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

GIVE IT A TRY!

G.1) The roles of the hanger frame, the stretch factors and the aligner frame*

If A = hanger.stretcher.aligner, then

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

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

and

$$\mathbf{A.}\left(\frac{s}{xstretch}\ \text{alignerframe[1]} + \frac{t}{ystretch}\ \text{alignerframe[2]}\right) = s\ \text{hangerframe[1]} + t\ \text{hangerframe[2]}$$

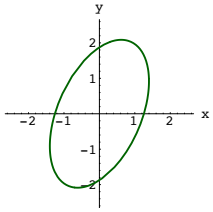
□G.1.a.i) The hanger frame and the ellipse

Here's a random 2D matrix A made by matrix maker ingredients together with a plot of ellipse you get when you hit the unit circle centered on {0,0} with A:

```
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, {2.2, 2.7}], Random[Real, {0.6, 1.6}]};
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]
```

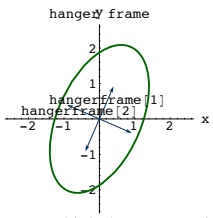
```
Clear[x, y, t];
{x[t_], y[t_]} = {Cos[t], Sin[t]};
{tlow, thigh} = {0, 2 π};
ranger = 1.2 Max[{xstretch, ystretch, 1}];
ellipseplot = ParametricPlot[A.{x[t], y[t]},
  {t, tlow, thigh}, PlotStyle → {{Thickness[0.01], GosiaGreen}},
  PlotRange → {{-ranger, ranger}, {-ranger, ranger}},
  AxesLabel → {"x", "y"}];
```

```
( 0.301369  -1.36594
  2.02835   -0.491183 )
```



Look at this plot of the hanger frame and its negatives:

```
Show[ellipseplot,
  Arrow[hangerframe[1], Tail → {0, 0}, VectorColor → Indigo],
  Arrow[-hangerframe[1], Tail → {0, 0}, VectorColor → Indigo],
  Arrow[hangerframe[2], Tail → {0, 0}, VectorColor → Indigo],
  Arrow[-hangerframe[2], Tail → {0, 0}, VectorColor → Indigo],
  Graphics[Text["hangerframe[1]", 0.6 hangerframe[1]]],
  Graphics[Text["hangerframe[2]", 0.6 hangerframe[2]]],
  Axes → True, AxesLabel → {"x", "y"},
  PlotRange → {{-ranger, ranger}, {-ranger, ranger}},
  PlotLabel → "hanger frame"];
```

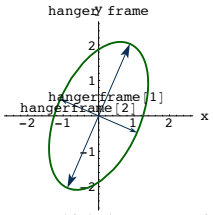


Do you think the outcome is an accident?
If not explain why you see what you see.

□G.1.a.ii) Multiplying the hangerframe vectors by the stretch factors

Stay with the same set up as in part i) and look at this plot in which the unit hanger frame vectors have been multiplied by their associated stretch factors:

```
lastplot = Show[ellipseplot,
  Arrow[xstretch hangerframe[1], Tail → {0, 0}, VectorColor → Indigo],
  Arrow[-xstretch hangerframe[1],
   Tail → {0, 0}, VectorColor → Indigo],
  Arrow[ystretch hangerframe[2], Tail → {0, 0}, VectorColor → Indigo],
  Arrow[-ystretch hangerframe[2],
   Tail → {0, 0}, VectorColor → Indigo],
  Graphics[Text["hangerframe[1]", 0.6 hangerframe[1]]],
  Graphics[Text["hangerframe[2]", 0.6 hangerframe[2]]],
  Axes → True, AxesLabel → {"x", "y"},
  PlotRange → {{-ranger, ranger}, {-ranger, ranger}},
  PlotLabel → "hanger frame"];
```

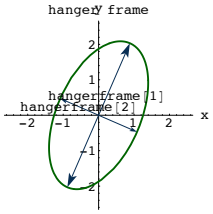


Do you think the outcome is an accident?
If not explain why you see what you see.

□G.1.a.iii) Measurements on the ellipse

Here's another look at the plot in part ii) above:

```
Show[lastplot];
```



You have enough information to measure
→ the length of the long axis of this ellipse
→ the length of the short axis of this ellipse

→ The area enclosed by this ellipse.
Do it.

□G.1.b) If A = hanger.stretcher.aligner, then

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

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

$$\mathbf{A.}\left(\frac{s}{xstretch}\ \text{alignerframe[1]} + \frac{t}{ystretch}\ \text{alignerframe[2]}\right) = s\ \text{hangerframe[1]} + t\ \text{hangerframe[2]}$$

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

```
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]
```

```
( 0.278261   1.67835
  -1.80064   0.621189 )
```

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.
Put answer here.

Stay with the last matrix A and look at these two calculations involving two cleared numbers s and t:

```
Clear[s, t];
Expand[A.(s/xstretch alignerframe[1] + t/ystretch alignerframe[2])]
```

```
{0.88345 s + 0.468526 t, -0.468526 s + 0.88345 t}
```

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

```
{0.88345 s + 0.468526 t, -0.468526 s + 0.88345 t}
```

Explain why that happened.
Put answer here.

□G.1.c.i) Variable hangers change the tilt but not the shape

Here's a matrix function A[s]:

```
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, {2.2, 2.7}], Random[Real, {0.6, 1.6}]};
stretcher = ((xstretch 0) (0 ystretch));
Clear[hangerframe, s, hanger];
{hangerframe[1], hangerframe[2]} =
  N[{{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}}];
hanger[s_] = Transpose[{hangerframe[1], hangerframe[2]}];
Clear[A];
A[s_] = hanger[s].stretcher.aligner;
MatrixForm[A[s]]
```

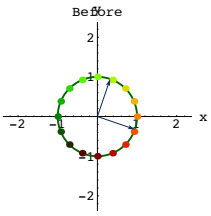
$$\begin{pmatrix} 2.24883\cos[s] - 0.318917\sin[s] & -0.765166\cos[s] - 0.937302\sin[s] \\ 0.318917\cos[s] + 2.24883\sin[s] & 0.937302\cos[s] - 0.765166\sin[s] \end{pmatrix}$$

When s changes, the hanger frame for A[s]
changes but the stretch factors and the aligner frame do not change.

Here's the unit circle:

```
Clear[x, y, t, s];
{tlow, thigh} = {0, 2 π};
ranger = Max[{xstretch, ystretch, 1.2}];
{x[t_], y[t_]} = {Cos[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)/16;
```

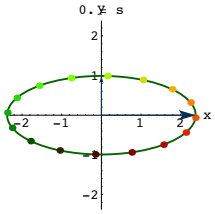
```
hitplotter[matrix2D_] := ParametricPlot[matrix2D.{x[t], y[t]},
   {t, tlow, thigh}, PlotStyle → {{Thickness[0.01], GosiaGreen}},
   PlotRange → {{-ranger, ranger}, {-ranger, ranger}},
   AxesLabel → {"x", "y"}, DisplayFunction → Identity];
hitpointplotter[matrix2D_] :=
  Table[Graphics[{pointcolor[t], PointSize[0.035],
     Point[matrix2D.{x[t], y[t]}]}], {t, tlow, thigh - jump, jump}];
hitframeplotter[matrix2D_] := Table[Arrow[matrix2D.alignerframe[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}];
before = Show[hitplotter[IdentityMatrix[2]], hitpointplotter[
    IdentityMatrix[2]], hitframeplotter[IdentityMatrix[2]],
   PlotLabel → "Before", DisplayFunction → $DisplayFunction];
```

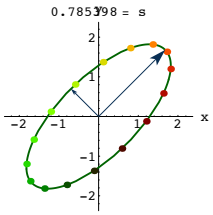


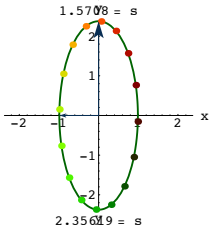
That's the aligner frame for A sitting inside the unit circle.

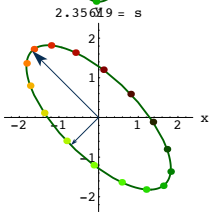
Here's a table showing what happens when you hit this circle with A[s] with s running from 0 to 2π in increments of $\frac{\pi}{4}$:

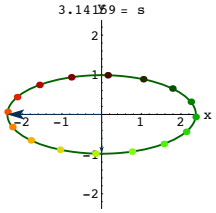
```
sjump = π/4;
Table[Show[hitplotter[A[s]], hitpointplotter[A[s]],
   hitframeplotter[A[s]], PlotLabel → N[s] "= s",
   DisplayFunction → $DisplayFunction], {s, 0, 2 π - sjump, sjump}];
```

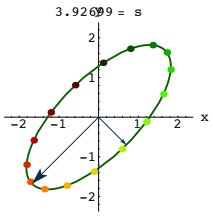


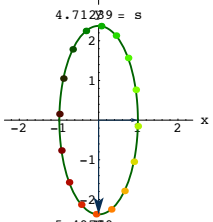


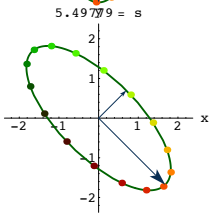












Grab and animate.

Your job is to analyze the code that went into defining A[s] and then explain why the plots came out the way they did.

□G.1.b.ii) Variable aligners do not change the tilt or the shape

Here's a new matrix function A[s]:

```
Clear[s, alignerframe, aligner];
{alignerframe[1], alignerframe[2]} =
  N[{{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}}];
aligner[s_] := {alignerframe[1], alignerframe[2]};
{xstretch, ystretch} =
  {Random[Real, {2.2, 2.7}], Random[Real, {0.6, 1.6}]};
stretcher = ((xstretch 0) (0 ystretch));
Clear[hangerframe, hanger];
```

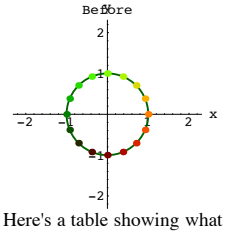
```
t = Random[Real, {0.5, 1.5}];
{hangerframe[1], hangerframe[2]} =
 N[{{Cos[t], Sin[t]}, {Cos[t + π/2], Sin[t + π/2]}}];
hanger = Transpose[{hangerframe[1], hangerframe[2]}];
Clear[A];
A[s_] = hanger.stretcher.aligner[s];
MatrixForm[A[s]]
```

$$\begin{pmatrix} 1.64984\,\text{Cos}[s] + 0.568833\,\text{Sin}[s] & -0.568833\,\text{Cos}[s] + 1.64984\,\text{Sin}[s] \\ 1.61527\,\text{Cos}[s] - 0.581007\,\text{Sin}[s] & 0.581007\,\text{Cos}[s] + 1.61527\,\text{Sin}[s] \end{pmatrix}$$

When s changes, the aligner frame for A[s]
changes but the stretch factors and the hanger frame do not change.

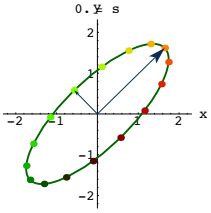
Here's the unit circle:

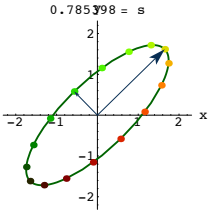
```
ranger = Max[{xstretch, ystretch, 1.2}];
Show[hitplotter[IdentityMatrix[2]],
  hitpointplotter[IdentityMatrix[2]], PlotLabel → "Before",
  DisplayFunction → $DisplayFunction];
```

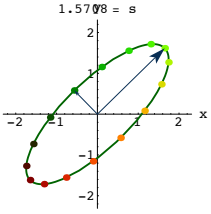


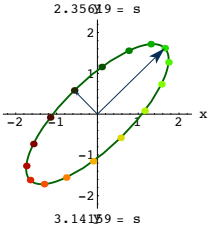
Here's a table showing what happens when you hit this circle with A[s] with s running from 0 to 2π in increments of $\frac{\pi}{4}$:

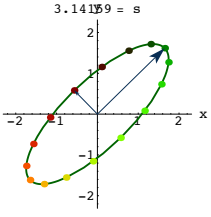
```
sjump = π/4;
Table[Show[hitplotter[A[s]], hitpointplotter[A[s]],
   hitframeplotter[A[s]], PlotLabel → N[s] "= s",
   DisplayFunction → $DisplayFunction], {s, 0, 2 π - sjump, sjump}];
```

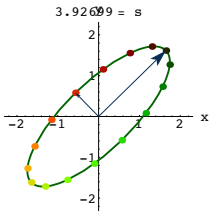


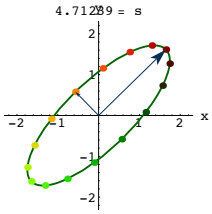


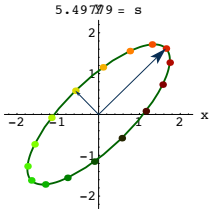












Grab and animate and watch the color-coded points dance around the ellipse.

Your job is to analyze the code that went into defining A[s] and then explain why those points danced around the same physical ellipse.

□G.1.b.iii) Variable stretchers do not change the tilt or the shape, but they do change the size

Here's a new matrix function A[s]:

```
Clear[alignerframe, aligner];
s = Random[Real, {0.5, 1.5}];
{alignerframe[1], alignerframe[2]} =
  N[{{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}}];
aligner = {alignerframe[1], alignerframe[2]};
Clear[s, stretcher, xstretch, ystretch];
{xstretch[s_], ystretch[s_]} =
  s {Random[Real, {2.2, 2.7}], Random[Real, {0.6, 1.6}]};
```

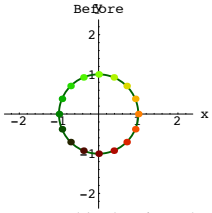
```
stretcher[s_] = ( xstretch[s]      0
                  0           ystretch[s] );
Clear[hangerframe];
t = Random[Real, {0.5, 1.5}];
{hangerframe[1], hangerframe[2]} =
  N[{{Cos[t], Sin[t]}, {Cos[t + π/2], Sin[t + π/2]}}];
hanger = Transpose[{hangerframe[1], hangerframe[2]}];
Clear[A];
A[s_] = hanger.stretcher[s].aligner;
MatrixForm[A[s]]
```

$$\begin{pmatrix} 1.59836\,s & -0.261683\,s \\ 0.706635\,s & 2.24695\,s \end{pmatrix}$$

When s changes, the stretch factors for A[s]
change but the hanger frame and the aligner frame do not change.

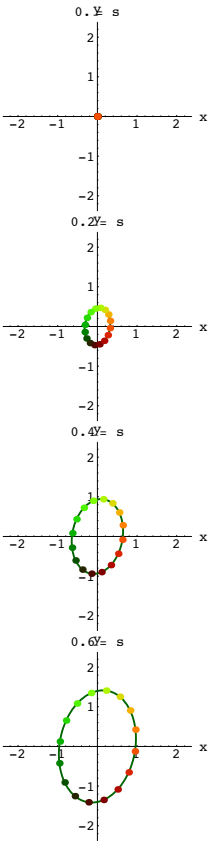
Here's the unit circle:

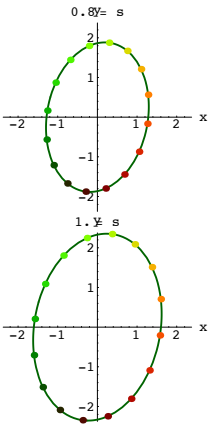
```
ranger = Max[{xstretch[1], ystretch[1], 1.2}];
Show[hitplotter[IdentityMatrix[2]],
  hitpointplotter[IdentityMatrix[2]], PlotLabel → "Before",
  DisplayFunction → $DisplayFunction];
```



Here's a table showing what happens when you hit this circle with A[s] with s running from 0 to 1 in increments of 0.2:

```
Table[Show[hitplotter[A[s]],
   hitpointplotter[A[s]], PlotLabel → N[s] "= s",
   DisplayFunction → $DisplayFunction], {s, 0, 1, 0.2}];
```





Grab and animate.

Your job is to analyze the code that went into defining A[s] and then explain why the plots came out the way they did.

□G.1.c) A special ellipse

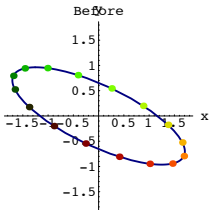
Here's a random matrix made with the matrix maker ingredients:

```
Clear[alignerframe];
s = Random[Real, {-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, {0.5, 1.0}], Random[Real, {1.2, 2.0}]};
stretcher = DiagonalMatrix[{xstretch, ystretch}];
Clear[hangerframe];
t = Random[Real, {-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;
MatrixForm[A]
```

$$\begin{pmatrix} 0.739528 & 0.432386 \\ 0.432386 & 1.45506 \end{pmatrix}$$

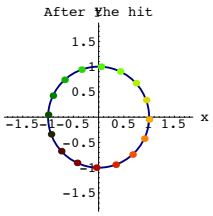
Here's a very special ellipse parametrized by
$\{x[t], y[t]\} = \frac{\text{Cos[t] alignerframe[1]}}{\text{xstretch}} + \frac{\text{Sin[t] alignerframe[2]}}{\text{ystretch}}$:

```
Clear[x, y, t];
a = 1/xstretch;
b = 1/ystretch;
{x[t_], y[t_]} = a Cos[t] alignerframe[1] + b Sin[t] alignerframe[2];
{tlow, thigh} = {0, 2 π};
Clear[hitplotter, hitpointplotter,
  pointcolor, actionarrows, matrix2D];
ranger = Max[{xstretch, ystretch, 1/xstretch, 1/ystretch, 1}];
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}];
before = Show[hitplotter[IdentityMatrix[2]],
   hitpointplotter[IdentityMatrix[2]], PlotLabel → "Before",
   DisplayFunction → $DisplayFunction];
```



Here's what happens when you hit this ellipse with A:

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



This curve is the circle of radius 1 centered at {0,0}.
Is this just an accident?
If it is not an accident, then explain why it is not an accident and why it will happen any time you use the matrix maker (going with positive stretch factors).

For a heavy tip, click on the right.

The ellipse is given by:
$\{x[t], y[t]\} = \frac{\text{Cos[t] alignerframe[1]}}{\text{xstretch}} + \frac{\text{Sin[t] alignerframe[2]}}{\text{ystretch}}$;
By linearity of matrix hits,
$A.\{x[t], y[t]\} = \frac{\text{Cos[t] A. alignerframe[1]}}{\text{xstretch}} + \frac{\text{Sin[t] A.alignerframe[2]}}{\text{ystretch}}$.
Now remember:

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

True

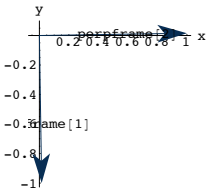
```
A.alignerframe[2] == ystretch hangerframe[2]
```

True

□G.1.d) Calculus Cal is clueless

That stupid lab pest Calculus Cal is going with a random perpendicular frame:

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


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

Show[frameplot, Axes -> True, AxesLabel -> {"x", "y"}];
```



Then Cal goes with xstretch = 1 and ystretch = 1 and he uses this plotted perpendicular frame for his aligner frame and his hanger frame and makes the resulting matrix

A = hanger.stretcher.aligner

and gets:

```
Clear[alignerframe];

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

stretcher = {{1, 0}, {0, 1}};

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

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

$$\begin{pmatrix} 1. & 0 \\ 0 & 1. \end{pmatrix}$$

Cal is baffled.

He says:" There must be some mistake. No matter what random perpendicular frame I use, the machine gives me the identity matrix:"

What do you say to Cal?

G.2) Reset a parameter

□G.2.a.i) Reset the ystretch factor so that the resulting matrix is not invertible

Here's a matrix A made with matrix maker ingredients:

```
Clear[alignerframe];
s = 0.2 Pi;
```

```
{alignerframe[1], alignerframe[2]} =
     N[{{Cos[s], Sin[s]}, {Cos[s + Pi / 2], Sin[s + Pi / 2]}}];
aligner = {alignerframe[1], alignerframe[2]};

{xstretch, ystretch} = {8.4, 1.3};
stretcher = {{xstretch, 0}, {0, ystretch}};
Clear[hangerframe];
s = - 0.4 Pi;
{hangerframe[1], hangerframe[2]} =
     N[{{Cos[s], Sin[s]}, {Cos[s + Pi / 2], Sin[s + Pi / 2]}}];
hanger = Transpose[{hangerframe[1], hangerframe[2]}];
A = hanger.stretcher.aligner;
MatrixForm[A]
```

$$\begin{pmatrix} 1.37328 & 2.52599 \\ -6.69926 & -4.37074 \end{pmatrix}$$

Reset the ystretch factor so that the resulting matrix is not invertible.

□G.2.a.ii) Reset the hangerframe so that the resulting matrix is positive definite (a frame stretcher)

Here's the same matrix as in part i) above:

```
Clear[alignerframe];
s = 0.2 Pi;
{alignerframe[1], alignerframe[2]} =
     N[{{Cos[s], Sin[s]}, {Cos[s + Pi / 2], Sin[s + Pi / 2]}}];
aligner = {alignerframe[1], alignerframe[2]};

{xstretch, ystretch} = {8.4, 1.3};
stretcher = {{xstretch, 0}, {0, ystretch}};
Clear[hangerframe];
s = - 0.4 Pi;
{hangerframe[1], hangerframe[2]} =
     N[{{Cos[s], Sin[s]}, {Cos[s + Pi / 2], Sin[s + Pi / 2]}}];
hanger = Transpose[{hangerframe[1], hangerframe[2]}];
A = hanger.stretcher.aligner;
MatrixForm[A]
```

$$\begin{pmatrix} 1.37328 & 2.52599 \\ -6.69926 & -4.37074 \end{pmatrix}$$

Reset the hangerframe so that the resulting matrix is positive definite (a frame stretcher).

□G.2.a.iii) Reset xstretch, ystretch and the hangerframe so that the resulting matrix is a frame flipper (reflection matrix).

Here's the same matrix as in part i) above:

```
Clear[alignerframe];
s = 0.2 Pi;
{alignerframe[1], alignerframe[2]} =
     N[{{Cos[s], Sin[s]}, {Cos[s + Pi / 2], Sin[s + Pi / 2]}}];
aligner = {alignerframe[1], alignerframe[2]};

{xstretch, ystretch} = {8.4, 1.3};
stretcher = {{xstretch, 0}, {0, ystretch}};
Clear[hangerframe];
s = - 0.4 Pi;
{hangerframe[1], hangerframe[2]} =
     N[{{Cos[s], Sin[s]}, {Cos[s + Pi / 2], Sin[s + Pi / 2]}}];
hanger = Transpose[{hangerframe[1], hangerframe[2]}];
A = hanger.stretcher.aligner;
MatrixForm[A]
```

$$\begin{pmatrix} 1.37328 & 2.52599 \\ -6.69926 & -4.37074 \end{pmatrix}$$

Reset xstretch, ystretch and the hangerframe so that the resulting matrix is a frame flipper (reflection matrix).

□G.2.a.iv) Reset the xstretch factor, the ystretch factor and both the aligner frame and the hanger frame so that the resulting matrix is a rotation matrix

Here's the same matrix as in part i) above:

```
Clear[alignerframe];
s = 0.2 Pi;
{alignerframe[1], alignerframe[2]} =
     N[{{Cos[s], Sin[s]}, {Cos[s + Pi / 2], Sin[s + Pi / 2]}}];
aligner = {alignerframe[1], alignerframe[2]};

{xstretch, ystretch} = {8.4, 1.3};
stretcher = {{xstretch, 0}, {0, ystretch}};
Clear[hangerframe];
s = - 0.4 Pi;
{hangerframe[1], hangerframe[2]} =
     N[{{Cos[s], Sin[s]}, {Cos[s + Pi / 2], Sin[s + Pi / 2]}}];
hanger = Transpose[{hangerframe[1], hangerframe[2]}];
A = hanger.stretcher.aligner;
MatrixForm[A]
```

$$\begin{pmatrix} 1.37328 & 2.52599 \\ -6.69926 & -4.37074 \end{pmatrix}$$

Reset the xstretch factor, the ystretch factor and both the aligner frame and the hanger frame so that the resulting matrix is a matrix whose hits
→ rotate everything by $\frac{\pi}{4}$ counterclockwise radians .
→ neither stretch nor shrink.

G.3) Make a matrix*

□G.3.a) Make a matrix A so that

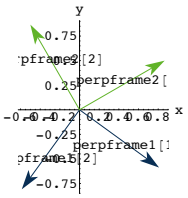
A.perpframe1[1] = 1.2 perpframe2[1]

and

A.perpframe1[2] = 2.6 perpframe2[2]

Here are two perpendicular frames:

```
Clear[perpframe1, perpframe2];
s = -π/5;
{perpframe1[1], perpframe1[2]} =
     N[{{Cos[s], Sin[s]}, -{Cos[s + π/2], Sin[s + π/2]}}];

t = π/6;
{perpframe2[1], perpframe2[2]} =
     N[{{Cos[t], Sin[t]}, {Cos[t + π/2], Sin[t + π/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 -> CinnabarGreen, HeadSize -> 0.2], {k, 1, 2}],
  Graphics[Text["perpframe2[1]", 0.6 perpframe2[1]]],
  Graphics[Text["perpframe2[2]", 0.6 perpframe2[2]]]};
Show[frame1plot, frame2plot, Axes -> True, AxesLabel -> {"x", "y"}];
```



Make a matrix A so that

A.perpframe1[1] = 1.2 perpframe2[1]

and

A.perpframe1[2] = 2.6 perpframe2[2]

Confirm with a convincing calculation.

Click on the right for a friendly tip.

If you make A = hanger.stretcher.aligner, then you are guaranteed that

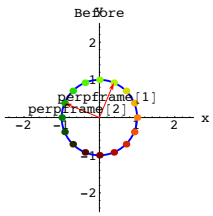
A.alignerframe[1] = xstretch hangerframe[1],

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

☐G.3.b) 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], 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}];
hitframeplotter[matrix2D_] := Table[Arrow[matrix2D.perpframe[k],
   Tail → {0, 0}, VectorColor → Red, 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];
```

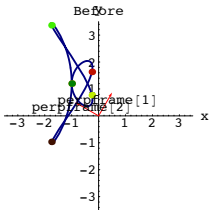


Your job is to make a positive definite (framestretcher) matrix A whose hits
-> stretch every measurement in the direction of perpframe[1] by a factor of 2.0
and
-> stretch every measurement in the direction of perpframe[2] by a factor of 1.1.
Show off your matrix with action movie.

☐G.3.c) Making reflection matrices

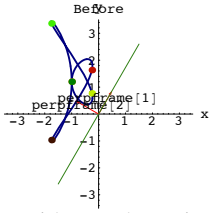
Here's a curve shown together with a perpendicular frame:

```
Clear[x, y, t, perpframe, s];
s = π / 3;
{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 = 3.5;
{x[t_], y[t_]} = {-1, 1.2} + {2 Sin[t]^2 Cos[t], (1 - 3 Cos[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) / 6;
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 → Red], {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];
```

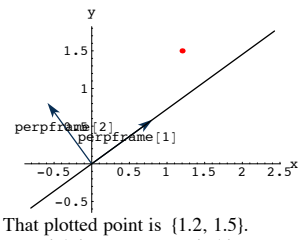


Your job to use the matrix maker ingredients to come up with a matrix A that picks up the curve and flips it over the plotted line to get its mirror image with respect to the plotted line.
Plot to confirm.

☐G.3.d.i) Making a projection matrix

Look at this set-up:

```
Clear[x, y, t, perpframe, s];
s = 0.2 π;
{perpframe[1], perpframe[2]} =
  N[{{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}}];
frameplot = {Table[Arrow[perpframe[k], Tail → {0, 0},
     VectorColor → Indigo, HeadSize → 0.2], {k, 1, 2}],
   Graphics[Text["perpframe[1]", 0.6 perpframe[1]]],
   Graphics[Text["perpframe[2]", 0.6 perpframe[2]]]};
point = {1.2, 1.5};
pointplot = Graphics[{Red, PointSize[0.02], Point[point]}];
lineplot = Graphics[Line[{-perpframe[1], 3 perpframe[1]}]];
Show[frameplot, pointplot, lineplot,
  PlotRange → All, Axes → True, AxesLabel → {"x", "y"}];
```



That plotted point is {1.2, 1.5}.
Your job is to use a matrix hit to come up with the point on the line that is closest to the plotted point.

☐G.3.d.ii) Using your projection matrix

Stay with the same setup as in part iii) and use a matrix hit to answer this question:
Given a cleared point {x,y}, what is the formula (in terms of x and y) for the point on the line that is closest to {x,y}?

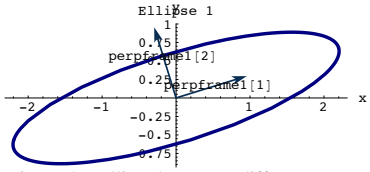
☐G.3.e) Making a matrix that rotates, stretches and then rotates again

Make a matrix A whose action is described as follows:
The hit by A rotates everything $\frac{\pi}{4}$ counterclockwise radians, then stretches by a factor of 1.8 along the x-axis and a factor of 0.7 along the y-axis and then rotates the result by $\frac{\pi}{3}$ clockwise radians.

☐G.3.f) Making a matrix that hits one ellipse into another

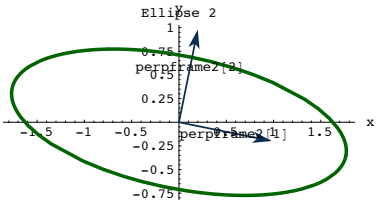
Here is an ellipse hung on a perpendicular frame:

```
Clear[perpframe1, ellipse1plotter, t];
s = 0.3;
    {perpframe1[1], perpframe1[2]} =
     N[{{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/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]]]};

ellipse1plotter[t_] =
  2.3 Cos[t] perpframe1[1] + 0.6 Sin[t] perpframe1[2];

ellipse1plot = ParametricPlot[ellipse1plotter[t],
   {t, 0, 2 Pi}, PlotStyle -> {{Thickness[0.01], NavyBlue}},
   AxesLabel -> {"x", "y"}, PlotLabel -> "Ellipse 1",
   DisplayFunction -> Identity];

Show[ellipse1plot, frame1plot, DisplayFunction -> $DisplayFunction];
```



Here is another ellipse hung on a different perpendicular frame:

```
Clear[perpframe2, ellipse2plotter];
s = -0.2;
    {perpframe2[1], perpframe2[2]} =
     N[{{Cos[s], Sin[s]}, {Cos[s + Pi / 2], Sin[s + Pi / 2]}}];
frame2plot = {Table[
  Arrow[perpframe2[k], Tail -> {0, 0},
     VectorColor -> Indigo, HeadSize -> 0.2], {k, 1, 2}],
  Graphics[Text["perpframe2[1]", 0.6 perpframe2[1]]],
  Graphics[Text["perpframe2[2]", 0.6 perpframe2[2]]]};

ellipse2plotter[t_] =
  1.8 Cos[t] perpframe2[1] + 0.7 Sin[t] perpframe2[2];

ellipse2plot = ParametricPlot[ellipse2plotter[t],
   {t, 0, 2 Pi}, PlotStyle -> {{Thickness[0.01], GosiaGreen}},
   AxesLabel -> {"x", "y"}, PlotLabel -> "Ellipse 2",
   DisplayFunction -> Identity];

Show[ellipse2plot, frame2plot, DisplayFunction -> $DisplayFunction];
```



Your job is to make a matrix A so that when you hit the first ellipse with A, you get the second ellipse.

G.4) Invertible and non-invertible matrices*

□G.4.a) Action of A^{-1} in terms of the action of A

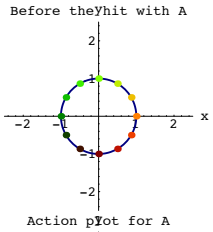
Here's a partially random matrix made with the matrix maker ingredients:

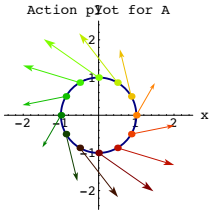
```
Clear[alignerframe];
s = Random[Real, {-π/2, π/2}];
{alignerframe[1], alignerframe[2]} =
  N[{{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}}];
aligner = {alignerframe[1], alignerframe[2]};
{xstretch, ystretch} = {2.5, 1.7};
stretcher = ( xstretch    0
              0        ystretch );
Clear[hangerframe];
s = Random[Real, {-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;
MatrixForm[A]
```

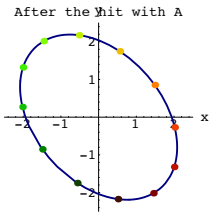
$$\begin{pmatrix} 1.49103 & -1.45517 \\ 0.85309 & 2.01781 \end{pmatrix}$$

Here's an action movie for A showing what a hit with A does to the unit circle:

```
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)/12;
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}];
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}];
Abefore = Show[hitplotter[IdentityMatrix[2]],
   hitpointplotter[IdentityMatrix[2]], PlotLabel →
    "Before the hit with A", DisplayFunction → $DisplayFunction];
Aaction = Show[Abefore, actionarrows[A], PlotLabel →
    "Action plot for A", DisplayFunction → $DisplayFunction];
Aafter = Show[hitplotter[A], hitpointplotter[A], PlotLabel →
    "After the hit with A", DisplayFunction → $DisplayFunction];
```



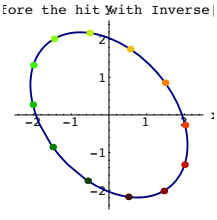


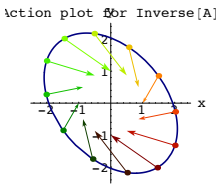


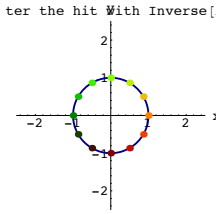
Grab all three plots, align and animate slowly.
To get maximum viewing pleasure and maximum information,
look at one point and follow it.
Then pick another point and follow it.
Keep doing this until you have a good visual grasp of the action.

Now see an action movie depicting what a hit with A^{-1} = Inverse[A] does to the curve you see immediately above:

```
Clear[curveplotter];
curveplotter[t_] = A.{x[t], y[t]};
Clear[x, y];
{x[t_], y[t_]} = curveplotter[t];
AInversebefore = Show[hitplotter[IdentityMatrix[2]],
   hitpointplotter[IdentityMatrix[2]],
   PlotLabel → "Before the hit with Inverse[A]",
   DisplayFunction → $DisplayFunction];
AInverseaction = Show[AInversebefore, actionarrows[Inverse[A]],
   PlotLabel → "Action plot for Inverse[A]",
   DisplayFunction → $DisplayFunction];
AInverseafter = Show[hitplotter[Inverse[A]], hitpointplotter[
    Inverse[A]], PlotLabel → "After the hit with Inverse[A]",
   DisplayFunction → $DisplayFunction];
```



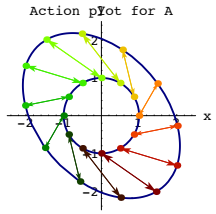




Grab the last three plots and animate.
Then grab all six plots, align and animate slowly.

Take a look at the action arrows for A and for A^{-1} in the same plot.

```
Show[Aaction, AInverseaction]
```



- Graphics -

Review both action movies. Describe ways in which they are similar; describe ways in which they are different. Explain why they had to come out come out the way they did.

□G.4.b) Reflection matrices (flippers) and their inverses

Here's a random reflection matrix :

```
s = Random[Real, {0.1 Pi, 0.4 Pi}]
{perpframe[1], perpframe[2]} =
    N[{{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}}];
```

```
0.457359
```

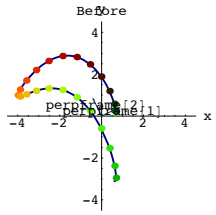
```
Clear[alignerframe];
{alignerframe[1], alignerframe[2]} = {perpframe[1], perpframe[2]};
aligner = {alignerframe[1], alignerframe[2]};
stretcher = {{1, 0}, {0, 1}};
Clear[hangerframe];
{hangerframe[1], hangerframe[2]} = {perpframe[1], -perpframe[2]};
hanger = Transpose[{hangerframe[1], hangerframe[2]}];
flipper = hanger.stretcher.aligner;
MatrixForm[flipper]
```

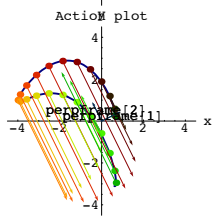
$$\begin{pmatrix} 0.610014 & 0.792391 \\ 0.792391 & -0.610014 \end{pmatrix}$$

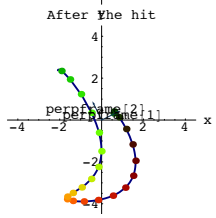
See this matrix do its work:

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



A = flipper;

Clear[x, y, t, hitplotter,
  hitpointplotter, pointcolor, actionarrows, matrix2D];
{tlow, thigh} = {-1, 1};
ranger = 4.5;
{x[t_], y[t_]} =
 {-1, 1} + {-3 Cos[3 t] Cos[t], 2 Sin[t] (1.5 Sin[5 t] - 1) }

pointcolor[t_] = RGBColor[0.5 (Cos[3 t] + 1), 0.5 (Sin[3 t] + 1), 0];

jump = (thigh - tlow)/24;

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}];

actionarrows[matrix2D_] :=
  Table[Arrow[matrix2D.{x[t], y[t]} - {x[t], y[t]}, Tail → {x[t], y[t]}
    VectorColor → pointcolor[t], HeadSize → 0.25],
   {t, tlow, thigh - jump, jump}];

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

Show[before, actionarrows[A], frameplot,
  PlotLabel → "Action plot", DisplayFunction → $DisplayFunction];

Show[hitplotter[A], hitpointplotter[A], frameplot,
  PlotLabel → "After the hit", DisplayFunction → $DisplayFunction];
```

```
{-1 - 3 Cos[t] Cos[3 t], 1 + 2 Sin[t] (-1 + 1.5 Sin[5 t])}
```







Grab all three plots, align and animate slowly.

Look again at the reflection matrix:

```
MatrixForm[flipper]
```

$$\begin{pmatrix} 0.610014 & 0.792391 \\ 0.792391 & -0.610014 \end{pmatrix}$$

And look at Mathematica's calculation of flipper^{-1}

```
MatrixForm[Inverse[flipper]]
```

$$\begin{pmatrix} 0.610014 & 0.792391 \\ 0.792391 & -0.610014 \end{pmatrix}$$

Say why you are totally cool with this.

□G.4.c.i) Using an inverse matrix (Junior grade)

Here's a random matrix A made with the matrix maker ingredients with positive stretch factors:

```
Clear[alignerframe];
s = Random[Real, {-π/4, π/4}];
{alignerframe[1], alignerframe[2]} =
  N[{{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}}];
aligner = {alignerframe[1], alignerframe[2]};
{xstretch, ystretch} = {2.2, 1.3};
stretcher = DiagonalMatrix[{xstretch, ystretch}];
Clear[hangerframe];
t = Random[Real, {-π/4, π/4}];
{hangerframe[1], hangerframe[2]} =
  N[{{Cos[t], Sin[t]}, {Cos[t + π/2], Sin[t + π/2]}}];
hanger = Transpose[{hangerframe[1], hangerframe[2]}];
A = hanger.stretcher.aligner;
MatrixForm[A]
```

$$\begin{pmatrix} 1.95422 & 0.432146 \\ -0.986643 & 1.24532 \end{pmatrix}$$

This matrix is guaranteed to be invertible because it was made with positive stretch factors.
Use a hit with A^{-1} to come up with the {x,y} that makes
A.{x,y} = {1.94, 2.68}.

□G.4.c.ii) Using an inverse matrix (Utility grade)

Here's a new matrix A made with the matrix maker ingredients:

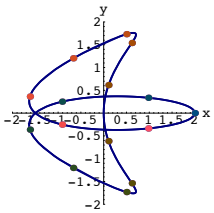
```
Clear[alignerframe];
s = -π/8;
```

```
{alignerframe[1], alignerframe[2]} =
  N[{{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}}];
aligner = {alignerframe[1], alignerframe[2]};
{xstretch, ystretch} = {2.2, 1.3};
stretcher = DiagonalMatrix[{xstretch, ystretch}];
Clear[hangerframe];
s = 0.1 π;
{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]
```

$$\begin{pmatrix} 1.77932 & -1.17184 \\ 1.10123 & 0.882098 \end{pmatrix}$$

Here's a parameterization and a plot of a curve:

```
Clear[curveplotter, pointcolor, t];
{tlow, thigh} = {0, 2 π};
ranger = 2.0;
curveplotter[t_] = 2 {Cos[t] Cos[5 t], Sin[t] Cos[3 t]};
pointcolor[t_] = RGBColor[0.5 (Cos[t] + 1), 0.3, 0.5 (Sin[t] + 1)];
jump = (thigh - tlow)/30;
curveplot = ParametricPlot[curveplotter[t],
  {t, tlow, thigh}, PlotStyle → {{Thickness[0.01], NavyBlue}},
  PlotRange → {{-ranger, ranger}, {-ranger, ranger}},
  AxesLabel → {"x", "y"}, DisplayFunction → Identity];
pointplot = Table[Graphics[{pointcolor[t], PointSize[0.035],
     Point[curveplotter[t]]}], {t, tlow, thigh - jump, jump}];
givencurve = Show[curveplot, pointplot,
   DisplayFunction → $DisplayFunction];
```



The parameterization of this curve is:

```
curveplotter[t]
```

```
{2 Cos[t] Cos[5 t], 2 Cos[3 t] Sin[t]}
```

Your job is to use the inverse of A to come up with the parameterization {x[t],y[t]} of a curve so that

A.{x[t], y[t]} = curveplotter[t].

Show off your work with a convincing plot.

□G.4.d.i) Stretch factors and invertibility

Here's a partially random matrix made with matrix maker ingredients:

```
Clear[alignerframe];
s = Random[Real, {-Pi/2, Pi/2}];
{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, {0.2, 6}], Random[Real, {0.2, 6}]};
stretcher = DiagonalMatrix[{xstretch, ystretch}];
Clear[hangerframe];
s = Random[Real, {-Pi/2, Pi/2}];
{hangerframe[1], hangerframe[2]} =
     {{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]
```

$$\begin{pmatrix} 0.827871 & -2.43194 \\ 2.05102 & 1.16396 \end{pmatrix}$$

Looking at the stretcher:

```
MatrixForm[stretcher]
```

$$\begin{pmatrix} 2.19877 & 0 \\ 0 & 2.70677 \end{pmatrix}$$

you announce that this matrix is invertible. You are right. Why are you right?

□G.4.d.ii) Matrix maker ingredients used to make A to duplicate A^{-1}

Stay with the matrix A in part i).

The inverse, A^{-1}, of A is:

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

$$\begin{pmatrix} 0.195573 & 0.408622 \\ -0.344618 & 0.139101 \end{pmatrix}$$

Use matrix maker ingredients used to make A to duplicate A^{-1}.

□G.4.d.iii) Is it true that the hangerframe of A is the alignerframe of A^{-1}?

Is it true that the alignerframe of A is the hangerframe of A^{-1}?

Is it true that the stretch factors of A^{-1} are the reciprocals of the stretch factors of A?

Stay with the same matrix A as in part i) immediately above.

Is it true that the hangerframe you used to make of A is the alignerframe of A^{-1}?

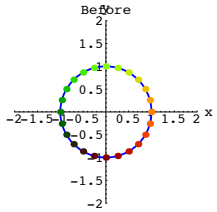
Is it true that the alignerframe of A is the hangerframe you used to make of A^{-1}?

Is it true that the stretch factors of A^{-1} are the reciprocals of the stretch factors of A?

□G.4.e.i) Non-invertible matrices

Here's the unit circle centered at {0,0}:

```
Clear[x, y, t, perpframe, s];
{tlow, thigh} = {0, 2 π};
ranger = 2.0;
{x[t_], y[t_]} = {Cos[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)/24;
hitplotter[matrix2D_] :=
  ParametricPlot[matrix2D.{x[t], y[t]}, {t, tlow, thigh},
   PlotStyle → {{Thickness[0.01], Blue}}, PlotRange →
    {{-ranger, ranger}, {-ranger, ranger}}, AspectRatio → Automatic,
   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 → Red], {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}];
before = Show[hitplotter[IdentityMatrix[2]],
   hitpointplotter[IdentityMatrix[2]], PlotLabel → "Before",
   DisplayFunction → $DisplayFunction];
```



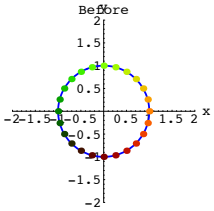
Here's a non-invertible matrix made with matrix-maker ingredients:

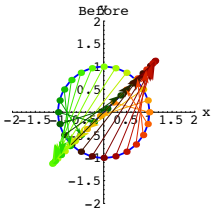
```
Clear[alignerframe];
s = Random[Real, {-π/2, π/2}];
{alignerframe[1], alignerframe[2]} =
  N[{{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}}];
aligner = {alignerframe[1], alignerframe[2]};
{xstretch, ystretch} = {1.6, 0};
stretcher = DiagonalMatrix[{xstretch, ystretch}];
Clear[hangerframe, s, hanger];
s = 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]
```

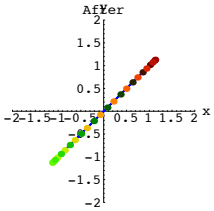
$$\begin{pmatrix} 0.214801 & -1.11079 \\ 0.214801 & -1.11079 \end{pmatrix}$$

Here's what a hit with A does to the unit circle:

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

Show[before, hitplotter[A], hitpointplotter[A],
  actionarrows[A], DisplayFunction → $DisplayFunction];

after = Show[hitplotter[A], hitpointplotter[A],
   PlotLabel → "After", DisplayFunction → $DisplayFunction];
```







Go back and look at the stretch factors used to make the matrix and then say why you are not surprised with the outcome of the hit..

□G.4.e.ii) Distinguishing between invertible matrices and non-invertible matrices by looking at the outcomes of hits on the unit circle.

Explain how you can visually distinguish between invertible matrices and non-invertible matrices by looking at the outcomes of hits on the unit circle.

□G.4.e.iii) Hanger frames and non invertible matrices

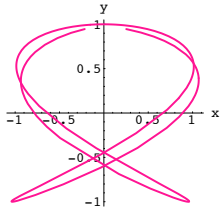
Here's an otherwise random noninvertible matrix:

```
Clear[alignerframe];
s = Random[Real, {-π/2, π/2}];
{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, {0.2, 1.3}], 0};
stretcher = DiagonalMatrix[{xstretch, ystretch}];
Clear[hangerframe];
s = Random[Real, {-π/2, -π/3}];
{hangerframe[1], hangerframe[2]} =
    {{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]
```

$$\begin{pmatrix} 0.00150134 & 0.0761779 \\ -0.00461602 & -0.234217 \end{pmatrix}$$

Here's a partially random curve:

```
a = Random[Real, {-0.3, 0.1}];
b = Random[Real, {1, 6}];
c = Random[Real, {1, 6}];
{x[t_], y[t_]} = {e^(a t) Sin[b t], Cos[c t]};
curveplot = ParametricPlot[{x[t], y[t]},
   {t, -2, 2}, PlotStyle -> {{Thickness[0.01], DeepPink}},
   PlotRange -> All, AxesLabel -> {"x", "y"}];
```

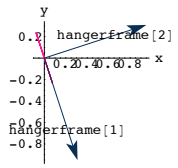


Here's what you get when you hit this curve with the noninvertible matrix A calculated above:

```
hitcurveplot = ParametricPlot[A.{x[t], y[t]},
   {t, -2, 2}, PlotStyle -> {{Thickness[0.01], DeepPink}},
   PlotRange -> All, AxesLabel -> {"x", "y"}];
```

See this line plotted with the hangerframe used to make A:

```
hangerframeplot = {Table[
   Arrow[hangerframe[k],
      Tail -> {0, 0}, VectorColor -> Indigo], {k, 1, 2}],
   Graphics[Text["hangerframe[1]", 0.7 hangerframe[1]]],
   Graphics[Text["hangerframe[2]", 0.7 hangerframe[2]]]};
Show[hitcurveplot, hangerframeplot];
```



Go back and rerun all parts.

Got any idea why that happened and why it happens every time?

G.5) Orientation:Counterclockwise versus clockwise*

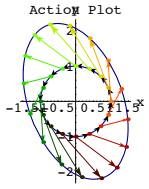
□G.5.a.i) Action diagram

Here's an action diagram depicting what a hit with the matrix

$$A = \begin{pmatrix} 0.92 & -1.17 \\ 1.16 & 1.88 \end{pmatrix}$$

does to the counterclockwise unit circle:

```
A = {{0.92, -1.17}, {1.16, 1.88}};
Clear[x, y, t];
{tlow, thigh} = {0, 2 π};
{x[t_], y[t_]} = {Cos[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)/16;
hitplotter[matrix2D_] := ParametricPlot[matrix2D.{x[t], y[t]},
   {t, tlow, thigh}, PlotStyle → {{Thickness[0.01], NavyBlue}},
   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}];
pointers[matrix_] := Table[ArrowHead[
    matrix.{x[t], y[t]}, matrix.{x'[t], y'[t]},
              HeadSize -> 0.2, VectorColor -> Black,
    Aperture -> 0.4], {t, tlow, thigh, jump}];

Show[hitplotter[IdentityMatrix[2]],
  pointers[IdentityMatrix[2]], hitpointplotter[IdentityMatrix[2]],
  actionarrows[A], hitplotter[A], hitpointplotter[A],
  PlotLabel → "Action Plot", DisplayFunction → $DisplayFunction];

MatrixForm[A]
```



$$\begin{pmatrix} 0.92 & -1.17 \\ 1.16 & 1.88 \end{pmatrix}$$

Look at the diagram and then you make the call:
Does a hit with this matrix preserve or reverse orientation?

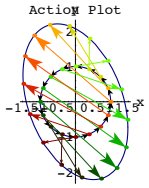
□G.5.a.ii) Another action diagram

Here's an action diagram depicting what a hit with the matrix

$$A = \begin{pmatrix} 0.92 & -1.17 \\ 1.16 & 1.88 \end{pmatrix}$$

does to the counterclockwise unit circle:

```
A = {{-1.17, 0.92}, {1.88, 1.16}};
Show[hitplotter[IdentityMatrix[2]],
  pointers[IdentityMatrix[2]], hitpointplotter[IdentityMatrix[2]],
  actionarrows[A], hitplotter[A], hitpointplotter[A],
  PlotLabel → "Action Plot", DisplayFunction → $DisplayFunction];
MatrixForm[A]
```



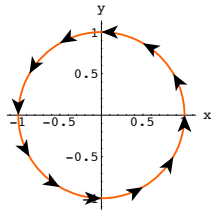
$$\begin{pmatrix} -1.17 & 0.92 \\ 1.88 & 1.16 \end{pmatrix}$$

Look at the diagram and then you make the call:
Does a hit with this matrix preserve or reverse orientation?

□G.5.b) Resetting the hanger frame

Here's the unit circle shown with tangent vectors that indicate the direction of the parameterization

```
Clear[x, y, pointers, hitplot, matrix, t];
{tlow, thigh} = {0, 2 π};
{x[t_], y[t_]} = {Cos[t], Sin[t]};
hitplot[matrix_] := ParametricPlot[matrix.{x[t], y[t]},
  {t, tlow, thigh}, PlotStyle -> {{Thickness[0.01], CadmiumOrange}},
  AxesLabel -> {"x", "y"}, DisplayFunction -> Identity];
tjump = (thigh - tlow)/12;
pointers[matrix_] :=
  Table[ArrowHead[matrix.{x[t], y[t]}, matrix.{x'[t], y'[t]},
              HeadSize -> 0.2, VectorColor -> Black,
    Aperture -> 0.4], {t, tlow, thigh, tjump}];
circleplot = Show[hitplot[IdentityMatrix[2]], pointers[
    IdentityMatrix[2]], DisplayFunction -> $DisplayFunction];
```



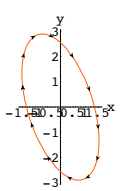
Make this matrix:

```
s = 2.7;
aligner = {{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}};

{xstretch, ystretch} = {1.3, 3.0};
stretcher = {{xstretch, 0}, {0, ystretch}};

t = 0.3;
hanger =
 Transpose[ {{Cos[t], Sin[t]}, -{Cos[t + π/2], Sin[t + π/2]}}];
A = hanger.stretcher.aligner;
MatrixForm[A]
```

$$\begin{pmatrix} -1.5017 & -0.270736 \\ 0.877552 & 2.75527 \end{pmatrix}$$

See what a hit with this matrix does to the counterclockwise unit circle:

```
hitcircleplot =
 Show[hitplot[A], pointers[A], DisplayFunction -> $DisplayFunction];
```



Hits with this matrix reverse orientation.
Here's the code that made this matrix.

```
s = 2.7;
aligner = {{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}};

{xstretch, ystretch} = {1.3, 3.0};
stretcher = {{xstretch, 0}, {0, ystretch}};

t = 0.3;
hanger =
 Transpose[ {{Cos[t], Sin[t]}, -{Cos[t + π/2], Sin[t + π/2]}}];
A = hanger.stretcher.aligner;
MatrixForm[A]
```

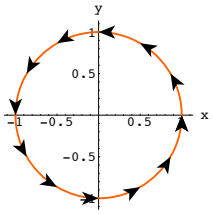
$$\begin{pmatrix} -1.5017 & -0.270736 \\ 0.877552 & 2.75527 \end{pmatrix}$$

Reset the hanger frame so that hits with the resulting matrix preserve orientation.
Show off your work with a convincing plot.

□G.5.c) One, two and three hits

Here's the unit circle shown with tangent vectors that indicate the direction of the parameterization

```
Clear[x, y, pointers, hitplot, matrix, t];
{tlow, thigh} = {0, 2 π};
{x[t_], y[t_]} = {Cos[t], Sin[t]};
hitplot[matrix_] := ParametricPlot[matrix.{x[t], y[t]},
  {t, tlow, thigh}, PlotStyle -> {{Thickness[0.01], CadmiumOrange}},
  AxesLabel -> {"x", "y"}, DisplayFunction -> Identity];
tjump = (thigh - tlow)/12;
pointers[matrix_] :=
  Table[ArrowHead[matrix.{x[t], y[t]}, matrix.{x'[t], y'[t]},
              HeadSize -> 0.2, VectorColor -> Black,
    Aperture -> 0.4], {t, tlow, thigh, tjump}];
circleplot = Show[hitplot[IdentityMatrix[2]], pointers[
    IdentityMatrix[2]], DisplayFunction -> $DisplayFunction];
```



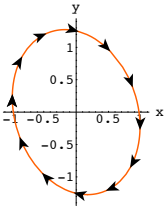
Make this matrix:

```
s = 2.7;
aligner = {{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}};

{xstretch, ystretch} = {0.95, 1.3};
stretcher = {{xstretch, 0}, {0, ystretch}};

t = 0.3;
hanger =
 Transpose[ {{Cos[t], Sin[t]}, -{Cos[t + π/2], Sin[t + π/2]}}];
A = hanger.stretcher.aligner;
MatrixForm[A]
```

$$\begin{pmatrix} -0.984698 & 0.040554 \\ 0.276966 & 1.24279 \end{pmatrix}$$

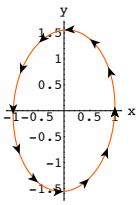
See what a hit with A does to the counterclockwise unit circle:

```
hitcircleplot =
 Show[hitplot[A], pointers[A], DisplayFunction -> $DisplayFunction];
```



Hits with A reverse orientation.
Now see what two hits with A do to the counterclockwise unit circle:

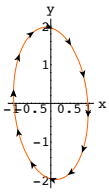
```
twohitcircleplot = Show[hitplot[A.A],
  pointers[A.A], DisplayFunction -> $DisplayFunction];
```



Give your take on why that happened.
Put answer here.

Now see what three hits with A do to the counterclockwise unit circle:

```
threehitcircleplot = Show[hitplot[A.A.A],
  pointers[A.A.A], DisplayFunction -> $DisplayFunction];
```



Give your take on why that happened.
Put answer here.

□G.5.d.i) Made with a right hand aligner and a left hand hanger

Here is a matrix A made with the matrix maker:

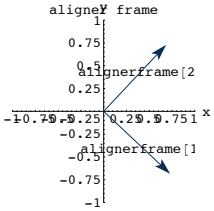
```
Clear[alignerframe];
s = Random[Real, {-1.5, 1.5}];
{alignerframe[1], alignerframe[2]} =
  N[{{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}}];
aligner = {alignerframe[1], alignerframe[2]};
{xstretch, ystretch} =
  {Random[Real, {1, 2}], Random[Real, {1, 2}]};
stretcher = ( xstretch    0
              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]
```

$$\begin{pmatrix} -0.992032 & -1.53355 \\ -1.55318 & 1.0184 \end{pmatrix}$$

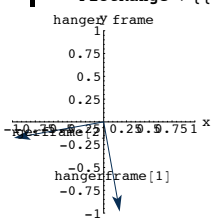
Here is a plot of the aligner frame:

```
alignerframeplot = Show[Table[Arrow[alignerframe[k],
     Tail → {0, 0}, VectorColor → Indigo], {k, 1, 2}],
   Graphics[Text["alignerframe[1]", 0.6 alignerframe[1]]],
   Graphics[Text["alignerframe[2]", 0.6 alignerframe[2]]],
   Axes → True, AxesLabel → {"x", "y"},
   PlotRange → {{-1, 1}, {-1, 1}}, PlotLabel → "aligner frame"];
```



The aligner frame is a right hand frame.
Check out the hanger frame:

```
hangerframeplot = Show[
   Table[Arrow[hangerframe[k], Tail → {0, 0}, VectorColor → Indigo],
    {k, 1, 2}], Graphics[Text["hangerframe[1]", 0.6 hangerframe[1]]],
   Graphics[Text["hangerframe[2]", 0.6 hangerframe[2]]],
   Axes → True, AxesLabel → {"x", "y"},
   PlotRange → {{-1, 1}, {-1, 1}}, PlotLabel → "hanger frame"];
```



The hanger frame is a left hand frame.

Check out the stretch factors:

```
{xstretch, ystretch}
```

```
{1.85829, 1.82542}
```

Multiple choice:
a) A hit with this matrix can be described as a flip followed by a stretch followed by a rotation.

b) A hit with this matrix can be described as a rotation followed by a stretch followed by a flip.

c) A hit with this matrix can be described as a rotation followed by a stretch followed by another rotation.
My choice is.....

Agree or disagree:
A hit with this matrix reverses orientation.
Agree..... Disagree.....

□G.5.d.ii) Made with a right hand aligner and a right hand hanger

You make a matrix
A = hanger.stretcher.aligner
with
- the aligner frame based on a right hand perpendicular frame
- the hanger frame based on a right hand perpendicular frame
- positive stretch factors.

Multiple choice:
a) A hit with this matrix can be described as a flip followed by a stretch followed by a rotation.

b) A hit with this matrix can be described as a rotation followed by a stretch followed by a flip.

c) A hit with this matrix can be described as a rotation followed by a stretch followed by another rotation.
My choice is.....

Agree or disagree:
A hit with this matrix reverses orientation.
Agree..... Disagree.....

□G5.d.iii) Made with a left hand aligner and a right hand hanger

You make a matrix
A = hanger.stretcher.aligner
with
- the aligner frame based on a left hand perpendicular frame
- the hanger frame based on a right hand perpendicular frame
- positive stretch factors.

Multiple choice:
a) A hit with this matrix can be described as a flip followed by a stretch followed by a rotation.

b) A hit with this matrix can be described as a rotation followed by a stretch followed by a flip.

c) A hit with this matrix can be described as a rotation followed by a stretch followed by another rotation.
My choice is.....

Agree or disagree:
A hit with this matrix reverses orientation.
Agree..... Disagree.....

□G.5.d.iv) Made with a left hand aligner and a left hand hanger

You make a matrix
A = hanger.stretcher.aligner
with
- the aligner frame based on a left hand perpendicular frame
- the hanger frame based on a left hand perpendicular frame
- positive stretch factors.

Multiple choice:
a) A hit with this matrix can be described as a flip followed by a stretch followed by a rotation.

b) A hit with this matrix can be described as a rotation followed by a stretch followed by a flip.

c) A hit with this matrix can be described as a rotation followed by a stretch followed by another rotation.
My choice is.....

```
Careful, click on the right for an issue to think about.
```

In this setup,

$$\text{aligner} = \begin{pmatrix} \text{alignerframe[1]} \rightarrow \\ \text{alignerframe[2]} \rightarrow \end{pmatrix},$$

$$\text{hanger} = \begin{pmatrix} \text{hangerframe[1]} & \text{hangerframe[2]} \\ \downarrow & \downarrow \end{pmatrix},$$

$$A = \begin{pmatrix} \text{hangerframe[1]} & \text{hangerframe[2]} \\ \downarrow & \downarrow \end{pmatrix} . \begin{pmatrix} \text{xstretch} & 0 \\ 0 & \text{ystretch} \end{pmatrix} . \begin{pmatrix} \text{alignerframe[1]} \rightarrow \\ \text{alignerframe[2]} \rightarrow \end{pmatrix}$$

and both

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

are left hand frames.

Consequently both

{alignerframe[1],-alignerframe[2]} and {hangerframe[1],-hangerframe[2]}

are right hand frames.

And here's the kicker:

$$A = \begin{pmatrix} \text{hangerframe[1]} & -\text{hangerframe[2]} \\ \downarrow & \downarrow \end{pmatrix} . \begin{pmatrix} \text{xstretch} & 0 \\ 0 & \text{ystretch} \end{pmatrix} . \begin{pmatrix} \text{alignerframe[1]} \rightarrow \\ -\text{alignerframe[2]} \rightarrow \end{pmatrix}$$

Reason: The minus signs cancel out,

The juicy consequence:

When you make a matrix

A = hanger.stretcher.aligner with

- the aligner frame based on a left hand perpendicular frame
- the hanger frame based on a left hand perpendicular frame
- positive stretch factors,

you can make the same matrix

A = newhanger . stretcher . newaligner, with

- the new aligner frame based on a right hand perpendicular frame
- the new hanger frame based on a right hand perpendicular frame
- positive stretch factors.

Agree or disagree:
A hit with this matrix reverses orientation.
Agree..... Disagree.....

G.6) A flip followed by another flip results in one rotation.

A flip followed by a rotation results in one flip.

A rotation followed by a flip results in one flip

□G.6.a.i) Two flips result in one rotation

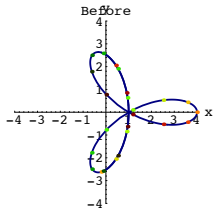
To flip a curve about the line through {0,0} defined by
{Cos[s],Sin[s]}
you hit with this reflection matrix:

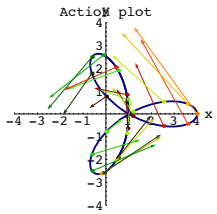
```
Clear[perpframe, s, flipper];
{perpframe[1], perpframe[2]} =
  {{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}};
Clear[alignerframe];
{alignerframe[1], alignerframe[2]} = {perpframe[1], perpframe[2]};
aligner = {alignerframe[1], alignerframe[2]};
stretcher = {{1, 0}, {0, 1}};
Clear[hangerframe];
{hangerframe[1], hangerframe[2]} = {perpframe[1], -perpframe[2]};
hanger = Transpose[{hangerframe[1], hangerframe[2]}];
flipper[s_] = Simplify[hanger.stretcher.aligner];
MatrixForm[flipper[s]]
```

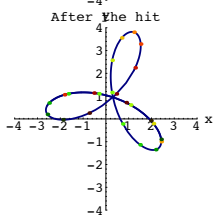
$$\begin{pmatrix} \text{Cos[2 s]} & \text{Sin[2 s]} \\ \text{Sin[2 s]} & -\text{Cos[2 s]} \end{pmatrix}$$

See an action movie depicting what happens when you hit with
flipper[0.4 π].flipper[0.2 π]:

```
A = flipper[0.4 π].flipper[0.2 π];

Clear[x, y, t, hitplotter,
  hitpointplotter, pointcolor, actionarrows, matrix2D];
{tlow, thigh} = {-2, 2};
ranger = 4;
{x[t_], y[t_]} = {1, 0} + {3 Cos[3 t] Cos[t] , 3 Sin[t] Cos[3 t]};
pointcolor[t_] = RGBColor[0.5 (Cos[3 t] + 1), 0.5 (Sin[3 t] + 1), 0];

jump = (thigh - tlow)/24;

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[0.02],
     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], HeadSize → 0.25],
   {t, tlow, thigh - jump, jump}];

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

Show[before, actionarrows[A],
  PlotLabel → "Action plot", DisplayFunction → $DisplayFunction];

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







Grab and animate.

Sure looks like flipper[0.4 π].flipper[0.2 π] is a rotation.
Investigate further:
When you do two flips - the first corresponding to an angle s and the second corresponding to an angle t, you hit with:

```
MatrixForm[flipper[t].flipper[s]]
```

$$\begin{pmatrix} \text{Cos[2 s] Cos[2 t]} + \text{Sin[2 s] Sin[2 t]} & \text{Cos[2 t] Sin[2 s]} - \text{Cos[2 s] Sin[2 t]} \\ -\text{Cos[2 t] Sin[2 s]} + \text{Cos[2 s] Sin[2 t]} & \text{Cos[2 s] Cos[2 t]} + \text{Sin[2 s] Sin[2 t]} \end{pmatrix}$$

Clean this up by applying trig identities:

```
MatrixForm[Simplify[flipper[t].flipper[s]]]
```

$$\begin{pmatrix} \text{Cos[2 (s - t)]} & \text{Sin[2 (s - t)]} \\ -\text{Sin[2 (s - t)]} & \text{Cos[2 (s - t)]} \end{pmatrix}$$

When you use a rotation matrix to rotate about {0,0} by an angle θ, you hit with:

```
Clear[rotator, θ];
rotator[θ_] =
  Transpose[{{Cos[θ], Sin[θ]}, {Cos[θ + π/2], Sin[θ + π/2]}}];
MatrixForm[rotator[θ]]
```

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

Remembering that
Sin[−x] = −Sin[x] and that Cos[−x] = Cos[x],

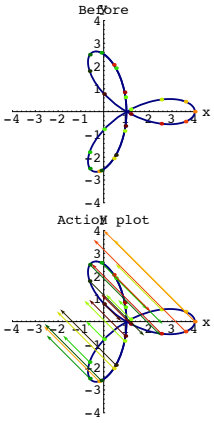
use what you see above to set
θ in terms of s and t
so that the two flips - the first corresponding to an angle s and the second corresponding to an angle t gives the same result as rotation about {0, 0} by the angle θ.

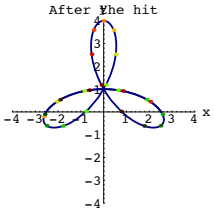
After you have done this, you have proved a theorem:
Namely that two successive flips (reflections) result in one rotation.

□G.6.a.ii) A rotation followed by a flip results in one flip

See an action movie depicting what happens when you hit with
flipper[0.4 π].rotator[0.3π]:

```
A = flipper[0.4 π].rotator[0.3 π];


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

Show[before, actionarrows[A],
  PlotLabel → "Action plot", DisplayFunction → $DisplayFunction];

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





Sure looks like flipper[0.4 π].rotator[0.3π] is a reflection matrix (flipper).
Investigate further:
When you rotate by s radians and then you flip corresponding to an angle t, you hit with

```
MatrixForm[flipper[t].rotator[s]]
```

$$\begin{pmatrix} \text{Cos[s] Cos[2 t]} + \text{Sin[s] Sin[2 t]} & -\text{Cos[2 t] Sin[s]} + \text{Cos[s] Sin[2 t]} \\ -\text{Cos[2 t] Sin[s]} + \text{Cos[s] Sin[2 t]} & -\text{Cos[s] Cos[2 t]} - \text{Sin[s] Sin[2 t]} \end{pmatrix}$$

Clean this up by applying trig identities:

```
MatrixForm[Simplify[flipper[t].rotator[s]]]
```

$$\begin{pmatrix} \text{Cos[s - 2 t]} & -\text{Sin[s - 2 t]} \\ -\text{Sin[s - 2 t]} & -\text{Cos[s - 2 t]} \end{pmatrix}$$

When you use a flipper matrix corresponding to angle θ, you hit with:

```
MatrixForm[flipper[θ]]
```

$$\begin{pmatrix} \text{Cos[2 }\theta\text{]} & \text{Sin[2 }\theta\text{]} \\ \text{Sin[2 }\theta\text{]} & -\text{Cos[2 }\theta\text{]} \end{pmatrix}$$

Remembering that
Sin[−x] = −Sin[x] and that Cos[−x] = Cos[x],
use what you see above to set
θ in terms of s and t
so that
flipper[t].rotator[s] = flipper[θ]

After you have done this, you will have proved a theorem-
Namely that a rotation followed by a flip results in one flip..

□G.6.a.iii) A flip followed by a rotation results in one flip?

Copy, paste and edit the work in part ii) immediately above to try to answer this question:
Does a flip followed by a rotation result in one flip?

G.7) The neat relationship between the inverse and the transpose *

If

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

then

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

and

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

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

□G.7.a) Hits with A , A^{-1} and A^t

Thanks to Mark Anderson of University of Illinois at Urbana-Champaign
suggesting this new, improved version of this problem.

Here's a random matrix made with matrix maker ingredients

```
Clear[alignerframe];
s = Random[Real, {-Pi / 2, Pi / 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, 3}], Random[Real, {0.5, 1.5}]};
stretcher = {{xstretch, 0}, {0, ystretch}};

Clear[hangerframe];
s = Random[Real, {-Pi / 2, Pi / 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;
MatrixForm[A]
```

$$\begin{pmatrix} 1.04448 & 0.470615 \\ -0.73045 & 1.38628 \end{pmatrix}$$

Here are A^t, the transpose of A, and A^{-1} ,the inverse of A:

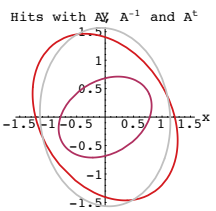
```
MatrixForm[Transpose[A]]
MatrixForm[Inverse[A]]
```

$$\begin{pmatrix} 1.04448 & -0.73045 \\ 0.470615 & 1.38628 \end{pmatrix}$$

$$\begin{pmatrix} 0.773725 & -0.262665 \\ 0.407686 & 0.582954 \end{pmatrix}$$

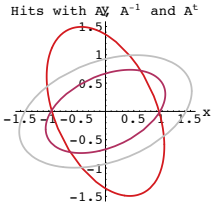
They don't look much alike.
But see what they do when you hit A , A^{-1} and A^t on the unit circle:

```
Clear[x, y, t, s];
{tlow, thigh} = {0, 2 π};
ranger = Max[{xstretch, ystretch, 1 / xstretch, 1 / ystretch, 1.2}];
{x[t_], y[t_]} = {Cos[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) / 8;
hitplotter[matrix2D_, color_] :=
  ParametricPlot[matrix2D.{x[t], y[t]}, {t, tlow, thigh},
   PlotStyle → {{Thickness[0.01], color}},
   AxesLabel → {"x", "y"}, DisplayFunction → Identity];
Show[hitplotter[Transpose[A], VenetianRed], hitplotter[A, Gray],
 hitplotter[Inverse[A], Maroon],
    PlotRange -> {{-ranger, ranger}, {-ranger, ranger}},
    PlotLabel -> "Hits with A, A^-1 and A^t",
    DisplayFunction -> $DisplayFunction];
```



Try it for a new random matrix A:

```
Clear[alignerframe];
s = Random[Real, {-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, {1.5, 3}], Random[Real, {0.5, 1.3}]};
stretcher = ( xstretch   0
              0          ystretch );

Clear[hangerframe];
s = Random[Real, {-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;
ranger = Max[{xstretch, ystretch, 1 / xstretch, 1 / ystretch, 1.2}];
Show[hitplotter[Transpose[A], VenetianRed], hitplotter[A, Gray],
 hitplotter[Inverse[A], Maroon],
    PlotRange -> {{-ranger, ranger}, {-ranger, ranger}},
    PlotLabel -> "Hits with A, A^-1 and A^t",
    DisplayFunction -> $DisplayFunction];
```



Rerun lots of times until you spot the pattern that emerges.
Describe what you see and try to explain why you see it.
Some questions to ponder:
Which ellipse comes from the hit with A?
Which ellipse comes from the hit with A^t?
Which ellipse comes from the hit with A^{-1}?
Two of the ellipses seem to be hanging on the same perpendicular frame. What perpendicular frame is it?

Why does the long axis of one of the ellipses line up with the short axis of one of the others?

Click on the right for some tips.

According to one of the Basics, 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:

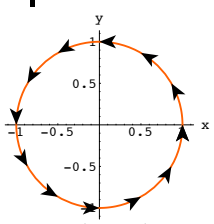
You can make A^t, the transpose of A, by

- 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 using with the same stretch factors as you used for A.

□G.7.b) Orientation and A , A^{-1} and A^t

Here's the unit circle shown with tangent vectors that indicate the direction of the parameterization

```
Clear[x, y, pointers, hitplot, matrix, t];
{tlow, thigh} = {0, 2π};
{x[t_], y[t_]} = {Cos[t], Sin[t]};
hitplot[matrix_] := ParametricPlot[matrix.{x[t], y[t]},
  {t, tlow, thigh}, PlotStyle -> {{Thickness[0.01], CadmiumOrange}},
  AxesLabel -> {"x", "y"}, DisplayFunction -> Identity];
tjump = (thigh - tlow) / 12;
pointers[matrix_] :=
  Table[ArrowHead[matrix.{x[t], y[t]}, matrix.{x'[t], y'[t]},
                  HeadSize -> 0.2, VectorColor -> Black,
    Aperture -> 0.4], {t, tlow, thigh, tjump}];
circleplot = Show[hitplot[IdentityMatrix[2]], pointers[
    IdentityMatrix[2]], DisplayFunction -> $DisplayFunction];
```



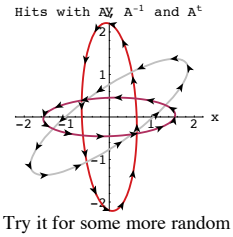
Here's another random matrix made with matrix maker ingredients

```
Clear[alignerframe];
s = Random[Real, {-Pi / 2, 0}];
{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, 3}], Random[Real, {0.5, 1.5}]};
stretcher = ( xstretch   0
              0          ystretch );

Clear[hangerframe];
s = Random[Real, {0.4, Pi / 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]
```

$$\begin{pmatrix} -0.303487 & -1.82801 \\ 0.573307 & -1.24143 \end{pmatrix}$$

Here's what hits with A, A^t, the transpose of A, and A^{-1} ,the inverse of A do to the counterclockwise unit circle:

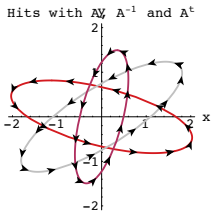
```
Clear[x, y, t, s];
{tlow, thigh} = {0, 2π};
ranger = Max[{xstretch, ystretch, 1 / xstretch, 1 / ystretch, 1.2}];
{x[t_], y[t_]} = {Cos[t], Sin[t]};
Clear[hitplotter, hitpointplotter,
  pointcolor, actionarrows, matrix2D];
pointcolor[t_] = RGBColor[0.5 (Cos[t] + 1), 0.5 (Sin[t] + 1), 0];
jump = (thigh - tlow) / 8;
hitplotter[matrix2D_, color_] :=
  ParametricPlot[matrix2D.{x[t], y[t]}, {t, tlow, thigh},
   PlotStyle → {{Thickness[0.01], color}},
   AxesLabel → {"x", "y"}, DisplayFunction → Identity];

Show[hitplotter[Transpose[A], VenetianRed], hitplotter[A, Gray],
 hitplotter[Inverse[A], Maroon], pointers[A],
 pointers[Transpose[A]], pointers[Inverse[A]],
    PlotRange -> {{-ranger, ranger}, {-ranger, ranger}},
    PlotLabel -> "Hits with A, A^-1 and A^t",
    DisplayFunction -> $DisplayFunction];
```



Try it for some more random matrices:

```
Clear[alignerframe];
s = Random[Real, {-Pi / 2, 0}];
{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, 3}], Random[Real, {0.5, 1.5}]};
stretcher = ( xstretch   0
              0          ystretch );

Clear[hangerframe];
s = Random[Real, {0.4, Pi / 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;

ranger = Max[{xstretch, ystretch, 1 / xstretch, 1 / ystretch, 1.2}];


Show[hitplotter[Transpose[A], VenetianRed], hitplotter[A, Gray],
 hitplotter[Inverse[A], Maroon], pointers[A],
 pointers[Transpose[A]], pointers[Inverse[A]],
    PlotRange -> {{-ranger, ranger}, {-ranger, ranger}},
    PlotLabel -> "Hits with A, A^-1 and A^t",
    DisplayFunction -> $DisplayFunction];
```



Rerun many times observing that each time all the arrowheads are all pointing in the clockwise direction
or all the arrowheads are all pointing in the counterclockwise direction
Try to explain why it had to turn out that way.

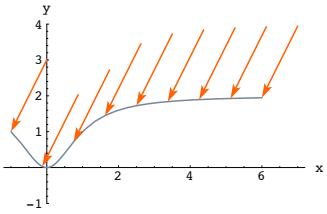
G.8) Parabolic, spherical, elliptic and hyperbolic reflectors; stealth technology

□G.8.a.i) Bouncing light rays off a curve

Here's a curve and parallel light rays coming from a distant source:

```
Clear[x, y, t, ray];
{x[t_], y[t_]} = {t, 2 t^2 / (1 + t^2)};
{tlow, thigh} = {-1, 6};
jump = (thigh - tlow) / 8;
```

```
curveplot = ParametricPlot[{x[t], y[t]}, {t, tlow, thigh},
  PlotStyle → {{LightSlateGray, Thickness[0.005]}},
  PlotRange → {-1, 4}, AspectRatio → Automatic,
  AxesLabel → {"x", "y"}, DisplayFunction → Identity];
ray[t_] = {-1, -2};
rayplots = Table[Arrow[ray[t], Tail → {x[t], y[t]} - ray[t],
    VectorColor → CadmiumOrange], {t, tlow, thigh, jump}];
setup = Show[curveplot, rayplots,
   DisplayFunction → $DisplayFunction];
```



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

□G.8.a.ii) Stealth technology

Look carefully at the plot you did in part i) and think of the incoming rays as radar beams coming from a distant radar station and think of the curve as part the skin of an airplane.

Which part of the skin is more likely to send reflected radar beams back to the radar station: The curved part or the flat part?

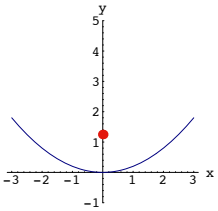
Why do you think stealth aircraft tend to have skin made of flat panels rather than nice streamlined curves?

□G.8.b.i) Parabolic reflectors

To find the focus of the parabola
$y = a x^2$,
you put
$\frac{1}{4p} = a$; so that $p = \frac{1}{(4a)}$
and then say
$\{0, p\} = \{0, \frac{1}{4a}\}$
is the focus of the parabola.
Here's such a parabola and its focus:

```
Clear[x, y, t, ray];
a = 0.2;
{x[t_], y[t_]} = {t, a t^2};
focus = {0, 1 / (4 a)};
```

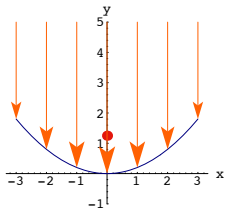
```
{tlow, thigh} = {-3, 3};
curveplot = ParametricPlot[{x[t], y[t]},
  {t, tlow, thigh}, PlotStyle → {{NavyBlue, Thickness[0.005]}},
  PlotRange → {-1, 5}, AspectRatio → Automatic,
  AxesLabel → {"x", "y"}, DisplayFunction → Identity];
focusplot = Graphics[
   {DeepCadmiumRed, PointSize[0.05], Point[focus]}];
Show[curveplot, focusplot, DisplayFunction → $DisplayFunction];
```



Here come parallel vertical light rays from above:

```
Clear[ray];
ray[t_] = {x[t], y[t]} - {x[t], 5};
jump = (thigh - tlow) / 6
rayplots =
  Table[Arrow[ray[t], Tail → {x[t], 5}, VectorColor → CadmiumOrange],
   {t, tlow, thigh, jump}];
setup = Show[curveplot, focusplot, rayplots,
   DisplayFunction → $DisplayFunction];
```

1



Your job is to plot the reflected light rays. Show your reflected rays together with the focus and say how the focus is related to the reflected rays.

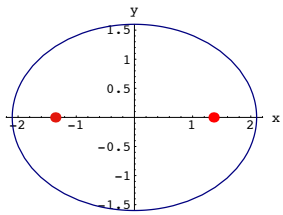
□G.8.b.ii) Uses of parabolic reflectors

Use your plot from part i) to help to explain why TV dish antennas shaped in the form of parabolas.
Why are parabolic reflectors often used in automobile headlights and search lights?

□G.8.c.i) Elliptical reflectors

Given $a > b > 0$, to find the focus of the ellipse
$(\frac{x}{a})^2 + (\frac{y}{b})^2 = 1$,
you put
$p = \sqrt{a^2 - b^2}$
and then say
$\{p, 0\} = \{-p, 0\}$
are the two focuses of the ellipse.
Here's such an ellipse and its two focuses:

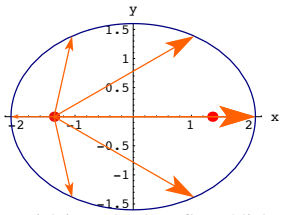
```
Clear[x, y, t, ray];
a = 2.1;
b = 1.6;
focus1 = {Sqrt[a^2 - b^2], 0};
focus2 = {-Sqrt[a^2 - b^2], 0};
{x[t_], y[t_]} = {a Cos[t], b Sin[t]};
{tlow, thigh} = {0, 2 π};
curveplot = ParametricPlot[{x[t], y[t]},
  {t, tlow, thigh}, PlotStyle → {{NavyBlue, Thickness[0.005]}},
  PlotRange → {-b, b}, AspectRatio → Automatic,
  AxesLabel → {"x", "y"}, DisplayFunction → Identity];
focusplot = {Graphics[{Red, PointSize[0.04], Point[focus1]}],
   Graphics[{DeepCadmiumRed, PointSize[0.04], Point[focus2]}]};
Show[curveplot, focusplot, DisplayFunction → $DisplayFunction];
```



Here come light rays emanating from focus2:

```
Clear[ray];
ray[t_] = {x[t], y[t]} - focus2;
rayplots =
  Table[Arrow[ray[t], Tail → focus2, VectorColor → CadmiumOrange],
```

```
    {t, tlow, thigh, (thigh - tlow)/6}];
setup = Show[curveplot, focusplot, rayplots,
   DisplayFunction → $DisplayFunction];
```



Your job is to plot the reflected light rays. Show your reflected rays together with the both focuses and say how the other focus is related to the reflected rays.

□G.8.c.ii) Uses of elliptical reflectors

The ceiling of the "whispering gallery" in the United States Capitol building in Washington, DC is in the form of an elliptical dish. Tourists are often surprised that they can sometimes hear very clearly what strangers well across the room are saying.
Use your plot from part i) to explain this spooky phenom.

BTW: Medical doctors use the reflection property of the ellipse to try to break up kidney stones. The patient is positioned inside an ellipse so that the kidney stone is at one focus. Then small shock waves are sent out from the other focus. The ellipse concentrates the small shock waves where they will do the most good - on the kidney stone.

□G.8.d) Hyperbolic reflectors

Given $a > b > 0$, to find the focus of the hyperbola

$(\frac{x}{a})^2 - (\frac{y}{b})^2 = 1,$

you put

$p = \sqrt{a^2 + b^2}$

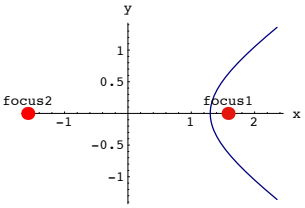
and then say

{p,0} = {-p,0}

are the two focuses of the hyperbola.
Here's such a hyperbola and its focuses:

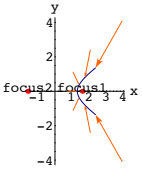
```
Clear[x, y, t, ray];
a = 1.3;
b = 0.9;
focus1 = {√(a² + b²), 0};
focus2 = {-√(a² + b²), 0};
{x[t_], y[t_]} = {a Cosh[t], b Sinh[t]};
{tlow, thigh} = {-1.2, 1.2};
```

```
curveplot = ParametricPlot[{x[t], y[t]},
  {t, tlow, thigh}, PlotStyle → {{NavyBlue, Thickness[0.005]}},
  PlotRange → All, AspectRatio → Automatic,
  AxesLabel → {"x", "y"}, DisplayFunction → Identity];
h = 0.2;
focusplot =
  {Graphics[{DeepCadmiumRed, PointSize[0.05], Point[focus1]}],
   Graphics[{Red, PointSize[0.05], Point[focus2]}],
   Graphics[Text["focus1", focus1 + {0, h}]],
   Graphics[Text["focus2", focus2 + {0, h}]]};
Show[curveplot, focusplot, DisplayFunction → $DisplayFunction];
```



Here come light rays all aimed at focus1:

```
Clear[ray];
ray[t_] = 2 (focus1 - {x[t], y[t]});
rayplots = Table[Arrow[ray[t], Tail → focus1 - 1.5 ray[t],
    VectorColor → CadmiumOrange], {t, tlow, thigh, (thigh - tlow)/6}];
setup = Show[curveplot, focusplot, rayplots,
   DisplayFunction → $DisplayFunction];
```



Your job is to plot the reflected light rays. Show your reflected rays together with the both focuses and say how the other focus is related to the reflected rays.

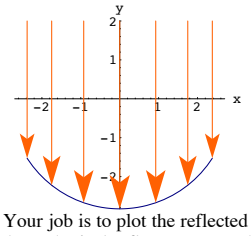
□G.8.e) Spherical reflectors

Here's part of the circle

$x^2 + y^2 = 8$

shown together with vertical light rays coming in from above:

```
Clear[x, y, t, ray];
r = √8;
{x[t_], y[t_]} = -{r Sin[t], r Cos[t]};
{tlow, thigh} = {-1, 1};
curveplot = ParametricPlot[{x[t], y[t]},
  {t, tlow, thigh}, PlotStyle → {{NavyBlue, Thickness[0.005]}},
  PlotRange → {-r, 2}, AspectRatio → Automatic,
  AxesLabel → {"x", "y"}, DisplayFunction → Identity];
Clear[ray];
ray[t_] = {x[t], y[t]} - {x[t], 2};
jump = (thigh - tlow)/6;
rayplots =
  Table[Arrow[ray[t], Tail → {x[t], 2}, VectorColor → CadmiumOrange],
   {t, tlow, thigh, jump}];
Show[curveplot, rayplots, DisplayFunction → $DisplayFunction];
```



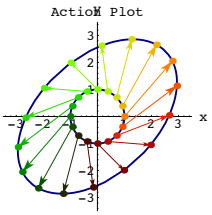
Your job is to plot the reflected light rays. And then answer this question:
Are spherical reflectors as good at concentrating light as parabolic reflectors?

G.9) Matrix Norm: Using the xstretch and ystretch factors to plotting advantage

□G.9.a) The advantage of setting ranger = Max[{xstretch,ystretch,1.01}]

Here's an action movie for a random matrix hitting on the unit circle:

```
Clear[alignerframe];
s = Random[Real, {-N[π/2], N[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, {0, 5.0}], Random[Real, {0, 5.0}]};
stretcher = DiagonalMatrix[{xstretch, ystretch}];
Clear[hangerframe];
s = Random[Real, {-N[π/2], N[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;
ranger = Max[{xstretch, ystretch, 1.01}];
Clear[x, y, t];
{tlow, thigh} = {0, 2 π};
{x[t_], y[t_]} = {Cos[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)/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}];

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]],
  actionarrows[A], hitplotter[A], hitpointplotter[A],
  PlotLabel → "Action Plot", DisplayFunction → $DisplayFunction];
```



Rerun several times and note that the plot ranges on the x-axis and on the y-axis change as the matrix changes.

Reason: Each time you run the code the plot range is automatically set by putting

`ranger = Max[{xstretch,ystretch,1.01}]` (= the maximum of xstretch, ystretch and 1.01)

and putting

`PlotRange →{{-ranger,ranger},{-ranger,ranger}}` .

Explain these two statements:

▪ If you put ranger any number much smaller than this, you'll run the risk of losing part of at least one plot.

Put answer here:

▪ If you put ranger any number much bigger than this, you'll get more wasted space in all of your plots.

Put answer here:

```
FYI: Lots of the advanced  folks like to call
          Max[{xstretch,ystretch}]
          by the name "matrix norm."
```