Matrices, Geometry&*Mathematica*

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

Producer: Bruce Carpenter

Publisher: Math Everywhere, Inc.

MGM.03 Using Aligners, Stretchers and Hangers to Make 2D Matrices Do What You Want

BASICS

B.1) The matrix maker ingredients:

hangers, stretchers, aligners and how to use them to make custom 2D matrices via

A = hanger. $\begin{pmatrix} \text{xstretch} & 0 \\ 0 & \text{ystretch} \end{pmatrix}$ **.aligner**

In this case,

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

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

This is the single most important problem in the whole course.
Nearly everything in the rest of the course is based on the ideas in this problem.

□B.1.a.i) Ordering up a matrix by specifying an aligner, an xystretcher, and a hanger and putting

yourmatrix = hanger.stretcher.aligner

When you order up a meal in a nice restaurant, you make three choices:
1) an appetizer,
2) a main course and
3) a dessert
and then you eat them in that order.

When you order up a matrix, you also make three choices:
1) **a perpendicular frame (an alignerframe) to define your aligner,**
2) **two stretch factors to define your stretcher**
3) **another perpendicular frame (a hangerframe) to define your hanger**
and then you multiply them out in that order.
yourmatrix = hanger.xystretcher.aligner
These are the matrix maker ingredients.

Try them out:
Go with this sample right hand perpendicular frame for your aligner:

```
Clear[alignerframe];
s = 0.4 π;
{alignerframe[1], alignerframe[2]} =
     N[{{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}}];
aligner = {alignerframe[1], alignerframe[2]};
MatrixForm[aligner]
```

$$\begin{pmatrix} 0.309017 & 0.951057 \\ -0.951057 & 0.309017 \end{pmatrix}$$

alignerframe[1] is the first horizontal row of aligner.
alignerframe[2] is the second horizontal row of aligner.

Go with these sample stretch factors for your stretcher:

```
{xstretch, ystretch} = {2.2, 0.5};
stretcher = {{xstretch, 0}, {0, ystretch}};
MatrixForm[stretcher]
```

$$\begin{pmatrix} 2.2 & 0 \\ 0 & 0.5 \end{pmatrix}$$

Go with this sample right hand perpendicular frame for your hanger:

```
Clear[hangerframe];
s = 0.2 π;
{hangerframe[1], hangerframe[2]} =
     N[{{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}}];
hanger = Transpose[{hangerframe[1], hangerframe[2]}];
MatrixForm[hanger]
```

$$\begin{pmatrix} 0.809017 & -0.587785 \\ 0.587785 & 0.809017 \end{pmatrix}$$

hangerframe[1] is the first vertical column of hanger.
hangerframe[2] is the second vertical column of hanger.

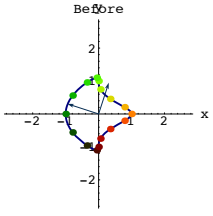
Now multiply them out in order:

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

$$\begin{pmatrix} 0.829508 & 1.60191 \\ 0.0148879 & 1.35484 \end{pmatrix}$$

Here's a curve shown with the perpendicular frame chosen for the aligner:

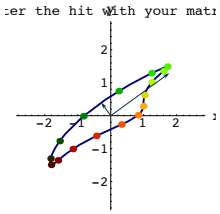
```
Clear[x, y, t, s];
{tlow, thigh} = {0, 2 π};
ranger := 1.3 Max[{1.0, xstretch, ystretch}];

{x[t_], y[t_]} = {Cos[t]^3, Sin[t] (1 - 0.5 Cos[t])};

Clear[hitplotter, hitpointplotter,
  pointcolor, actionarrows, matrix2D];
pointcolor[t_] = RGBColor[0.5 (Cos[t] + 1), 0.5 (Sin[t] + 1), 0];

jump = (thigh - tlow)/16;

hitplotter[matrix2D_] := ParametricPlot[matrix2D.{x[t], y[t]},
   {t, tlow, thigh}, PlotStyle → {{Thickness[0.01], NavyBlue}},
   PlotRange → {{-ranger, ranger}, {-ranger, ranger}},
   AxesLabel → {"x", "y"}, DisplayFunction → Identity];

hitpointplotter[matrix2D_] :=
  Table[Graphics[{pointcolor[t], PointSize[0.035],
     Point[matrix2D.{x[t], y[t]}]}], {t, tlow, thigh - jump, jump}];

hitframeplotter[matrix2D_] :=
 Table[Arrow[matrix2D.alignerframe[k], Tail → {0, 0},
   VectorColor → Indigo, HeadSize → 0.2], {k, 1, 2}]

before = Show[hitplotter[IdentityMatrix[2]],
   hitpointplotter[IdentityMatrix[2]],
   hitframeplotter[IdentityMatrix[2]], PlotLabel → "Before",
   DisplayFunction → $DisplayFunction];
```



The plotted perpendicular frame is the frame chosen for the aligner.

Here's what a hit with your matrix does to to this curve and to your aligner frame:

```
after = Show[hitplotter[yourmatrix],
    hitpointplotter[yourmatrix],
    hitframeplotter[yourmatrix],
    PlotLabel -> "After the hit with your matrix",
    DisplayFunction -> $DisplayFunction];
```

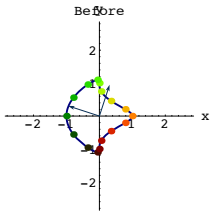


Use the specifications of the aligner, the stretcher and the hanger to explain what a hit with this matrix does.

Answer:

Before the hit, the aligner frame and the curve look this way:

```
Show[before];
```

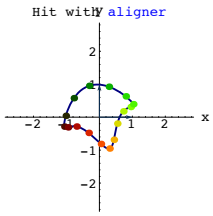


When you hit with

yourmatrix = hanger.xystetcher.aligner,

you first hit with aligner. See what this does:

```
alignhit = Show[hitplotter[aligner],
  hitpointplotter[aligner],
  hitframeplotter[aligner],
  PlotLabel -> "Hit with aligner",
  DisplayFunction -> $DisplayFunction];
```

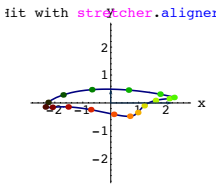


Grab the last two plots and animate.

The hit with aligner aligns alignerframe[1] with the positive x-axis and this aligns alignerframe[2] with the positive y-axis and aligns the curve the same way..

Next, you hit with stretcher; see the result:

```
stretchalignhit = Show[hitplotter[stretcher.aligner],
  hitpointplotter[stretcher.aligner],
  hitframeplotter[stretcher.aligner],
  PlotLabel -> "Hit with stretcher.aligner",
  DisplayFunction -> $DisplayFunction];
```



Grab the last two plots and animate.

Looking at:

```
MatrixForm[stretcher]
```

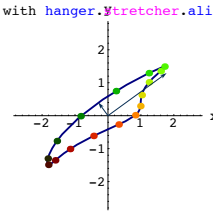
$$\begin{pmatrix} 2.2 & 0 \\ 0 & 0.5 \end{pmatrix}$$

you can see that the x coordinates were stretched by a factor of 2.2 and the y coordiantes were stretched by a factor of 0.5.

And finally, you hang this curve on your chosen hangerframe by hitting with hanger.

See the result:

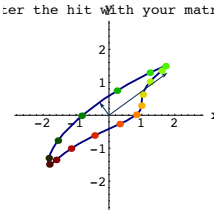
```
hangstretchalignhit =
 Show[hitplotter[hanger.stretcher.aligner],
    hitpointplotter[hanger.stretcher.aligner],
    hitframeplotter[hanger.stretcher.aligner],
    PlotLabel -> "Hit with hanger.stretcher.aligner",
    DisplayFunction -> $DisplayFunction];
```



Now you see the final product hanging on the frame chosen for your hanger matrix.
Grab the last four plots and animate

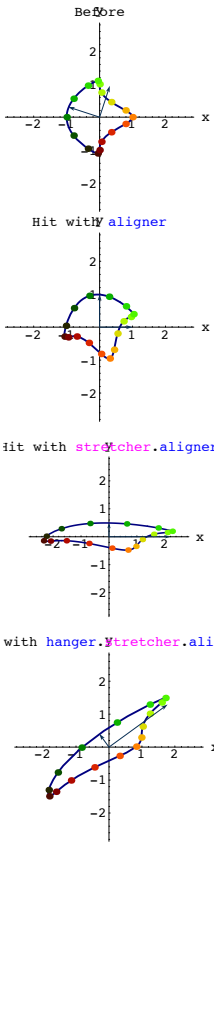
Compare:

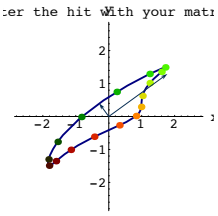
```
Show[after];
```



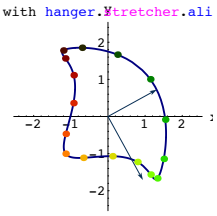
Review:

```
Show[before];
Show[alignhit];
Show[stretchalignhit];
Show[hangstretchalignhit];
Show[after];
```





Grab the plots and animate slowly.

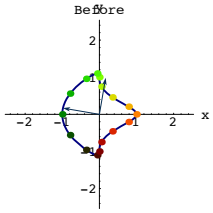


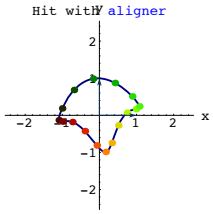
Grab,animate and then rerun.

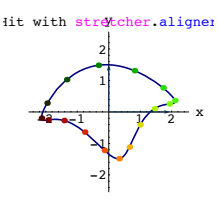
Do the same thing for random selections of hanger, stretcher and aligner:

```
Clear[alignerframe];
s = Random[Real, {-1.5, 1.5}];
{alignerframe[1], alignerframe[2]} = {{Cos[s], Sin[s]},
    ((-1)^Random[Integer,{0,1}]) {Cos[s + π/2], Sin[s + π/2]}};

aligner = {alignerframe[1], alignerframe[2]};

{xstretch, ystretch} =
  {Random[Real, {0.5, 3}], Random[Real, {0.5, 3}]};

stretcher = ( xstretch    0
              0       ystretch );

Clear[hangerframe];
s = Random[Real, {-1.5, 1.5}];
{hangerframe[1], hangerframe[2]} =
  {{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}};

hanger = Transpose[{hangerframe[1], hangerframe[2]}];
A = hanger.stretcher.aligner;

"This matrix is" MatrixForm[A]
before = Show[hitplotter[IdentityMatrix[2]],
   hitpointplotter[IdentityMatrix[2]],
   hitframeplotter[IdentityMatrix[2]], PlotLabel → "Before",
   DisplayFunction → $DisplayFunction];

alignhit = Show[hitplotter[aligner],
  hitpointplotter[aligner],
  hitframeplotter[aligner],
  PlotLabel -> "Hit with aligner",
  DisplayFunction -> $DisplayFunction];

stretchalignhit = Show[hitplotter[stretcher.aligner],
  hitpointplotter[stretcher.aligner],
  hitframeplotter[stretcher.aligner],
  PlotLabel -> "Hit with stretcher.aligner",
  DisplayFunction -> $DisplayFunction];

hangstretchalignhit =
 Show[hitplotter[hanger.stretcher.aligner],
   hitpointplotter[hanger.stretcher.aligner],
   hitframeplotter[hanger.stretcher.aligner],
   PlotLabel -> "Hit with hanger.stretcher.aligner",
   DisplayFunction -> $DisplayFunction];
```

This matrix is $\begin{pmatrix} -1.1406 & 1.1496 \\ -1.00292 & -1.56399 \end{pmatrix}$







□**B.1.b.i) If A = hanger.stretcher.aligner,**

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

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

Here's a random matrix made with the matrix maker ingredients aligner, stretcher, and hanger:

```
MatrixForm[{{xstretch, 0}, {0, ystretch}}]
```

$\begin{pmatrix} 1.95234 & 0 \\ 0 & 1.50427 \end{pmatrix}$

```
Clear[alignerframe];
s = Random[Real, {-1.5, 1.5}];
{alignerframe[1], alignerframe[2]} =
  N[{{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}}];

aligner = {alignerframe[1], alignerframe[2]};

{xstretch, ystretch} = {Random[Real, {0, 2}], Random[Real, {0, 2}]};

stretcher = ( xstretch    0
              0       ystretch );

Clear[hangerframe];
s = Random[Real, {-1.5, 1.5}];
{hangerframe[1], hangerframe[2]} =
  N[{{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}}];

hanger = Transpose[{hangerframe[1], hangerframe[2]}];
A = hanger.stretcher.aligner;

MatrixForm[A]
```

$\begin{pmatrix} 0.784831 & -0.866916 \\ 1.17021 & -1.097 \end{pmatrix}$

Look at this:

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

True

True

Rerun both cells in this part several times.

Explain what this means.

□**Answer:**

This says that the total effect of a hit with a matrix A made with the matrix maker can be fairly concisely described by saying:

- The hit with A picks up alignerframe[1], stretches it by a factor of xstretch, and then plunks the result down in the direction of hangerframe[1],

and

▪ The hit with A picks up alignerframe[2], stretches it by a factor of ystretch, and then plunks the result down in the direction of hangerframe[2].

This tells you that A.alignerframe[1] guaranteed to be perpendicular to A.alignerframe[2]:

```
(A.alignerframe[1]).(A.alignerframe[2])
```

0

□ B.1.b.ii) Making a matrix with prescribed hits

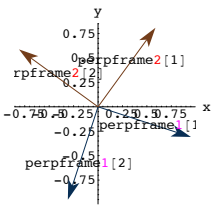
A.perpframe1[1] = 0.5 perpframe2[1]

and

A.perpframe1[2] = 1.6 perpframe2[2]

Here are two perpendicular frames:

```
Clear[perpframe1, perpframe2];
s = -0.1 Pi;
{perpframe1[1], perpframe1[2]} =
     N[{{Cos[s], Sin[s]}, -{Cos[s + Pi / 2], Sin[s + Pi / 2]}}];

s = 0.3 Pi;
{perpframe2[1], perpframe2[2]} =
     N[{{Cos[s], Sin[s]}, {Cos[s + Pi / 2], Sin[s + Pi / 2]}}];
frame1plot = {Table[
  Arrow[perpframe1[k], Tail -> {0, 0},
    VectorColor -> Indigo, HeadSize -> 0.2], {k, 1, 2}],
  Graphics[Text["perpframe1[1]", 0.6 perpframe1[1]]],
  Graphics[Text["perpframe1[2]", 0.6 perpframe1[2]]]};
frame2plot = {Table[
  Arrow[perpframe2[k], Tail -> {0, 0},
    VectorColor -> DeepOchre, HeadSize -> 0.2], {k, 1, 2}],
  Graphics[Text["perpframe2[1]", 0.6 perpframe2[1]]],
  Graphics[Text["perpframe2[2]", 0.6 perpframe2[2]]]};
frameplots =
  Show[frame1plot, frame2plot, Axes -> True, AxesLabel -> {"x", "y"}];
```



Make a matrix A so that

A.perpframe1[1] = 0.5 perpframe2[1]

and

A.perpframe1[2] = 1.6 perpframe2[2].

Back up your answer with a convincing calculation.

□ Answer:

Remember that when you make a matrix with matrix maker ingredients:

aligner,stretcher,hanger, then you are guaranteed that

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

and

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

The job here is to make a matrix A so that

A.perpframe1[1] = 0.5 perpframe2[1]

and

A.perpframe1[2] = 1.6 perpframe2[2]

You can arrange for this by going with:

alignerframe = perpframe1

xstretch = 0.5

ystretch = 1.6

and

hangerframe = perpframe2.

Go for it:

```
aligner = {perpframe1[1], perpframe1[2]};
MatrixForm[aligner]
```

$$\begin{pmatrix} 0.951057 & -0.309017 \\ -0.309017 & -0.951057 \end{pmatrix}$$

perpframe1[1] is the first horizontal row of aligner.
perpframe1[2] is the second horizontal row of aligner.

```
{xstretch, ystretch} = {0.5, 1.6};
stretcher = (xstretch  0
             0  ystretch);
MatrixForm[stretcher]
```

$$\begin{pmatrix} 0.5 & 0 \\ 0 & 1.6 \end{pmatrix}$$

```
hanger = Transpose[{perpframe2[1], perpframe2[2]}];
MatrixForm[hanger]
```

$$\begin{pmatrix} 0.587785 & -0.809017 \\ 0.809017 & 0.587785 \end{pmatrix}$$

hangerframe[1] is the first vertical column of hanger.
hangerframe[2] is the second vertical column of hanger.

Now multiply them out in order to get the matrix you are after:

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

$$\begin{pmatrix} 0.679508 & 1.14026 \\ 0.0940934 & -1.01943 \end{pmatrix}$$

Check whether

A.perpframe1[1] = 0.5 perpframe2[1]

and

A.perpframe1[2] = 1.6 perpframe2[2]:

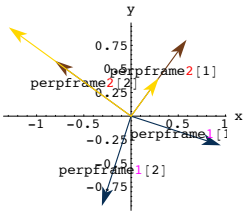
```
A.perpframe1[1] == 0.5 perpframe2[1]
    A.perpframe1[2] == 1.6 perpframe2[2]
```

True

True

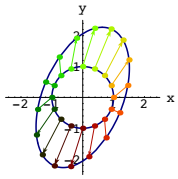
See it:

```
Show[frameplots, Arrow[A.perpframe1[1], Tail -> {0, 0},
  VectorColor -> Gold, HeadSize -> 0.2], Arrow[A.perpframe1[2],
  Tail -> {0, 0}, VectorColor -> Gold, HeadSize -> 0.2]];
```



Had it all the way.

B.2) Using matrix maker ingredients to make positive definite matrices (frame stretchers) and reflection matrices (frame flippers)

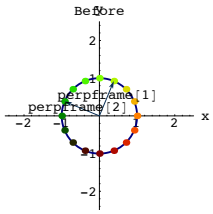


□ B.2.a) Making positive definite matrices (frame stretchers)

Here's a perpendicular frame shown with the unit circle centered at {0,0}::

```
Clear[x, y, t, perpframe, s];
{tlow, thigh} = {0, 2 π};
ranger = 2.5;
{x[t_], y[t_]} = {Cos[t], Sin[t]};

s = 3 π / 8;
{perpframe[1], perpframe[2]} =
  N[{{Cos[s], Sin[s]}, {Cos[s + π / 2], Sin[s + π / 2]}}];

Clear[hitplotter, hitpointplotter,
  pointcolor, actionarrows, matrix2D];
pointcolor[t_] = RGBColor[0.5 (Cos[t] + 1), 0.5 (Sin[t] + 1), 0];

jump = (thigh - tlow) / 16;

hitplotter[matrix2D_] := ParametricPlot[matrix2D.{x[t], y[t]},
   {t, tlow, thigh}, PlotStyle → {{Thickness[0.01], NavyBlue}},
   PlotRange → {{-ranger, ranger}, {-ranger, ranger}},
   AxesLabel → {"x", "y"}, DisplayFunction → Identity];

hitpointplotter[matrix2D_] :=
  Table[Graphics[{pointcolor[t], PointSize[0.035],
     Point[matrix2D.{x[t], y[t]}]}], {t, tlow, thigh - jump, jump}];

hitframeplotter[matrix2D_] :=
 Table[Arrow[matrix2D.perpframe[k], Tail → {0, 0},
```

```
    VectorColor → Indigo, HeadSize → 0.2], {k, 1, 2}]

actionarrows[matrix2D_] :=
  Table[Arrow[matrix2D.{x[t], y[t]} - {x[t], y[t]}, Tail → {x[t], y[t]},
    VectorColor → pointcolor[t]], {t, tlow, thigh - jump, jump}];

framelabels = {Graphics[Text["perpframe[1]", 0.6 perpframe[1]]],
   Graphics[Text["perpframe[2]", 0.6 perpframe[2]]]};

before = Show[hitplotter[IdentityMatrix[2]],
   hitpointplotter[IdentityMatrix[2]],
   hitframeplotter[IdentityMatrix[2]], framelabels,
   PlotLabel → "Before", DisplayFunction → $DisplayFunction];
```



Your job is to make a positive definite matrix (frame stretcher) A whose hits
- stretch every measurement in the direction of perpframe[1] by a factor of 2.4

and
- stretch every measurement in the direction of perpframe[2] by a factor of 1.3.

Show off your matrix with action movie.

Answer:

The job here is to make a frame stretcher matrix A whose hits

- stretch every measurement in the direction of perpframe[1] by a factor of 2.4

and

- stretch every measurement in the direction of perpframe[2] by a factor of 1.3.

This is the same as saying

A.perpframe[1] = 2.4 perpframe[1]

and

A.perpframe[2] = 1.3 perpframe[2].

Remember that when you make a matrix with matrix maker ingredients:

aligner,stretcher,hanger, then you are guaranteed that

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

and

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

You can arrange for this by going with:

alignerframe = the given perpendicular frame

hangerframe = the same given perpendicular frame

xstretch = 2.4,

and

ystretch = 1.3.

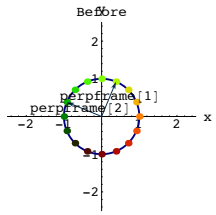
The Matrix A you are after is:

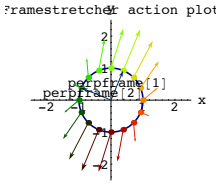
```
aligner = {perpframe[1], perpframe[2]};

{xstretch, ystretch} = {2.4, 1.3};
stretcher = ( xstretch      0
                0       ystretch );

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

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

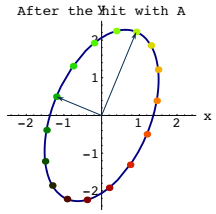
$$\begin{pmatrix} 1.46109 & 0.388909 \\ 0.388909 & 2.23891 \end{pmatrix}$$

Here comes the action movie:

```
before = Show[hitplotter[IdentityMatrix[2]],
    hitpointplotter[IdentityMatrix[2]],
        hitframeplotter[IdentityMatrix[2]],
   framelabels, PlotLabel -> "Before",
    DisplayFunction -> $DisplayFunction];

Show[before, actionarrows[A],
  PlotLabel -> "Framestretcher action plot",
   DisplayFunction -> $DisplayFunction];

Show[hitplotter[A],
    hitpointplotter[A],
        hitframeplotter[A],
    PlotLabel -> "After the hit with A",
    DisplayFunction -> $DisplayFunction];
```







Grab and animate the last three plots,running at various speeds.

The action of positive definite matrix (frame stretcher).

You may be interested as to why frame stretcher matrices have the name positive definite matrices.

If so, click on the right,

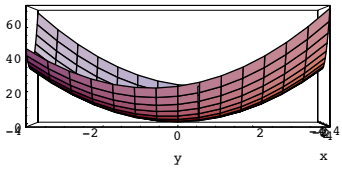
When you make a frame stretcher matrix A, you stretch every measurement in the direction of the given frames by a positive amount.
This results in

{x,y}.(A.{x,y}) > 0 unless {x,y} is {0,0}.

That's why these frame stretching matrices are often called positive definite.

Check this out by plotting {x,y}.(A.{x,y})

```
Plot3D[{x, y}.(A.{x, y}), {x, -4, 4}, {y, -4, 4},
  ViewPoint -> 12 {1, 0, 0}, Axes -> True, AxesLabel -> {"x", "y", ""}];
```

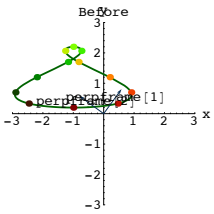


B.2.b.i) Making reflection matrices (frame flippers)

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

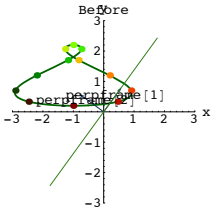
```
Clear[x, y, t, perpframe, s];
s = 0.3 π;
{perpframe[1], perpframe[2]} =
  N[{{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}}];

framelabels = {Graphics[Text["perpframe[1]", 0.7 perpframe[1]]],
   Graphics[Text["perpframe[2]", 0.7 perpframe[2]]]};

{tlow, thigh} = {0, 2 π};
ranger = 3.0;
{x[t_], y[t_]} = {-1, 1.2} + {2 Cos[t] (0.6 - Sin[t]), Sin[t]};

Clear[hitplotter, hitpointplotter,
  pointcolor, actionarrows, matrix2D];
pointcolor[t_] = RGBColor[0.5 (Cos[t] + 1), 0.5 (Sin[t] + 1), 0];

jump = (thigh - tlow)/12;

hitplotter[matrix2D_] := ParametricPlot[matrix2D.{x[t], y[t]},
   {t, tlow, thigh}, PlotStyle → {{Thickness[0.01], GosiaGreen}},
   PlotRange → {{-ranger, ranger}, {-ranger, ranger}},
   AxesLabel → {"x", "y"}, DisplayFunction → Identity];

hitpointplotter[matrix2D_] :=
  Table[Graphics[{pointcolor[t], PointSize[0.035],
     Point[matrix2D.{x[t], y[t]}]}], {t, tlow, thigh - jump, jump}];

hitframeplotter[matrix2D_] :=
 Table[Arrow[matrix2D.perpframe[k], Tail → {0, 0},
   VectorColor → Indigo], {k, 1, 2}]

actionarrows[matrix2D_] :=
  Table[Arrow[matrix2D.{x[t], y[t]} - {x[t], y[t]}, Tail → {x[t], y[t]},
    VectorColor → pointcolor[t]], {t, tlow, thigh - jump, jump}];
```

```
Show[hitplotter[IdentityMatrix[2]],
  hitpointplotter[IdentityMatrix[2]],
  hitframeplotter[IdentityMatrix[2]], framelabels,
  PlotLabel → "Before", DisplayFunction → $DisplayFunction];
```



Put a line through perpframe[1]:

```
flipline =
Graphics[{SapGreen, Thickness[0.005],
    Line[{-3 perpframe[1], 3 perpframe[1]}]}];

before = Show[hitplotter[IdentityMatrix[2]],
    hitpointplotter[IdentityMatrix[2]],
    hitframeplotter[IdentityMatrix[2]], framelabels,
    flipline,
    PlotLabel -> "Before",
    DisplayFunction -> $DisplayFunction];
```



Your job to use the matrix maker ingredients to come up with a matrix A whose hits pick up the curve and flips it over the plotted line to get its mirror image with respect to the plotted line.
Plot to confirm.

□ Answer:

The job here is to make a matrix A whose hits

→ preserve everything in the direction of perpframe[1] (along the line)

→ multiply everything in the direction of perpframe[2] by -1.

This is the same as saying

A.perpframe[1] = perpframe[1]

and

A.perpframe[2] = −perpframe[2].

When you make a matrix

A = hanger.stretcher.aligner

with matrix maker ingredients:

aligner,stretcher,hanger, then you are guaranteed that

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

and

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

You can arrange everything you want by going with:

xstretch = ystretch = 1,

alignerframe = {perpframe[1], perpframe[2]}

hangerframe = {perpframe[1],-perpframe[2]}.

Note the minus sign on perpframe[2].

The matrix A you are after is:

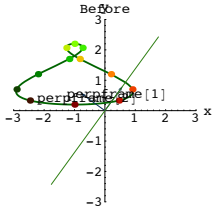
```
aligner = {perpframe[1], perpframe[2]};

{xstretch, ystretch} = {1.0, 1.0};
stretcher = (xstretch  0
             0  ystretch);

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

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

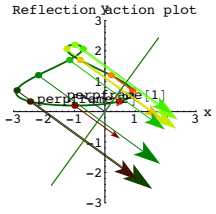
$$\begin{pmatrix} -0.309017 & 0.951057 \\ 0.951057 & 0.309017 \end{pmatrix}$$

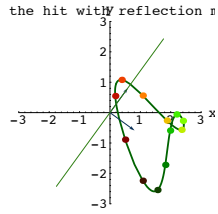
See A do its work:

```
before = Show[hitplotter[IdentityMatrix[2]],
   hitpointplotter[IdentityMatrix[2]],
       hitframeplotter[IdentityMatrix[2]],
   framelabels, flipline, PlotLabel -> "Before",
    DisplayFunction -> $DisplayFunction];
```

```
Show[before, actionarrows[A],
  flipline, PlotLabel -> "Reflection action plot",
   DisplayFunction -> $DisplayFunction];

Show[hitplotter[A],
    hitpointplotter[A],
        hitframeplotter[A], flipline,
    PlotLabel -> "After the hit with reflection matrix",
    DisplayFunction -> $DisplayFunction];
```







Grab and animate the last three plots,running at various speeds.

The action of a reflection matrix (frame flipper).

□ B.2.b.ii) Another way

Is there another easy way to use the matrix maker ingredients to make the frameflipper matrix in part i)?

□ Answer:

Yes, provided you are willing to use a negative stretch factor.

Remember that the job was to make a framestretcher matrix A whose hits

→ preserve everything in the direction of perpframe[1] (along the line)

→ multiply everything in the direction of perpframe[2] by -1.

This is the same as saying

A.perpframe[1] = perpframe[1]

and

A.perpframe[2] = −perpframe[2].

When you make a matrix with matrix maker ingredients:

aligner, stretcher, hanger, then you are guaranteed that

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

and

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

You can arrange for what you want by going with:

xstretch = 1,

ystretch = -1

hangerframe = alignerframe = {perpframe[1], perpframe[2]}

The resulting alternative matrix A is:

```
aligner = {perpframe[1], perpframe[2]};

{xstretch, ystretch} = {1.0, -1.0};
stretcher = (xstretch  0
             0  ystretch);

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

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

$$\begin{pmatrix} -0.309017 & 0.951057 \\ 0.951057 & 0.309017 \end{pmatrix}$$

And this is the same as the A from part i) above:

```
aligner = {perpframe[1], perpframe[2]};
```

```
{xstretch, ystretch} = {1.0, 1.0};
stretcher = ( xstretch    0
              0       ystretch );

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

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

$$\begin{pmatrix} -0.309017 & 0.951057 \\ 0.951057 & 0.309017 \end{pmatrix}$$

B.3) Making the inverse matrix and the transpose matrix.

If

$$A = \text{hanger}.\begin{pmatrix} \text{xstretch} & 0 \\ 0 & \text{ystretch} \end{pmatrix}.\text{aligner},$$

then

$$A^{-1} = \text{aligner}^t \cdot \begin{pmatrix} \frac{1}{\text{xstretch}} & 0 \\ 0 & \frac{1}{\text{ystretch}} \end{pmatrix} \cdot \text{hanger}^t \cdot$$

and

$$A^t = \text{aligner}^t \cdot \begin{pmatrix} \text{xstretch} & 0 \\ 0 & \text{ystretch} \end{pmatrix} \cdot \text{hanger}^t \cdot$$

The hanger frame for A^{-1} and for A^t is the aligner frame for A.

The aligner frame for A^{-1} and for A^t is the hanger frame for A.

☐ B.3.a.i) Using the hangerframe, aligner frame and the stretch factors to make the inverse matrix

Here's a matrix A made with matrix maker ingredients using
a random aligner frame,
two random stretch factors
and
a random hanger frame:

```
Clear[alignerframe];
s = Random[Real, {-π/2, π/2}];
```

```
{alignerframe[1], alignerframe[2]} = N[{{Cos[s], Sin[s]},
    (-1)^Random[Integer,{0,1}] {Cos[s + π/2], Sin[s + π/2]}}];

aligner = {alignerframe[1], alignerframe[2]};

{xstretch, ystretch} =
  {Random[Real, {0.1, 3.0}], Random[Real, {0.1, 3.0}]};
stretcher = ( xstretch    0
              0       ystretch );
Clear[hangerframe];
s = Random[Real, {-π/2, π/2}];

{hangerframe[1], hangerframe[2]} = N[{{Cos[s], Sin[s]},
    (-1)^Random[Integer,{0,1}] {Cos[s + π/2], Sin[s + π/2]}}];

hanger = Transpose[{hangerframe[1], hangerframe[2]}];
A = hanger.stretcher.aligner;

MatrixForm[A]
```

$$\begin{pmatrix} 0.203477 & 0.788456 \\ -0.423625 & 0.350447 \end{pmatrix}$$

Look at this calculation of

$$\text{aligner}^t \cdot \begin{pmatrix} \frac{1}{\text{xstretch}} & 0 \\ 0 & \frac{1}{\text{ystretch}} \end{pmatrix} \cdot \text{hanger}^t :$$

```
MatrixForm[
 Transpose[aligner].( 1/xstretch    0
                      0         1/ystretch ).Transpose[hanger]]
```

$$\begin{pmatrix} 0.864624 & -1.94528 \\ 1.04517 & 0.502019 \end{pmatrix}$$

Compare with *Mathematica*'s calculation of A^{-1}:

```
MatrixForm[Inverse[A]]
```

$$\begin{pmatrix} 0.864624 & -1.94528 \\ 1.04517 & 0.502019 \end{pmatrix}$$

Got it!
Explain why this is guaranteed to work every time you run it.

☐ **Answer:**

When you think about what results from hits by aligners and hangers, you gotta agree that:

- a hit with hanger undoes whatever a hit with aligner does
- a hit with aligner undoes whatever a hit with hanger does.

So:

For any perpendicular frame,

$$\text{aligner}^{-1} = \text{hanger} = \text{aligner}^t$$

and

$$\text{hanger}^{-1} = \text{aligner} = \text{hanger}^t.$$

The upshot:
To invert any aligner or any hanger,
you just take its transpose.

Now stop for a minute and think:
In real life:
When you get up in the morning ,
1) you put on your **socks**,
2) you put on your **shoes**,
3) you tie your shoe **laces** ---in that order.

When you go to bed at night ,
1) you untie your shoe **laces**,
2) you take off on your **shoes**,
3) you take off your **socks**, --- in that order.
To undo what you did in the morning, you reverse the order.

In the world of matrices:
To undo a hit with

$$A = \text{hanger}.\begin{pmatrix} \text{xstretch} & 0 \\ 0 & \text{ystretch} \end{pmatrix}.\text{aligner},$$

you hit with

$$\text{aligner}^{-1}\begin{pmatrix} \text{xstretch} & 0 \\ 0 & \text{ystretch} \end{pmatrix}^{-1}.\text{hanger}^{-1}$$

$$= \text{aligner}^t\begin{pmatrix} \text{xstretch} & 0 \\ 0 & \text{ystretch} \end{pmatrix}^{-1}.\text{hanger}^t$$

And to invert $\begin{pmatrix} \text{xstretch} & 0 \\ 0 & \text{ystretch} \end{pmatrix}$ you just undo the stretches by hitting with $\begin{pmatrix} \frac{1}{\text{xstretch}} & 0 \\ 0 & \frac{1}{\text{ystretch}} \end{pmatrix}$.
That's why

$$A^{-1} = \text{aligner}^t \cdot \begin{pmatrix} \frac{1}{\text{xstretch}} & 0 \\ 0 & \frac{1}{\text{ystretch}} \end{pmatrix} \cdot \text{hanger}^t \cdot$$

Review:

```
Show[Graphics[
  Text["A = hanger.( xstretch    0
                     0       ystretch ).aligner", {0, 0}]],
 AspectRatio -> 1/5];
Show[Graphics[
  Text["  A^-1  =  aligner^t .( 1/xstretch    0
                                0         1/ystretch ) .hanger^t", {0, 0}]],
 AspectRatio -> 1/5];
```

$$A = \text{hanger}.\begin{pmatrix} \text{xstretch} & 0 \\ 0 & \text{ystretch} \end{pmatrix}.\text{aligner}$$

$$A^{-1} = \text{aligner}^t \cdot \begin{pmatrix} \frac{1}{\text{xstretch}} & 0 \\ 0 & \frac{1}{\text{ystretch}} \end{pmatrix} \cdot \text{hanger}^t$$

Grab and animate slowly

□B.3.a.ii) A and A^{-1} have inverse stretch factors.

The hanger frame for A^{-1} is the aligner frame for A.

The aligner frame for A^{-1} is the hanger frame for A

Here's a matrix A made with matrix maker ingredients using
a random aligner frame,
two random stretch factors and
a random hanger frame:

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

aligner = {alignerframe[1], alignerframe[2]};
{xstretch, ystretch} =
  {Random[Real, {0.1, 3.0}], Random[Real, {0.1, 3.0}]};
stretcher = DiagonalMatrix[{xstretch, ystretch}];

Clear[hangerframe];
s = Random[Real, {-π/2, π/2}];

{hangerframe[1], hangerframe[2]} = N[{{Cos[s], Sin[s]},
    (-1)^Random[Integer,{0,1}] {Cos[s + π/2], Sin[s + π/2]}}];

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

A = hanger.stretcher.aligner;

MatrixForm[A]
```

$$\begin{pmatrix} 0.460031 & -1.74622 \\ -0.394682 & 2.17131 \end{pmatrix}$$

Calculating A^{-1}, the inverse of A, is so easy you can do it by hand if you are so inclined.
You just:
→ Use the hanger frame of A as the aligner frame of A^{-1}
→ Use the aligner frame of A as the hanger frame of A^{-1}
→ And invert the stretch factors of A.

```
alignerforinverse = {hangerframe[1], hangerframe[2]};
unstretcher = DiagonalMatrix[{1/xstretch, 1/ystretch}];

hangerforinverse = Transpose[{alignerframe[1], alignerframe[2]}];
Ainverse = hangerforinverse.unstretcher.alignerforinverse;

MatrixForm[Ainverse]
```

$$\begin{pmatrix} 7.01176 & 5.63904 \\ 1.27454 & 1.48557 \end{pmatrix}$$

Compare with *Mathematica*'s calculation of A^{-1}:

```
MatrixForm[Inverse[A]]
```

$$\begin{pmatrix} 7.01176 & 5.63904 \\ 1.27454 & 1.48557 \end{pmatrix}$$

Got it.

Rerun all cells above from the beginning of the problem to this stage several times.

Explain why this is guaranteed to work every time you run it.

□Answer:

You bet!

When you make a matrix with the matrix maker, you go with an alignerframe

{alignerframe[1], alignerframe[2]},

stretch factors

xstretch and ystretch

and a hangerframe

{hangerframe[1], hangerframe[2]}

and you put

A = hanger.stretcher.aligner.

The matrix A has the properties:

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

and

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

Hit both sides of

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

with A^{-1} to get

A^{-1}. A.alignerframe[1]= A^{-1}.(xstretch hangerframe[1]) = xstretch A^{-1}. hangerframe[1]

Reason: This comes from the linearity of matrix hits.
In particular:

A^{-1}.(xstretch hangerframe[1]) = xstretch A^{-1}. hangerframe[1]

This is the same as:

alignerframe[1]= xstretch A^{-1}. hangerframe[1]

Reason: Hitting with the inverse of A undoes everything a hit with A does.

And this is the same as

A^{-1}.hangerframe[1] = $\frac{1}{\text{xstretch}}$ alignerframe[1].

By the same steps of reasoning, you get

A^{-1}.hangerframe[2] = $\frac{1}{\text{ystretch}}$ alignerframe[2]

These two equalities tell you that to make A^{-1} with the matrix maker ingredients, you can go with these choices:

Aligner frame for A^{-1} is the hanger frame of A.

Hanger frame for A^{-1} is the aligner frame of A.

Stretch factors for A^{-1} are inverses of the stretch factors of A.

That's why everything works.

□B.3.b) If A = hanger.stretcher.aligner, then A^t = aligner^t.stretcher. hanger^t.

A and A^t have the same stretch factors.

The aligner frame for A^t is the hanger frame for A.

The hanger frame for A^t is the aligner frame for A

Here's a matrix A made with matrix maker ingredients using
a random aligner frame,
two random stretch factors
and
a random hanger frame:

```
Clear[alignerframe];
s = Random[Real, {-π/2, π/2}];

{alignerframe[1], alignerframe[2]} = N[{{Cos[s], Sin[s]},
    (-1)^Random[Integer,{0,1}] {Cos[s + π/2], Sin[s + π/2]}}];

aligner = {alignerframe[1], alignerframe[2]};

{xstretch, ystretch} =
  {Random[Real, {0.1, 3.0}], Random[Real, {0.1, 3.0}]};
stretcher = ( xstretch   0
              0          ystretch );
Clear[hangerframe];
s = Random[Real, {-π/2, π/2}];

{hangerframe[1], hangerframe[2]} = N[{{Cos[s], Sin[s]},
    (-1)^Random[Integer,{0,1}] {Cos[s + π/2], Sin[s + π/2]}}];

hanger = Transpose[{hangerframe[1], hangerframe[2]}];
A = hanger.stretcher.aligner;

MatrixForm[A]
```

$$\begin{pmatrix} -0.403014 & -1.73912 \\ -2.14222 & 0.771153 \end{pmatrix}$$

According to one of the problems above, you can make A^{-1}, the inverse of A, by
-> Using the hanger frame of A as the aligner frame of A^{-1} ;
-> Using the aligner frame of A as the hanger frame of A^{-1} ;
-> And inverting the stretch factors of A:

```
alignerforinverse = {hangerframe[1], hangerframe[2]};
unstretcher = {{1/xstretch, 0}, {0, 1/ystretch}};
hangerforinverse = Transpose[{alignerframe[1], alignerframe[2]}];
Ainverse = hangerforinverse.unstretcher.alignerforinverse;
MatrixForm[Ainverse]
```

$$\begin{pmatrix} -0.191052 & -0.430863 \\ -0.530731 & 0.0998459 \end{pmatrix}$$

Compare:

```
MatrixForm[Inverse[A]]
```

$$\begin{pmatrix} -0.191052 & -0.430863 \\ -0.530731 & 0.0998459 \end{pmatrix}$$

Modify this to make A^t.

□ **Answer:**

You can make A^t, the transpose of A, by

-> Using the hanger frame of A as the aligner frame of A^t ;

-> Using the aligner frame of A as the hanger frame of A^t ;

-> And going with the **same stretch factors** as you used for A:

```
alignerfortranspose = {hangerframe[1], hangerframe[2]};
stretcher = {{xstretch, 0}, {0, ystretch}};
hangerfortranspose =
  Transpose[{alignerframe[1], alignerframe[2]}];
Atranspose = hangerfortranspose.stretcher.alignerfortranspose;
MatrixForm[Atranspose]
```

$$\begin{pmatrix} -0.403014 & -2.14222 \\ -1.73912 & 0.771153 \end{pmatrix}$$

Compare:

```
MatrixForm[Transpose[A]]
```

$$\begin{pmatrix} -0.403014 & -2.14222 \\ -1.73912 & 0.771153 \end{pmatrix}$$

The upshots:

- If A = hanger.stretcher.aligner, then A^t = aligner^t.stretcher. hanger^t.
- A and A^t have the same stretch factors.
- The aligner frame for A^t is the hanger frame for A.
- The hanger frame for A^t is the aligner frame for A.

B.4) Using a zero stretch factor to make matrices that cannot be inverted. Why these matrices deposit all their hits on a line and why they hit infinitely many points into one single point.

□ B.4.a.i) To make a noninvertible matrix, you go with a stretch factor = 0

Use the matrix maker ingredients to make a 2D matrix that cannot be inverted.

□ **Answer:**

Go with any aligner frame and any hanger frame you like, but make xstretch or ystretch = 0.

Here's one:

```
Clear[alignerframe];
s = π/3;

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

aligner = {alignerframe[1], alignerframe[2]};

{xstretch, ystretch} = {2, 0};

stretcher = DiagonalMatrix[{xstretch, ystretch}];

Clear[hangerframe];
s = π/3;
{hangerframe[1], hangerframe[2]} =
  N[{{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}}];

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

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

$$\begin{pmatrix} 0.5 & 0.866025 \\ 0.866025 & 1.5 \end{pmatrix}$$

Try to invert A:

```
Inverse[A]
```

```
Inverse::luc : Result for Inverse of badly conditioned matrix
  {{0.5, 0.866025}, {0.866025, 1.5}} may contain significant numerical errors.
```

$$\{\{4.51809 \times 10^{16}, -2.60852 \times 10^{16}\}, \{-2.60852 \times 10^{16}, 1.50603 \times 10^{16}\}\}$$

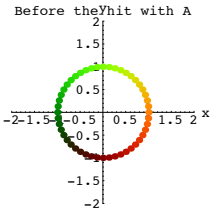
That's garbage resulting from roundoff errors.

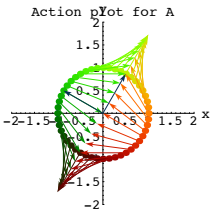
This matrix A is not invertible.

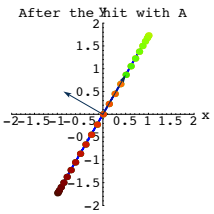
To get an idea why this A is not invertible, look at this action movie:

```
Clear[x, y, t, hitplotter,
  hitpointplotter, pointcolor, actionarrows, matrix2D];
{tlow, thigh} = {0, 2 π};
ranger = Max[{xstretch, ystretch, 1.2}];
{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)/48;

hitplotter[matrix2D_] := ParametricPlot[matrix2D.{x[t], y[t]},
   {t, tlow, thigh}, PlotStyle → {{Thickness[0.01], Blue}},
   PlotRange → {{-ranger, ranger}, {-ranger, ranger}},
   AxesLabel → {"x", "y"}, DisplayFunction → Identity];

hitpointplotter[matrix2D_] :=
  Table[Graphics[{pointcolor[t], PointSize[0.035],
     Point[matrix2D.{x[t], y[t]}]}], {t, tlow, thigh - jump, jump}];

actionarrows[matrix2D_] :=
  Table[Arrow[matrix2D.{x[t], y[t]} - {x[t], y[t]}, Tail → {x[t], y[t]},
    VectorColor → pointcolor[t]], {t, tlow, thigh - jump, jump}];

hangerframeplot =
  {Arrow[hangerframe[1], Tail → {0, 0}, VectorColor -> Indigo],
   Arrow[hangerframe[2], Tail → {0, 0}, VectorColor -> Indigo]};

Abefore = Show[hitplotter[IdentityMatrix[2]],
   hitpointplotter[IdentityMatrix[2]], PlotLabel →
    "Before the hit with A", DisplayFunction → $DisplayFunction];

Aaction = Show[Abefore, actionarrows[A],
   hangerframeplot, PlotLabel → "Action plot for A",
   DisplayFunction → $DisplayFunction];
```

```
Aafter = Show[hitplotter[A], hitpointplotter[A],
   hangerframeplot, PlotLabel → "After the hit with A",
   DisplayFunction → $DisplayFunction];
```







That's hangerframe[1] pointing in the direction of the line.
And perpendicular to the line is hangerframe[2].

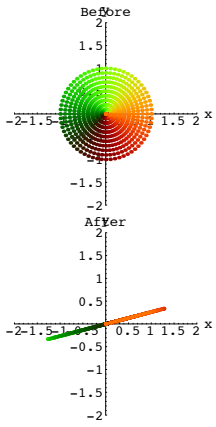
Grab and animate at various speeds paying special attention to the fact that the hit with A sends two different points on the circle onto the same point on the line.

Once these two hit points have been deposited on the same point on the line, it's impossible to tell which part of the circle they came from. That's why it's impossible to undo a hit by this matrix.

And that's how you knew in advance that the 2D matrix A is not invertible.

Watch other random noninvertible matrices smash bunches of points onto lines:

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

{xstretch, ystretch} = {Random[Real, {1, 2}], 0};
stretcher = DiagonalMatrix[{xstretch, ystretch}];

Clear[hangerframe];
s = Random[Real, N[{-π/2, π/2}]];
{hangerframe[1], hangerframe[2]} =
  N[{{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}}];
hanger = Transpose[{hangerframe[1], hangerframe[2]}];

A = hanger.stretcher.aligner;

Clear[x, y, t, r, s];
ranger = xstretch;
{x[t_], y[t_]} = {Cos[t], Sin[t]};

Clear[hitpointplotter, pointcolor, matrix2D, r];
pointcolor[t_] = RGBColor[0.5 (Cos[t] + 1), 0.5 (Sin[t] + 1), 0];
tjump = (thigh - tlow)/64;
rjump = 0.1;

hitpointplotter[matrix2D_] :=
  Table[Graphics[{pointcolor[t], PointSize[0.02],
     Point[matrix2D.{r x[t], r y[t]}]}],
   {t, tlow, thigh - tjump, tjump}, {r, 0, 1, rjump}];

ranger = 2;
before = Show[hitpointplotter[IdentityMatrix[2]],
   PlotLabel → "Before", Axes → True, AxesLabel → {"x", "y"},
   PlotRange → {{-ranger, ranger}, {-ranger, ranger}},
   DisplayFunction → $DisplayFunction];

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



Grab and animate.

□B.4.b.ii) When you go with a zero stretch factor, you get a noninvertible matrix which deposits its hits on a line determined by a hanger frame vector

Part of the job of math is to explain why things work out the way they do.
Explain this:
When you make a matrix A with ystretch = 0, then all hits with A land on the line through {0,0} determined by hangerframe[1].

□Answer:

Take any point and resolve it into perpendicular components in the directions of the alignerframe vectors to get

point = (point.alignerframe[1]) **alignerframe[1] +** (point.alignerframe[2]) **alignerframe[2]**.

This gives, thanks to linearity of matrix hits,

A.point = (point.alignerframe[1]) **A.alignerframe[1] +** (point.alignerframe[2]) **A.alignerframe[2]**.

This is the same as

A.point = (point.alignerframe[1]) **xstretch hangerframe[1] +**

(point.alignerframe[2]) **ystretch hangerframe[2]**

Reason:
A.alignerframe[1] = xstretch hangerframe[1] and A.alignerframe[2] = ystretch hangerframe[2]

And now because ystretch = 0, the second term on the right drops out leaving you with

A.point = (point.alignerframe[1]) **xstretch hangerframe[1]**

This tells you that all hits with A are multiples of hangerframe[1].

This puts all hits with A on the line through {0,0} determined by hangerframe[1].

□B.4.b.iii) When you go with a zero stretch factor, you get a noninvertible matrix because it hits infinitely many points into one point

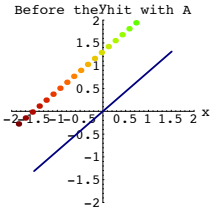
Here is a matrix A which is random in most respects except that ystretch = 0:

```
Clear[alignerframe];
s = Random[Real, N[{-π/2, -π/6}]];
{alignerframe[1], alignerframe[2]} =
  N[{{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}}];
aligner = {alignerframe[1], alignerframe[2]};

{xstretch, ystretch} = {Random[Real, {0.5, 1.5}], 0};
stretcher = DiagonalMatrix[{xstretch, ystretch}];

Clear[hangerframe];
s = Random[Real, N[{π/6, π/4}]];
{hangerframe[1], hangerframe[2]} =
  N[{{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}}];
hanger = Transpose[{hangerframe[1], hangerframe[2]}];

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

$$\begin{pmatrix} 0.380307 & -0.441543 \\ 0.330449 & -0.383656 \end{pmatrix}$$

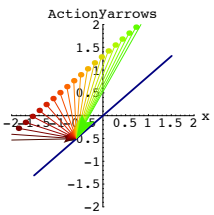
Here are a bunch of points on the line running through a random point in the direction of alignerframe[2] shown with the line running in the direction of hangerframe[1]:

```
point = {Random[Real, {-1, 1}], Random[Real, {0.5, 1}]};
{tlow, thigh} = {-2, 2};

pointcolor[t_] = RGBColor[0.5 (Cos[t] + 1), 0.5 (Sin[t] + 1), 0];
tjump = 0.2;
alignerframe2line =
 Table[Graphics[{pointcolor[t], PointSize[0.03],
    Point[point + t alignerframe[2]]}], {t, tlow, thigh, tjump}];
hangerframe1line = ParametricPlot[t hangerframe[1],
  {t, tlow, thigh}, PlotStyle -> {{Thickness[0.01], NavyBlue}},
  AxesLabel -> {"x", "y"}, DisplayFunction -> Identity];
before = Show[hangerframe1line, alignerframe2line,
   PlotRange -> {{-2, 2}, {-2, 2}}, DisplayFunction ->
    $DisplayFunction, PlotLabel -> "Before the hit with A"];
```



The points are on the line through the random point running in the direction of alignerframe[2]
The solid line is the line through {0,0} running in the direction of hangerframe[2]

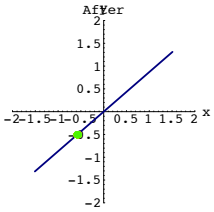
```
actionarrows = Table[
  Arrow[A.(point + t alignerframe[2]) - (point + t alignerframe[2]),
   Tail → (point + t alignerframe[2]), VectorColor → pointcolor[t]],
  {t, tlow, thigh, tjump}];

action = Show[hangerframe1line, alignerframe2line, actionarrows,
   PlotRange -> {{-2, 2}, {-2, 2}}, DisplayFunction ->
    $DisplayFunction, PlotLabel -> "Action arrows"];
```



Here's what you get after you hit all those points with A:

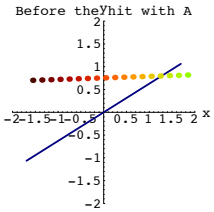
```
hitalignerframe2line =
 Table[Graphics[{pointcolor[t], PointSize[0.04], Point[
     A.(point + t alignerframe[2])]}], {t, tlow, thigh, tjump}];
```

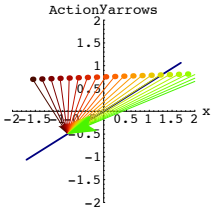
```
Show[hangerframe1line,
  hitalignerframe2line, PlotRange -> {{-2, 2}, {-2, 2}},
  DisplayFunction -> $DisplayFunction, PlotLabel -> "After"];
```

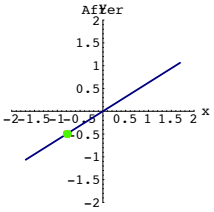


Do some more:

```
Clear[alignerframe];
s = Random[Real, N[{-π/2, -π/6}]];
{alignerframe[1], alignerframe[2]} =
  N[{{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}}];
aligner = {alignerframe[1], alignerframe[2]};

{xstretch, ystretch} = {Random[Real, {0.5, 1.5}], 0};
stretcher = DiagonalMatrix[{xstretch, ystretch}];

Clear[hangerframe];
s = Random[Real, N[{π/6, π/4}]];
{hangerframe[1], hangerframe[2]} =
  N[{{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}}];
hanger = Transpose[{hangerframe[1], hangerframe[2]}];

A = hanger.stretcher.aligner;


point = {Random[Real, {-1, 1}], Random[Real, {0.5, 1}]};
{tlow, thigh} = {-2, 2};
pointcolor[t_] = RGBColor[0.5 (Cos[t] + 1), 0.5 (Sin[t] + 1), 0];
tjump = 0.2;
alignerframe2line =
 Table[Graphics[{pointcolor[t], PointSize[0.03],
    Point[point + t alignerframe[2]]}], {t, tlow, thigh, tjump}];
hangerframe1line = ParametricPlot[t hangerframe[1],
  {t, tlow, thigh}, PlotStyle -> {{Thickness[0.01], NavyBlue}},
  AxesLabel -> {"x", "y"}, DisplayFunction -> Identity];
before = Show[hangerframe1line, alignerframe2line,
   PlotRange -> {{-2, 2}, {-2, 2}}, DisplayFunction ->
    $DisplayFunction, PlotLabel -> "Before the hit with A"];
actionarrows = Table[Arrow[A.(point + t alignerframe[2]) -
    (point + t alignerframe[2]), Tail → (point + t alignerframe[2]),
   VectorColor → pointcolor[t]], {t, tlow, thigh, tjump}];

action = Show[hangerframe1line, alignerframe2line, actionarrows,
   PlotRange -> {{-2, 2}, {-2, 2}}, DisplayFunction ->
    $DisplayFunction, PlotLabel -> "Action arrows"];

hitalignerframe2line =
 Table[Graphics[{pointcolor[t], PointSize[0.04], Point[
     A.(point + t alignerframe[2])]}], {t, tlow, thigh, tjump}];

Show[hangerframe1line,
  hitalignerframe2line, PlotRange -> {{-2, 2}, {-2, 2}},
  DisplayFunction -> $DisplayFunction, PlotLabel -> "After"];
"This noninvertible matrix is" MatrixForm[A]
```







This noninvertible matrix is $\begin{pmatrix} 0.0357649 & -1.05015 \\ 0.0224573 & -0.659407 \end{pmatrix}$

Grab,animate and then rerun several times

The hit with A sent all those points into the **same point**.

- Explain why it works out this way for any matrix made with ystretch = 0.
- And then explain why any matrix made with ystretch = 0 cannot be invertible.

Answer:

In the last part you saw that if ystretch = 0, then

A.point = (point.alignerframe[1]) **xstretch hangerframe[1]**

If you need to see the reason again, click on the right.

Take any point and resolve it into perpendicular components in the directions of the alignerframe vectors to get

point = (point.alignerframe[1]) **alignerframe[1] +** (point.alignerframe[2]) **alignerframe[2]**.

This gives, thanks to linearity of matrix hits,

A.point = (point.alignerframe[1]) **A.alignerframe[1] +** (point.alignerframe[2]) **A.alignerframe[2]**.

This is the same as

A.point = (point . alignerframe[1]) **xstretch hangerframe[1] +**

(point . alignerframe[2]) **ystretch hangerframe[2]**

Reason:
A.alignerframe[1] = xstretch hangerframe[1] and A.alignerframe[2] = ystretch hangerframe[2]

And now because ystretch = 0, the second term on the right drops out leaving you with

A.point = (point.alignerframe[1]) **xstretch hangerframe[1]**

When you put

point = randompoint + t alignerframe[2]

you get

A.(randompoint + t alignerframe[2])

= ((randompoint + t alignerframe[2]).alignerframe[1]) **xstretch hangerframe[1]**

= (randompoint.alignerframe[1] + t alignerframe[2].alignerframe[1]) **xstretch hangerframe[1]**

= (randompoint.alignerframe[1] + t 0) **xstretch hangerframe[1]**

Reason:
alignerframe[1] and alignerframe[2] are perpendicular.
So alignerframe[2].alignerframe[1] = 0.

= (randompoint.alignerframe[1]) **xstretch hangerframe[1].**

The upshot:

When you hit A at any point on the line defined by

randompoint + t alignerframe[2],

you get the same point- namely

(randompoint.alignerframe[1]) **xstretch hangerframe[1].**

This tells you that hits with A send infinitely many points into the same point.

Once these infinitely many hit points have been deposited on the same point, it's impossible to tell where they came from.

That's why it's impossible to undo a hit by this matrix. And that's why any matrix made with ystretch = 0 cannot be invertible.