Matrices, Geometry&*Mathematica*

Authors: Bruce Carpenter, Bill Davis and Jerry Uhl

Producer: Bruce Carpenter

Publisher: Math Everywhere, Inc.

MGM.01 Perpendicular Frames
TUTORIALS

T.1) Area and volume measurements resulting from stretch factors

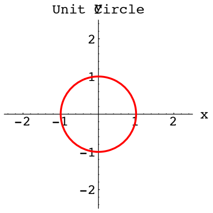
□T.1.a.i) Measuring the area enclosed by an ellipse

You plot the circle of radius 1 centered at {0,0} by plotting

{Cos[t], Sin[t]}

running t from 0 to 2 π:

```
Clear[t];
ranger = 2.5;
{tlow, thigh} = {0, 2 Pi};
    circleplot = ParametricPlot[{Cos[t], Sin[t]}, {t, tlow, thigh},
  PlotStyle -> {{Thickness[0.01], Red}}, AxesLabel -> {"x", "y"},
      PlotRange -> {{-ranger, ranger}, {-ranger, ranger}},
      AspectRatio -> Automatic, PlotLabel -> "Unit Circle"];
```

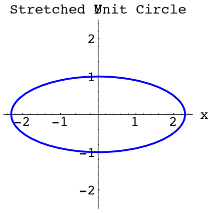


Here's the ellipse you get ellipse by plotting

{2.3 Cos[t], Sin[t]}

running t from 0 to 2 π:

```
Clear[t];
xstretch = 2.3;
{tlow, thigh} = {0, 2 Pi};
 ellipseplot =
  ParametricPlot[{xstretch Cos[t], Sin[t]}, {t, tlow, thigh},
   PlotStyle -> {{Thickness[0.01], Blue}}, AxesLabel -> {"x", "y"},
           PlotRange -> {{-ranger, ranger}, {-ranger, ranger}},
       AspectRatio -> Automatic,
   PlotLabel -> "Stretched Unit Circle"];
```



Grab and animate.

Given that the area enclosed by the circle measures out to π square units, measure the area enclosed by the plotted ellipse.

□**Answer:**

You plot the circle by plotting

{Cos[t], Sin[t]} and running t from 0 to 2 π:.

You plot the ellipse by putting xstretch = 2.3 and plotting

{xstretch Cos[t], Sin[t]} and running t from 0 to 2 π:.

In other words, you start with

{Cos[t], Sin[t]} on the circle

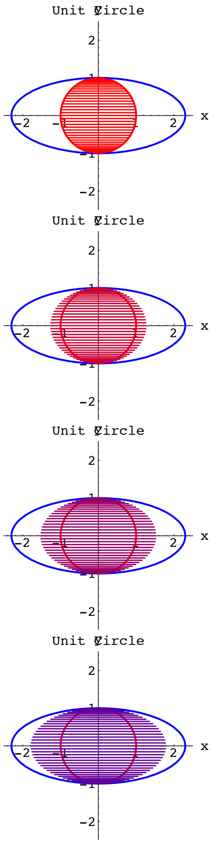
and stretch the x coordinate by a factor of xstretch to get to the corresponding point

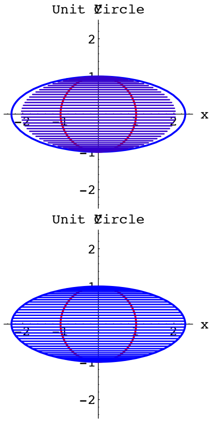
{xstretch Cos[t], Sin[t]} on the ellipse.

So to get from the circle to the ellipse you just stretch all x's on the circle by a factor of xstretch = 2.3.

See it happen::

```
Clear[bars, s, x];
stretcherbars[s_] :=
  Graphics[{RGBColor[(xstretch - s)/(xstretch - 1), 0, (s - 1)/(xstretch - 1)], Table[Line[
      {{-s Cos[t], Sin[t]}, {s Cos[t], Sin[t]}}], {t, -π/2, π/2, π/40}]}];

Table[Show[circleplot, ellipseplot, stretcherbars[s],
    Axes → True, AxesLabel → {"x", "y"}, AspectRatio → Automatic],
  {s, 1, xstretch, (xstretch - 1)/5}];
```





Grab and animate.

This tells you that

ellipse area = (xstretch) (circle area) = (2.3) π.

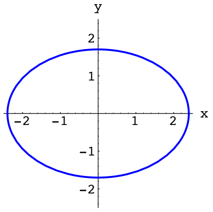
□T.1.a.ii) Measuring the area enclosed by another ellipse

Here's the ellipse you by plotting

{2.4 Cos[t], 1.7 Sin[t]}

and running t from 0 to 2 π:

```
Clear[t];
xstretch = 2.4;
ystretch = 1.7;
ranger = 2.5;
{tlow, thigh} = {0, 2 Pi};
    ellipseplot =
  ParametricPlot[{xstretch Cos[t], ystretch Sin[t]}, {t, tlow, thigh},
   PlotStyle -> {{Thickness[0.01], Blue}}, AxesLabel -> {"x", "y"},
         PlotRange -> {{-ranger, ranger}, {-ranger, ranger}},
         AspectRatio -> Automatic];
```



Measure the area enclosed by the plotted ellipse.

□ **Answer:**

To get this ellipse, you plotted

{xstretch Cos[t], ystretch Sin[t]} and running t from 0 to 2 π:

The area enclosed by this ellipse measures out (in square units) to:

```
(xstretch) (ystretch) Pi
```

```
12.8177
```

Reason:

You plot the unit circle by plotting

{Cos[t], Sin[t]} and running t from 0 to 2 π.

The area enclosed by tis circle measure out to π square units.

You plot the ellipse by plotting

{xstretch Cos[t], ystretch Sin[t]} and running t from 0 to 2 π:.

In other words, you start with

{Cos[t], Sin[t]} on the circle

and stretch the x coordinate by a factor of xstretch and stretch the y coordinate by a factor of ystretch to get to the corresponding point

{xstretch Cos[t], ystretch Sin[t]} on the ellipse.

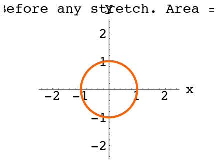
This tells you that

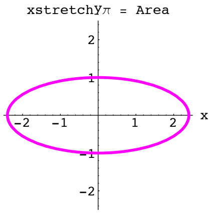
ellipse area = (xstretch) (ystretch) (circle area) = (2.4) (1.7) (circle area) = (2.4) (1.7) π.

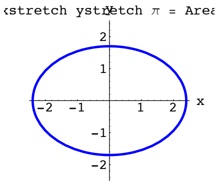
See it happen in two stages:

```
Clear[t];
ranger = 2.5;
```

```
xstretch = 2.4;
ystretch = 1.7;
{tlow, thigh} = {0, 2 Pi};
stage0plot = ParametricPlot[{Cos[t], Sin[t]}, {t, tlow, thigh},
       PlotStyle -> {{Thickness[0.015], CadmiumOrange}},
  AxesLabel -> {"x", "y"},
          PlotRange -> {{-ranger, ranger}, {-ranger, ranger}},
          AspectRatio -> Automatic,
  PlotLabel -> " Before any stretch. Area =" π];

 stage1plot =
 ParametricPlot[{xstretch Cos[t], Sin[t]}, {t, tlow, thigh},
       AxesLabel -> {"x", "y"},
     PlotStyle -> {{Thickness[0.015], RGBColor[1, 0, 1]}},
       PlotRange -> {{-ranger, ranger}, {-ranger, ranger}},
       AspectRatio -> Automatic,
     PlotLabel -> "xstretch π = Area" ];

 stage2plot = ParametricPlot[{xstretch Cos[t], ystretch Sin[t]},
   {t, tlow, thigh}, AxesLabel -> {"x", "y"},
    PlotStyle -> {{Thickness[0.015], Blue}},
       PlotRange -> {{-ranger, ranger}, {-ranger, ranger}},
       AspectRatio -> Automatic,
     PlotLabel -> "xstretch ystretch π = Area" ];
```



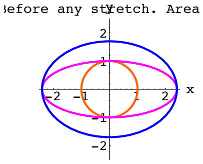




Grab and animate slowly.

See them all:

```
Show[stage0plot, stage1plot, stage2plot];
```



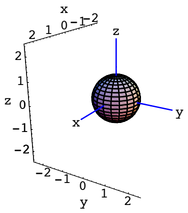
□T.1.b.i) Measuring the volume enclosed by a 3D ellipsoid

You plot the sphere of radius 1 centered at {0,0,0} by plotting

{Sin[s] Cos[t], Sin[s] Sin[t], Cos[s]}

running t from 0 to 2 π and running s from 0 to π:

```
Clear[s, t];
{slow, shigh} = {0, π};
{tlow, thigh} = {0, 2 π};

ranger = 2.5;
sphereplot = ParametricPlot3D[{Sin[s] Cos[t], Sin[s] Sin[t], Cos[s]},
   {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[sphereplot, Axes3D[ranger], DisplayFunction → $DisplayFunction];
```

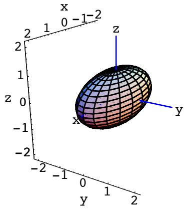


Here's the ellipsoid (football) you get by plotting

{2.1 Sin[s] Cos[t], Sin[s] Sin[t], Cos[s]}

running t from 0 to 2 π and running s from 0 to π:

```
xstretch = 2.1;
Clear[s, t];
{slow, shigh} = {0, π};
{tlow, thigh} = {0, 2 π};

ranger = 2.2;
ellipsoidplot =
 ParametricPlot3D[{xstretch Sin[s] Cos[t], Sin[s] Sin[t], Cos[s]},
  {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[ellipsoidplot, Axes3D[ranger],
  DisplayFunction → $DisplayFunction];
```



Given that the volume enclosed by the sphere measures out to $\frac{4}{3}\pi$ cubic units, measure the volume enclosed by the plotted ellipsoid..

□ **Answer:**

You plot the sphere by plotting

{Sin[s] Cos[t], Sin[s] Sin[t], Cos[s]}

running t from 0 to 2 π and running s from 0 to π:

You plot the ellipsoid by putting xstretch = 2.1 and plotting

{xstretch Sin[s] Cos[t], Sin[s] Sin[t], Cos[s]}

running t from 0 to 2 π and running s from 0 to π:

In other words, you start with

{Sin[s] Cos[t], Sin[s] Sin[t], Cos[s]} on the sphere

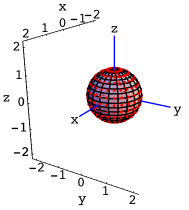
and stretch the x coordinate by a factor of xstretch to get to the corresponding point

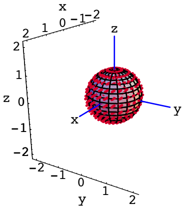
{xstretch Sin[s] Cos[t], Sin[s] Sin[t], Cos[s]} on the ellipsoid.

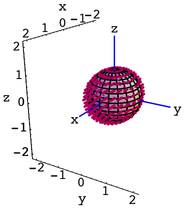
So to get from the sphere to the ellipse you just stretch all x's on the sphere by a factor of xstretch .

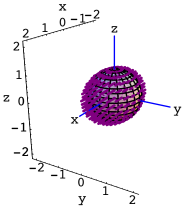
See it happen:

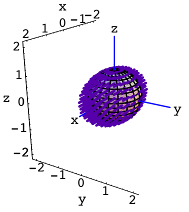
```
Clear[bars, v, x];
stretcherbars[v_] := Graphics3D[
   {Thickness[0.015], RGBColor[(xstretch - v)/(xstretch - 1), 0, (v - 1)/(xstretch - 1)],
    Table[Line[{{-v Sin[s] Cos[t], Sin[s] Sin[t], Cos[s]},
       {v Sin[s] Cos[t], Sin[s] Sin[t], Cos[s]}}],
     {t, -π/2, π/2, π/12}, {s, 0, π, π/12}]}];

Table[Show[sphereplot, stretcherbars[v], Axes3D[ranger], PlotRange →
    {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
   Axes → True, AxesLabel → {"x", "y", "z"}, Boxed → False,
   ViewPoint → CMView, DisplayFunction → $DisplayFunction],
  {v, 1, xstretch, (xstretch - 1)/6}];

Show[ellipsoidplot, Axes3D[ranger],
  DisplayFunction → $DisplayFunction];
```

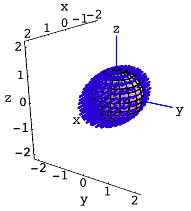


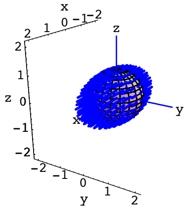


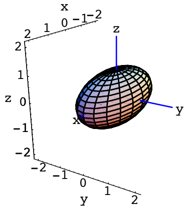












Grab and animate slowly.

This tells you that

ellipsoid volume = (xstretch) (sphere volume) = (xstretch) ($\frac{4}{3}$ π):

```
(xstretch) (4 / 3) π
8.79646
```

□T.1.b.ii) Measuring the volume enclosed by another ellipsoid

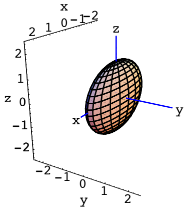
Here's the ellipsoid (pancake!) you get by plotting

{2.1 Sin[s] Cos[t], 0.5 Sin[s] Sin[t], 1.5 Cos[s]}

running t from 0 to 2 π and running s from 0 to π:

```
xstretch = 2.1;
ystretch = 0.5;
zstretch = 1.5;
Clear[s, t];
{slow, shigh} = {0, π};
```

```
{tlow, thigh} = {0, 2 π};

ranger = 2.5;
ellipsoidplot = ParametricPlot3D[
  {xstretch Sin[s] Cos[t], ystretch Sin[s] Sin[t], zstretch Cos[s]},
  {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[ellipsoidplot, Axes3D[ranger],
  DisplayFunction → $DisplayFunction];
```



Measure the volume enclosed by the plotted ellipsoid..

□Answer:

To get this ellipsoid, you plotted

{xstretch Sin[s] Cos[t], ystretch Sin[s] Sin[t], zstretch Cos[s]}

running t from 0 to 2 π and running s from 0 to π:

The volume enclosed by this ellipsoid measures out (in cubic units) to:

```
xstretch = 2.1;
ystretch = 0.3;
zstretch = 1.5;
(xstretch) (ystretch) (zstretch) (4 / 3) Pi
3.95841
```

Reason:

You plot the unit sphere by plotting

{Sin[s] Cos[t], Sin[s] Sin[t], Cos[s]}

running t from 0 to 2 π and running s from 0 to π:

You plot the ellipsoid by plotting

{xstretch Sin[s] Cos[t], ystretch Sin[s] Sin[t], zstretch Cos[s]}

running t from 0 to 2 π and running s from 0 to π:

In other words, you start with

{Sin[s] Cos[t], Sin[s] Sin[t], Cos[s]} on the sphere

and stretch the

x coordinate (first slot) by a factor of xstretch

y coordinate (second slot) by a factor of ystretch

z coordinate (second slot) by a factor of zstretch

to get to the corresponding point

{xstretch Sin[s] Cos[t], ystretch Sin[s] Sin[t], zstretch Cos[s]}

on the ellipsoid.

Each time you multiply by one of the factors, you multiply volume by that factor.

So

ellipsoid volume = (xstretch) (ystretch) (zstretch) (sphere volume):

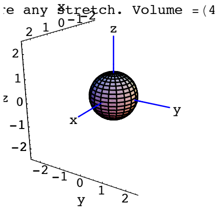
```
(xstretch) (ystretch) (zstretch) (4 π / 3)
```

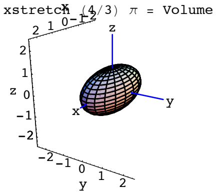
```
3.95841
```

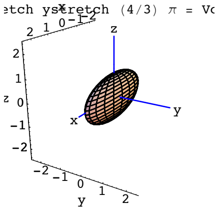
See it happen in three stages:

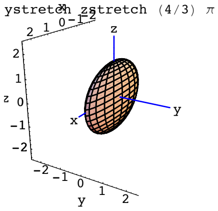
```
Clear[s, t];
{slow, shigh} = {0, π};
{tlow, thigh} = {0, 2 π};
xstretch = 2.1;
ystretch = 0.3;
zstretch = 1.5;
ranger = 2.5;
stage0plot = ParametricPlot3D[{Sin[s] Cos[t], Sin[s] Sin[t], Cos[s]},
   {s, slow, shigh}, {t, tlow, thigh}, PlotRange →
    {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
   Axes → True, AxesLabel → {"x", "y", "z"}, Boxed → False,
   ViewPoint → CMView, DisplayFunction → Identity];
Show[stage0plot, Axes3D[ranger],
   PlotLabel → "Before any stretch. Volume =(4/3)" π,
   DisplayFunction → $DisplayFunction];
stage1plot = ParametricPlot3D[{xstretch Sin[s] Cos[t], Sin[s] Sin[t],
    Cos[s]}, {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[stage1plot, Axes3D[ranger],
   PlotLabel → "xstretch (4/3) π = Volume",
   DisplayFunction → $DisplayFunction];
```

```
stage2plot = ParametricPlot3D[
   {xstretch Sin[s] Cos[t], ystretch Sin[s] Sin[t], Cos[s]},
   {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[stage2plot, Axes3D[ranger],
   PlotLabel → "xstretch ystretch (4/3) π = Volume",
   DisplayFunction → $DisplayFunction];
stage3plot = ParametricPlot3D[
   {xstretch Sin[s] Cos[t], ystretch Sin[s] Sin[t], zstretch Cos[s]},
   {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[stage3plot, Axes3D[ranger],
   PlotLabel → "xstretch ystretch zstretch (4/3) π = Volume",
   DisplayFunction → $DisplayFunction];
```









Grab the plots and animate.

T.2) Using perpendicular frames to plot and measure tilted 2D ellipses and 3D ellipsoids

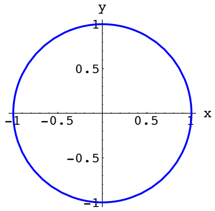
□ T.2.a.i) Plotting tilted 2D ellipses

Use stretch factors and perpendicular frames to plot tilted 2D ellipses.

□ **Answer:**

You get the unit circle centered at {0,0} by plotting {Cos[t], Sin[t]} running t from 0 to 2π:

```
Clear[t];
{tlow, thigh} = {0, 2 Pi};
    circleplot = ParametricPlot[{Cos[t], Sin[t]}, {t, tlow, thigh},
   PlotStyle -> {{Thickness[0.01], Blue}}, AxesLabel -> {"x", "y"}];
```

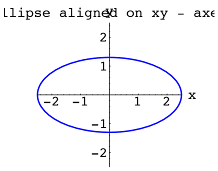


To plot an ellipse centered at {0,0} and aligned on the x and y axes you stretch the circle out on the x and y -axes by selecting stretch factors xstretch and ystretch and then plotting

{x[t],y[t]} = {xstretch Cos[t], ystretch Sin[t]}

like this:

```
xstretch = 2.5;
ystretch = 1.3;
ranger = Max[{xstretch, ystretch}];
Clear[t, x, y];
{x[t_], y[t_]} = {xstretch Cos[t], ystretch Sin[t]};
stretched = ParametricPlot[{x[t], y[t]}, {t, tlow, thigh},
   PlotStyle -> {{Thickness[0.01], Blue}}, PlotRange ->
    {{-ranger, ranger}, {-ranger, ranger}}, AxesLabel -> {"x", "y"},
    PlotLabel -> "Ellipse aligned on xy - axes",
  DisplayFunction -> $DisplayFunction];
{x[t], y[t]}
```



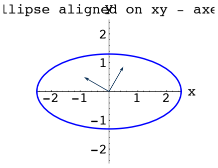
```
{2.5 Cos[t], 1.3 Sin[t]}
```

The parameterization {x[t],y[t]} of this ellipse is shown directly below the plot.

Next you choose a perpendicular frame.

Here's a sample:

```
s = π/3;
Clear[perpframe];
{perpframe[1], perpframe[2]} =
  {{Cos[s], Sin[s]}, {-Sin[s], Cos[s]}};
frameplot = {Arrow[perpframe[1], Tail → {0, 0}, VectorColor → Indigo],
    Arrow[perpframe[2], Tail → {0, 0}, VectorColor → Indigo]};

Show[stretched, frameplot,
  PlotRange → {{-ranger, ranger}, {-ranger, ranger}},
  AspectRatio → Automatic,
  PlotLabel → "Ellipse aligned on xy - axes",
  DisplayFunction → $DisplayFunction];
```



And then you hang that ellipse on the perpendicular frame with

perpframe[1] playing the former role of the positive x-axis

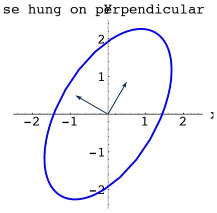
and with

perpframe[2] playing the former role of the positive y-axis.

by plotting

x[t] perpframe[1] + y[t] perpframe[2].

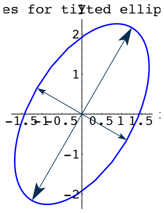
```
hung = ParametricPlot[x[t] perpframe[1] + y[t] perpframe[2],
  {t, 0, 2 π}, PlotStyle → {{Thickness[0.01], Blue}},
  AxesLabel → {"x", "y"}, DisplayFunction → Identity];

Show[hung, frameplot, AspectRatio → Automatic,
  PlotRange → {{-ranger, ranger}, {-ranger, ranger}},
  PlotLabel → "Ellipse hung on perpendicular frame",
  DisplayFunction → $DisplayFunction];
```



The parameterization of this ellipse is shown directly below the plot.
Grab the last two plots and animate.

Make it look pretty by stretching out the perpframe vectors:

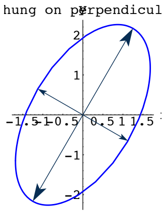
```
adjustedframeplot =
  {Arrow[xstretch perpframe[1], Tail → {0, 0}, VectorColor → Indigo],
   Arrow[-xstretch perpframe[1], Tail → {0, 0}, VectorColor → Indigo],
   Arrow[ystretch perpframe[2], Tail → {0, 0}, VectorColor → Indigo],
   Arrow[-ystretch perpframe[2],
    Tail → {0, 0}, VectorColor → Indigo]};

Show[hung, adjustedframeplot, AspectRatio → Automatic,
  PlotLabel → "Axes for tilted ellipse",
  DisplayFunction → $DisplayFunction];
```



□ T.2.a.ii) Measuring the area enclosed by a tilted ellipse

Here's another look at that hung ellipse at the end of part i}

```
Show[hung, adjustedframeplot, AspectRatio -> Automatic,
PlotLabel -> "Ellipse hung on perpendicular frame",
 DisplayFunction -> $DisplayFunction];
```



Measure the area enclosed by this ellipse.

□ Answer:

The area of this ellipse measures out to (xstretch) (ystretch) π

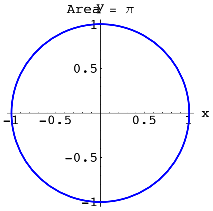
```
(xstretch) (ystretch) π
10.2102
```

Reason:

To make this ellipse you start by plotting the unit circle.

The area of the unit circle measures out to π square units:

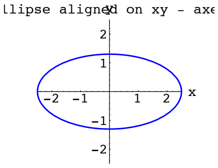
```
Show[circleplot, PlotLabel -> "Area = π"];
```



Then you stretch this circle out the x-axis by a factor of xstretch

and then you stretch the result out the y-axis by a factor of ystretch:

```
Show[stretched];
```

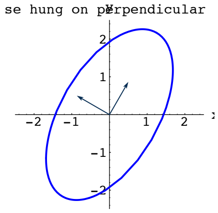


The area enclosed by this aligned ellipse (See T.1) measures out to

(xstretch) (ystretch) π:

Next you hang this ellipse on the plotted perpendicular frame:

```
Show[hung, frameplot, AspectRatio -> Automatic,
 PlotRange -> {{-ranger, ranger}, {-ranger, ranger}},
 PlotLabel -> "Ellipse hung on perpendicular frame",
DisplayFunction -> $DisplayFunction];
```



This hung ellipse is the same physical ellipse as the original version immediately above. So the area it encloses measures to the same thing as the area enclosed by the original ellipse. Namely -

(xstretch) (ystretch) π .

□ T.2.b.i) Plotting tilted 3D ellipsoids

Use stretch factors and perpendicular frames to plot tilted 3D ellipsoids.

□ Answer:

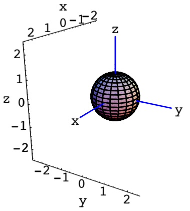
You plot the sphere of radius 1 centered at {0,0,0} by plotting

{Sin[s] Cos[t], Sin[s] Sin[t], Cos[s]}

running t from 0 to 2 π and running s from 0 to π:

```
Clear[s, t];
{slow, shigh} = {0, π};
{tlow, thigh} = {0, 2 π};
```

```
ranger = 2.5;
sphereplot = ParametricPlot3D[{Sin[s] Cos[t], Sin[s] Sin[t], Cos[s]},
    {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[sphereplot, Axes3D[ranger], DisplayFunction → $DisplayFunction];
```

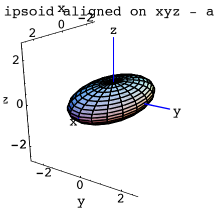


To plot an ellipsoid centered at {0,0,0} and aligned on the x , y and z axes you stretch the sphere out on the x, y and z -axes by selecting stretch factors xstretch, ystretch and zstretch and then plotting

{x[t], y[t], z[t]} =
{xstretch Sin[s] Cos[t], ystretch Sin[s] Sin[t], zstretch Cos[s]}

like this:

```
xstretch = 2.75;
ystretch = 1.60;
zstretch = 0.76;
ranger = Max[{xstretch, ystretch, zstretch}];
Clear[x, y, z, s, t];
{x[s_, t_], y[s_, t_], z[s_, t_]} =
  {xstretch Sin[s] Cos[t], ystretch Sin[s]  Sin[t], zstretch Cos[s]};

stretchedellipse  = ParametricPlot3D[{x[s, t], y[s, t], z[s, t]},
    {s, slow, shigh}, {t, tlow, thigh}, DisplayFunction -> Identity];

 stretchedplot  =
  Show[stretchedellipse, Axes3D[ranger], PlotRange ->
     {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
         Axes -> True, AxesLabel -> {"x", "y", "z"},
     Boxed -> False, ViewPoint -> CMView,
     PlotLabel -> "Ellipsoid aligned on xyz - axes",
   DisplayFunction -> $DisplayFunction];
{x[s, t], y[s, t], z[s, t]}
```



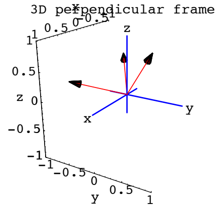
```
{2.75 Cos[t] Sin[s], 1.6 Sin[s] Sin[t], 0.76 Cos[s]}
```

The parameterization {x[s,t],y[s,t],z[s,t]} of this ellipsoid is shown directly below the plot.

Next you choose a perpendicular frame.

Here's a sample (later in the course you will see where this formula comes from):

```
r = π/3;
s = 0.4 π;
t = π/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]}}

frameplot = {Arrow[perpframe[1], Tail → {0, 0, 0}, VectorColor → Red],
    Arrow[perpframe[2], Tail → {0, 0, 0}, VectorColor → Red],
    Arrow[perpframe[3], Tail → {0, 0, 0}, VectorColor → Red]};

Show[frameplot, Axes3D[1],
  PlotRange → {{-1, 1}, {-1, 1}, {-1, 1}}, Axes → True,
  AxesLabel → {"x", "y", "z"}, Boxed → False, ViewPoint → CMView,
  AspectRatio → Automatic, PlotLabel → "3D perpendicular frame"];
{{0.16432, 0.542787, 0.823639},
 {-0.721626, -0.503118, 0.475528}, {0.672499, -0.672499, 0.309017}}
```



And then you hang that ellipse on the chosen perpendicular frame with

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

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

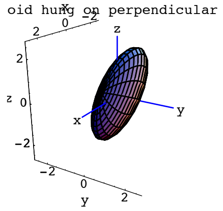
and with

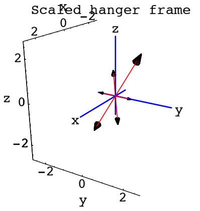
perpframe[3] playing the former role of{0,0,1} pointing out the positive z-axis

by plotting

x[s, t] perpframe[1] + y[s, t] perpframe[2] + z[s, t] perpframe[3]:

```
hung = ParametricPlot3D[
  x[s, t] perpframe[1] + y[s, t] perpframe[2] + z[s, t] perpframe[3],
  {s, slow, shigh}, {t, tlow, thigh}, DisplayFunction → Identity];

hungplot = Show[hung, Axes3D[ranger], PlotRange →
    {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
   Axes → True, AxesLabel → {"x", "y", "z"},
   Boxed → False, ViewPoint → CMView,
   PlotLabel → "Ellipsoid hung on perpendicular frame",
   DisplayFunction → $DisplayFunction];

adjustedframeplot =
  {Arrow[xstretch perpframe[1], Tail → {0, 0, 0}, VectorColor → Red],
   Arrow[ystretch perpframe[2], Tail → {0, 0, 0}, VectorColor → Red],
   Arrow[zstretch perpframe[3], Tail → {0, 0, 0}, VectorColor → Red],
   Arrow[-xstretch perpframe[1], Tail → {0, 0, 0}, VectorColor → Red],
   Arrow[-ystretch perpframe[2], Tail → {0, 0, 0}, VectorColor → Red],
   Arrow[-zstretch perpframe[3], Tail → {0, 0, 0}, VectorColor → Red]};

Show[adjustedframeplot, Axes3D[ranger], PlotRange →
   {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
  Axes → True, AxesLabel → {"x", "y", "z"}, Boxed → False,
  ViewPoint → CMView, PlotLabel → "Scaled hanger frame",
  AspectRatio → Automatic];
```





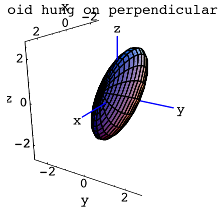
Grab and animate both plots slowly and then fast.

Serious 3D action..

□T.2.b.ii) Measuring the volume enclosed by a tilted 3D ellipsoid

Here's another look at that hung ellipsoid at the end of part i}

```
Show[hung, Axes3D[ranger],
 PlotRange ->
    {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
         Axes -> True, AxesLabel -> {"x", "y", "z"},
     Boxed -> False, ViewPoint -> CMView,
     PlotLabel -> "Ellipsoid hung on perpendicular frame",
     DisplayFunction -> $DisplayFunction];
```



Measure the volume enclosed by this ellipsoid.

□ **Answer:**

The volume of this hung ellipsoid measures out to

(xstretch) (ystretch) (zstretch) (4/3) π cubic units

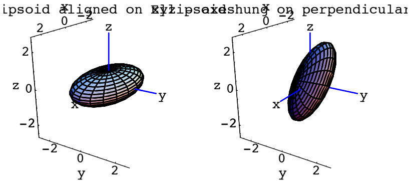
```
(xstretch) (ystretch) (zstretch) (4 / 3) π
```

```
14.0073
```

Reason:

The stretched aligned ellipse and the hung ellipse:

```
Show[GraphicsArray[{stretchedplot, hungplot}]];
```



are the same physical ellipse. Only their positions are different. The volume enclosed by stretched aligned ellipse measures out to

(xstretch) (ystretch) (zstretch) (4/3) π cubic units

```
(xstretch) (ystretch) (zstretch) (4 / 3) π
```

```
14.0073
```

T.3) Lines through {0,0} in 2D and lines through {0,0,0} in 3D; using perpendicular frame components to come up with closest points

□ T.3.a.i) Plotting 2D lines through {0,0}

Here's a unit vector in 2D:

```
s = 0.4;
U = {Cos[s], Sin[s]}
```

```
{0.921061, 0.389418}
```

Come up with a parametric formula for the line that runs through {0, 0} and runs parallel to U.
Show the line, the vector, and the point in a single plot.

□ Answer:

Here comes the parametric formula:

```
Clear[line, t];
line[t_] = t U
```

```
{0.921061 t, 0.389418 t}
```

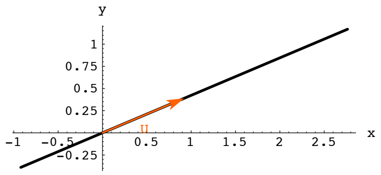
Note how the parametric formula displays the unit vector U:

```
U
```

```
{0.921061, 0.389418}
```

Here comes the plot:

```
unitvectorplot = Arrow[U, Tail → {0, 0},
    VectorColor → CadmiumOrange, HeadSize -> 0.2];
lineplot = ParametricPlot[line[t], {t, -1, 3},
    PlotStyle → Thickness[0.008],
    AxesLabel → {"x", "y"}, DisplayFunction → Identity];
Ulabel = Graphics[{CadmiumOrange, Text["U", 0.6 U, {1, 2}]}];

all = Show[lineplot, unitvectorplot,
    Ulabel, DisplayFunction → $DisplayFunction];
```

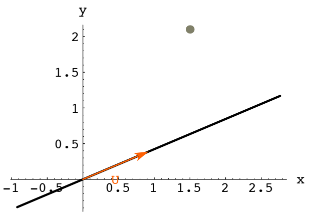


Done.

□ T.3.a.ii) Closest points via components

Here's the line from part i) plotted with point:

```
point = {1.5, 2.1};
pointplot = Graphics[{WarmGray, PointSize[0.03], Point[point]}];

setup = Show[all, pointplot];
```



Come up with the point on the line closest to the plotted point.

□ Answer:

Take the component of the plotted point in the direction of U:

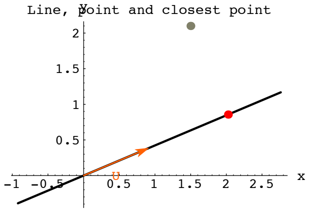
```
closestpoint = (point.U) U
```

```
{2.02575, 0.856475}
```

U is the unit vector that defines the direction of the line.

See it:

```
closestplot =
  Show[setup, Graphics[{Red, PointSize[0.03], Point[closestpoint]}],
   PlotLabel -> "Line, point and closest point"];
```

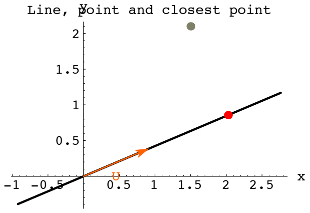


Looking good and feeling great.

□ T.3.a.iii) Why that worked

Take another look at the last plot:

```
Show[closestplot];
```

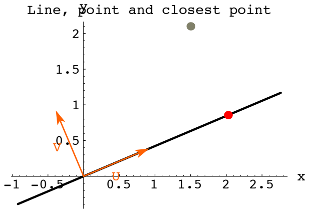


To get that closest point, you took the line defined by the unit vector U a found that the point on the line closest to the given point is the component of the point in the direction of U.
Explain why this was guaranteed to work.

□ Answer:

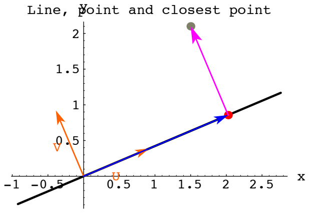
Throw in a unit vector V so that {U,V} is a perpendicular frame:

```
V = {Cos[s + π/2], Sin[s + π/2]};
Vlabel = Graphics[{CadmiumOrange, Text["V", 0.7 V, {1, 2}]}];

more = Show[closestplot, Arrow[V, Tail -> {0, 0},
    VectorColor -> CadmiumOrange, HeadSize -> 0.2], Vlabel];
```



Resolve the point into perpendicular frame components parallel to U and V:

```
Show[more, Arrow[(point.U) U, Tail -> {0, 0},
                VectorColor -> Blue, HeadSize -> 0.2],
  Arrow[(point.V) V, Tail -> (point.U) U, VectorColor -> Magenta]];
```



That component of the point in the direction of V (magenta) is perpendicular to the line and runs through the given point. Because perpendicular distance is the shortest distance, that plotted point on the line is closest to the given point off the line.

□ T.3.b.i) 3D lines

Here's a unit vector in 3D:

```
s = 0.4;
t = -0.2;
U = {Sin[s] Cos[t], Sin[s] Sin[t], Cos[s]}
```

```
{0.381656, -0.0773655, 0.921061}
```

Come up with a parametric formula for the line that runs through {0, 0, 0} and runs parallel to U.
Show the line, the vector, and the point in a single plot.

□ **Answer:**

Here comes the parametric formula:

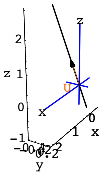
```
Clear[line, t];
line[t_] = t U
{0.381656 t, -0.0773655 t, 0.921061 t}
```

Note how the parametric formula displays the unit vector U:

```
U
{0.381656, -0.0773655, 0.921061}
```

Here comes the plot:

```
unitvectorplot = Arrow[U, Tail → {0, 0, 0},
   VectorColor → CadmiumOrange, HeadSize -> 0.2];
lineplot = ParametricPlot3D[line[t], {t, -1, 3}, Axes -> True,
  AxesLabel → {"x", "y", "z"}, DisplayFunction → Identity];
Ulabel = Graphics3D[{CadmiumOrange, Text["U", 0.6 U, {1, 2}]}];

all = Show[lineplot, unitvectorplot,
  Ulabel,  Axes3D[2, 0.1], Boxed → False,
  ViewPoint -> CMView, DisplayFunction → $DisplayFunction];
```

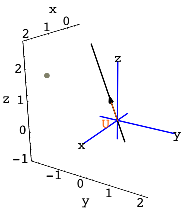


Done.

□ T.3.b.ii) Closest points in 3D via components

Here's the line from part i) plotted with a point:

```
givenpoint = {1.3, -1.8, 1.6};
pointplot =
  Graphics3D[{WarmGray, PointSize[0.03], Point[givenpoint]}];
setup = Show[all, pointplot, PlotRange -> All];
```



Come up with the point on the line closest to the plotted point and plot it.

□ **Answer:**

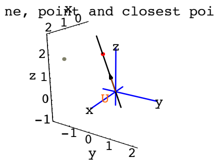
You do the same thing you do in 2D.

The point on the line closest to the given point is the component of the given point in the direction of the unit vector:

```
closestpoint = (givenpoint.U) U
{0.804954, -0.163172, 1.94262}
        U is the unit vector that defines the direction of the line.
```

See it:

```
closestplot = Show[setup,
  Graphics3D[{Red, PointSize[0.03], Point[closestpoint]}],
  PlotLabel -> "Line, point and closest point"];
```



Just as in 2D, the vector running from the red point on the line to the given point is perpendicular to the line:

```
(givenpoint - closestpoint).U
2.92579 × 10^-18
```

Because perpendicular distance is the shortest distance, that plotted point on the line is closest to the given point off the line.

T.4) Planes through {0,0,0} in 3D, closest points, plotting 2D curves on 3D planes

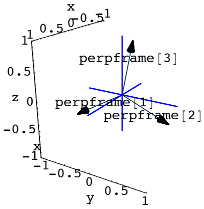
□ T.4.a.i) Using a perpendicular frame to plot a plane in 3D.

How do you use a perpendicular frame to plot a plane in 3D?

□ **Answer:**

One easy way to get a plane through {0,0,0} is to start with a 3D perpendicular frame:

```
r = -0.3;
s = -0.25;
t = 0.1;
Clear[perpframe];
{perpframe[1], perpframe[2], perpframe[3]} =
  {{Cos[r] Cos[t] - Cos[s] Sin[r] Sin[t],
    Cos[t] Sin[r] + Cos[r] Cos[s] Sin[t], Sin[s] Sin[t]},
   {-Cos[s] Cos[t] Sin[r] - Cos[r] Sin[t],
    Cos[r] Cos[s] Cos[t] - Sin[r] Sin[t], Cos[t] Sin[s]},
   {Sin[r] Sin[s], -Cos[r] 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"}];
```



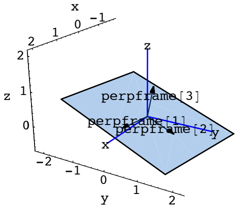
You get the plane through {0,0,0} determined by perpframe[1] and perpframe[2] by going with all combinations

s perpframe[1] + t perpframe[2]

with s and t running over all real numbers.

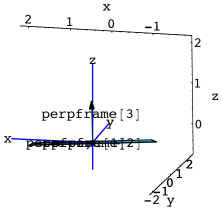
Here's how you plot a piece of the plane through {0,0,0} determined by perpframe[1] and perpframe[2]:

```
Clear[s, t];
s perpframe[1] + t perpframe[2];
{slow, shigh} = {-1.5, 1.5};
{tlow, thigh} = {-2.0, 2.0};

plane =
  ParametricPlot3D[s perpframe[1] + t perpframe[2], {s, slow, shigh},
   {t, tlow, thigh}, PlotPoints -> {2, 2}, DisplayFunction → Identity];

planeplot = Show[frameplot, plane, Axes → Automatic,
   PlotRange → All, ViewPoint → CMView, Boxed -> False,
   BoxRatios → Automatic, DisplayFunction → $DisplayFunction];
```



See it from a viewpoint in the direction of perpframe[1]:

```
Show[planeplot, ViewPoint -> 12 perpframe[2]];
```



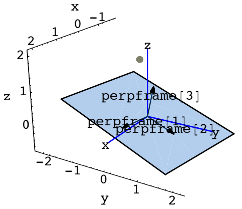
Perpframe[1] and perpframe[2] are in the plane.

Perpframe[3] is perpendicular to the plane.

□ T.4.a.ii) Closest points

Here's that same plane again shown with a point not on the plane.

```
point = {-0.4, -0.5, 1.5};
pointplot = Graphics3D[{WarmGray, PointSize[0.03], Point[point]}];
newsetup = Show[planeplot, pointplot];
```



Come up with the point on the plane closest to the plotted point.
Throw this point into the plot.

□ **Answer:**

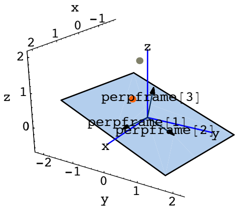
Add the perpendicular frame components of the point in the directions of perpframe[1] and perpframe[2]:

```
closestpoint =
    (point.perpframe[1]) perpframe[1] +
  (point.perpframe[2]) perpframe[2]
```

```
{-0.495481, -0.808666, 0.234652}
```

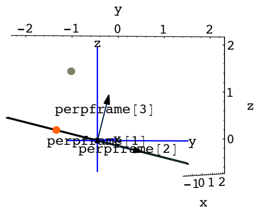
See it:

```
closestpointplot =
  Graphics3D[{CadmiumOrange, PointSize[0.03], Point[closestpoint]}];
 closeplot = Show[newsetup, closestpointplot];
```



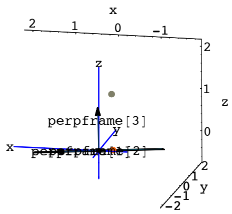
See it from a viewpoint far out in the direction of perpframe[1]::

```
Show[closeplot, ViewPoint -> 30 perpframe[1]];
```



See it from a viewpoint far out in the direction of perpframe[2]::

```
Show[closeplot, ViewPoint -> 30 perpframe[2]];
```



Lookin' good.

Notice that the line segment running from the closest point to the given point is parallel to perpframe[3] and is perpendicular to the plane.

If you want to see why this calculation worked, go on to the next part.

□ T.4.a.iii) Why the calculation works

Here is what happened in part i) above:
Given a perpendicular frame {perpframe[1], perpframe[2], perpframe[3]},
you make the plane through {0,0,0} determined by perpframe[1] and perpframe[2] by taking all combinations

s perpframe[1] + t perpframe[2]

with s and t running over all real numbers.

```
Fancy folks often like to call s perpframe[1] + t perpframe[2] by the name
        "linear combonation of perpframe[1] and perpframe[2]."
```

According to the calculation in part i), if {x, y, z} is a given 3D point, then the point on the plane closest to {x, y, z} is

closest = ..

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

Explain why this calculation is guaranteed to work.

□ **Answer:**

Break up {x,y,z} into components in the directions of perpframe[1], perpframe[2] and perpframe[3] this way:

comp[1] = ({x, y, z}.perpframe[1]) perpframe[1]

comp[2] = ({x, y, z}.perpframe[2]) perpframe[2]

comp[3] = {x, y, z}.perpframe[3] perpframe[3].

At this stage, you are guaranteed that

{x, y, z} = comp[1] + comp[2] + comp[3].

You are also guaranteed that

comp[1] + comp[2]

is in the plane through {0,0,0} determined by perpframe[1] and perpframe[2].

```
                Reason:
  comp[1] runs in the direction of perpframe[1]
and comp[2] runs in the direction of perpframe[2].
```

Now to get from

comp[1] + comp[2] to {x, y, z},

you move along comp[3], a vector which is perpendicular to the plane.

```
                Reason:
  comp[3] runs in the direction of perpframe[3]
  and perpframe[3] is perpendicular to the plane.
```

In other words, the vector running from

comp[1] + comp[2] to {x, y, z}

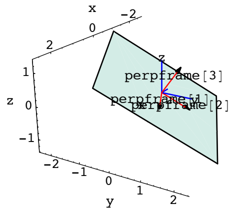
is perpendicular to the plane.

And because perpendicular distance is the shortest distance, you are guaranteed that
comp[1] + comp[2] = ({x,y,z}.perpframe[1]) perpframe[1]+ ({x,y,z}.perpframe[2]) perpframe[2]
is the point on the plane closest to {x,y,z}.
Explanation complete.

□ T.4.b) Hanging 2D curves on 3D planes

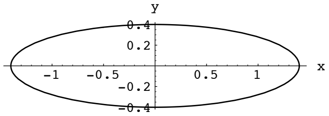
Here's a perpendicular frame {perpframe[1],perpframe[2],perpframe[3]} and part of the plane through {0,0,0} determined by perpframe[1] and perpframe[2]:

```
r = 0.3;
s = -0.6;
t = 0.2;
Clear[perpframe];
{perpframe[1], perpframe[2], perpframe[3]} =
  {{Cos[r] Cos[t] - Cos[s] Sin[r] Sin[t],
    Cos[t] Sin[r] + Cos[r] Cos[s] Sin[t], Sin[s] Sin[t]},
   {-Cos[s] Cos[t] Sin[r] - Cos[r] Sin[t],
    Cos[r] Cos[s] Cos[t] - Sin[r] Sin[t], Cos[t] Sin[s]},
   {Sin[r] Sin[s], -Cos[r] Sin[s], Cos[s]}};
frameplot = {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[planeplotter, s, t];
planeplotter[s_, t_] = s perpframe[1] + t perpframe[2];
{slow, shigh} = {-2, 2};
{tlow, thigh} = {-2, 2};

plane =
  ParametricPlot3D[Evaluate[planeplotter[s, t]], {s, slow, shigh},
   {t, tlow, thigh}, PlotPoints -> {2, 2}, DisplayFunction → Identity];

planeplot = Show[frameplot, plane, Axes3D[1.0, 0.1],

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



Here's an ellipse plotted in 2D:

```
Clear[x, y, t];
{x[t_], y[t_]} = {1.4 Cos[t], 0.4 Sin[t]};
ellipseplot =
  ParametricPlot[{x[t], y[t]}, {t, 0, 2 Pi}, AxesLabel -> {"x", "y"}];
```



You job is to duplicate this ellipse by hanging it on the plane with perpframe[1] playing the role of {1,0} (x-axis) and perpframe[2] playing the role of {0,1} (y-axis).

□ **Answer:**

The function used to plot the ellipse is:

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

To hang this curve on the plane with

perpframe[1] playing the role of {1,0} (x-axis)

and

perpframe[2] playing the role of {0,1} (y-axis) ,

change

{1,0} to perpframe[1]

and change

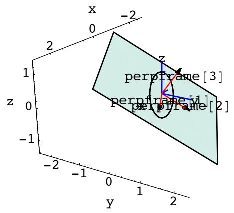
{0,1} to perpframe[2]

to get

x[t] perpframe[1] + y[t] perpframe[2]

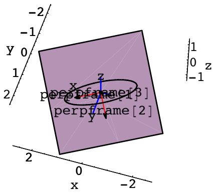
and plotting. Try it out:

```
hungellipseplot =
  ParametricPlot3D[ x[t] perpframe[1] + y[t] perpframe[2] ,
   {t, 0, 2 Pi}, DisplayFunction -> Identity];

all = Show[planeplot,
   hungellipseplot, DisplayFunction → $DisplayFunction];
```



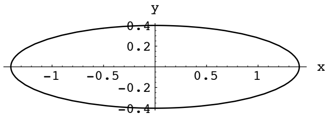
See it from the viewpoint far out in the direction of perpframe[3]:

```
Show[all, ViewPoint -> 12 perpframe[3]];
```



Compare:

```
Show[ellipseplot];
```



Dandy.

T.5) The cross product X×Y of two 3D vectors is perpendicular to both X and Y

Using the cross product to build a perpendicular frame

□T.5.a.i) The cross product X×Y of two 3D vectors is perpendicular to both X and Y

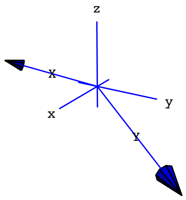
Take two vectors X and Y from the same dimension.
Testing for perpendicularity by checking whether
X.Y = 0
is quick and easy.
In three dimensions, another product comes to the front.
This product is also related to perpendicularity.
To calculate the cross product
X×Y for X = {a, b, c}; and Y = {d, e, f},
you can use this formula:

```
Clear[a, b, c, d, e, f];
X = {a, b, c};
Y = {d, e, f};
XcrossY = Cross[X, Y]
```

```
{-c e + b f, c d - a f, -b d + a e}
                              Don't try to memorize this formula.
```

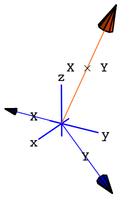
That's nasty and you will learn a short cut later on.
Meanwhile, see a sample 3D vector X and a sample 3D vector Y:

```
X = {0.9, -1.0, 0.5};
Y = {1.2, 2.0, -0.7};
XandY = Show[Arrow[X, Tail → {0, 0, 0}], Arrow[Y, Tail → {0, 0, 0}],
   Axes3D[1, 0.2], Graphics3D[Text["X", X/2]],
   Graphics3D[Text["Y", Y/2]], Boxed -> False,
   BoxRatios → Automatic, ViewPoint → CMView, PlotRange → All];
```



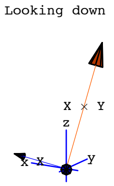
Now throw in the cross product X×Y:

```
XcrossY = Cross[X, Y];
crossplot = Show[XandY,
   Arrow[XcrossY, Tail → {0, 0, 0}, VectorColor → CadmiumOrange],
       Graphics3D[Text["X × Y", XcrossY/2]],
   ViewPoint → CMView, BoxRatios → Automatic, PlotRange → All];
```



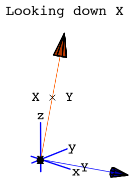
Golly, X×Y appears to be perpendicular to X and to Y.
See the same thing from a viewpoint far out in the direction of Y

```
Show[crossplot, ViewPoint -> 12 Y, PlotLabel -> "Looking down Y"];
```



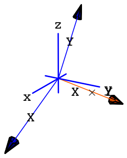
X×Y sure seems to be perpendicular to X.
See the same thing from a viewpoint far out in the direction of X

```
Show[crossplot, ViewPoint -> 12 X, PlotLabel -> "Looking down X"];
```

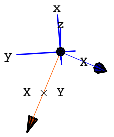


X×Y sure seems to be perpendicular to Y.
Try it again with new random vectors X and to Y:

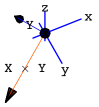
```
X = {Random[Real, {-2, 2}],
    Random[Real, {-2, 2}], Random[Real, {-2, 2}]};
Y = {Random[Real, {-2, 2}], Random[Real, {-2, 2}],
    Random[Real, {-2, 2}]};
XcrossY = Cross[X, Y];
crossplot = Show[Axes3D[1, 0.2],
    Arrow[X, Tail → {0, 0, 0}], Arrow[Y, Tail → {0, 0, 0}],
    Graphics3D[Text["X", X/2]], Graphics3D[Text["Y", Y/2]],
    Arrow[XcrossY, Tail → {0, 0, 0}, VectorColor → CadmiumOrange],
    Graphics3D[Text["X × Y", XcrossY/2]], Boxed -> False,
    ViewPoint → CMView, BoxRatios → Automatic, PlotRange → All];
Show[crossplot, ViewPoint -> 12 Y, PlotLabel -> "Looking down Y"];
Show[crossplot, ViewPoint -> 12 X, PlotLabel -> "Looking down X"];
```



Looking down Y



Looking down X



Rerun many times.

Each time, the cross product X × Y appears to be perpendicular to both X and Y.

For the record, explain why it is that when you take any two 3D 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[a, b, c, d, e, f];
X = {a, b, c};
Y = {d, e, f};
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!

□ T.5.a.ii) The length of X×Y is ||X || ||Y|| |Sin[angle between]|

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

```
Clear[a, b, c, d, e, f];
X = {a, b, c};
Y = {d, e, f};
XcrossY = Cross[X, Y];
Expand[(X.Y)^2 + XcrossY.XcrossY]
```

$a^2 d^2 + b^2 d^2 + c^2 d^2 + a^2 e^2 + b^2 e^2 + c^2 e^2 + a^2 f^2 + b^2 f^2 + c^2 f^2$

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

$a^2 d^2 + b^2 d^2 + c^2 d^2 + a^2 e^2 + b^2 e^2 + c^2 e^2 + a^2 f^2 + b^2 f^2 + c^2 f^2$

Remembering that

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

explain how these calculations reveal that

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

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

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

so that

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

□ **Answer:**

See the calculation again:

```
Clear[a, b, c, d, e, f];
X = {a, b, c};
Y = {d, e, f};
XcrossY = Cross[X, Y];
Expand[(X.Y)^2 + XcrossY.XcrossY]
```

$a^2 d^2 + b^2 d^2 + c^2 d^2 + a^2 e^2 + b^2 e^2 + c^2 e^2 + a^2 f^2 + b^2 f^2 + c^2 f^2$

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

$a^2 d^2 + b^2 d^2 + c^2 d^2 + a^2 e^2 + b^2 e^2 + c^2 e^2 + a^2 f^2 + b^2 f^2 + c^2 f^2$

The calculations reveal:

$(X.Y)^2 + (X \times Y).(X \times Y) = (X.X)\,(Y.Y).$

This is the same as:

$\|X\|^2 \, \|Y\|^2 \, \mathrm{Cos[angle\ between]}^2 + \|X \times Y\|^2 = \|X\|^2 \, \|Y\|^2.$

This is the same as:

$\|X \times Y\|^2 = \|X\|^2 \, \|Y\|^2 - \|X\|^2 \, \|Y\|^2 \, \mathrm{Cos[angle\ between]}^2$

This is the same as:

$\|X \times Y\|^2 = \|X\|^2 \, \|Y\|^2 \, (1 - \mathrm{Cos[angle\ between]}^2)$

This is the same as:

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

Reason: If θ is any angle, then

$\mathrm{Cos}[\theta]^2 + \mathrm{Sin}[\theta]^2 = 1.$

This is the same as:

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

Explanation complete.

□ T.5.a.iii) Using the cross product to extend two perpendicular unit vectors to a perpendicular frame

No matter what s you take, the following 3D vectors are perpendicular:

```
Clear[X, Y, s];
X[s_] = {Cos[s]/√2, Cos[s]/√2, Sin[s]};
Y[s_] = {-Sin[s]/√2, -Sin[s]/√2, Cos[s]};
Simplify[X[s].Y[s]]
```

0

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

```
Simplify[X[s].X[s]]
```

1

```
Simplify[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]];
Simplify[cross.cross]
```

1

Explain why:

If you take any two perpendicular 3D unit vectors X and Y, then

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

→ X × Y is also a unit vector.

so that

{X, Y, X × Y}

is a 3D perpendicular frame

□ **Answer:**

You are guaranteed that X × Y is perpendicular to both X and Y for any 3D vectors X and Y.

To see why X × Y is also a unit vector in the case that X and Y, are perpendicular unit vectors, use

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

In the case that X and Y, are perpendicular unit vectors,

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

reduces to

$\|X \times Y\| = (1)\,(1)\,|\, \mathrm{Sin[angle\ between]} \,|$

Reason: ||X|| = ||Y|| = 1

And now because X is perpendicular to Y,

$\|X \times Y\| = (1)\,(1)\,|\, \mathrm{Sin[angle\ between]} \,|$

reduces to

$\|X \times Y\| = (1)\,(1)\,(1) = 1$

Reason: $|\,\mathrm{Sin}[\frac{\pi}{2}]\,| = \mathrm{Sin}[-\frac{\pi}{2}] = 1$:

```
Abs[Sin[Pi / 2]] == 1
Abs[Sin[-Pi / 2]] == 1
```

```
True
True
```

This explains why when you start with two perpendicular 3D unit vectors X and Y, then

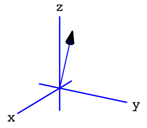
{X, Y, X × Y}

gives you a perpendicular frame.

□ T.5.b) Extending a single vector to a custom 3D perpendicular frame

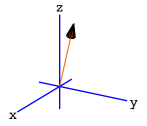
Here is a single vector X in 3D:

```
X = {0.2, 0.4, 1.3};
ranger = 1.5;

Show[Arrow[X, Tail → {0, 0, 0}, VectorColor → Blue],
  Axes3D[1.5, 0.2], PlotRange →
   {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
  ViewPoint → CMView, Boxed → False, AxesLabel → {"x", "y", "z"}];
```



Make X into a unit vector , unitX, and plot:

```
unitX = X/√(X.X);
ranger = 1;

Show[Arrow[unitX, Tail → {0, 0, 0}, VectorColor → CadmiumOrange],
  Axes3D[ranger, 0.1], PlotRange →
   {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
  ViewPoint → CMView, Boxed → False, AxesLabel → {"x", "y", "z"}];
```



Come up with a 3D perpendicular frame

{perpframe[1],perpframe[2],perpframe[3]}

with

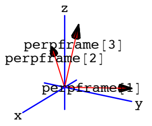
perpframe[3] = unitX.

Show off your perpendicular frame with a good plot.

□ **Answer:**

Do this:

```
perpframe[3] = unitX;
throwawayvector = {Random[Real, {-1, 1}],
   Random[Real, {-1, 1}], Random[Real, {-1, 1}]};
Y = throwawayvector × perpframe[3];
perpframe[1] = Y/√(Y.Y);
perpframe[2] = perpframe[3] × perpframe[1];

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[ranger, 0.1], PlotRange →
   {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
  Boxed → False, ViewPoint → CMView, AxesLabel → {"x", "y", "z"}];
```



Grab all three plots and animate.

Rerun both cells several times. You will probably get different perpframe[1] and perpframe[2] vectors each time.

□ T.5.b.ii) Why that worked

Why did that work?

□ **Answer:**

Look at the code but don't run it:

```
perpframe[3] = unitX;
throwawayvector = {Random[Real, {-1, 1}],
   Random[Real, {-1, 1}], Random[Real, {-1, 1}]};
Y = throwawayvector × perpframe[3];
perpframe[1] = Y/√(Y.Y);
perpframe[2] = perpframe[3] × perpframe[1];
```

Make note of the following facts:

→ Because Y is the cross product of unitX and another vector, Y is guaranteed to be perpendicular to unitX = perpframe[3].

→ perpframe[1] = Y/Sqrt[Y.Y] points in the same direction as Y. So perpframe[1] and perpframe[3] are perpendicular unit vectors.

→ Because perpframe[2] is the cross product of the perpendicular unit vectors perpframe[1] and perpframe[3], perpframe[2] is automatically (See part a.iii) above) a unit vector perpendicular to both perpframe[1] and perpframe[3] .

T.6) Plotting Tips:

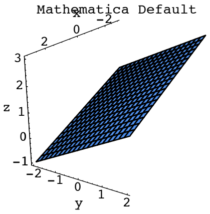
Flatness and plotting via the options PlotPoints and true scale plots via the options AspectRatio → Automatic and BoxRatios → Automatic

□ T.6.a.i) Flatness and the PlotPoints option

Here are three ways of plotting the same piece of the plane

$\frac{x}{3} - \frac{y}{2} + z = 1$:

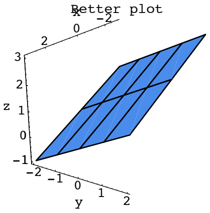
```
Clear[x, y];
defaultplot = ParametricPlot3D[{x, y, 1 - x/3 + y/2}, {x, -3, 3},
   {y, -2, 2}, ViewPoint → CMView, PlotRange → All, Boxed → False,
   PlotLabel → "Mathematica Default", AxesLabel → {"x", "y", "z"}];
```



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 → {3, 5},
  ViewPoint → CMView, PlotRange → All, Boxed → False,
  PlotLabel → "Better plot", AxesLabel → {"x", "y", "z"}];
```



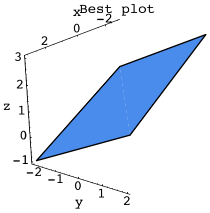
That ran a lot faster than the default plot.

The reason: Fewer bricks resulting from the plotting option

PlotPoints → {3, 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: The plotting option
    PlotPoints → {2, 2}.
Explain what's going on.

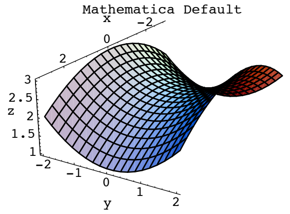
□ **Answer:**

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.

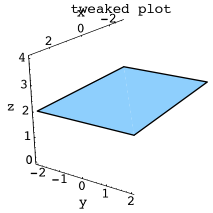
Here is a situation in which the default plot is very good:

```
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"}];
```



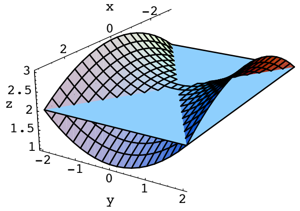
If you tinker with this, the way you tweaked the plot of the plane, then disaster strikes:

```
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.5.a.ii) Other situations that benefit from this type of tweaking

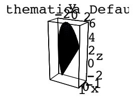
What other situations benefit from this type of tweaking?

□ **Answer:**

Anytime flatness is around.

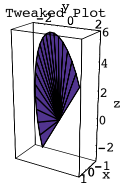
Look at this:

```
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:

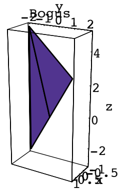
```
tweakedplot = ParametricPlot3D[r Cos[t] vector1 + r Sin[t] vector2,
  {r, 0, 1}, {t, 0, π}, PlotPoints → {2, Automatic}, ViewPoint → CMView,
  AxesLabel → {"x", "y", "z"}, PlotLabel → "Tweaked Plot"];
```



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.5.b) Perpendicularity and Aspect Ratios

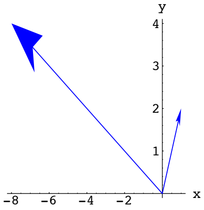
Look at this:

```
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];
```



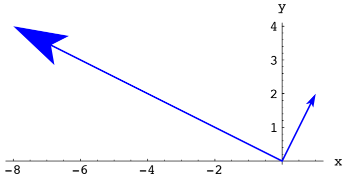
Although you knew in advance that these vectors are perpendicular, they didn't plot out perpendicularly.
What gives?

□ **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.