Matrices, Geometry&*Mathematica*

Authors: Bruce Carpenter, Bill Davis and Jerry Uhl ©2001

Producer: Bruce Carpenter

Publisher: Math Everywhere, Inc.

MGM.01 Perpendicular Frames

BASICS

B.1) Vectors:

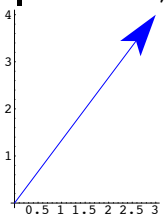
How you move them, how you add them, how you subtract them and how you multiply them by numbers. Length of a vector, dot product, linearity of dot products, and the distance between two points

This will be review for most of you.

□B.1.a.i) Playing with vectors

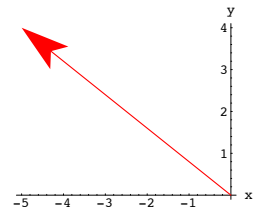
A vector X in 2D is a stick with:
→ one end (its tail) at {0, 0} and
→ the other end (its tip) at a specified location {x[1], x[2]}.
Most folks like to draw a vector as an arrow with the arrowhead at its tip.
Here's a look at the vector running from {0, 0} to {3, 4}:

```
X = {3, 4};
Show[Arrow[X, Tail → {0, 0}, VectorColor → Blue],
  Axes → True, PlotRange → All, AxesOrigin → {0, 0}];
```



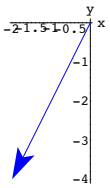
Most folks like to see the arrowhead at the tip so they can tell which end of the stick is the tip. Here is the vector running from
{0, 0} (tail) to {−5, 4} (tip):

```
X = {-5, 4};
Show[Arrow[X, VectorColor → Red], Axes → True,
  PlotRange → All, AxesLabel → {"x", "y"}, AxesOrigin → {0, 0}];
```



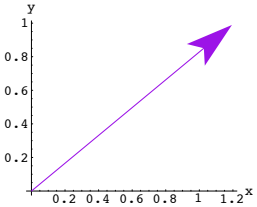
Here are some others:

```
X = {-2, -4};
Show[Arrow[X], Axes → True, PlotRange → All,
  AxesLabel → {"x", "y"}, AxesOrigin → {0, 0}];
```



See some random vectors in random colors

```
X = {Random[Real, {-2, 2}], Random[Real, {-2, 2}]};
randomcolor = RGBColor[Random[Real, {0, 1}],
                       Random[Real, {0, 1}], Random[Real, {0, 1}]];
Show[Arrow[X, Tail -> {0, 0}, VectorColor -> randomcolor], Axes → True,
  PlotRange → All, AxesLabel → {"x", "y"}, AxesOrigin → {0, 0}];
```

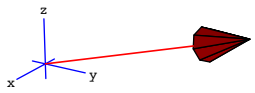


Rerun several times.

A vector
X = {x[1], x[2]}
is written the same as the coordinates of its tip because everyone knows its tail is at {0, 0}.

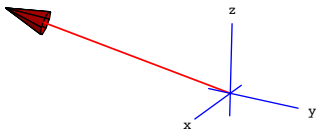
You can also work with three-dimensional vectors by writing
X = {x[1], x[2], x[3]}
for the vector whose tail is at {0, 0, 0} and whose tip is at
{x[1], x[2], x[3]}.
Here's the vector {1.5, 5.3, 2.0} shown along with the three-dimensional coordinate axes:

```
X = {1.5, 5.3, 2.0};
spacer = 0.2;
h = 1;

threedims = Axes3D[h, spacer];

Show[Arrow[X, VectorColor → Red], threedims,
  PlotRange → All, ViewPoint → CMView, Boxed → False];
```



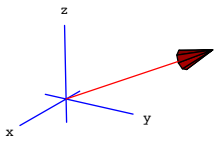
Here's another:

```
X = {1, -3, 1};
Show[Arrow[X, VectorColor → Red], threedims,
  PlotRange → All, ViewPoint → CMView, Boxed → False];
```



Here's a vector parallel to the yz-plane:

```
X = {0, 2, 1};
Show[Arrow[X, VectorColor → Red], threedims,
  PlotRange → All, ViewPoint → CMView, Boxed → False];
```



Play by rerunning with three-dimensional vectors of your own choice.

□B.1.a.i) Moving vectors

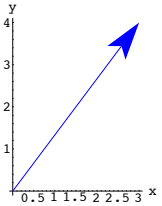
How do you move vectors to new positions?

□Answer:

Very easily.

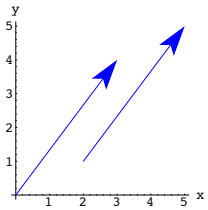
Here is a vector in two dimensions with its tail at {0, 0}:

```
X = {3, 4};
Show[Arrow[X, Tail → {0, 0}],
  AxesLabel → {"x", "y"}, Axes → True, AxesOrigin → {0, 0}];
```



Here's the same vector X shown twice, once with its tail at {0, 0} and once with its tail at {2, 1}:

```
X = {3, 4};
Show[Arrow[X, Tail → {0, 0}],
       Arrow[X, Tail → {2, 1}],
      Axes → True, AxesLabel → {"x", "y"}, PlotRange → All,
      AxesOrigin → {0, 0}];
```

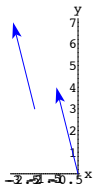


The two arrows have equal lengths and they both point in the same direction.

They're parallel.

Try it again for a different vector and a different tail:

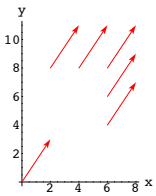
```
X = {-1, 4};
Show[Arrow[X, Tail → {0, 0}], Arrow[X, Tail → {-2, 3}],
     Axes → True, AxesLabel → {"x", "y"},
  PlotRange → All, AxesOrigin → {0, 0}];
```



Same length; same direction.

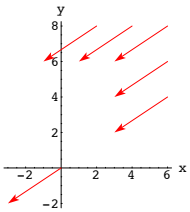
Here's a new vector X shown with its tail at {0, 0}, and shown with a whole squadron of its transplants:

```
X = {2, 3};
Clear[tail, k];
tail[1] = {0, 0};
tail[2] = {6, 4};
tail[3] = {6, 6};
tail[4] = {6, 8};
tail[5] = {4, 8};
tail[6] = {2, 8};

squadron =
  Table[Arrow[X, Tail → tail[k], VectorColor → Red], {k, 1, 6}];

Show[squadron, Axes → True,
  AxesLabel → {"x", "y"}, PlotRange → All, AxesOrigin → {0, 0}];
```



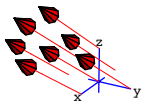
Check it out for a different X:

```
X = {-3, -2};
Clear[tail, k];
tail[1] = {0, 0};
tail[2] = {6, 4};
tail[3] = {6, 6};
tail[4] = {6, 8};
tail[5] = {4, 8};
tail[6] = {2, 8};

squadron =
  Table[Arrow[X, Tail → tail[k], VectorColor → Red], {k, 1, 6}];

Show[squadron, Axes → True,
  AxesLabel → {"x", "y"}, PlotRange → All, AxesOrigin → {0, 0}];
```



Check it out in three dimensions:

```
X = {3, 0, 2};
Clear[tail, k];
tail[1] = {0, 0, 0};
tail[2] = {1, 0, 0};
tail[3] = {-1, 0, 0};
tail[4] = {0, 1, 0};
tail[5] = {0, -1, 0};
tail[6] = {0, 0, 1};
tail[7] = {1, 0, 1};

squadron =
  Table[Arrow[X, Tail → tail[k], VectorColor → Red], {k, 1, 7}];

Show[squadron, threedims,
  PlotRange → All, ViewPoint → CMView, Boxed → False];
```



Same length; same direction.

□B.1.a.ii) Rules about moving vectors to new positions

Are there any rules about moving vectors to new positions?

□Answer:

You can put the tail anywhere you like, but you must be careful not to change the direction or the length of the vector.

□B.1.b.i) Adding vectors

How do you add vectors?

□Answer:

Very easily.

For instance, if X = {3, 8} and Y = {5, 4} are vectors, then you add them to get

X + Y = {3, 8} + {5, 4} = {8, 12}

just by adding the corresponding components.

Mathematica can do this too:

```
X = {3, 8};
Y = {5, 4};
X + Y
```

```
{8, 12}
```

You can add three-dimensional vectors:

```
X = {3, -5, 2};
Y = {3, 4, -7};
X + Y
```

```
{6, -1, -5}
```

You cannot add vectors from different dimensions:

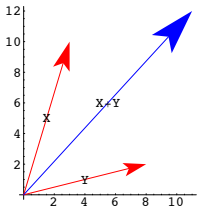
```
X = {3, -5};
Y = {3, 4, -7};
X + Y
```

```
Thread::tdlen : Objects of unequal length in {3, -5} + {3, 4, -7} cannot be combined.
{3, -5} + {3, 4, -7}
```

Here is a way of seeing what's happening in two dimensions:

Look at a picture of X = {3, 10} and Y = {8, 2} and X + Y:

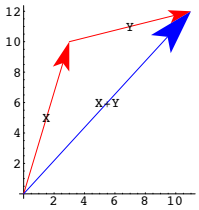
```
X = {3, 10};
Y = {8, 2};
Show[Arrow[X, Tail → {0, 0}, VectorColor → Red],
  Graphics[Text["X", X/2]],
  Arrow[Y, Tail → {0, 0}, VectorColor → Red], Graphics[Text["Y", Y/2]],
  Arrow[X + Y, Tail → {0, 0}, VectorColor → Blue],
  Graphics[Text["X+Y", (X + Y)/2]], Axes → Automatic];
```



X + Y represents the combined push of X and Y.

See what happens when you move Y without changing its direction, so that its tail is at the tip of X:

```
Show[Arrow[X, Tail → {0, 0}, VectorColor → Red],
  Graphics[Text["X", X/2]], Arrow[Y, Tail → X, VectorColor → Red],
  Graphics[Text["Y", X + Y/2]],
  Arrow[X + Y, Tail → {0, 0}, VectorColor → Blue],
  Graphics[Text["X+Y", (X + Y)/2]], Axes → Automatic];
```



A triangle!

This triangle shows that you have your choice of ways of getting from {0, 0} to the tip of X + Y.

→ Route 1: You can walk directly on the vector X + Y from its tail to its tip.

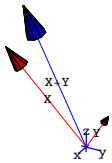
→ Route 2: You can walk on X to the tip of X, and then hook the tail of Y on the tip of X, and finish the trip by walking along Y to its tip.

Lazy folks usually take Route 1.

Rerun this for two-dimensional vectors of your own choice.

Now check out what happens in three dimensions:

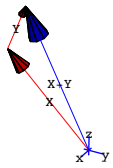
```
X = {3, -4, 7};
Y = {1, 2, 3};

Show[Arrow[X, Tail → {0, 0, 0}, VectorColor → Red],
  Graphics3D[Text["X", X/2]], Arrow[Y, Tail → {0, 0, 0},
   VectorColor → Red], Graphics3D[Text["Y", Y/2]],
  Arrow[X + Y, Tail → {0, 0, 0}, VectorColor → Blue],
  Graphics3D[Text["X+Y", (X + Y)/2]], threedims,
  PlotRange → All, ViewPoint → CMView, Boxed → False];
```



Again, in three dimensions, X + Y represents the combined push of X and Y.

See what happens when you move Y without changing its direction so that its tail is at the tip of X:

```
Show[Arrow[X, Tail → {0, 0, 0}, VectorColor → Red],
  Graphics3D[Text["X", X/2]], Arrow[Y, Tail → X, VectorColor → Red],
  Graphics3D[Text["Y", X + Y/2]],
  Arrow[X + Y, Tail → {0, 0, 0}, VectorColor → Blue],
  Graphics3D[Text["X+Y", (X + Y)/2]], threedims,
  PlotRange → All, ViewPoint → CMView, Boxed → False];
```



Another triangle.

Just as in two dimensions, this triangle shows that you have your choice of ways of getting from {0, 0} to the tip of X + Y.

→ Route 1: You can walk directly on the vector X + Y from its tail to its tip.

→ Route 2: You can walk on X to the tip of X, and then hook the tail of Y on the tip of X, and finish the trip by walking along Y to its tip.

Rerun the pictures above for different vectors X and Y until you get the hang of vector addition.

□B.1.b.ii) Subtracting vectors

How do you subtract vectors?

□Answer:

With no trouble.

For instance, if X = {8, 2} and Y = {3, 4} are vectors, then you subtract Y from X to get

X − Y = {8, 2} − {3, 4} = {5, −2}.

Mathematica can do this too:

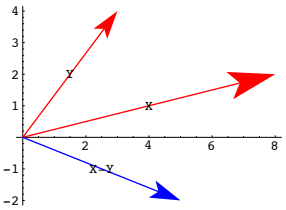
```
X = {8, 2};
Y = {3, 4};

X - Y
```

```
{5, -2}
```

Here is a way of seeing what's happening.

Look at a picture of X = {8, 2}, Y = {3, 4}, and X − Y:

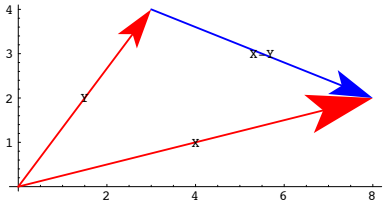
```
X = {8, 2};
Y = {3, 4};

Show[Arrow[X, Tail → {0, 0}, VectorColor → Red],
  Graphics[Text["X", X/2]],
  Arrow[Y, Tail → {0, 0}, VectorColor → Red], Graphics[Text["Y", Y/2]],
  Arrow[X - Y, Tail → {0, 0}, VectorColor → Blue],
  Graphics[Text["X-Y", (X - Y)/2]], Axes → Automatic];
```



This time X is the combined push of Y and X − Y.

See what happens when you move X − Y without changing its direction so that its tail is at the tip of Y:

```
Show[Arrow[X, Tail → {0, 0}, VectorColor → Red],
  Graphics[Text["X", X/2]], Arrow[Y, Tail → {0, 0}, VectorColor → Red],
  Graphics[Text["Y", Y/2]], Arrow[X - Y, Tail → Y, VectorColor → Blue],
  Graphics[Text["X-Y", Y + (X - Y)/2]], Axes → Automatic];
```



Another triangle.

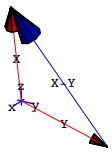
You can get to the tip of X by going directly along X, or you can go from the tail of Y to the tip of Y, and then ride on X − Y to the tip of X.

This is not a surprise because X = (X − Y) + Y.

Rerun for some other vectors of your own choice.

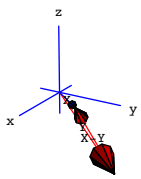
Here it is in three dimensions:

```
X = {4, 2, 8};
Y = {-2, 6, -3};

spacer = 0.2;
h = 1;
threedims = Axes3D[h, spacer];
CMView = {2.7, 1.6, 1.2};

Show[Arrow[X, Tail → {0, 0, 0}, VectorColor → Red],
    Graphics3D[Text["X", X/2]],
    Arrow[Y, Tail → {0, 0, 0}, VectorColor → Red],
    Graphics3D[Text["Y", Y/2]],
    Arrow[X - Y, Tail → Y, VectorColor → Blue],
    Graphics3D[Text["X-Y", Y + (X - Y)/2]],
    threedims, PlotRange → All, ViewPoint → CMView, Boxed → False];
```



Here it is for two random vectors in three dimensions:

```
X = 2 {Random[], Random[], Random[]};
Y = 2 {Random[], Random[], Random[]};

spacer = 0.2;
h = 1;
threedims = Axes3D[h, spacer];
CMView = {2.7, 1.6, 1.2};

Show[Arrow[X, Tail → {0, 0, 0}, VectorColor → Blue],
    Graphics3D[Text["X", X/2]],
    Arrow[Y, Tail → {0, 0, 0}, VectorColor → Red],
    Graphics3D[Text["Y", Y/2]],
    Arrow[X - Y, Tail → Y, VectorColor → Red],
    Graphics3D[Text["X-Y", Y + (X - Y)/2]],
    threedims, PlotRange → All, ViewPoint → CMView, Boxed → False];
```



Rerun for other X and Y of your own choice until you get the picture down pat.

□B.1.c) Multiplying vectors by numbers

```
Numbers are what some fancy folks call "scalars."
```

How do you multiply vectors by numbers?

□Answer:

You just do it.

For instance, if X = {3, 1} , then 2 X = {6, 2}.

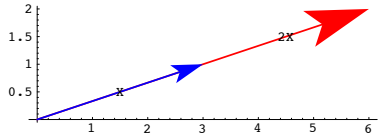
Mathematica can do this too:

```
X = {3, 1};
2 X
```

```
{6, 2}
```

Here are X and 2 X both shown with their tails at {0, 0}:

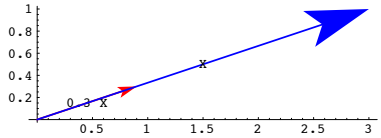
```
X = {3, 1};
Show[Arrow[2 X, Tail → {0, 0}, VectorColor → Red],
        Graphics[Text["2X", 3 X/2]],
        Arrow[X, Tail → {0, 0}, VectorColor → Blue],
        Graphics[Text["X", X/2]], Axes → Automatic];
```



2 X points the same direction as X, but 2 X is twice as long as X. That makes some sense because 2 X is the same thing as X + X.

Here are X and 0.3 X both shown with their tails at {0, 0}:

```
Show[Arrow[0.3 X, Tail → {0, 0}, VectorColor → Red],
        Graphics[Text["0.3 X", 0.3 X/2]],
        Arrow[X, Tail → {0, 0}, VectorColor → Blue],
        Graphics[Text["X", X/2]], Axes → Automatic];
```



0.3 X points in the same direction as X points.

The only difference is that the length of 0.3 X is 0.3 times the length of X.

The same idea carries over to three dimensions as well.

□B.1.d.i) Dot products

Here are two cleared vectors
X = {x[1], x[2]} and Y = {y[1], y[2]}
in two dimensions:

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

```
{x[1], x[2]}
```

```
Y = {y[1], y[2]}
```

```
{y[1], y[2]}
```

Here is the dot product, X.Y, of these vectors:

```
X.Y
```

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

Here are two vectors
X = {x[1], x[2], x[3]} and Y = {y[1], y[2], y[3]}
in three dimensions:

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

```
{x[1], x[2], x[3]}
```

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

```
{y[1], y[2], y[3]}
```

Here is the dot product, X.Y, of the three-dimensional vectors X and Y:

```
X.Y
```

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

Describe how dot products are calculated.

□Answer:

In two or three dimensions, the dot product X.Y just multiplies each slot in X by the corresponding slot in Y, and then adds them up.

Check it out:

```
{1, 2}.{0, 1}
```

```
2
```

```
{1, 2}.{0, 1} == 1 0 + 2 1
```

```
True
```

```
{-1, 1}.{8, 2}
```

```
-6
```

```
{-1, 1}.{8, 2} == -8 + 2
```

```
True
```

```
{1, 0, 1}.{1, 2, 3}
```

```
4
```

```
{1, 0, 1}.{1, 2, 3} == 1 + 0 + 3
```

```
True
```

You cannot take the dot product of vectors from different dimensions:

```
X = {1, 0};
Y = {2, 1, 4};
X.Y
```

```
Dot::dotsh : Tensors {1, 0} and {2, 1, 4} have incompatible shapes.
{1, 0}.{2, 1, 4}
```

That hacked off *Mathematica*.

□B.1.d.ii) Linearity of dot products:

If t is a number and X and Y are vectors, then (t X).Y = t (X.Y)

Here's a random 2D vector X:

```
X = {Random[Real, {-10, 10}], Random[Real, {-10, 10}]}
```

```
{8.24276, 7.68267}
```

And another random 2D vector Y:

```
Y = {Random[Real, {-10, 10}], Random[Real, {-10, 10}]}
```
```
{-8.74066, -1.77902}
```

Here's a random number t:

```
t = Random[Real, {-10, 10}]
```
```
-2.81018
```

Look at this calculation of (t X).Y:

```
(t X).Y
```
```
240.874
```

Look at this calculation of t (X.Y)::

```
t (X.Y)
```
```
240.874
```

They are the same.
Explain why this will happen for any vectors X and Y and any number t.

□ Answer:

This is pure bean counting

Go with two cleared vectors

X = {x[1],x[2]} and Y = {y[1],y[2]} and a cleared number t.

Calculate:

$t\,(X.Y) = t\,(x[1]\,y[1] + x[2]\,y[2]) = t\,x[1]\,y[1] + t\,x[2]\,y[2]$

$(t\,X).Y = \{t\,x[1], t\,x[2]\}.\{y[1], y[2]\} = t\,x[1]\,y[1] + t\,x[2]\,y[2].$

The upshot:

$t\,(X.Y) = t\,(X.Y)$

is a mathematical certainty.

The same explanation (with more typing) also works in 3D.

Let *Mathematica* do it for you.

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

Expand[t (X.Y)]
```
```
t x[1] y[1] + t x[2] y[2] + t x[3] y[3]
```
```
Expand[(t X).Y]
```
```
t x[1] y[1] + t x[2] y[2] + t x[3] y[3]
```

They are the same.

□ B.1.d.iii) More Linearity of dot products:

If X,Y and Z are vectors, then (X + Y).Z = X.Z + Y.Z

Here's a random 2D vector X:

```
X = {Random[Real, {-10, 10}], Random[Real, {-10, 10}]}
```
```
{-2.85447, -6.92796}
```

And another random 2D vector Y:

```
Y = {Random[Real, {-10, 10}], Random[Real, {-10, 10}]}
```
```
{-9.41601, -7.62906}
```

And another random 2D vector Z:

```
Z = {Random[Real, {-10, 10}], Random[Real, {-10, 10}]}
```
```
{-2.00641, -2.06031}
```

Look at this calculation of (X + Y).Z :

```
(X + Y).Z
```
```
54.6115
```

Look at this calculation of X.Z + Y.Z::

```
X.Z + Y.Z
```
```
54.6115
```

They are the same.
Explain why this will happen for any three vectors X , Y and Z.

□ Answer:

This is pure bean counting

Go with three cleared vectors

X = {x[1],x[2]} , Y = {y[1],y[2]} and Z = {z[1],z[2]}

Calculate:

$(X + Y).Z = \{x[1] + y[1], x[2] + y[2]\} + x[2]\,y[2]).\{z[1], z[2]\}$
$= (x[1] + y[1])\,z[1] + (x[2] + y[2])\,z[2]$
$= x[1]\,z[1] + y[1)\,z[1] + x[2]\,z[2] + y[2]\,z[2].$

And calculate:

$X.Z + Y.Z: = x[1]\,z[1] + x[2]\,z[2] + y[1]\,z[1] + y[2]\,z[2].$

The upshot:

$(X + Y).Z = X.Z + Y.Z$

is a mathematical certainty.

The same explanation (with more typing) also works in 3D.

Let *Mathematica* do it for you:

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

They are the same.

Enough bean counting.

□ B.1.e.i) The length of a vector X is $\sqrt{X.X}$

Given a two-dimensional vector,
X = {x[1], x[2]},
the length of X is measured by
$\|X\| = \sqrt{x[1]^2 + x[2]^2}$.

Given a three-dimensional vector,
X = {x[1], x[2], x[3]},
the length of X is measured by
$\|X\| = \sqrt{x[1]^2 + x[2]^2 + x[3]^2}$.
Explain why the formula
$\|X\| = \sqrt{X.X}$
works in either dimension.

□ Answer:

Try it out in two dimensions:

```
Clear[x];
X = {x[1], x[2]};
length = Sqrt[x[1]^2 + x[2]^2]
```
$\sqrt{x[1]^2 + x[2]^2}$

$\sqrt{X.X}$ is given by:

```
Sqrt[x.x]
```
$\sqrt{x[1]^2 + x[2]^2}$

This tells you that the formula
$\|X\| = \sqrt{X.X}$ works in two dimensions.

Try it out in three dimensions:

```
Clear[x];
X = {x[1], x[2], x[3]};
length = Sqrt[x[1]^2 + x[2]^2 + x[3]^2]
```
$\sqrt{x[1]^2 + x[2]^2 + x[3]^2}$

$\sqrt{X.X}$ is given by:

```
Sqrt[x.x]
```
$\sqrt{x[1]^2 + x[2]^2 + x[3]^2}$

This tells you that the formula
$\|X\| = \sqrt{X.X}$ works in three dimensions too.

Handy little formula

And it's nothing more or less than your old friend, the Pythagorean theorem, in action.

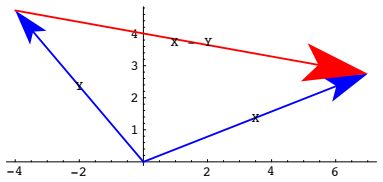
□ B.1.e.ii) Dot products and distance between points

Why does
$\|X - Y\| = \sqrt{(X - Y).(X - Y)}$
calculate the distance between the tip of X and the tip of Y when X and Y are positioned so that their tails are at the origin?

□ Answer:

Look at a picture:

```
X = {7, Random[Real, {1, 5}]};
Y = {-4, Random[Real, {2, 8}]};

Show[Arrow[X, Tail → {0, 0}, VectorColor → Blue],
    Graphics[Text["X", X/2]],
    Arrow[Y, Tail → {0, 0}, VectorColor → Blue],
    Graphics[Text["Y", Y/2]],
    Arrow[X - Y, Tail → Y, VectorColor → Red],
    Graphics[Text["X - Y", Y + (X - Y)/2]], Axes → True];
```



Rerun a couple of times.

The pictures tell you that the distance between the tip of X and the tip of Y is the same as the length of X − Y.

The length of X − Y is

$\sqrt{(X-Y).(X-Y)}$,

so the distance between the tip of X and the tip of Y is given by

$\|X-Y\| = \sqrt{(X-Y).(X-Y)}$.

That's all there is to it.

□B.1.e.iii) Measuring some distances between points

Measure the distance between {2.2, 7.8} and {3.0, −5.4} in 2D.
Measure the distance between {2.2, 7.8, −6.2} and {3.0, −5.4, 5.8} in 3D.

□Answer:

You measure the distance between the points {2, 7} and {3, −5} in 2D by running:

```
X = {2.2, 7.8};
Y = {3.0, -5.4};
√(X - Y) . (X - Y)
```

```
13.2242
```

You measure distance between the points {2.2, 7.8, −6.2} and {3.0, −5.4, 5.8} in 3D by running:

```
X = {2.2, 7.8, -6.2};
Y = {3.0, -5.4, 5.8};
√(X - Y) . (X - Y)
```

```
17.8572
```

B.2) 2D Perpendicular frames:

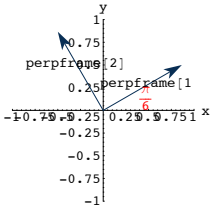
Hanging curves on 2D perpendicular frames

□B.2.a) Perpendicular frames in 2D

A 2D perpendicular frame consists of two perpendicular unit vectors which you set by specifying and angle s.
Here's one:

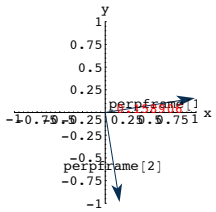
A unit vector is s vector whose length is 1.
A perpendicular frame is any pair of perpendicular unit vectors.
You can dial up one by setting s and then putting
perpframe[1] = {Cos[s], Sin[s]}
and
perpframe[2] ={Cos[s + $\frac{\pi}{2}$], Sin[s + $\frac{\pi}{2}$]}

```
s = π/6;
Clear[perpframe];
{perpframe[1], perpframe[2]} =
  {{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}};

Show[Table[Arrow[perpframe[k], Tail → {0, 0},
    VectorColor → Indigo, HeadSize → 0.2], {k, 1, 2}],
  Graphics[Text["perpframe[1]", 0.6 perpframe[1]]],
  Graphics[Text["perpframe[2]", 0.6 perpframe[2]]],
  Graphics[{Red, Text[s, 0.25 ({1, 0} + perpframe[1])]}], Axes → True,
  AxesLabel → {"x", "y"}, PlotRange → {{-1, 1}, {-1, 1}}];
```



s (= $\frac{\pi}{6}$ in this plot) is the counterclockwise angle measured from the positive x-axis to perpframe[1]
See a random 2D perpendicular frame.

```
s = Random[Real, {-π/4, π/4}];
Clear[perpframe];
{perpframe[1], perpframe[2]} =
  {{Cos[s], Sin[s]}, (-1)^Random[Integer,{0,1}] {Cos[s + π/2], Sin[s + π/2]}};

Show[Table[Arrow[perpframe[k], Tail → {0, 0},
    VectorColor → Indigo, HeadSize → 0.2], {k, 1, 2}],
  Graphics[Text["perpframe[1]", 0.6 perpframe[1]]],
  Graphics[Text["perpframe[2]", 0.6 perpframe[2]]],
  Graphics[{Red, Text[s, 0.25 ({1, 0} + perpframe[1])]}], Axes → True,
  AxesLabel → {"x", "y"}, PlotRange → {{-1, 1}, {-1, 1}}];
```

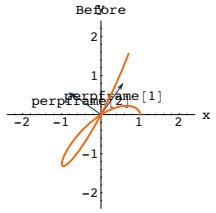


Rerun a couple of times

□B.2.b.i) Hanging a curve on a perpendicular frame

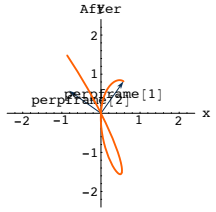
Here's a curve shown together with a perpendicular frame:

```
Clear[x, y, t];
{x[t_], y[t_]} = {Cos[t], 0.4 t Cos[t]};
ranger = 2.4;
{tlow, thigh} = {0, 5.5};
curveplot = ParametricPlot[{x[t], y[t]}, {t, tlow, thigh},
  PlotStyle → {{CadmiumOrange, Thickness[0.01]}},
  DisplayFunction → Identity];
s = Random[Real, {π/4, π/3}];

Clear[perpframe];
{perpframe[1], perpframe[2]} =
  {{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}};

frameplot = {Table[Arrow[perpframe[k], Tail → {0, 0},
    VectorColor → Indigo, HeadSize → 0.2], {k, 1, 2}],
  Graphics[Text["perpframe[1]", 0.6 perpframe[1]]],
  Graphics[Text["perpframe[2]", 0.6 perpframe[2]]]};

before = Show[frameplot, curveplot,
  PlotRange → {{-ranger, ranger}, {-ranger, ranger}},
  Axes → True, AxesLabel → {"x", "y"}, PlotLabel → "Before"];
```



Now look at this:

```
newcurveplot = ParametricPlot[x[t] perpframe[1] + y[t] perpframe[2],
  {t, tlow, thigh}, PlotStyle → {{CadmiumOrange, Thickness[0.01]}},
  DisplayFunction → Identity];
after = Show[frameplot, newcurveplot,
  PlotRange → {{-ranger, ranger}, {-ranger, ranger}},
  Axes → True, AxesLabel → {"x", "y"}, PlotLabel → "After"];
```



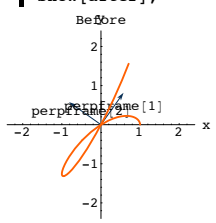
Grab both plots and animate slowly.

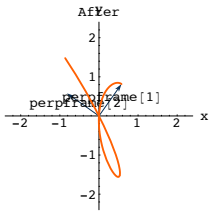
Describe what you see and explain why you see it.

□Answer:

Take another look:

```
Show[before];
Show[after];
```





Grab both plots and animate slowly.

In the second plot, you see the same physical curve as in the second plot.

But in the second plot, the curve is hanging the plotted perpendicular frame with

perpframe[1] playing the former role of {1, 0} pointing out of the positive x-axis

and with

perpframe[2] playing the former role of {0, 1} pointing out the positive y-axis.

To explain why this happened, look at the formula plotted for the first curve:

$$\{x[t],y[t]\} = x[t]\ \{1,0\} + y[t]\ \{0,1\}\ .$$

This means that to get to a point {x[t],y[t]} on the original curve,
you advance x[t] units in the direction of {1,0}
and then you adavnce y[t] units in the direction of {0,1}.

The formula plotted for the second curve was

$$x[t]\ \text{perpeframe[1]} + y[t]\ \text{perpframe[2]}\ .$$

This means that to get to a point on the original curve,
you advance x[t] units in the direction of perpframe[1]
and then you adavnce y[t] units in the direction of perpframe[2]..

The second curve is physically the same as the first curve except the second curve is now hanging on the plotted perpendicular frame

perpframe[1] stepping in to replace {1,0}

and

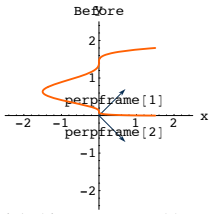
perpframe[2] stepping in to replace {1,0}

□B.2.a.ii) Another sample of hanging a curve on a perpendicular frame

Here's a new curve shown together with a perpendicular frame:

```
Clear[x, y, t];

{x[t_], y[t_]} = {1.5 Cos[t]^3, 1.8 Sin[t/4]^3};

ranger = 2.5;
{tlow, thigh} = {0, 2 π};
```

```
curveplot = ParametricPlot[{x[t], y[t]}, {t, tlow, thigh},
   PlotStyle → {{CadmiumOrange, Thickness[0.01]}},
   DisplayFunction → Identity];
s = π/4;

Clear[perpframe];
{perpframe[1], perpframe[2]} =
  {{Cos[s], Sin[s]}, -{Cos[s + π/2], Sin[s + π/2]}};

frameplot = {Table[Arrow[perpframe[k], Tail → {0, 0},
     VectorColor → Indigo, HeadSize → 0.2], {k, 1, 2}],
   Graphics[Text["perpframe[1]", 0.6 perpframe[1]]],
   Graphics[Text["perpframe[2]", 0.6 perpframe[2]]]};

before = Show[frameplot, curveplot,
   PlotRange → {{-ranger, ranger}, {-ranger, ranger}},
   Axes → True, AxesLabel → {"x", "y"}, PlotLabel → "Before"];
```



Pick this curve up and hang it on the plotted frame with

perpframe[1] playing the former role {1, 0} pointing out of the positive x-axis

and with

perpframe[2] playing the former role {0, 1} pointing out the positive y-axis.

□Answer:

You just take the xy- parameterization

$$\{x[t], y[t]\} = x[t]\ \{1, 0\} + y[t]\ \{0, 1\},$$

and change {1, 0} to perpframe[1] and change {0, 1} to perpframe[2]

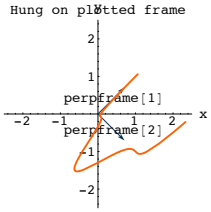
to get

$$x[t]\ \text{perpframe[1]} + y[t]\ \text{perpframe[2]}.$$

And plot:

```
hungcurveplot = ParametricPlot[x[t] perpframe[1] + y[t] perpframe[2],
  {t, tlow, thigh}, PlotStyle → {{CadmiumOrange, Thickness[0.01]}},
  DisplayFunction → Identity];
```

```
after = Show[frameplot, hungcurveplot,
   PlotRange → {{-ranger, ranger}, {-ranger, ranger}}, Axes → True,
   AxesLabel → {"x", "y"}, PlotLabel → "Hung on plotted frame"];
```



Grab both plots and animate.

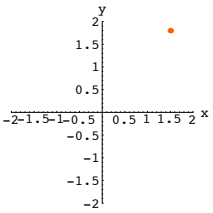
There you go.

B.3) Resolution of 2D vectors into perpendicular components; perpendicular frame coordinates

□B.3.a.i) Getting to the point

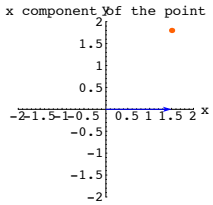
Here's a point sitting contentedly in the xy plane:

```
point = {1.5, 1.8};
setup =
  Show[Graphics[{CadmiumOrange, PointSize[0.03], Point[point]}],
   Axes → True, AxesLabel → {"x", "y"}, PlotRange → {{-2, 2}, {-2, 2}}];
```



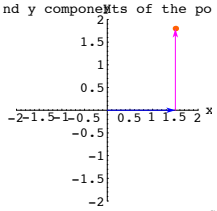
One way to get to the plotted point from {0,0} is to advance on the x axis until you are directly under the point:

```
xunitvector = {1, 0};
xsetup = Show[setup, Arrow[(point.xunitvector) xunitvector ,
   Tail -> {0, 0}, VectorColor -> Blue, HeadSize -> 0.2],
  PlotLabel -> "x component of the point"];
```



And then advance parallel to the y axis until you get to the point:

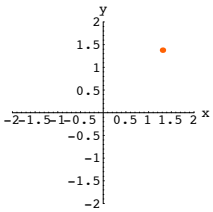
```
yunitvector = {0, 1};
xysetup = Show[xsetup, Arrow[(point.yunitvector) yunitvector ,
   Tail -> (point.xunitvector) xunitvector, VectorColor -> Magenta,
   HeadSize -> 0.2], PlotLabel -> "x and y components of the point"];
```



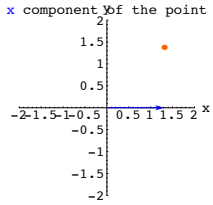
Grab both plots and animate slowly.

Do some more:

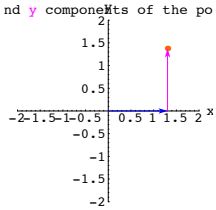
```
point = {Random[Real, {-2, 2}], Random[Real, {1, 2}]};
setup =
  Show[Graphics[{CadmiumOrange, PointSize[0.03], Point[point]}],
   Axes → True, AxesLabel → {"x", "y"}, PlotRange → {{-2, 2}, {-2, 2}}];
xsetup = Show[setup, Arrow[point[[1]] {1, 0},
   Tail -> {0, 0}, VectorColor -> Blue, HeadSize -> 0.2],
  PlotLabel -> "x component of the point"];

xysetup = Show[xsetup,
  Arrow[point[[2]] {0, 1}, Tail -> point[[1]] {1, 0},
   VectorColor -> Magenta, HeadSize -> 0.2],
  PlotLabel -> "x and y components of the point"];
```



x component of the point



nd y components of the po



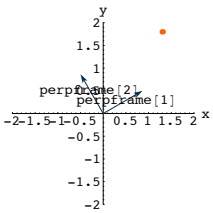
Grab all three plots and animate and then rerun a couple of times.

Now look at this perpendicular frame and this point:

A pependicular frame is any pair of perpendicular unit vectors.
You can dial up one by setting s and then putting

perpframe[1] = {Cos[s], Sin[s]}
and
perpframe[2] = {Cos[s + $\frac{\pi}{2}$], Sin[s + $\frac{\pi}{2}$]}

```
s = π/6;
Clear[perpframe];
{perpframe[1], perpframe[2]} =
  {{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}};
point = {1.3, 1.8};
setup = Show[Table[Arrow[perpframe[k], Tail → {0, 0},
     VectorColor → Indigo, HeadSize → 0.2], {k, 1, 2}],
   Graphics[Text["perpframe[1]", 0.6 perpframe[1]]],
   Graphics[Text["perpframe[2]", 0.6 perpframe[2]]],
   Graphics[{CadmiumOrange, PointSize[0.03], Point[point]}],
   Axes → True, AxesLabel → {"x", "y"}, PlotRange → {{-2, 2}, {-2, 2}}];
```



Your eyes tell you that you can get to the plotted point by
→ advancing from {0,0} in the direction of perpframe[1],
stopping and then
→ advancing parallel to perpframe[2] until you get to the plotted point.
Just how do you do it?

□ **Answer:**

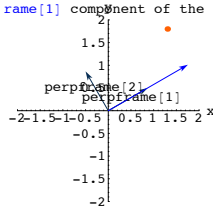
Very easily.

You advance

→ (point •perpframe[1]) units in the direction of perpframe[1]:

See it happen:

```
perp1component = Show[setup,
  Arrow[(point.perpframe[1]) perpframe[1], Tail -> {0, 0},
    VectorColor -> Blue, HeadSize -> 0.2],
  PlotLabel -> "Perpframe[1] component of the point"];
```

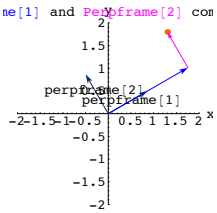


And then you stop and then advance

→ (point •perpframe[2]) units in the direction of perpframe[2].

See it happen:

```
both = Show[perp1component, Arrow[(point.perpframe[2]) perpframe[2],
   Tail -> (point.perpframe[1]) perpframe[1], VectorColor -> Magenta],
  PlotLabel -> "Perpframe[1] and Perpframe[2] components"];
```

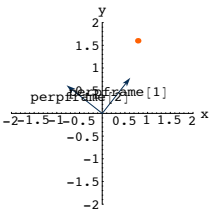


Grab all three plots and animate and then rerun a couple of times.

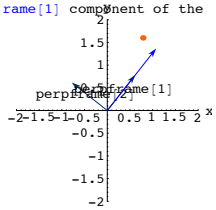
Nailed it.

Do some more, going with random plotted points and random perpendicular frames

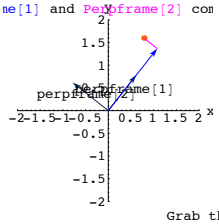
```
point = {Random[Real, {-2, 2}], 1.6};
s = Random[Real, {π/4, 3 π/8}];
Clear[perpframe];
{perpframe[1], perpframe[2]} =
  {{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}};
setup = Show[Table[Arrow[perpframe[k], Tail → {0, 0},
     VectorColor → Indigo, HeadSize → 0.2], {k, 1, 2}],
   Graphics[Text["perpframe[1]", 0.6 perpframe[1]]],
   Graphics[Text["perpframe[2]", 0.6 perpframe[2]]],
   Graphics[{CadmiumOrange, PointSize[0.03], Point[point]}],
   Axes → True, AxesLabel → {"x", "y"}, PlotRange → {{-2, 2}, {-2, 2}}];
perp1component = Show[setup,
  Arrow[(point.perpframe[1]) perpframe[1], Tail -> {0, 0},
    VectorColor -> Blue, HeadSize -> 0.2],
  PlotLabel -> "Perpframe[1] component of the point"];
both = Show[perp1component, Arrow[(point.perpframe[2]) perpframe[2],
   Tail -> (point.perpframe[1]) perpframe[1], VectorColor -> Magenta],
  PlotLabel -> "Perpframe[1] and Perpframe[2] components"];
```



rame[1] component of the



me[1] and Perpframe[2] com



Grab the last three plots and animate slowly.
Then rerun several times.

Each time you advance

→ (point •perpframe[1]) units in the direction of perpframe[1]

and then you stop and then advance

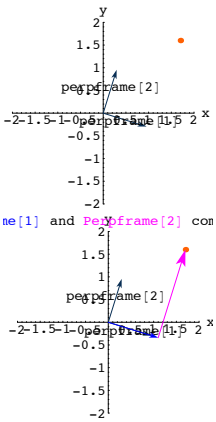
→ (point •perpframe[2]) units in the direction of perpframe[2].

□ **B.3.a.ii) Resolution into perpendicular components and coordinates of a point relative to a perpendicular frame**

Review what you've been up to:

```
point = {1.7, 1.6};
s = -0.3;
```

```
Clear[perpframe];
{perpframe[1], perpframe[2]} =
  {{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}};
setup = Show[Table[Arrow[perpframe[k], Tail → {0, 0},
     VectorColor → Indigo, HeadSize → 0.2], {k, 1, 2}],
   Graphics[Text["perpframe[1]", 0.6 perpframe[1]]],
   Graphics[Text["perpframe[2]", 0.6 perpframe[2]]],
   Graphics[{CadmiumOrange, PointSize[0.03], Point[point]}],
   Axes → True, AxesLabel → {"x", "y"}, PlotRange → {{-2, 2}, {-2, 2}}];

both =
 Show[setup, Arrow[(point.perpframe[1]) perpframe[1], Tail -> {0, 0},
                   VectorColor -> Blue, HeadSize -> 0.2],
  Arrow[(point.perpframe[2]) perpframe[2],
   Tail -> (point.perpframe[1]) perpframe[1], VectorColor -> Magenta],
  PlotLabel -> "Perpframe[1] and Perpframe[2] components"];
```



The blue vector shooting out perpframe[1] is (point.perpframe[1]) perpframe[1]
The magenta vector shooting out parallel to perpframe[2] is (point.perpframe[2]) perpframe[2],

Folks call these vectors by the name perpendicular frame components of the given point.

Why do the same folks call
{point.perpframe[1] , point.perpframe[2]}

the coordinates of the given point relative to the given perpendicular frame?

□Answer:

Because when you advance
→ (point •perpframe[1]) units in the direction of perpframe[1]
and then you stop and then advance
→ (point •perpframe[2]) units in the direction of perpframe[2],
you get to the given point.

□B.3.a.iii) A sample

Here is a random point:

```
ranger = 3;
point =
 {Random[Real, {-ranger , ranger }], Random[Real, {-ranger , ranger }]}
```

```
{-0.104366, -1.18358}
```

And a random perpendicular frame:

```
Clear[perpframe];
s = Random[Real, {-π, π}];
{perpframe[1], perpframe[2]} =
  {{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}}
```

```
{{-0.201078, 0.979575}, {-0.979575, -0.201078}}
```

Calculate the coordinates of the given point relative to the given perpendicular frame.

Then resolve the point into two perpendicular components with one parallel to perpframe[1] and the other parallel to perpframe[2].

□Answer:

The coordinates of the given point relative to the given perpendicular frame are:

```
{point.perpframe[1] , point.perpframe[2]}
```

```
{-1.13842, 0.340226}
```

The component of the point in the direction of perpframe[1] is:

```
Clear[perpcomp];
perpcomp[1] = (point.perpframe[1]) perpframe[1]
```

```
{0.228911, -1.11517}
```

The component of the point in the direction of perpframe[2] is:

```
perpcomp[2] = (point.perpframe[2]) perpframe[2]
```

```
{-0.333277, -0.068412}
```

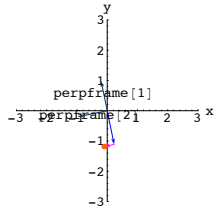
When you add perpcomp[1] and perpcomp[2], you get the given point:

```
perpcomp[1] + perpcomp[2] == point
```

```
True
```

See it:

```
Show[Table[Arrow[perpframe[k], Tail → {0, 0},
    VectorColor → Indigo, HeadSize → 0.2], {k, 1, 2}],
  Arrow[perpcomp[1], Tail -> {0, 0}, VectorColor -> Blue,
   HeadSize -> 0.2],
  Arrow[perpcomp[2], Tail -> perpcomp[1], VectorColor -> Magenta,
   HeadSize -> 0.2], Graphics[Text["perpframe[1]", 0.6 perpframe[1]]],
  Graphics[Text["perpframe[2]", 0.6 perpframe[2]]],
  Graphics[{CadmiumOrange, PointSize[0.03], Point[point]}],
  Axes → True, AxesLabel → {"x", "y"},
  PlotRange → {{-ranger, ranger}, {-ranger, ranger}}];
```



That's all there is to it.

□B.3.a.iv) Why it works

Why did all this work?

□Answer:

Enter cleared perpendicular frame:

```
Clear[perpframe, s, x, y];
{perpframe[1], perpframe[2]} =
  {{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}}
```

```
{{Cos[s], Sin[s]}, {-Sin[s], Cos[s]}}
```

Now enter a cleared point {x,y} and calculate

(point . perpframe[1]) perpframe[1] + (point . perpframe[2]) perpframe[2]

```
point = {x, y};
calculation = (point.perpframe[1]) perpframe[1] +
   (point.perpframe[2]) perpframe[2]
```

```
{-Sin[s] (y Cos[s] - x Sin[s]) + Cos[s] (x Cos[s] + y Sin[s]),
 Cos[s] (y Cos[s] - x Sin[s]) + Sin[s] (x Cos[s] + y Sin[s])}
```

Apply trig identities:

```
TrigExpand[(point.perpframe[1]) perpframe[1] +
   (point.perpframe[2]) perpframe[2]]
```

```
{x, y}
```

This tells you that when you take any old point {x,y} and any old perpframe and then advance
→ (point •perpframe[1]) units in the direction of perpframe[1]
and then you stop and then advance
→ (point •perpframe[2]) units in the direction of perpframe[2].,
you will always get to the point {x,y}.

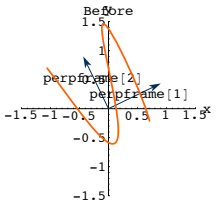
B.4) Using a perpendicular frame coordinates to align a 2D curve on the x-y axes

□B.) Using a perpendicular frame to align a curve on the x and y axes

Here's a curve {x[t],y[t]} shown with a perpendicular frame:
The formula for the curve is not an issue here.

```
Clear[perpframe];
s = 0.45;
{perpframe[1], perpframe[2]} =
  {{Cos[s], Sin[s]}, {-Sin[s], Cos[s]}};

Clear[x, y, t];
{x[t_], y[t_]} =
  {0.7 (1 - Cos[t]) - 0.4 Sin[3 t] - 1, 0.3 (1 - Cos[t]) + 0.9 Sin[3 t]};
ranger = 1.5;
{tlow, thigh} = {0.8, 3.5};

curveplot = ParametricPlot[{x[t], y[t]}, {t, tlow, thigh},
   PlotStyle → {{CadmiumOrange, Thickness[0.01]}},
   DisplayFunction → Identity];

frameplot = {Table[Arrow[perpframe[k], Tail → {0, 0},
     VectorColor → Indigo, HeadSize → 0.2], {k, 1, 2}],
   Graphics[Text["perpframe[1]", 0.6 perpframe[1]]],
   Graphics[Text["perpframe[2]", 0.6 perpframe[2]]]};
```

```
before = Show[frameplot, curveplot,
   PlotRange → {{-ranger, ranger}, {-ranger, ranger}},
   Axes → True, AxesLabel → {"x", "y"}, PlotLabel → "Before",
   DisplayFunction → $DisplayFunction];
```

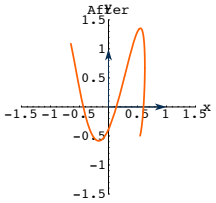


The coordinates of a point {x[t],y[t]} on this curve relative to the given perpendicular frame are:

{{x[t],y[t]}.perpframe[1],{x[t],y[t]}.perpframe[2]};

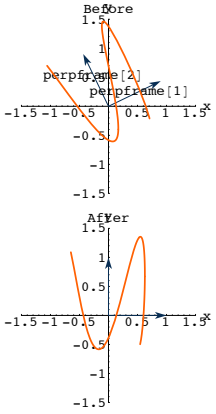
See what happens when you plot this as functions of t:

```
newcurveplot = ParametricPlot[
  {{x[t], y[t]}.perpframe[1], {x[t], y[t]}.perpframe[2]},
  {t, tlow, thigh}, PlotStyle → {{CadmiumOrange, Thickness[0.01]}},
  DisplayFunction → Identity];

xyunitvectors =
  {Arrow[{1, 0}, Tail → {0, 0}, VectorColor → Indigo, HeadSize → 0.2],
   Arrow[{0, 1}, Tail → {0, 0}, VectorColor → Indigo, HeadSize → 0.2]};

after = Show[xyunitvectors, newcurveplot,
   PlotRange → {{-ranger, ranger}, {-ranger, ranger}},
   Axes → True, AxesLabel → {"x", "y"}, PlotLabel → "After",
   DisplayFunction → $DisplayFunction];
```



See both:

```
Show[before];
Show[after];
```



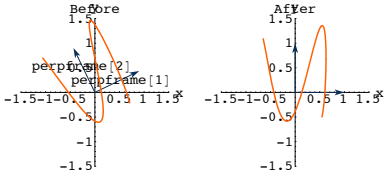
Grab the last two plots and animate slowly.

Describe what you see and explain why you see it

□ Answer:

In the second plot you see the same physical curve as in the first plot:

```
Show[GraphicsArray[{before, after}]];
```



But in the second plot, the curve is aligned on the x and y - axes with

{1, 0} playing the former role of perpframe[1]

and with

{0, 1} playing the former role of perpframe[2]

To explain why this happened, resolve {x[t],y[t]} into components in the directions of perpframe[1] and perpframe[2].

This gives you

{x[t], y[t]} = ({x[t], y[t]}.perpframe[1]) perpframe[1] +
({x[t], y[t]}.perpframe[2]) perpframe[2].

This means that to get to a point {x[t],y[t]} on the curve,
you advance {x[t],y[t]}.perpframe[1] units in the direction of perpframe[1]
and then advance {x[t],y[t]}.perpframe[2] units in the direction of perpframe[2].

If the fact that
{x[t], y[t]} = ({x[t], y[t]}.perpframe[1]) perpframe[1]
+({x[t], y[t]}.perpframe[2]) perpframe[2].
bothers you,revisit B.3) above

Now change perpframe[1] to {1,0} and change perpframe[2] to {0,1} to get

Change the red ones but not the blue ones.

({x[t], y[t]}.perpframe[1]) {1, 0} + ({x[t], y[t]}.perpframe[2]) {0, 1}

={ {x[t], y[t]}.perpframe[1] , {x[t], y[t]}.perpframe[2]}.

This means that to get to a point on the second curve,
you advance {x[t],y[t]}.perpframe[1] units in the direction of {1,0}
and then advance {x[t],y[t]}.perpframe[2] units in the direction of {0,1}.

That's why the new curve is same as the old curve, except the new curve is aligned on the x and y - axes with

{1, 0} playing the former former role of perpframe[1]

and with

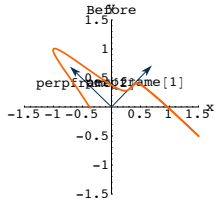
{0, 1} playing the former former role of perpframe[2].

□ B.3.b.ii) Another sample of aligning

Here's a new curve shown together with a perpendicular frame:

```
s = 0.8;
Clear[perpframe];
{perpframe[1], perpframe[2]} =
  {{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}};

Clear[x, y, t];
{x[t_], y[t_]} = {-Cos[t]^5 + 0.5 Sin[t/2], Cos[t]^5 + 0.5 Sin[t/2]};
ranger = 1.5;
{tlow, thigh} = {-π/4, 3π/2};

curveplot = ParametricPlot[{x[t], y[t]}, {t, tlow, thigh},
   PlotStyle → {{CadmiumOrange, Thickness[0.01]}},
   DisplayFunction → Identity];
```

```
frameplot = {Table[Arrow[perpframe[k], Tail → {0, 0},
     VectorColor → Indigo, HeadSize → 0.2], {k, 1, 2}],
   Graphics[Text["perpframe[1]", 0.6 perpframe[1]]],
   Graphics[Text["perpframe[2]", 0.6 perpframe[2]]]};

before = Show[frameplot, curveplot,
   PlotRange → {{-ranger, ranger}, {-ranger, ranger}},
   Axes → True, AxesLabel → {"x", "y"}, PlotLabel → "Before"];
```



Pick this curve up and align it on the usual x and y axes with

{1,0} playing the former role of perpframe[1]

and with

{0,1} playing the former role of perpframe[2].

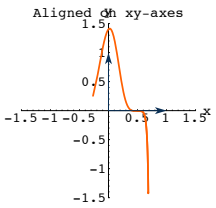
□ Answer:

You just take the given xy- parameterization {x[t],y[t]} and go with the coordinates of {x[t],t[t]} relative to the given perpendicualr frame:

{{x[t], y[t]}. perpframe[1], {x[t], y[t]}. perpframe[2]}.

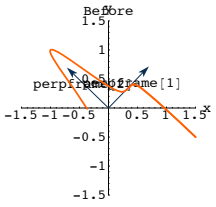
And plot:

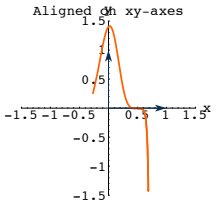
```
aligncurveplot = ParametricPlot[
   {{x[t], y[t]}.perpframe[1], {x[t], y[t]}.perpframe[2]},
   {t, tlow, thigh}, PlotStyle → {{CadmiumOrange, Thickness[0.01]}},
   DisplayFunction → Identity];
xyunitvectors = {Arrow[{1, 0}, Tail → {0, 0},
    VectorColor → Indigo, HeadSize → 0.2],
   Arrow[{0, 1}, Tail → {0, 0}, VectorColor → Indigo, HeadSize → 0.2]};


after = Show[aligncurveplot, xyunitvectors,
   PlotRange → {{-ranger, ranger}, {-ranger, ranger}}, Axes → True,
   AxesLabel → {"x", "y"}, PlotLabel → "Aligned on xy-axes",
   DisplayFunction -> $DisplayFunction];
```



See both:

```
Show[before];
Show[after];
```





Grab both plots and animate slowly.

Let your eyes confirm that the original action in the direction of perpframe[2], shows up in the aligned plot in the direction of the y-axis.

And let your eyes confirm that the original action in the direction of perpframe[1], shows up in the aligned plot in the direction of the x-axis.

B.5) 3D perpendicular frames. Hanging and aligning in 3D

□B.5.a.i) 3D perpendicular frames and Euler angles

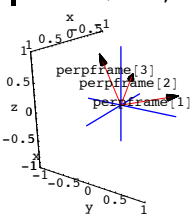
You dial up a 3D perpendicular frame by setting three angles r, s and t:

```
Clear[perpframe, r, s, t];
{perpframe[1], perpframe[2], perpframe[3]} =
 {{Cos[r] Cos[t] - Cos[s] Sin[r] Sin[t],
   Cos[s] Cos[t] Sin[r] + Cos[r] Sin[t], Sin[r] Sin[s]},
  {-Cos[t] Sin[r] - Cos[r] Cos[s] Sin[t],
   Cos[r] Cos[s] Cos[t] - Sin[r] Sin[t], Cos[r] Sin[s]},
  {Sin[s] Sin[t], -Cos[t] Sin[s], Cos[s]}}
```

```
{{Cos[r] Cos[t] - Cos[s] Sin[r] Sin[t],
  Cos[s] Cos[t] Sin[r] + Cos[r] Sin[t], Sin[r] Sin[s]},
 {-Cos[t] Sin[r] - Cos[r] Cos[s] Sin[t],
  Cos[r] Cos[s] Cos[t] - Sin[r] Sin[t], Cos[r] Sin[s]},
 {Sin[s] Sin[t], -Cos[t] Sin[s], Cos[s]}}
```

Here's one:

```
r = Pi / 4;
s = Pi / 8;
t = Pi / 3;
Clear[perpframe];
{perpframe[1], perpframe[2], perpframe[3]} =
  {{Cos[r] Cos[t] - Cos[s] Sin[r] Sin[t],
    Cos[s] Cos[t] Sin[r] + Cos[r] Sin[t], Sin[r] Sin[s]},
   {-Cos[t] Sin[r] - Cos[r] Cos[s] Sin[t],
    Cos[r] Cos[s] Cos[t] - Sin[r] Sin[t], Cos[r] Sin[s]},
   {Sin[s] Sin[t], -Cos[t] Sin[s], Cos[s]}};
ranger = 1.0;
frameplot = Show[Table[
  Arrow[perpframe[k],
    Tail -> {0, 0, 0}, VectorColor -> Red], {k, 1, 3}],
  Graphics3D[Text["perpframe[1]", 0.4 perpframe[1]]],
  Graphics3D[Text["perpframe[2]", 0.7 perpframe[2]]],
  Graphics3D[Text["perpframe[3]", 0.7 perpframe[3]]],
  Axes3D[2, 0.1],
  PlotRange ->
    {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
  Boxed -> False,
  Axes -> True, ViewPoint -> CMView, AxesLabel -> {"x", "y", "z"}];
```



Lots of folks call the angles r, s and t by the name "Euler angles."
What is the physical meaning meaning of the Euler angles r, s and t?

```
It's really hard to find a part of math that Euler didn't contribute to.
```

□Answer:

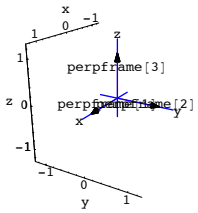
As Euler himself once said, one example gives the idea.

In the example above

$r = \frac{\pi}{4}; s = \frac{\pi}{8}; t = \frac{\pi}{3}.$

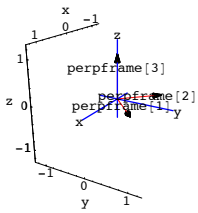
To see how this was built up, start with the usual perpendicular frame pointing out the positive x,y and z axes:

```
Clear[perpframe];
{perpframe[1], perpframe[2], perpframe[3]} =
  {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};
ranger = 1.3;
orginalframeplot = Show[Table[
  Arrow[perpframe[k],
    Tail -> {0, 0, 0}, VectorColor -> Red], {k, 1, 3}],
  Graphics3D[Text["perpframe[1]", 0.4 perpframe[1]]],
  Graphics3D[Text["perpframe[2]", 0.7 perpframe[2]]],
  Graphics3D[Text["perpframe[3]", 0.7 perpframe[3]]],
  Axes3D[1.3, 0.1],
  PlotRange ->
    {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
  Boxed -> False,
  Axes -> True, ViewPoint -> CMView, AxesLabel -> {"x", "y", "z"}];
```



To see the effect of r, go with Euler angles $r = \frac{\pi}{4}$, s = 0, t = 0:

```
r = Pi / 4;
s = 0;
t = 0;
Clear[perpframe];
{perpframe[1], perpframe[2], perpframe[3]} =
  {{Cos[r] Cos[t] - Cos[s] Sin[r] Sin[t],
    Cos[s] Cos[t] Sin[r] + Cos[r] Sin[t], Sin[r] Sin[s]},
   {-Cos[t] Sin[r] - Cos[r] Cos[s] Sin[t],
    Cos[r] Cos[s] Cos[t] - Sin[r] Sin[t], Cos[r] Sin[s]},
   {Sin[s] Sin[t], -Cos[t] Sin[s], Cos[s]}};
rframeplot = Show[Table[
  Arrow[perpframe[k],
    Tail -> {0, 0, 0}, VectorColor -> Red], {k, 1, 3}],
  Graphics3D[Text["perpframe[1]", 0.4 perpframe[1]]],
  Graphics3D[Text["perpframe[2]", 0.7 perpframe[2]]],
  Graphics3D[Text["perpframe[3]", 0.7 perpframe[3]]],
  Axes3D[1.3, 0.1],
  PlotRange ->
    {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
  Boxed -> False,
  Axes -> True, ViewPoint -> CMView, AxesLabel -> {"x", "y", "z"}];
```

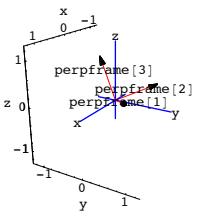


You get the perpendicular frame corresponding to Euler angles $r = \frac{\pi}{4}$, s = 0 and t = 0 by rotating everything by r radians about the z axis.

To see the combined effect of r and s, go with Euler angles $r = \frac{\pi}{4}$, $s = \frac{\pi}{8}$ and t = 0 :

```
r = Pi / 4;
s = Pi / 8;
t = 0;
Clear[perpframe];
{perpframe[1], perpframe[2], perpframe[3]} =
  {{Cos[r] Cos[t] - Cos[s] Sin[r] Sin[t],
    Cos[s] Cos[t] Sin[r] + Cos[r] Sin[t], Sin[r] Sin[s]},
   {-Cos[t] Sin[r] - Cos[r] Cos[s] Sin[t],
    Cos[r] Cos[s] Cos[t] - Sin[r] Sin[t], Cos[r] Sin[s]},
   {Sin[s] Sin[t], -Cos[t] Sin[s], Cos[s]}};
frameplot = Show[Table[
  Arrow[perpframe[k],
    Tail -> {0, 0, 0}, VectorColor -> Red], {k, 1, 3}],
  Graphics3D[Text["perpframe[1]", 0.4 perpframe[1]]],
  Graphics3D[Text["perpframe[2]", 0.7 perpframe[2]]],
  Graphics3D[Text["perpframe[3]", 0.7 perpframe[3]]],
  Axes3D[1.3, 0.1],
```

```
  PlotRange ->
    {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
  Boxed -> False,
  Axes -> True, ViewPoint -> CMView, AxesLabel -> {"x", "y", "z"}];
```



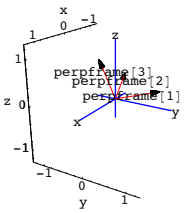
You get the perpendicular frame corresponding to r = $\frac{\pi}{4}$,s = $\frac{\pi}{8}$ and t = 0 by

->first rotating everything by r radians about the z axis

-> and then rotating everything by s radians about the x-axis.

To see the combined effect of r, s and t go with Euler angles r = $\frac{\pi}{4}$,s = $\frac{\pi}{8}$ and t = $\frac{\pi}{3}$:

```
r = Pi / 4;
s = Pi / 8;
t = Pi / 3;
Clear[perpframe];
{perpframe[1], perpframe[2], perpframe[3]} =
  {{Cos[r] Cos[t] - Cos[s] Sin[r] Sin[t],
     Cos[s] Cos[t] Sin[r] + Cos[r] Sin[t], Sin[r] Sin[s]},
   {-Cos[t] Sin[r] - Cos[r] Cos[s] Sin[t],
     Cos[r] Cos[s] Cos[t] - Sin[r] Sin[t], Cos[r] Sin[s]},
   {Sin[s] Sin[t], -Cos[t] Sin[s], Cos[s]}};
frameplot = Show[Table[
  Arrow[perpframe[k],
     Tail -> {0, 0, 0}, VectorColor -> Red], {k, 1, 3}],
  Graphics3D[Text["perpframe[1]", 0.4 perpframe[1]]],
  Graphics3D[Text["perpframe[2]", 0.7 perpframe[2]]],
  Graphics3D[Text["perpframe[3]", 0.7 perpframe[3]]],
  Axes3D[1.3, 0.1],
  PlotRange ->
    {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
  Boxed -> False,
  Axes -> True, ViewPoint -> CMView, AxesLabel -> {"x", "y", "z"}];
```



You get the perpendicular frame corresponding to r = $\frac{\pi}{4}$,s = $\frac{\pi}{8}$ and t = $\frac{\pi}{3}$ by

->first rotating everything by r radians about the z axis

-> and then rotating everything by s radians about the x-axis

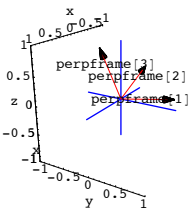
and finally

-> then rotating everything by t radians about the z-axis (again).

□ B.5.b.i) Hanging an ellipsoid on a 3D perpendicular frame

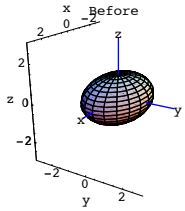
Here's a 3D perpendicular frame:

```
r = Pi / 6;
s = Pi / 8;
t = Pi / 3;
Clear[perpframe];
{perpframe[1], perpframe[2], perpframe[3]} =
  {{Cos[r] Cos[t] - Cos[s] Sin[r] Sin[t],
     Cos[s] Cos[t] Sin[r] + Cos[r] Sin[t], Sin[r] Sin[s]},
   {-Cos[t] Sin[r] - Cos[r] Cos[s] Sin[t],
     Cos[r] Cos[s] Cos[t] - Sin[r] Sin[t], Cos[r] Sin[s]},
   {Sin[s] Sin[t], -Cos[t] Sin[s], Cos[s]}};
ranger = 1.0;
frameplot =
  Show[Table[Arrow[perpframe[k], Tail → {0, 0, 0}, VectorColor → Red],
    {k, 1, 3}], Graphics3D[Text["perpframe[1]", 0.4 perpframe[1]]],
   Graphics3D[Text["perpframe[2]", 0.7 perpframe[2]]],
   Graphics3D[Text["perpframe[3]", 0.7 perpframe[3]]],
   Axes3D[2, 0.1], PlotRange → {{-ranger, ranger},
     {-ranger, ranger}, {-ranger, ranger}}, Boxed → False,
   Axes → True, ViewPoint → CMView, AxesLabel → {"x", "y", "z"}];
```



Here's a fat ellipsoid (football) skewered on the the x,y,and z axes:

```
Clear[x, y, z, s, t, pointcolor];
{x[s_, t_], y[s_, t_], z[s_, t_]} =
  {2.3 Sin[s] Cos[t], 1.6 Sin[s] Sin[t], 1.2 Cos[s]};
{slow, shigh} = {0, π};
{tlow, thigh} = {0, 2 π};
ranger = 3.0;

football = ParametricPlot3D[{x[s, t], y[s, t], z[s, t]},
   {s, slow, shigh}, {t, tlow, thigh}, PlotRange →
    {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
   Axes → True, AxesLabel → {"x", "y", "z"}, Boxed → False,
   ViewPoint → CMView, DisplayFunction → Identity];

Show[football, Axes3D[3, 0.2],
  PlotLabel → "Before", DisplayFunction → $DisplayFunction];
```



Hang this ellipsoid on the plotted perpendicular frame with
perpframe[1] playing the former role of the positive x-axis,
perpframe[2] playing the former role of the positive y-axis, and with
perpframe[3] playing the former role of the positive z-axis.

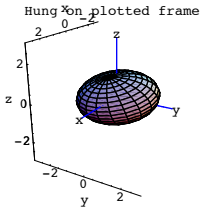
□ **Answer:**

You just take the xyz- parameterization {x[s,t], y[s,t], z[s,t]}, rip off x[s,t] , y[s,t] and z[s,t] and insert them like this

hangplotter[s, t] = x[s, t] perpframe[1] + y[s, t] perpframe[2] + z[s, t] perpframe[3].

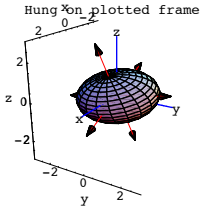
And plot:

```
Clear[hangplotter];
hangplotter[s_, t_] =
  x[s, t] perpframe[1] + y[s, t] perpframe[2] + z[s, t] perpframe[3];

hungfootball = ParametricPlot3D[hangplotter[s, t],
   {s, slow, shigh}, {t, tlow, thigh}, PlotRange →
    {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
   Axes → True, AxesLabel → {"x", "y", "z"}, Boxed → False,
   ViewPoint → CMView, DisplayFunction → Identity];

hungplot = Show[hungfootball,
   Axes3D[3, 0.2], PlotLabel → "Hung on plotted frame",
   DisplayFunction → $DisplayFunction];
```



See this plot together with scaled versions of the perpendicular frame:

```
scaledframeplot = {Table[
  Arrow[perpframe[k], Tail -> {0, 0, 0},
     ScaleFactor -> 3, VectorColor -> Red], {k, 1, 3}],
   Table[Arrow[-perpframe[k], Tail -> {0, 0, 0},
     ScaleFactor -> 3, VectorColor -> Red], {k, 1, 3}]};
   Show[hungfootball, scaledframeplot, Axes3D[3, 0.2],
  PlotLabel -> "Hung on plotted frame",
  DisplayFunction -> $DisplayFunction];
```



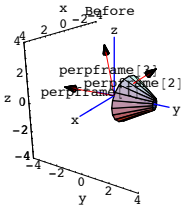
Grab and animate the last three plots.

The new football is skewered on the plotted perpendicuar frame the same way that the orginal football is skewered on the x,y, and z axes.

□B.5.b.ii) Hanging a surface on a 3D perpendicular frame

Here's surface shown together with a scaled 3D perpendicular frame:

```
Clear[x, y, z, s, t];
{x[s_, t_], y[s_, t_], z[s_, t_]} =
  s {2 Cos[t], 1, 1.5 Sin[t]} + (1 - s) {0.3 Cos[t], 3, 0.4 Sin[t]};

{tlow, thigh} = {0, 2 π};
{slow, shigh} = {0, 1};
ranger = 4;

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

Clear[perpframe];
{perpframe[1], perpframe[2], perpframe[3]} =
  {{Cos[r] Cos[t] - Cos[s] Sin[r] Sin[t],
     Cos[s] Cos[t] Sin[r] + Cos[r] Sin[t], Sin[r] Sin[s]},
    {-Cos[t] Sin[r] - Cos[r] Cos[s] Sin[t],
     Cos[r] Cos[s] Cos[t] - Sin[r] Sin[t], Cos[r] Sin[s]},
    {Sin[s] Sin[t], -Cos[t] Sin[s], Cos[s]}} /.
  {r → 0.3, s → π/4, t → -π/4};

scaledframeplot = {Table[Arrow[ranger perpframe[k],
     Tail → {0, 0, 0}, VectorColor → Red], {k, 1, 3}],
   Graphics3D[Text["perpframe[1]", 0.5 ranger perpframe[1]]],
   Graphics3D[Text["perpframe[2]", 0.5 ranger perpframe[2]]],
   Graphics3D[Text["perpframe[3]", 0.5 ranger perpframe[3]]]};

before =
  Show[scaledframeplot, surfaceplot, Axes3D[ranger], PlotRange →
    {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
   AspectRatio → Automatic, Axes → True, AxesLabel → {"x", "y", "z"},
   ViewPoint → CMView, PlotLabel → "Before",
   Boxed → False, DisplayFunction → $DisplayFunction];
```



Pick up this surface up and hang it on the plotted frame with
perpframe[1] playing the former role of {1,0,0} pointing along the positive x-axis
perpframe[2] playing the former role of {0,1,0} pointing along the positive y-axis
and with
perpframe[3] playing the former role of {0,0,1} pointing along the positive y-axis

□Answer:

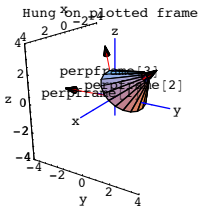
All you do is:

You just take the xyz- parameterization {x[s,t], y[s,t], z[s,t]}, rip off x[s,t] , y[s,t] and z[s,t] and insert them like this

hangplotter[s, t] = x[s, t] perpframe[1] + y[s, t] perpframe[2] + z[s, t] perpframe[3].

And plot:

```
Clear[hangplotter];
hangplotter[s_, t_] =
  x[s, t] perpframe[1] + y[s, t] perpframe[2] + z[s, t] perpframe[3];

hungsurface = ParametricPlot3D[hangplotter[s, t], {s, slow, shigh},
   {t, tlow, thigh}, PlotPoints → {2, Automatic}, PlotRange →
    {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
   Axes → True, AxesLabel → {"x", "y", "z"}, Boxed → False,
   ViewPoint → CMView, DisplayFunction → Identity];

hungplot = Show[hungsurface, Axes3D[ranger],
   scaledframeplot, PlotLabel → "Hung on plotted frame",
   DisplayFunction → $DisplayFunction];
```



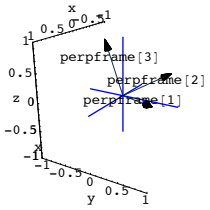
Grab both plots and animate.

Done.

□B.5.c.i) Aligning an ellipsoid on the x,y and z axes in 3D

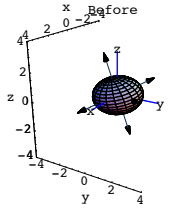
Here's a 3D perpendicular frame:

```
r = 0.6;
s = 0.3;
t = 0.4;
Clear[perpframe];
{perpframe[1], perpframe[2], perpframe[3]} =
  {{Cos[r] Cos[t] - Cos[s] Sin[r] Sin[t],
    Cos[s] Cos[t] Sin[r] + Cos[r] Sin[t], Sin[r] Sin[s]},
   {-Cos[t] Sin[r] - Cos[r] Cos[s] Sin[t],
    Cos[r] Cos[s] Cos[t] - Sin[r] Sin[t], Cos[r] Sin[s]},
   {Sin[s] Sin[t], -Cos[t] Sin[s], Cos[s]}};
ranger = 1.0;
frameplot = Show[Table[
  Arrow[perpframe[k],
     Tail -> {0, 0, 0}, VectorColor -> Indigo], {k, 1, 3}],
  Graphics3D[Text["perpframe[1]", 0.4 perpframe[1]]],
  Graphics3D[Text["perpframe[2]", 0.7 perpframe[2]]],
  Graphics3D[Text["perpframe[3]", 0.7 perpframe[3]]],
  Axes3D[2, 0.1],
  PlotRange ->
    {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
  Boxed -> False,
  Axes -> True, ViewPoint -> CMView, AxesLabel -> {"x", "y", "z"}];
```



Here's a fat ellipsoid (football) skewered on the same perpendicular frame:

```
Clear[x, y, z, s, t, pointcolor];
{x[s_, t_], y[s_, t_], z[s_, t_]} = 2.3 Sin[s] Cos[t] perpframe[1] +
   1.6 Sin[s] Sin[t] perpframe[2] + 1.2 Cos[s] perpframe[3];

{slow, shigh} = {0, π};
{tlow, thigh} = {0, 2 π};
ranger = 4.0;

football = ParametricPlot3D[{x[s, t], y[s, t], z[s, t]},
   {s, slow, shigh}, {t, tlow, thigh}, PlotRange →
    {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
   Axes → True, AxesLabel → {"x", "y", "z"}, Boxed → False,
   ViewPoint → CMView, DisplayFunction → Identity];

scaledframeplot =
  {Table[Arrow[perpframe[k], Tail → {0, 0, 0}, ScaleFactor → 3,
     VectorColor → Indigo], {k, 1, 3}],
   Table[Arrow[-perpframe[k], Tail → {0, 0, 0},
     ScaleFactor → 3, VectorColor → Indigo], {k, 1, 3}]};

Show[football, Axes3D[3, 0.2], scaledframeplot,
  PlotLabel → "Before", DisplayFunction → $DisplayFunction];
```



Align this ellipsoid on the x,y and z- axes with
{1,0,0} pointing along the positive x- axis playing the former role of perpframe[1],

{0,1,0} pointing along the positive y- axis playing the former role of perpframe[2],
{0,0,1} pointing along the positive z- axis playing the former role of perpframe[3].

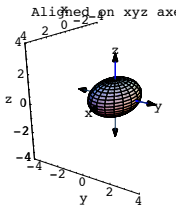
□ Answer:

Just as in 2D, you take the given xyz- parameterization {x[s,t],y[s,t],z[s,t]} of the ellipsoid and go with the coordinates of {x[s,t],y[s,t],z[s,t]} and then plot

.

{{x[s, t], y[s, t], z[s, t]}. perpframe[1],
{x[s, t], y[s, t], z[s, t]}. perpframe[2],
{x[s, t], y[s, t], z[s, t]}. perpframe[3]} :

```
alignedfootball =
 ParametricPlot3D[{{x[s, t], y[s, t], z[s, t]}.perpframe[1],
   {x[s, t], y[s, t], z[s, t]}.perpframe[2],
   {x[s, t], y[s, t], z[s, t]}.perpframe[3]},
  {s, slow, shigh}, {t, tlow, thigh}, PlotRange →
   {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
  Axes → True, AxesLabel → {"x", "y", "z"}, Boxed → False,
  ViewPoint → CMView, DisplayFunction → Identity];
xyzperpframe[1] = {1, 0, 0};
xyzperpframe[2] = {0, 1, 0};
xyzperpframe[3] = {0, 0, 1};

scaledxyzunitvectors =
  {Table[Arrow[xyzperpframe[k], Tail → {0, 0, 0}, ScaleFactor → 3,
     VectorColor → Indigo], {k, 1, 3}],
   Table[Arrow[-xyzperpframe[k], Tail → {0, 0, 0},
     ScaleFactor → 3, VectorColor → Indigo], {k, 1, 3}]};


Show[alignedfootball, Axes3D[3, 0.2],
  scaledxyzunitvectors, PlotLabel → "Aligned on xyz axes",
  DisplayFunction → $DisplayFunction];
```



Grab both plots and animate.

There you go.

This ellipsoid is now aligned on the x,y and z- axes with

{1,0,0} pointing along the positive x- axis playing the former role of perpframe[1],

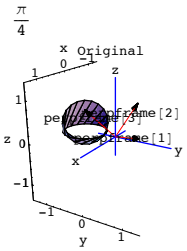
{0,1,0} pointing along the positive y- axis playing the former role of perpframe[2],

{0,0,1} pointing along the positive z- axis playing the former role of perpframe[3].

□ B.5.c.ii) Aligning a surface on the x,y and z axes in 3D

Here's a 3D surface shown with a plot of a 3D perpendicular frame::

```
r = Pi / 8;
s = Pi / 4;
t = Pi / 4

Clear[perpframe];
{perpframe[1], perpframe[2], perpframe[3]} =
  {{Cos[r] Cos[t] - Cos[s] Sin[r] Sin[t],
    Cos[s] Cos[t] Sin[r] + Cos[r] Sin[t], Sin[r] Sin[s]},
   {-Cos[t] Sin[r] - Cos[r] Cos[s] Sin[t],
    Cos[r] Cos[s] Cos[t] - Sin[r] Sin[t], Cos[r] Sin[s]},
   {Sin[s] Sin[t], -Cos[t] Sin[s], Cos[s]}};

ranger = 1.0;
scaledframeplot =
  {Table[Arrow[perpframe[k], Tail → {0, 0, 0}, VectorColor → Red],
    {k, 1, 3}], Graphics3D[Text["perpframe[1]", 0.4 perpframe[1]]],
   Graphics3D[Text["perpframe[2]", 0.7 perpframe[2]]],
   Graphics3D[Text["perpframe[3]", 0.7 perpframe[3]]]};

Clear[x, y, z, s, t];
{x[s_, t_], y[s_, t_], z[s_, t_]} =
  0.5 s (Cos[t] perpframe[1] + Sin[t] perpframe[2]) +
   perpframe[3] - 0.5 {Cos[t], Sin[t], Cos[s]};

{slow, shigh} = {0, 1};
{tlow, thigh} = {0, 2 π};

ranger = 1.4;
surfaceplot = ParametricPlot3D[
   {x[s, t], y[s, t], z[s, t]}, {s, slow, shigh}, {t, tlow, thigh},
   PlotPoints → {2, Automatic}, DisplayFunction → Identity];

before =
  Show[surfaceplot, Axes3D[ranger], scaledframeplot, PlotRange →
    {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
    Axes → True, AxesLabel → {"x", "y", "z"}, Boxed → False,
    PlotLabel → "Original", ViewPoint → CMView,
    DisplayFunction → $DisplayFunction];
```



Align this surface on the x,y and z- axes with
{1,0,0} pointing along the positive x- axis playing the former role of perpframe[1],
{0,1,0} pointing along the positive y- axis playing the former role of perpframe[2],
{0,0,1} pointing along the positive z- axis playing the former role of perpframe[3].

□ Answer:

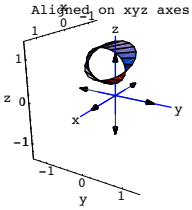
Just as in 2D, you take the given xyz- parameterization {x[s,t],y[s,t],z[s,t]} of the surface and go with the coordinates of {x[s,t],y[s,t],z[s,t]}realitive to the perpendicular frame

.

{{x[s, t], y[s, t], z[s, t]}. perpframe[1],
{x[s, t], y[s, t], z[s, t]}. perpframe[2],
{x[s, t], y[s, t], z[s, t]}. perpframe[3]} :

And then plot:

```
alignedsurface =
 ParametricPlot3D[{{x[s, t], y[s, t], z[s, t]}.perpframe[1],
   {x[s, t], y[s, t], z[s, t]}.perpframe[2],
   {x[s, t], y[s, t], z[s, t]}.perpframe[3]},
  {s, slow, shigh}, {t, tlow, thigh}, PlotRange →
   {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
  Axes → True, AxesLabel → {"x", "y", "z"},
  PlotPoints -> {2, Automatic}, Boxed → False,
  ViewPoint → CMView, DisplayFunction → Identity];
xyzperpframe[1] = {1, 0, 0};
xyzperpframe[2] = {0, 1, 0};
xyzperpframe[3] = {0, 0, 1};

xyzunitvectors = {Table[
    Arrow[xyzperpframe[k], Tail → {0, 0, 0}, VectorColor → Indigo],
    {k, 1, 3}], Table[Arrow[-xyzperpframe[k], Tail → {0, 0, 0},
     VectorColor → Indigo], {k, 1, 3}]};


after = Show[alignedsurface, xyzunitvectors,
   Axes3D[ranger], PlotLabel → "Aligned on xyz axes",
   DisplayFunction → $DisplayFunction];
```



Grab both plots,align amd animate.

Now the z-axis pierces the aligned surfrace just the way perpframe[3] pierces the orginal surface.

Done.

B.6) The formula

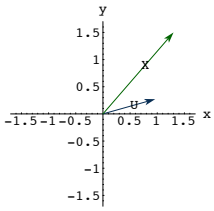
X.Y = ||X|| ||Y|| Cos[angle between X and Y]

X.Y = 0 tells you that X is perpendicular to Y

□ B.6.a.i) If U is a unit vector and X is any vector, then X . U = ||X|| Cos[angle between X and U]

Here is a unit vector U shown with another vector X:

```
s = Random[Real, {0.2, 0.4}];
X = {1.3, 1.5};
U = {Cos[s], Sin[s]};
ranger = 1.7;
setup = Show[
   Arrow[U, Tail -> {0, 0}, VectorColor -> Indigo, HeadSize -> 0.2],
   Arrow[X, Tail -> {0, 0}, VectorColor -> GosiaGreen,
    HeadSize -> 0.2],
   Graphics[Text["X", 0.6 X]], Graphics[Text["U", 0.6 U]],
   Axes → True, AxesLabel → {"x", "y"},
   PlotRange → {{-ranger, ranger}, {-ranger, ranger}}];
```



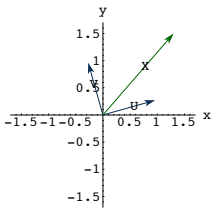
Explain why the dot product

$$X \cdot U = \|X\| \text{ Cos[angle between X and U]}.$$

□ Answer:

Throw in a unit vector V perpendicular to U:

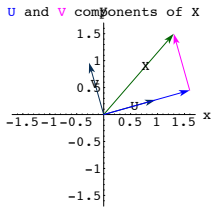
```
V = {Cos[s + π/2], Sin[s + π/2]};
newsetup = Show[setup,
   Arrow[V, Tail -> {0, 0}, VectorColor -> Indigo, HeadSize -> 0.2],
   Graphics[Text["V", 0.6 V]]];
```



{U,V} is a perpendicular frame.

Resolve the point at the tip of X into perpendicular components in the directions of U and V:

```
UcompofX = (X.U) U;
VcompofX = (X.V) V;
revealer = Show[newsetup, Arrow[UcompofX, Tail -> {0, 0},
                  VectorColor -> Blue, HeadSize -> 0.2],
  Arrow[VcompofX, Tail -> UcompofX, VectorColor -> Magenta],
  PlotLabel -> "U and V components of X"];
```



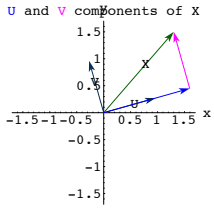
Because {U,V} is a perpendicular frame and

- the component of X in the direction of U is parallel to U and
- the component of X in the direction of V is parallel to V,

the triangle you are looking at is guaranteed to be a right triangle.

Take another look:

```
Show[revealer];
```



The blue vector on the bottom is (X.U) U.

Because U is a unit vector, this vector runs a distance of X.U units.

So

$$\text{Cos[angle between U and X]} = \frac{\text{adjacent}}{\text{hypotenuse}} = \frac{(X.U)}{\|X\|}.$$

Rearrange to get:

$$X.U = \|X\| \text{ Cos[angle between U and X]}.$$

The same explanation works for any unit vector U and any vector X, so the explanation is over.

Short and just a bit sweet.

□ B.6.a.ii) If X and Y are any vectors, then X.Y = ||X|| ||Y|| Cos[angle between X and Y]

Explain this bold statement:

If X and Y are any vectors, then

$$X.Y = \|X\| \|Y\| \text{ Cos[angle between X and Y]}.$$

□ Answer:

```
This is what some folks would call a corollary of part i) immediately above.
                  Some folks call this a scaling argument.
```

Go with any two vectors X and Y and make the observation that

$$U = \frac{Y}{\|Y\|}$$

is a unit vector pointing in the same direction as Y.

Also note that

- $Y = \|Y\| \frac{Y}{\|Y\|} = \|Y\| U.$

So:

$$X.Y = X.(\|Y\| U) = \|Y\| (X.U)$$

Because U is a unit vector,

Part i) immediately above steps in to tell you that $X.(U). = \|X\| \text{ Cos[angle between X and U]}$.

So:

$$X.Y = X.(\|Y\| U) = \|Y\| (X.U) = \|Y\| \|X\| \text{ Cos[angle between X and U]}.$$

But now because U is a unit vetor pointing in the same direction as Y,

$$X.Y = \|Y\| \|X\| \text{ Cos[angle between X and U]} = \|Y\| \|X\| \text{ Cos[angle between X and Y]}.$$

This is the same as

$$X.Y = \|X\| \|Y\| \text{ Cos[angle between X and Y]}.$$

Explanation complete and you're out of here.

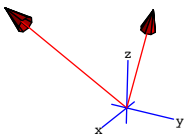
□ B.6..b.i) The perpendicularity test: X.Y = 0 tells you that X is perpendicular to Y

Look at

$$X = \{1.0, 1.2, 3.5\} \text{ and } Y = \{-1.0, -5.0, 2.0\}$$

with their tails at {0, 0, 0}:

```
X = {1.0, 1.2, 3.5};
Y = {-1.0, -5.0, 2.0};

Show[Arrow[X, VectorColor → Red],
        Arrow[Y, VectorColor → Red],
        Axes3D[1.5, 0.2],
  PlotRange → All, ViewPoint → CMView, Boxed → False];
```



These vectors look like they might be perpendicular.
How can you tell for sure?

□ Answer:

Just look at X.Y:

```
X.Y
```

0

Now you know that X.Y = 0; so you know for sure that X is perpendicular to Y.

If you want to find out why this test works, go on.

□ B.6.b.ii) Explanation of the perpendicularity test

Explain this statement:

If X.Y = 0, then X is perpendicular to Y.

□ Answer:

You know that

$$X.Y = \|X\| \|Y\| (\text{Cos[b]}),$$

where b is the angle between X and Y.

So if X.Y = 0, you can read off

$\text{Cos}[b] = 0 = \text{Cos}[\frac{\pi}{2}] = \text{Cos}[-\frac{\pi}{2}]$.

This tells you that the angle b between X and Y is a right angle.