2D and 3D Measurements

Produced by Bruce Carpenter Published by Math Everywhere, Inc.
www.matheverywhere.com

VC.02 Perpendicularity Basics

B.1) The cross product X×Y of two 3D vectors is perpendicular to both X and Y

Take two vectors X and Y from the same dimension.
Testing for perpendicularity by checking whether
X . Y = 0
is quick and easy.
In three dimensions another product comes to the front. This product is also related to perpendicularity.
To calculate the cross product
X×Y for X = {1, 2, 3} and Y = {−2, 0, 7},
you make a matrix with
{i, j, k}
in the top (horizontal) row, with X in the middle (horizontal) row, and with Y in the bottom row:

```
X = {1, 2, 3};
Y = {-2, 0, 7};
Clear[i, j, k]
MatrixForm[{{i, j, k}, X, Y}]
```

$$\begin{pmatrix} i & j & k \\ 1 & 2 & 3 \\ -2 & 0 & 7 \end{pmatrix}$$

Next, take the determinant of this matrix:

```
Det[{{i, j, k}, X, Y}]
```

```
14 i - 13 j + 4 k
```

To arrive at the cross product X×Y, you replace
i by {1, 0, 0},
replace
j by {0, 1, 0}
and replace
k by {0, 0, 1}:

```
XcrossY =
 Det[{{i, j, k}, X, Y}] /. {i → {1, 0, 0}, j → {0, 1, 0}, k → {0, 0, 1}}
```

```
{14, -13, 4}
```

The dot product is a number. The cross product is a vector.

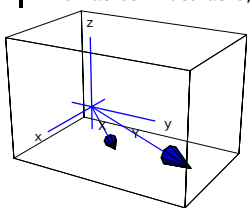
Mathematica also knows how to calculate the cross product X×Y:

```
XcrossY = Cross[X, Y]
```

```
{14, -13, 4}
```

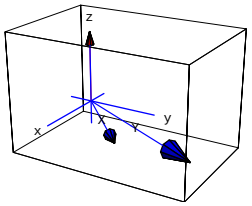
Here are two new vectors X and Y in 3D:

```
X = {1, 1, 0};
Y = {1, 2, 0};
threedims = Axes3D[1, 0.2];
XandY =
 Show[threedims, Arrow[X, Tail → {0, 0, 0}], Arrow[Y, Tail → {0, 0, 0}],
  Graphics3D[Text["X", X/2]], Graphics3D[Text["Y", Y/2]],
  BoxRatios → Automatic, ViewPoint → CMView, PlotRange → All];
```



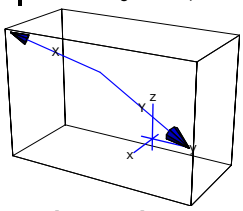
Now throw in the cross product X×Y:

```
Clear[i, j, k]
XcrossY = Cross[X, Y];
Show[XandY, Arrow[XcrossY, Tail → {0, 0, 0}, VectorColor → Red],
 ViewPoint → CMView, BoxRatios → Automatic, PlotRange → All];
```



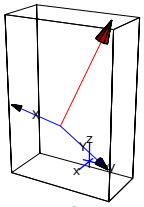
Golly, X×Y appears to be perpendicular to X and to Y.
Try it again with new vectors and a new place to hang the tails:

```
X = {1, -2, 1};
Y = {1, 3, -1};
newtail = {1, -1, 2};
XandY = Show[threedims, Arrow[X, Tail → newtail],
  Arrow[Y, Tail → newtail], Graphics3D[Text["X", newtail + X/2]],
  Graphics3D[Text["Y", newtail + Y/2]], ViewPoint → CMView,
  PlotRange → All, BoxRatios → Automatic];
```



Now throw in the cross product X×Y:

```
Clear[i, j, k]
XcrossY = Cross[X, Y];
Show[XandY, Arrow[XcrossY, Tail → newtail, VectorColor → Red],
 ViewPoint → CMView, BoxRatios → Automatic, PlotRange → All];
```



Again, the cross product X×Y appears to be perpendicular to both X and Y.
Rerun this with new vectors X and Y and new tails.

□B.1.a.i)

For the record, explain why it is that when you take any two three-dimensional vectors X and Y, then their cross product
X×Y
is perpendicular to both X and Y.

□Answer:

Clear the vectors and test with the dot product:

```
Clear[X, x, Y, y, i, j, k]
X = {x[1], x[2], x[3]};
Y = {y[1], y[2], y[3]};
XcrossY = Cross[X, Y];
Expand[X . XcrossY]
```

```
0
```

Ah-ha!

No matter what X is, the vector X×Y is perpendicular to X.

```
Expand[Y . XcrossY]
```

```
0
```

No matter what Y is, X×Y is perpendicular to Y.

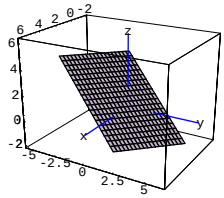
That's all there is to it.

Ah, the joy of automated algebra!

B.2) Planes in 3D

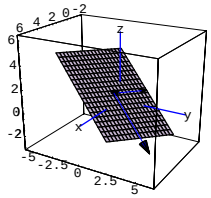
Just as a line is determined by a point and one direction vector, a plane in three dimensions is determined by a point and TWO direction vectors.
Here's how you plot part of the plane determined by the following point and two direction vectors:

```
point = {1, 0, 1};
vector1 = {-1, 1, 0};
vector2 = {0.3, 1, -1};

Clear[planeplotter, u, v]
planeplotter[u_, v_] = point + u vector1 + v vector2;
{ulow, uhigh} = {-2, 2};
{vlow, vhigh} = {-3, 3};

plane = ParametricPlot3D[Evaluate[planeplotter[u, v]],
  {u, ulow, uhigh}, {v, vlow, vhigh}, DisplayFunction → Identity];
threedims = Axes3D[6, 0.2];

planeplot = Show[threedims, plane, Axes → Automatic,
  PlotRange → All, ViewPoint → CMView, BoxRatios → Automatic,
  DisplayFunction → $DisplayFunction];
```



Look at the plane together with the point and the two vectors that determine this particular plane:

```
scalefactor = 2;
Show[planeplot, Arrow[ vector1, Tail → point, VectorColor → Blue,
  ScaleFactor → scalefactor], Arrow[scalefactor vector2,
  Tail → point, VectorColor → Blue, ScaleFactor → scalefactor],
 Graphics3D[{PointSize[0.04], Point[point]}], BoxRatios → Automatic];
```

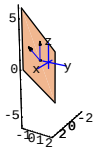


There you are.
The plane is what you get by sticking the tails of vector1 and vector2 at the point and nailing a flat piece of plywood onto the braces made from those vectors. The function

planeplotter[u, v] = point + u vector1 + v vector2

plots the plane by moving to the given point and then adding on multiples of the two determining vectors.
Try it for a different point and a different pair of vectors:

```
point = {0, -1, 0};
vector1 = {0, 0, 1};
vector2 = {1, 0, 1};

Clear[planeplotter, u, v]
planeplotter[u_, v_] = point + u vector1 + v vector2;
{ulow, uhigh} = {-3, 3};
{vlow, vhigh} = {-3, 3};

plane =
 ParametricPlot3D[Evaluate[planeplotter[u, v]], {u, ulow, uhigh},
  {v, vlow, vhigh}, PlotPoints → {2, 2}, DisplayFunction → Identity];
threedims = Axes3D[2, 0.2];
scalefactor = 2;
Show[plane, threedims, Arrow[ vector1, Tail → point,
  VectorColor → Blue, ScaleFactor → scalefactor], Arrow[ vector2,
  Tail → point, VectorColor → Blue, ScaleFactor → scalefactor],
 Graphics3D[{PointSize[0.04], Point[point]}],
 Axes → Automatic, PlotRange → All, ViewPoint → CMView, Boxed → False,
 BoxRatios → Automatic, DisplayFunction → $DisplayFunction];
```



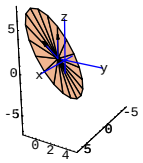
□B.2.a)

Plot an elliptical piece of the plane plotted above.

□Answer:

That's not too hard; you just modify the plotting function as follows:

```
point = {0, -1, 0};
vector1 = {0, 0, 1};
vector2 = {1, 0, 1};
Clear[planeplotter, r, t]
planeplotter[r_, t_] = point + 4 r Cos[t] vector1 + 6 r Sin[t] vector2;
{rlow, rhigh} = {0, 1};
{tlow, thigh} = {0, 2 π};
plane = ParametricPlot3D[
  Evaluate[planeplotter[r, t]], {r, rlow, rhigh}, {t, tlow, thigh},
  PlotPoints → {2, Automatic}, DisplayFunction → Identity];
scalefactor = 3;
threedims = Axes3D[5, 0.2];
elliptical = Show[plane, threedims, Arrow[vector1, Tail → point,
   VectorColor → Blue, ScaleFactor → scalefactor], Arrow[vector2,
   Tail → point, VectorColor → Blue, ScaleFactor → scalefactor],
  Graphics3D[{PointSize[0.04], Point[point]}], Axes → Automatic,
  PlotRange → All, ViewPoint → CMView, Boxed → False,
  BoxRatios → Automatic, DisplayFunction → $DisplayFunction];
```



No sweat.

□B.2.b) Normal vector to a plane

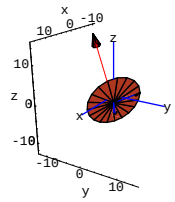
A normal vector for a given plane is a vector perpendicular to that plane.
How do you come up with a normal (perpendicular) vector for a plane determined by a point and two direction vectors?

□Answer:

To get a normal vector for a plane, just take the cross product of two direction (non-parallel) vectors.

Try it; you'll like it:

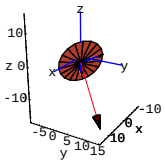
```
point = {0, 0, -2};
vector1 = {-1, 2, 1};
vector2 = {-2, -1, -2};
Clear[i, j, k]
normal = Cross[vector1, vector2];
Clear[x, y, z, r, t]
Clear[planeplotter, r, t]
planeplotter[r_, t_] = point + 3 r Cos[t] vector1 + 5 r Sin[t] vector2;
{rlow, rhigh} = {0, 1};
{tlow, thigh} = {0, 2 π};
plane = ParametricPlot3D[
  Evaluate[planeplotter[r, t]], {r, rlow, rhigh}, {t, tlow, thigh},
  PlotPoints → {2, Automatic}, DisplayFunction → Identity];
scalefactor = 3;
threedims = Axes3D[14, 0.8];

Show[
 plane, threedims, Arrow[ vector1, Tail → point, VectorColor → Blue,
  ScaleFactor → scalefactor], Arrow[ vector2, Tail → point,
  VectorColor → Blue, ScaleFactor → scalefactor], Arrow[normal,
  Tail → point, VectorColor → Red, ScaleFactor → scalefactor],
 Graphics3D[{PointSize[0.04], Point[point]}],
 Axes → Automatic, PlotRange → All, ViewPoint → CMView,
 Boxed → False, BoxRatios → Automatic, AxesLabel → {"x", "y", "z"},
 DisplayFunction → $DisplayFunction];
```



Lookin' fairly good.

If you want the normal to point the other way, just multiply the normal calculated above by −1:

```
Show[plane, threedims,
 Arrow[ vector1, Tail → point, VectorColor → Blue,
  ScaleFactor → scalefactor], Arrow[ vector2, Tail → point,
  VectorColor → Blue, ScaleFactor → scalefactor], Arrow[ (-normal),
  Tail → point, VectorColor → Red, ScaleFactor → scalefactor],
 Graphics3D[{PointSize[0.04], Point[point]}],
 Axes → Automatic, PlotRange → All, ViewPoint → CMView,
 Boxed → False, BoxRatios → Automatic, AxesLabel → {"x", "y", "z"},
 DisplayFunction → $DisplayFunction];
```



Lookin' great.

□B.2.c) xyz-equation of a plane

How do you get the xyz-equation for a plane specified by a given point and two direction vectors?

□Answer:

You get it from the given point and the normal vector.

To see what's involved, go with the plane determined by the following point and direction vectors:

```
Clear[x, y, z]
point = {1, 2, 3};
vector1 = {2, 4, -1.2};
vector2 = {-1, 1.5, 0};
Clear[i, j, k]
normal = Cross[vector1, vector2]
```

```
{1.8, 1.2, 7.}
```

Saying that {x, y, z} is on this plane is the same as saying that when you stick the tail of the normal at the given point {1, 2, 3}, then the normal is perpendicular to the vector with tail at {1, 2, 3} and tip at {x, y, z}.

So saying that {x, y, z} is on this plane is the same as saying that

$$(\{x, y, z\} - \text{point}) \cdot \text{normal} = 0:$$

```
xyzeqn = ({x, y, z} - point) . normal == 0
```

```
1.8 (-1 + x) + 1.2 (-2 + y) + 7. (-3 + z) == 0
```

Notice how the given point {1, 2, 3} and the normal vector {1.8, 1.2, 7} are conveniently displayed in the plane equation.

You can solve this equation for z:

```
zsolved = Solve[xyzeqn, z]
```

```
{{z → -0.142857 (-21. + 1.8 (-1. + x) + 1.2 (-2. + y))}}
```

And you can use this to plot the part of the plane over any rectangle in the xy-plane you like.

```
Clear[f]
f[x_, y_] = z /. zsolved[[1]]
```

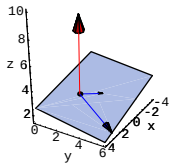
```
-0.142857 (-21. + 1.8 (-1. + x) + 1.2 (-2. + y))
```

Here comes the plot of the part of this plane over the rectangle

$$-4 \le x \le 4 \text{ and } 0 \le y \le 6$$

in the xy-plane:

```
plane = Plot3D[f[x, y], {x, -4, 4},
   {y, 0, 6}, PlotPoints → {2, 2}, DisplayFunction → Identity];
Show[plane, Arrow[vector1, Tail → point, VectorColor → Blue],
 Arrow[vector2, Tail → point, VectorColor → Blue],
 Arrow[normal, Tail → point, VectorColor → Red],
 Graphics3D[{PointSize[0.04], Point[point]}],
 Axes → Automatic, PlotRange → All, AxesLabel → {"x", "y", "z"},
 ViewPoint → CMView, Boxed → False, BoxRatios → Automatic,
 DisplayFunction → $DisplayFunction];
```



Bingo.

□B.2.d)

A normal vector and a point also determine a plane.
To see what the plane is, put the tail of the normal vector at the point.
The plane is what you get by putting a flat plywood sheet at the tail of the normal vector and aligning the plywood sheet to be perpendicular to the normal vector.
In fact, if the normal vector and the point are related to the xyz-equation for the plane by:

```
Clear[R, S, T, a, b, c, x, y, z]
normal = {R, S, T};
point = {a, b, c};
xyzequation = ({x, y, z} - {a, b, c}) . normal == 0
```

```
R (-a + x) + S (-b + y) + T (-c + z) == 0
```

Explain where this comes from, and use it to plot the plane passing through the point {4, 2, 1} with normal vector {2, −1, 1.5}.

□Answer:

Take another look:

```
normal = {R, S, T};
point = {a, b, c};
xyzequation = ({x, y, z} - point) . normal == 0
```

```
R (-a + x) + S (-b + y) + T (-c + z) == 0
```

This comes from the fact that saying that

→ {x, y, z} is on the plane passing through the given point {a, b, c} with normal vector {R, S, T}

is the same as saying that

→ the vector running from the given point {a, b, c} to {x, y, z} is perpendicular to the normal vector {R, S, T}.

This is the same as saying:

```
xyzequation = ({x, y, z} - point) . normal == 0
```

```
R (-a + x) + S (-b + y) + T (-c + z) == 0
```

To plot the plane passing through the point {4, 2, 1} with normal vector {2, −0.8, 1.5}, go with:

```
normal = {2, -0.8, 1.5};
point = {4, 2, 1};
xyzequation = ({x, y, z} - point) . normal == 0
```

```
2 (-4 + x) - 0.8 (-2 + y) + 1.5 (-1 + z) == 0
```

Here comes the plot:

```
zsolved = Solve[xyzequation, z]
```

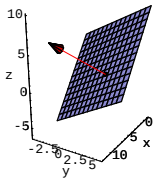
```
{{z → -0.666667 (-1.5 + 2. (-4. + x) - 0.8 (-2. + y))}}
```

```
Clear[f]
f[x_, y_] = z /. zsolved[[1]]
```

```
-0.666667 (-1.5 + 2. (-4. + x) - 0.8 (-2. + y))
```

```
plane =
 Plot3D[f[x, y], {x, -1, 7}, {y, -4, 6}, DisplayFunction → Identity];
scalefactor = 4;
Show[plane, Arrow[normal,
   Tail → point, VectorColor → Red, ScaleFactor → scalefactor ],
 Graphics3D[{PointSize[0.04], Point[point]}],
```

```
Axes → Automatic, AxesLabel → {"x", "y", "z"}, PlotRange → All,
ViewPoint → CMView, Boxed → False, BoxRatios → Automatic,
DisplayFunction → $DisplayFunction];
```



The flat plywood sheet is perpendicular to the normal vector at the point:

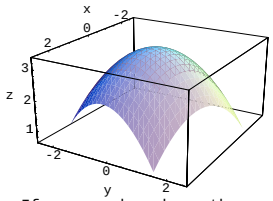
```
point
```
```
{4, 2, 1}
```

And that's all there is to it.

B.3) Normal vectors for curved surfaces in 3D

Here is a curved surface in three dimensions given in parametric form:

```
Clear[x, y, z, u, v]
x[u_, v_] = u + v/3;
y[u_, v_] = v - u/4;
z[u_, v_] = 1/3 (10 - u^2 - v^2);
{ulow, uhigh} = {-2, 2};
{vlow, vhigh} = {-2, 2};
surf = ParametricPlot3D[Evaluate[{x[u, v], y[u, v], z[u, v]}],
  {u, ulow, uhigh}, {v, vlow, vhigh}, DisplayFunction → Identity];
surface = Insert[surf, EdgeForm[], {1, 1}];
surfaceplot = Show[surface, Axes → Automatic,
  PlotRange → All, ViewPoint → CMView, BoxRatios → Automatic,
  AxesLabel → {"x", "y", "z"}, DisplayFunction → $DisplayFunction];
```



If you wonder where the usual little bricks went, study the code.

□B.3.a)

Take a point on the surface and say how to build a vector that is perpendicular to the surface at that point.
Show off your work with a sample plot.

□Answer:

Any point on the surface can be described by

{x[u, v], y[u, v], z[u, v]}

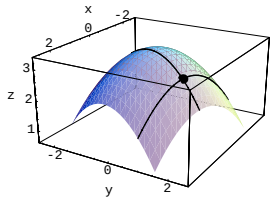
for specific choices of u and v.

Here's the point you get by setting u = 0.2 and v = 1.3:

```
uset = 0.2;
vset = 1.3;
Clear[surfacepoint]
surfacepoint[u_, v_] = {x[u, v], y[u, v], z[u, v]};
surfacepoint[uset, vset]
```
```
{0.633333, 1.25, 2.75667}
```

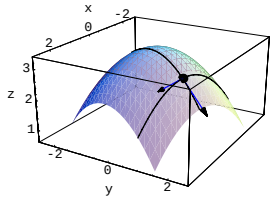
To build the perpendicular vector at this point on the surface, look at the point and two curves in the surface that meet at this point:

```
Clear[ucurve, vcurve]
ucurve[u_] = surfacepoint[u, vset];
vcurve[v_] = surfacepoint[uset, v];
ucurveplot = ParametricPlot3D[
  ucurve[u], {u, ulow, uhigh}, DisplayFunction → Identity];
vcurveplot = ParametricPlot3D[
  vcurve[v], {v, vlow, vhigh}, DisplayFunction → Identity];
pointplot =
 Graphics3D[{PointSize[0.04], Point[surfacepoint[uset, vset]]}];
firstlook = Show[surfaceplot, ucurveplot, vcurveplot, pointplot,
  DisplayFunction → $DisplayFunction];
```



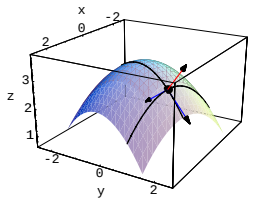
Now throw in the tangent vectors to each curve at the point in question:

```
utanvector = ucurve'[uset];
vtanvector = vcurve'[vset];
tangents = {Arrow[utanvector, Tail → surfacepoint[uset, vset]],
  Arrow[vtanvector, Tail → surfacepoint[uset, vset]]};
secondlook = Show[firstlook, tangents];
```



To build a vector perpendicular to the surface at the point in question, all you gotta do is take the cross product of the two tangent vectors:

```
Clear[i, j, k]
normal = Cross[utanvector, vtanvector];
proudlook = Show[secondlook,
  Arrow[normal, Tail → surfacepoint[uset, vset], VectorColor → Red],
  BoxRatios → Automatic];
```

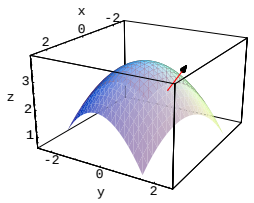


There it is!

Proud and perpendicular.

Most folks don't bother to show the curves and the tangents:

```
cleanplot = Show[surfaceplot,
  Arrow[normal, Tail → surfacepoint[uset, vset], VectorColor → Red]];
```



Math folks like to call this the normal vector at the point at the tail of the vector.

Play with this by rerunning with your own choices of set points

{u, v} = {uset, vset}.

□B.3.b)

What can you use this normal vector for?

□Answer:

You can use it to see what happens when you bounce light off a surface. You'll have a chance to give this a try later.

□B.3.c.i)

Say, all that talk about what you do to get the normal vector was pretty windy.
Can you give some concise code that efficiently produces the normal vector at a desired point without all the wind?

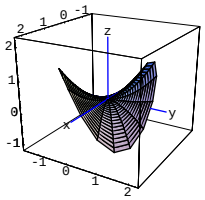
□Answer:

Be happy to.

To get the concise code that will work on any surface given in parametric form, just review what went on in part a) and try it out on a plot of the surface

$z = f[x, y] = -x\,y$:

```
Clear[f, x, y, z, u, v]
f[x_, y_] = -x y;
x[u_, v_] = u Cos[v];
y[u_, v_] = u Sin[v];
z[u_, v_] = f[x[u, v], y[u, v]];
{ulow, uhigh} = {0, 1.5};
{vlow, vhigh} = {0, 2 π};
surface = ParametricPlot3D[Evaluate[{x[u, v], y[u, v], z[u, v]}],
  {u, ulow, uhigh}, {v, vlow, vhigh}, DisplayFunction → Identity];
threedims = Axes3D[2, 0.2];
surfaceplot =
 Show[threedims, surface, Axes → Automatic, PlotRange → All,
  ViewPoint → CMView, DisplayFunction → $DisplayFunction];
```



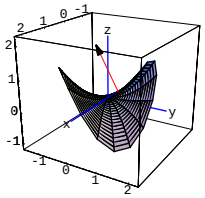
Here's a fairly quick way of throwing in the normal vector at the following point:

```
{uset, vset} = {0.5, 15 π / 16};
Clear[surfacepoint]
surfacepoint[u_, v_] = {x[u, v], y[u, v], z[u, v]};
N[surfacepoint[uset, vset]]
```

```
{-0.490393, 0.0975452, 0.0478354}
```

Here comes the normal:

```
Clear[ucurve, vcurve]
ucurve[u_] = surfacepoint[u, vset];
vcurve[v_] = surfacepoint[uset, v];
utanvector = ucurve'[uset];
vtanvector = vcurve'[vset];
Clear[i, j, k]
normal = Cross[utanvector, vtanvector];
scalefactor = 3;
onenormal =
 Show[surfaceplot, Arrow[normal, Tail → surfacepoint[uset, vset],
   VectorColor → Red, ScaleFactor → scalefactor ],
  ViewPoint → CMView];
```



Just fine, thank you.

VC.02 Perpendicularity Tutorials

T.1) True scale plots via the options `AspectRatio→Automatic` and `BoxRatios→Automatic`

□T.1.a)

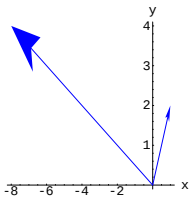
Look at this:

```
X = {1, 2};
Y = {-8, 4};
X . Y
```

```
0
```

This tells you that the two vectors are perpendicular.
Now look at this:

```
Show[Arrow[X], Arrow[Y], Axes → True, AxesLabel → {"x", "y"},
 AspectRatio → 1];
```



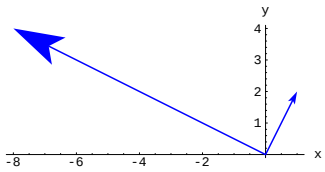
Although you knew in advance that these vectors are perpendicular, they didn't plot out perpendicularly.
What gives?

□Answer:

The two vectors weren't plotted in true scale. To guarantee a true scale plot in 2D graphics, use the plotting option AspectRatio → Automatic. This way you get the same scale on both axes.

Try it out:

```
Show[Arrow[X], Arrow[Y], Axes → True, AxesLabel → {"x", "y"},
 AspectRatio → Automatic];
```



Much better.

In 2D plots, if you don't use AspectRatio → Automatic, then actual perpendicularity can be obscured.

In 3D plots, if you don't use BoxRatios → Automatic, then actual perpendicularity can be obscured.

Moral:

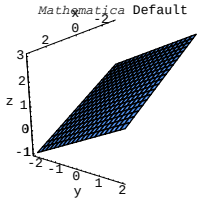
When you are studying anything related to perpendicularity, you need to plot in true scale. You are tempting fate if you don't use the options AspectRatio → Automatic or BoxRatios → Automatic as the situation demands.

T.2) Flatness and plotting

□T.2.a)

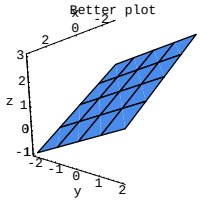
Here are three ways of plotting the same piece of the plane $\frac{x}{3} - \frac{y}{2} + z = 1$:

```
Clear[x, y]
defaultplot = ParametricPlot3D[{x, y, 1 - x/3 + y/2}, {x, -3, 3},
  {y, -2, 2}, ViewPoint → CMView, PlotRange → All, Boxed → False,
  PlotLabel → "Mathematica Default", AxesLabel → {"x", "y", "z"}];
```



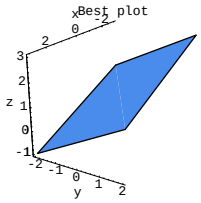
That took a while, and the plot is covered with little bricks.
Try this one:

```
betterplot = ParametricPlot3D[{x, y, 1 - x/3 + y/2},
  {x, -3, 3}, {y, -2, 2}, PlotPoints → {5, 5}, ViewPoint → CMView,
  PlotRange → All, Boxed → False, PlotLabel → "Better plot",
  AxesLabel → {"x", "y", "z"}];
```



That ran a lot faster than the default plot.
The reason: Fewer bricks. The reason: PlotPoints → {5, 5}.
Now try:

```
bestplot = ParametricPlot3D[{x, y, 1 - x/3 + y/2}, {x, -3, 3}, {y, -2, 2},
  PlotPoints → {2, 2}, ViewPoint → CMView, PlotRange → All,
  Boxed → False, PlotLabel → "Best plot", AxesLabel → {"x", "y", "z"}];
```



That ran lightning fast. And it looks really fine.
The reason: NO BRICKS. The reason: PlotPoints → {2, 2}.
Explain what's going on.

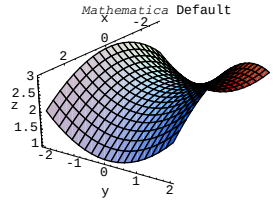
□ Answer:

The default plot is a good plot to try when the surfaces bend.

But when the surfaces are flat, you can help Mathematica by telling it to evaluate the functions only at the corners, and letting it string up one clean, flat surface.

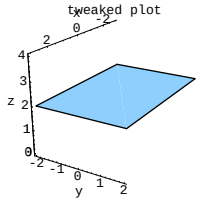
Here is a situation in which the default plot is very good:

```
Clear[x, y]
defaultplot = ParametricPlot3D[{x, y, 2 - (x/3)^2 + (y/2)^2}, {x, -3, 3},
  {y, -2, 2}, ViewPoint → CMView, PlotRange → All, Boxed → False,
  PlotLabel → "Mathematica Default", AxesLabel → {"x", "y", "z"}];
```



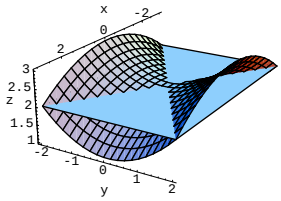
If you tinker with this, the way you tweaked the plot of the plane, then disaster strikes:

```
Clear[x, y]
tweakedplot = ParametricPlot3D[{x, y, 2 - (x/3)^2 + (y/2)^2},
  {x, -3, 3}, {y, -2, 2}, PlotPoints → {2, 2}, ViewPoint → CMView,
  PlotRange → All, Boxed → False, PlotLabel → "tweaked plot",
  AxesLabel → {"x", "y", "z"}];
```



This plot is not even close to reality.

```
Show[defaultplot, tweakedplot, ViewPoint → CMView, PlotRange → All,
 PlotLabel → None, Boxed → False, AxesLabel → {"x", "y", "z"}];
```



This tweaked plot is on the money only at the corners.

The reason: The actual surface you are plotting is not flat.

□ T.2.b)

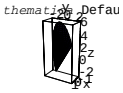
What other situations benefit from this type of tweaking?

□ Answer:

Anytime flatness is around.

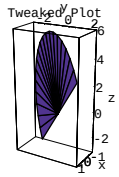
Look at this:

```
vector1 = {1, 2, 3};
vector2 = {-1, -2, 5};
Clear[r, t]
defaultplot = ParametricPlot3D[r Cos[t] vector1 + r Sin[t] vector2,
  {r, 0, 1}, {t, 0, π}, ViewPoint → CMView, AxesLabel → {"x", "y", "z"},
  PlotLabel → "Mathematica Default"];
```



There is flatness around in the r variable. Try this and look at the PlotPoints setting:

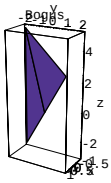
```
tweakedplot = ParametricPlot3D[r Cos[t] vector1 + r Sin[t] vector2,
  {r, 0, 1}, {t, 0, π}, PlotPoints → {2, Automatic}, ViewPoint → CMView,
  AxesLabel → {"x", "y", "z"}, PlotLabel → "Tweaked Plot"];
```



Nice.

But tinkering with t does not pay off:

```
ParametricPlot3D[r Cos[t] vector1 + r Sin[t] vector2,
 {r, 0, 1}, {t, 0, π}, PlotPoints → {2, 3}, ViewPoint → CMView,
 AxesLabel → {"x", "y", "z"}, PlotLabel → "Bogus"];
```



The reason:

With respect to r, the plot is flat. But the plot is not flat with respect to t.

□T.2.c)

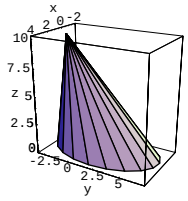
Plot, in true scale, the cone whose base sits on the ellipse

$(\frac{x-1}{3})^2 + (\frac{y-2}{5})^2 = 1$

in the xy-plane and whose top is at {1, −2, 7}.

□Answer:

```
top = {1, -2, 10};
Clear[baseplotter, coneplotter, s, t]
baseplotter[t_] = {1, 2, 0} + {3 Cos[t], 5 Sin[t], 0};
coneplotter[s_, t_] = baseplotter[t] + s (top - baseplotter[t]);

ParametricPlot3D[coneplotter[s, t], {s, 0, 1},
 {t, 0, 2 π}, PlotPoints → {2, Automatic}, ViewPoint → CMView,
 BoxRatios → Automatic, AxesLabel → {"x", "y", "z"}];
```



T.3) Unit vectors and perpendicularity:

Plotting curves on planes and a new, easy way of calculating the cross product.

A unit vector is just a vector whose length is 1. Any vector X can be converted to a unit vector unless all its slots are zero.
To wit:

```
X = {-4, 2};
unitX = X / Sqrt[X . X]
```

$\{-\frac{2}{\sqrt{5}}, \frac{1}{\sqrt{5}}\}$

Check:

```
unitX . unitX
```

1

and

```
X = {-4, 2, 8};
unitX = X / Sqrt[X . X]
```

$\{-\frac{2}{\sqrt{21}}, \frac{1}{\sqrt{21}}, \frac{4}{\sqrt{21}}\}$

Check:

```
unitX . unitX
```

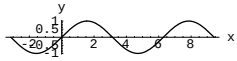
1

Unit vectors are the handiest thing since zip-lock bags. Why?
Wait and see.

□T.3.a)

Here is a true scale plot of y = Sin[x] on the xy-plane:

```
Clear[x]
ParametricPlot[{x, Sin[x]}, {x, -π, 3 π}, AspectRatio → Automatic,
 AxesLabel → {"x", "y"}];
```



Plot a duplicate copy of this curve in true scale on the plane through {1, 1, −2} determined by the direction vectors {0.3, −1, −0.2} and {0, 0.4, −1}.

□Answer:

```
point = {1, 1, 2};
vector1 = {0.3, 1, -0.2};
vector2 = {0, 0.4, -1};
Clear[i, j, k]
normal = Cross[vector1, vector2]
```

{-0.92, 0.3, 0.12}

When you think about it for a minute, you see that the same plane is also determined by the given point and the vectors

unitvector1 and unitnewvector2

calculated as follows:

```
unitvector1 = vector1 / Sqrt[vector1 . vector1]
```

{0.282216, 0.940721, -0.188144}

```
Clear[i, j, k]
newvector2 = Cross[vector1, normal];
unitnewvector2 = newvector2 / Sqrt[newvector2 . newvector2]
```

{0.173656, 0.142783, 0.974401}

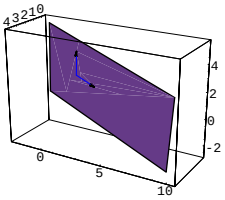
The great thing about unitvector1 and unitnewvector2 is that they are unit vectors and they are perpendicular:

```
Chop[unitvector1 . unitnewvector2]
```

0

As perpendicular unit vectors, they define axes for plotting on the plane:

```
Clear[x, y, z, s, t]
{x[s_, t_], y[s_, t_], z[s_, t_]} =
 point + s unitvector1 + t unitnewvector2;
{slow, shigh} = {-π, 3 π};
{tlow, thigh} = {-3, 3};

plane = ParametricPlot3D[
  Evaluate[{x[s, t], y[s, t], z[s, t]}], {s, slow, shigh},
  {t, tlow, thigh}, PlotPoints → {2, 2}, DisplayFunction → Identity];

planeplot = Show[plane, Arrow[2 unitvector1, Tail → point],
  Arrow[2 unitnewvector2, Tail → point], Axes → Automatic,
  PlotRange → All, ViewPoint → CMView, BoxRatios → Automatic,
  DisplayFunction → $DisplayFunction];
```

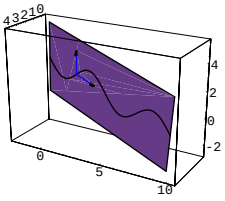


Here is a true scale duplicate copy of the curve

$y = \text{Sin}[x] \text{ for } -\pi \le x \le 3\pi$

plotted right on this plane.

```
sineplot =
 ParametricPlot3D[point + s unitvector1 + Sin[s] unitnewvector2,
  {s, -π, 3 π}, DisplayFunction → Identity];
Show[planeplot, sineplot, DisplayFunction → $DisplayFunction];
```

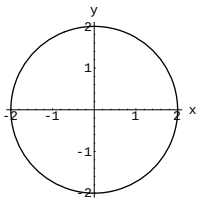


Keen.

□T.3.b) A new way of calculating the cross product.

Here is a true scale plot of the circle $x^2 + y^2 = 4$ on the xy-plane:

```
Clear[t]
ParametricPlot[2 {Cos[t], Sin[t]}, {t, 0, 2 π}, AxesLabel → {"x", "y"}];
```



Plot a duplicate copy of this circle on the plane through {1, 2, −1} determined by the direction vectors {1, 0, 2} and {−2, 1, 0}. Center the circle at {1, 2, −1}.

□ **Answer:**

```
point = {1, 2, -1};
vector1 = {1, 0, 2};
vector2 = {-2, 1, 0};
normal = Cross[vector1, vector2]
```

```
{-2, -4, 1}
```

```
Notice the shorthand command for calculating the cross product.
                Feel free use it from now on.
```

When you think about it for a minute, you see that the same plane is also determined by the given point and the vectors

unitvector1 and unitnewvector2

calculated as follows:

```
unitvector1 = vector1 / Sqrt[vector1 . vector1]
```

$$\left\{\frac{1}{\sqrt{5}}, 0, \frac{2}{\sqrt{5}}\right\}$$

```
newvector2 = Cross[vector1, normal];
unitnewvector2 = newvector2 / Sqrt[newvector2 . newvector2]
```

$$\left\{\frac{8}{\sqrt{105}}, -\sqrt{\frac{5}{21}}, -\frac{4}{\sqrt{105}}\right\}$$

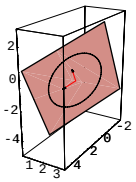
The great thing about unitvector1 and unitnewvector2 is that they are unit vectors and they are perpendicular:

```
unitvector1 . unitnewvector2
```

```
0
```

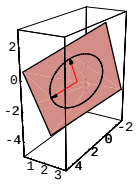
As perpendicular unit vectors, they define axes for plotting on the plane, and allow you to plot the circle where you want it.

```
Clear[circleplotter, t];
circleplotter[t_] =
 point + 2 Cos[t] unitvector1 + 2 Sin[t] unitnewvector2;
{tlow, thigh} = {0, 2 π};
circle = ParametricPlot3D[Evaluate[circleplotter[t]],
  {t, tlow, thigh}, DisplayFunction → Identity];
Clear[planeplotter, r, s];
planeplotter[r_, s_] = point + r unitvector1 + s unitnewvector2;
{rlow, rhigh} = {-3, 3};
{slow, shigh} = {-3, 3};

plane =
 ParametricPlot3D[Evaluate[planeplotter[r, s]], {r, rlow, rhigh},
  {s, slow, shigh}, PlotPoints → {2, 2}, DisplayFunction → Identity];
Show[
 plane, circle, Arrow[unitvector1, Tail → point, VectorColor → Red],
 Arrow[unitnewvector2, Tail → point, VectorColor → Red],
 Axes → Automatic, PlotRange → All, ViewPoint → CMView,
 BoxRatios → Automatic, DisplayFunction → $DisplayFunction];
```



You can make the vectors touch the circle:

```
Show[plane, circle, Arrow[unitvector1, Tail → point, VectorColor → Red,
  ScaleFactor → 2], Arrow[ unitnewvector2, Tail → point,
  VectorColor → Red, ScaleFactor → 2], Axes → Automatic,
 PlotRange → All, ViewPoint → CMView, BoxRatios → Automatic,
 DisplayFunction → $DisplayFunction];
```



The reason this worked:

The vectors

unitvector1 and unitnewvector2

are perpendicular unit vectors.

If you had used the original vectors

vector1 and vector2,

you would not have gotten a circle.

T.4) Unit vectors and perpendicularity:

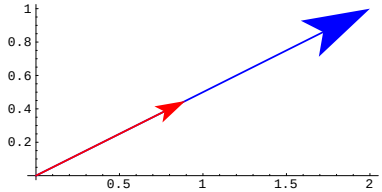
Main unit normals, binormals, tubes, horns, and corrugations

Unit vectors are just vectors whose lengths are 1.
Unit vectors are useful because they convey the idea of direction only.
You can take any vector and make it into a unit vector by dividing it by its length:

```
X = {2, 1};
unitX = X / Sqrt[X . X]
```

$$\left\{\frac{2}{\sqrt{5}}, \frac{1}{\sqrt{5}}\right\}$$

```
Show[Arrow[X], Arrow[unitX, VectorColor → Red], Axes → Automatic];
```



The unit vector points in the same direction as X, but its length is 1:

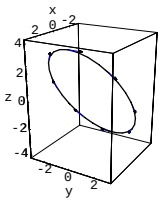
```
Sqrt[unitX . unitX]
```

```
1
```

□T.4.a.i)

Here is a curve in three dimensions with some of its unit tangents plotted in true scale:

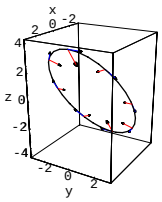
```
Clear[P, x, y, z, t]
x[t_] = 3 Cos[t];
y[t_] = 3 Sin[t];
z[t_] = 4 Cos[t];
P[t_] = {x[t], y[t], z[t]};

curve = ParametricPlot3D[Evaluate[P[t]],
  {t, 0, 2 π}, PlotPoints → 30, DisplayFunction → Identity];

Clear[unittan]
unittan[t_] = D[P[t], t] / Sqrt[D[P[t], t] . D[P[t], t]];

unittanvectors = Table[
  Arrow[unittan[t], Tail → P[t], VectorColor → Blue], {t, 0, 2 π, π/4}];
curveandtans = Show[curve, unittanvectors,
  ViewPoint → CMView, PlotRange → All, BoxRatios → Automatic,
  AxesLabel → {"x", "y", "z"}, DisplayFunction → $DisplayFunction];
```



Add to the plot the vectors D[unittan[t], t] with tails at the same places as the tails of the unit tangents.
Describe what you see.

□ **Answer:**

```
Clear[newvector]
newvector[t_] = D[unittan[t], t];
newvectors = Table[
  Arrow[newvector[t], Tail → P[t], VectorColor → Red], {t, 0, 2 π, π/4}];
Show[curveandtans, newvectors];
```



They look perpendicular to the curve!

Check this out:

```
Table[Chop[N[unittan[t] . newvector[t]]], {t, 0, 2 π, π/4}]
```

```
{0, 0, 0, 0, 0, 0, 0, 0, 0}
```

They are perpendicular to the curve.

□T.4.a.ii) The main normal for a 3D curve

Is this outcome peculiar to this curve or does it happen for all other curves too?

□ **Answer:**

Try it for a general (cleared) curve in three dimensions:

```
Clear[x, y, z, t, P, unittan, newvector]
P[t_] = {x[t], y[t], z[t]};
unittan[t_] = D[P[t], t] / √(D[P[t], t] . D[P[t], t]);
newvector[t_] = D[unittan[t], t];
Simplify[unittan[t] . newvector[t]]
```

```
0
```

No matter what curve you have, the vector D[unittan[t], t] is perpendicular to the curve at P[t]

Most folks make this into a unit vector and call the result the main unit normal vector for the curve.

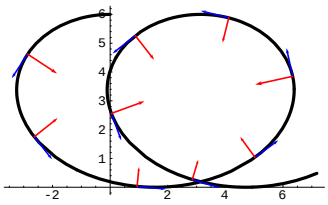
□T.4.a.iii) The main normal for a 2D curve

Does this work in two dimensions as well?

□ **Answer:**

Try it and see:

```
Clear[P, t]
P[t_] = {t/2 - 4 Sin[t], 3 + 3 Cos[t]};
curve = ParametricPlot[Evaluate[P[t]], {t, 0, 10},
  PlotStyle → Thickness[0.01], DisplayFunction → Identity];
Clear[unittan]
unittan[t_] = P'[t] / √(P'[t] . P'[t]);
unittanvectors = Table[Arrow[unittan[t], Tail → P[t]], {t, 1, 9}];
Clear[mainnormal]
mainnormal[t_] = Simplify[D[unittan[t], t]];
mainnormalvectors = Table[
  Arrow[mainnormal[t], Tail → P[t], VectorColor → Red], {t, 1, 9}];
Show[curve, unittanvectors, mainnormalvectors,
 AspectRatio → Automatic, DisplayFunction → $DisplayFunction];
```



Works like a charm in two dimensions.

When you're hot, you're hot.

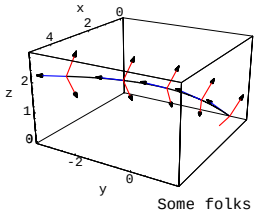
□T.4.b) The binormal for making tubes and horns

Given a curve P[t] = {x[t], y[t], z[t]}:
→ The unit tangent
unittan[t] = $\frac{D[P[t],t]}{\sqrt{D[P[t],t].D[P[t],t]}}$
is tangent to the curve at P[t].
→ The main normal
mainnorm[t] = D[unittan[t], t]
is perpendicular to the curve at P[t].
→ The main unit normal
mainunitnorm[t] = $\frac{D[unittan[t],t]}{\sqrt{D[unittan[t],t].D[unittan[t],t]}}$
is also perpendicular to the curve at P[t].
You can construct another normal vector by putting
binormal[t] = unittan[t]×mainunitnorm[t].
This second normal vector always turns out to be a unit vector, for reasons you will learn when you give it a try.
Take a look:

The N[]'s are included in the specifications for the main unit normal and the binormal to make the plotting instructions below run as fast as possible.

```
Clear[P, x, y, z, t, unittan, mainunitnormal, binormal]
x[t_] = t^2;
y[t_] = 1 - 2 t;
z[t_] = t;
P[t_] = {x[t], y[t], z[t]};
curve = ParametricPlot3D[
  Evaluate[P[t]], {t, 0, 2}, DisplayFunction → Identity];
```

```
unittan[t_] = P'[t] / √(P'[t] . P'[t]);
unittanvectors = Table[Arrow[unittan[t], Tail → P[t]], {t, 0, 2, 0.5}];
mainunitnormal[t_] = N[unittan'[t] / √(Expand[unittan'[t] . unittan'[t]])];
mainnormalvectors = Table[Arrow[mainunitnormal[t],
   Tail → P[t], VectorColor → Red], {t, 0, 2, 0.5}];
binormal[t_] = N[Cross[unittan[t], mainunitnormal[t]]];
binormalvectors = Table[
  Arrow[binormal[t], Tail → P[t], VectorColor → Red], {t, 0, 2, 0.5}];

everything =
 Show[curve, unittanvectors, mainnormalvectors, binormalvectors,
  ViewPoint → CMView, PlotRange → All, BoxRatios → Automatic,
  AxesLabel → {"x", "y", "z"}, DisplayFunction → $DisplayFunction];
```



Some folks call these moving frames.

Neato.
What's this stuff good for?

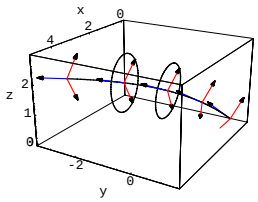
□ **Answer:**

You can put in a couple of circles that

→ are centered on the curve, and

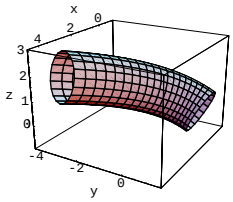
→ lie in planes that cut the curve perpendicularly:

```
radius = 1;
circle1 = ParametricPlot3D[N[P[1.0] +
    radius Cos[s] mainunitnormal[1.0] + radius Sin[s] binormal[1.0]],
  {s, 0, 2 π}, DisplayFunction → Identity];
circle2 = ParametricPlot3D[N[P[1.5] +
    radius Cos[s] mainunitnormal[1.5] + radius Sin[s] binormal[1.5]],
  {s, 0, 2 π}, DisplayFunction → Identity];
Show[everything, circle1, circle2];
```



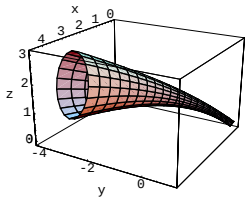
But why settle for just a couple of circles when you can go for the whole tube composed of all such circles?

```
This might take a while.
ParametricPlot3D[Evaluate[
  P[t] + radius Cos[s] mainunitnormal[t] + radius Sin[s] binormal[t]],
 {t, 0, 2}, {s, 0, 2 π}, ViewPoint → CMView, BoxRatios → Automatic,
 AxesLabel → {"x", "y", "z"}];
```



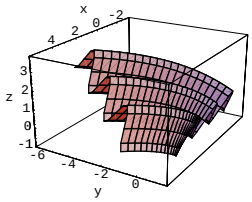
Or a horn:

```
Clear[radius]
radius[t_] = (t/2)^2 + 0.1;
ParametricPlot3D[Evaluate[P[t] +
   radius[t] Cos[s] mainunitnormal[t] + radius[t] Sin[s] binormal[t]],
 {t, 0, 2}, {s, 0, 2 π}, ViewPoint → CMView, BoxRatios → Automatic,
 AxesLabel → {"x", "y", "z"}];
```



Or a corrugation:

```
ParametricPlot3D[
 P[t] + s mainunitnormal[t] + Cos[3 s] binormal[t], {t, 0, 2},
 {s, -π, π}, PlotPoints → {Automatic, 25}, ViewPoint → CMView,
 BoxRatios → Automatic, AxesLabel → {"x", "y", "z"}];
```



Every slice of this surface cut by any plane perpendicular to the original curve is the same Cosine wave.

VC.02 Perpendicularity
Give It a Try!

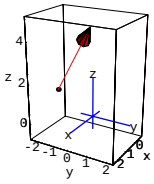
Experience with the starred problems will be useful for understanding developments later in the course.

G.1) Plane fundamentals*

□G.1.a)

Here are a point and a vector with the tail of the vector stuck at the given point:

```
point = {0, -2, 1};
vector = {2, 3, 4};
threedims = Axes3D[2, 0.2];
pointandvector =
 Show[Arrow[vector, Tail → point, VectorColor → Red], threedims,
  Graphics3D[{PointSize[0.04], Point[point]}], ViewPoint → CMView,
  Axes → Automatic, AxesLabel → {"x", "y", "z"}, PlotRange → All];
```

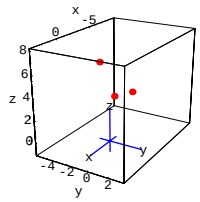


Throw a plot of a plane perpendicular to the given vector and passing through the given point into the plot above. Write down the xyz-equations for this plane.

□G.1.b.i)

Here are three points:

```
point1 = {-2, 1, 4};
point2 = {3, 1, 8};
point3 = {-9, -5, 0};
threedims = Axes3D[3, 0.2];
pointandvector =
 Show[Graphics3D[{Red, PointSize[0.04], Point[point1]}],
  Graphics3D[{Red, PointSize[0.04], Point[point2]}],
  Graphics3D[{Red, PointSize[0.04], Point[point3]}], threedims,
  Axes → Automatic, PlotRange → All, ViewPoint → CMView,
  AxesLabel → {"x", "y", "z"}];
```



Pick one of the points and add the two vectors running from this point to each of the other two points.
Use what you see to plot the plane on which all three points reside.
Confirm your work by showing the three points on the plane plot.

□Tip:

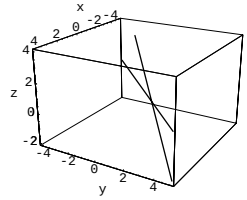
You can use the three points to write down two direction vectors.

Once you do this, your thinking is done.

□G.1.b.ii)

Here are two lines in 3D:

```
Clear[line1, line2, s, t]
line1[t_] = {1, 2, 1} + t {-1, -1, 1};
line2[s_] = {-1, 0, 1} + s {1, 1, 0};
line1plot = ParametricPlot3D[line1[t],
   {t, -3, 3}, PlotPoints → 2, DisplayFunction → Identity];
line2plot = ParametricPlot3D[line2[s],
   {s, -4, 5}, PlotPoints → 2, DisplayFunction → Identity];

Show[line1plot, line2plot, AxesLabel → {"x", "y", "z"},
 ViewPoint → CMView, Axes → Automatic, PlotRange → All,
 DisplayFunction → $DisplayFunction];
```



At what point do these lines intersect?
Plot the plane on which both of these lines reside.
Confirm your work by showing the lines and the plane in the same plot.

□G.1.c.i)

What do you get when you intersect the planes with xyz-equations

$x + y - z = 1$

and

$-x + 2y + 3z = 6$?

Give a parametric formula that describes the points on this intersection and confirm your work with a plot.

□Tip:

If you add the two equations

$x + y - z = 1$, and

$-x + 2y + 3z = 6$

and get

$3y + 2z = 7$,

you aren't done, because $3y + 2z = 7$ is the xyz-equation of a plane passing through $\{1, 0, \frac{7}{2}\}$ with normal vector {0, 3, 2}.

□G.1.c.ii)

How can you tell without plotting that the planes

$x + y - z = 1$

and

$x + 4y + 5z = 6$

cut each other at right angles?

□Tip:

Are their normal vectors perpendicular?

□G.1.d)

When you take the aggregate (collection, set) of all lines through the point

{3, −4, 7}

that are also perpendicular to a given vector

{a, b, c},

what do you get?

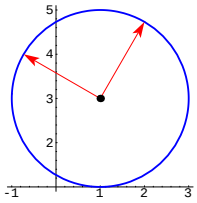
Give an equation that describes the points {x, y, z} on this aggregate of lines.

G.2) Plotting on planes*

□G.2.a)

Look at:

```
Clear[a, b, t, radius, vector1, vector2, curve, vectors, point]
{a, b} = {1, 3};
t = π/3;
radius = 2;
vector1 = {Cos[t], Sin[t]};
vector2 = {Cos[t + π/2], Sin[t + π/2]};
Clear[s]
curve = ParametricPlot[
  {a, b} + radius (Cos[s] vector1 + Sin[s] vector2), {s, 0, 2 π},
  PlotStyle → {{Blue, Thickness[0.01]}}, DisplayFunction → Identity];
vectors = {Arrow[vector1, Tail → {a, b},
   VectorColor → Red, ScaleFactor → radius], Arrow[vector2,
   Tail → {a, b}, VectorColor → Red, ScaleFactor → radius]};
point = Graphics[{PointSize[0.04], Point[{a, b}]}];
Show[curve, vectors, point, AspectRatio → Automatic,
 DisplayFunction → $DisplayFunction];
```



Looks like a circle.

How does the following information tell you for sure that this is a circle and not just some other ellipse?

```
{vector1 . vector1 == vector2 . vector2, vector1 . vector2 == 0}
```

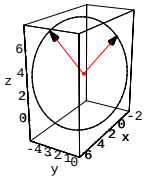
{True, True}

What do you get when you change t?

□G.2.b)

Look at:

```
{a, b, c} = {2, -2, 3};
Clear[s, t];
vector1 = 2 {1, -1, 2};
vector2 = 2 {-2, 1/5 (-2 + 2 √6), 1/5 (4 + √6)};
curve = ParametricPlot3D[{a, b, c} + Cos[s] vector1 + Sin[s] vector2,
  {s, 0, 2 π}, DisplayFunction → Identity];
vectors = {Arrow[vector1, Tail → {a, b, c}, VectorColor → Red],
  Arrow[vector2, Tail → {a, b, c}, VectorColor → Red]};
point = Graphics3D[{Red, PointSize[0.03], Point[{a, b, c}]}];
Show[point, vectors, curve, ViewPoint → CMView,
 Axes → True, AxesLabel → {"x", "y", "z"}, PlotRange → All,
 BoxRatios → Automatic, DisplayFunction → $DisplayFunction];
```



Looks like a circle.

How does the following information tell you for sure that this is a circle and not just some other ellipse?

```
Simplify[{vector1 . vector1 == vector2 . vector2, vector1 . vector2 == 0}]
```

{True, True}

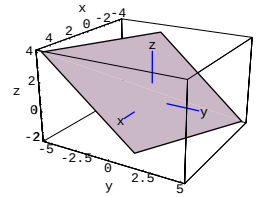
Come up with an xyz-equation of the plane in which this circle resides.

□G.2.c.i)

Even though a plane starts out with two direction vectors, any given plane has infinitely many direction vectors.

Here is a true scale piece of the plane determined by the following point and two direction vectors:

```
point = {0.5, 0, 1};
vector1 = {-2, 1, 0};
vector2 = {-0.2, 1, -1};
Clear[planeplotter, u, v]
planeplotter[u_, v_] = point + u vector1 + v vector2;
{ulow, uhigh} = {-2, 2};
{vlow, vhigh} = {-3, 3};
plane =
 ParametricPlot3D[Evaluate[planeplotter[u, v]], {u, ulow, uhigh},
  {v, vlow, vhigh}, PlotPoints → {2, 2}, DisplayFunction → Identity];
threedims = Axes3D[3.5, 0.4];
planeplot = Show[threedims, plane, PlotRange → All, Axes → Automatic,
  AxesLabel → {"x", "y", "z"}, PlotRange → All, ViewPoint → CMView,
  BoxRatios → Automatic, DisplayFunction → $DisplayFunction];
```



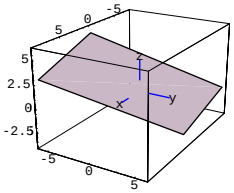
You get the same plane by using new direction vectors

newvector1 = 0.4 vector1 + 0.7 vector2

and

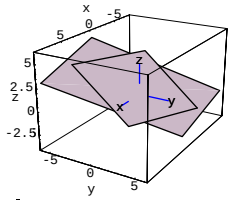
newvector2 = 2 vector1 − 1.5 vector2:

```
newvector1 = 0.4 vector1 + 0.7 vector2;
newvector2 = 2 vector1 - 1.5 vector2;
Clear[newplaneplotter, u, v]
newplaneplotter[u_, v_] = point + u newvector1 + v newvector2;
{ulow, uhigh} = {-5, 5};
{vlow, vhigh} = {-1, 1};
newplane =
 ParametricPlot3D[Evaluate[newplaneplotter[u, v]], {u, ulow, uhigh},
  {v, vlow, vhigh}, PlotPoints → {2, 2}, DisplayFunction → Identity];
newplaneplot = Show[threedims, newplane, Axes → Automatic,
  PlotRange → All, ViewPoint → CMView, BoxRatios → Automatic,
  DisplayFunction → $DisplayFunction];
```

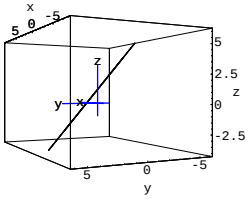


But when you plot the plane using the new direction vectors, as above, you plot a different piece of the plane than you plotted with the original direction vectors:

```
Show[planeplot, newplaneplot];
```



```
Show[planeplot, newplaneplot, ViewPoint → vector1];
```



Go with the plane whose xyz-equation is

$2(x+1)+3(y-1)+(z-1)=0,$

and come up with a point on this plane, and two perpendicular unit vectors that serve as direction vectors for this plane.

□ Tip:

The point of the lead-in to this question is to stress that there is no single correct answer to this problem. In fact, there are infinitely many correct answers to this problem.

□G.2.c.ii)

Plot the circle of radius 3 centered at the point $\{-1, 1, 1\}$ in the plane whose xyz-equation is

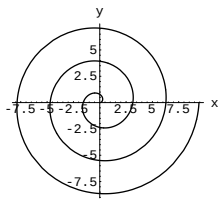
$2(x+1)+3(y-1)+(z-1)=0.$

Include in your plot a big enough piece of the plane to accommodate the circle.

□G.2.c.iii)

Here's a plot of a spiral in the xy-plane:

```
Clear[spiral, t]
spiral[t_] = {t Cos[2 t], t Sin[2 t]};
ParametricPlot[spiral[t], {t, 0, 3 π}, AxesLabel → {"x", "y"}];
```



Use your answer to part i) above to help plot a true scale duplicate copy of this spiral on the plane with xyz-equation

$2(x+1)+3(y-1)+(z-1)=0.$

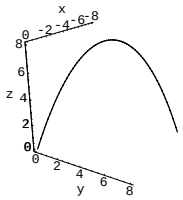
Center your spiral at $\{-1, 1, 1\}$ and include in your plot a big enough hunk of the plane to accomodate the spiral.

G.3) Serious 3D plots: Tubes and ribbons*

□G.3.a)

Here's a 3D curve:

```
Clear[P, t]
P[t_] = {-t, t, 1/2 t (8 - t)};
arch = ParametricPlot3D[P[t], {t, 0, 8}, Axes → Automatic,
  AxesLabel → {"x", "y", "z"}, PlotRange → All, Boxed → False,
  ViewPoint → CMView, BoxRatios → Automatic];
```



You are asked to plot the tube whose skin consists of all circles of radius 0.5 that

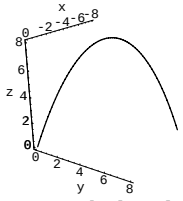
→ are centered on this curve, and

→ lie in planes that cut this curve perpendicularly.

□G.3.b)

Here's the same 3D curve:

```
Clear[P, t]
P[t_] = {-t, t, 1/2 t (8 - t)};
arch = ParametricPlot3D[P[t], {t, 0, 8}, Axes → Automatic,
  AxesLabel → {"x", "y", "z"}, PlotRange → All, Boxed → False,
  ViewPoint → CMView, BoxRatios → Automatic];
```



You are asked to plot a ribbon two units wide whose center curve coincides with the curve plotted above. Corrugate the ribbon if you like.

□ Tip:

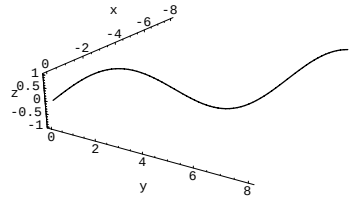
As you may have noted, there are many such ribbons; you are asked to plot just one.

You'll probably want to use binormal[t].

□G.3.c)

Here's a different 3D curve:

```
Clear[curveplotter, t]
curveplotter[t_] = {-t, t, Sin[t]};
snake = ParametricPlot3D[curveplotter[t], {t, 0, 8}, Axes → Automatic,
  AxesLabel → {"x", "y", "z"}, PlotRange → All, Boxed → False,
  ViewPoint → CMView, BoxRatios → Automatic];
```



This time you're asked to add to this plot four additional curves with the property that if you slice the original curve by any plane perpendicular to the original curve, then:
→ The four additional curves will show up on this plane as dots at the corners of a square one unit on a side, and
→ the original curve will show up as a dot in the center of the square.

□ **Tip:**

As you may have noted, there are many such sets of four curves; you are asked to plot just one set.

You'll probably want to use mainunitnormal[t] and binormal[t]. Your plot may reveal an unexpected phenom. If so, then try to explain why it happened.

□G.3.d)

Now it's time to take the shackles off. Turn yourself loose and do something artistic and creative with a curve of your choice.
Show off.

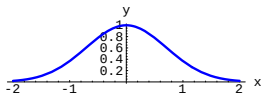
G.4) Experiments with linearizations*

Linearization is a technique much favored by advanced scientists. You will run across linearizations throughout most of the lessons to follow, so you might as well get your feet wet here.

□G.4.a) Linearizations of f[x]

Here is a simple curve in two dimensions:

```
Clear[f, x]
f[x_] = E^-x^2;
fplot = Plot[f[x], {x, -2, 2}, AxesLabel → {"x", "y"},
  PlotStyle → {{Blue, Thickness[0.01]}}];
```



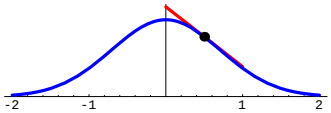
Here's what a lot of folks call the linearization of f[x] at x = a:

```
Clear[linearf, a]
linearf[x_, a_] = f[a] + f'[a] (x - a)
```

```
E^-a^2 - 2 a E^-a^2 (-a + x)
```

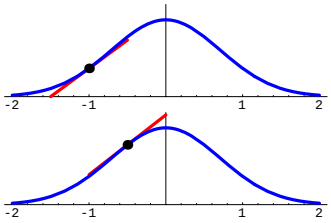
And a plot in the case a = 0.5:

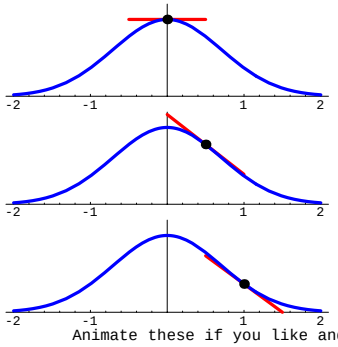
```
Clear[a, linplot, pointplot]
linplot[a_] := Plot[linearf[x, a], {x, a - 0.5, a + 0.5},
  PlotStyle → {{Red, Thickness[0.01]}}, PlotRange → {0, 1.2},
  Ticks → {Automatic, None}, DisplayFunction → Identity];
pointplot[a_] := Graphics[{PointSize[0.03], Point[{a, f[a]}]}];
Show[linplot[0.5], fplot, pointplot[0.5],
 DisplayFunction → $DisplayFunction];
```



A little movie:

```
Table[Show[linplot[a], fplot,
  pointplot[a], DisplayFunction → $DisplayFunction],
 {a, -1, 1, 2/4}];
```





Animate these if you like and run at a slow speed.

That's right; linearf[x, a] is nothing but the tangential approximation line at x = a.
As you can see, it's a pretty darn good approximation of f[x] for x's near a. This is why the folks like linearizations–they approximate well and they are line functions so they are easy to work with.
Play, as above, with linearizations for functions f[x] (of one variable) of your own choice at points of your own choice. You may want to change the value of the PlotRange specification.

□G.4.b.i)

With somewhat more work than needed for functions of one variable, you can linearize a function f[x, y] of two variables.
Here's the idea:
According to the work in B.3), you can calculate a normal to the surface z = f[x, y] at a point {a, b, f[a, b]} as follows:

```
Clear[x, y, f, a, b, surfacepoint]
surfacepoint[x_, y_] = {x, y, f[x, y]};
Clear[xcurve, ycurve]
xcurve[x_] = surfacepoint[x, b];
ycurve[y_] = surfacepoint[a, y];
xtanvector = D[xcurve[x], x] /. x → a;
ytanvector = D[ycurve[y], y] /. y → b;
normal = Cross[xtanvector, ytanvector]
```

```
{-f^(1,0)[a, b], -f^(0,1)[a, b], 1}
```

Notice the shortened command for calculating the cross product.

This tells you that the xyz-equation for a plane touching the surface z = f[x, y] at {a, b, f[a, b]} and sharing the same normal vector comes from:

```
normal . {x - a, y - b, z - f[a, b]} == 0
```

```
z - f[a, b] - (-b + y) f^(0,1)[a, b] - (-a + x) f^(1,0)[a, b] == 0
```

This gives you

$$z = f[a, b] + \frac{\partial f[a,b]}{\partial x}(x - a) + \frac{\partial f[a,b]}{\partial y}(y - b).$$

The pros call this the linearization,

linearf[x, y],

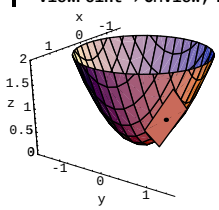
of f[x, y] at {a, b}.
Here's a plot of

$$z = f[x, y] = x^2 + y^2$$

and its linearization at

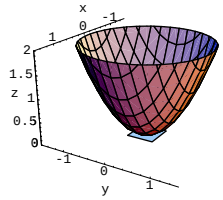
{a, b} = {0.5, 0.8}:

```
Clear[f, linearf, x, y]
f[x_, y_] = x^2 + y^2;
{a, b} = {0.5, 0.8};
A = D[f[x, y], x] /. {x → a, y → b};
B = D[f[x, y], y] /. {x → a, y → b};
linearf[x_, y_] = f[a, b] + A (x - a) + B (y - b);
surface = Plot3D[f[x, y], {x, -1.5, 1.5}, {y, -1.5, 1.5},
  PlotRange → {0, 2}, ClipFill → None, DisplayFunction → Identity];
linearized = Plot3D[linearf[x, y], {x, a - 0.3, a + 0.3},
  {y, b - 0.3, b + 0.3}, PlotPoints → {2, 2}, DisplayFunction → Identity];
pointoflinearization =
 Graphics3D[{PointSize[0.02], Point[{a, b, f[a, b]}]}];
Show[surface, linearized, pointoflinearization,
 AxesLabel → {"x", "y", "z"}, BoxRatios → Automatic, Boxed → False,
 ViewPoint → CMView, DisplayFunction → $DisplayFunction];
```



Here's what you get when you linearize the same function at
{a, b} = {0, 0}:

```
{a, b} = {0, 0};
Clear[linearf]
A = D[f[x, y], x] /. {x → a, y → b};
B = D[f[x, y], y] /. {x → a, y → b};
linearf[x_, y_] = f[a, b] + A (x - a) + B (y - b);
linearized = Plot3D[linearf[x, y], {x, a - 0.3, a + 0.3},
  {y, b - 0.3, b + 0.3}, PlotPoints → {2, 2}, DisplayFunction → Identity];
pointoflinearization =
 Graphics3D[{PointSize[0.02], Point[{a, b, f[a, b]}]}];
Show[surface, linearized, pointoflinearization,
 AxesLabel → {"x", "y", "z"}, BoxRatios → Automatic, Boxed → False,
 ViewPoint → CMView, DisplayFunction → $DisplayFunction];
```

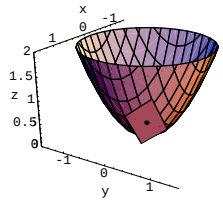


Plot the same function and its linearization at
{a, b} = {1, 0}.
Then plot the same function and its linearization at
{a, b} = {0.1, 0.5}.
Describe what you see and then say what relation the linearized surface has to the actual surface at the point of linearization.
How well do the linearized versions approximate the actual function at the points of linearization?

□G.4.b.ii)

Stay with the same function as above and look at the surface and its linearization at
{a, b} = {0.7, 0.4}:

```
{a, b} = {0.7, 0.4};
Clear[linearf]
A = D[f[x, y], x] /. {x → a, y → b};
B = D[f[x, y], y] /. {x → a, y → b};
linearf[x_, y_] = f[a, b] + A (x - a) + B (y - b);
linearized = Plot3D[linearf[x, y], {x, a - 0.3, a + 0.3},
  {y, b - 0.3, b + 0.3}, PlotPoints → {2, 2}, DisplayFunction → Identity];
pointoflinearization =
 Graphics3D[{PointSize[0.02], Point[{a, b, f[a, b]}]}];
Show[surface, linearized, pointoflinearization,
 AxesLabel → {"x", "y", "z"}, BoxRatios → Automatic, Boxed → False,
 ViewPoint → CMView, DisplayFunction → $DisplayFunction];
```



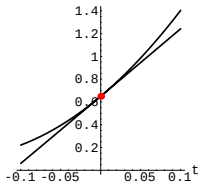
What you see are plots of the function and its linearized version at the point of linearization at {0.7, 0.4}.
Put
$\{x[t], y[t]\} = \{0.7 - 4\,\mathrm{Sin}[t],\ 0.4\,e^t\}$
and note that this curve passes through the point of linearization, {0.7, 0.4}, when t = 0:

```
Clear[x, y, t]
{x[t_], y[t_]} = {0.7 + 4 Sin[t], 0.4 E^t};
{x[t], y[t]} /. t → 0
```

```
{0.7, 0.4}
```

Now look at plots of the curves
f[x[t], y[t]] and linearf[x[t], y[t]]
for
$-0.1 \le t \le 0.1$:

```
h = 0.1;
Plot[{f[x[t], y[t]], linearf[x[t], y[t]]}, {t, 0 - h, 0 + h},
 PlotStyle → Thickness[0.01], AspectRatio → 1, AxesLabel → {"t", ""},
 Epilog → {Red, PointSize[0.04], Point[{0, f[x[0], y[0]]}]}];
```



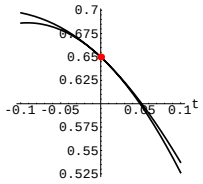
This time put
$\{x[t], y[t]\} = \{0.7\,\mathrm{Cos}[3\,t],\ 0.4 - t\}$
and note that this curve also passes through the point of linearization at {0.7, 0.4} when t = 0:

```
Clear[x, y, t]
{x[t_], y[t_]} = {0.7 Cos[3 t], 0.4 - t};
{x[t], y[t]} /. t → 0
```

```
{0.7, 0.4}
```

Now look at plots of the curves f[x[t], y[t]], and linearf[x[t], y[t]] for
$-0.1 \le t \le 0.1$:

```
h = 0.1;
Plot[{f[x[t], y[t]], linearf[x[t], y[t]]}, {t, 0 - h, 0 + h},
 PlotStyle → Thickness[0.01], AspectRatio → 1, AxesLabel → {"t", ""},
 Epilog → {Red, PointSize[0.04], Point[{0, f[x[0], y[0]]}]}];
```



Describe what you see in the two plots above and try to account for why you see it.
Explain why you will see the same phenomenon no matter what
{x[t], y[t]}
you go with provided
{x[t], y[t]} = {0.7, 0.4} when t = 0.

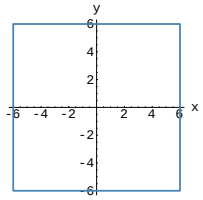
G.5) Badger borings

Calculus&Mathematica thanks Professor Rod Smart of the University of Wisconsin for suggesting this problem. Professor Smart mentions that a company in Madison, Wisconsin actually called him to ask how to solve problems of this type.

□G.5.a.i)

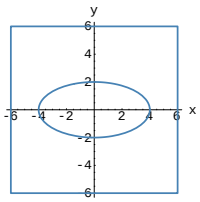
You are the chief engineer at the Badger Steel Plate Company in Madison, Wisonsin. In comes an order for 750 square steel plates, each measuring 12 inches wide and 12 inches long.
Go to the drawing board:

```
plate = Graphics[{SteelBlue, Thickness[0.01],
   Line[{{-6, -6}, {-6, 6}, {6, 6}, {6, -6}, {-6, -6}}]}];
plateplot = Show[plate, Axes → True, AxesOrigin → {0, 0},
  AspectRatio → Automatic, AxesLabel → {"x", "y"}];
```



But here's the kicker:
The plates are to have everything inside the ellipse
$(\frac{x}{4})^2 + (\frac{y}{2})^2 = 1$
cut out.
Take a look:

```
Clear[x, y, t]
x[t_] = 4 Cos[t];
y[t_] = 2 Sin[t];
hole = ParametricPlot[{x[t], y[t]}, {t, 0, 2 π}, PlotStyle →
   {{SteelBlue, Thickness[0.01]}}, DisplayFunction → Identity];
Show[plateplot, hole, DisplayFunction → $DisplayFunction];
```



You have a new robotic router that takes instructions from Mathematica and whose cutting center can be programmed to follow any curve you tell it to follow.
If you are going to use a bit 1 inch in diameter, then what curve should you program in as the path of the center of the router to cut out the ellipse?
After you have found the correct curve, add its plot to the plot above.

□ **Big Tip:**

If your curve is an ellipse, then you screwed up.

The normal vector D[unittan[t], t] could be very useful.

□G.5.a.ii)

Actually, the bit size of 1 inch in diameter used above was arbitrarily chosen by reaching into the drawer and pulling out a bit. You could always get by with a smaller bit. Why?
But you cannot use bits that are too large. Why?
Try to estimate the diameter of the largest bit that you could use to do the job.

G.6) Using the product rule to break acceleration vectors into normal and tangential components

□G.6.a.i)

At time t, an object is given to be at a point
P[t] = {x[t], y[t]}.
Its velocity at time t is just
velocity[t] = D[P[t], t].
The unit tangent vector at P[t] is
unittan[t] = $\frac{\text{velocity[t]}}{\sqrt{\text{velocity[t]}.\text{velocity[t]}}}$.
Explain why
velocity[t] = speed[t] unittan[t]
where
speed[t] = $\sqrt{\text{velocity[t]}.\text{velocity[t]}}$.

□G.6.a.ii)

The acceleration vector at P[t] is given by
accel[t] = D[velocity[t], t].
Use the fact that
velocity[t] = speed[t] unittan[t],
and use the product rule for taking derivatives to say why
accel[t] = D[speed[t], t] unittan[t] + speed[t] mainnormal[t].

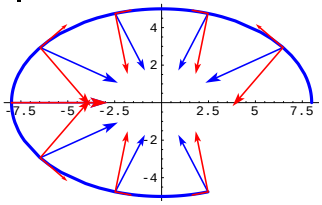
□G.6.a.iii)

When you write
accel[t] = D[speed[t], t] unittan[t] + speed[t] mainnormal[t],
then why is it fairly transparent that
D[speed[t], t] unittan[t]
is the tangential component of the acceleration and
speed[t] mainnormal[t]
is the normal component of the acceleration?

□G.6.a.iv)

Explain why the following true scale plot came out the way it did and discuss the information the plot exhibits.

```
Clear[P, t, velocity, accel, speed, unittan, tanaccel]
P[t_] = {8 Cos[0.8 t], 5 Sin[0.8 t]};
path = ParametricPlot[P[t], {t, 0, 2 π},
  PlotStyle → {{Blue, Thickness[0.01]}}, DisplayFunction → Identity];
velocity[t_] = D[P[t], t];
accel[t_] = D[velocity[t], t];
unittan[t_] = velocity[t] / Sqrt[velocity[t] . velocity[t]];
speed[t_] = Sqrt[velocity[t] . velocity[t]];
tanaccel[t_] = D[speed[t], t] unittan[t];
vectors = Table[{Arrow[accel[t], Tail → P[t]],
   Arrow[tanaccel[t], Tail → P[t], VectorColor → Red],
   Arrow[accel[t] - tanaccel[t], Tail → P[t], VectorColor → Red]},
  {t, π/4, 2 π, π/4}];
Show[path, vectors, AspectRatio → Automatic,
 DisplayFunction → $DisplayFunction];
```

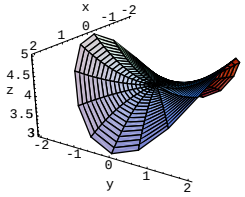


G.7) Using the normal vector to bounce light beams off surfaces

□G.7.a)

Here's a piece of the surface
$z = 4 - (\frac{x}{2})^2 + (\frac{y}{2})^2$:

```
Clear[f, x, y, z, r, t, surfaceplotter]
f[x_, y_] = 4 - (x/2)^2 + (y/2)^2;
x[r_, t_] = r Cos[t];
y[r_, t_] = r Sin[t];
z[r_, t_] = f[x[r, t], y[r, t]];
surfaceplotter[r_, t_] = {x[r, t], y[r, t], z[r, t]};

surface = ParametricPlot3D[surfaceplotter[r, t], {r, 0, 2},
  {t, 0, 2 π}, ViewPoint → CMView, PlotRange → All, Boxed → False,
  AxesLabel → {"x", "y", "z"}];
```

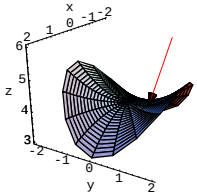


A light beam emanates from {−1, 1, 6} and strikes the surface at:

```
hit = surfaceplotter[1, 1.2]
{0.362358, 0.932039, 4.18435}
```

Here's a look:

```
lightsource = {-1, 1, 6};
Show[surface,
 Arrow[hit - lightsource, Tail → lightsource, VectorColor → Red],
 PlotRange → All];
```



Your job is to plot the vector on which the reflected light moves.

G.8) Kissing circles and curvature

A curve in two dimensions is plotted by specifying two functions x[t] and y[t] and plotting

P[t] = {x[t], y[t]}

for t running through a desired interval.

A curve in three dimensions is plotted by specifying three functions x[t], y[t], and z[t] and plotting

P[t] = {x[t], y[t], z[t]}

for t running through a desired interval.

In both situations, you get the unit tangent vector at P[t] through the formula:

unittan[t] = $\frac{D[P[t],t]}{\sqrt{D[P[t],t].D[P[t],t]}}$.

In both situations, the main normal at P[t] is given by

mainnormal[t] = D[unittan[t], t].

The vector mainnormal[t] is a measurement of how fast the unit tangent is turning as t progresses.

You can get an even better measurement of how the curve turns by calculating the instantaneous rate of change of the unit tangent as a function of arc length s measured on the curve from a specified reference point usually at one end of the curve.

This measurement is intrinsic to the shape of the curve, and does not depend on the specific parameterization of the curve you are using.

The chain rule tells you that at P[t], this measurement is given by

turn[t] = D[unittan[t], s]

= D[unittan[t], t] D[t, s]

= $\frac{D[unittan[t],t]}{\sqrt{D[P[t],t].D[P[t],t]}}$

because

D[t, s] = $\frac{1}{\sqrt{x'[t]^2 + y'[t]^2}}$

= $\frac{1}{\sqrt{D[P[t],t].D[P[t],t]}}$

For an elucidation on this formula for D[t, s], double click the box.

Two dimensions:

Arc length s is the integral of $\sqrt{x'[t]^2 + y'[t]^2}$.

So

D[s, t] = $\sqrt{x'[t]^2 + y'[t]^2}$

= $\sqrt{D[P[t], t] . D[P[t], t]}$

because D[P[t], t] = {x′[t], y′[t]}.

Accordingly,

D[t, s] = $\frac{1}{\sqrt{x'[t]^2 + y'[t]^2}}$

= $\frac{1}{\sqrt{D[P[t],t].D[P[t],t]}}$.

Three dimensions:

Arc length s is the integral of $\sqrt{x'[t]^2 + y'[t]^2 + z'[t]^2}$.

So

D[s, t] = $\sqrt{x'[t]^2 + y'[t]^2 + z'[t]^2}$

= $\sqrt{D[P[t], t] . D[P[t], t]}$,

because D[P[t], t] = {x′[t], y′[t], z′[t]}.

Accordingly

D[t, s] = $\frac{1}{\sqrt{x'[t]^2 + y'[t]^2 + z'[t]^2}}$

= $\frac{1}{\sqrt{D[P[t],t].D[P[t],t]}}$.

Make note of the fact that

→ turn[t] is a vector that points in the same direction as mainnormal[t].

Why?

Because it's a positive multiple of

D[unittan[t], t] = mainnormal[t].

This tells you that turn[t] is perpendicular to the curve at P[t].

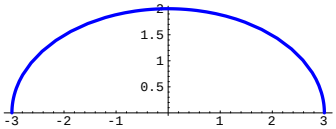
The length of turn[t] measures the roundness of the curve at P[t].

□G.8.a.i)

Here, in true scale, is the curve

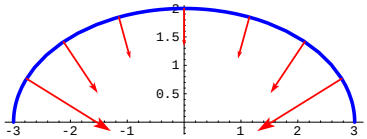
P[t] = {5 Cos[t], 3 Sin[t]} for 0 ≤ t ≤ π:

```
Clear[P, t]
P[t_] = {3 Cos[t], 2 Sin[t]};
curve = ParametricPlot[P[t], {t, 0, π},
  PlotStyle → {{Blue, Thickness[0.01]}}, AspectRatio → Automatic];
```



Here's the same curve together with a selection of vectors turn[t] with tails at P[t]:

```
Clear[unittan, turn]
unittan[t_] = D[P[t], t] / √(D[P[t], t] . D[P[t], t]);
turn[t_] = D[unittan[t], t] / √(D[P[t], t] . D[P[t], t]);
scalefactor = 3;
turnvectors = Table[Arrow[turn[t], Tail → P[t],
   VectorColor → Red, ScaleFactor → scalefactor], {t, π/8, 7π/8, π/8}];
Show[curve, turnvectors, AspectRatio → Automatic];
```



The turn vectors are shown 3 times longer than their actual lengths.

Describe, in words, how the lengths of the turn vectors are related to flatness or roundness of the curve.

Then plot the length

||turn[t]|| = $\sqrt{\text{turn[t] . turn[t]}}$ for 0 ≤ t ≤ π.

Discuss the relations between the two plots.

How does ||turn[t]|| plot out over intervals where the curve is fairly flat?

On what side of the curve does turn[t], with its tail at P[t], point?

□G.8.a.ii)

Here is the same curve shown in true scale with the circle of radius

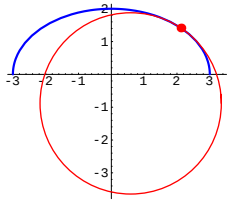
$\frac{1}{\|turn[a]\|}$ = $\frac{1}{\sqrt{turn[t].turn[t]}}$

centered at

P[a] + $\frac{turn[a]}{turn[a].turn[a]}$

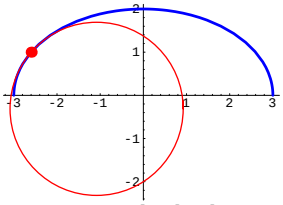
for a = $\frac{\pi}{4}$:

```
Clear[a, center, radius, circle, point]
center[a_] = P[a] + turn[a] / (turn[a] . turn[a]);
radius[a_] = 1 / √(turn[a] . turn[a]);
circle[a_] = Graphics[{Red, Circle[center[a], radius[a]]}];
point[a_] = Graphics[{Red, PointSize[0.04], Point[P[a]]}];
a = π/4;
Show[
 curve, circle[a], point[a], PlotRange → All, AspectRatio → Automatic];
```



Here's what happens when you go with a = $\frac{5\pi}{6}$:

```
a = 5π/6;
Show[
 curve, circle[a], point[a], PlotRange → All, AspectRatio → Automatic];
```



Experiment with what happens when you go with other choices of
a with $0 \le a \le \pi$,
paying special attention to what you get when you go with
$a = 0, \frac{\pi}{2}$, and π.
Make a movie if you like.
After you are done experimenting, say what you think is happening and try to explain why it is happening.

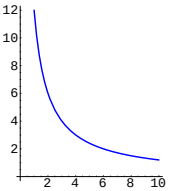
□G.8.a.iii) Kissing, smooching, osculating, and curvature

Fancy folks call the circles you were studying in the last part by the name "osculating circle" at P[a]. You might prefer to call them "kissing circles" or "smooching circles" at P[a].
Most everyone calls
$\|\text{turn}[a]\| = \sqrt{\text{turn}[t] \,.\, \text{turn}[t]} = \frac{1}{\text{radius}}$
(where radius stands for the radius of the kissing circle at P[a]) by the name "curvature" at P[a].
Explain what curvature tries to measure.
Does a high curvature number at P[a] mean that the curve is very flat or very rounded at P[a]?
What does a low curvature number mean?

□G.8.a.iv)

Here's a true scale parametric plot of part of the curve $y = \frac{12}{x}$:

```
Clear[P, t]
P[t_] = {t, 12/t};
curve = ParametricPlot[P[t], {t, 1, 10},
  PlotStyle → {{Blue, Thickness[0.01]}}, AspectRatio → Automatic];
```

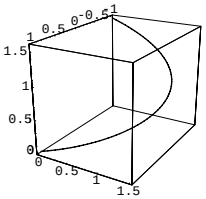


Estimate the point on this curve at which the curvature is biggest and plot the kissing circle at that point. Then, on a separate plot, show the curve, the kissing circle at the point of biggest curvature, and a few other kissing circles.
A good eye-ball estimate is OK.

□G.8.b.i) 3D curves

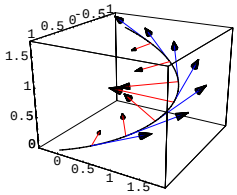
Here's a 3D curve:

```
Clear[P, t]
P[t_] = {Cos[t], 1.5 Sin[t], t/2};
curve = ParametricPlot3D[P[t], {t, 0, π}, BoxRatios → Automatic,
  ViewPoint → CMView];
```



Here's the same 3D curve together with a selection of vectors
unittan[t] and turn[t] with tails at P[t]:

```
Clear[unittan, turn]
unittan[t_] = D[P[t], t] / √(D[P[t], t] . D[P[t], t]);
turn[t_] = D[unittan[t], t] / √(D[P[t], t] . D[P[t], t]);
unittanvectors = Table[Arrow[unittan[t],
   Tail → P[t], VectorColor → Blue], {t, π/8, 7π/8, π/8}];
turnvectors = Table[
  Arrow[turn[t], Tail → P[t], VectorColor → Red], {t, π/8, 7π/8, π/8}];
Show[curve, unittanvectors, turnvectors, ViewPoint → CMView,
 BoxRatios → Automatic];
```

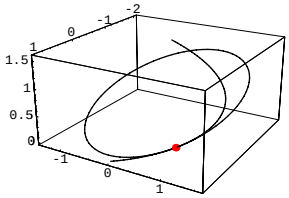


Describe how you believe the lengths and directions of the turn vectors are related to the shape of the curve.

□G.8.b.ii) Kissing, smooching, and osculating in 3D

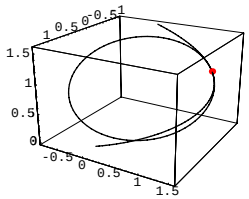
You can do kissing circles in three dimensions:
Given a curve P[t] and a value t = a, you center the circle at
center = $P[a] + \frac{\text{turn}[a]}{\text{turn}[a].\text{turn}[a]}$
and put
radius = $\frac{1}{\|\text{turn}[a]\|} = \frac{1}{\sqrt{\text{turn}[a].\text{turn}[a]}}$
and now you gotta choose two perpendicular unit vectors to determine the plane in which the circle resides. You want the circle to be tangent to the plane, so one good unit vector to go with is
vector1[a] = unittan[a].
The other vector you want is:
vector2[a] = $\frac{\text{turn}[a]}{\sqrt{\text{turn}[t].\text{turn}[t]}}$.
Try it out on the same curve as plotted above:

```
Clear[P, t]
P[t_] = {Cos[t], 1.5 Sin[t], t/2};
curve = ParametricPlot3D[P[t], {t, 0, π}, DisplayFunction → Identity];
Clear[unittan, turn]
unittan[t_] = D[P[t], t] / √(D[P[t], t] . D[P[t], t]);
turn[t_] = D[unittan[t], t] / √(D[P[t], t] . D[P[t], t]);
Clear[t, a, center, radius, point, vector1, vector2, kisser]
point[a_] = Graphics3D[{Red, PointSize[0.03], Point[P[a]]}];
center[a_] = P[a] + turn[a] / (turn[a] . turn[a]);
radius[a_] = 1 / √(turn[a] . turn[a]);
vector1[a_] = unittan[a];
vector2[a_] = turn[a] / √(turn[a] . turn[a]);
kisser[a_] := ParametricPlot3D[
  center[a] + radius[a] (Cos[t] vector1[a] + Sin[t] vector2[a]),
  {t, 0, 2π}, DisplayFunction → Identity];
a = π/4;
Show[curve, point[a], kisser[a], ViewPoint → CMView,
 BoxRatios → Automatic, DisplayFunction → $DisplayFunction];
```



Another:

```
a = 2π/3;
Show[curve, point[a], kisser[a], ViewPoint → CMView,
 BoxRatios → Automatic, DisplayFunction → $DisplayFunction];
```



Estimate the point on this curve at which the curvature is biggest and plot the kissing circle at that point.

G.9) Measurements with the cross product*

Given two vectors

X = {x[1], x[2], x[3]}

and

Y = {y[1], y[2], y[3]}:

```
Clear[X, x, Y, y, k]
X = {x[1], x[2], x[3]};
Y = {y[1], y[2], y[3]};
```

To calculate the cross product X×Y you make a matrix with

{i, j, k}

in the top (horizontal) row, with X in the middle (horizontal) row, and with Y in the bottom row:

```
Clear[i, j, k]
MatrixForm[{{i, j, k}, X, Y}]
```

$$\begin{pmatrix} i & j & k \\ x[1] & x[2] & x[3] \\ y[1] & y[2] & y[3] \end{pmatrix}$$

Next, take the determinant of this matrix:

```
Det[{{i, j, k}, X, Y}]
```

$$-k\,x[2]\,y[1] + j\,x[3]\,y[1] + k\,x[1]\,y[2] - i\,x[3]\,y[2] - j\,x[1]\,y[3] + i\,x[2]\,y[3]$$

To arrive at the cross product X×Y, you replace

i by {1, 0, 0},

replace

j by {0, 1, 0}

and replace

k by {0, 0, 1}:

```
XcrossY =
 Det[{{i, j, k}, X, Y}] /. {i → {1, 0, 0}, j → {0, 1, 0}, k → {0, 0, 1}}
```

```
{-x[3] y[2] + x[2] y[3], x[3] y[1] - x[1] y[3], -x[2] y[1] + x[1] y[2]}
```

The dot product is a number. The cross product is a vector.

For example, if X and Y are:

```
X = {-12, -37, 92};
Y = {0, -68, 49};
```

Then X×Y is:

```
XcrossY =
 Det[{{i, j, k}, X, Y}] /. {i → {1, 0, 0}, j → {0, 1, 0}, k → {0, 0, 1}}
```

```
{4443, 588, 816}
```

A quick command for this is built into Mathematica:

```
XcrossY == Cross[X, Y]
```

```
True
```

And the vector X×Y is autoamtically perpendicular to both X and Y:

```
X . XcrossY
```

```
0
```

```
Y . XcrossY
```

```
0
```

□G.9.a) The length of X×Y

The following calculations give clues to finding out what the length of X×Y measures:

```
Clear[X, x, Y, y];
X = {x[1], x[2], x[3]};
Y = {y[1], y[2], y[3]};
XcrossY = Cross[X, Y];
Expand[(X . Y)^2 + XcrossY . XcrossY]
```

```
x[1]^2 y[1]^2 + x[2]^2 y[1]^2 + x[3]^2 y[1]^2 + x[1]^2 y[2]^2 + x[2]^2 y[2]^2 +
 x[3]^2 y[2]^2 + x[1]^2 y[3]^2 + x[2]^2 y[3]^2 + x[3]^2 y[3]^2
```

```
Expand[(X . X) (Y . Y)]
```

```
x[1]^2 y[1]^2 + x[2]^2 y[1]^2 + x[3]^2 y[1]^2 + x[1]^2 y[2]^2 + x[2]^2 y[2]^2 +
 x[3]^2 y[2]^2 + x[1]^2 y[3]^2 + x[2]^2 y[3]^2 + x[3]^2 y[3]^2
```

Remembering that

$(X \cdot Y)^2 = \|X\|^2 \|Y\|^2 \text{Cos[angle between]}^2$,

explain how these calculations reveal that

$\|X \times Y\|^2$

$= (X \times Y) \cdot (X \times Y)$

$= \|X\|^2 \|Y\|^2 \text{Sin[angle between]}^2$,

so that

$\|X \times Y\| = \|X\| \|Y\| |\text{Sin[angle between]}|$.

□Tip:

Remember:

```
Clear[t]
TrigExpand[Sin[t]^2 + Cos[t]^2]
```

```
1
```

□G.9.b.i)

No matter what s you take, the following vectors are perpendicular:

```
Clear[X, Y, s];
X[s_] = {Cos[s]/Sqrt[2], Cos[s]/Sqrt[2], Sin[s]};
Y[s_] = {-Sin[s]/Sqrt[2], -Sin[s]/Sqrt[2], Cos[s]};
Expand[X[s] . Y[s], Trig → True]
```

```
0
```

They are unit vectors, because the length of each of them is 1:

```
TrigExpand[X[s] . X[s]]
```

```
1
```

```
TrigExpand[Y[s] . Y[s]]
```

```
1
```

No matter what s you take, you find that X[s]×Y[s] is a unit vector:

```
cross = Cross[X[s], Y[s]];
TrigExpand[cross . cross]
```

```
1
```

Explain why:

If you take any two perpendicular unit vectors X and Y, then

→ X×Y is perpendicular to both X and Y and

→ X×Y is also a unit vector.

If X and Y are unit vectors, but X is not perpendicular to Y, then can X×Y be a unit vector? Why not?

□G.9.b.ii)

Use this to explain why if you have a curve P[t] in three dimensions, then

binormal[t] = unittan[t]×mainunitnormal[t]

is a unit vector perpendicular to the curve at P[t] and perpendicular to mainunitnormal[t] at P[t].

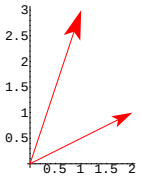
□Tip:

If you don't understand the terminology, see the Tutorial problem on tubes and horns.

□G.9.b.iii) Using the cross product to measure area of parallelograms

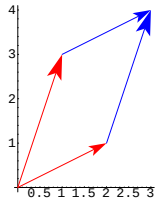
Here are two vectors in two dimensions:

```
X = {2, 1};
Y = {1, 3};
Show[Arrow[X, VectorColor → Red], Arrow[Y, VectorColor → Red],
 Axes → Automatic];
```



And here is the parallelogram they determine:

```
Show[Arrow[X, VectorColor → Red],
 Arrow[Y, VectorColor → Red], Arrow[Y, Tail → X, VectorColor → Blue],
 Arrow[X, Tail → Y, VectorColor → Blue], Axes → Automatic];
```



Now look at:

```
X;
Y;
XthreeD = {X[1], X[2], 0};
YthreeD = {Y[1], Y[2], 0};
cross = Cross[XthreeD, YthreeD];
√cross . cross
```

5

Use the fact that

$$\|X\times Y\| = \sqrt{(X\times Y).(X\times Y)} = \|X\|\,\|Y\|\,|\text{Sin[angle between]}|$$

to explain why the output from the last instruction above measures the area of the parallelogram plotted above.

□Tip:

Remember that the area of a parallelogram is given by

B h

where B is the length of the base and h is the height measured perpendicularly from the base.

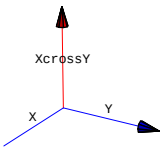
G.10) Thumbs up or thumbs down?

This is a chance for you to learn how the cross product X×Y is oriented with respect to X and Y. By this point, you know that X×Y is perpendicular to both X and Y. But you can say a little more after you do some experimentation.

□G.10.a)

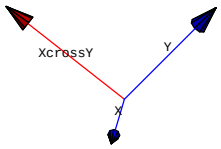
Here are several samples of plots of 3D vectors X, Y, and the cross product X×Y:

```
X = {1, 0, 0};
Y = {0, 1, 0};
XcrossY = Cross[X, Y];
Show[Arrow[X], Graphics3D[Text["X", X/2 + {0, 0, 0.1}]],
 Arrow[Y], Graphics3D[Text["Y", Y/2 + {0, 0, 0.1}]],
 Arrow[XcrossY, VectorColor → Red],
 Graphics3D[Text["XcrossY", XcrossY/2]], BoxRatios → Automatic,
 ViewPoint → CMView, Axes → None, Boxed → False];
```



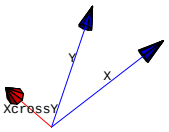
If you grab onto the shaft of X×Y with your right hand so as to use your fingers to push X onto Y through the smaller of the two possible angles, does your thumb point with X×Y or against X×Y?

```
X = {1, 0.4, 0};
Y = {0, 1, 1.2};
XcrossY = Cross[X, Y];
Show[Arrow[X, Tail → {0, 0, 0}, VectorColor → Blue],
 Graphics3D[Text["X", X/2 + {0, 0, 0.1}]],
 Arrow[Y, Tail → {0, 0, 0}, VectorColor → Blue],
 Graphics3D[Text["Y", Y/2 + {0, 0, 0.1}]],
 Arrow[XcrossY, Tail → {0, 0, 0}, VectorColor → Red],
 Graphics3D[Text["XcrossY", XcrossY/2]], PlotRange → All,
 BoxRatios → Automatic, ViewPoint → CMView, Axes → None, Boxed → False];
```



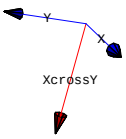
If you grab onto the shaft of X×Y with your right hand so as to use your fingers to push X onto Y through the smaller of the two possible angles, does your thumb point with X×Y or against X×Y?

```
X = {-1, 1, 1};
Y = {-1, 0, 1.3};
XcrossY = X×Y;
Show[Arrow[X], Graphics3D[Text["X", X/2 + {0, 0, 0.1}]],
 Arrow[Y], Graphics3D[Text["Y", Y/2 + {0, 0, 0.1}]],
 Arrow[XcrossY, VectorColor → Red],
 Graphics3D[Text["XcrossY", XcrossY/2]], PlotRange → All,
 BoxRatios → Automatic, ViewPoint → CMView, Axes → None, Boxed → False];
```



If you grab onto the shaft of X×Y with your right hand so as to use your fingers to push X onto Y through the smaller of the two possible angles, does your thumb point with X×Y or against X×Y?

```
X = {1, 1, 0};
Y = {1, -0.5, 0.3};
XcrossY = X×Y;
Show[Arrow[X], Graphics3D[Text["X", X/2]], Arrow[Y],
 Graphics3D[Text["Y", Y/2]], Arrow[XcrossY, VectorColor → Red],
 Graphics3D[Text["XcrossY", XcrossY/2]], ViewPoint → CMView,
 Axes → None, Boxed → False];
```



If you grab onto the shaft of X×Y with your right hand so as to use your fingers to push X onto Y through the smaller of the two possible angles, does your thumb point with X×Y or against X×Y?
Do more experiments like these and then describe how X×Y is oriented with respect to X and Y.

```
If your response is correct, then you have written down what a lot of
      folks call "the right hand rule for the cross product."
```

□G.10.b)

Given two 3-dimensional vectors X and Y, then how do you express Y×X in terms of X×Y?
Try some examples before you jump to a conclusion.
Use your ideas from part a) to say why this is natural.

□Tip:

Try some examples. Here's one:

```
X = {4, 0, 0};
Y = {0, 4, 0};
XcrossY =
  Det[{{i, j, k}, X, Y}] /. {i → {1, 0, 0}, j → {0, 1, 0}, k → {0, 0, 1}}
{0, 0, 16}
YcrossX =
  Det[{{i, j, k}, Y, X}] /. {i → {1, 0, 0}, j → {0, 1, 0}, k → {0, 0, 1}}
{0, 0, -16}
```

Another:

```
X = {47.86, 87.12, 19.55};
Y = {27.18, 31.14, 57.00};
XcrossY =
 N[Det[{{i, j, k}, X, Y}] /. {i → {1, 0, 0}, j → {0, 1, 0}, k → {0, 0, 1}}]
```

```
{4357.05, -2196.65, -877.561}
```

```
YcrossX =
 N[Det[{{i, j, k}, Y, X}] /. {i → {1, 0, 0}, j → {0, 1, 0}, k → {0, 0, 1}}]
```

```
{-4357.05, 2196.65, 877.561}
```

If you went to a high school that taught terminology instead of calculation, then you'll see that the cross product defines a multiplication that is not commutative.

If you went to a high school that emphasized calculation, then you have a good chance of knowing what properties of determinants force the situation to be as it is.