                     tan (-) + 1     tan (-) + 1
                          2               2
```

6.2 Critical and Inflection Points of a Curve Defined by an Explicit Function

The **critical points** of a function $\mathbf{f(x)}$ are the points $\mathbf{x_j}$ such that $\mathbf{f'(x_j) = 0}$. We will use $\mathbf{f'}$ to indicate the first derivative, and $\mathbf{f''}$ to indicate the second derivative. The **extrema** (ie., maxima and minima) are the values of the function at the critical points, provided the "slope" $\mathbf{f'}$ actually has a different sign on the opposite sides of the critical point.

Maximum: $\mathbf{f'(a) = 0}$,	$\mathbf{f'(x)}$	changes from $+$ to $-$,	$\mathbf{f''(a) < 0}$
Minimum: $\mathbf{f'(a) = 0}$,	$\mathbf{f'(x)}$	changes from $-$ to $+$,	$\mathbf{f''(a) > 0}$

If $\mathbf{f(x)}$ is a function with a continuous second derivative, and if, as $\mathbf{x}$ increases through the value $\mathbf{a}$, $\mathbf{f''(x)}$ changes sign, then the plot of $\mathbf{f(x)}$ has an **inflection point** at $\mathbf{x = a}$ and $\mathbf{f''(a) = 0}$. The **inflection point** requirement that $\mathbf{f''(x)}$ changes sign at $\mathbf{x = a}$ is equivalent to $\mathbf{f'''(a) \neq 0}$. We consider some simple examples taken from Sect. 126 of Analytic Geometry and Calculus, by Lloyd L. Smail, Appleton-Century-Crofts, N.Y., 1953 (a reference which "dates" the writer of these notes!).

6.2.1 Example 1: A Polynomial

To find the maxima and minima of the function $\mathbf{f(x) = 2\,x^3 - 3\,x^2 - 12\,x + 13}$, we use an "expression" (called **g**) rather than a Maxima function. We use **gp** (g prime) for the first derivative, and **gpp** (g double prime) as notation for the second derivative. Since we will want to make some simple plots, we use the package **qdraw**, available on the author's webpage, and discussed in detail in chapter five of these notes.

```
(%i1) load(draw)$
(%i2) load(qdraw);
              qdraw(...), qdensity(...), syntax: type qdraw();
(%o2)                          c:/work2/qdraw.mac
(%i3) g : 2*x^3 - 3*x^2 - 12*x + 13$
(%i4) gf : factor(g);
                                        2
(%o4)                        (x - 1) (2 x  - x - 13)
(%i5) gp : diff(g,x);
                                    2
(%o5)                             6 x  - 6 x - 12
(%i6) gpf : factor(gp);
(%o6)                           6 (x - 2) (x + 1)
(%i7) gpp : diff(gp,x);
(%o7)                               12 x - 6
(%i8) qdraw( ex(g,x,-4,4), key(bottom)  )$
```

Since the coefficients of the given polynomial are integral, we have tried out **factor** on both the given expression and its first derivative. We see from output **%o6** that the first derivative vanishes when **x = -1, 2** . We can confirm these critical points of the given function by using **qdraw** with the quick plotting argument **ex**, which gives us the plot:

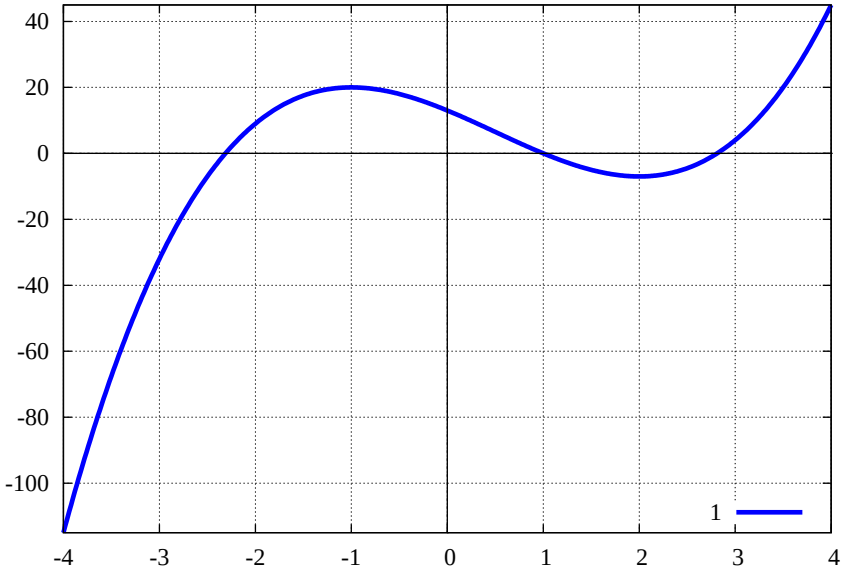

Figure 1: Plot of g over the range $[-4, 4]$

We can use the cursor on the plot to find that **g** takes on the value of roughly **-7** when **x = 2** and the value of about **20** when **x = -1**. We can use **solve** to find the roots of the equation **gp = 0**, and then evaluate the given expression **g**, as well as the second derivative **gpp** at the critical points found:

```
(%i9) solve(gp=0);
(%o9)                          [x = 2, x = - 1]
(%i10) [g,gpp],x=-1;
(%o10)                             [20, - 18]
(%i11) [g,gpp],x=2;
(%o11)                              [- 7, 18]
```

We next look for the **inflection points**, the points where the curvature changes sign, which is equivalent to the points where the second derivative vanishes.

```
(%i12) solve(gpp=0);
                                        1
(%o12)                             [x = -]
                                        2
(%i13) g,x=0.5;
(%o13)                               6.5
```

We have found one inflection point, that is a point on a smooth curve where the curve changes from concave downward to concave upward, or visa versa (in this case the former).
We next plot **g, gp, gpp** together.

```
(%i14) qdraw2( ex([g,gp,gpp],x,-3,3), key(bottom),
                pts([[-1,20]],ps(2),pc(magenta),pk("MAX") ),
             pts([[2,-7]],ps(2),pc(green),pk("MIN") ),
           pts([[0.5, 6.5]],ps(2),pk("INFLECTION POINT") ) )$
```

with the resulting plot:

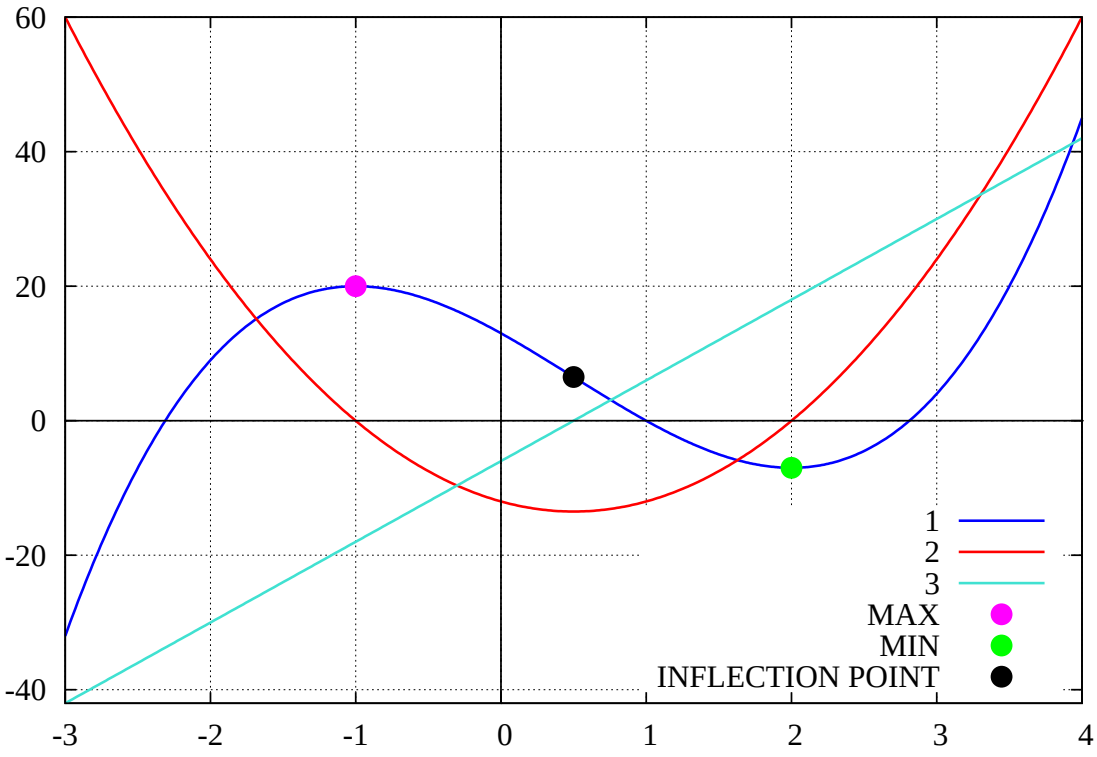

Figure 2: Plot of 1: g, 2: gp, and 3: gpp

The curve **g** is concave downward until **x = 0.5** and then is concave upward. Note that each successive differentiation tends to flatten out the kinks in the previous expression (function). In other words, **gp** is "flatter" than **g**, and **gpp** is "flatter" than **gp**. This behavior under **diff** is simply because each successive differentiation reduces by one the degree of the polynomial.

6.2.2 Automating Derivative Plots with plotderiv

In a separate text file **vcalc.mac** (available on the author's webpage) is a small homemade Maxima function called **plotderiv** which will plot a given expression together with as many derivatives as you want, using **qdraw** from Ch.5.

```
/* vcalc.mac */
plotderiv_syntax() :=
  disp("plotderiv(expr,x,x1,x2,y1,y2,numderiv)  constructs a list of
       the submitted expression expr and its first numderiv derivatives
       with respect to the independent variable, and then passes
       this list to qdraw(..). You need to have used load(draw)
       and load(qdraw) before using this function ")$

/*  version 1 commented out
plotderiv(expr,x,x1,x2,y1,y2,numderiv) :=
   block([plist],
     plist : [],
     for i thru numderiv do
        plist : cons(diff(expr,x,i), plist),
     plist : reverse(plist),
     plist : cons(expr, plist),
     display(plist),
     apply( 'qdraw, [ ex( plist,x,x1,x2 ),yr(y1,y2), key(bottom) ]))$ */
```

```
/* version 2  slightly more efficient  */
plotderiv(expr,x,x1,x2,y1,y2,numderiv) :=
   block([plist,aa],
     plist : [],
     aa[0] : expr,
     for i thru numderiv do (
        aa[i] : diff(aa[i-1],x),
        plist : cons(aa[i], plist)
         ),
     plist : reverse(plist),
     plist : cons(expr, plist),
     display(plist),
     apply( 'qdraw, [ ex( plist,x,x1,x2 ),yr(y1,y2), key(bottom) ]))$
```

We have provided two versions of this function, the first version commented out. If the expression to be plotted is a very complicated function and you are concerned with computing time, you can somewhat improve the efficiency of **plotderiv** by introducing a "hashed array" called **aa[j]**, say, to be able to simply differentiate the previous derivative expression. That is the point of version 2 which is not commented out.

We have added the line **display(plist)** to print a list containing the expression as well as all the derivatives requested. We test **plotderiv** on the expression **u^3**. Both **draw.lisp** and **qdraw.mac** must be loaded for this to work.

```
(%i15) load(vcalc)$
                vcalc.mac:   for syntax, type: vcalc_syntax();
 CAUTION: global variables set and used in this package:
       hhh1, hhh2, hhh3, uuu1, uuu2, uuu3, nnnsys
(%i16) plotderiv(u^3,u,-3,3,-27,27,4)$
                                    3     2
                          plist = [u , 3 u , 6 u, 6, 0]
```

which produces the plot:

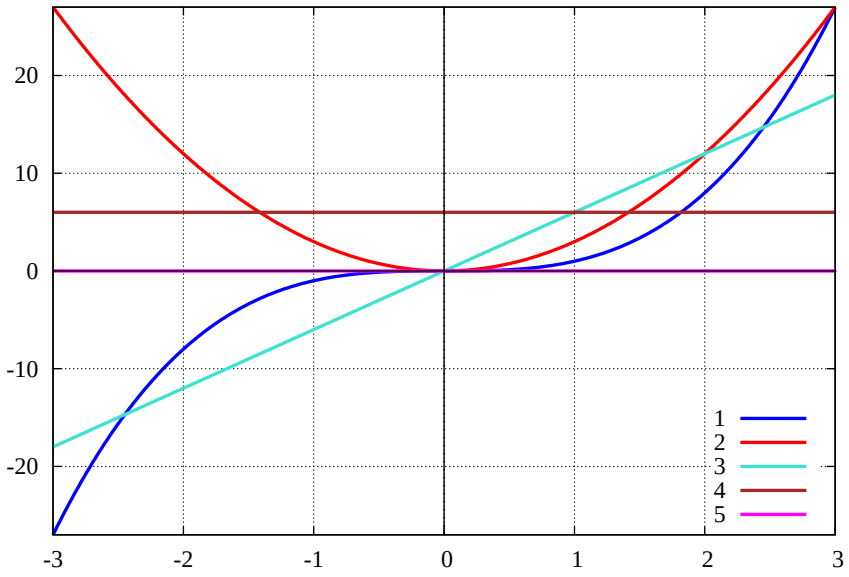

Figure 3: 1: $\mathbf{u^3}$, 2: $\mathbf{3\,u^2}$, 3: $\mathbf{6\,u}$, 4: $\mathbf{6}$, 5: $\mathbf{0}$

6.2.3 Example 2: Another Polynomial

We find the critical points, inflection points, and extrema of the expression $3\,x^4 - 4\,x^3$. We first look at the expression and its first two derivatives using **plotderiv**.

```
(%i17) plotderiv(3*x^4 - 4*x^3,x,-1,3,-5,5,2)$
                          4      3      3       2      2
             plist = [3 x  - 4 x , 12 x  - 12 x , 36 x  - 24 x]
```

which produces the plot:

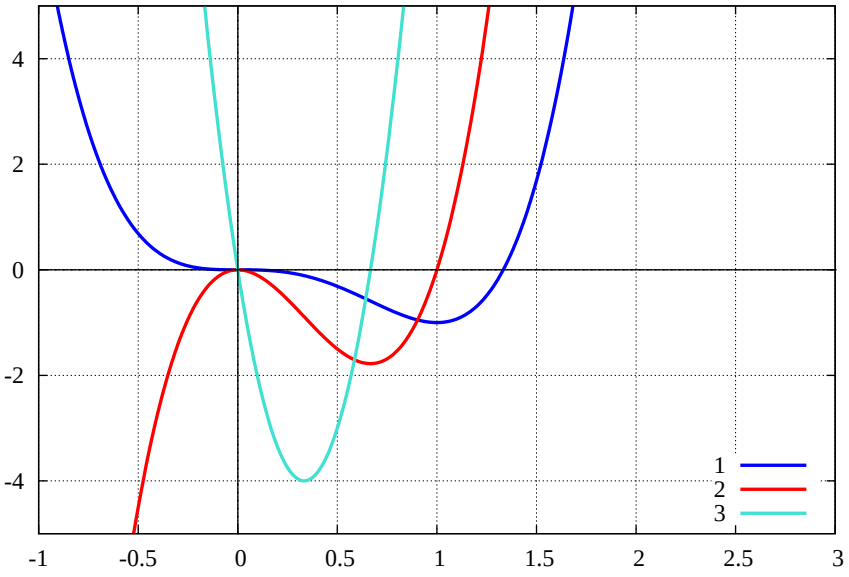

Figure 4: 1: $f = 3\,x^4 - 4\,x^3$, 2: f', 3: f''

We have inflection points at $x = 0,\ 2/3$ since the second derivative is zero at both points and changes sign as we follow the curve through those points. The curve changes from concave up to concave down passing through $x = 0$, and changes from concave down to concave up passing through $x = 2/3$. The following provides confirmation of our inferences from the plot:

```
(%i18) g:3*x^4 - 4*x^3$
(%i19) g1: diff(g,x)$
(%i20) g2 : diff(g,x,2)$
(%i21) g3 : diff(g,x,3)$
(%i22) x1 : solve(g1=0);
(%o22)                          [x = 0, x = 1]
(%i23) gcrit : makelist(subst(x1[i],g),i,1,2);
(%o23)                            [0, - 1]
(%i24) x2 : solve(g2=0);
                                   2
(%o24)                          [x = -, x = 0]
                                   3
(%i25) ginflec : makelist( subst(x2[i],g ),i,1,2 );
                                    16
(%o25)                            [- --, 0]
                                    27
(%i26) g3inflec : makelist( subst(x2[i],g3),i,1,2 );
(%o26)                           [24, - 24]
```

We have critical points where the first derivative vanishes at $x = 0,\ 1$. Since the first derivative does not change sign as the curve passes through the point $x = 0$, that point is neither a maximum nor a mimimum point. The point $x = 1$ is a minimum point since the first derivative changes sign from negative to positive as the curve passes through that point.

6.2.4 Example 3: $x^{2/3}$, Singular Derivative, Use of limit

Searching for points where a function has a minimum or maximum by looking at points where the first derivative is zero is useful as long as the first derivative is well behaved. Here is an example in which the given function of x has a minimum at $x = 0$, but the first derivative is singular at that point. Using the expression **g = x^(2/3)** with **plotderiv**:

```
(%i27) plotderiv(x^(2/3),x,-2,2,-2,2,1)$
```

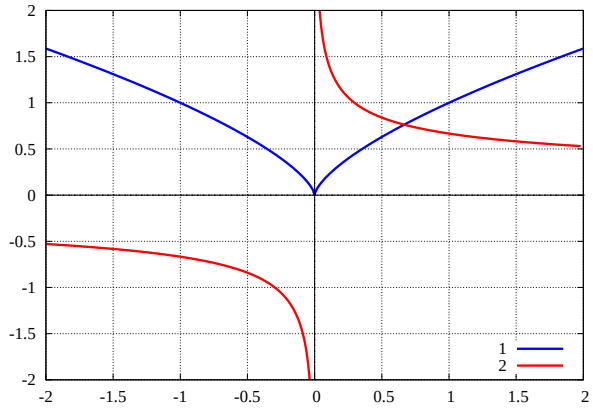

Figure 5: Plot of $\mathbf{1}:\ f(x) = x^{2/3},\ \mathbf{2}:\ f'$

We see that the first derivative is singular at $x = 0$ and approaches either positive or negative infinity, depending on your direction of approaching the point $x = 0$. We can see from the plot of $f(x) = x^{2/3}$ that the tangent line (the "slope") of the curve is negative for $x < 0$ and becomes increasingly negative (approaching minus infinity) as we approach the point $x = 0$ from the negative side. We also see from the plot of $f(x)$ that the tangent line ("slope") is positive for positive values of x and as we pass $x = 0$, moving from smaller x to larger x that the sign of the first derivative $f'(x)$ changes from $-$ to $+$, which is a signal of a local minimum of a function.

We can practice using the Maxima function **limit** with this example:

```
(%i28) gp : diff(x^(2/3),x);
                                      2
(%o28)                               ------
                                       1/3
                                     3 x
(%i29) limit(gp,x,0,plus);
(%o29)                                inf
(%i30) limit(gp,x,0,minus);
(%o30)                               minf
(%i31) limit(gp,x,0);
(%o31)                                und
```

The most frequent use of the Maxima **limit** function has the syntax

> Function: **limit (expr, x, val, dir)**
> Function: **limit (expr, x, val)**
> The first form computes the limit of **expr** as the real variable **x** approaches the value **val** from the direction **dir**. **dir** may have the value **plus** for a limit from above, **minus** for a limit from below.
> In the second form, **dir** is omitted, implying a "two-sided limit" is to be computed.
> **limit** uses the following special symbols: **inf** (positive infinity) and **minf** (negative infinity). On output it may also use **und** (undefined), **ind** (indefinite but bounded) and **infinity** (complex infinity).

Returning to our example, if we ignore the point $x = 0$, the slope is always decreasing, so the second derivative is always negative.

6.3 Tangent and Normal of a Point of a Curve Defined by an Explicit Function

The equation of a line with the form $\mathbf{y = m\,(x - x_0) + y_0}$, where $\mathbf{m}$, $\mathbf{x_0}$, and $\mathbf{y_0}$ are constants, is identically satisfied if we consider the point $\mathbf{(x, y) = (x_0, y_0)}$. Hence this line passes through the point $\mathbf{(x_0, y_0)}$. The "slope" of this line (the first derivative) is the constant $\mathbf{m}$.

Now we also assume that the point $\mathbf{(x_0, y_0)}$ is a point of a curve given by some explicit function of $\mathbf{x}$, say $\mathbf{y = f(x)}$; hence $\mathbf{y_0 = f(x_0)}$. The first derivative of this function, evaluated at the point $\mathbf{x_0}$ is the local "slope", which defines the **local tangent line**, $\mathbf{y = m\,(x - x_0) + y_0}$, if we use for $\mathbf{m}$ the value of $\mathbf{f'(x_0)}$, where the notation means first take the derivative for arbitrary $\mathbf{x}$ and then evaluate the derivative at $\mathbf{x = x_0}$.

Let the two dimensional vector **tvec** be a vector parallel to the tangent line (at the point of interest) with components **(tx,ty)**, such that the vector makes an angle θ with the positive $\mathbf{x}$ axis. Then $\mathbf{\tan(\theta) = t_y/t_x = m = d\,y/d\,x}$ is the slope of the tangent (line) at this point. Let the two dimensional vector **nvec** with components **(nx,ny)** be parallel to the line **perpendicular** to the tangent at the given point. This **normal** line will be perpendicular to the tangent line if the vectors **nvec** and **tvec** are perpendicular (i.e., "orthogonal").

In Maxima, we can represent vectors by lists:

```
(%i1) tvec : [tx,ty];
(%o1)                                [tx, ty]
(%i2) nvec : [nx, ny];
(%o2)                                [nx, ny]
```

Two vectors are "orthogonal" if the "dot product" is zero. A simple example will illustrate this general property. Consider the pair of unit vectors: **ivec = [1,0]**, parallel to the positive $\mathbf{x}$ axis and **jvec = [0,1]**, parallel to the positive $\mathbf{y}$ axis.

```
(%i3) ivec : [1,0]$
(%i4) jvec : [0,1]$
(%i5) ivec . jvec;
(%o5)                                  0
```

Since Maxima allows us to use a **period** to find the dot product (inner product) of two lists (two vectors), we will ensure that the normal vector is "at right angles" to the tangent vector by requiring **nvec . tvec = 0**

```
(%i6) eqn : nvec . tvec = 0;
(%o6)                          ny ty + nx tx = 0
(%i7) eqn : ( eqn/(nx*ty), expand(%%) );
                                   tx   ny
(%o7)                              -- + -- = 0
                                   ty   nx
```

We represent the normal (the line perpendicular to the tangent line) at the point $\mathbf{(x_0, y_0)}$ as $\mathbf{y = m_n\,(x - x_0) + y_0}$, where $\mathbf{m_n}$ is the slope of the normal line: **mn = ny/nx = - (1/(ty/tx)) = -1/m** .

In words, **the slope of the normal line is the negative reciprocal of the slope of the tangent line**.

Thus the equation of the **local normal** to the curve at the point $\mathbf{(x_0, y_0)}$ can be written as $\mathbf{y = -(x - x_0)/m + y_0}$, where $\mathbf{m}$ is the slope of the local tangent.

6.3.1 Example 1: x^2

As an easy first example, let's use the function $f(x) = x^2$ and construct the tangent line and normal line to this plane curve at the point `x = 1, y = f(1) = 1`. (We have discussed this same example in Ch. 5). The first derivative is $2x$, and its value at $x = 1$ is 2. Thus the equation of the local tangent to the curve at `(x,y) = (1,1)` is $y = 2x - 1$. The equation of the local normal at the same point is $y = -x/2 + 3/2$.

We will plot the curve $y = x^2$, the local tangent, and the local normal together. Special care is needed to get the horizontal "canvas" width about `1.4` times the vertical height to achieve a geometrically correct plot; otherwise the "normal" line would not cross the tangent line at right angles.

```
(%i8) qdraw( ex([x^2,2*x-1,-x/2+3/2],x,-1.4,2.8), yr(-1,2) ,
             pts( [ [1,1]],ps(2) ) )$
```

The plot looks like:

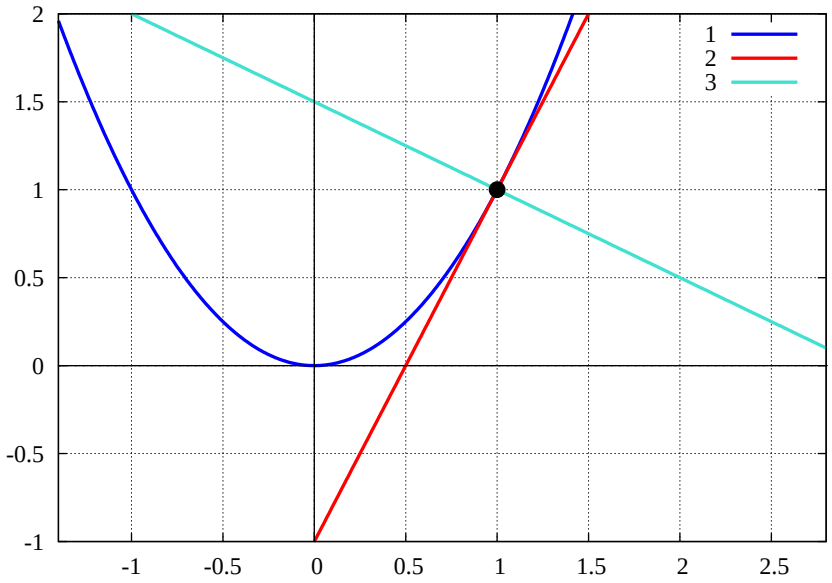

Figure 6: Tangent and Normal to x^2 at point $(1, 1)$

We can see directly from the plot that the slope of the tangent line has the value 2. Remember that we can think of "slope" as being the number derived from dividing the "rise" by the "run". Thus if we choose a "run" to be the x interval `(0,1)`, from the plot this corresponds to the "rise" of $1 - (-1) = 2$. Hence "rise/run" is 2.

6.3.2 Example 2: $\ln(x)$

Our next example repeats the analysis of Example 1 for the function $\ln(x)$, using the point $(x = 2, y = \ln(2))$. Maxima's natural log function is written `log`. Maxima does not have a "log to the base 10 function", although you can "roll your own" with a definition like `log10(x) := log(x)/log(10)`, which is equivalent to `log(x)/2.303`. Thus in Maxima, `log(10) = 2.303, log(2) = 0.693`, etc. We are constructing the local tangent and normal at the point $(x = 2, y = 0.693)$. The derivative of the natural log is

```
(%i9) diff(log(x),x);
                                      1
(%o9)                                 -
                                      x
```

so the "slope" of our curve (and the tangent) at the chosen point is `1/2`, and the slope of the local normal is `-2`. The equation of the tangent is then $(y - \ln(2)) = (1/2)(x - 2)$, or $y = x/2 - 0.307$. The equation of the normal

is $(\mathbf{y} - \mathbf{ln(2)}) = \mathbf{-2}\,(\mathbf{x} - \mathbf{2})$, or $\mathbf{y} = \mathbf{-2\,x + 4.693}$. In order to get a decent plot, we need to stay away from the singular point $\mathbf{x} = \mathbf{0}$, so we start the plot with $\mathbf{x}$ a small positive number. We choose to use the x-axis range $(\mathbf{0.1}, \mathbf{4})$, then $\mathbf{\Delta x} = \mathbf{3.9}$. Since we want the "canvas width" $\mathbf{\Delta x} \approx \mathbf{1.4\,\Delta y}$, we need $\mathbf{\Delta y} = \mathbf{2.786}$, which is satisfied if we choose the y-axis range to be $(\mathbf{-1}, \mathbf{1.786})$.

```
(%i10) qdraw(key(bottom), yr(-1,1.786),
          ex([log(x), x/2 - 0.307, -2*x + 4.693],x,0.1,4),
          pts( [ [2,0.693]],ps(2) )  );
```

which produces the plot:

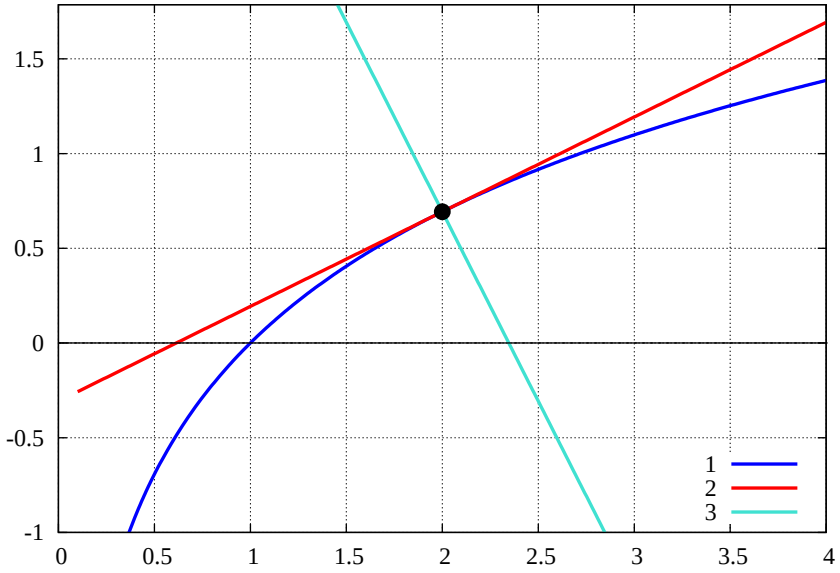

Figure 7: Tangent and Normal to $\mathbf{ln(x)}$ at point $(\mathbf{2}, \mathbf{ln(2)})$

6.4 Maxima and Minima of a Function of Two Variables

You can find proofs of the following criteria in calculus texts.

If $\mathbf{f(x, y)}$ and its first derivatives are continuous in a region including the point $(\mathbf{a}, \mathbf{b})$, a necessary condition that $\mathbf{f(a, b)}$ shall be an extreme (maximum or minimum) value of the function $\mathbf{f(x, y)}$ is that (I):

$$\frac{\partial f(a,b)}{\partial x} = 0, \quad \frac{\partial f(a,b)}{\partial y} = 0 \tag{6.6}$$

in which the notation means first take the partial derivatives for arbitrary $\mathbf{x}$ and $\mathbf{y}$ and then evaluate the resulting derivatives at the point $(\mathbf{x}, \mathbf{y}) = (\mathbf{a}, \mathbf{b})$.

Examples show that these two conditions (I) do not guarantee that the function actually takes on an extreme value at the point $(\mathbf{a}, \mathbf{b})$, although in many practical problems the existence and nature of an extreme value is often evident from the problem itself and no extra test is needed; all that is needed is the **location** of the extreme value.

If condition (I) is satisfied and if, in addition, at the point $(\mathbf{a}, \mathbf{b})$ we have (II):

$$\Delta = \left(\frac{\partial^2 f}{\partial x^2}\right)\left(\frac{\partial^2 f}{\partial y^2}\right) - \left(\frac{\partial^2 f}{\partial x\,\partial y}\right)^2 > 0 \tag{6.7}$$

then $\mathbf{f(x, y)}$ will have a maximum value or a minimum value given by $\mathbf{f(a, b)}$ according as $\partial^2 f/\partial x^2$ (or $\partial^2 f/\partial y^2$) is negative or positive for $\mathbf{x} = \mathbf{a}, \mathbf{y} = \mathbf{b}$. If condition (I) holds and $\mathbf{\Delta} < \mathbf{0}$ then $\mathbf{f(a, b)}$ is neither a maximum nor a minimum; if $\mathbf{\Delta} = \mathbf{0}$ the test fails to give any information.

6.4.1 Example 1: Minimize the Area of a Rectangular Box of Fixed Volume

Let the sides of a "rectangular box" (rectangular prism, or cuboid, if you wish) of fixed volume $\mathbf{v}$ be $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$. We wish to determine the shape of this box (for fixed $\mathbf{v}$) which minimizes the surface area $\mathbf{s} = \mathbf{2}\,(\mathbf{x}\,\mathbf{y} + \mathbf{y}\,\mathbf{z} + \mathbf{z}\,\mathbf{x})$. We can use the fixed volume relation to express $\mathbf{z}$ as a function of $\mathbf{x}$ and $\mathbf{y}$ and then achieve an expression for the surface area which only depends of the two variables $\mathbf{x}$ and $\mathbf{y}$. We then know that a necessary condition that the surface area be an extremum is that the two equations of condition I above be satisfied. We need to expose the hidden dependence of $\mathbf{z}$ on $\mathbf{x}$ and $\mathbf{y}$ via the volume constraint before we can correctly derive the two equations of condition I.

You probably already know the answer to this shape problem is a cube, in which all sides are equal in length, and the length of a side is the same as the cube root of the volume.

We require three equations to be satisfied, the first being the fixed volume $\mathbf{v} = \mathbf{x}\,\mathbf{y}\,\mathbf{z}$, and the other two being the equations of condition I above. There are multiple paths to such a solution in Maxima. Perhaps the simplest is to use **`solve`** to generate a solution of the three equations for the variables $(\mathbf{x}, \mathbf{y}, \mathbf{z})$, although the solutions returned will include non-physical solutions and we must select the physically realizable solution.

```
(%i1) eq1 : v = x*y*z;
(%o1)                                v = x y z
(%i2) solz : solve(eq1,z);
                                        v
(%o2)                                [z = ---]
                                        x y
(%i3) s : ( subst(solz,2*(x*y  + y*z + z*x) ), expand(%%) );
                                      2 v   2 v
(%o3)                           2 x y + --- + ---
                                       y     x
(%i4) eq2 : diff(s,x)=0;
                                      2 v
(%o4)                             2 y - --- = 0
                                       2
                                      x
(%i5) eq3 : diff(s,y) = 0;
                                      2 v
(%o5)                             2 x - --- = 0
                                       2
                                      y
(%i6) solxyz: solve([eq1,eq2,eq3],[x,y,z]);
             1/3       1/3       1/3
(%o6) [[x = v   , y = v   , z = v   ],
            1/3                                  1/3
         2 v                (sqrt(3) %i + 1) v
[x = --------------, y = - ----------------------,
     sqrt(3) %i - 1                   2
                         1/3                  1/3
      (sqrt(3) %i + 1) v                 2 v
z = - ----------------------], [x = - --------------,
                 2                   sqrt(3) %i + 1
                        1/3                          1/3
    (sqrt(3) %i - 1) v          (sqrt(3) %i - 1) v
y = --------------------, z = ----------------------]]
              2                             2
```

Since the physical solutions must be real, the first sublist is the physical solution which implies the cubical shape as the answer.

An alternative solution path is to solve **eq2** above for $\mathbf{y}$ as a function of $\mathbf{x}$ and $\mathbf{v}$ and use this to eliminate $\mathbf{y}$ from **eqn3**.

```
(%i7) soly : solve(eq2,y);
                                        v
(%o7)                              [y = --]
                                        2
                                       x
(%i8) eq3x : ( subst(soly, eq3),factor(%%) );
                                     3
                                  2 x (x  - v)
(%o8)                           - ------------ = 0
                                        v
```

The factored form **%o7** shows that, since $\mathbf{x} \neq \mathbf{0}$, the solution must have $\mathbf{x} = \mathbf{v^{1/3}}$. We can then use this solution for $\mathbf{x}$ to generate the solution for $\mathbf{y}$ and then for $\mathbf{z}$.

```
(%i9) solx : x = v^(1/3);
                                       1/3
(%o9)                             x = v
(%i10) soly : subst(solx, soly);
                                       1/3
(%o10)                            [y = v   ]
(%i11) subst( [solx,soly[1] ], solz );
                                       1/3
(%o11)                           [z = v   ]
```

We find that $\mathbf{\Delta > 0}$ and that the second derivative of $\mathbf{s(x, y)}$ with respect to $\mathbf{x}$ is positive, as needed for the cubical solution to define a minimum surface solution.

```
(%i12) delta : ( diff(s,x,2)*diff(s,y,2) - diff(s,x,1,y,1), expand(%%) );
                                    2
                                  16 v
(%o12)                            ----- - 2
                                   3  3
                                  x  y
(%i13) delta : subst([x^3=v,y^3=v], delta );
(%o13)                              14
(%i14) dsdx2 : diff(s,x,2);
                                    4 v
(%o14)                              ---
                                     3
                                    x
(%i15) dsdy2 : diff(s,y,2);
                                    4 v
(%o15)                              ---
                                     3
                                    y
```

6.4.2 Example 2: Maximize the Cross Sectional Area of a Trough

A long rectangular sheet of aluminum of width $\mathbf{L}$ is to be formed into a flat bottom trough by bending both a left and right hand length $\mathbf{x}$ up by an angle $\boldsymbol{\theta}$ with the horizontal. A cross section view is then:

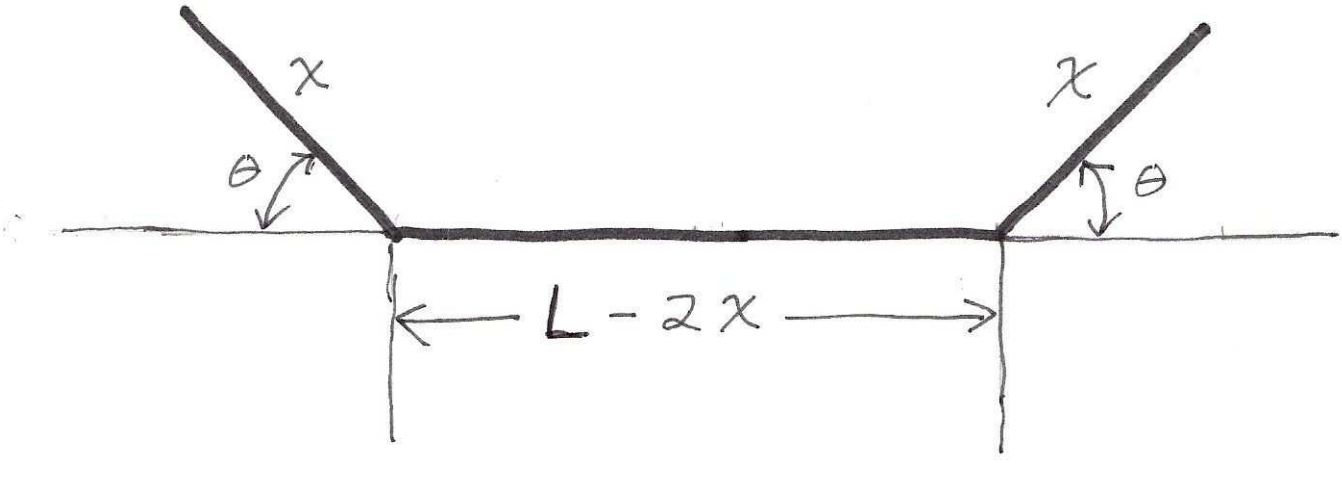

Figure 8: Flat Bottomed Trough

The lower base is $\mathbf{l} = \mathbf{L} - \mathbf{2}\,\mathbf{x}$, the upper base is $\mathbf{u} = \mathbf{l} + \mathbf{2}\,\mathbf{x}\cos(\boldsymbol{\theta})$. The altitude is $\mathbf{h} = \mathbf{x}\sin(\boldsymbol{\theta})$. The area of a trapezoid whose parallel sides are $\mathbf{l}$ and $\mathbf{u}$ and whose altitude is $\mathbf{h}$ is $(\mathbf{h}/\mathbf{2})\,(\mathbf{l}+\mathbf{u})$ (ie., the height times the average width: Exercise: check this by calculating the area of a rectange $\mathbf{h}\,\mathbf{u}$, where $\mathbf{u}$ is the larger base, and then subtracting the area of the two similar right triangles on the left and right hand sides.) Hence the area of the cross section (called **s** in our Maxima work) is $\boldsymbol{A = L\,x\,\sin(\theta) - 2\,x^2\,\sin(\theta) + x^2\,\sin(\theta)\,\cos(\theta)}$

Using condition I equations, we proceed to find the extremum solutions for $\mathbf{x}$ and $\boldsymbol{\theta}$ (which is called **th** in our Maxima work). Our method is merely one possible path to a solution.

```
(%i1) s : L*x*sin(th) - 2*x^2*sin(th) + x^2*sin(th)*cos(th);
                                                 2              2
(%o1)           sin(th) x L + cos(th) sin(th) x  - 2 sin(th) x
(%i2) dsdx : ( diff(s,x), factor(%%) );
(%o2)                   sin(th) (L + 2 cos(th) x - 4 x)
```

We see that **one way** we can get **dsdx** to be zero is to set $\sin(\boldsymbol{\theta}) = \mathbf{0}$, however this would mean the angle of bend was zero degrees, which is not the solution of interest. Hence our first equation comes from setting the second factor of **dsdx** equal to zero.

```
(%i3) eq1 : dsdx/sin(th) = 0;
(%o3)                       L + 2 cos(th) x - 4 x = 0
(%i4) solx : solve(eq1,x);
                                          L
(%o4)                          [x = - -------------]
                                     2 cos(th) - 4
(%i5) dsdth : ( diff(s,th), factor(%%) );
                                2             2
(%o5)        x (cos(th) L - sin (th) x + cos (th) x - 2 cos(th) x)
```

We see that we can get **dsdth** to be zero if we set $x = 0$, but this is not the physical situation we are examining. We assume (of course) that $x \neq 0$, and arrive at our second equation of condition I by setting the second factor of **dsdth** equal to zero.

```
(%i6) eq2 : dsdth/x = 0;
                            2             2
(%o6)         cos(th) L - sin (th) x + cos (th) x - 2 cos(th) x = 0
(%i7) eq2 : ( subst(solx,eq2), ratsimp(%%) );
                         2           2
                    (sin (th) + cos (th) - 2 cos(th)) L
(%o7)               ----------------------------------- = 0
                                2 cos(th) - 4
```

The only way we can satisfy **eq2** is to set the numerator of the left hand side equal to zero, and divide out the factor **L** which is positive:

```
(%i8) eq2 : num(lhs(eq2) )/L = 0;
                          2           2
(%o8)                  sin (th) + cos (th) - 2 cos(th) = 0
(%i9) eq2 : trigsimp(eq2);
(%o9)                           1 - 2 cos(th) = 0
(%i10) solcos : solve( eq2, cos(th) );
                                              1
(%o10)                            [cos(th) = -]
                                              2
(%i11) solx : subst(solcos, solx);
                                            L
(%o11)                                [x = -]
                                            3
(%i12) solth : solve(solcos,th);
'solve' is using arc-trig functions to get a solution.
Some solutions will be lost.
                                           %pi
(%o12)                               [th = ---]
                                            3
```

6.5 Tangent and Normal of a Point of a Curve Defined by an Implicit Function

In the following, Maxima assumes that **y** is independent of **x**:

```
(%i1) diff(x^2 + y^2,x);
(%o1)                                  2 x
```

We can indicate explicitly that **y** depends on **x** for purposes of taking this derivative by replacing **y** by **y(x)**.

```
(%i2) diff(x^2 + y(x)^2,x);
                                      d
(%o2)                    2 y(x) (-- (y(x))) + 2 x
                                      dx
```

Instead of using the functional notation to indicate dependency, we can use the **depends** function before taking the derivative.

```
(%i3) depends(y,x);
(%o3)                                [y(x)]
(%i4) g : diff(x^2 + y^2, x);
                                      dy
(%o4)                            2 y -- + 2 x
                                      dx
(%i5) grind(g)$
2*y*'diff(y,x,1)+2*x$
(%i6) gs1 : subst('diff(y,x) = x^3, g );
                                      3
(%o6)                            2 x  y + 2 x
```

In **%i4** we defined **g** as the derivative (with respect to **x**) of the expression **x^2 + y^2**, after telling Maxima that the symbol **y** is to be considered dependent on the value of **x**. Since Maxima, as yet, has no specific information about the nature of the dependence of **y** on **x**, the output is expressed in terms of the "noun form" of **diff**, which Maxima's pretty print shows as $\frac{dy}{dx}$. To see the "internal" form, we again use the **grind** function, which shows the explicit noun form **'diff(y,x,1)**. This is useful to know if we want to later replace that unknown derivative with a known result, as we do in input **%i6**.

However, Maxima doesn't do anything creative with the derivative substitution done in **%i6**, like working backwards to what the explicit function of **x** the symbol **y** might stand for (different answers differ by a constant). The output **%o6** still contains the symbol **y**.

Two ways to later implement our knowledge of **y(x)** are shown in steps **%i7** and **%i8**, which use the **ev** function. (In Chapter 1 we discussed using **ev** for making substitutions, although the use of **subst** is more generally recommended for that job.)

```
(%i7) ev(g,y=x^4/4,diff);
                                      7
                                     x
(%o7)                               -- + 2 x
                                     2
(%i8) ev(g,y=x^4/4,nouns);
                                      7
                                     x
(%o8)                               -- + 2 x
                                     2
(%i9) y;
                                      y
(%o9)
(%i10) g;
                                      dy
(%o10)                           2 y -- + 2 x
                                      dx
```

We see that Maxima does not bind the symbol **y** to anything when we call **ev** with an equation like **y = x^4/4**, and that the binding of **g** has not changed.

A list which reminds you of all dependencies in force is **dependencies**. The Maxima function **diff** is the only core function which makes use of the **dependencies** list. The functions **integrate** and **laplace** do not use the **depends** assignments; one must indicate the dependence explicitly by using functional notation.

In the following, we first ask for the contents of the **dependencies** list and then ask Maxima to remove the above dependency of **y** on **x**, using **remove**, then check the list contents again, and carry out the previous differentiation with Maxima no longer assuming that **y** depends on **x**.

```
(%i11) dependencies;
(%o11)                              [y(x)]
(%i12) remove(y, dependency);
(%o12)                               done
(%i13) dependencies;
(%o13)                                []
(%i14) diff(x^2 + y^2,x);
(%o14)                               2 x
```

On can also remove the properties associated with the symbol **y** by using **kill (y)**, although this is more drastic than using **remove**.

```
(%i15) depends(y,x);
(%o15)                               [y(x)]
(%i16) dependencies;
(%o16)                               [y(x)]
(%i17) kill(y);
(%o17)                               done
(%i18) dependencies;
(%o18)                                []
(%i19) diff(x^2+y^2,x);
(%o19)                               2 x
```

There are many varieties of using the **kill** function. The way we are using it here corresponds to the syntax:

> Function: **kill (a_1, ..., a_n)**
> Removes all bindings (value, function, array, or rule) from the arguments **a_1, ..., a_n**. An argument **a_k** may be a symbol or a single array element.

The list **dependencies** is one of the lists Maxima uses to hold information introduced during a work session. You can use the command **infolists** to obtain a list of the names of all of the information lists in Maxima.

```
(%i1) infolists;
(%o2) [labels, values, functions, macros, arrays, myoptions, props, aliases,
                   rules, gradefs, dependencies, let_rule_packages, structures]
```

When you first start up Maxima, each of the above named lists is empty.

```
(%i2) functions;
(%o2)                                 []
```

6.5.1 Tangent of a Point of a Curve Defined by $\mathbf{f(x,y)=0}$

Suppose some plane curve is defined by the equation $\mathbf{f(x,y)=0}$. Every point $\mathbf{(x,y)}$ belonging to the curve must satisfy that equation. For such pairs of numbers, changing $\mathbf{x}$ forces a change in $\mathbf{y}$ so that the equation of the curve is still satisfied. When we start plotting tangent lines to plane curves, we will need to evaluate the change in $\mathbf{y}$, given a change in $\mathbf{x}$, such that the numbers $\mathbf{(x,y)}$ are always points of the curve. Let's then regard $\mathbf{y}$ as depending on $\mathbf{x}$ via the equation of the curve. Given that the equation $\mathbf{f(x,y)=0}$ is valid, we obtain another valid equation by taking the derivative of both sides of the equation with respect to $\mathbf{x}$. Let's work just on the left hand side of the resulting equation for now:

```
(%i1) depends(y,x);
(%o1)                                [y(x)]
(%i2) diff(f(x,y),x);
                                  d
(%o2)                             -- (f(x, y))
                                  dx
```

We can make progress by assigning values to the partial derivative of $\mathbf{f}(\mathbf{x},\mathbf{y})$ with respect to the first argument $\mathbf{x}$ and also to the partial derivative of $\mathbf{f}(\mathbf{x},\mathbf{y})$ with respect to the second argument $\mathbf{y}$, using the **gradef** function. The assigned values can be explicit mathematical expressions or symbols. We are going to adopt the symbol **dfdx** to stand for the partial derivative

$$\mathbf{dfdx} = \left(\frac{\partial \mathbf{f}(\mathbf{x},\mathbf{y})}{\partial \mathbf{x}}\right)_{\mathbf{y}},$$

where the $\mathbf{y}$ subscript means "treat $\mathbf{y}$ as a constant when evaluating this derivative".

Likewise, we will use the symbol **dfdy** to stand for the partial derivative of $\mathbf{f}(\mathbf{x},\mathbf{y})$ with respect to $\mathbf{y}$, holding $\mathbf{x}$ constant:

$$\mathbf{dfdy} = \left(\frac{\partial \mathbf{f}(\mathbf{x},\mathbf{y})}{\partial \mathbf{y}}\right)_{\mathbf{x}}.$$

```
(%i3) gradef(f(x,y), dfdx, dfdy );
(%o3)                                f(x, y)
(%i4) g : diff( f(x,y), x );
                                      dy
(%o4)                           dfdy -- + dfdx
                                      dx
(%i5) grind(g)$
dfdy*'diff(y,x,1)+dfdx$
(%i6) g1 : subst('diff(y,x) = dydx, g);
(%o6)                          dfdy dydx + dfdx
```

We have adopted the symbol **dydx** for the "rate of change" of $\mathbf{y}$ as $\mathbf{x}$ varies, subject to the constraint that the numbers $(\mathbf{x},\mathbf{y})$ always satisfy the equation of the curve $\mathbf{f}(\mathbf{x},\mathbf{y}) = \mathbf{0}$, which implies that **g1 = 0**.

```
(%i7) solns : solve(g1=0, dydx);
                                          dfdx
(%o7)                           [dydx = - ----]
                                          dfdy
```

We see that Maxima knows enough about the rules for differentiation to allow us to get to a famous calculus formula. If $\mathbf{x}$ and $\mathbf{y}$ are constrained by the equation $\mathbf{f}(\mathbf{x},\mathbf{y}) = \mathbf{0}$, then the "slope" of the local tangent to the point $(\mathbf{x},\mathbf{y})$ is

$$\frac{\mathbf{d\,y}}{\mathbf{d\,x}} = -\frac{\left(\frac{\partial \mathbf{f}(\mathbf{x},\mathbf{y})}{\partial \mathbf{x}}\right)_{\mathbf{y}}}{\left(\frac{\partial \mathbf{f}(\mathbf{x},\mathbf{y})}{\partial \mathbf{y}}\right)_{\mathbf{x}}}. \tag{6.8}$$

Of course, this formal expression has no meaning at a point where the denominator is zero.

In your calculus book you will find the derivation of the differentiation rule embodied in our output **%o4** above:

$$\frac{d}{d\,x} f(x,y(x)) = \left(\frac{\partial f(x,y)}{\partial x}\right)_{y} + \left(\frac{\partial f(x,y)}{\partial y}\right)_{x} \frac{d\,y(x)}{d\,x} \tag{6.9}$$

Remember that Maxima knows only what the code writers put in; we normally assume that the correct laws of mathematics are encoded as an aid to calculation. If Maxima were not consistent with the differentiation rule Eq. (6.9), then a bug would be present and would have to be removed.

The result Eq.(6.8) provides a way to find the slope (which we call **m**) at a general point $(\mathbf{x},\mathbf{y})$.

This "slope" should be evaluated at the curve point $(\mathbf{x_0},\mathbf{y_0})$ of interest to get the tangent and normal in numerical form. Recall our discussion in the first part of Section(6.3) where we derived the relation between the slopes of the tangent and normal. If the numerical value of the slope is $\mathbf{m_0}$, then the tangent (line) is the equation $\mathbf{y} = \mathbf{m_0}\,(\mathbf{x} - \mathbf{x_0}) + \mathbf{y_0}$, and the normal (line) is the equation $\mathbf{y} = -(\mathbf{x} - \mathbf{x_0})/\mathbf{m_0} + \mathbf{y_0}$

A Function which Solves for $\mathbf{d\,y/d\,x}$ given that $\mathbf{f(x,y)=0}$

Given an equation relating $\mathbf{x}$ and $\mathbf{y}$, we assume we can rewrite that equation in the form $\mathbf{f(x,y)=0}$, and then define the following function:

```
(%i1) dydx(expr,x,y) := -diff(expr,x)/diff(expr,y);
                                          - diff(expr, x)
(%o1)                dydx(expr, x, y) := ---------------
                                           diff(expr, y)
```

As a first example, consider finding $\mathbf{d\,y/d\,x}$ given that $\mathbf{x^3+y^3=1}$.

```
(%i2) dydx( x^3+y^3-1,x,y);
                                        2
                                       x
(%o2)                                - --
                                        2
                                       y
```

Next find $\mathbf{d\,y/d\,x}$ given that $\mathbf{\cos(x^2-y^2)=y\,cos(x)}$.

```
(%i3) r1 : dydx( cos(x^2 - y^2) - y*cos(x),x,y );
                                  2    2
                          - 2 x sin(y  - x ) - sin(x) y
(%o3)                    ------------------------------
                                   2    2
                           - 2 y sin(y  - x ) - cos(x)
(%i4) r2 : (-num(r1))/(-denom(r1));
                                 2    2
                          2 x sin(y  - x ) + sin(x) y
(%o4)                     ---------------------------
                                   2    2
                            2 y sin(y  - x ) + cos(x)
```

Maxima does not simplify **r1** by cancelling the minus signs automatically, and we have resorted to dividing the negative of the numerator by the negative of the denominator(!) to get the form of **r2**. Notice also that Maxima has chosen to write $\mathbf{\sin(y^2-x^2)}$ instead of $\mathbf{\sin(x^2-y^2)}$.

Finally, find $\mathbf{d\,y/d\,x}$ given that $\mathbf{3\,y^4+4\,x-x^2\sin(y)-4=0}$.

```
(%i5) dydx(3*y^4 +4*x -x^2*sin(y) - 4, x, y );
                                 2 x sin(y) - 4
(%o5)                          -----------------
                                   3    2
                                12 y  - x  cos(y)
```

6.5.2 Example 1: Tangent and Normal of a Point of a Circle

As an easy first example, lets consider a circle of radius $\mathbf{r}$ defined by the equation $\mathbf{x^2+y^2=r^2}$. Let's choose $\mathbf{r=1}$. Then $\mathbf{f(x,y)=x^2+y^2-1=0}$ defines the circle of interest, and we use the "slope" function above to calculate the slope $\mathbf{m}$ for a general point $\mathbf{(x,y)}$.

```
(%i1) dydx(expr,x,y) := -diff(expr,x)/diff(expr,y)$
(%i2) f:x^2 + y^2-1$
(%i3) m : dydx(f,x,y);
                                        x
(%o3)                                 - -
                                        y
```

We then choose a point of the circle, evaluate the slope at that point, and construct the tangent and normal at that point.

```
(%i4) fpprintprec:8$
(%i5) [x0,y0] : [0.5,0.866];
(%o5)                              [0.5, 0.866]
(%i6) m : subst([x=x0,y=y0],m );
(%o6)                               - 0.577367
(%i7) tangent : y = m*(x-x0) + y0;
(%o7)                    y = 0.866 - 0.577367 (x - 0.5)
(%i8) normal : y = -(x-x0)/m + y0;
(%o8)                       y = 0.866 - 1.732 (0.5 - x)
```

We can then use **qdraw** to show the circle with both the tangent and normal.

```
(%i9) load(draw);
(%o9)   C:/PROGRA~1/MAXIMA~4.0/share/maxima/5.15.0/share/draw/draw.lisp
(%i10) load(qdraw);
               qdraw(...), qdensity(...), syntax: type qdraw();

(%o10)                           c:/work2/qdraw.mac
(%i11) ratprint:false$
(%i12) qdraw(key(bottom),ipgrid(15),
          imp([ f = 0,tangent,normal],x,-2.8,2.8,y,-2,2 ),
           pts([ [0.5,0.866]],ps(2) ) )$
```

The result is

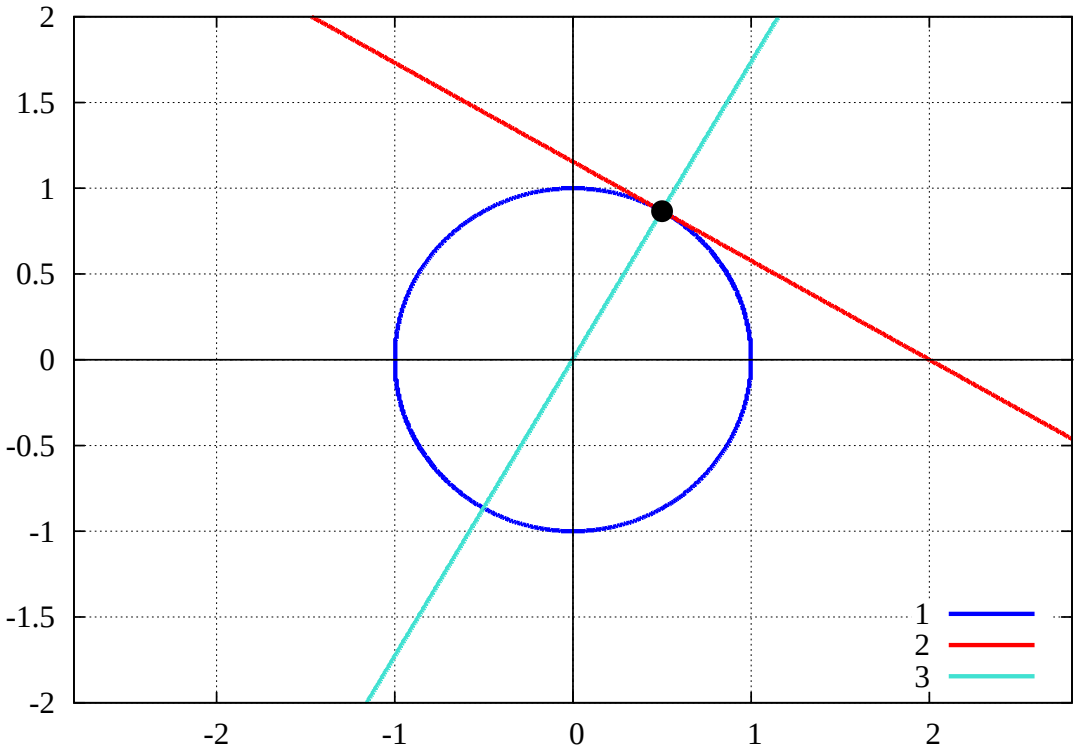

Figure 9: Tangent and Normal to $\mathbf{x^2 + y^2 = 1}$ at point $(\mathbf{0.5, 0.866})$

6.5.3 Example 2: Tangent and Normal of a Point of the Curve $\sin(2\,x)\ \cos(y) = 0.5$

A curve is defined by $\mathbf{f(x,y) = \sin(2\,x)\cos(y) - 0.5 = 0}$, with $\mathbf{0 \le x \le 2}$ and $\mathbf{0 \le y \le 2}$. Using Eq. (6.8) we calculate the slope of the tangent to this curve at some point $(\mathbf{x,y})$ and then specialize to the point $(\mathbf{x = 1, y = y_0})$ with $\mathbf{y_0}$ to be found:

```
(%i13) f : sin(2*x)*cos(y) - 0.5$
(%i14) m : dydx(f,x,y);
                                2 cos(2 x) cos(y)
(%o14)                          -----------------
                                 sin(2 x) sin(y)
(%i15) s1 : solve(subst(x=1,f),y);
'solve' is using arc-trig functions to get a solution.
Some solutions will be lost.
                                          1
(%o15)                       [y = acos(--------)]
                                       2 sin(2)
(%i16) fpprintprec:8$
(%i17) s1 : float(s1);
(%o17)                          [y = 0.988582]
(%i18) m : subst([ x=1, s1[1] ], m);
                                1.3166767 cos(2)
(%o18)                          ----------------
                                     sin(2)
(%i19) m : float(m);
(%o19)                          - 0.602587
(%i20) mnorm : -1/m;
(%o20)                           1.6595113
(%i21) y0 : rhs( s1[1] );
(%o21)                           0.988582
(%i22) qdraw( imp([f = 0, y - y0 = m*(x - 1),
               y - y0 = mnorm*(x - 1) ],x,0,2,y,0,1.429),
              pts( [ [1,y0] ], ps(2) ), ipgrid(15))$
```

which produces the plot

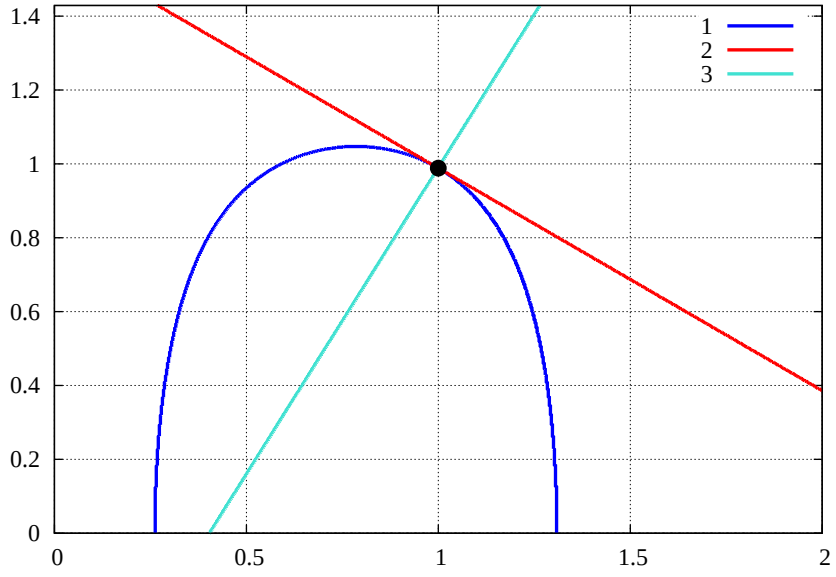

Figure 10: Tangent and Normal to $\mathbf{\sin(2\,x)\cos(y) = 0.5}$ at point $\mathbf{(1, 0.988)}$

6.5.4 Example 3: Tangent and Normal of a Point on a Parametric Curve: $x = \sin(t), y = \sin(2t)$

This example is the parametric curve example we plotted in Ch. 5. If we divide the differential of y by the differential of x, the common factor of dt will cancel out and we will have an expression for the slope of the tangent to the curve at a point determined by the value the parameter t, which in this example must be an angle expressed in radians (as usual in calculus).

We then specialize to a point on the curve corresponding to $x = 0.8$, with $0 \le t \le \pi/2$, which we solve for:

```
(%i23) m : diff(sin(2*t))/diff(sin(t));
                                  2 cos(2 t)
(%o23)                            ----------
                                    cos(t)
(%i24) x0 : 0.8;
(%o24)                                0.8
(%i25) tsoln : solve(x0 = sin(t), t);
                                                4
(%o25)                             [t = asin(-)]
                                                5
(%i26) tsoln : float(tsoln);
(%o26)                            [t = 0.927295]
(%i27) t0 : rhs( tsoln[1] );
(%o27)                              0.927295
(%i28) m : subst( t = t0, m);
(%o28)                             - 0.933333
(%i29) mnorm : -1/m;
(%o29)                              1.0714286
(%i30) y0 : sin(2*t0);
(%o30)                                0.96
(%i31) qdraw( xr(0, 2.1), yr(0,1.5), ipgrid(15),nticks(200),
              para(sin(t),sin(2*t),t,0,%pi/2, lc(brown) ),
             ex([y0+m*(x-x0),y0+mnorm*(x-x0)],x,0,2.1 ),
            pts( [ [0.8, 0.96]],ps(2) ) )$
```

with the resulting plot

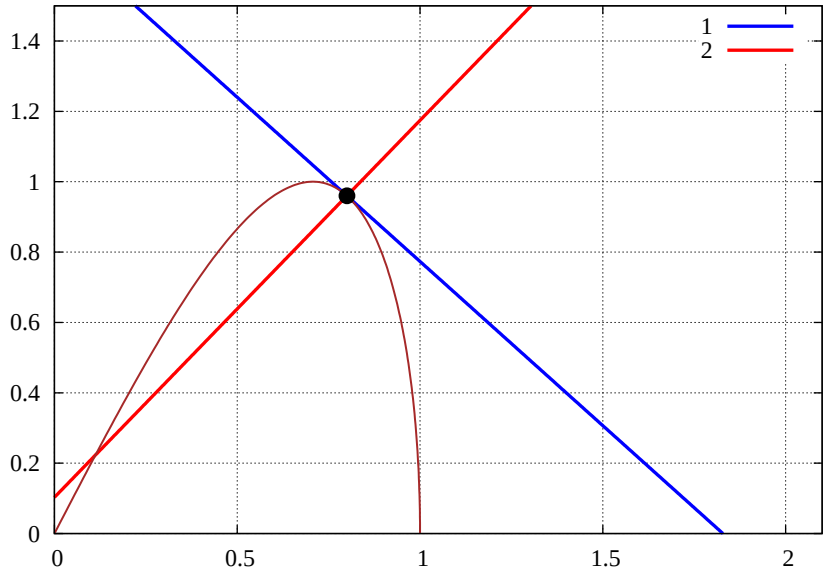

Figure 11: Tangent and Normal to $x = \sin(t)$, $y = \sin(2t)$ at point $t = 0.927$ radians

6.5.5 Example 4: Tangent and Normal of a Point on a Polar Plot: $\mathbf{x = r(t)\cos(t), y = r(t)\sin(t)}$

We use the polar plot example from ch. 5, in which we took $\mathbf{r(t) = 10/t}$, and again $\mathbf{t}$ is in radians, and we consider the interval $\mathbf{1 \le t \le \pi}$ and find the tangent and normal at the curve point corresponding to $\mathbf{t = 2}$ radians. We find the general slope of the tangent by again forming the ratio of the differential $\mathbf{dy}$ to the differential $\mathbf{dx}$.

```
(%i32) r : 10/t$
(%i33) xx : r * cos(t)$
(%i34) yy : r * sin(t)$
(%i35) m : diff(yy)/diff(xx)$
(%i36) m : ratsimp(m);
                                sin(t) - t cos(t)
(%o36)                          -----------------
                                t sin(t) + cos(t)
(%i37) m : subst(t = 2.0, m);
(%o37)                               1.2418222
(%i38) mnorm : -1/m;
(%o38)                             - 0.805268
(%i39) x0 : subst(t = 2.0, xx);
(%o39)                             - 2.0807342
(%i40) y0 : subst(t = 2.0, yy);
(%o40)                               4.5464871
(%i41) qdraw(polar(10/t,t,1,3*%pi,lc(brown) ),
           xr(-6.8,10),yr(-3,9),
           ex([y0 + m*(x-x0),y0 + mnorm*(x-x0)],x,-6.8,10 ),
           pts( [ [x0,y0] ], ps(2),pk("t = 2 rad") ) );
```

which produces the plot:

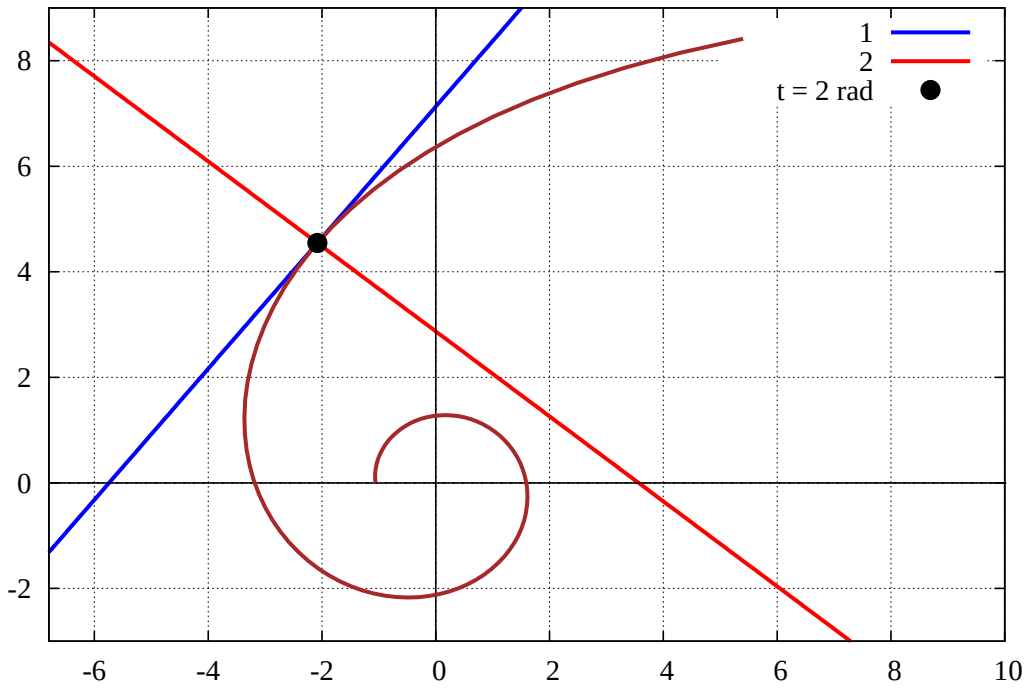

Figure 12: Tangent and Normal to $\mathbf{x = 10\cos(t)/t}$, $\mathbf{y = 10\sin(t)/t}$ at point $\mathbf{t = 2}$ radians

6.6 Limit Examples Using Maxima's limit Function

Maxima has a powerful **limit** function which uses l'Hospital's rule and Taylor series expansions to investigate the limit of a univariate function as the variable approaches some point. The Maxima manual has the following description the Maxima function **limit**:

> Function: **limit(expr, x, val, dir)**
> Function: **limit(expr, x, val)**
> Function: **limit(expr)**
> Computes the limit of **expr** as the real variable **x** approaches the value **val** from the direction **dir**. **dir** may have the value **plus** for a limit from above, **minus** for a limit from below, or may be omitted (implying a two-sided limit is to be computed).
> **limit** uses the following special symbols: **inf** (positive infinity) and **minf** (negative infinity). On output it may also use **und** (undefined), **ind** (indefinite but bounded) and **infinity** (complex infinity).
> **lhospitallim** is the maximum number of times L'Hospital's rule is used in **limit**. This prevents infinite looping in cases like **limit (cot(x)/csc(x), x, 0)**.
> **tlimswitch** when **true** will allow the **limit** command to use Taylor series expansion when necessary.
> **limsubst** prevents **limit** from attempting substitutions on unknown forms. This is to avoid bugs like **limit (f(n)/f(n+1),n,inf)** giving 1. Setting **limsubst** to **true** will allow such substitutions.
> **limit** with one argument is often called upon to simplify constant expressions, for example, **limit (inf-1)**.
> **example (limit)** displays some examples.

Here is the result of calling Maxima's **example** function:

```
(%i1) example(limit)$
(%i2) limit(x*log(x),x,0,plus)
(%o2)                                 0
(%i3) limit((x+1)^(1/x),x,0)
(%o3)                                 %e
(%i4) limit(%e^x/x,x,inf)
(%o4)                                 inf
(%i5) limit(sin(1/x),x,0)
(%o5)                                 ind
```

Most use of **limit** will use the first two ways to call **limit**. The "direction" argument is optional. The default values of the option switches mentioned above are:

```
(%i6) [lhospitallim,tlimswitch,limsubst];
(%o6)                          [4, true, false]
```

Thus the default Maxima behavior is to allow the use of a Taylor series expansion in finding the correct limit. (We will discuss Taylor series expansions soon in this chapter.) The default is also to prevent "substitutions" on unknown (formal) functions. The third (single argument) syntax is illustrated by

```
(%i7) limit(inf - 1);
(%o7)                                 inf
```

The expression presented to the **limit** function in input **%i7** contains only known constants, so there are no unbound (formal) parameters like **x** for **limit** to worry about.

Here is a use of **limit** which mimics the calculus definition of a derivative of a power of **x**.

```
(%i8) limit( ( (x+eps)^3 - x^3 )/eps, eps, 0 );
                                       2
(%o8)                                3 x
```

And a similar use of **limit** with $\ln(x)$:

```
(%i9) limit( (log(x+eps) - log(x))/eps, eps,0 );
                                      1
(%o9)                                 -
                                      x
```

What does Maxima do with a typical calculus definition of a derivative of a trigonometric function?

```
(%i10) limit((sin(x+eps)-sin(x))/eps, eps,0 );
Is  sin(x)  positive, negative, or zero?
p;
Is  cos(x)  positive, negative, or zero?
p;
(%o10)                                cos(x)
```

We see above a typical Maxima query before producing an answer. Using **p;** instead of **positive;** is allowed. Likewise one can use **n;** instead of **negative;**.

6.6.1 Discontinuous Functions

A simple example of a discontinuous function can be created using Maxima's **abs** function.
abs(expr) returns either the absolute value **expr**, or (if **expr** is complex) the complex modulus of **expr**.

We first plot the function $|x|/x$.

```
(%i11)  load(draw)$
(%i12)  load(qdraw)$
(%i13) qdraw( yr(-2,2),lw(8),ex(abs(x)/x,x,-1,1 ) )$
```

Here is that plot of $|x|/x$:

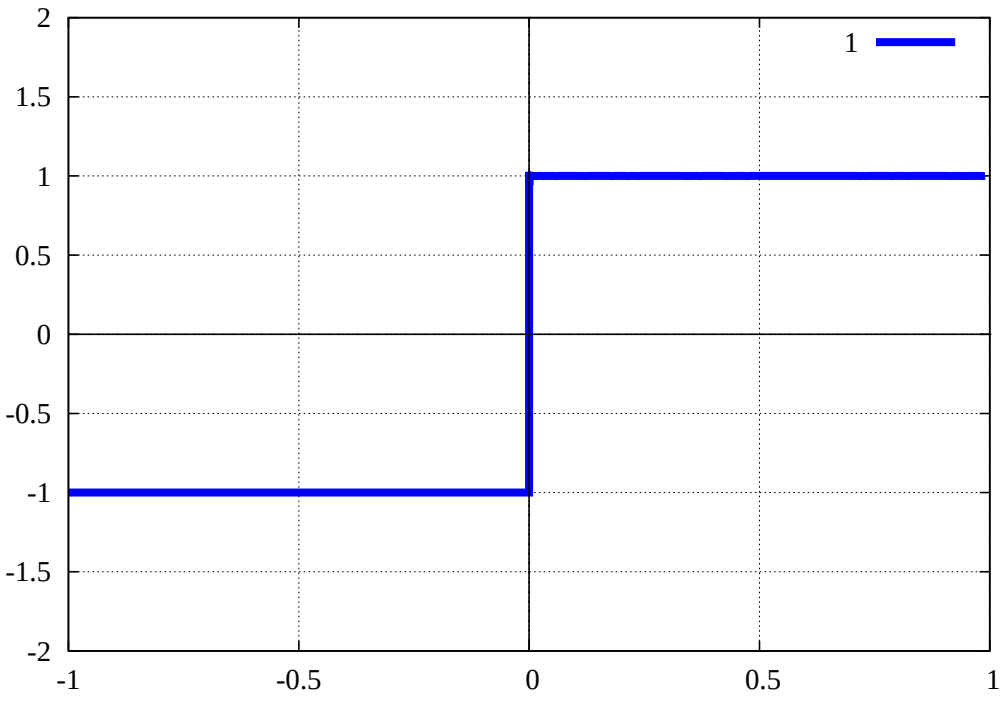

Figure 13: $|x|/x$

Maxima correctly evaluates the one-sided limits:

```
(%i14) limit(abs(x)/x,x,0,plus);
(%o14)                                1
(%i15) limit(abs(x)/x,x,0,minus);
(%o15)                               - 1
(%i16) limit(abs(x)/x,x,0);
(%o16)                               und
```

and Maxima also computes a derivative:

```
(%i17) g : diff(abs(x)/x,x);
                                  1       abs(x)
(%o17)                          ------ - ------
                                abs(x)       2
                                            x
(%i18) g, x = 0.5;
(%o18)                               0.0
(%i19) g, x = - 0.5;
(%o19)                               0.0
(%i20) g,x=0;
Division by 0
 -- an error.  To debug this try debugmode(true);
(%i21) limit(g,x,0,plus);
(%o21)                                0
(%i22) limit(g,x,0,minus);
(%o22)                                0
(%i23) load(vcalc)$
(%i24) plotderiv(abs(x)/x,x,-2,2,-2,2,1)$
                             abs(x)     1       abs(x)
                  plist = [------, ------ - ------]
                               x     abs(x)       2
                                                 x
```

The derivative does not simplify to $\mathbf{0}$ since the derivative is undefined at $\mathbf{x = 0}$. The plot of the step function and its derivative, as returned by **`plotderiv`** is

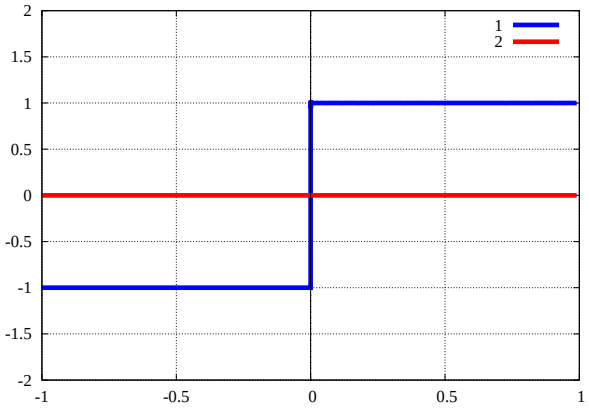

Figure 14: $|x|/x$ and its Maxima derivative

A homemade unit step function can now be defined by adding $\mathbf{1}$ to lift the function up to the value of $\mathbf{0}$ for $\mathbf{x} < \mathbf{0}$ and then dividing the result by $\mathbf{2}$ to get a "unit step function".

```
(%i25) mystep : ( (1 + abs(x)/x)/2 , ratsimp(%%) );
                                    abs(x) + x
(%o25)                              ----------
                                       2 x
```

We then use **qdraw** to plot the definition

```
(%i26) qdraw(yr(-1,2),lw(5),ex(mystep,x,-1,1) )$
```

with the result:

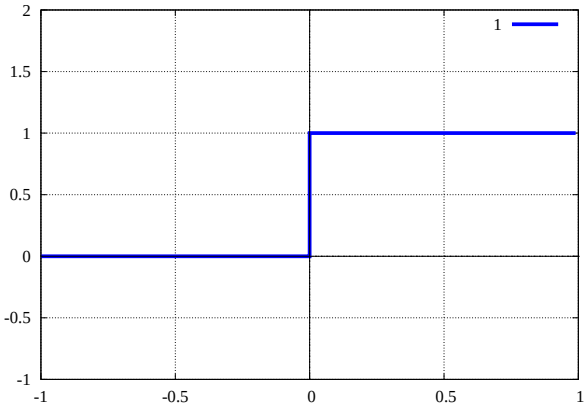

Figure 15: mystep

Maxima has a function called **unit_step** available if you load the **orthopoly** package. This function is "left continuous", since it has the value $\mathbf{0}$ for $\mathbf{x} \leq \mathbf{1}$. However, no derivative is defined, although you can use **gradef**.

```
(%i27) load(orthopoly)$
(%i28) map(unit_step,[-1/10,0,1/10] );
(%o28)                              [0, 0, 1]
(%i29) diff(unit_step(x),x);
                                d
(%o29)                          -- (unit_step(x))
                                dx
(%i30) gradef(unit_step(x),0);
(%o30)                            unit_step(x)
(%i31) diff(unit_step(x),x);
(%o31)                                  0
```

Of course, defining the derivative to be $\mathbf{0}$ everywhere can be dangerous in some circumstances. You can use **unit_step** in a plot using, say, **qdraw(yr(-1,2),lw(5),ex(unit_step(x),x,-1,1));**. Here we use **unit_step** to define a "unit pulse" function **upulse(x,x0,w)** which is a function of x which becomes equal to $\mathbf{1}$ when $\mathbf{x} = \mathbf{x_0}$ and has width $\mathbf{w}$.

```
(%i32) upulse(x,x0,w) := unit_step(x-x0) - unit_step(x - (x0+w))$
```

and then make a plot of three black pulses of width $\mathbf{0.5}$.

```
(%i33) qdraw(yr(-1,2), xr(-3,3),
           ex1( upulse(x,-3,0.5),x,-3,-2.49,lw(5)),
          ex1( upulse(x,-1,0.5),x,-1,-.49,lw(5)),
           ex1( upulse(x,1,0.5),x,1,1.51,lw(5)) )$
```

with the result:

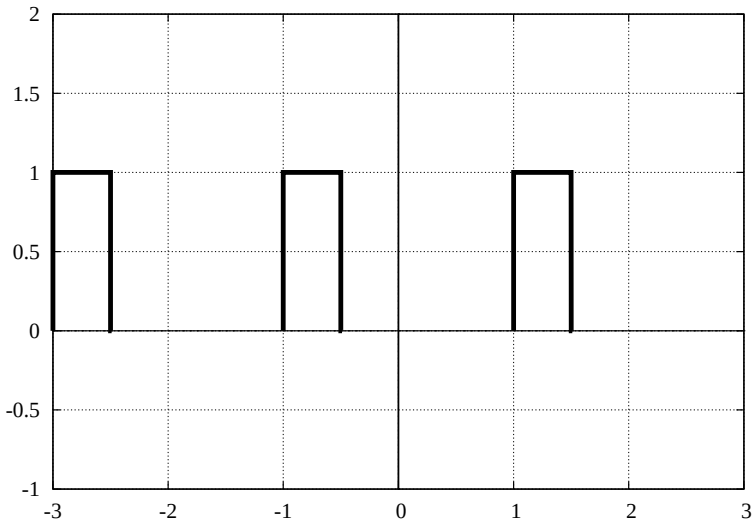

Figure 16: Using upulse(x,x0,w)

Of course, the above drawing can be done more easily with the **poly** function in **qdraw**, which uses **draw2d**'s **polygon** function.

6.6.2 Indefinite Limits

The **example** run showed that $\mathbf{sin(1/x)}$ has no definite limit as the variable $\mathbf{x}$ approaches zero. This function oscillates increasingly rapidly between $\pm\mathbf{1}$ as $\mathbf{x} \to \mathbf{0}$. It is instructive to make a plot of this function near the origin using the smallest line width:

```
(%i34) qdraw(lw(1),ex(sin(1/x),x,0.001,0.01));
```

The eps image reproduced here actually uses a finer line width than the Windows Gnuplot console window:

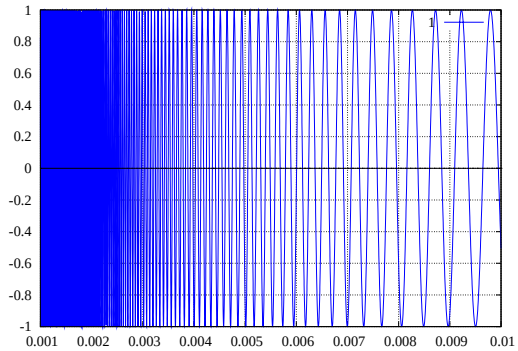

Figure 17: $\mathbf{sin(1/x)}$ Behavior Near $\mathbf{x = 0}$

The Maxima **limit** function correctly returns **ind** for "indefinite but bounded" when asked to find the limit as $x \to 0^+$.

```
(%i35) limit(sin(1/x),x,0,plus);
(%o35)                                ind
```

An example of a function which is well behaved at $x = 0$ but whose derivative is indefinite but bounded is $x^2 \sin(1/x)$, which approaches the value 0 at $x = 0$.

```
(%i36) g : x^2*sin(1/x)$
(%i37) limit(g,x,0);
(%o37)                                 0
(%i38) dgdx : diff(g,x);
                                     1          1
(%o38)                         2 sin(-) x - cos(-)
                                     x          x
(%i39) limit(dgdx,x,0);
(%o39)                                ind
```

In the first term of the derivative, $x \sin(1/x)$ is driven to 0 by the factor x, but the second term oscillates increasingly rapidly between ± 1 as $x \to 0$. For a plot, we use the smallest line width and color blue for the derivative, and use a thicker red curve for the original function $x^2 \sin(1/x)$.

```
(%i40) qdraw( yr(-1.5,1.5),ex1(2*x*sin(1/x)-cos(1/x),x,-1,1,lw(1),lc(blue)),
              ex1(x^2*sin(1/x),x,-1,1,lc(red) ) )$
```

Here is the plot:

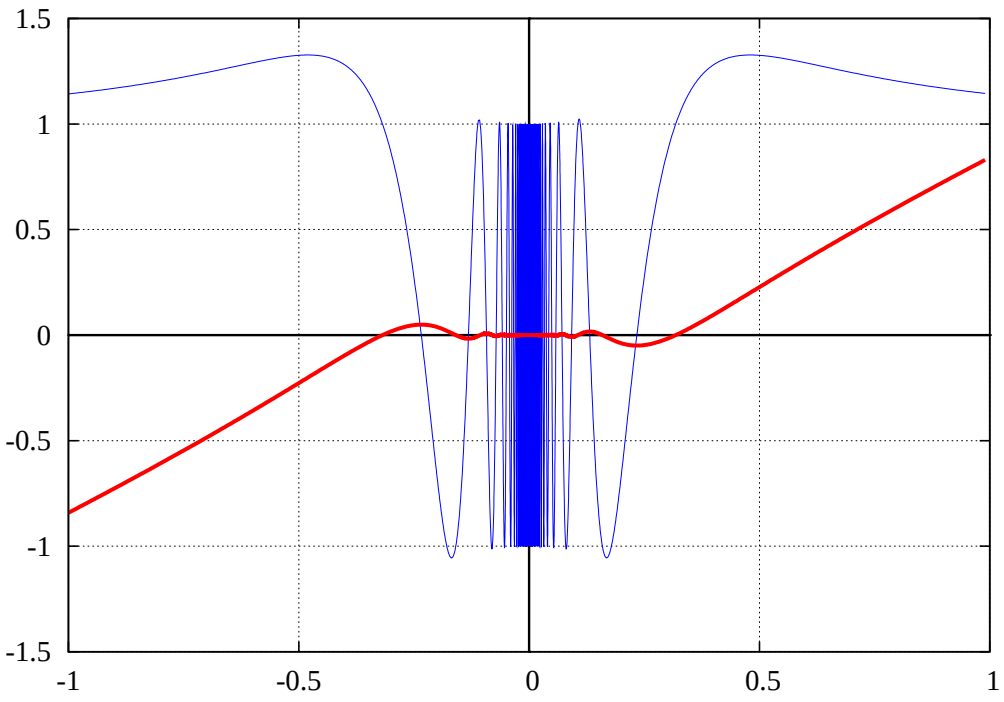

Figure 18: Indefinite but Bounded Behavior of Derivative

6.7 Taylor Series Expansions using taylor

A Taylor (or Laurent) series expansion of a univariate expression has the syntax

taylor(expr, x, a, n)

which will return a Taylor or Laurent series expansion of **expr** in the variable **x** about the point **x = a**, through terms proportional to **(x - a)^n** . Here are some examples of first getting the expansion, then evaluating the expansion at some other point, using both **at** and **subst**.

```
(%i1) t1 : taylor(sqrt(1+x),x,0,5);
                              2    3      4      5
                         x   x    x    5 x    7 x
(%o1)/T/            1 + - - -- + -- - ---- + ---- + . . .
                         2   8    16   128    256
(%i2) [ at( t1, x=1 ),subst(x=1, t1 ) ];
                                  365  365
(%o2)                            [---, ---]
                                  256  256
(%i3) float(%);
(%o3)                    [1.42578125, 1.42578125]
(%i4) t2: taylor (cos(x) - sec(x), x, 0, 5);
                                     4
                                  2   x
(%o4)/T/                       - x  - -- + . . .
                                     6
(%i5) [ at( t2, x=1 ),subst(x=1, t2 ) ];
                                    7    7
(%o5)                              [- -, - -]
                                    6    6
(%i6) t3 : taylor(cos(x),x,%pi/4,4);
                              %pi                  %pi 2                %pi 3
                    sqrt(2) (x - ---)   sqrt(2) (x - ---)    sqrt(2) (x - ---)
          sqrt(2)             4                    4                    4
(%o6)/T/ ------- - ---------------- - ------------------ + ------------------
             2              2                   4                    12
                                                                   %pi 4
                                                         sqrt(2) (x - ---)
                                                                    4
                                                    + ----------------- + . . .
                                                              48
(%i7) ( at(t3, x = %pi/3), factor(%%) );
                        4          3           2
           sqrt(2) (%pi  + 48 %pi  - 1728 %pi  - 41472 %pi + 497664)
(%o7)     ------------------------------------------------------
                                 995328
```

To expand an expression depending on two variables, say **(x,y)** , there are two essentially different forms. The first form is

taylor(expr, [x,y], [a,b],n)

which will expand in **x** about **x = a** and expand in the variable **y** about **y = b**, up through combined powers of order **n**. If **a = b**, then the simpler syntax:

taylor(expr, [x,y], a, n)

will suffice.

```
(%i8) t4 : taylor(sin(x+y),[x,y],0,3);
                            3        2      2      3
                           x  + 3 y x  + 3 y  x + y
(%o8)/T/           y + x - ------------------------ + . . .
                                       6
(%i9) (subst( [x=%pi/2, y=%pi/2], t4), ratsimp(%%) );
                                       3
                                    %pi  - 6 %pi
(%o9)                             - ------------
                                       6
(%i10) ( at( t4, [x=%pi/2,y=%pi/2]), ratsimp(%%) );
                                       3
                                    %pi  - 6 %pi
(%o10)                            - ------------
                                       6
```

Note the crucial difference between this last example and the next.

```
(%i11) t5 : taylor(sin(x+y),[x,0,3],[y,0,3] );
             3                  2                    3
            y                  y                  y   y             2
(%o11)/T/ y - -- + . . . + (1 - -- + . . .) x + (- - + -- + . . .) x
            6                  2                  2   12
                                                          2
                                                     1   y            3
                                                + (- - + -- + . . .) x  + . . .
                                                     6   12
(%i12) (subst([x=%pi/2,y=%pi/2],t5),ratsimp(%%) );
                              5         3
                           %pi  - 32 %pi  + 192 %pi
(%o12)                     ------------------------
                                    192
```

Thus the syntax
`taylor( expr, [x,a,nx], [y,b,ny] )`
will expand to higher combined powers in **`x`** and **`y`**.

We can differentiate and integrate a taylor series returned expression:

```
(%i13) t6 : taylor(sin(x),x,0,7 );
                                3    5      7
                               x    x      x
(%o13)/T/                  x - -- + --- - ---- + . . .
                               6    120    5040
(%i14) diff(t6,x);
                                 2    4    6
                                x    x    x
(%o14)/T/                   1 - -- + -- - --- + . . .
                                2    24   720
(%i15) integrate(%,x);
                                  7     5     3
                                 x     x     x
(%o15)                         - ---- + --- - -- + x
                                 5040  120   6
(%i16) integrate(t6,x);
                                  8      6     4    2
                                 x      x     x    x
(%o16)                         - ----- + --- - -- + --
                                 40320  720   24   2
```

A common use of the Taylor series is to expand a function of **x** in the form $\mathbf{f(x+dx)}$, for **dx** small.

```
(%i17) taylor(f(x+dx),dx,0,3);
                                                                !
                                                 2              !
                                                d               !           2
                                               (---- (f(x + dx))!       ) dx
                                                   2           !
                                                  ddx          !
                              !                                 !
                    d         !                                 !dx = 0
(%o17)/T/ f(x) + (--- (f(x + dx))!      ) dx + ------------------------------
                  ddx         !                           2
                              !dx = 0
                                                          !
                                             3            !
                                            d             !           3
                                           (---- (f(x + dx))!       ) dx
                                              3           !
                                             ddx          !
                                                          !dx = 0
                                        + ------------------------------ + . . .
                                                               6
(%i18) taylor(cos(x+dx),dx,0,4);
                                      2            3            4
                               cos(x) dx    sin(x) dx    cos(x) dx
(%o18)/T/ cos(x) - sin(x) dx - ---------- + ---------- + ---------- + . . .
                                    2            6            24
```

Another frequent use of the Taylor series is the expansion of a function which contains a small parameter which we will symbolize by **e**.

```
(%i19) g : log(a/e + sqrt(1+(a/e)^2 ) );
                                          2
                                   a      a
(%o19)                        log(- + sqrt(-- + 1))
                                   e      2
                                         e
(%i20) taylor(g,e,0,2);
                                                2
                                               e
(%o20)/T/          - log(e) + log(2 a) + . . . + ---- + . . .
                                                 2
                                              4 a
```

More examples of Taylor and Laurent series expansions can be found in the Maxima manual.

6.8 Vector Calculus Calculations and Derivations using vcalc.mac

The package **vcalc.mac** has been mentioned in the introduction to this chapter. Here we give a few examples of using this package for vector calculus work.

General Comments: Working with the Batch File vcalcdem.mac

A separate file **vcalcdem.mac** is available with this chapter on the author's webpage. This file is designed to be introduced into Maxima using the command **batch(vcalcdem)** (if **vcalcdem.mac** is in your work directory, say, and you have set up your file search paths as suggested in Ch. 1) or **batch("vcalcdem.mac")**. You should first load in the **vcalc.mac** file which contains the definitions of the vcalc vector calculus functions.

There are three points to emphasize when using batch files in this mode. The first feature is the use of lines like **"this is a comment"$** in the file, which allow you to place readable comments between the usual Maxima input lines. Secondly, a Maxima input which simply defines something and for which there is no need to see the "output" line should end with the dollar sign **$**. Finally, when you "batch in" the file, you see neither the dollar signs, nor the semi-colons which end the other inputs. However, input and output numbers of the form **(%i12)** and **(%o12)** will be

seen for inputs which end with a semi-colon, whereas there are no output numbers for inputs which end with the dollar sign. Once you get a little practice reading the text form of the batch file and comparing that to the Maxima display of the "batched in" file, you should have no problem understanding what is going on. The first lines of **vcalcdem.mac** are:

```
" vcalcdem.mac: sample calculations and derivations"$
" default coordinates are cartesian (x,y,z)"$
" gradient and laplacian of a scalar field "$
depends(f,[x,y,z]);
grad(f);
lap(f);
```

Here is what you will see in your Maxima session:

```
(%i1) load(vcalc);
                vcalc.mac:   for syntax, type: vcalc_syntax();

 CAUTION: global variables set and used in this package:

       hhh1, hhh2, hhh3, uuu1, uuu2, uuu3, nnnsys, nnnprint, tttsimp

(%o1)                          c:/work2/vcalc.mac
(%i2) batch(vcalcdem)$
read and interpret file: #pc:/work5/vcalcdem.mac
(%i3)          vcalcdem.mac: sample calculations and derivations
(%i4)              default coordinates are cartesian (x,y,z)
(%i5)              gradient and laplacian of a scalar field
(%i6)                         depends(f, [x, y, z])
(%o6)                             [f(x, y, z)]
(%i7)                                grad(f)
cartesian  [x, y, z]
                                      df  df  df
(%o7)                                 [--, --, --]
                                      dx  dy  dz
(%i8)                                   lap(f)
cartesian  [x, y, z]
                                     2     2     2
                                    d f   d f   d f
(%o8)                               --- + --- + ---
                                      2     2     2
                                    dz    dy    dx
```

The default coordinates are cartesian **(x,y,z)**, and by telling Maxima that the otherwise undefined symbol **f** is to be treated as an explicit function of **(x,y,z)**, we get symbolic output from **grad(f)** and **lap(f)** which respectively produce the **gradient** and **laplacian** in the current coordinate system.

For each function used, a reminder is printed to your screen concerning the current coordinate system and the current choice of independent variables.

Three dimensional vectors (the only kind allowed by this package) are represented by lists with three elements. In the default cartesian coordinate system **(x,y,z)**, the first slot is the **x** component of the vector, the second slot is the **y** component, and the third slot is the **z** component.

Now that you have seen the difference between the file **vcalcdem.mac** and the Maxima response when using "batch", we will just show the latter for brevity. You can, of course, look at the file **vcalcdem.mac** with a text editor.

Here the batch file displays the divergence and curl of a general 3-vector in a cartesian coordinate system.

```
(%i9)                 divergence and curl of a vector field
(%i10)                         avec : [ax, ay, az]
(%o10)                            [ax, ay, az]
(%i11)                       depends(avec, [x, y, z])
(%o11)                [ax(x, y, z), ay(x, y, z), az(x, y, z)]
(%i12)                                div(avec)
cartesian  [x, y, z]
                                    daz   day   dax
(%o12)                              --- + --- + ---
                                    dz    dy    dx
(%i13)                                curl(avec)
cartesian  [x, y, z]
                          daz   day  dax   daz  day   dax
(%o13)                   [--- - ---, --- - ---, --- - ---]
                          dy    dz   dz    dx   dx    dy
```

We next have two vector calculus identities:

```
(%i14)           vector identities true in any coordinate system
(%i15)                              curl(grad(f))
cartesian  [x, y, z]
cartesian  [x, y, z]
(%o15)                                 [0, 0, 0]
(%i16)                              div(curl(avec))
cartesian  [x, y, z]
cartesian  [x, y, z]
(%o16)                                     0
```

and an example of finding the Laplacian of a vector field (rather than a scalar field) with an explicit example:

```
(%i17)                      laplacian of a vector field
                                  3       3     2    3
(%i18)                     aa : [x  y, x y  z, x  y z ]
                                  3       3     2    3
(%o18)                        [x  y, x y  z, x  y z ]
(%i19)                                lap(aa)
cartesian  [x, y, z]
                                                3      2
(%o19)                  [6 x y, 6 x y z, 2 y z  + 6 x  y z]
```

and an example of the use of the vector cross product which is included with this package:

```
(%i20)                          vector cross product
(%i21)                           bvec : [bx, by, bz]
(%o21)                              [bx, by, bz]
(%i22)                           lcross(avec, bvec)
(%o22)             [ay bz - az by, az bx - ax bz, ax by - ay bx]
```

We next change the current coordinate system to cylindrical with a choice of the symbols **rho**, **phi**, and **z** as the independent variables. We again start with some general expressions

```
(%i23)               cylindrical coordinates using (rho,phi,z)
(%i24)                       setcoord(cy(rho, phi, z))
(%o24)                                 true
(%i25)               gradient and laplacian of a scalar field
(%i26)                       depends(g, [rho, phi, z])
(%o26)                          [g(rho, phi, z)]
```

```
(%i27)                                grad(g)
cylindrical  [rho, phi, z]
                                        dg
                                        ----
                                  dg    dphi   dg
(%o27)                           [----, ----, --]
                                  drho  rho    dz
(%i28)                                lap(g)
cylindrical  [rho, phi, z]
                                    2
                                   d g
                             dg    -----
                             ----       2     2      2
                             drho   dphi    d g    d g
(%o28)                       ---- + ----- + --- + -----
                             rho        2      2       2
                                      rho     dz    drho
(%i29)                divergence and curl of a vector field
(%i30)                        bvec : [brh, bp, bz]
(%o30)                             [brh, bp, bz]
(%i31)                    depends(bvec, [rho, phi, z])
(%o31)        [brh(rho, phi, z), bp(rho, phi, z), bz(rho, phi, z)]
(%i32)                              div(bvec)
cylindrical  [rho, phi, z]
                                    dbp
                                    ----
                              brh   dphi   dbz   dbrh
(%o32)                        --- + ---- + --- + ----
                              rho   rho    dz    drho
(%i33)                              curl(bvec)
cylindrical  [rho, phi, z]
                 dbz                          dbrh
                 ----                         ----
                 dphi   dbp  dbrh   dbz       dphi   bp    dbp
(%o33)          [---- - ---, ---- - ----, - ---- + --- + ----]
                 rho    dz    dz    drho      rho    rho   drho
```

Instead of using **`setcoord`** to change the current coordinate system, we can insert an extra argument in the vector calculus function we are using. Here is an example of not changing the coordinate system, but telling Maxima to use **`r`** instead of **`rho`** with the currently operative cylindrical coordinates.

```
(%i34)   change from cylindrical coordinate label rho to r on the fly:
                                        1
(%i35)                          bvec : [0, -, 0]
                                        r
(%i36)                       div(bvec, cy(r, phi, z))
cylindrical  [r, phi, z]
(%o36)                                0
```

Here is an example of using this method to switch to spherical polar coordinates:

```
(%i37)         change to spherical polar coordinates on the fly
                                      sin(theta)
(%i38)                      cvec : [0, 0, ----------]
                                          2
                                         r
(%i39)                     div(cvec, s(r, theta, phi))
spherical polar  [r, theta, phi]
(%o39)                                0
```

```
(%i40)   coordinate system remains spherical unless explicitly changed
(%i41)                     cvec : [0, 0, r sin(theta)]
(%i42)                              div(cvec)
spherical polar  [r, theta, phi]
(%o42)                                0
(%i43)          example of div(vec) = 0 everywhere except r = 0
                                     1
(%i44)                          div([--, 0, 0])
                                      2
                                     r
spherical polar  [r, theta, phi]
(%o44)                                0
```

The best way to get familiar with **`vcalc.mac`** is just to play with it. Some syntax descriptions are built into the package.

6.9 Maxima Derivation of Vector Calculus Formulas in Cylindrical Coordinates

In this section we start with the cartesian (rectangular) coordinate expressions for the gradient and Laplacian of a scalar expression, and the divergence and curl of a vector expression. We then use Maxima to find the correct forms of these vector calculus formulas in a cylindrical coordinate system.

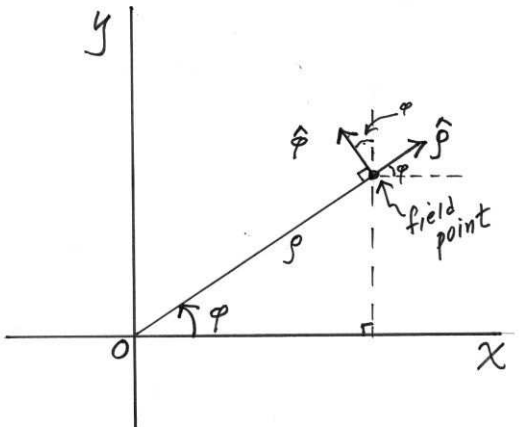

Figure 19: $\boldsymbol{\rho}$ and $\boldsymbol{\phi}$ unit vectors

Consider a change of variable from cartesian coordinates (x, y, z) to cylindrical coordinates (ρ, φ, z). Given (ρ, φ), we obtain (x, y) from the pair of equations $x = \rho\,\mathbf{cos}\,\varphi$ and $y = \rho\,\mathbf{sin}\,\varphi$. Given (x, y), we obtain (ρ, φ) from the equations $\rho = \sqrt{x^2 + y^2}$ (or $\rho^2 = x^2 + y^2$) and $\mathbf{tan}\,\varphi = y/x$ (or $\varphi = \mathbf{arctan}(y/x)$).

We consider the derivatives of a general scalar function $f(\rho, \varphi, z)$ which is an **explicit** function of the cylindrical coordinates (ρ, φ, z) and an **implicit** function of the cartesian coordinates (x, y) via the dependence of ρ and φ on x and y.

Likewise we consider a general three dimensional vector $\mathbf{B}(\rho, \varphi, z)$ which is an **explicit** function of ρ, φ, and z and an **implicit function** of (x, y).

Since many of the fundamental equations governing processes in the physical sciences and engineering can be written down in terms of the Laplacian, the divergence, the gradient, and the curl, we concentrate on using Maxima to do the "heavy lifting" (ie., the tedious algebra) to work out formulas for these operations in cylindrical coordinates.

Our approach to the use of vectors here is based on choosing an explicit set of three element lists to represent the cartesian unit vectors. We then use the built-in dot product of lists to implement the scalar product of three-vectors, used to check orthonormality, and we also provide a simple cross product function which can be used to check that the unit vectors we write down are related by the vector cross product to each other in the conventional way (the "right hand rule").

6.9.1 The Calculus Chain Rule in Maxima

Let $\mathbf{g}$ be some scalar function which depends **implicitly** on $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ via an **explicit** dependence on $(\mathbf{u}, \mathbf{v}, \mathbf{w})$. We want to express the partial derivative of $\mathbf{g}$ with respect to $\mathbf{x}$, $\mathbf{y}$, or $\mathbf{z}$ in terms of derivatives of $\mathbf{g}$ with respect to $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{w}$.

We first tell Maxima to treat $\mathbf{g}$ as an **explicit** function of $(\mathbf{u}, \mathbf{v}, \mathbf{w})$.

```
(%i1) depends(g,[u,v,w]);
(%o1)                              [g(u, v, w)]
```

If, at this point, we ask for the derivative of $\mathbf{g}$ with respect to $\mathbf{x}$, we get zero, since Maxima has no information yet about dependence of $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{w}$ on $\mathbf{x}$.

```
(%i2) diff(g,x);
(%o2)                                  0
```

We now need to use further **depends** statements, as in

```
(%i3) depends([u,v,w],[x,y,z]);
(%o3)                 [u(x, y, z), v(x, y, z), w(x, y, z)]
```

which now allows Maxima to demonstrate its knowledge of the "calculus chain rule", which Maxima writes as:

```
(%i4) diff(g,x);
                              dg dw   dg dv   dg du
(%o4)                         -- -- + -- -- + -- --
                              dw dx   dv dx   du dx
```

Note how Maxima writes the "chain rule" in this example, which would be written (using partial derivative notation) in a calculus text as

$$\frac{\partial \mathbf{g}(\mathbf{u},\mathbf{v},\mathbf{w})}{\partial \mathbf{x}} = \frac{\partial \mathbf{g}(\mathbf{u},\mathbf{v},\mathbf{w})}{\partial \mathbf{u}} \frac{\partial\, \mathbf{u}(\mathbf{x},\mathbf{y},\mathbf{z})}{\partial\, \mathbf{x}} + \frac{\partial \mathbf{g}(\mathbf{u},\mathbf{v},\mathbf{w})}{\partial \mathbf{v}} \frac{\partial\, \mathbf{v}(\mathbf{x},\mathbf{y},\mathbf{z})}{\partial\, \mathbf{x}} + \frac{\partial \mathbf{g}(\mathbf{u},\mathbf{v},\mathbf{w})}{\partial \mathbf{w}} \frac{\partial\, \mathbf{w}(\mathbf{x},\mathbf{y},\mathbf{z})}{\partial\, \mathbf{x}} \tag{6.10}$$

In addition to using a **depends** statement to tell Maxima about the $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ dependence of $(\mathbf{u}, \mathbf{v}, \mathbf{w})$, we can use the **gradef** function to replace derivatives with chosen substitutes.

```
(%i5)  (gradef(u,x,dudx),gradef(u,y,dudy),gradef(u,z,dudz),
         gradef(v,x,dvdx),gradef(v,y,dvdy),gradef(v,z,dvdz),
          gradef(w,x,dwdx),gradef(w,y,dwdy),gradef(w,z,dwdz) )$
```

Use of **diff** now produces the expression:

```
(%i6) diff(g,x);
                                   dg          dg          dg
(%o6)                        dwdx -- + dvdx -- + dudx --
                                   dw          dv          du
(%i7) grind(%)$
dwdx*'diff(g,w,1)+dvdx*'diff(g,v,1)+dudx*'diff(g,u,1)$
```

This is the method we will use in both this section on cylindrical coordinates and in the next section on spherical polar coordinates.

In the following sections we discuss successive parts of a batch file **cylinder.mac**, available with this chapter on the author's webpage. This file is designed to be introduced into Maxima with the input: **batch(cylinder)** (if **cylinder.mac** is in your work directory, say, and you have set up your file search paths as suggested in Ch. 1), or **batch("c:/work5/cylinder.mac")** if you need to supply the complete path. We have given some orientation concerning batch files in the previous section.

Relating (x, y) to (ρ, φ)

In **cylinder.mac** we use **rh** to represent ρ and **p** to represent the angle φ expressed in radians. The ranges of the independent variables are $\mathbf{0} < \rho < \infty$, $\mathbf{0} \leq \varphi \leq \mathbf{2}\,\boldsymbol{\pi}$, and $-\infty \leq \mathbf{z} \leq +\infty$.

Here is the beginning of the batch file **cylinder.mac**. First is defined **c3rule** as a list of replacement "rules" in the form of equations which can be used later by the **subst** function as described by **c3sub**. **rhxy** stands for an expression which produces ρ given (x, y). To get automatic simplification of **rh/abs(rh)** we use the **assume** function.

```
" ------------  cylinder.mac   ----------------------------"$
"  cylindrical coordinates (rho, phi, z ) = (rh, p, z)  "$

" replacement rules x,y,z to rh, p, z  "$

c3rule : [x = rh*cos(p), y = rh*sin(p)  ]$
c3sub(expr) := (subst(c3rule,expr),trigsimp(%%) )$

rhxy : sqrt(x^2 + y^2)$

assume(rh > 0)$
```

which Maxima displays as:

```
(%i1) batch("cylinder.mac")$
read and interpret file: #pc:/work5/cylinder.mac
(%i2)      ------------  cylinder.mac   ----------------------------
(%i3)         cylindrical coordinates (rho, phi, z ) = (rh, p, z)
(%i4)                replacement rules x,y,z to rh, p, z
(%i5)               c3rule : [x = rh cos(p), y = rh sin(p)]
(%i6)         c3sub(expr) := (subst(c3rule, expr), trigsimp(%%))
                                          2    2
(%i7)                          rhxy : sqrt(y  + x )
(%i8)                             assume(rh > 0)
```

Now that you have seen what Maxima does with a file introduced into Maxima via **batch("cylinder.mac")**, we will stop displaying the contents of **cylinder.mac** but just show you Maxima's response. You can look at **cylinder.mac** with a text editor to compare input with output.

We next tell Maxima how to work with the cylindrical coordinates and their derivatives: We let the symbol **drhdx**, for example, hold (for later use) $\boldsymbol{\partial}\,\rho/\boldsymbol{\partial}\,x$.

```
(%i9)           partial derivatives of rho and phi wrt x and y
(%i10)               drhdx : (diff(rhxy, x), c3sub(%%))
(%o10)                              cos(p)
(%i11)               drhdy : (diff(rhxy, y), c3sub(%%))
(%o11)                              sin(p)
                                      y
(%i12)              dpdx : (diff(atan(-), x), c3sub(%%))
                                      x
                                    sin(p)
(%o12)                            - ------
                                      rh
                                      y
(%i13)              dpdy : (diff(atan(-), y), c3sub(%%))
                                      x
                                    cos(p)
(%o13)                              ------
                                      rh
```

We thus have established the derivatives

$$\partial\rho(x,y)/\partial x = \cos\varphi \tag{6.11}$$

$$\partial\rho(x,y)/\partial y = \sin\varphi \tag{6.12}$$

$$\partial\varphi(x,y)/\partial x = -\sin\varphi/\rho \tag{6.13}$$

$$\partial\varphi(x,y)/\partial y = \cos\varphi/\rho \tag{6.14}$$

The batch file has not used any **depends** or **gradef** assignments so far. What the batch file did could have been done **interactively**, starting with the derivative of ρ with respect to x, say, as follows

```
(%i1) rhxy : sqrt(x^2 + y^2);
                                        2    2
(%o1)                                sqrt(y  + x )
(%i2) diff(rhxy,x);
                                        x
(%o2)                                -------------
                                        2    2
                                     sqrt(y  + x )
(%i3) subst([x=rh*cos(p),y=rh*sin(p)],%);
                                       cos(p) rh
(%o3)                  -------------------------------
                                 2      2      2      2
                       sqrt(sin (p) rh  + cos (p) rh )
(%i4) trigsimp(%);
                                      cos(p) rh
(%o4)                                 ---------
                                       abs(rh)
(%i5) assume(rh > 0)$
(%i6) ev (%o4);
(%o6)                                  cos(p)
```

and if we had used **assume(rh > 0)** earlier, we would not have had to include the **ev** step later, as you can verify by restarting Maxima or using **kill(all)** to redo the calculation.

Returning to the batch file, the necessary variable dependencies and the desired derivative replacements are assigned:

```
(%i14)              tell Maxima rh=rh(x,y) and p = p(x,y)
(%i15)           (gradef(rh, x, drhdx), gradef(rh, y, drhdy))
(%i16)             (gradef(p, x, dpdx), gradef(p, y, dpdy))
(%i17)                     depends([rh, p], [x, y])
```

6.9.2 Laplacian $\nabla^2 f(\rho,\varphi,z)$

The Laplacian of a scalar function f depending on (x,y,z),is defined as

$$\nabla^2 f(x,y,z) \equiv \frac{\partial^2 f(x,y,z)}{\partial x^2} + \frac{\partial^2 f(x,y,z)}{\partial y^2} + \frac{\partial^2 f(x,y,z)}{\partial z^2} \tag{6.15}$$

The batch file now calculates the Laplacian of a scalar function f when the function depends **explicitly** on the cylindrical coordinates (ρ,φ,z) and hence depends **implicitly** on (x,y) since ρ and φ each depend on (x,y). This latter dependence has already been introduced via **depends**, together with an automatic derivative substitution via **gradef**.

The batch file tells Maxima to treat **f** as an explicit function of **rh**, **p**, and **z** via **depends(f,[rh,p,z])**. At that point, if the batch file had asked for the first derivative of **f** with respect to **x**, the response would be

```
(%i98) diff (f,x);
                                        df
                                        -- sin(p)
                            df          dp
(%o98)                      --- cos(p) - ---------
                            drh            rh
```

The cartesian form of the Laplacian of f can thus be expressed in terms of the fundamental set of derivatives $\partial\,\rho/\partial\,x$, etc., which have already been established above. The next section of **cylinder.mac** then produces the response:

```
(%i18)          ------------------------------------------------
(%i19)            Laplacian of a scalar function f
(%i20)          ------------------------------------------------
(%i21)            tell Maxima to treat scalar function f as an
(%i22)                     explicit function of (rh,p,z)
(%i23)                       depends(f, [rh, p, z])
(%o23)                            [f(rh, p, z)]
(%i24)       calculate the Laplacian of the scalar function f(rh,p,z)
(%i25)                     using the cartesian definition
(%i26) (diff(f, z, 2) + diff(f, y, 2) + diff(f, x, 2), trigsimp(%%),
                                                              multthru(%%))
                                    2
                                   d f
                            df     ---
                            ---      2     2      2
                            drh    dp     d f    d f
(%o26)                      --- + --- + --- + ----
                            rh       2      2       2
                                    rh     dz     drh
(%i27)                               grind(%)
'diff(f,rh,1)/rh+'diff(f,p,2)/rh^2+'diff(f,z,2)+'diff(f,rh,2)$
```

We then have the result:

$$\boldsymbol{\nabla}^2 f(\rho,\varphi,z) = \frac{1}{\rho}\frac{\partial\, f}{\partial\,\rho} + \frac{\partial^2 f}{\partial\,\rho}^2 + \frac{1}{\rho^2}\frac{\partial^2 f}{\partial\,\varphi^2} + \frac{\partial^2 f}{\partial\, z^2} \tag{6.16}$$

The two terms involving derivatives with respect to ρ can be combined:

$$\frac{1}{\rho}\frac{\partial\, f}{\partial\,\rho} + \frac{\partial^2 f}{\partial\,\rho^2} = \frac{1}{\rho}\frac{\partial}{\partial\,\rho}\left(\rho\frac{\partial\, f}{\partial\rho}\right) \tag{6.17}$$

We clearly need to avoid points for which $\rho = 0$.

It is one thing to use Maxima to help derive the correct form of an operation like the Laplacian operator in cylindrical coordinates, and another to build a usable function which can be used (without repeating a derivation each time!) to calculate the Laplacian when we have some concrete function we are interested in.

The file **vcalc.mac**, available with this chapter on the author's webpage, contains usable functions for the Laplacian, gradient, divergence, and curl for the three coordinate systems: cartesian, cylindrical, and spherical polar.

However, here is a Maxima function specifically designed only for cylindrical coordinates which will calculate the Laplacian of a scalar expression in cylindrical coordinates. No effort at simplification is made inside this function; the result can be massaged by the alert user which Maxima always assumes is available. (Note: the Laplacian and other functions in **vcalc.mac** start with the general expressions valid in any orthonormal coordinate system, using the appropriate "scale factors" (h1,h2,h3), and some simplification is done. Also, the laplacian in **vcalc.mac** will correctly compute the Laplacian of a vector field as well as a scalar field.)

```
(%i1) cylaplacian(expr,rho,phi,z) :=
             (diff(expr,rho)/rho + diff(expr,phi,2)/rho^2 +
                diff(expr,rho,2) + diff(expr,z,2))$
```

We can show that the combinations $\boldsymbol{\rho^n \cos(n\,\varphi)}$ and $\boldsymbol{\rho^n \sin(n\,\varphi)}$ are each solutions of Laplace's equation $\boldsymbol{\nabla^2 u = 0}$.

```
(%i2) ( cylaplacian(rh^n*cos(n*p),rh,p,z), factor(%%) );
(%o2)                                0
(%i3) ( cylaplacian(rh^n*sin(n*p),rh,p,z), factor(%%) );
(%o3)                                0
(%i4) ( cylaplacian(rh^(-n)*cos(n*p),rh,p,z), factor(%%) );
(%o4)                                0
(%i5) ( cylaplacian(rh^(-n)*sin(n*p),rh,p,z), factor(%%) );
(%o5)                                0
```

Another general solution is $\boldsymbol{\ln(\rho)}$:

```
(%i6) cylaplacian(log(rh),rh,p,z);
(%o6)                                0
```

Here is another example of the use of this Maxima function. The expression $\mathbf{u = (2/3)\,(\rho - 1/\rho)\sin(\varphi)}$ is proposed as the solution to a problem defined by: (partial differential equation: pde) $\boldsymbol{\nabla^2 u = 0}$ for $\mathbf{1 \le \rho \le 2}$, and (boundary conditions: bc) $\mathbf{u(1,\varphi) = 0}$, and $\mathbf{u(2,\varphi) = \sin(\varphi)}$. Here we use Maxima to check the three required solution properties.

```
(%i7) u : 2*(rh - 1/rh)*sin(p)/3$
(%i8) (cylaplacian(u,rh,p,z), ratsimp(%%) );
(%o8)                                0
(%i9) [subst(rh=1,u),subst(rh=2,u) ];
(%o9)                          [0, sin(p)]
```

6.9.3 Gradient $\boldsymbol{\nabla f(\rho,\varphi,z)}$

There are several ways one can work with vector calculus problems. One method (not used here) is to use symbols for a set of orthogonal unit vectors, and assume a set of properties for these unit vectors without connecting the set to any specific list basis representation. One can then construct the dot product, the cross product and the derivative operations in terms of the coefficients of these symbolic unit vectors (using **`ratcoeff`**, for example).

Alternatively (and used here), one can define 3-vectors in terms of three element lists, in which the first element of the list contains the $\mathbf{x}$ axis component of the vector, the second element of the list contains the $\mathbf{y}$ axis component, and the third element contains the $\mathbf{z}$ component.

This second method is less abstract and closer to the beginning student's experience of vectors, and provides a straightforward path which can be used with any orthonormal coordinate system.

The unit vector along the $\mathbf{x}$ axis ($\hat{\mathbf{x}}$) is represented by **`xu = [1,0,0]`** (for "x - unit vector"), the unit vector along the $\mathbf{y}$ axis ($\hat{\mathbf{y}}$) is represented by **`yu = [0,1,0]`**, and the unit vector along the $\mathbf{z}$ axis ($\hat{\mathbf{z}}$) is represented by **`zu = [0,0,1]`**.

Let's return to the batch file **`cylinder.mac`**, where we define **`lcross(u,v)`** to calculate the vector cross product $\mathbf{u \times v}$ when we are using three element lists to represent vectors.

Note that our "orthonormality checks" use the built-in "dot product" of lists provided by the period **`.`** (which is also used for non-commutative matrix multiplication). The dot product of two vectors represented by Maxima lists is obtained by placing a period between the lists. We have checked that the cartesian vectors defined are "unit vectors" (the dot product yields unity) and are also mutually orthogonal (ie., at right angles to each other) which is equivalent to the dot product of a pair of different unit vectors being zero.

This section of **cylinder.mac** begins with the code for **lcross** which looks like:

```
lcross(u,v) := (
          ( u[2]*v[3] - u[3]*v[2] )*xu +
           ( u[3]*v[1] - u[1]*v[3]  )*yu +
           ( u[1]*v[2] - u[2]*v[1] )*zu )$
```

Note, in the batch file output, using **display2d:true** (the default), how the list element numbers are displayed as subscripts.

```
(%i28)           ------------------------------------------
(%i29)                        Unit Vectors
(%i30)           ------------------------------------------
(%i31)          cross product rule when using lists for vectors
(%i32) lcross(u, v) := (u  v  - u  v ) zu + (u  v  - u  v ) yu
                        1  2    2  1         3  1    1  3
                                                    + (u  v  - u  v ) xu
                                                        2  3    3  2
(%i33)             cross product of parallel vectors is zero
(%i34)                lcross([a, b, c], [n a, n b, n c])
(%o34)                                0
(%i35)                  a function we can use with map
(%i36)                  apcr(ll) := apply('lcross, ll)
(%i37)                3d cartesian unit vectors using lists
(%i38)          (xu : [1, 0, 0], yu : [0, 1, 0], zu : [0, 0, 1])
(%i39)           orthonormality checks on cartesian unit vectors
(%i40)       [xu . xu, yu . yu, zu . zu, xu . yu, xu . zu, yu . zu]
(%o40)                       [1, 1, 1, 0, 0, 0]
(%i41)     low tech check of cross products of cartesian unit vectors
(%i42)                       lcross(xu, yu) - zu
(%o42)                            [0, 0, 0]
(%i43)                       lcross(yu, zu) - xu
(%o43)                            [0, 0, 0]
(%i44)                       lcross(zu, xu) - yu
(%o44)                            [0, 0, 0]
(%i45)          [lcross(xu, xu), lcross(yu, yu), lcross(zu, zu)]
(%o45)                  [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
(%i46)    high tech checks of cross products of cartesian unit vectors
(%i47)     map('apcr, [[xu, yu], [yu, zu], [zu, xu]]) - [zu, xu, yu]
(%o47)                  [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
(%i48)            map('apcr, [[xu, xu], [yu, yu], [zu, zu]])
(%o48)                  [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

Thus we have cartesian unit vector relations such as $\hat{\mathbf{x}} \cdot \hat{\mathbf{x}} = 1$, and $\hat{\mathbf{x}} \cdot \hat{\mathbf{y}} = 0$, and $\hat{\mathbf{x}} \times \hat{\mathbf{y}} = \hat{\mathbf{z}}$.

The unit vector $\hat{\boldsymbol{\rho}}$ along the direction of increasing ρ at the point (ρ, φ, z) is defined in terms of the cartesian unit vectors $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ via

$$\hat{\boldsymbol{\rho}} = \hat{\mathbf{x}} \cos\varphi + \hat{\mathbf{y}} \sin\varphi \tag{6.18}$$

Because the direction of $\hat{\boldsymbol{\rho}}$ depends on φ, a more explicit notation would be $\hat{\boldsymbol{\rho}}(\varphi)$, but following convention, we surpress that dependence in the following.

The unit vector $\hat{\boldsymbol{\varphi}}$ along the direction of increasing φ at the point (ρ, φ, z) is

$$\hat{\boldsymbol{\varphi}} = -\hat{\mathbf{x}} \sin\varphi + \hat{\mathbf{y}} \cos\varphi \tag{6.19}$$

Again, because the direction of $\hat{\boldsymbol{\varphi}}$ depends on φ, a more explicit notation would be $\hat{\boldsymbol{\varphi}}(\varphi)$, but following conventional use we surpress that dependence in the following. We use the symbol **rhu** ("rh-unit-vec") for $\hat{\boldsymbol{\rho}}$, and the symbol **pu** for $\hat{\boldsymbol{\varphi}}$.

```
(%i49)       cylindrical coordinate unit vectors rho-hat, phi-hat
(%i50)                    rhu : sin(p) yu + cos(p) xu
(%o50)                        [cos(p), sin(p), 0]
(%i51)                    pu : cos(p) yu - sin(p) xu
(%o51)                       [- sin(p), cos(p), 0]
(%i52)                orthonormality checks on unit vectors
(%i53) ([rhu . rhu, pu . pu, zu . zu, rhu . pu, rhu . zu, pu . zu],
                                                          trigsimp(%%))
(%o53)                        [1, 1, 1, 0, 0, 0]
(%i54)                  low tech check of cross products
(%i55)                (lcross(rhu, pu), trigsimp(%%)) - zu
(%o55)                              [0, 0, 0]
(%i56)                (lcross(pu, zu), trigsimp(%%)) - rhu
(%o56)                              [0, 0, 0]
(%i57)                (lcross(zu, rhu), trigsimp(%%)) - pu
(%o57)                              [0, 0, 0]
(%i58)                 [lcross(rhu, rhu), lcross(pu, pu)]
(%o58)                       [[0, 0, 0], [0, 0, 0]]
(%i59)                 high tech checks of cross products
(%i60) (map('apcr, [[rhu, pu], [pu, zu], [zu, rhu]]), trigsimp(%%))
                                                      - [zu, rhu, pu]
(%o60)                  [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
(%i61)                 map('apcr, [[rhu, rhu], [pu, pu]])
(%o61)                       [[0, 0, 0], [0, 0, 0]]
```

Thus we have cylindrical unit vector relations such as $\hat{\boldsymbol{\rho}} \cdot \hat{\boldsymbol{\rho}} = 1$, $\hat{\boldsymbol{\rho}} \cdot \hat{\boldsymbol{\varphi}} = 0$, and $\hat{\boldsymbol{\rho}} \times \hat{\boldsymbol{\varphi}} = \hat{\mathbf{z}}$.

The gradient of a scalar function $f(x, y, z)$ is defined by

$$\boldsymbol{\nabla} f(x, y, z) = \hat{\mathbf{x}} \frac{\partial f}{\partial x} + \hat{\mathbf{y}} \frac{\partial f}{\partial y} + \hat{\mathbf{z}} \frac{\partial f}{\partial z} \tag{6.20}$$

For a scalar function f which depends explicitly on (ρ, φ, z) and hence implicitly on (x, y), we can use the relations already established in **`cylinder.mac`** to convert the cartesian expression of the gradient of f (called here **`fgradient`**) into an expression in terms of derivatives with respect to (ρ, φ, z).

Returning to the output of **`cylinder.mac`**:

```
(%i62)          -----------------------------------------
(%i63)                  Gradient of a Scalar Function f
(%i64)          -----------------------------------------
(%i65)             cartesian def. of gradient of scalar f
(%i66)     fgradient : diff(f, z) zu + diff(f, y) yu + diff(f, x) xu
```

The $\boldsymbol{\rho}$ component of a vector $\mathbf{A}$ (at the point $(\boldsymbol{\rho}, \boldsymbol{\varphi}, \mathbf{z})$) is given by

$$A_\rho = \hat{\boldsymbol{\rho}} \cdot \mathbf{A}, \tag{6.21}$$

and the $\boldsymbol{\varphi}$ component of a vector $\mathbf{A}$ at the point $(\boldsymbol{\rho}, \boldsymbol{\varphi}, \mathbf{z})$ is given by

$$A_\varphi = \hat{\boldsymbol{\varphi}} \cdot \mathbf{A}. \tag{6.22}$$

Our batch file follows this pattern to get the (ρ, φ, z) components of the vector $\boldsymbol{\nabla} f$:

```
(%i67)               rho, phi, and z components of grad(f)
(%i68)          fgradient_rh : (rhu . fgradient, trigsimp(%%))
                                    df
(%o68)                              ---
                                    drh
```

```
(%i69)           fgradient_p : (pu . fgradient, trigsimp(%%))
                                      df
                                      --
                                      dp
(%o69)                                --
                                      rh
(%i70)           fgradient_z : (zu . fgradient, trigsimp(%%))
                                      df
(%o70)                                --
                                      dz
```

Hence we have derived

$$\boldsymbol{\nabla} f(\rho,\varphi,z) = \hat{\boldsymbol{\rho}}\,\frac{\boldsymbol{\partial} f}{\boldsymbol{\partial}\rho} + \hat{\boldsymbol{\varphi}}\,\frac{1}{\rho}\,\frac{\boldsymbol{\partial} f}{\boldsymbol{\partial}\varphi} + \hat{\mathbf{z}}\,\frac{\boldsymbol{\partial} f}{\boldsymbol{\partial} z}. \tag{6.23}$$

6.9.4 Divergence $\boldsymbol{\nabla}\cdot\mathbf{B}(\rho,\varphi,z)$

In cartesian coordinates the divergence of a three dimensional vector field $\mathbf{B}(x,y,z)$ can be calculated with the equation

$$\boldsymbol{\nabla}\cdot\mathbf{B}(x,y,z) = \frac{\partial B_x}{\partial x} + \frac{\partial B_y}{\partial y} + \frac{\partial B_z}{\partial z} \tag{6.24}$$

Consider a vector field $\mathbf{B}(\rho,\varphi,z)$ which is an explicit function of the cylindrical coordinates $\rho,\varphi,z)$ and hence an implicit function of (x,y). We will use the symbol **bvec** to represent $\mathbf{B}(\rho,\varphi,z)$, and use the symbols **bx**, **by**, and **bz** to represent the x, y, and z components of $\mathbf{B}(\rho,\varphi,z)$.

The ρ component (the component in the direction of increasing ρ with constant φ and constant z) of $\mathbf{B}(\rho,\varphi,z)$ at the point (ρ,φ,z) is given by

$$B_\rho(\rho,\varphi,z) = \hat{\boldsymbol{\rho}}\cdot\mathbf{B}(\rho,\varphi,z). \tag{6.25}$$

We use the symbol **brh** for $B_\rho(\rho,\varphi,z)$. The component of $\mathbf{B}(\rho,\varphi,z)$ at the point (ρ,φ,z) in the direction of increasing φ (with constant ρ and constant z) is given by the equation

$$B_\varphi(\rho,\varphi,z) = \hat{\boldsymbol{\varphi}}\cdot\mathbf{B}(\rho,\varphi,z). \tag{6.26}$$

We use the symbol **bp** for $B_\varphi(\rho,\varphi,z)$. Returning to the output of **cylinder.mac** batch file:

```
(%i71)          ------------------------------------------------
(%i72)                Divergence of a Vector bvec
(%i73)          ------------------------------------------------
(%i74)                   bvec : bz zu + by yu + bx xu
(%o74)                             [bx, by, bz]
(%i75)         two equations which relate cylindrical components
(%i76)                 of bvec to the cartesian components
(%i77)                     eq1 : brh = rhu . bvec
(%o77)                   brh = by sin(p) + bx cos(p)
(%i78)                      eq2 : bp = pu . bvec
(%o78)                   bp = by cos(p) - bx sin(p)
(%i79)                     invert these equations
(%i80)       sol : (linsolve([eq1, eq2], [bx, by]), trigsimp(%%))
(%o80)    [bx = brh cos(p) - bp sin(p), by = brh sin(p) + bp cos(p)]
(%i81)                    [bx, by] : map('rhs, sol)
(%i82)          tell Maxima to treat cylindrical components as
(%i83)                   explicit functions of (rh,p,z)
(%i84)                depends([brh, bp, bz], [rh, p, z])
(%o84)           [brh(rh, p, z), bp(rh, p, z), bz(rh, p, z)]
(%i85)                calculate the divergence of bvec
(%i86) bdivergence : (diff(bz, z) + diff(by, y) + diff(bx, x), trigsimp(%%),
                                                              multthru(%%))
                                dbp
                                ---
                           brh   dp    dbz   dbrh
(%o86)                     --- + --- + --- + ----
                           rh    rh    dz    drh
```

Hence we have derived the result:

$$\boldsymbol{\nabla}\cdot\mathbf{B}(\rho,\varphi,z)=\frac{1}{\rho}\frac{\partial}{\partial\rho}\left(\rho\,B_\rho\right)+\frac{1}{\rho}\frac{\partial}{\partial\varphi}B_\varphi+\frac{\partial B_z}{\partial z}, \tag{6.27}$$

in which we have used the identity

$$\frac{1}{\rho}\frac{\partial}{\partial\rho}\left(\rho\,B_\rho\right)=\frac{B_\rho}{\rho}+\frac{\partial}{\partial\rho}B_\rho \tag{6.28}$$

6.9.5 Curl $\boldsymbol{\nabla}\times\mathbf{B}(\rho,\varphi,z)$

In cartesian coordinates the curl of a three dimensional vector field $\mathbf{B}(x,y,z)$ can be calculated with the equation

$$\boldsymbol{\nabla}\times\mathbf{B}(x,y,z)=\hat{\mathbf{x}}\left(\frac{\partial B_z}{\partial y}-\frac{\partial B_y}{\partial z}\right)+\hat{\mathbf{y}}\left(\frac{\partial B_x}{\partial z}-\frac{\partial B_z}{\partial x}\right)+\hat{\mathbf{z}}\left(\frac{\partial B_y}{\partial x}-\frac{\partial B_x}{\partial y}\right) \tag{6.29}$$

Remember that we have already bound the symbols **bx** and **by** to linear combinations of **brh** and **bp**, and have told Maxima (using **depends**) to treat **brh** and **bp** as explicit functions of **(rh,p,z)**. Hence our assignment of the symbol **bcurl** in **cylinder.mac** will result in Maxima using the calculus chain rule. We can then extract the **cylindrical components** of $\mathbf{B}$ by taking the dot product of the cylindrical unit vectors with $\mathbf{B}$, just as we did to get the cylindrical components of $\boldsymbol{\nabla}f$.

Returning to the output of **cylinder.mac**:

```
(%i87)     ------------------------------------------------------
(%i88)          Cylindrical Components of Curl(bvec)
(%i89)     ------------------------------------------------------
(%i90)                  cartesian definition of curl(vec)
(%i91) bcurl : (diff(by, x) - diff(bx, y)) zu + (diff(bx, z) - diff(bz, x)) yu
                                              + (diff(bz, y) - diff(by, z)) xu
(%i92)              find cylindrical components of curl(bvec)
(%i93)       bcurl_rh : (rhu . bcurl, trigsimp(%%), multthru(%%))
                                    dbz
                                    ---
                                    dp    dbp
(%o93)                              --- - ---
                                    rh    dz
(%i94)        bcurl_p : (pu . bcurl, trigsimp(%%), multthru(%%))
                                   dbrh   dbz
(%o94)                             ---- - ---
                                    dz    drh
(%i95)        bcurl_z : (zu . bcurl, trigsimp(%%), multthru(%%))
                                  dbrh
                                  ----
                                   dp    bp   dbp
(%o95)                          - ---- + -- + ---
                                   rh    rh   drh
(%i96)      ------------------------------------------------------
```

Hence we have derived

$$\boldsymbol{\nabla}\times\mathbf{B}(\rho,\varphi,z)=\hat{\boldsymbol{\rho}}\left(\frac{1}{\rho}\frac{\partial B_z}{\partial\varphi}-\frac{\partial B_\varphi}{\partial z}\right)+\hat{\boldsymbol{\varphi}}\left(\frac{\partial B_\rho}{\partial z}-\frac{\partial B_z}{\partial\rho}\right)+\hat{\mathbf{z}}\left(\frac{1}{\rho}\frac{\partial}{\partial\rho}\left(\rho\,B_\varphi\right)-\frac{1}{\rho}\frac{\partial B_\rho}{\partial\varphi}\right) \tag{6.30}$$

in which we have used

$$\frac{B_\varphi}{\rho}+\frac{\partial B_\phi}{\partial\rho}=\frac{1}{\rho}\frac{\partial}{\partial\rho}\left(\rho\,B_\varphi\right). \tag{6.31}$$

6.10 Maxima Derivation of Vector Calculus Formulas in Spherical Polar Coordinates

The batch file **sphere.mac**, available on the author's webpage with this chapter, uses Maxima to apply the **same method** we used in the previous section, but here we derive **spherical polar coordinate expressions** for the gradient, divergence, curl and Laplacian. We will include less commentary in this section since the general approach is the same.

We consider a change of variable from cartesian coordinates (x, y, z) to spherical polar coordinates (r, θ, φ). Given (r, θ, φ), we obtain (x, y, z) from the equations $x = r\sin\theta\cos\varphi$, $y = r\sin\theta\sin\varphi$, and $z = r\cos\theta$. Given (x, y, z), we obtain (r, θ, φ) from the equations $r = \sqrt{x^2 + y^2 + z^2}$, $\cos\theta = z/\sqrt{x^2 + y^2 + z^2}$, and $\tan\varphi = y/x$.

In **sphere.mac** we use **t** to represent the angle θ and **p** to represent the angle φ (both expressed in radians). The ranges of the independent variables are $0 < r < \infty$, $0 < \theta < \pi$, and $0 \le \varphi < 2\pi$.

We first define **s3rule** as a list of replacement "rules" in the form of equations which can be used later by the **subst** function employed by **s3sub**. To get automatic simplification of **r/abs(r)** and **sin(t)/abs(sin(t))** we use the **assume** function.

Here is the beginning of the **sphere.mac** batch file output when used with Maxima ver. 5.21.1:

```
(%i1) batch("sphere.mac")$
read and interpret file: #pc:/work5/sphere.mac
(%i2)       ------------  sphere.mac   ----------------------------
(%i3)        spherical polar coordinates (r,theta,phi ) = (r,t,p)
(%i4)                  replacement rules x,y,z to r,t,p
(%i5)  s3rule : [x = r sin(t) cos(p), y = r sin(t) sin(p), z = r cos(t)]
(%i6)         s3sub(expr) := (subst(s3rule, expr), trigsimp(%%))
(%i7)                       assume(r > 0, sin(t) > 0)
                                        2    2    2
(%i8)                       rxyz : sqrt(z  + y  + x )
(%i9)      partial derivatives of r, theta, and phi wrt x, y, and z
(%i10)                  drdx : (diff(rxyz, x), s3sub(%%))
(%o10)                            cos(p) sin(t)
(%i11)                  drdy : (diff(rxyz, y), s3sub(%%))
(%o11)                            sin(p) sin(t)
(%i12)                  drdz : (diff(rxyz, z), s3sub(%%))
(%o12)                                cos(t)
                                        z
(%i13)              dtdx : (diff(acos(----), x), s3sub(%%))
                                       rxyz
                                  cos(p) cos(t)
(%o13)                            -------------
                                        r
                                        z
(%i14)              dtdy : (diff(acos(----), y), s3sub(%%))
                                       rxyz
                                  sin(p) cos(t)
(%o14)                            -------------
                                        r
                                        z
(%i15)              dtdz : (diff(acos(----), z), s3sub(%%))
                                       rxyz
                                     sin(t)
(%o15)                            - ------
                                        r
                                        y
```

```
(%i16)               dpdx : (diff(atan(-), x), s3sub(%%))
                                          x
                                       sin(p)
(%o16)                              - --------
                                       r sin(t)
                                          y
(%i17)               dpdy : (diff(atan(-), y), s3sub(%%))
                                          x
                                       cos(p)
(%o17)                              --------
                                       r sin(t)
(%i18)               tell Maxima r=r(x,y,z), t = t(x,y,z),
(%i19)                           and p = p(x,y)
(%i20)   (gradef(r, x, drdx), gradef(r, y, drdy), gradef(r, z, drdz))
(%i21)   (gradef(t, x, dtdx), gradef(t, y, dtdy), gradef(t, z, dtdz))
(%i22)              (gradef(p, x, dpdx), gradef(p, y, dpdy))
(%i23)                    depends([r, t], [x, y, z])
(%i24)                        depends(p, [x, y])
```

The batch file next calculates the Laplacian of a scalar function f.

```
(%i25)         ------------------------------------------------
(%i26)           Laplacian of a scalar function f
(%i27)         ------------------------------------------------
(%i28)           tell Maxima to treat scalar function f as an
(%i29)                     explicit function of (r,t,p)
(%i30)                       depends(f, [r, t, p])
(%o30)                          [f(r, t, p)]
(%i31)      calculate the Laplacian of the scalar function f(r,t,p)
(%i32)                    using the cartesian definition
(%i33) (diff(f, z, 2) + diff(f, y, 2) + diff(f, x, 2), trigsimp(%%),
                                                  scanmap('multthru, %%))
                                    2                2
                                   d f              d f
                 df                ---            df   ---
                 -- cos(t)           2          2 --     2    2
                 dt                dp             dr   dt    d f
(%o33)           --------- + ---------- + ---- + --- + ---
                  2           2    2        r      2      2
                 r  sin(t)   r  sin (t)           r     dr
(%i34)                          grind(%)
'diff(f,t,1)*cos(t)/(r^2*sin(t))+'diff(f,p,2)/(r^2*sin(t)^2)+2*'diff(f,r,1)/r
                               +'diff(f,t,2)/r^2+'diff(f,r,2)$
```

Hence the form of the Laplacian of a scalar field in spherical polar coordinates:

$$\boldsymbol{\nabla}^2 f(r,\theta,\varphi) = \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\,\frac{\partial f}{\partial r}\right) + \frac{1}{r^2\,\sin\theta}\,\frac{\partial}{\partial\,\theta}\left(\sin\theta\,\frac{\partial f}{\partial\theta}\right) + \frac{1}{r^2\,\sin^2\theta}\frac{\partial^2 f}{\partial\,\varphi^2} \tag{6.32}$$

in which we have used the identities

$$\frac{\partial^2 f}{\partial\,r^2} + \frac{2}{r}\,\frac{\partial f}{\partial\,r} = \frac{1}{r^2}\frac{\partial}{\partial\,r}\left(r^2\,\frac{\partial f}{\partial\,r}\right) \tag{6.33}$$

and

$$\frac{\partial^2 f}{\partial\,\theta^2} + \frac{\cos\theta}{\sin\theta}\,\frac{\partial f}{\partial\,\theta} = \frac{1}{\sin\theta}\,\frac{\partial}{\partial\,\theta}\left(\sin\theta\,\frac{\partial f}{\partial\,\theta}\right) \tag{6.34}$$

We need to avoid $r = 0$ and $\sin\theta = 0$. The latter condition means avoiding $\theta = 0$ and $\theta = \pi$.

Gradient of a Scalar Field

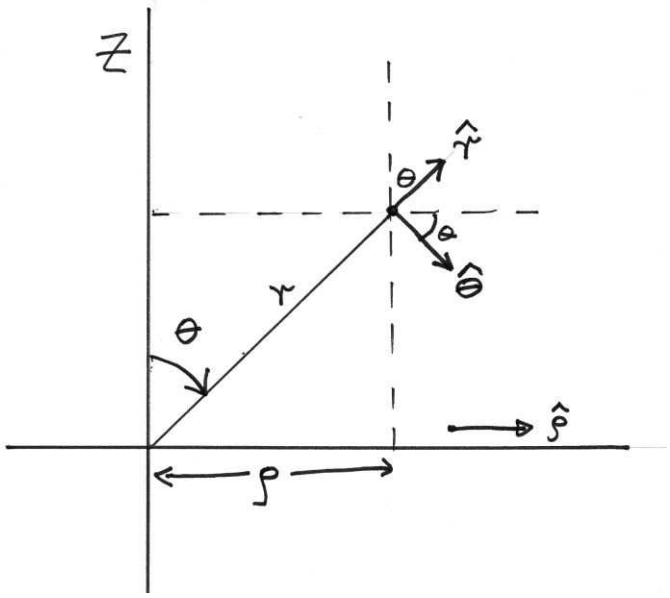

Figure 20: $\mathbf{r}$ and $\boldsymbol{\theta}$ unit vectors

The batch file sphere.mac introduces the cartesian and spherical polar unit vectors corresponding to the coordinates (r, θ, φ). The unit vector $\hat{\boldsymbol{\varphi}}$ is the same as in cylindrical coordinates:

$$\hat{\boldsymbol{\varphi}} = -\hat{\mathbf{x}} \sin\varphi + \hat{\mathbf{y}} \cos\varphi \tag{6.35}$$

We can express $\hat{\mathbf{r}}$ and $\hat{\boldsymbol{\theta}}$ in terms of the cylindrical $\hat{\boldsymbol{\rho}}$ and $\hat{\mathbf{z}}$ (from the figure above):

$$\hat{\mathbf{r}} = \hat{\mathbf{z}} \cos\theta + \hat{\boldsymbol{\rho}} \sin\theta \tag{6.36}$$

and

$$\hat{\boldsymbol{\theta}} = -\hat{\mathbf{z}} \sin\theta + \hat{\boldsymbol{\rho}} \cos\theta. \tag{6.37}$$

Hence we have

$$\hat{\mathbf{r}} = \hat{\mathbf{x}} \sin\theta \cos\varphi + \hat{\mathbf{y}} \sin\theta \sin\varphi + \hat{\mathbf{z}} \cos\theta \tag{6.38}$$

and

$$\hat{\boldsymbol{\theta}} = \hat{\mathbf{x}} \cos\theta \cos\varphi + \hat{\mathbf{y}} \cos\theta \sin\varphi - \hat{\mathbf{z}} \sin\theta \tag{6.39}$$

Maxima has already been told to treat f as an explicit function of **(r,t,p)** which we are using for (r, θ, φ). The vector components of $\boldsymbol{\nabla} f$ are isolated by using the dot product of the unit vectors with $\boldsymbol{\nabla} f$. For example,

$$(\boldsymbol{\nabla} f)_r = \hat{\mathbf{r}} \cdot \boldsymbol{\nabla} f \tag{6.40}$$

```
(%i35)          -----------------------------------------
(%i36)                      Unit Vectors
(%i37)          -----------------------------------------
(%i38)                   cartesian unit vectors
(%i39)       (xu : [1, 0, 0], yu : [0, 1, 0], zu : [0, 0, 1])
(%i40)          spherical polar coordinate unit vectors
(%i41)     ru : cos(t) zu + sin(t) sin(p) yu + sin(t) cos(p) xu
(%o41)              [cos(p) sin(t), sin(p) sin(t), cos(t)]
(%i42)    tu : - sin(t) zu + cos(t) sin(p) yu + cos(t) cos(p) xu
(%o42)            [cos(p) cos(t), sin(p) cos(t), - sin(t)]
(%i43)                  pu : cos(p) yu - sin(p) xu
(%o43)                     [- sin(p), cos(p), 0]
```

```
(%i44)          ------------------------------------------
(%i45)                  Gradient of a Scalar Function f
(%i46)          ------------------------------------------
(%i47)             cartesian def. of gradient of scalar f
(%i48)     fgradient : diff(f, z) zu + diff(f, y) yu + diff(f, x) xu
(%i49)             r, theta, and phi components of grad(f)
(%i50)          fgradient_r : (ru . fgradient, trigsimp(%%))
                                  df
(%o50)                            --
                                  dr
(%i51)          fgradient_t : (tu . fgradient, trigsimp(%%))
                                  df
                                  --
                                  dt
(%o51)                            --
                                  r
(%i52)          fgradient_p : (pu . fgradient, trigsimp(%%))
                                  df
                                  --
                                  dp
(%o52)                           --------
                                 r sin(t)
```

Thus we have the gradient of a scalar field in spherical polar coordinates:

$$\boldsymbol{\nabla} f(r,\theta,\varphi) = \hat{\mathbf{r}}\,\frac{\partial f}{\partial r} + \hat{\boldsymbol{\theta}}\,\frac{1}{r}\,\frac{\partial f}{\partial \theta} + \hat{\boldsymbol{\varphi}}\,\frac{1}{r\,\sin\theta}\,\frac{\partial f}{\partial \varphi} \tag{6.41}$$

Divergence of a Vector Field

The path here is the same as in the cylindrical case. For example, we define the spherical polar components of the vector $\mathbf{B}$ via

$$B_r = \hat{\mathbf{r}}\cdot\mathbf{B}, B_\theta = \hat{\boldsymbol{\theta}}\cdot\mathbf{B}, B_\varphi = \hat{\boldsymbol{\varphi}}\cdot\mathbf{B}. \tag{6.42}$$

```
(%i53)          ------------------------------------------
(%i54)                Divergence of a Vector bvec
(%i55)          ------------------------------------------
(%i56)                   bvec : bz zu + by yu + bx xu
(%o56)                           [bx, by, bz]
(%i57)      three equations which relate spherical polar components
(%i58)                   of bvec to the cartesian components
(%i59)                        eq1 : br = ru . bvec
(%o59)       br = by sin(p) sin(t) + bx cos(p) sin(t) + bz cos(t)
(%i60)                        eq2 : bt = tu . bvec
(%o60)      bt = - bz sin(t) + by sin(p) cos(t) + bx cos(p) cos(t)
(%i61)                        eq3 : bp = pu . bvec
(%o61)                     bp = by cos(p) - bx sin(p)
(%i62)                       invert these equations
(%i63)   sol : (linsolve([eq1, eq2, eq3], [bx, by, bz]), trigsimp(%%))
(%o63) [bx = br cos(p) sin(t) + bt cos(p) cos(t) - bp sin(p),
by = br sin(p) sin(t) + bt sin(p) cos(t) + bp cos(p),
bz = br cos(t) - bt sin(t)]
(%i64)                    [bx, by, bz] : map('rhs, sol)
(%i65)        tell Maxima to treat spherical polar components as
(%i66)                     explicit functions of (r,t,p)
(%i67)                 depends([br, bt, bp], [r, t, p])
```

```
(%i68)              divergence of bvec
(%i69) bdivergence : (diff(bz, z) + diff(by, y) + diff(bx, x), trigsimp(%%),
                                                    scanmap('multthru, %%))
                                  dbp      dbt
                                  ---      ---
                  bt cos(t)       dp       dt     2 br    dbr
(%o69)            --------- + -------- + --- + ---- + ---
                  r sin(t)     r sin(t)     r      r      dr
```

Hence we have the spherical polar coordinate version of the divergence of a vector field:

$$\boldsymbol{\nabla}\cdot\mathbf{B}(r,\theta,\varphi)=\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\,B_r\right)+\frac{1}{r\,\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\,B_\theta\right)+\frac{1}{r\,\sin\theta}\frac{\partial\,B_\varphi}{\partial\,\varphi} \tag{6.43}$$

using the identities

$$\frac{2}{r}\,B_r+\frac{\partial\,B_r}{\partial\,r}=\frac{1}{r^2}\frac{\partial}{\partial\,r}\left(r^2\,B_r\right) \tag{6.44}$$

and

$$\frac{\cos\theta}{\sin\theta}\,B_\theta+\frac{\partial\,B_\theta}{\partial\,\theta}=\frac{1}{\sin\theta}\frac{\partial}{\partial\,\theta}\left(\sin\theta\,B_\theta\right) \tag{6.45}$$

Curl of a Vector Field

Again the path is essentially the same as in the cylindrical case:

```
(%i70)     ------------------------------------------------------
(%i71)        Spherical Polar Components of Curl(bvec)
(%i72)     ------------------------------------------------------
(%i73)                  cartesian curl(bvec) definition
(%i74) bcurl : (diff(by, x) - diff(bx, y)) zu + (diff(bx, z) - diff(bz, x)) yu
                                              + (diff(bz, y) - diff(by, z)) xu
(%i75)              spherical polar components of curl(bvec)
(%i76)   bcurl_r : (ru . bcurl, trigsimp(%%), scanmap('multthru, %%))
                                       dbt      dbp
                                       ---      ---
                         bp cos(t)     dp       dt
(%o76)                   --------- - -------- + ---
                         r sin(t)    r sin(t)    r
(%i77)   bcurl_t : (tu . bcurl, trigsimp(%%), scanmap('multthru, %%))
                                  dbr
                                  ---
                                  dp       bp   dbp
(%o77)                          -------- - -- - ---
                                r sin(t)   r    dr
(%i78)   bcurl_p : (pu . bcurl, trigsimp(%%), scanmap('multthru, %%))
                                       dbr
                                       ---
                                  bt   dt     dbt
(%o78)                            -- - --- + ---
                                  r     r     dr
(%i79)      ------------------------------------------------------
```

which produces the spherical polar coordinate version of the curl of a vector field:

$$(\boldsymbol{\nabla}\times\mathbf{B})_r=\frac{1}{r\,\sin\theta}\left[\frac{\partial}{\partial\theta}\left(\sin\theta\,B_\varphi\right)-\frac{\partial\,B_\theta}{\partial\,\varphi}\right] \tag{6.46}$$

$$(\boldsymbol{\nabla}\times\mathbf{B})_\theta=\frac{1}{r}\left[\frac{1}{\sin\theta}\frac{\partial\,B_r}{\partial\varphi}-\frac{\partial}{\partial\,r}\left(r\,B_\varphi\right)\right] \tag{6.47}$$

$$(\boldsymbol{\nabla}\times\mathbf{B})_\varphi=\frac{1}{r}\left[\frac{\partial}{\partial r}\left(r\,B_\theta\right)-\frac{\partial\,B_r}{\partial\,\theta}\right] \tag{6.48}$$

in which we have used Eq.(6.45) with B_θ replaced by B_φ and also Eq.(6.31) with ρ replaced by r.