**Matrices, Geometry&*Mathematica***

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

**Producer: Bruce Carpenter**

**Publisher: Math Everywhere, Inc.**

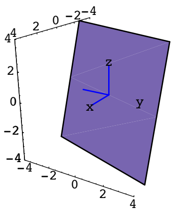
# MGM.05 3D Matrices
## *TUTORIALS*

## T.1) Making 3D perpendicular projections onto planes.
## Making 3D positive definite matrices (frame stretchers).
## Making 3D reflection matrices (plane flippers)
## Making matrices for bouncing light rays off surfaces in 3D

### □ T.1.a) Making a perpendicular projection onto a plane

Any two non-parallel vectors determine a plane through {0,0,0}.
Here's a sample:

```
Clear[planevector];
planevector[1] = {0.95, 1.71, -1.19};
planevector[2] = {0.24, 1.83, 1.06};

ranger = 4;
b = 3;
planeplot = Graphics3D[
    Polygon[{-b planevector[1] - b planevector[2], -b planevector[1] +
        b planevector[2], b planevector[1] + b planevector[2],
       b planevector[1] - b planevector[2]}]];

Show[planeplot, ThreeAxes[2], PlotRange →
    {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
   Axes → True, Boxed → False, ViewPoint → CMView];
```



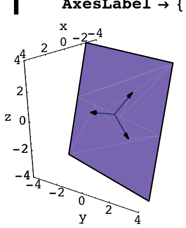
The question here is to come up with a matrix P so that when you hit a point {x,y,z} with your matrix P, you get the point on the plane that is closest to {x,y,z}.
Illustrate the action of your matrix with decisive plots.

**□ Answer:**

```
Lots of folks like to call this matrix by the name "perpendicular projection."
```

This is a job for a custom perpendicular frame. Here's one:

```
normal = planevector[1] × planevector[2];

unitnormal = normal / √(normal.normal);

perpframe[3] = unitnormal;

perpframe[1] = planevector[1] / √(planevector[1].planevector[1]);

perpframe[2] = perpframe[3] × perpframe[1];

scalefactor = 0.7 b;
frameplot = Table[Arrow[scalefactor perpframe[k],
     Tail → {0, 0, 0}, VectorColor → Indigo], {k, 1, 3}];

newplaneplot = Show[frameplot, planeplot, PlotRange →
     {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
    Axes → True, ViewPoint → CMView, Boxed → False,
    AxesLabel → {"x", "y", "z"}];
```



```
The normal vector is perpframe[3].
perpframe[1] and perpframe[2] frame the whole plane:
```

The job is to come up with a matrix P so that when you hit a point {x, y, z} with your matrix P, you get the point on the plane that is closest to {x, y, z}.

Notice that if {x, y, z} is on the plane, then {x, y, z} is the point on the plane closest to

{x, y, z} + any multiple of perpframe[3].

```
perpframe[3] is normal to the plane
and
the shortest distance from a point to the plane is the perpendicular distance.
```

In other words, if {x, y, z} is on the plane, then

P.({x, y, z} + any multiple of perpframe[3]) = {x, y, z}.

This tells you that

P.prepframe[3] = {0, 0, 0}

Also if {x, y, z} is on the plane, then {x, y, z} is the point on the plane closest to {x, y, z}.
And because perpframe[1] and perpframe[2] are on the plane, you now know that

P.perpframe[1] = perpframe[1]

and

P.perpframe[2] = perpframe[2].

This tells all.

The matrix P you want is:

```
Clear[alignerframe, hangerframe, k];
{alignerframe[1], alignerframe[2], alignerframe[3]} =
  {perpframe[1], perpframe[2], perpframe[3]};
aligner = {alignerframe[1], alignerframe[2], alignerframe[3]};

{xstretch, ystretch, zstretch} = {1, 1, 0};
stretcher = DiagonalMatrix[{xstretch, ystretch, zstretch}];

{hangerframe[1], hangerframe[2], hangerframe[3]} =
  {perpframe[1], perpframe[2], perpframe[3]};
hanger = Transpose[{hangerframe[1],
    hangerframe[2], hangerframe[3]}];

P = hanger.stretcher.aligner;
MatrixForm[P]
```

$$\begin{pmatrix} 0.177436 & 0.266458 & -0.273776 \\ 0.266458 & 0.913685 & 0.0886856 \\ -0.273776 & 0.0886856 & 0.908879 \end{pmatrix}$$

Check it:

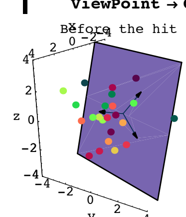
```
P.perpframe[1] == perpframe[1]
P.perpframe[2] == perpframe[2]
P.perpframe[3]
```

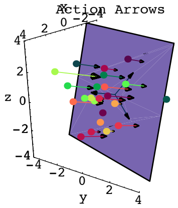
```
True
```

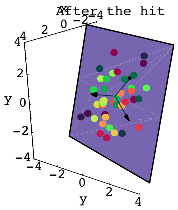
```
True
{0, 0, 0}
```

Watch this matrix do its work in this action movie:

```
a = 3;
Clear[k, pointcolor];
points = Table[{Random[Real, {-a, a}],
     Random[Real, {-a, a}], Random[Real, {-a, a}]}, {k, 1, 40}];
pointcolor[k_] = RGBColor[0.5 (Sin[2 π k / Length[points]] + 1),
    0.5 (Cos[2 π k / Length[points]] + 1), 0.3];

pointplot = Table[
   Graphics3D[{PointSize[0.04], pointcolor[k], Point[points[[k]]]}],
   {k, 1, Length[points]}];
hitpointplot = Table[Graphics3D[{PointSize[0.04], pointcolor[k],
      Point[P.points[[k]]]}], {k, 1, Length[points]}];

actionarrows = Table[Arrow[P.points[[k]] - points[[k]], Tail → points[[k]],
     VectorColor → pointcolor[k]], {k, 1, Length[points]}];

before = Show[pointplot, newplaneplot,
    Axes → True, AxesLabel → {"x", "y", "z"}, PlotRange →
     {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
    ViewPoint → CMView, Boxed → False, PlotLabel → "Before the hit"];

action = Show[before, actionarrows, PlotRange →
     {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
    ViewPoint → CMView, PlotLabel → "Action Arrows"];

after = Show[hitpointplot, newplaneplot, Axes → True, Axes → True,
    AxesLabel → {"x", "y", "y"}, Boxed → False, PlotRange →
     {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
    ViewPoint → CMView, PlotLabel → "After the hit"];
```



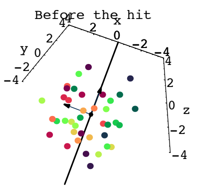


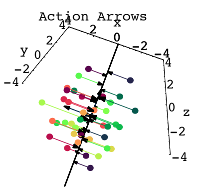



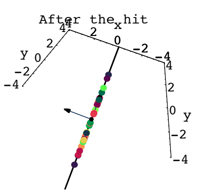

Grab, align and animate.

See the same thing from the view point of 6 perpframe[1].

```
Show[before, ViewPoint → 6 perpframe[1]];
Show[action, ViewPoint → 6 perpframe[1]];
Show[after, ViewPoint → 6 perpframe[1]];
```




Grab and animate.

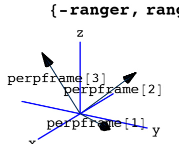
You can see why some folks call this matrix by the name "perpendicular projection".

### ◻ T.1.b) Making a 3D positive definite matrix (frame stretcher)

Here's a 3D perpendicular frame:

Where this formula comes from will be explained in one of the later Tutorials

```
{r, s, t} = N[{0.1 π, π/6, 0.2 π}];

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;
Show[
  Table[Arrow[perpframe[k], Tail → {0, 0, 0}, VectorColor → Indigo],
   {k, 1, 3}], Graphics3D[Text["perpframe[1]", 0.6 perpframe[1]]],
  Graphics3D[Text["perpframe[2]", 0.6 perpframe[2]]],
  Graphics3D[Text["perpframe[3]", 0.6 perpframe[3]]],
  ThreeAxes[1], ViewPoint → CMView, PlotRange → {{-ranger, ranger},
    {-ranger, ranger}, {-ranger, ranger}}, Boxed → False];
```




Make a 3D positive definite ( a frame stretcher) matrix A whose hits stretch
vectors in the direction of perpframe[1] by a factor of 2.5,
vectors in the direction of perpframe[2] by a factor of 4.1, and
vectors in the direction of perpframe[3] by a factor of 3.2.
Illustrate with plots.

◻ **Answer:**

You want a matrix A with

A.perpframe[1] = 2.5 perpframe[1]

A.perpframe[2] = 4.1 perpframe[2]

A.perpframe[3] = 3.2 perpframe[3].

To make A, you go with the given perpendicular frame for both your aligner frame and your hanger frame and use the indicated numbers for your stretches.

Here you go:

```
Clear[alignerframe, hangerframe, k];
{alignerframe[1], alignerframe[2], alignerframe[3]} =
  {perpframe[1], perpframe[2], perpframe[3]};
aligner = {alignerframe[1], alignerframe[2], alignerframe[3]};

{xstretch, ystretch, zstretch} = {2.5, 4.1, 3.2};
stretcher = DiagonalMatrix[{xstretch, ystretch, zstretch}];

{hangerframe[1], hangerframe[2], hangerframe[3]} =
  {perpframe[1], perpframe[2], perpframe[3]};
hanger = Transpose[{hangerframe[1],
    hangerframe[2], hangerframe[3]}];

A = hanger.stretcher.aligner;
MatrixForm[A]
```

$$\begin{pmatrix} 3.42276 & -0.652547 & -0.380391 \\ -0.652547 & 2.99044 & 0.123563 \\ -0.380391 & 0.123563 & 3.3868 \end{pmatrix}$$

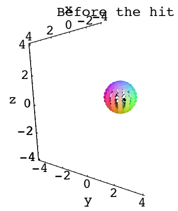
Check:

```
A.perpframe[1] == xstretch perpframe[1]
A.perpframe[2] == ystretch perpframe[2]
A.perpframe[3] == zstretch perpframe[3]
```

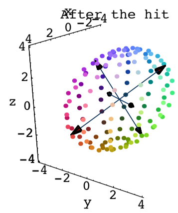
True
True
True

To illustrate, hit the unit sphere with A and see the resulting football:

```
Clear[x, y, s, t, pointcolor];
{x[s_, t_], y[s_, t_], z[s_, t_]} =
  {Sin[s] Cos[t], Sin[s] Sin[t], Cos[s]};

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

ranger = Max[{xstretch, ystretch, zstretch, 1.2}];
pointcolor[s_, t_] =
  RGBColor[0.5 (x[s, t] + 1), 0.5 (y[s, t] + 1), 0.5 (z[s, t] + 1)];
sjump = (shigh - slow)/12;
tjump = (thigh - tlow)/12;

Clear[hitplotter, hitpointplotter, matrix3D];
hitplotter[matrix3D_] :=
  ParametricPlot3D[matrix3D.{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];

hitpointplotter[matrix3D_] :=
  Show[{Table[Graphics3D[{pointcolor[s, t], PointSize[0.025],
       Point[matrix3D.{x[s, t], y[s, t], z[s, t]}]}],
     {s, slow, shigh - sjump, sjump}, {t, tlow, thigh - tjump, tjump}],
    Table[Arrow[matrix3D.alignerframe[k],
      Tail → {0, 0, 0}, VectorColor → Red], {k, 1, 3}],
    Table[Arrow[-matrix3D.alignerframe[k], Tail → {0, 0, 0},
      VectorColor → Red], {k, 1, 3}]}, PlotRange →
    {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
   Axes → True, AxesLabel → {"x", "y", "z"}, Boxed → False,
   ViewPoint → CMView, DisplayFunction → Identity];

hitframeplotter[matrix3D_] :=
  {Table[Arrow[matrix3D.alignerframe[k],
     Tail → {0, 0, 0}, VectorColor → Indigo], {k, 1, 3}],
   Table[Arrow[-matrix3D.alignerframe[k], Tail → {0, 0, 0},
     VectorColor → Indigo], {k, 1, 3}]};

pointsbefore = Show[hitpointplotter[IdentityMatrix[3]],
   hitframeplotter[IdentityMatrix[3]], PlotLabel → "Before the hit",
   DisplayFunction → $DisplayFunction];
```



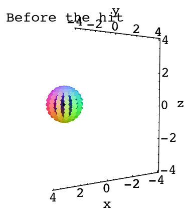

```
pointsafter = Show[hitpointplotter[A], hitframeplotter[A],
   PlotLabel → "After the hit", DisplayFunction → $DisplayFunction];
```

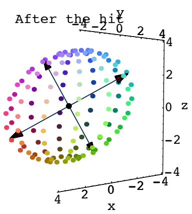



Grab both plots and animate at various speeds.

See the same thing from the view points of each of the given perpendicular frame vectors:

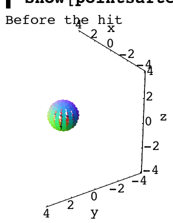
```
Show[pointsbefore, ViewPoint → 10 perpframe[1]];
Show[pointsafter, ViewPoint → 10 perpframe[1]];
```

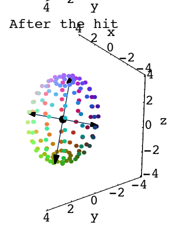



Grab both plots and animate at various speeds.

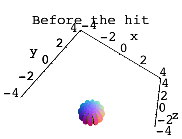
```
Show[pointsbefore, ViewPoint → 10 perpframe[2]];
Show[pointsafter, ViewPoint → 10 perpframe[2]];
```

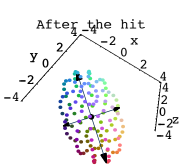



Grab both plots and animate at various speeds.

```
Show[pointsbefore, ViewPoint → 10 perpframe[3]];
Show[pointsafter, ViewPoint → 10 perpframe[3]];
```



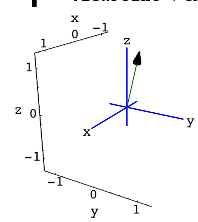

Grab both plots and animate at various speeds.

About as sensual as math gets.

### T.1.c) Making a 3D reflection matrix ( plane flipper)

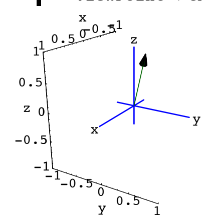
Here is a single vector in 3D:

```
normal = {0.2, 0.4, 1.3};

ranger = Max[{normal⟦1⟧, normal⟦2⟧, normal⟦3⟧}];

Show[Arrow[normal, Tail → {0, 0, 0}, VectorColor → GosiaGreen],
  Axes3D[ranger], PlotRange → {{-ranger, ranger},
    {-ranger, ranger}, {-ranger, ranger}}, Axes → True,
  ViewPoint → CMView, AxesLabel → {"x", "y", "z"}, Boxed → False];
```



Make this vector into a unit vector and plot:

```
unitnormal = normal / √(normal.normal);

ranger = 1;
Show[Arrow[unitnormal, Tail → {0, 0, 0}, VectorColor → GosiaGreen],
  Axes3D[1], PlotRange → {{-ranger, ranger},
    {-ranger, ranger}, {-ranger, ranger}}, Axes → True,
  ViewPoint → CMView, AxesLabel → {"x", "y", "z"}, Boxed → False];
```

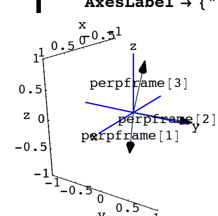


Envision this unit vector as a unit normal for a plane passing through {0,0,0}.
Call

perpframe[3] = unitnormal

You can use cross products to come up with
perpframe[1] and perpframe[2]
to frame this plane.
Here's how it goes:

```
perpframe[3] = unitnormal;

throwawayvector = {Random[Real, {-1, 1}],
   Random[Real, {-1, 1}], Random[Real, {-1, 1}]};

planevector = throwawayvector × perpframe[3];

perpframe[1] = planevector / Norm[planevector];

perpframe[2] = perpframe[3] × perpframe[1];

frameplot = Table[Arrow[perpframe[k],
    Tail → {0, 0, 0}, VectorColor → Indigo], {k, 1, 3}];
framelabels = {Graphics3D[Text["perpframe[1]", 0.6 perpframe[1]]],
   Graphics3D[Text["perpframe[2]", 0.6 perpframe[2]]],
   Graphics3D[Text["perpframe[3]", 0.6 perpframe[3]]]};

customframe = Show[frameplot, framelabels, ThreeAxes[1], PlotRange →
    {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
   Axes → True, ViewPoint → CMView, Boxed → False,
   AxesLabel → {"x", "y", "z"}];
```

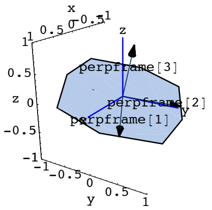



Rerun until you get a nice one.

Throw in a plot of the plane framed by perpframe[1] and perpframe[2]:

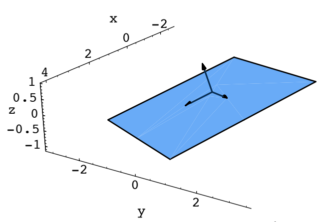
```
{slow, shigh} = {-1, 1};
{tlow, thigh} = {-1, 1};
ranger = 1.4;
Clear[planeplotter, s, t];
planeplotter[s_, t_] = s perpframe[1] + t perpframe[2];
planeplot = ParametricPlot3D[planeplotter[s, t],
```

```
   {s, slow, shigh}, {t, tlow, thigh}, PlotRange → All,
   PlotPoints → {2, 2}, DisplayFunction → Identity];

planeframe = Show[customframe, planeplot];
```




Rerun both cells several times. You will probably get different perpframe[1] and perpframe[2] vectors each time, but you will always get the same plane because you get the same normal vector (perpframe[3]) each time.
Here's a new plane through {0,0,0} and a perpendicular frame set so that perpframe[1] and perpframe[2] frame the plane:

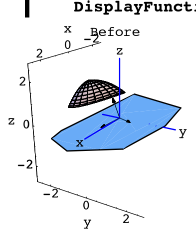
```
normal = {0.2, -0.3, 1.1};
perpframe[3] = normal / Norm[normal];
throwawayvector = {Random[Real, {-1, 1}],
   Random[Real, {-1, 1}], Random[Real, {-1, 1}]};
planevector = throwawayvector × perpframe[3];
perpframe[1] = planevector / Norm[planevector];
perpframe[2] = perpframe[3] × perpframe[1];

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

{slow, shigh} = {-2, 3};
{tlow, thigh} = {-2, 3};

Clear[planeplotter, s, t];
planeplotter[s_, t_] = s perpframe[1] + t perpframe[2];
planeplot = ParametricPlot3D[planeplotter[s, t],
   {s, slow, shigh}, {t, tlow, thigh}, PlotRange → All,
   PlotPoints → {2, 2}, DisplayFunction → Identity];

planeandframe =
  Show[planeplot, frameplot, PlotRange → All, BoxRatios → Automatic,
   Axes → True, AxesLabel → {"x", "y", "z"}, Boxed → False,
   ViewPoint → CMView, DisplayFunction → $DisplayFunction];
```



That's perpframe[3] = unit normal vector sticking out of the plane.

Throw in a 3D surface:

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

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

sjump = (shigh - slow) / 5;
tjump = (thigh - tlow) / 5;

ranger = 2.8;
Clear[hitplotter, matrix];
hitplotter[matrix3D_] :=
  ParametricPlot3D[matrix3D.{x[s, t], y[s, t], z[s, t]},
   {s, slow, shigh}, {t, tlow, thigh}, PlotRange →
    {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
   BoxRatios → Automatic, Axes → True, AxesLabel → {"x", "y", "z"},
   Boxed → False, ViewPoint → CMView, DisplayFunction → Identity];
before = Show[hitplotter[IdentityMatrix[3]],
   planeandframe, Axes3D[ranger], PlotLabel → "Before",
   DisplayFunction → $DisplayFunction];
```




Use a 3D matrix hit to flip this surface underneath the plane.

□ **Answer:**

Remember that perpframe[3] is perpendicular to the plane and perpframe[1] and perpframe[2] frame the plane.
To make a matrix whose hits flip about the plane,
→ align with {perpframe[1], perpframe[2], perpframe[3]}
→ go with stretch factors all equal to 1
→ hang on the reversed frame {perpframe[1], perpframe[2], -perpframe[3]}:

Note the minus sign on perpframe[3].

```
{alignerframe[1], alignerframe[2], alignerframe[3]} =
                              {perpframe[1],
   perpframe[2], perpframe[3]};

{xstretch, ystretch, zstretch} = {1, 1, 1};

{hangerframe[1], hangerframe[2], hangerframe[3]} =
                              {perpframe[1],
  perpframe[2], -perpframe[3]};

aligner = {alignerframe[1], alignerframe[2], alignerframe[3]};
stretcher = DiagonalMatrix[{xstretch, ystretch, zstretch}];
hanger =
  Transpose[{hangerframe[1], hangerframe[2], hangerframe[3]}];

A = hanger.stretcher.aligner;
MatrixForm[A]
```

$$\begin{pmatrix} 0.940299 & 0.0895522 & -0.328358 \\ 0.0895522 & 0.865672 & 0.492537 \\ -0.328358 & 0.492537 & -0.80597 \end{pmatrix}$$

Hits with this matrix preserve everything in the plane framed by perpframe[1] and perpframe[2]:

```
Clear[s, t];
Expand[A.(s perpframe[1] + t perpframe[2])]
```

{0.828234 s + 0.533084 t,
 -0.483926 s + 0.835854 t, -0.282568 s + 0.131036 t}

```
s perpframe[1] + t perpframe[2]
```

{0.828234 s + 0.533084 t,
 -0.483926 s + 0.835854 t, -0.282568 s + 0.131036 t}

Hits with this matrix reverse the direction of anything in the direction of perpframe[3]:

```
A.(s perpframe[3])
```

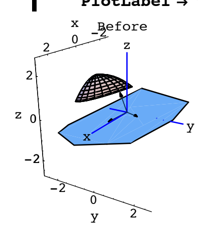
{-0.172774 s, 0.259161 s, -0.950255 s}

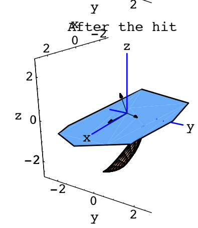
```
-s perpframe[3]
```

{-0.172774 s, 0.259161 s, -0.950255 s}

See a hit with this matrix do the flip:

```
Show[before];
after = Show[hitplotter[A], planeandframe, Axes3D[ranger],
    PlotLabel → "After the hit", DisplayFunction → $DisplayFunction];
```

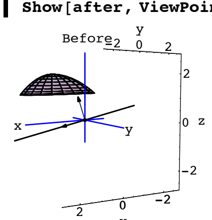


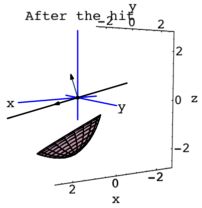




Grab, ALIGN and animate both plots.

See both from the viewpoint of perpframe[2]:

```
Show[before, ViewPoint → 6 perpframe[2]];
Show[after, ViewPoint → 6 perpframe[2]];
```





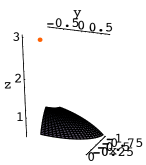

Grab, ALIGN and animate both plots.

Just a little groovy.

### □T.1.d.i) Making a matrix for bouncing a light ray off a 3D surface

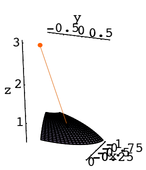
Here are a surface in 3D and a point above the surface all plotted in true scale:

```
Clear[x, y, z, r, s, t];
{x[s_, t_], y[s_, t_], z[s_, t_]} =
  {-1, -1, 0} + {Sin[s] Cos[t], 2 Sin[s] Sin[t], Cos[s]};

{{slow, shigh}, {tlow, thigh}} = {{0.2, 1.0}, {0.2, 1.5}};

point = {0, -0.5, 3};
Clear[surfaceplotter];
surfaceplotter[s_, t_] = {x[s, t], y[s, t], z[s, t]};

surfaceplot = ParametricPlot3D[Evaluate[surfaceplotter[s, t]],
   {s, slow, shigh}, {t, tlow, thigh}, Boxed → False,
   BoxRatios → Automatic, ViewPoint → CMView,
   AxesLabel → {"x", "y", "z"}, DisplayFunction → Identity];
pointplot = Graphics3D[{CadmiumOrange,
    PointSize[0.05], Point[point]}];

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



A light ray emanates from the point and hits the surface:

```
Clear[ray];
ray[s_, t_] = surfaceplotter[s, t] - point;

sjump = (shigh - slow)/2;
tjump = (thigh - tlow)/2;

rayplots =
  Table[Arrow[ray[s, t], Tail → point, VectorColor → MarsYellow,
    HeadSize → 0.15], {s, slow + sjump, shigh - sjump, sjump},
   {t, tlow + tjump, thigh - tjump, tjump}];

setup = Show[surfaceplot, pointplot, rayplots, Boxed → False,
   BoxRatios → Automatic, ViewPoint → CMView, PlotRange → All,
   AxesLabel → {"x", "y", "z"}, DisplayFunction → $DisplayFunction];
```



Your job is to plot the reflected light ray.
Do it.

**□Answer:**

At a point {x[s, t], y[s, t], z[s, t]} on the surface, the vectors

tan1[s, t] = {D[x[s, t], s], D[y[s, t], s], D[z[s, t], s]};

and

tan2[s, t] = {D[x[s, t], t], D[y[s, t], t], D[z[s, t], t]}

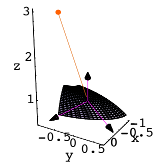
are tangent to the surface.

At each point

{x[s, t], y[s, t], z[s, t]}

on the surface, use these two tangential vectors to make a 3D perpendicular frame including two tangential vectors and one normal vector and plot:

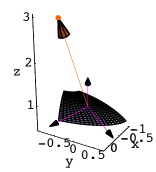
```
Clear[tan1, tan2];
tan1[s_, t_] = {∂s x[s, t], ∂s y[s, t], ∂s z[s, t]};
tan2[s_, t_] = {∂t x[s, t], ∂t y[s, t], ∂t z[s, t]};

Clear[alignerframe, s, t, cross];
alignerframe[1, s_, t_] := tan1[s, t] / Sqrt[tan1[s, t].tan1[s, t]];
cross[s_, t_] := tan1[s, t] × tan2[s, t];
alignerframe[3, s_, t_] := cross[s, t] / Sqrt[cross[s, t].cross[s, t]];
alignerframe[2, s_, t_] :=
 alignerframe[1, s, t] × alignerframe[3, s, t];

frameplots =
  Table[Arrow[alignerframe[k, s, t], Tail → {x[s, t], y[s, t], z[s, t]}
    VectorColor -> Magenta], {s, slow + sjump, shigh - sjump, sjump},
   {t, tlow + tjump, thigh - tjump, tjump}, {k, 1, 3}];

step1 = Show[setup, rayplots,
   frameplots, DisplayFunction → $DisplayFunction];
```



The two tangential vectors are alignerframe[1,s,t] and alignerframe[2,s,t].
The normal vector is alignerframe[3,s,t].

Now reverse the light vector:

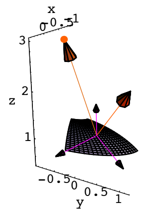
```
reversedrayplots =
  Table[Arrow[-ray[s, t], Tail → {x[s, t], y[s, t], z[s, t]},
    VectorColor → MarsYellow], {s, slow + sjump, shigh - sjump, sjump},
   {t, tlow + tjump, thigh - tjump, tjump}];

step2 = Show[surfaceplot, pointplot,
   reversedrayplots, frameplots, Boxed → False,
   BoxRatios → Automatic, ViewPoint → CMView, PlotRange → All,
   AxesLabel → {"x", "y", "z"}, DisplayFunction → $DisplayFunction];
```



alignerframe[1,s,t] and alignerframe[2,s,t] are tangent to the surface.

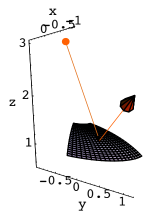
These are not the reflected rays. To get the reflected rays, just use these choices of aligner, stretcher and hanger and plot:

```
Clear[aligner, hanger, hangerframe, reflectormatrix];
aligner[s_, t_] = {alignerframe[1, s, t],
  alignerframe[2, s, t], alignerframe[3, s, t]};

{xstretch, ystretch, zstretch} = {1, 1, 1};
stretcher = DiagonalMatrix[{xstretch, ystretch, zstretch}];

{hangerframe[1, s_, t_],
  hangerframe[2, s_, t_], hangerframe[3, s_, t_]} =
 {-alignerframe[1, s, t], -alignerframe[2, s, t],
  alignerframe[3, s, t]};

hanger[s_, t_] = Transpose[ {hangerframe[1, s, t],
    hangerframe[2, s, t], hangerframe[3, s, t]}];

reflectormatrix[s_, t_] = hanger[s, t].(stretcher.aligner[s, t]);

reflectedrayplots =
Table[Arrow[reflectormatrix[s, t].(-ray[s, t]),
   Tail -> {x[s, t], y[s, t], z[s, t]}, VectorColor -> CadmiumOrange],
{s, slow + sjump , shigh - sjump, sjump},
        {t, tlow + tjump, thigh - tjump, tjump}];

step3 = Show[surfaceplot, pointplot,
   reversedrayplots, frameplots, reversedrayplots,
   reflectedrayplots, frameplots, PlotRange -> All,
     DisplayFunction -> $DisplayFunction];
```



Clean it up with a plot showing both the incoming rays and the reflected rays:

```
finalproduct = Show[setup, rayplots,
   reflectedrayplots, DisplayFunction → $DisplayFunction];
```

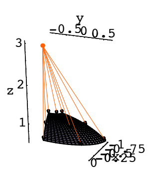


Done.

### □ T.1.d.ii) Making matrices for bouncing lots of light rays off a 3D surface

Here's the same setup as in part i), but this time there are many light rays hitting the surface:

```
Clear[ray];
ray[s_, t_] = surfaceplotter[s, t] - point;

sjump = (shigh - slow)/2;
tjump = (thigh - tlow)/2;

rayplots = Table[Arrow[ray[s, t], Tail → point,
    VectorColor → CadmiumOrange, HeadSize → 0.1],
   {s, slow, shigh, sjump}, {t, tlow, thigh, tjump}];

setup = Show[surfaceplot, pointplot, rayplots, Boxed → False,
   BoxRatios → Automatic, ViewPoint → CMView, PlotRange → All,
   AxesLabel → {"x", "y", "z"}, DisplayFunction → $DisplayFunction];
```



Your job is to plot the reflected light rays.
Do it.

**□ Answer:**

At a point {x[s, t], y[s, t], z[s, t]} on the surface, the vectors

tan1[s, t] = {D[x[s, t], s], D[y[s, t], s], D[z[s, t], s]};

and

tan2[s, t] = {D[x[s, t], t], D[y[s, t], t], D[z[s, t], t]}

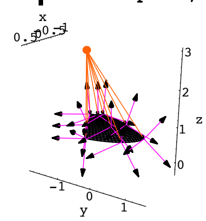
are tangent to the surface.

At each point

{x[s, t], y[s, t], z[s, t]}

on the surface, use these two tangential vectors to make a 3D perpendicular frame including two tangential vectors and one normal vector and plot:

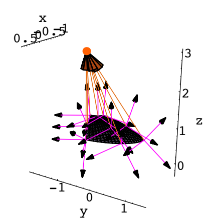
```
Clear[tan1, tan2];
tan1[s_, t_] = {∂s x[s, t], ∂s y[s, t], ∂s z[s, t]};
tan2[s_, t_] = {∂t x[s, t], ∂t y[s, t], ∂t z[s, t]};

Clear[alignerframe, s, t, cross];
alignerframe[1, s_, t_] := tan1[s, t] / Sqrt[tan1[s, t].tan1[s, t]];
cross[s_, t_] := tan1[s, t] × tan2[s, t];
alignerframe[3, s_, t_] := cross[s, t] / Sqrt[cross[s, t].cross[s, t]];
alignerframe[2, s_, t_] :=
 alignerframe[1, s, t] × alignerframe[3, s, t];

frameplots = Table[Arrow[alignerframe[k, s, t],
     Tail → {x[s, t], y[s, t], z[s, t]}, VectorColor -> Magenta],
    {s, slow, shigh, sjump}, {t, tlow, thigh, tjump}, {k, 1, 3}];

step1 = Show[setup, rayplots,
    frameplots, DisplayFunction → $DisplayFunction];
```



At each point,the two tangential vectors are alignerframe[1,s,t] and alignerframe[2,s,t].
The normal vector is alignerframe[3,s,t].

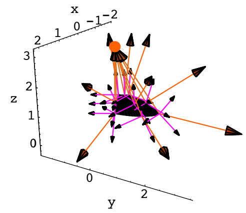
Now reverse the light vectors:

```
reversedrayplots = Table[Arrow[-ray[s, t],
     Tail → {x[s, t], y[s, t], z[s, t]}, VectorColor → MarsYellow],
    {s, slow, shigh, sjump}, {t, tlow, thigh, tjump}];

step2 = Show[surfaceplot, pointplot,
    reversedrayplots, frameplots, Boxed → False,
    BoxRatios → Automatic, ViewPoint → CMView, PlotRange → All,
    AxesLabel → {"x", "y", "z"}, DisplayFunction → $DisplayFunction];
```



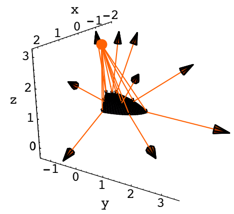
These are not the reflected rays. To get the reflected rays, just use these choices of aligner, stretcher and hanger and plot:

```
Clear[aligner, hanger, hangerframe, reflectormatrix];
aligner[s_, t_] = {alignerframe[1, s, t],
  alignerframe[2, s, t], alignerframe[3, s, t]};

{xstretch, ystretch, zstretch} = {1, 1, 1};
stretcher = DiagonalMatrix[{xstretch, ystretch, zstretch}];

{hangerframe[1, s_, t_],
  hangerframe[2, s_, t_], hangerframe[3, s_, t_]} =
 {-alignerframe[1, s, t], -alignerframe[2, s, t],
  alignerframe[3, s, t]};

hanger[s_, t_] = Transpose[ {hangerframe[1, s, t],
    hangerframe[2, s, t], hangerframe[3, s, t]}];

reflectormatrix[s_, t_] = hanger[s, t].stretcher.aligner[s, t];

reflectedrayplots =
Table[Arrow[reflectormatrix[s, t].(-ray[s, t]),
    Tail -> {x[s, t], y[s, t], z[s, t]}, VectorColor -> CadmiumOrange],
{s, slow , shigh, sjump},
        {t, tlow, thigh, tjump}];

step3 = Show[surfaceplot, pointplot,
   reversedrayplots, frameplots, reversedrayplots,
   reflectedrayplots, frameplots, PlotRange -> All,
     DisplayFunction -> $DisplayFunction];
```



Clean it up with a plot showing both the incoming rays and the reflected rays:

```
finalproduct = Show[setup, rayplots,
  reflectedrayplots, DisplayFunction → $DisplayFunction];
```



Done.

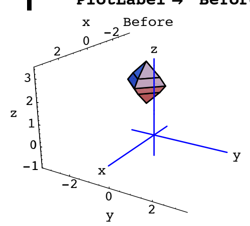
---

## T.2) 3D rotations: Rotating about lines in 3D

### □ T.2.a.i) Making matrices whose hits rotate about the z-axis

Here's a double pyramid:

```
ranger = 3.5;
Clear[x, y, z, s, t];
{x[s_, t_], y[s_, t_], z[s_, t_]} =
  {-1, -1, 2} + {Sin[s] Cos[t], Sin[s] Sin[t], Cos[s]^3};

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

ranger = 3.5;
Clear[hitplotter, matrix];
hitplotter[matrix3D_] :=
  ParametricPlot3D[matrix3D.{x[s, t], y[s, t], z[s, t]},
   {s, slow, shigh}, {t, tlow, thigh}, PlotPoints → {5, 5},
   PlotRange → {{-ranger, ranger}, {-ranger, ranger}, {-1, ranger}},
   BoxRatios → Automatic, Axes → True, AxesLabel → {"x", "y", "z"},
   Boxed → False, ViewPoint → CMView, DisplayFunction → Identity];

surfacebefore = Show[hitplotter[IdentityMatrix[3]], Axes3D[ranger],
  PlotLabel → "Before", DisplayFunction → $DisplayFunction];
```




That line is pointing out the z-axis.

Use hits with a 3D rotation matrix to rotate this surface about the z-axis.

**□ Answer:**

In 2D, to rotate by s counterclockwise radians, you hit with:

```
Clear[rotater2D, s];
rotater2D[s_] =
 Transpose[{{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}}];

MatrixForm[rotater2D[s]]
```

$$\begin{pmatrix} \text{Cos}[s] & -\text{Sin}[s] \\ \text{Sin}[s] & \text{Cos}[s] \end{pmatrix}$$

You can carry this over a 3D matrix whose hits rotate about the z-axis this way:

```
Clear[zrotater3D, s];
zrotater3D[s_] = Transpose[
  {{Cos[s], Sin[s], 0}, {Cos[s + π/2], Sin[s + π/2], 0}, {0, 0, 1}}];

MatrixForm[zrotater3D[s]]
```

$$\begin{pmatrix} \text{Cos}[s] & -\text{Sin}[s] & 0 \\ \text{Sin}[s] & \text{Cos}[s] & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

To see why this works, look at:

```
rotater2D[s].{x, y}
```

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

And look at:

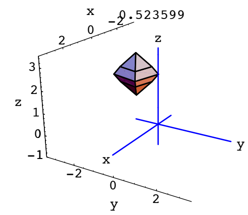
```
zrotater3D[s].{x, y, z}
```

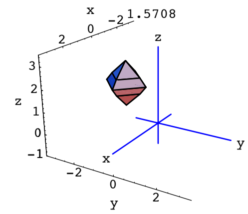
{x Cos[s] - y Sin[s], y Cos[s] + x Sin[s], z}

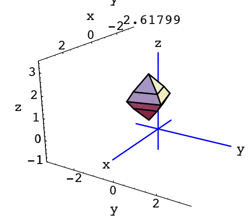
A hit with rotater3D[s] rotates the x's and y's just the way a hit with rotater2D[s] does and rotater3D[s] does not disturb the z's at all.

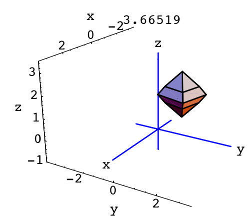
Watch hits with rotater3D[s] whirl the double pyramid around the z - axis:

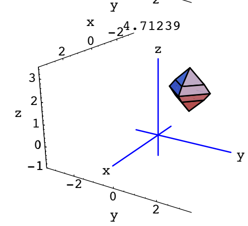
```
jump = 2 π/6;
Table[Show[hitplotter[zrotater3D[s]], Axes3D[ranger],
  PlotLabel → N[s], DisplayFunction → $DisplayFunction],
 {s, jump/2, 2 π - jump/2, jump}]
```

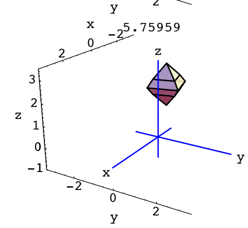


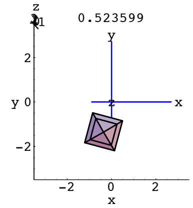



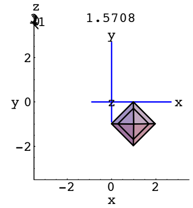

{- Graphics3D -, - Graphics3D -, - Graphics3D -,
 - Graphics3D -, - Graphics3D -, - Graphics3D -}

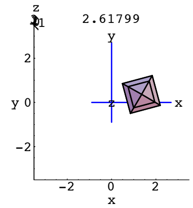
Grab, align and animate at various speeds.

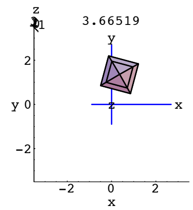
See the same thing from a view point way out the positive z axis.

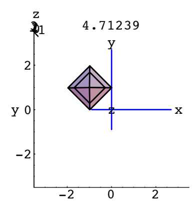
```
jump = 2 π/6;
Table[Show[hitplotter[zrotater3D[s]],
  Axes3D[0.8 ranger], PlotLabel → N[s], ViewPoint → 12 {0, 0, 1},
  DisplayFunction → $DisplayFunction], {s, jump/2, 2 π - jump/2, jump}]
```

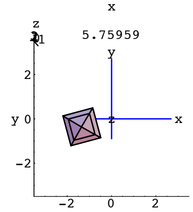


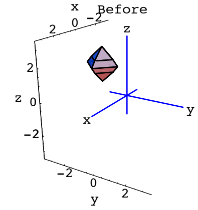

```
{- Graphics3D -, - Graphics3D -, - Graphics3D -,
 - Graphics3D -, - Graphics3D -, - Graphics3D -}
```

Just as it was made to do.

### □ T.2.a.ii) Making matrices whose hits rotate about the x-axis

Here's another double pyramid :

```
ranger = 4;
Clear[x, y, z, s, t];
{x[s_, t_], y[s_, t_], z[s_, t_]} =
  {1, -1, 2} + {Sin[s] Cos[t], Sin[s] Sin[t], Cos[s]^3};

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

ranger = 3.5;
Clear[hitplotter, matrix];
hitplotter[matrix3D_] :=
  ParametricPlot3D[matrix3D.{x[s, t], y[s, t], z[s, t]},
   {s, slow, shigh}, {t, tlow, thigh}, PlotPoints → {5, 5}, PlotRange →
    {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
   BoxRatios → Automatic, Axes → True, AxesLabel → {"x", "y", "z"},
   Boxed → False, ViewPoint → CMView, DisplayFunction → Identity];

surfacebefore = Show[hitplotter[IdentityMatrix[3]], Axes3D[ranger],
   PlotLabel → "Before", DisplayFunction → $DisplayFunction];
```




Use hits with a 3D rotation matrix to rotate this surface about the plotted x-axis.

**□ Answer:**

In 2D, to rotate by s counterclockwise radians, you hit with:

```
Clear[rotater2D, s];
rotater2D[s_] =
 Transpose[{{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}}];

MatrixForm[rotater2D[s]]
```

$$\begin{pmatrix} \mathrm{Cos}[s] & -\mathrm{Sin}[s] \\ \mathrm{Sin}[s] & \mathrm{Cos}[s] \end{pmatrix}$$

You can carry this over a 3D matrix whose hits rotate about the x-axis this way:

```
Clear[xrotater3D, s];
xrotater3D[s_] = Transpose[
  {{1, 0, 0}, {0, Cos[s], Sin[s]}, {0, Cos[s + π/2], Sin[s + π/2]}}];

MatrixForm[xrotater3D[s]]
```

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & \mathrm{Cos}[s] & -\mathrm{Sin}[s] \\ 0 & \mathrm{Sin}[s] & \mathrm{Cos}[s] \end{pmatrix}$$

To see why this works, look at:

```
rotater2D[s].{y, z}
```

```
{y Cos[s] - z Sin[s], z Cos[s] + y Sin[s]}
```

And look at:

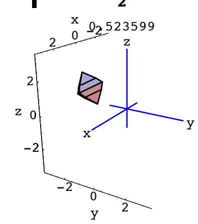
```
xrotater3D[s].{x, y, z}
```

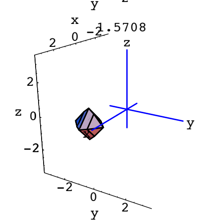
```
{x, y Cos[s] - z Sin[s], z Cos[s] + y Sin[s]}
```

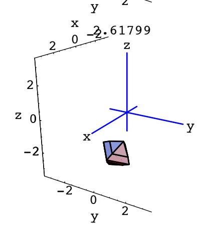
A hit with xrotater3D[s] rotates the y's and z's just the way a hit with rotater2D[s] does and xrotater3D[s] does not disturb the x's at all.

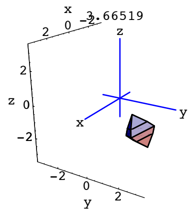
Watch hits with xrotater3D[s] whirl the double pyramid around the x - axis:

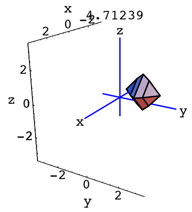
```
jump = 2 π/6;
Table[Show[hitplotter[xrotater3D[s]], Axes3D[ranger],
  PlotLabel → N[s], DisplayFunction → $DisplayFunction],
 {s, jump/2, 2 π - jump/2, jump}]
```

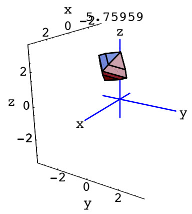






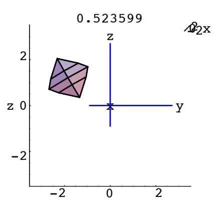


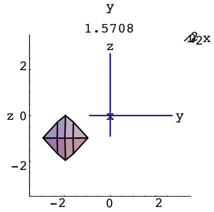

```
{- Graphics3D -, - Graphics3D -, - Graphics3D -,
 - Graphics3D -, - Graphics3D -, - Graphics3D -}
```

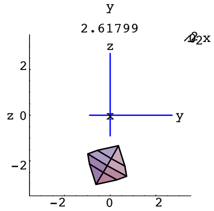
Grab, align and animate at various speeds.

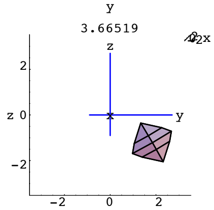
See the same thing from a view point way out the positive x axis.

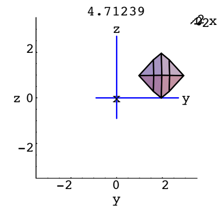
```
jump = 2π/6;
Table[Show[hitplotter[xrotater3D[s]],
  Axes3D[0.8 ranger], PlotLabel → N[s], ViewPoint → 12 {1, 0, 0},
  DisplayFunction → $DisplayFunction], {s, jump/2, 2π - jump/2, jump}]
```

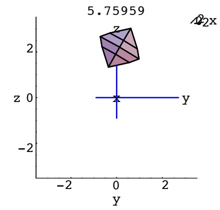













```
{- Graphics3D -, - Graphics3D -, - Graphics3D -,
 - Graphics3D -, - Graphics3D -, - Graphics3D -}
```

Grab, align and animate at various speeds.

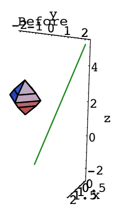
Just as it was made to do.

Whirlaway.

### □ T.3.a.iii) Rotating about 3D lines through {0,0,0}

Here's the same setup as in part i) shown with a line through {0,0,0}:

```
linedirectionvector = {0.1, 1.1, 2.6};
newline = Graphics3D[{ForestGreen,
    Line[{-linedirectionvector, 2 linedirectionvector}]}];

Show[hitplotter[IdentityMatrix[3]], newline, PlotRange -> All,
  PlotLabel → "Before", DisplayFunction → $DisplayFunction];
```

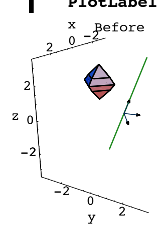



Use matrix hits to rotate this surface about the plotted line.

#### □ Answer:

Make a custom perpendicular frame, taking care that perpframe[3] is a unit vector pointing in the direction of the plotted line:

```
perpframe[3] = linedirectionvector / Norm[linedirectionvector];

throwawayvector = {Random[Real, {-1, 1}],
   Random[Real, {-1, 1}], Random[Real, {-1, 1}]};

planevector = throwawayvector × perpframe[3];
perpframe[1] = planevector / Norm[planevector];
perpframe[2] = perpframe[3] × perpframe[1];

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

Show[hitplotter[IdentityMatrix[3]], frameplot, newline,
  PlotLabel → "Before", DisplayFunction → $DisplayFunction];
```




Rerun until you see all three unit vectors in the custom frame.

Activate:

```
Clear[zrotater3D, s];
zrotater3D[s_] = Transpose[
   {{Cos[s], Sin[s], 0}, {Cos[s + π/2], Sin[s + π/2], 0}, {0, 0, 1}}];

MatrixForm[zrotater3D[s]]
```

$$\begin{pmatrix} \text{Cos}[s] & -\text{Sin}[s] & 0 \\ \text{Sin}[s] & \text{Cos}[s] & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Use the custom frame for the aligner frame and for the hanger frame:

```
Clear[alignerframe, hangerframe];
{alignerframe[1], alignerframe[2], alignerframe[3]} =
  {perpframe[1], perpframe[2], perpframe[3]};
aligner = {alignerframe[1], alignerframe[2], alignerframe[3]};

MatrixForm[aligner]
```

$$\begin{pmatrix} 0.669932 & 0.674151 & -0.310985 \\ -0.741578 & 0.627607 & -0.237004 \\ 0.0353996 & 0.389396 & 0.92039 \end{pmatrix}$$

```
{hangerframe[1], hangerframe[2], hangerframe[3]} =
  {perpframe[1], perpframe[2], perpframe[3]};
hanger = Transpose[{hangerframe[1],
    hangerframe[2], hangerframe[3]}];

MatrixForm[hanger]
```

$$\begin{pmatrix} 0.669932 & -0.741578 & 0.0353996 \\ 0.674151 & 0.627607 & 0.389396 \\ -0.310985 & -0.237004 & 0.92039 \end{pmatrix}$$

The matrix you want is:

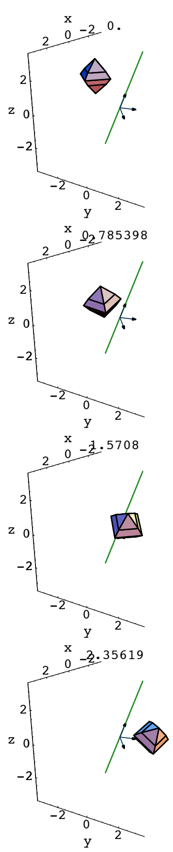
```
Clear[linerotator];
linerotater[s_] = Expand[hanger.zrotater3D[s].aligner]
```

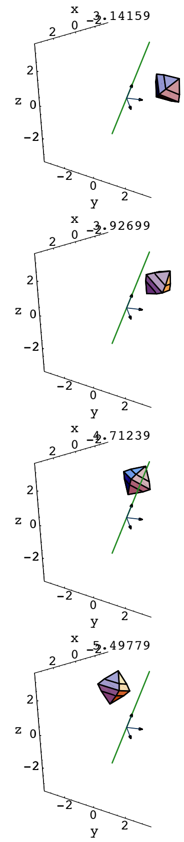
```
{{0.00125313 + 0.998747 Cos[s],
  0.0137845 - 0.0137845 Cos[s] - 0.92039 Sin[s],
  0.0325815 - 0.0325815 Cos[s] + 0.389396 Sin[s]},
 {0.0137845 - 0.0137845 Cos[s] + 0.92039 Sin[s],
  0.151629 + 0.848371 Cos[s],
  0.358396 - 0.358396 Cos[s] - 0.0353996 Sin[s]},
 {0.0325815 - 0.0325815 Cos[s] - 0.389396 Sin[s],
  0.358396 - 0.358396 Cos[s] + 0.0353996 Sin[s],
  0.847118 + 0.152882 Cos[s]}}
```

Ugly, but effective.

Watch this ugly rotation matrix do its work:

```
jump = π/4;
Table[Show[hitplotter[linerotater[s]],
  frameplot, newline, PlotLabel → N[s],
  DisplayFunction → $DisplayFunction], {s, 0, 2 π - jump/2, jump}]
```



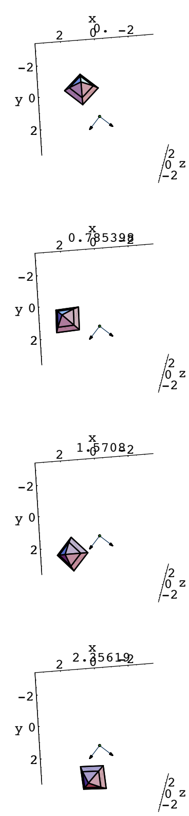



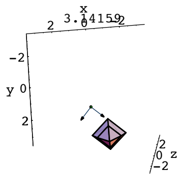

```
{- Graphics3D -, - Graphics3D -, - Graphics3D -, - Graphics3D -,
 - Graphics3D -, - Graphics3D -, - Graphics3D -, - Graphics3D -}
```

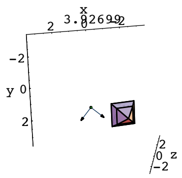
Grab, align and animate.

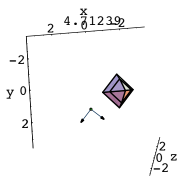
See it from the viewpoint of perpframe[3]:

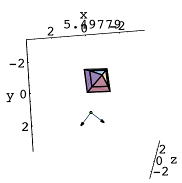
```
Table[Show[hitplotter[linerotater[s]], frameplot,
  newline, PlotLabel → N[s], ViewPoint → 6 perpframe[3],
  DisplayFunction → $DisplayFunction], {s, 0, 2 π - jump/2, jump}]
```











```
{- Graphics3D -, - Graphics3D -, - Graphics3D -, - Graphics3D -,
 - Graphics3D -, - Graphics3D -, - Graphics3D -, - Graphics3D -}
```

Grab, align and animate.

Getting dizzy.

If you want a plain language explanation of why
linerotater[s] = hanger. zrotater3D[s].aligner.
does the job, click on the right.

The hit with aligner replots the surface with

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

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

and the positive z - axis playing the former role of perpframe[3].

Then the hit with zrotater3D[s] rotates about the z-axis.

And the hit with hanger replots the rotated surface with

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

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

perpframe[3] playing the former role of the positive z-axis.

The final result is nothing more or less than a rotation about perpframe[3]

---

## T.3) Euler angles and the 3D right hand frame maker. Right and left hand perpendicular frames in 3D.

### T.3.a.i) Euler angles r, s and t and the 3D right hand frame maker

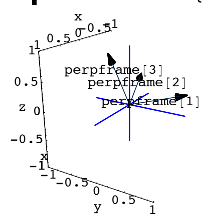
Here's that nasty formula that produces beautiful 3D right hand perpendicular frames.

```
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]}}
```

See one:

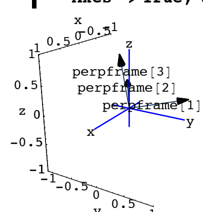
```
r = π/4;
s = π/8;
t = π/3;
Clear[perpframe];
{perpframe[1], perpframe[2], perpframe[3]} =
  {{Cos[r] Cos[t] - Cos[s] Sin[r] Sin[t],
    Cos[s] Cos[t] Sin[r] + Cos[r] Sin[t], Sin[r] Sin[s]},
   {-Cos[t] Sin[r] - Cos[r] Cos[s] Sin[t],
    Cos[r] Cos[s] Cos[t] - Sin[r] Sin[t], Cos[r] Sin[s]},
   {Sin[s] Sin[t], -Cos[t] Sin[s], Cos[s]}};
ranger = 1.0;
frameplot = Show[
   Table[Arrow[perpframe[k], Tail → {0, 0, 0}, VectorColor → 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"}];
```




See some more:

```
r = Random[Real, {-π/2, π/2}];
s = Random[Real, {-π/2, π/2}];
t = Random[Real, {-π/2, π/2}];
{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;
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[1, 0.1],
  PlotRange ->
    {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
  Boxed -> False,
  Axes -> True, ViewPoint -> CMView, AxesLabel -> {"x", "y", "z"}];
```




Rerun many times.

Explain what the parameters r, s and t mean.

**Answer:**

This is one situation in which the explanation is easier than the formula.

Dial up the matrix zrot[r] whose hits rotate everything r radians about the z-axis:

```
Clear[zrot, r];
zrot[r_] = Transpose[
   {{Cos[r], Sin[r], 0}, {Cos[r + π/2], Sin[r + π/2], 0}, {0, 0, 1}}];
MatrixForm[zrot[r]]
```

$$\begin{pmatrix} \text{Cos}[r] & -\text{Sin}[r] & 0 \\ \text{Sin}[r] & \text{Cos}[r] & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Dial up the matrix xrot[s] whose hits rotate everything s radians about the x-axis:

```
Clear[xrot, s];
xrot[s_] = Transpose[
   {{1, 0, 0}, {0, Cos[s], Sin[s]}, {0, Cos[s + π/2], Sin[s + π/2]}}];
MatrixForm[xrot[s]]
```

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & \text{Cos}[s] & -\text{Sin}[s] \\ 0 & \text{Sin}[s] & \text{Cos}[s] \end{pmatrix}$$

Now start with the usual x-y-z axes perpendicular frame

{{1, 0, 0}, {0, 1, 0}, {0, 0, 1}}

and hit it with zrot[r], thereby rotating r radians about the z- axis.

This gives a new perpendicular frame:

```
{zrot[r].{1, 0, 0},
   zrot[r].{0, 1, 0},
   zrot[r].{0, 0, 1}}
```

```
{{Cos[r], Sin[r], 0}, {-Sin[r], Cos[r], 0}, {0, 0, 1}}
```

Hit this perpendicular frame with xrot[s], thereby rotating s radians about the x- axis.

This gives a new perpendicular frame:

```
{xrot[s].zrot[r].{1, 0, 0},
   xrot[s].zrot[r].{0, 1, 0},
   xrot[s].zrot[r].{0, 0, 1}}
```

```
{{Cos[r], Cos[s] Sin[r], Sin[r] Sin[s]},
 {-Sin[r], Cos[r] Cos[s], Cos[r] Sin[s]}, {0, -Sin[s], Cos[s]}}
```

Finally hit this perpendicular frame with zrot[t], thereby rotating t radians about the z- axis (again).

This is the perpendicular frame corresponding to the Euler angles r, s and t:

```
{zrot[t].xrot[s].zrot[r].{1, 0, 0},
   zrot[t].xrot[s].zrot[r].{0, 1, 0},
   zrot[t].xrot[s].zrot[r].{0, 0, 1}}
```

```
{{0.092929 Cos[r] - 0.995673 Cos[s] Sin[r],
  0.995673 Cos[r] + 0.092929 Cos[s] Sin[r], Sin[r] Sin[s]},
 {-0.995673 Cos[r] Cos[s] - 0.092929 Sin[r],
  0.092929 Cos[r] Cos[s] - 0.995673 Sin[r], Cos[r] Sin[s]},
 {0.995673 Sin[s], -0.092929 Sin[s], Cos[s]}}
```

This is the same as the formula

{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]}}.

Now you know where this formula comes from.

And you know that the Euler angles r, s and t specify:

An initial rotation by r radians about the z-axis,

followed by a rotation by s radians about the x-axis

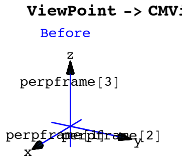
and then

followed by a rotation by t radians about the z-axis.

See it happen in stages:

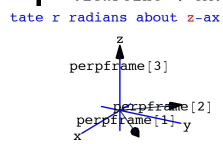
Start with r, s and t zeroed out:

```
r = 0;
s = 0;
t = 0;
{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;
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[1, 0.1],
  PlotRange ->
   {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
  Boxed -> False, PlotLabel -> "Before",
  ViewPoint -> CMView, AxesLabel -> {"x", "y", "z"}];
```



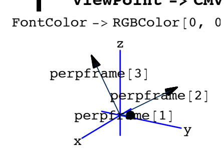

First rotate this frame r radians about the z- axis:

```
r = π/4;
s = 0;
t = 0;
{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;
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[1, 0.1],
  PlotRange ->
   {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
  Boxed -> False, PlotLabel -> "Rotate r radians about z-axis",
  ViewPoint -> CMView, AxesLabel -> {"x", "y", "z"}];
```




Second rotate the frame above by s radians about the x-axis

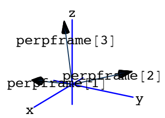
```
r = π/4;
s = π/6;
t = 0;
{perpframe[1], perpframe[2], perpframe[3]} =
  {{Cos[r] Cos[t] - Cos[s] Sin[r] Sin[t],
    Cos[s] Cos[t] Sin[r] + Cos[r] Sin[t], Sin[r] Sin[s]},
   {-Cos[t] Sin[r] - Cos[r] Cos[s] Sin[t],
    Cos[r] Cos[s] Cos[t] - Sin[r] Sin[t], Cos[r] Sin[s]},
   {Sin[s] Sin[t], -Cos[t] Sin[s], Cos[s]}};
ranger = 1;
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[1, 0.1],
  PlotRange ->
   {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
  Boxed -> False, PlotLabel -> "Then rotate\\ StyleBox[" ",
FontColor -> RGBColor[0, 0, 1]]s radians about x-axis",
 ViewPoint -> CMView, AxesLabel -> {"x", "y", "z"}];
```




Finally rotate the frame above by t radians about the z-axis

```
r = π/4;
s = π/6;
t = -π/4;
{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;
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[1, 0.1],
  PlotRange ->
   {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
  Boxed -> False, PlotLabel -> "Finally rotate\\ StyleBox[" ",
FontColor -> RGBColor[0, 0, 1]]t radians about z-axis",
  ViewPoint -> CMView, AxesLabel -> {"x", "y", "z"}];
```

.n FontColor -> RGBColor[0,




Grab all four plots and animate. Then go back and change the specifications of r, s and t rerun.

### T.3.a.ii) The 3D frame maker produces right hand perpendicular frames

A right hand 3D perpendicular frame is any frame that has the same orientation as the x-y-z coordinate frame {{1,0.0},{0,1,0},{0,0,1}}:

```
{1, 0, 0}.Cross[{0, 1, 0}, {0, 0, 1}]
```

1

So a 3D perpendicular frame
{perpframe[1], perpframe[2] , perpframe[3]}

is a right hand frame if
perpframe[1]. ( perpframe[2] × perpframe[3]) = 1.

And a hand 3D perpendicular frame
{perpframe[1], perpframe[2] , perpframe[3]}
is a left hand frame if
perpframe[1]. ( perpframe[2] × perpframe[3]) = −1.

Explain why the 3D perpendicular frame produced by:

```
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]}}
```

is a right hand frame.

**Answer:**

From the last part

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

comes from successive rotations of

{{1,0.0},{0,1,0},{0,0,1}};

so it has the same relative orientation as

{{1,0.0},{0,1,0},{0,0,1}}.

So the fact that

perpframe[1]. ( perpframe[2] × perpframe[3]) = 1

is guaranteed:

```
Simplify[perpframe[1].Cross[perpframe[2], perpframe[3]]]
```

1

### T.3.a.iii) Left hand perpendicular frames in 3D

Look at this:

```
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]}}
```

Make a new perpendicular frame by interchanging perpframe[1] and perpframe[2] and see whether the new frame is a right hand frame:

```
Clear[newperpframe];
newperpframe[1] = perpframe[2];
newperpframe[2] = perpframe[1];
newperpframe[3] = perpframe[3];
Simplify[newperpframe[1].Cross[newperpframe[2], newperpframe[3]]]
```

-1

You make the call:
Is
{newperpframe[1],newperpframe[2],{newperpframe[3]}
a left or a right hand frame in 3D?

**Answer:**

Look at newperpframe[1]. ( newperpframe[2] × newperpframe[3]) again:

```
Simplify[newperpframe[1].Cross[newperpframe[2], newperpframe[3]]]
```

-1

This signals loudly and clearly that

{newperpframe[1],newperpframe[2],{newperpframe[3]} is a left hand perpendicular frame.

---

## T.4) Det[A .B] = Det[A] Det[B]

**If A is a 3D diagonal matrix, then Det[A] = product of diagonal entries**

**Why Det[$A^{-1}$] = $\frac{1}{\text{Det}[A]}$**

**If A is a 3D hanger or aligner based on a right hand frame, then Det[A] = 1.**

**If A is a 3D hanger or aligner based on a left hand frame, then Det[A] = −1.**

**Why Det[$A^t$] = Det[A]**

### T.4.a.) Det[A.B] = Det[A] Det[B]

Here are two random 3D matrices A and B

```
A =
 ( Random[Real, {-4, 4}]  Random[Real, {-4, 4}]  Random[Real, {-4, 4}]
   Random[Real, {-4, 4}]  Random[Real, {-4, 4}]  Random[Real, {-4, 4}]
   Random[Real, {-4, 4}]  Random[Real, {-4, 4}]  Random[Real, {-4, 4}] );
B =
 ( Random[Real, {-4, 4}]  Random[Real, {-4, 4}]  Random[Real, {-4, 4}]
   Random[Real, {-4, 4}]  Random[Real, {-4, 4}]  Random[Real, {-4, 4}]
   Random[Real, {-4, 4}]  Random[Real, {-4, 4}]  Random[Real, {-4, 4}] );
MatrixForm[A]
MatrixForm[B]
```

$$\begin{pmatrix} 3.73636 & -3.03642 & 0.188843 \\ -3.04138 & -2.86992 & -1.34501 \\ 2.37052 & 3.03679 & 1.29534 \end{pmatrix}$$

$$\begin{pmatrix} -3.38532 & -1.22694 & -2.39366 \\ -1.12707 & -0.378595 & -1.83774 \\ 0.948037 & -0.2051 & -1.03671 \end{pmatrix}$$

Here are calculations of Det[A.B] and Det[A] Det[B]:

```
Det[A.B]
Det[A] Det[B]
```

```
-2.88399
-2.88399
```

All clued in matrix folks know that when you go with two 3D matrices A and B, then you can be sure that

Det[A.B] = Det[A].Det[B].

Explain this.

**Answer:**

This is a job for pure bean-counting. Doing it by hand would be a big project. But turning the supreme bean-counter - namely the computer loose on this one makes the explanation into a snap.

Enter a cleared matrix 3D A:

```
Clear[a, b, c, d, e, f, g, h, i, j]
A = {{a, b, c}, {d, f, g}, {h, i, j}};
MatrixForm[A]
```

$$\begin{pmatrix} a & b & c \\ d & f & g \\ h & i & j \end{pmatrix}$$

Apply the formula Det[A] = col[1].(col[2] × col[3]) to calculate Det[A]

```
Det[A]
-c f h + b g h + c d i - a g i - b d j + a f j
```

Enter another cleared matrix B:

```
Clear[r, s, t, u, v, w, x, y, z]
B = {{r, s, t}, {u, v, w}, {x, y, z}};
MatrixForm[B]
```

$$\begin{pmatrix} r & s & t \\ u & v & w \\ x & y & z \end{pmatrix}$$

Apply the formula Det[B] = col[1].(col[2] × col[3]) to calculate Det[B]

```
Det[B]
-t v x + s w x + t u y - r w y - s u z + r v z
```

Calculate A.B:

```
MatrixForm[A.B]
```

$$\begin{pmatrix} a r + b u + c x & a s + b v + c y & a t + b w + c z \\ d r + f u + g x & d s + f v + g y & d t + f w + g z \\ h r + i u + j x & h s + i v + j y & h t + i w + j z \end{pmatrix}$$

Apply the formula Det[A.B] = col[1].(col[2]×col[3]) to calculate Det[A.B]

```
productdet = Expand[Det[A.B]]
c f h t v x - b g h t v x - c d i t v x + a g i t v x + b d j t v x - a f j t v x -
 c f h s w x + b g h s w x + c d i s w x - a g i s w x - b d j s w x + a f j s w x -
 c f h t u y + b g h t u y + c d i t u y - a g i t u y - b d j t u y + a f j t u y +
 c f h r w y - b g h r w y - c d i r w y + a g i r w y + b d j r w y - a f j r w y +
 c f h s u z - b g h s u z - c d i s u z + a g i s u z + b d j s u z - a f j s u z -
 c f h r v z + b g h r v z + c d i r v z - a g i r v z - b d j r v z + a f j r v z
```

Now calculate Det[A] times Det[B]

```
Expand[Det[A] Det[B]]
c f h t v x - b g h t v x - c d i t v x + a g i t v x + b d j t v x - a f j t v x -
 c f h s w x + b g h s w x + c d i s w x - a g i s w x - b d j s w x + a f j s w x -
 c f h t u y + b g h t u y + c d i t u y - a g i t u y - b d j t u y + a f j t u y +
 c f h r w y - b g h r w y - c d i r w y + a g i r w y + b d j r w y - a f j r w y +
 c f h s u z - b g h s u z - c d i s u z + a g i s u z + b d j s u z - a f j s u z -
 c f h r v z + b g h r v z + c d i r v z - a g i r v z - b d j r v z + a f j r v z
```

Both give you the same thing:

```
Expand[Det[A.B]] == Expand[Det[A] Det[B]]
True
```

And because A and B could stand for any choices of 3D matrices A and B, you see conclusively that

Det[A.B] = Det[A] Det[B]

is a sure bet for any 3D matrices A and B.

### T.4.b) If A is 3D diagonal matrix , then Det[A] = product of diagonal entries

Here's a random 3D diagonal matrix

```
Clear[diagonalentry];
diagonalentry[1] = Random[Real, {-2, 2}];
diagonalentry[2] = Random[Real, {-2, 2}];
diagonalentry[3] = Random[Real, {-2, 2}];
diagonalmatrix =
  {{diagonalentry[1], 0, 0}, {0, diagonalentry[2], 0}, {0, 0, diagonalentry[3]}};
MatrixForm[diagonalmatrix]
```

$$\begin{pmatrix} 1.71832 & 0 & 0 \\ 0 & -1.78377 & 0 \\ 0 & 0 & -1.28872 \end{pmatrix}$$

Det[diagonalmatrix] is:

```
Det[diagonalmatrix]
3.95005
```

Compare:

```
diagonalentry[1] diagonalentry[2] diagonalentry[3]
3.95005
```

Explain why the same thing happens for any and all 3D diagonal matrices.

**Answer:**

The easiest way to see this is to use the formula

Det[A] = col[1].(col[2] × col[3])

from the Basics.

Applying this formula to

$$\text{Det[diagonalmatrix]} = \text{Det}\left[\begin{pmatrix} \text{diagonalentry[1]} & 0 & 0 \\ 0 & \text{diagonalentry[2]} & 0 \\ 0 & 0 & \text{diagonalentry[3]} \end{pmatrix}\right]$$

gives

```
Clear[col, diagonalentry];
col[1] = {diagonalentry[1], 0, 0};
col[2] = {0, diagonalentry[2], 0};
col[3] = {0, 0, diagonalentry[3]};
col[1].Cross[col[2], col[3]]
diagonalentry[1] diagonalentry[2] diagonalentry[3]
```

There you go.

### T.4.c) Why $\text{Det}[A^{-1}] = \frac{1}{\text{Det}[A]}$

Look at these calculations of $\text{Det}[A^{-1}]$ and $\frac{1}{\text{Det}[A]}$ for a random matrix A:

```
A = {{Random[Real, {-3, 3}], Random[Real, {-3, 3}]}, {Random[Real, {-3, 3}], Random[Real, {-3, 3}]}};
Det[Inverse[A]]
-0.251186
1/Det[A]
-0.251186
```

Apparently, $\text{Det}[A^{-1}] = \frac{1}{\text{Det}[A]}$.

Explain why this is guaranteed for any and all invertible 2D matrices A.

**Answer:**

Identity = $A^{-1}$. A

So

1 = Det[Identity] = $\text{Det}[A^{-1}]$ Det[A].

And so

$\frac{1}{\text{Det}[A]} = \text{Det}[A^{-1}]$ .

That's all there is to it.

### T.4.d.i) The determinant of a 3D hanger or aligner based on a right hand perpendicular frame is equal to +1

Explain this:

The determinant of a hanger or aligner based on a right hand perpendicular frame is equal to 1.

**Answer:**

Go with a right hand 3D perpendicular frame {perpframe[1],perpframe[2],perpframe[3]} so that

perpframe[1]. ( perpframe[2] × perpframe[3]) = 1.

The hanger matrix based on this right hand perpendicular frame is

$$\text{hanger} = \begin{pmatrix} \text{perpframe[1]} & \text{perpframe[2]} & \text{perpframe[3]} \\ \downarrow & \downarrow & \downarrow \end{pmatrix}.$$

In other words column[j] of A is perpframe[j].

So Det[hanger] = column[1]. ( column[2] × column[3]) = perpframe[1]. ( perpframe[2] × perpframe[3]) = 1.

The aligner matrix based on this right hand frame is

$$\text{aligner} = \begin{pmatrix} \text{perpframe}[1] & \rightarrow \\ \text{perpframe}[2] & \rightarrow \\ \text{perpframe}[3] & \rightarrow \end{pmatrix}.$$

Remembering that hanger and aligner are mutually inverse, you get

Identity = hanger.aligner.

So

1 = Det[Identity] = Det[hanger] Det[aligner].

And because Det[ hanger] = 1, you get

1 = 1 Det[aligner].

This tells you that Det[aligner] = 1.

### T.4.d.ii) The determinant of a 3D hanger or aligner based on a left hand perpendicular frame is equal to -1

Explain this:

The determinant of a hanger or aligner based on a left hand perpendicular frame is equal to -1.

**Answer:**

Go with a left hand 3D perpendicular frame {perpframe[1],perpframe[2],perpframe[3]} so that

perpframe[1]. ( perpframe[2] × perpframe[3]) = -1.

The hanger matrix based on this right hand perpendicular frame is

$$\text{hanger} = \begin{pmatrix} \text{perpframe}[1] & \text{perpframe}[2] & \text{perpframe}[3] \\ \downarrow & \downarrow & \downarrow \end{pmatrix}.$$

In other words column[j] of A is perpframe[j].

So Det[hanger] = column[1]. ( column[2] × column[3]) = perpframe[1]. ( perpframe[2] × perpframe[3]) = -1.

The aligner matrix based on this right hand frame is

$$\text{aligner} = \begin{pmatrix} \text{perpframe}[1] & \rightarrow \\ \text{perpframe}[2] & \rightarrow \\ \text{perpframe}[3] & \rightarrow \end{pmatrix}.$$

Remembering that hanger and aligner are mutually inverse, you get

Identity = hanger.aligner.

So

1 = Det[Identity] = Det[hanger] Det[aligner].

And because Det[ hanger] = -1, you get

1 = -1 Det[aligner].

This tells you that Det[aligner] = -1.

### T.3.e) Why Det[$A^t$] = Det[A]

Look at these calculations of Det[A] and Det[$A^t$] for random 3D matrices A:

```
A =
 (Random[Real, {-5, 5}]  Random[Real, {-5, 5}]  Random[Real, {-5, 5}]
  Random[Real, {-5, 5}]  Random[Real, {-5, 5}]  Random[Real, {-5, 5}]
  Random[Real, {-5, 5}]  Random[Real, {-5, 5}]  Random[Real, {-5, 5}]
 );
Det[A]
Det[Transpose[A]]
```

-21.8972

-21.8972

Rerun many times.

This is strong evidence that when you go with any 3D matrix A , then both A and $A^t$ have the same determinant.

Explain why this is guaranteed.

**Answer:**

This is the same explanation used in 2D in the last lesson.

Go with any 3D matrix

A = hanger.stretcher.aligner.

This gives

$A^t$ = $\text{aligner}^t$.stretcher.$\text{hanger}^t$.

So

Det[A] = Det[hanger] Det[stretcher] Det[aligner]

and

Det[$A^t$] = Det[$\text{aligner}^t$] Det[stretcher] Det[$\text{hanger}^t$].

But Det[$\text{aligner}^t$] = Det[aligner] and Det[hanger] = Det[$\text{hanger}^t$]

Reasons: If the aligner frame is a right hand frame, then $\text{aligner}^t$ is a hanger based on the same right hand frame.
So Det[aligner] = Det[$\text{aligner}^t$] = 1.
If the aligner frame is a left hand frame, then $\text{aligner}^t$ is a hanger based on the same left hand frame.
So Det[aligner] = Det[$\text{aligner}^t$] = -1.
If the hanger frame is a right hand frame, then $\text{hanger}^t$ is a aligner based on the same right hand frame.
So Det[hanger] = Det[$\text{hanger}^t$] = 1.
If the hanger frame is a left hand frame, then $\text{hanger}^t$ is a aligner based on the same left hand frame.
So Det[hanger] = Det[$\text{hanger}^t$] = -1.

The upshot: Det[A] and Det[$A^t$] are both the product of the same three numbers.

This makes them equal.

---

## T.5) Hits with 3D matrices with positive determinants preserve orientation. Hits with 3D matrices with negative determinants reverse orientation

### T.3.a.i) Saying that A is a 3D matrix with a positive determinant is the same as saying that hits with A preserve orientation in the sense that vector[1].(vector[2] × vector[3]) and (A.vector[1]).((A.vector[2]) × (A.vector[3]) have the same sign

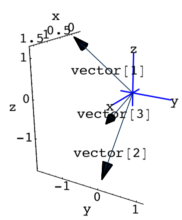
Here is a 3D matrix A with a positive determinant

```
A = (-1.2  -0.7  -1.7
     -1.8  -1.1  -0.3
     -1.4   0.6  -1.4);
Det[A]
```

3.86

Here are three vectors in 3D:

```
Clear[vector];
vector[1] = {0, -1.7, 1.2};
vector[2] = {1.3, -0.1, -1.8};
vector[3] = {1.6, 0.2, -0.2};
Show[Table[
  Arrow[vector[k],
    Tail -> {0, 0, 0}, VectorColor -> Indigo], {k, 1, 3}],
  Graphics3D[Text["vector[1]", 0.4 vector[1]]],
  Graphics3D[Text["vector[2]", 0.7 vector[2]]],
  Graphics3D[Text["vector[3]", 0.7 vector[3]]],
  Axes3D[1, 0.1],
  PlotRange -> All,
  Boxed -> False,
  Axes -> True, ViewPoint -> CMView, AxesLabel -> {"x", "y", "z"}];
```




Check the orientation of {vector[1],vector[2],vector[3]} by calculating vector[1]. (vector[2]×vector[3]):

```
vector[1].Cross[vector[2], vector[3]]
```

4.958

{vector[1],vector[2],vector[3]} are positively oriented.

Now check the orientation of {A.vector[1],A.vector[2],A.vector[3]} by calculating (A.vector[1]). ((A.vector[2])×(A.vector[3]))::

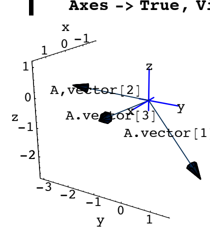
```
(A.vector[1]).Cross[A.vector[2], A.vector[3]]
```

19.1379

{A.vector[1],A.vector[2],A.vector[3]} are also positively oriented.

See them:

```
Show[Table[
  Arrow[A.vector[k], Tail -> {0, 0, 0}, VectorColor -> Indigo],
  {k, 1, 3}],
  Graphics3D[Text["A.vector[1]", 0.4 A.vector[1]]],
  Graphics3D[Text["A,vector[2]", 0.7 A.vector[2]]],
  Graphics3D[Text["A.vector[3]", 0.7 A. vector[3]]],
  Axes3D[1, 0.1],
  PlotRange -> All,
  Boxed -> False,
  Axes -> True, ViewPoint -> CMView, AxesLabel -> {"x", "y", "z"}];
```




Explain this:

Saying that A is a 3D matrix with a positive determinant is the same as saying that hits

with A preserve orientation in the sense that
vector[1].(vector[2] × vector[3]) and (A.vector[1]).((A.vector[2]) × (A.vector[3])
have the same sign (positive or negative) no matter what choices you make of vector[1],vector[2] and vector[3].

**Answer:**

Make a new matrix

$$B = \begin{pmatrix} \text{vector}[1] & \text{vector}[2] & \text{vector}[3] \\ \downarrow & \downarrow & \downarrow \end{pmatrix}.$$

vector[j] is in column[j] of B

The product A.B $= \begin{pmatrix} \text{A.vector}[1] & \text{A.vector}[2] & \text{A.vector}[3] \\ \downarrow & \downarrow & \downarrow \end{pmatrix}.$

On one hand,

Det[A.B] = col[1].(col[2] × col[3]) = A.vector[1].(A.vector[2] × A.vector[3]).

On the other hand,

Det[A.B] = Det[A] Det[B] = **Det[A]** (vector[1].(vector[2] × vector[3])).

So

A.vector[1].(A.vector[2] × A.vector[3]) = **Det[A]** (vector[1].(vector[2] × vector[3])) .

So saying that hits with A preserve orientation in the sense that

vector[1].(vector[2] × vector[3]) and (A.vector[1]).((A.vector[2]) × (A.vector[3]) have the same sign

is the same as saying that

**Det[A] > 0**.

**T.3.a.ii) Saying that A is a 3D matrix with a negative determinant is the same as saying that hits with A reverse orientation in the sense that vector[1].(vector[2] × vector[3]) and (A.vector[1]).((A.vector[2]) × (A.vector[3]) have the opposite signs**

Explain this:
Saying that A is a 3D matrix with a negative determinant is the same as saying that hits with A reverse orientation in the sense that
vector[1].(vector[2] × vector[3]) and (A.vector[1]).((A.vector[2]) × (A.vector[3])
have opposite signs no matter what choices you make of vector[1],vector[2] and vector[3].

**Answer:**

Make a new matrix

$$B = \begin{pmatrix} \text{vector}[1] & \text{vector}[2] & \text{vector}[3] \\ \downarrow & \downarrow & \downarrow \end{pmatrix}.$$

vector[j] is in column[j] of B

The product A.B $= \begin{pmatrix} \text{A.vector}[1] & \text{A.vector}[2] & \text{A.vector}[3] \\ \downarrow & \downarrow & \downarrow \end{pmatrix}.$

On one hand,

Det[A.B] = col[1].(col[2] × col[3]) = A.vector[1].(A.vector[2] × A.vector[3]).

On the other hand,

Det[A.B] = Det[A] Det[B] = **Det[A]** (vector[1].(vector[2] × vector[3])).

So

A.vector[1].(A.vector[2] × A.vector[3]) = **Det[A]** (vector[1].(vector[2] × vector[3])) .

So saying that hits with A reverse orientation in the sense that

vector[1].(vector[2] × vector[3]) and (A.vector[1]).((A.vector[2]) × (A.vector[3]) have the opposites signs

is the same as saying that

**Det[A] < 0**.

## T.6) Matrices that hit on 2D and hang in 3D. Matrices that hit on 3D and hang in 2D

### T.6.a.i) Matrices that hit on 2D and hang in 3D

Here's a new kind of matrix:

```
A = ( 2.    -0.4
      1.3    1.8
     -0.9    1.2 );
MatrixForm[A]
```

$$\begin{pmatrix} 2. & -0.4 \\ 1.3 & 1.8 \\ -0.9 & 1.2 \end{pmatrix}$$

When you hit this matrix on a 2D Point {x,y}, you get a 3D point:

```
Clear[x, y];
A.{x, y}
```

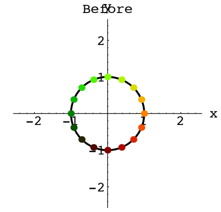
{2. x - 0.4 y, 1.3 x + 1.8 y, -0.9 x + 1.2 y}

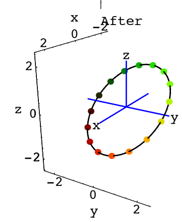
Note the three slots in A.{x,y}.

Watch this matrix hit the 2D unit circle and hang the unit circle in 3D:

```
Clear[x, y, t, hitplotter,
  hitpointplotter, pointcolor, actionarrows, matrix2D];
{tlow, thigh} = {0, 2 π};
ranger = Max[1.2, Max[SingularValues[A][[2]]]];

{x[t_], y[t_]} = {Cos[t], Sin[t]};

pointcolor[t_] = RGBColor[0.5 (Cos[t] + 1), 0.5 (Sin[t] + 1), 0];
jump = (thigh - tlow)/16;

twoDcircleplot = ParametricPlot[{x[t], y[t]},
   {t, tlow, thigh}, PlotStyle → {{Thickness[0.01]}},
   PlotRange → {{-ranger, ranger}, {-ranger, ranger}},
   AxesLabel → {"x", "y"}, DisplayFunction → Identity];
twoDpointplot = Table[Graphics[{pointcolor[t], PointSize[0.035],
     Point[{x[t], y[t]}]}], {t, tlow, thigh - jump, jump}];

before = Show[twoDcircleplot, twoDpointplot,
   PlotLabel → "Before", DisplayFunction → $DisplayFunction];
```

```
threeDhitplot = ParametricPlot3D[A.{x[t], y[t]},
   {t, tlow, thigh}, PlotRange →
    {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
   AxesLabel → {"x", "y", "z"}, DisplayFunction → Identity];
threeDhitpointplot = Table[Graphics3D[
    {pointcolor[t], PointSize[0.035], Point[A.{x[t], y[t]}]}],
   {t, tlow, thigh - jump, jump}];

after = Show[threeDhitplot, threeDhitpointplot,
   ThreeAxes[2], PlotLabel → "After", ViewPoint → CMView,
   Boxed → False, DisplayFunction → $DisplayFunction];
```



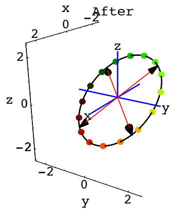

When you hit the 2D unit circle with A you get an ellipse centered on {0,0,0} sitting on a plane in 3D.
How do you frame up this ellipse?

**Answer:**

You do exactly the same thing you do for 2D matrices:

```
Clear[alignerframe, hangerframe, k];
{alignerframe[1], alignerframe[2]} = SingularValues[A][[3]];
{xstretch, ystretch} = SingularValues[A][[2]];
{hangerframe[1], hangerframe[2]} = SingularValues[A][[1]];

frameup = Show[after,
  Arrow[xstretch hangerframe[1], Tail → {0, 0, 0}, VectorColor → Red],
  Arrow[-xstretch hangerframe[1],
   Tail → {0, 0, 0}, VectorColor → Red],
```

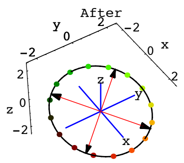
```
Arrow[ystretch hangerframe[2], Tail → {0, 0, 0},
  VectorColor → Red],
Arrow[-ystretch hangerframe[2],
  Tail → {0, 0, 0}, VectorColor → Red]];
```




There you go.

See the ellipse from a view point perpendicular to the plane determined by the ellipse:

```
planenormal = Cross[hangerframe[1], hangerframe[2]];

Show[frameup, ViewPoint → 10 planenormal, Boxed → False];
```

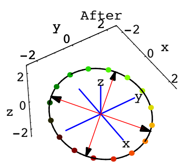



Just like 2D.

### □T.6.a.ii) Measuring the planar area enclosed by the ellipse

Take another look at the ellipse from part i)

```
Show[frameup, ViewPoint → 10 planenormal, Boxed → False];
```




Measure the planar area enclosed by this ellipse.

**□Answer:**

Remember that you got this ellipse by hitting the unit circle with A.

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

```
circlearea = π;
ellipsearea = xstretch ystretch circlearea
```

```
17.5616
```

Again just like 2D.

### □T.6.a.iii) Would the determinant help?

Could you have used the determinant of A to help in measuring the area enclosed by the ellipse?

**□Answer:**

Try it and see:

```
Det[A]
```

```
Det::matsq :
 Argument {{2., -0.4}, {1.3, 1.8}, {-0.9, 1.2}} at position 1 is not a square matrix.
```

```
Det[{{2., -0.4}, {1.3, 1.8}, {-0.9, 1.2}}]
```

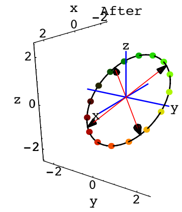
Useless.

Reason: The determinant of a matrix that hits on 2D and hangs in 3D is not defined.

### □T.6.a.iv) Another way of framing the ellipse

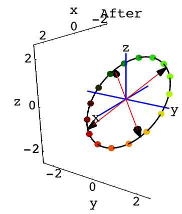
Take another look at the frame up of the ellipse in part i):

```
Show[after,
Arrow[xstretch hangerframe[1],
   Tail -> {0, 0, 0}, VectorColor -> Red],
Arrow[-xstretch hangerframe[1], Tail -> {0, 0, 0},
   VectorColor -> Red],
Arrow[ystretch hangerframe[2], Tail -> {0, 0, 0}, VectorColor -> Red],
Arrow[-ystretch hangerframe[2],
   Tail -> {0, 0, 0}, VectorColor -> Red]];
```




Now look at this:

```
Show[after,
Arrow[A.alignerframe[1], Tail -> {0, 0, 0}, VectorColor -> Red],
Arrow[-A.alignerframe[1], Tail -> {0, 0, 0}, VectorColor -> Red],
Arrow[A.alignerframe[2], Tail -> {0, 0, 0}, VectorColor -> Red],
Arrow[-A.alignerframe[2], Tail -> {0, 0, 0}, VectorColor -> Red]];
```




Exactly the same thing.
Explain why this was guaranteed to happen.

**□Answer:**

In the first plot, the plotted frame ups are

xstretch hangerframe[1]

and

ystretch hangerframe[2]

and their negatives.

In the second plot, the plotted frame ups are

A.alignerframe[1]

and

A.alignerframe[2]

and their negatives.

The plots are the same because, just as in 2D, SVD analysis guarantees that

A.alignerframe[1] = xstretch hangerframe[1]

and

A.alignerframe[2] = ystretch hangerframe[2]:

```
A.alignerframe[1] == xstretch hangerframe[1]
A.alignerframe[2] == ystretch hangerframe[2]
```

```
True
True
```

### □T.6.a.v) The rank of A is 2

Stay with the same matrix A as in the earlier parts and say why the rank of A is 2.

**□Answer:**

The rank of a matrix is the number of dimensions that matrix uses to hang its hits.

When the above matrix A is hit on all of on all of 2D, it hangs its hits on the two dimensional plane (within 3D) defined by hangerframe[1] and hangerframe[2].

That's why the rank of A is 2.

### □T.6.b.i) Matrices that hit on 3D and hang in 2D

Here's another new kind of matrix:

```
A = (2    -0.4   1.2
     1.3   1.8  -0.9);
MatrixForm[A]
```

$$\begin{pmatrix} 2 & -0.4 & 1.2 \\ 1.3 & 1.8 & -0.9 \end{pmatrix}$$

When you hit this matrix on a 3D Point {x,y,z}, you get a 2D point:

```
Clear[x, y, z];
A.{x, y, z}
```

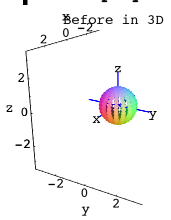
```
{2 x - 0.4 y + 1.2 z, 1.3 x + 1.8 y - 0.9 z}
```

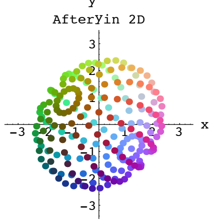
Note the two slots in A.{x,y,z}.

Watch this matrix hit points on the 3D unit sphere and hang its hits in 2D:

```
Clear[x, y, z, s, t, pointcolor];
{x[s_, t_], y[s_, t_], z[s_, t_]} =
  {Sin[s] Cos[t], Sin[s] Sin[t], Cos[s]};

{slow, shigh} = {0, π};
{tlow, thigh} = {0, 2 π};
ranger = 3.5;
sjump = (shigh - slow)/15;
```

```
tjump = (thigh - tlow)/15;

pointcolor[s_, t_] =
  RGBColor[0.5 (x[s, t] + 1), 0.5 (y[s, t] + 1), 0.5 (z[s, t] + 1)];
threeDpointplot = Table[Graphics3D[{pointcolor[s, t],
     PointSize[0.025], Point[{x[s, t], y[s, t], z[s, t]}]}],
   {s, slow, shigh - sjump, sjump}, {t, tlow, thigh - tjump, tjump}];

pointsbefore = Show[threeDpointplot,
   ThreeAxes[2], PlotLabel → "Before in 3D", PlotRange →
    {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
   Axes → True, Boxed → False, AxesLabel → {"x", "y", "z"},
   ViewPoint → CMView, DisplayFunction → $DisplayFunction];

twoDhitpointplot =
  Table[Graphics[{pointcolor[s, t], PointSize[0.035],
     Point[A.{x[s, t], y[s, t], z[s, t]}]}],
   {s, slow, shigh - sjump, sjump}, {t, tlow, thigh - tjump, tjump}];

pointsafter = Show[twoDhitpointplot,
   PlotRange → {{-ranger, ranger}, {-ranger, ranger}},
   Axes → True, AxesLabel → {"x", "y"}, PlotLabel → "After in 2D",
   DisplayFunction → $DisplayFunction];
```




After the hit, the points on the 3D unit sphere seems to have been deposited on and inside an ellipse in 2D. Try to identify and plot this ellipse.

□ **Answer:**

Do the same thing you do for 2D matrices that are of rank 2

```
Clear[alignerframe, hangerframe, k];
{alignerframe[1], alignerframe[2]} = SingularValues[A]⟦3⟧;
aligner = {alignerframe[1], alignerframe[2]};

MatrixForm[aligner]
```

$$\begin{pmatrix} 0.907798 & 0.415401 & 0.0578371 \\ 0.270294 & -0.6849 & 0.676649 \end{pmatrix}$$

```
{xstretch, ystretch} = SingularValues[A]⟦2⟧;
stretcher = DiagonalMatrix[{xstretch, ystretch}];

MatrixForm[stretcher]
```

$$\begin{pmatrix} 2.54422 & 0 \\ 0 & 2.20611 \end{pmatrix}$$

```
{hangerframe[1], hangerframe[2]} = SingularValues[A]⟦1⟧;
hanger = Transpose[{hangerframe[1], hangerframe[2]}];

MatrixForm[hanger]
```

$$\begin{pmatrix} 0.675586 & 0.737281 \\ 0.737281 & -0.675586 \end{pmatrix}$$

Check:

```
MatrixForm[hanger.stretcher.aligner]
MatrixForm[A]
```

$$\begin{pmatrix} 2. & -0.4 & 1.2 \\ 1.3 & 1.8 & -0.9 \end{pmatrix}$$

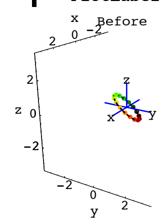
$$\begin{pmatrix} 2 & -0.4 & 1.2 \\ 1.3 & 1.8 & -0.9 \end{pmatrix}$$

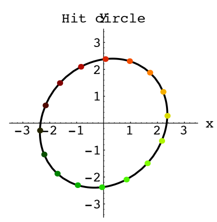
This checks

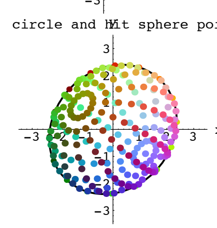
The ellipse you are after comes from hitting the two dimensional planar unit circle (within 3D) defined by alignerframe[1] and alignerframe[2] with A.

See it happen:

```
Clear[threeDcircleplotter, t];
threeDcircleplotter[t_] =
  Cos[t] alignerframe[1] + Sin[t] alignerframe[2];

{tlow, thigh} = {0, 2 π};

threeDcircle = ParametricPlot3D[threeDcircleplotter[t],
   {t, tlow, thigh}, DisplayFunction → Identity];

pointcolor[t_] = RGBColor[0.5 (Cos[t] + 1), 0.5 (Sin[t] + 1), 0];
tjump = π/8;

threeDcirclepoints =
  Table[Graphics3D[{pointcolor[t], PointSize[0.025], Point[
      threeDcircleplotter[t]]}], {t, tlow, thigh - tjump, tjump}];

beforehit =
  Show[threeDcircle, threeDcirclepoints, ThreeAxes[1.5], PlotRange →
    {{-ranger, ranger}, {-ranger, ranger}, {-ranger, ranger}},
   Axes → True, Boxed → False, AxesLabel → {"x", "y", "z"},
   PlotLabel → "Before", ViewPoint → CMView,
   DisplayFunction → $DisplayFunction];

hitthreeDcircle =
  ParametricPlot[A.threeDcircleplotter[t], {t, tlow, thigh},
   PlotStyle → {{Thickness[0.01]}}, DisplayFunction → Identity];

hitthreeDcirclepoints =
  Table[Graphics[{pointcolor[t], PointSize[0.03], Point[
      A.threeDcircleplotter[t]]}], {t, tlow, thigh - tjump, tjump}];

afterhit = Show[hitthreeDcircle, hitthreeDcirclepoints,
   PlotRange → {{-ranger, ranger}, {-ranger, ranger}},
   PlotLabel → "Hit circle", AxesLabel → {"x", "y"},
   DisplayFunction → $DisplayFunction];

Show[afterhit, pointsafter,
  PlotLabel → "Hit circle and hit sphere points"];
```



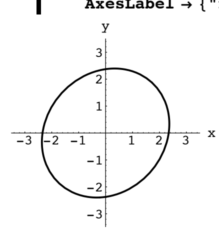



There you go.

When you hit the 3D unit sphere with A, you get everything inside and on this ellipse:

```
ellipseplot = Show[hitthreeDcircle,
   PlotRange → {{-ranger, ranger}, {-ranger, ranger}},
   AxesLabel → {"x", "y"}, DisplayFunction → $DisplayFunction];
```



A parametric formula for this ellipse is:

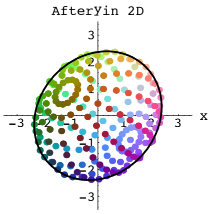
```
Expand[A.threeDcircleplotter[t]]
```

{1.71884 Cos[t] + 1.62653 Sin[t], 1.87581 Cos[t] - 1.49042 Sin[t]}

### □ T.6.b.ii) Framing up the ellipse

Take another look at the ellipse in part i):

```
Show[pointsafter, ellipseplot];
```

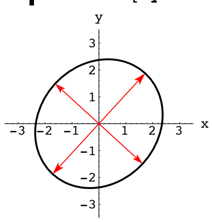



Frame up this ellipse.

**Answer:**

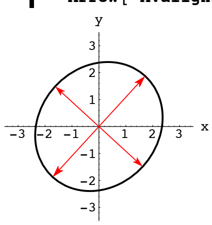
You've got two choices which end up being the same:

```
frameup = Show[ellipseplot,
   Arrow[xstretch hangerframe[1], Tail → {0, 0}, VectorColor → Red],
   Arrow[-xstretch hangerframe[1], Tail → {0, 0}, VectorColor → Red],
   Arrow[ystretch hangerframe[2], Tail → {0, 0}, VectorColor → Red],
   Arrow[-ystretch hangerframe[2], Tail → {0, 0}, VectorColor → Red]];
```




Or:

```
Show[ellipseplot,
  Arrow[A.alignerframe[1], Tail → {0, 0}, VectorColor → Red],
  Arrow[-A.alignerframe[1], Tail → {0, 0}, VectorColor → Red],
  Arrow[A.alignerframe[2], Tail → {0, 0}, VectorColor → Red],
  Arrow[-A.alignerframe[2], Tail → {0, 0}, VectorColor → Red]];
```




They are the same because SVD analysis always guarantees:

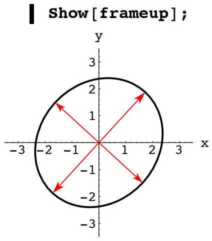
```
A.alignerframe[1] == xstretch hangerframe[1]
A.alignerframe[2] == ystretch hangerframe[2]
```

```
True
True
```

**T.6.b.iii) Measuring the area enclosed by the ellipse**

Take another look at the framed ellipse in part ii):

```
Show[frameup];
```




Measure the area enclosed by this ellipse.

**Answer:**

The area enclosed by the ellipse measures out to:

```
xstretch ystretch π
```

```
17.6333
```

Another way of seeing this is to take the parametric formula for the ellipse:

```
Expand[A.threeDcircleplotter[t]]
```

```
{1.71884 Cos[t] + 1.62653 Sin[t], 1.87581 Cos[t] - 1.49042 Sin[t]}
```

Put:

$$B = \begin{pmatrix} 1.71884 & -1.62653 \\ 1.87581 & 1.49042 \end{pmatrix};$$

```
MatrixForm[B]
```

$$\begin{pmatrix} 1.71884 & -1.62653 \\ 1.87581 & 1.49042 \end{pmatrix}$$

Note that the ellipse is also parameterized by:

```
B.{Cos[t], Sin[t]}
```

```
{1.71884 Cos[t] - 1.62653 Sin[t], 1.87581 Cos[t] + 1.49042 Sin[t]}
```

Knowing what you do about 2D matrices, you find quickly that the area enclosed by the ellipse measures out to:

```
Abs[Det[B]] π
```

```
17.6333
```

**T.6.b.iv) Would the determinant help?**

Could you have used the determinant of A to help in measuring the area enclosed by the ellipse?

**Answer:**

Try it and see:

```
Det[A]
```

```
Det::matsq : Argument {{2, -0.4, 1.2}, {1.3, 1.8, -0.9}} at position 1 is not a square matrix.
Det[{{2, -0.4, 1.2}, {1.3, 1.8, -0.9}}]
```

Useless.

Reason: The determinant of a matrix that hits on 3D and hangs in 2D is not defined.

**T.6.b.v) The rank of A is 2**

Stay with the same matrix A as in the earlier parts and say why the rank of A is 2.

**Answer:**

The rank of a matrix is the number of dimensions that matrix uses to hang its hits.

When the above matrix A is hit on all of on all of 3D, it hangs its hits on all of 2D.

That's why the rank of A is 2.