Introduction to Maple for Differential Geometry

This worksheet introduces some basic Maple commands for studying the geometry of curves. Also, the beginnings of Maple programming are given. The following commands install the packages for plotting and using tools from linear algebra.

```
> with(plots):with(LinearAlgebra):
```

One thing to keep in mind is that, to get help on a topic, you can always type ? topic instead of going to the toolbar. For instance, to get help on vectors, type ? Vector. Vectors are defined by the following. Note that you can get either row or column vectors.

```
> A:=<a|b|c>; B:=<d|e|f>;C:=<g,h,i>;
```

The dot product and cross product are calculated as follows.

```
> DotProduct(A,B,conjugate=false);
> CrossProduct(A,B);
```

Another way to get the cross product is by forming a determinant with the standard vectors ***i, j, k*** and the two vectors in question. Note that this is just a device to help us remember the formula. There is no true determinant mixing these quantities. But when we calculate cross products by hand, instead of trying to remember formulas, we can just remember this trick of taking a "determinant".

```
> collect(Determinant(Matrix(3,3,[[i,j,k],[a,b,c],[d,e,f]])),{i,j,k});
```

A matrix is defined by either of the following methods. The first lists rows and the second columns.

```
> M:=Matrix(3,3,[[1,2,3],[0,3,0],[5,7,9]]);      MM:=Matrix(3,3,[<1,0,5>,<2,3,7>,
  <3,0,9>]);
```

The matrix can be applied to a vector by

```
> M.Transpose(A);        MM.C;
```

Note that the row vector *A* must be made into a column vector by taking transpose. Now let's make some curves. The first is called the Witch of Agnesi.

```
> witch:=<2*tan(t)|2*cos(t)^2>;
```

There is a funny Maple syntax that has to be used to plot planar parametrized curves. Namely, the parameter bounds have to be put inside the square brackets with the parametrization. Also, the parametrization must be in a list, not in a vector.

```
> plot([2*tan(t),2*cos(t)^2,t=-1.3..1.3],scaling=constrained,axes=framed,color=
  red);
> plot([witch[1],witch[2],t=-1.3..1.3],scaling=constrained,axes=framed,color=
  red);
```

Now let's plot a parametrized circle of radius 2.

```
> circ:=<2*cos(t)|2*sin(t)>;
```
Plots can be "saved" and displayed all at once. The following does this. Note that you must end a named (and therefore, saved) plot with a colon, not a semi-colon.
```
> circplot:=plot([circ[1],circ[2],t=0..2*Pi],color=black):
> display(circplot);
> witchplot:=plot([2*tan(t),2*cos(t)^2,t=-1.3..1.3],scaling=constrained,axes=
  framed,color=red,thickness=2):
> display({circplot,witchplot},scaling=constrained,axes=framed);
```
You can define a general type of curve and then specialize using the "subs" command. (You can also use the "eval" command as we show.)
```
> ellipse:=<a*cos(t)|b*sin(t)>;
> ell:=subs({a=4,b=1},ellipse);      ell2:=eval(ellipse,{a=4,b=1});
```
Let's plot this ellipse two ways. In the first, Maple tries to use the whole plotting screen. In the second, Maple gives the true representation.
```
> plot([ell[1],ell[2],t=0..2*Pi],color=blue,scaling=unconstrained);
> plot([ell[1],ell[2],t=0..2*Pi],color=blue,scaling=constrained);
```
The following shows how curves can be changed by applying linear transformations. In particular, the matrix below is a rotation matrix. The following curve is called an astroid.
```
> astr:=<cos(t)^3,sin(t)^3>;
> plot([astr[1],astr[2],t=0..2*Pi],scaling=constrained,color=green);
> R:=Matrix([[cos(Pi/4),-sin(Pi/4)],[sin(Pi/4),cos(Pi/4)]]);
```
We now apply the matrix to the vector to get a rotated astroid.
```
> astr2:=R.<astr[1],astr[2]>;
> plot([astr2[1],astr2[2],t=0..2*Pi],scaling=constrained,color=magenta,view=[-1.
  .1,-1..1]);
```
The next curve is very important in the history of Mathematics as applied to Classical Mechanics. It is called a cycloid.
```
> cycloid:=<t+sin(t),1+cos(t)>;
> plot([cycloid[1],cycloid[2],t=Pi..3*Pi],scaling=constrained,color=orange);
```
Now let's create some curves in 3-space. Curves are plotted by "spacecurve", but if you want to see them better (especially when plotted together with surfaces), use "tubeplot". Once the plot is created, if you click on the picture and hold the mouse button down, you can rotate the plot. Options such as "orientation" allow you to plot a curve from any fixed viewing angle.
```
> hel:=<a*cos(t)|a*sin(t)|b*t>;
> hel1:=subs({a=4,b=2},hel);
> spacecurve(convert(hel1,list),t=0..5*Pi,scaling=constrained,thickness=3,
```

```
color=black,axes=framed,orientation=[39,50,1],tickmarks=[3,3,3]);
> tubeplot(convert(hel1,list),t=0..5*Pi,radius=1.2,scaling=constrained,shading=
XY,lightmodel=light3,style=patchnogrid,axes=framed,orientation=[39,50,1]);
> tubeplot(convert(hel1,list),t=0..5*Pi,radius=1.2,scaling=constrained,shading=
zhue,lightmodel=light3,style=patchnogrid,axes=framed,orientation=[39,50,1])
;
```

The following commands display Viviani's curve; it is the intersection of a cylinder and a sphere. The sphere and cylinder are also displayed below.

```
> viv:=<a*(1+cos(t))|a*sin(t)|a*2*sin(t/2)>;
> viv1:=spacecurve(convert(subs(a=1,viv),list),t=-Pi..Pi,color=black):
> viv2:=tubeplot(convert(subs(a=1,viv),list),t=-2*Pi..2*Pi,radius=0.05,color=
black):
> sphere:=plot3d([2*cos(u)*cos(v),2*sin(u)*cos(v),2*sin(v)],u=0..2*Pi,v=-Pi/2..
Pi/2,shading=XY,lightmodel=light2):
> cyl:=plot3d([cos(u)+1,sin(u),v],u=0..2*Pi,v=-2..2,shading=XYZ,lightmodel=
light1):
> display({viv2,sphere,cyl},scaling=constrained,orientation=[-27,69]);
```

To see the curve better, we can use the Maple option "transparency".

```
> display({viv2,sphere,cyl},scaling=constrained,orientation=[-27,69],
transparency=0.75);
```

To plot surfaces, we use the plot3d command as you just saw above. There are various options that give different coloring, lighting, orientation and scaling, for instance. Here are a few examples. The first is a (elliptic) paraboloid $z = x^2 + y^2$ plotted in cartesian coordinates and then as a parametrized surface. Which is better?

```
> plot3d(x^2+y^2,x=-1..1,y=-1..1,scaling=constrained,shading=xy,orientation=
[44,74,11]);
> plot3d([u*cos(v),u*sin(v),u^2],u=0..1.5,v=0..2*Pi,scaling=constrained,
shading=xy,orientation=[44,74,11]);
> plot3d([u*cos(v),u*sin(v),u^2],u=0..1.5,v=0..2*Pi,scaling=constrained,
shading=xyz,orientation=[44,74,11]);
> plot3d([u*cos(v),u*sin(v),u^2],u=0..1.5,v=0..2*Pi,scaling=constrained,
shading=zhue,orientation=[44,74,11]);
```

Here is a hyperbolic paraboloid $z = x^2 - y^2$. For the parametrization of the elliptic paraboloid above, we used the basic fact that $\cos^2(\theta) + \sin^2(\theta) = 1$. That doesn't work here because we have a minus sign. But we know that hyperbolic trig functions satisfy a similar equation: $\cosh^2(t) - \sinh^2(t) = 1$. Check that the following parametrization works.

```
> plot3d([u*cosh(v),u*sinh(v),u^2],u=-1..1,v=-1..1,scaling=constrained,
  shading=zhue,lightmodel=light3);
```

Notice the problem. We can only get the part of the surface above the x-y plane. Here is a case where the simpler parametrization $\boldsymbol{x}(u, v) = \left(u, v, u^2 - v^2\right)$ works better. This is just the usual graph of the function $f(u, v) = u^2 - v^2$. Such a parametrization is called a ***Monge*** parametrization in honor of one of the founders of Differential Geometry, Gaspard Monge. Here is the plot.

```
> plot3d(x^2-y^2,x=-2..2,y=-2..2,scaling=unconstrained,shading=zhue,
  orientation=[65,68,4]);
```

Let's create the torus we discussed in class. But let's do this in a way where we can introduce the radii R and r as inputs to a program. Maple's version of programming is called a procedure. It always starts the same way so that Maple knows you are writing a procedure. Also, when you enter the "code" for the procedure, you **cannot hit ENTER** at the end of a line. Maple will try to execute the procedure if you do, but you haven't finished yet! Instead hit "**SHIFT+ ENTER**" and this will move the cursor to the next line without trying to execute. Begin by giving a name to your procedure (using := to define the name) and listing what the inputs will be. Then write the commands you want to be executed. Finally, end your procedure by "**END**". Hit **ENTER** now. This reads your procedure into Maple.

```
> torusplot:=proc(R,r)
  plot3d([(R+r*cos(u))*cos(v),(R+r*cos(u))*sin(v),r*sin(u)],u=0..2*Pi,v=0..2*Pi,
  scaling=constrained,shading=zhue,lightmodel=light3,orientation=[45,68,2]);
  end:
```

In order to "do" your procedure to create a torus, you write the following (where you can choose any inputs you want for R and r that make sense).

```
> torusplot(5,2);
> torusplot(6,2);
> torusplot(10,2);
```

The following is another example of a Maple procedure. The procedure draws the great circle arc between two points A and B on Earth and gives the distance between the points (assuming the radius of the Earth to be 3970 miles). A and B are given by longitude and latitude coordinates. For example A=[100,25] means that the point A has longitude 100 degrees and latitude 25 degrees. We assume the Earth is a sphere with parametrization $[R\cos(u)\cos(v), R\sin(u)\cos(v), R\sin(v)]$, R being the radius of the Earth. Try to understand the inputs and how they are used.

```
> great_circle:=proc(place1,place2,theta,phi)
```

```
local point1,point2,angle,A,B,sphere,circ,alpha,unitcp,alp1,alp2,p1,p2;
A:=map(evalf,[Pi/180*place1[1],Pi/180*place1[2]]);
B:=map(evalf,[Pi/180*place2[1],Pi/180*place2[2]]);
point1:=<cos(A[1])*cos(A[2])|sin(A[1])*cos(A[2])|sin(A[2])>;
point2:=<cos(B[1])*cos(B[2])|sin(B[1])*cos(B[2])|sin(B[2])>;
angle:=evalf(arccos(DotProduct(point1,point2,conjugate=false)));
print(`Distance is`,3970*angle,`miles`);
unitcp:=1/sin(angle)*CrossProduct(point1,point2);
alp1:=ScalarMultiply(point1,cos(t));
alp2:=ScalarMultiply(CrossProduct(unitcp,point1),sin(t));
alpha:=alp1+alp2;
sphere:=plot3d([cos(u)*cos(v),sin(u)*cos(v),sin(v)],u=0..2*Pi,
v=-Pi/2..Pi/2,shading=XY,lightmodel=light2,grid=[25,19]):
circ:=tubeplot(convert(alpha,list),t=0..angle,radius=0.01,
color=navy):
p1:=pointplot3d(convert(point1,list),color=cyan,symbol=solidcircle,
symbolsize=24):
p2:=pointplot3d(convert(point2,list),color=green,symbol=solidbox,
symbolsize=24):
display({sphere,circ,p1,p2},scaling=constrained,orientation=
[theta,phi]);
end:
```

Here are other longs and lats (in that order): New York (-73.94, 40.67), Chicago (-87.68, 41.84), Paris (2.30, 48.83), Moscow (37.62, 55.75), Atlanta (-84.42, 33.76), Toronto (-79.60, 43.67), Seoul (127.05,37.52), Beijing (116.35,39.90), Sydney (151.30, -33.90), New Delhi (77.22, 28.90), Rio de Janeiro (-42.70, -22.49). How about New York to Moscow? London to Paris? Cleveland to Sydney?

```
> great_circle([-73.94,40.67],[37.62,55.75],-21,60);
> great_circle([-81.68,41.48],[151.30,-33.9],-156,75);
```

Look up some longs and lats and plot some other great circle arcs.