Matrices, Geometry&*Mathematica*

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

Producer: Bruce Carpenter

Publisher: Math Everywhere, Inc.

MGM.02 2D Matrix Action
BASICS

B.1) 2D matrix hits via rows and via columns:

A.{x, y} = {row1.{x, y}, row2.{x, y}}

and

A.{x, y} = x column1 + y column2

give you the same results

□B.1.a.i) Rows

Here is a sample 2D matrix:

$$A = \begin{pmatrix} 1.6 & 0.4 \\ 1.5 & 2.9 \end{pmatrix};$$

```
MatrixForm[A]
```

$$\begin{pmatrix} 1.6 & 0.4 \\ 1.5 & 2.9 \end{pmatrix}$$

What do folks mean when they talk about the horizontal rows of A?

□**Answer:**

Look at A again:

$$A = \begin{pmatrix} 1.6 & 0.4 \\ 1.5 & 2.9 \end{pmatrix};$$

```
MatrixForm[A]
```

$$\begin{pmatrix} 1.6 & 0.4 \\ 1.5 & 2.9 \end{pmatrix}$$

Folks say that the first horizontal row of A is

row1 = {1.6, 0.4}

and the second horizontal row of A is

row2 = {1.5, 2.9}

Mathematica can fish them out for you:

```
row1 = A[[1]]
{1.6, 0.4}
row2 = A[[2]]
{1.5, 2.9}
```

□B.1.a.ii) Columns

Here is a new sample 2D matrix:

$$A = \begin{pmatrix} -1.7 & 1.4 \\ 0.6 & -4.8 \end{pmatrix};$$

```
MatrixForm[A]
```

$$\begin{pmatrix} -1.7 & 1.4 \\ 0.6 & -4.8 \end{pmatrix}$$

What do folks mean when they talk about the vertical columns of A?

□**Answer:**

Look at A again:

$$A = \begin{pmatrix} -1.7 & 1.4 \\ 0.6 & -4.8 \end{pmatrix};$$

```
MatrixForm[A]
```

$$\begin{pmatrix} -1.7 & 1.4 \\ 0.6 & -4.8 \end{pmatrix}$$

Folks say that the first vertical column of A is

column1 = {−1.7, 0.6}

and the second horizontal vertical column of A is

column2 = {1.4, −4.8}

Mathematica can fish them out for you:

```
column1 = Transpose[A][[1]]
{-1.7, 0.6}
column2 = Transpose[A][[2]]
{1.4, -4.8}
```

□B.1.a.iii) Matrix hits via rows and via columns:

A.{x, y} = {row1.{x, y}, row2.{x, y}}

and

A.{x, y} = x column1 + y column2

give you the same result

Here is a new sample 2D matrix:

$$A = \begin{pmatrix} 1.5 & 3. \\ 0.6 & -1.7 \end{pmatrix};$$

```
MatrixForm[A]
```

$$\begin{pmatrix} 1.5 & 3. \\ 0.6 & -1.7 \end{pmatrix}$$

Here is *Mathematica*'s calculation of what you get when you hit A on a cleared point {x,y}

```
Clear[x, y];
A.{x, y}
{1.5 x + 3. y, 0.6 x - 1.7 y}
  Some folks say that A.{x,y} is what you get when you multiply {x,y} by A.
```

Explain what happened in terms of the rows and columns of A.

□**Answer:**

Rows:

Look at A again:

$$A = \begin{pmatrix} 1.5 & 3. \\ 0.6 & -1.7 \end{pmatrix};$$

```
MatrixForm[A]
```

$$\begin{pmatrix} 1.5 & 3. \\ 0.6 & -1.7 \end{pmatrix}$$

To see how to get A.{x,y} in terms of rows,look at A.{x,y} again:

```
A.{x, y}
{1.5 x + 3. y, 0.6 x - 1.7 y}
```

Fish out the horizontal rows of A:

```
row1 = A[[1]]
{1.5, 3.}
row2 = A[[2]]
{0.6, -1.7}
```

In terms of rows, A.{x,y} is given by

A.{x, y} = {row1.{x, y}, row2.{x, y}}

```
A.{x, y} == {(row1.{x, y}), (row2.{x, y})}
True
```

Columns:

To see how to get A.{x,y} in terms of columns, look at A

```
MatrixForm[A]
```

$$\begin{pmatrix} 1.5 & 3. \\ 0.6 & -1.7 \end{pmatrix}$$

Look at A.{x,y} again:

```
A.{x, y}
{1.5 x + 3. y, 0.6 x - 1.7 y}
```

Fish out the vertical columns of A:

```
column1 = Transpose[A][[1]]
{1.5, 0.6}
column2 = Transpose[A][[2]]
{3., -1.7}
```

In terms of columns, A.{x,y} is given by

A.{x, y} = x column1 + y column2:

```
A.{x, y} == x column1 + y column2
True
```

□B.1.b) Hitting a matrix on the unit circle

What do folks like to do with matrix hits?

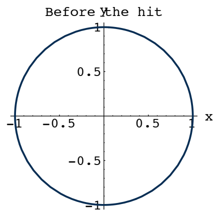
□**Answer:**

Folks like to hit matrices on curves.

A sample:

Here's the unit circle centered at {0.0}:

```
Clear[x, y, t];
{x[t_], y[t_]} = {Cos[t], Sin[t]};
{tlow, thigh} = {-π, π};

curveplot = ParametricPlot[{x[t], y[t]}, {t, tlow, thigh},
   PlotStyle → {{Thickness[0.01], Indigo}}, AspectRatio → Automatic,
   AxesLabel → {"x", "y"}, PlotLabel → "Before the hit"];
```



Here's a matrix A:

```
A = ( 0.5   0.9
     -0.3   0.4 );
MatrixForm[A]
```

```
( 0.5   0.9
 -0.3   0.4 )
```

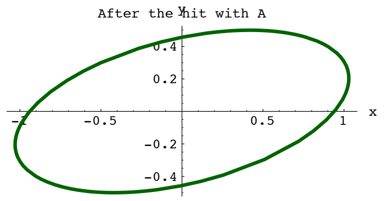
When you hit a point {x[t],y[t]} on the unit circle with A you get:

```
A.{x[t], y[t]}
```

```
{0.5 Cos[t] + 0.9 Sin[t], -0.3 Cos[t] + 0.4 Sin[t]}
```

See how A.{x[t],y[t]} plots out:

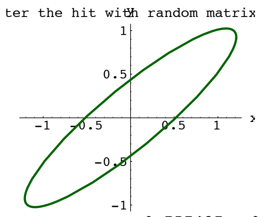
```
hitcurveplot = ParametricPlot[A.{x[t], y[t]}, {t, tlow, thigh},
PlotStyle -> {{Thickness[0.01], GosiaGreen}},
  AspectRatio -> Automatic,
AxesLabel -> {"x", "y"}, PlotLabel -> "After the hit with A"];
```



After you hit the unit circle with A, you get a tilted ellipse.

Try it for random matrices A:

```
A = ( Random[Real, {-1, 1}]  Random[Real, {-1, 1}]
      Random[Real, {-1, 1}]  Random[Real, {-1, 1}] );
hitcurveplot = ParametricPlot[A.{x[t], y[t]}, {t, tlow, thigh},
PlotStyle -> {{Thickness[0.01], GosiaGreen}},
  AspectRatio -> Automatic,
AxesLabel -> {"x", "y"},
  PlotLabel -> "After the hit with random matrix A"];
"Here A is" MatrixForm[A]
```

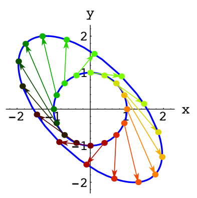


```
Here A is ( 0.757427  -0.943913
            0.236445  -0.996578 )
```

Rerun many times,
letting your eyes drink in what matrix hits do to the unit circle.

You get an ellipse every time.

B.2) Matrix Action movies:

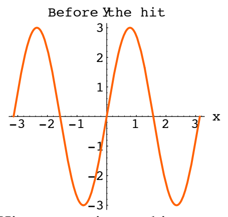


Hitting with a 2D matrix on curves and assessing the result

□B.2.a.i) Hitting a curve with a matrix

Here's an ordinary sine wave plotted in true scale:

```
Clear[x, y, t];
{x[t_], y[t_]} = {t, 3 Sin[2 t]};
{tlow, thigh} = {-π, π};

curveplot = ParametricPlot[{x[t], y[t]}, {t, tlow, thigh},
  PlotStyle → {{Thickness[0.01], CadmiumOrange}},
  AspectRatio → Automatic, AxesLabel → {"x", "y"},
  PlotLabel → "Before the hit"];
```



Hit every point on this curve with the 2D matrix

$$A = \begin{pmatrix} 0.5 & 0.9 \\ -0.8 & 1.0 \end{pmatrix}$$

and plot the result.

Describe what you see.

□**Answer:**

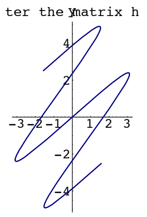
When you hit a point {x[t], y[t]} on the plotted curve you get:

```
A = ( 0.5   0.9
     -0.8   1.0 );
A.{x[t], y[t]}
```

```
{0.5 t + 2.7 Sin[2 t], -0.8 t + 3. Sin[2 t]}
```

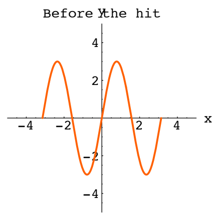
When you hit every point {x[t], y[t]} on the plotted curve and plot the result you get:

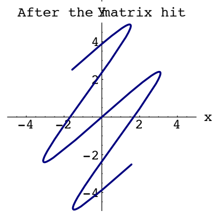
```
A = ( 0.5   0.9
     -0.8   1.0 );
hitcurveplot = ParametricPlot[A.{x[t], y[t]}, {t, tlow, thigh},
PlotStyle -> {{Thickness[0.01], NavyBlue}}, AspectRatio -> Automatic,
AxesLabel -> {"x", "y"}, PlotLabel -> "After the matrix hit"];
```



See both one after the other:

```
ranger = 5;
Show[curveplot,
  PlotRange -> {{-ranger, ranger}, {-ranger, ranger}}];
Show[hitcurveplot, PlotRange ->
    {{-ranger, ranger}, {-ranger, ranger}}];
```





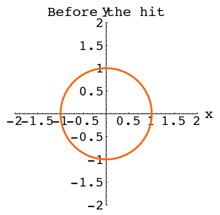
Grab, align and animate both plots.

At this stage, it's hard to describe in precise terms what the hit by A did. But in general terms, you can say that the hit by A simultaneously deformed the curve a bit and then rotated it quite a bit.

□B.2.a.ii) Hitting the unit circle with a matrix

Here's the unit circle plotted in true scale:

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

ranger = 2;
curveplot = ParametricPlot[{x[t], y[t]}, {t, tlow, thigh},
  PlotStyle → {{Thickness[0.01], CadmiumOrange}},
  PlotRange → {{-ranger, ranger}, {-ranger, ranger}},
  AspectRatio → Automatic, AxesLabel → {"x", "y"},
  PlotLabel → "Before the hit"];
```



Hit this every point on this curve with the 2D matrix

$$A = \begin{pmatrix} 0.4 & -0.7 \\ 0.7 & 1.3 \end{pmatrix}$$

and plot the result.
Describe what you see.

□ Answer:

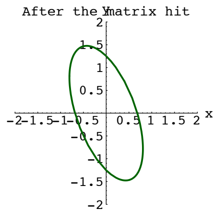
When you hit a point {x[t],y[t]} on the plotted curve you get:

```
A = (0.4  -0.7
     0.7   1.3);
A.{x[t], y[t]}
```

```
{0.4 Cos[t] - 0.7 Sin[t], 0.7 Cos[t] + 1.3 Sin[t]}
```

When you hit every point {x[t],y[t]} on the plotted curve and plot the result you get:

```
hitcurveplot = ParametricPlot[A.{x[t], y[t]}, {t, tlow, thigh},
        PlotStyle → {{Thickness[0.01], GosiaGreen}},
  AspectRatio → Automatic,
  PlotRange → {{-ranger, ranger}, {-ranger, ranger}},
  AxesLabel → {"x", "y"}, PlotLabel → "After the matrix hit"];
```

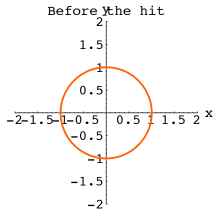


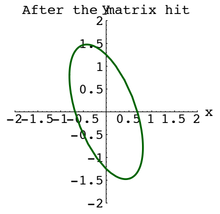
```
Grab,align and animate both plots.
```

An ellipse!

See both one after the other:

```
ranger = 2;
Show[curveplot,

  PlotRange -> {{-ranger, ranger}, {-ranger, ranger}}];
Show[hitcurveplot, PlotRange ->
   {{-ranger, ranger}, {-ranger, ranger}}];
```





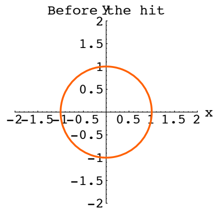
At this stage, it's hard to describe precisely what the hit by A did.

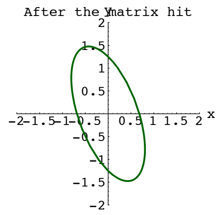
But in general terms, you can say that hit by A simultaneously deformed the curve and maybe A rotated it too.

□B.2.a.iii) Matrix action movies

Look at the two plots from part i):

```
ranger = 2;
Show[curveplot,
  PlotRange -> {{-ranger, ranger}, {-ranger, ranger}}];
Show[hitcurveplot, PlotRange ->
   {{-ranger, ranger}, {-ranger, ranger}}];
```



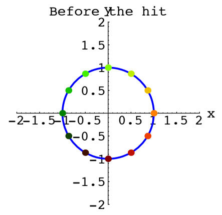


Surely the hit by A deformed the circle, but how can you get a feeling for whether A also rotated the circle?

□ Answer:

Color code the points and watch where they go this way:

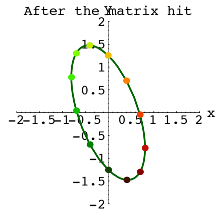
```
Clear[x, y, t];
{tlow, thigh} = {0, 2 π};
ranger = 2;
{x[t_], y[t_]} = {Cos[t], Sin[t]};

pointcolor[t_] = RGBColor[0.5 (Cos[t] + 1), 0.5 (Sin[t] + 1), 0];
jump = (thigh - tlow)/12;

curveplot = ParametricPlot[{x[t], y[t]},
   {t, tlow, thigh}, PlotStyle → {{Thickness[0.01], Blue}},
   PlotRange → {{-ranger, ranger}, {-ranger, ranger}},
   AxesLabel → {"x", "y"}, PlotLabel → "Before the hit",
   DisplayFunction → Identity];

pointplot = Table[
   Graphics[{pointcolor[t], PointSize[0.035], Point[{x[t], y[t]}]}],
   {t, tlow, thigh - jump, jump}];

Show[curveplot, pointplot, DisplayFunction → $DisplayFunction];
```



To see what happens after the hit by A, copy paste and edit the plotting instructions above, replacing all

{x[t],y[t]}'s with A.{x[t],y[t]}:

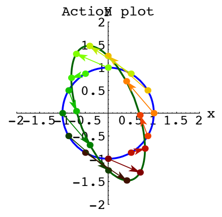
```
hitcurveplot = ParametricPlot[A.{x[t], y[t]}, {t, tlow, thigh},
        PlotStyle → {{Thickness[0.01], GosiaGreen}},
   PlotRange → {{-ranger, ranger}, {-ranger, ranger}},
   AxesLabel → {"x", "y"}, PlotLabel → "After the matrix hit",
   DisplayFunction → Identity];

hitpointplot = Table[Graphics[{pointcolor[t], PointSize[0.035],
      Point[A.{x[t], y[t]}]}], {t, tlow, thigh - jump, jump}];

Show[hitcurveplot, hitpointplot,
   DisplayFunction → $DisplayFunction];
```



```
Grab both plots,align and animate slowly.
```

The following graphic tries to explain what the color coding means:

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

Show[curveplot, hitcurveplot, pointplot, hitpointplot, actionarrows,
   PlotLabel → "Action plot", DisplayFunction → $DisplayFunction];
```



In this plot, each arrow points from a point

{x[t],y[t]}

on the circle to the point

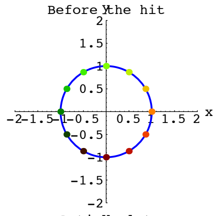
A.{x[t],y[t]}

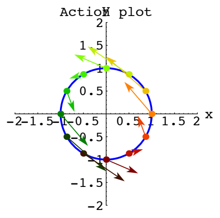
that results from hitting {x[t],y[t]} with A.

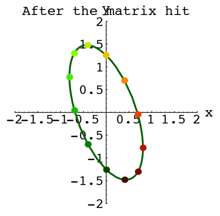
As you can see, there is a strong hint of a rotation, but there is also a big stretch and squeeze.

Here's a review:

```
Show[curveplot, pointplot, DisplayFunction → $DisplayFunction];

Show[curveplot, pointplot, actionarrows,
  PlotLabel → "Action plot", DisplayFunction → $DisplayFunction];

Show[hitcurveplot, hitpointplot,
  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.
```

Matrix action.

□B.2.a.iv) Another matrix action movie

Go with this 2D matrix:

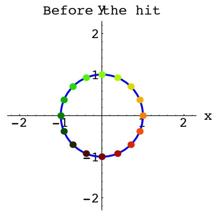
$$A = \begin{pmatrix} 1.77 & 0.85 \\ -1.79 & 0.9 \end{pmatrix}.$$

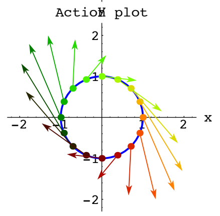
Make an action movie and analyze what a hit with this matrix does to the unit circle.

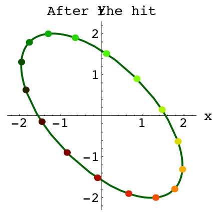
□Answer:

Copy, paste and edit to make an action movie:

```
A = ( 1.77   0.85 );
    ( -1.79  0.9  )
Clear[x, y, t];
{tlow, thigh} = {0, 2 π};
ranger = 2.3;
{x[t_], y[t_]} = {Cos[t], Sin[t]};

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

curveplot = ParametricPlot[{x[t], y[t]},
  {t, tlow, thigh}, PlotStyle → {{Thickness[0.01], Blue}},
  PlotRange → {{-ranger, ranger}, {-ranger, ranger}},
  AxesLabel → {"x", "y"}, PlotLabel → "Before the hit",
  DisplayFunction → Identity];

pointplot = Table[
   Graphics[{pointcolor[t], PointSize[0.035], Point[{x[t], y[t]}]}],
   {t, tlow, thigh - jump, jump}];

hitcurveplot = ParametricPlot[A.{x[t], y[t]},
  {t, tlow, thigh}, PlotStyle → {{Thickness[0.01], GosiaGreen}},
  PlotRange → {{-ranger, ranger}, {-ranger, ranger}},
  AxesLabel → {"x", "y"}, PlotLabel → "After the hit",
  DisplayFunction → Identity];

hitpointplot = Table[Graphics[{pointcolor[t], PointSize[0.035],
     Point[A.{x[t], y[t]}]}], {t, tlow, thigh - jump, jump}];

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

Show[curveplot, pointplot, DisplayFunction → $DisplayFunction];

Show[curveplot, pointplot, actionarrows,
  PlotLabel → "Action plot", DisplayFunction → $DisplayFunction];

Show[hitcurveplot, hitpointplot,
  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.
```

There is a huge hint of simultaneous rotation and there is an equally huge hint of a big stretch, bigger in one direction than another.

□B.2.a.v) Matrix Action movie code

Shorten up the code needed to make a matrix action movie.

□Answer:

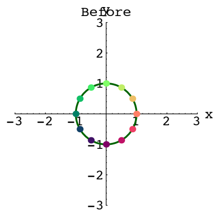
Here is a modest attempt; you might be able to do better.

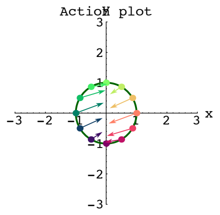
First enter the matrix A and the parametrization for {x[t],y[t]} and run:

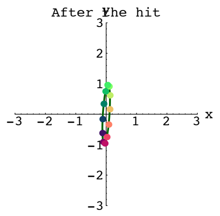
```
The code below goes with a random matrix.
```

```
A = ( Random[Real, {-1, 1}]  Random[Real, {-1, 1}] );
    ( Random[Real, {-1, 1}]  Random[Real, {-1, 1}] )
Clear[x, y, t, hitplotter,
  hitpointplotter, pointcolor, actionarrows, matrix2D];
{tlow, thigh} = {0, 2 π};

ranger = 3;
{x[t_], y[t_]} = {Cos[t], Sin[t]};

pointcolor[t_] = RGBColor[0.5 (Cos[t] + 1), 0.5 (Sin[t] + 1), 0.4];
        thigh - tlow
jump = ─────────────;
             12

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

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

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];
```



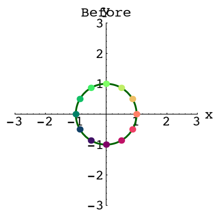


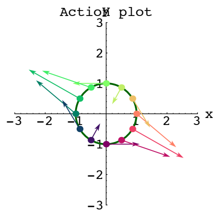


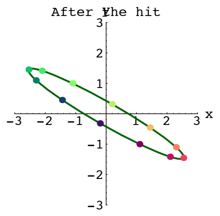
Grab all three plots, align and animate slowly.
To get maximum viewing pleasure and maximum information,
look at one point and follow it.
Then pick another point and follow it.
Keep doing this until you have a good visual grasp of the action.
Then rerun several times.

You can change the matrix A:

```
A = (2.3  -1.1
     -1.1  1.0);

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

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

after = Show[hitplotter[A], hitpointplotter[A],
   PlotLabel → "After the hit", 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.

Play.

B.3) Making your own 2D matrices: The identity matrix $\begin{pmatrix}1&0\\0&1\end{pmatrix}$ and the xy-stretchers $\begin{pmatrix}\text{xstretch}&0\\0&\text{ystretch}\end{pmatrix}$

□B.3.a) The 2D identity matrix $\begin{pmatrix}1&0\\0&1\end{pmatrix}$

Here's the 2D identity matrix:

```
MatrixForm[IdentityMatrix[2]]
```

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

```
(1  0
 0  1)
```

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

Why do nearly all folks like to call this matrix by the name "identity matrix?"

□**Answer:**

Folks like to call this matrix by the name identity matrix because when you hit {x,y} with this matrix, absolutely nothing new happens:

```
Clear[x, y];
IdentityMatrix[2].{x, y}
```

```
{x, y}
```

Hitting with the identity matrix is just like multiplying by 1.

This happens because

$$\begin{pmatrix}1&0\\0&1\end{pmatrix}.\{x,y\} = x\{1,0\} + y\{0,1\} = \{x,0\} + \{0,y\} = \{x,y\}.$$

□B.3.b.i) Diagonal matrices: The xy-stretcher matrices $\begin{pmatrix}\text{xstretch}&0\\0&\text{ystretch}\end{pmatrix}$

Here's a cleared 2D diagonal matrix:

```
Clear[a, d];
DiagonalMatrix[{a, d}]
```

```
{{a, 0}, {0, d}}
```

```
MatrixForm[DiagonalMatrix[{a, d}]]
```

$\begin{pmatrix}a&0\\0&d\end{pmatrix}$

Nearly all folks like to call this matrix by the name "diagonal matrix because all its non-zero entries lie on what the same folks call the main diagonal.
Why do some folks (especially clued-in eager folks such as you) also like to call 2D diagonal matrices by the name xy-stretchers?

□**Answer:**

Here's a 2D diagonal matrix: and an action movie showing what it does to the unit circle:

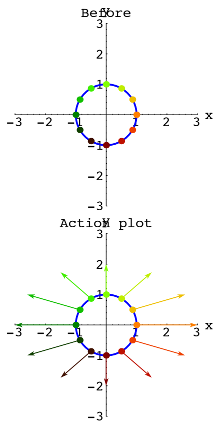
```
xstretch = 3.0;
ystretch = 2.0;
A = DiagonalMatrix[{xstretch, ystretch}];
MatrixForm[A]
```

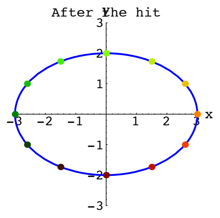
$\begin{pmatrix}3.&0\\0&2.\end{pmatrix}$

And here's an action movie 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 = 3;
{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], Blue}},
   PlotRange → {{-ranger, ranger}, {-ranger, ranger}},
   AxesLabel → {"x", "y"}, DisplayFunction → Identity];

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

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

When you hit the unit circle with:

```
MatrixForm[A]
```

$$\begin{pmatrix} 3. & 0 \\ 0 & 2. \end{pmatrix}$$

you stretched out all measurements

along the x-axis by a factor of xstretch = 3

and all measurements

along the y-axis by a factor of ystretch = 2.

That's why it's hard to resist the urge to call diagonal matrices like this one by the name "xystretchers.".

Check it out for cleared stretch factors:

```
Clear[xstretch, ystretch];
A = DiagonalMatrix[{xstretch, ystretch}];
MatrixForm[A]
```

$$\begin{pmatrix} \text{xstretch} & 0 \\ 0 & \text{ystretch} \end{pmatrix}$$

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

```
{x xstretch, y ystretch}
```

This happens because

$$\begin{pmatrix} \text{xstretch} & 0 \\ 0 & \text{ystretch} \end{pmatrix}.\{x,y\} = x\{\text{xstretch},0\} + y\{0,\text{ystretch}\}$$

$$= \{\text{xstretch}\ \ x, 0\} + y\ \{0,\text{ystretch}\ y\}$$

$$= \{\text{xstretch}\ \ x\ \ , \text{ystretch}\ y\}.$$

□B.3.b.ii) Making your own xy-stretchers

Make an xy-stretcher matrix that stretches by a factor of 1.5 in the direction of the x-axis and shrinks by a factor of 0.4 direction of the y-axis.
Show off your answer with an action movie.

□ Answer:

The matrix you want is:

```
{xstretch, ystretch} = {1.5, 0.4};
A = DiagonalMatrix[{xstretch, ystretch}];
MatrixForm[A]
```

$$\begin{pmatrix} 1.5 & 0 \\ 0 & 0.4 \end{pmatrix}$$

Check what it does to {x,y}:

```
A.{x, y}
```

```
{1.5 x, 0.4 y}
```

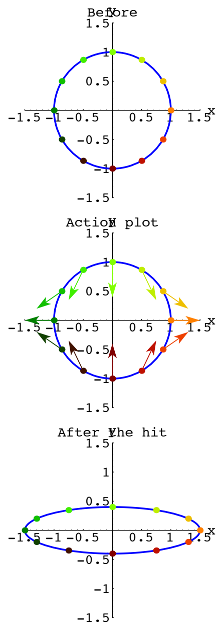
Hits with A stretch by a factor of 1.5 in the direction of the x-axis and shrinks by a factor of 0.4 in the direction of the y-axis.

Just what it was made to do.

See this matrix do its work:

```
ranger = Max[{xstretch, ystretch, 1}]
Clear[x, y, t];
{tlow, thigh} = {0, 2 π};

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

```
1.5
```



```
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.
```

In this case, one of the stretching factors is actually a shrinking factor.

Play with your own choices of xstretch and ystretch.

B.4) Making your own 2D matrices:

Using perpendicular frames to make matrices for hanging, aligning and rotating.

To align, you load the perpendicular frame into the rows

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

To hang, you load the perpendicular frame into the columns

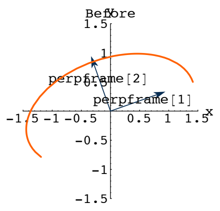
$$\text{hanger} = \begin{pmatrix} \text{perpframe[1]} & \text{perpframe[2]} \\ \downarrow & \downarrow \end{pmatrix}.$$

To rotate, you hang on right hand perpendicular frame

□ B.4.a.i) Making aligner matrices to align on the x and y axes

Here's an ellipse hung on a perpendicular frame:

```
s = 0.34;
Clear[perpframe];
{perpframe[1], perpframe[2]} =
  {{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}};

Clear[x, y, t];
{x[t_], y[t_]} = 1.5 Cos[t] perpframe[1] + 0.9 Sin[t] perpframe[2];

ranger = 1.5;
{tlow, thigh} = {0, 9 π/8};

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

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

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



Now make a matrix (called aligner)

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

with

perpframe[1] in the first horizontal row

and

perpframe[2] in the second horizontal row.

Here's how you do it:

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

```
( 0.942755   0.333487 )
( -0.333487  0.942755 )
```

Compare

```
perpframe[1]
perpframe[2]
```

```
{0.942755, 0.333487}
{-0.333487, 0.942755}
```

Sure enough,

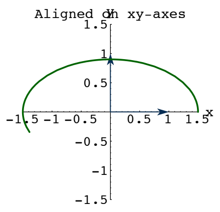
perpframe[1] is in the first horizontal row of aligner

and

perpframe[2] is in the second horizontal row of aligner.

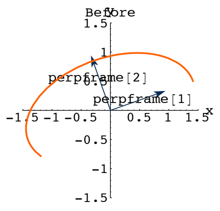
Now see what happens when you hit this matrix on all points {x[t],y[t]} of the plotted curve:

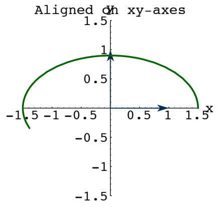
```
hitcurveplot =
 ParametricPlot[aligner.{x[t], y[t]}, {t, tlow, thigh},
PlotStyle -> {{GosiaGreen, Thickness[0.01]}},
  DisplayFunction -> Identity];

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


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



Review both plots:

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





Grab both plots and animate.

The hit with aligner aligns the curve on the x and y axes with

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

and with

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

Explain why this was guaranteed to happen no matter what perpendicular frame you start with.

□ Answer:

The matrix (called aligner)

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

was made with

perpframe[1] in the first horizontal row

and

perpframe[2] in the second horizontal row.

According to B.1) when you hit {x[t], y[t]} with

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

you get

$$\text{aligner} .\{x[t],y[t]\} = \begin{pmatrix} \text{perpframe[1]} \rightarrow \\ \text{perpframe[2]} \rightarrow \end{pmatrix}.\{x[t], y[t]\}$$

$$= \{(\{x[t], y[t]\}.\ \text{perpframe[1]}), (\{x[t], y[t]\}.\ \text{perpframe[2]})\}$$

These are the coordinates of {x[t],y[t]} relative to the perpendicular frame.

You worked with these in the last lesson.

And, according to the last lesson, this exactly what you what you plot to take a curve {x[t], y[t]} and align it on the x and y -axes with

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

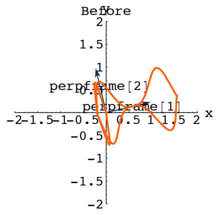
and with

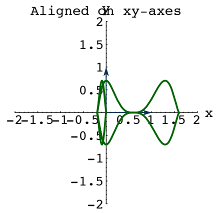
{0, 1} pointing out the positive y-axis playing the former role of perpframe[2].

See it happen for some random perpendicular frames and random curves:

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

Clear[x, y, t];
{x[t_], y[t_]} =
  0.8 (Cos[t] - 1) Cos[Random[Integer, {1, 3}] t] perpframe[1] +
   0.7 Sin[Random[Integer, {1, 5}] t]^3 perpframe[2];
ranger = 2;
{tlow, thigh} = {0, 2 π};

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

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

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

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

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





```
Grab both plots and animate.
```

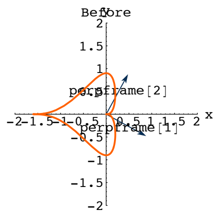
Rerun a couple of times, always noticing that the long axis of the curve lines up on the x-axis.

That's because the long axis of the original curve lined up on perpframe[1].

□B.4.b) Making hanger matrices to hang curves on perpendicular frames

Here's a curve shown with a perpendicular frame:

```
Clear[x, y, t];
{x[t_], y[t_]} = {0.8 (1 - Cos[t]) Cos[t], 0.9 Sin[t]^5};
ranger = 2;
{tlow, thigh} = {0, 2 π};

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

s = -0.5;
Clear[perpframe];
{perpframe[1], perpframe[2]} =
  {{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}};

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

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



Now make a matrix (called hanger)

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

with

perpframe[1] in the first vertical column

and

perpframe[2] in the second vertical column.

Here's how you do it:

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

$$\begin{pmatrix} 0.877583 & 0.479426 \\ -0.479426 & 0.877583 \end{pmatrix}$$

Compare

```
perpframe[1]
perpframe[2]
```

```
{0.877583, -0.479426}
{0.479426, 0.877583}
```

Sure enough,

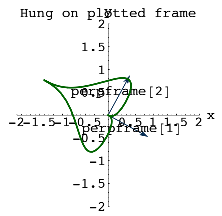
perpframe[1] is in the first vertical column of hanger

and

perpframe[2] is in the second vertical column of hanger.

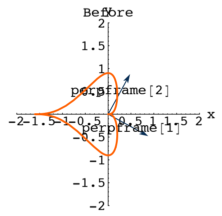
Now see what happens when you hit this hanger matrix on all points {x[t],y[t]} of the plotted curve:

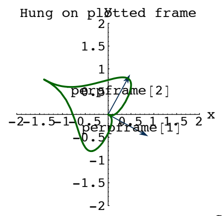
```
hitcurveplot = ParametricPlot[hanger.{x[t], y[t]}, {t, tlow, thigh},
PlotStyle -> {{GosiaGreen, Thickness[0.01]}},
  DisplayFunction -> Identity];

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



Review both plots:

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





```
Grab both plots and animate slowly.
```

The hit with hanger hangs the ellipse on the plotted perpendicular frame with

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

and with

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

Explain why this was guaranteed to happen no matter what perpendicular frame you started with.

□Answer:

The matrix (called hanger)

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

was made with

perpframe[1] in the first vertical column

and

perpframe[2] in the second vertical column

According to B.1) when you hit {x[t], y[t]} with

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

you get

$$\text{hanger.}\{x[t],y[t]\} = \begin{pmatrix} \text{prepframe[1]} & \text{perpframe[2]} \\ \downarrow & \downarrow \end{pmatrix}.\{x[t],\ y[t]\}$$

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

and this is exactly the formula used in the last lesson to take a curve {x[t], y[t]} and hang it on

{perpframe[1], perpframe[2]}

with

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

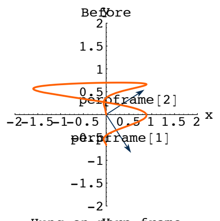
and with

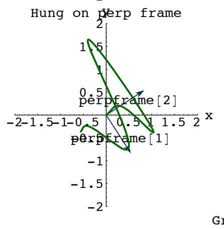
perpframe[2] playing the former role {0, 1} pointing out the positive y-axis.

See it happen for some random perpendicular frames and random curves:

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

Clear[x, y, t];
{x[t_], y[t_]} = {0.8 (Cos[t] - 1) Cos[Random[Integer, {1, 3}] t],
   0.7 Sin[0.7 Random[Integer, {1, 3}] t]};
ranger = 2;
{tlow, thigh} = {0, 2 π};

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

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

before = Show[frameplot, curveplot,
```

```
   PlotRange → {{-ranger, ranger}, {-ranger, ranger}},
   Axes → True, AxesLabel → {"x", "y"}, PlotLabel → "Before"];

aligner = {perpframe[1], perpframe[2]};
hitcurveplot = ParametricPlot[hanger.{x[t], y[t]},
  {t, tlow, thigh}, PlotStyle → {{GosiaGreen, Thickness[0.01]}},
  DisplayFunction → Identity];

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





```
Grab both plots and animate slowly.
```

Rerun a couple of times, always noticing that the long axis of the ellipse lines up with perpframe[1].
That's because the long axis of the original ellipse lined up on the x-axis.

B.5) Matrix multiplication:

A hit with A.B is the same a hit with B followed by a hit with A .

To get the first column of A.B , you hit A on the first column of B.

To get the second column of A.B , you hit A on the second column of B.

Why you shouldn't expect that A.B = B.A

□B.5.a.i) A hit with A.B is the same a hit with B followed by a hit with A

Here is a cleared 2D matrix:

```
Clear[a, b, c, d];
A = (a  b
     c  d);
MatrixForm[A]
```

```
(a  b
 c  d)
```

Here's another cleared 2D matrix:

```
Clear[r, s, t, v];
B = (r  s
     t  v);
MatrixForm[B]
```

```
(r  s
 t  v)
```

Here is *Mathematica*'s calculation of A times B:

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

```
(a r + b t   a s + b v
 c r + d t   c s + d v)
```

Maybe not what you expected.
Why do almost all folks agree that this is the right way to multiply matrices?

□Answer:

Folks all across the world have agreed on this way of matrix multiplication so that a hit with A.B is the same as a hit with B followed by a hit with A.

Watch it happen:

```
Clear[x, y];
hitbyB = B.{x, y}
```

```
{r x + s y, t x + v y}
```

```
hitbyBfollowedbyhitbyA = Expand[A.hitbyB]
```

```
{a r x + b t x + a s y + b v y, c r x + d t x + c s y + d v y}
```

```
hitbyAB = Expand[(A.B).{x, y}]
```

```
{a r x + b t x + a s y + b v y, c r x + d t x + c s y + d v y}
```

```
Expand[(A.B).{x, y}] == Expand[A.(B.{x, y})]
```

```
True
```

□B.5.a.ii) What matrix multiplication is good for

What's matrix multiplication good for?

□Answer:

You can do two hits at once. By multiplying A.B and then hitting, you get the same result as hitting by B and then hitting by A.

Sometimes you see what A.B does in terms of what A and B do.

To see more on this, go in to part iii)

□B.5.a.iii) To get the first column of A.B, you hit A on the first column of B. To get the second column of A.B, you hit A on the second column of B

Given two 2D matrices A and B, explain how to calculate A.B by hitting A on the columns of B.

□Answer:

To get the first column of A.B, you hit A on the first column of B.

To get the second column of A.B, you hit A on the second column of B.

Try this out on these cleared 2D matrices A and B:

```
Clear[a, b, c, d, r, s, t, v];
A = (a  b
     c  d);
B = (r  s
     t  v);

"A is" MatrixForm[A]

"B is" MatrixForm[B]
```

```
A is (a  b
      c  d)

B is (r  s
      t  v)
```

The first vertical column of B is

```
Clear[Bcol];
Bcol[1] = {r, t}
```

```
{r, t}
```

The second vertical column of B is

```
Bcol[2] = {s, v}
```

```
{s, v}
```

Here's what you get when you hit A on the first column of B:

```
A.Bcol[1]
```

```
{a r + b t, c r + d t}
```

That;s the same as the vector that's in first column of A.B:

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

$$\begin{pmatrix} a\,r + b\,t & a\,s + b\,v \\ c\,r + d\,t & c\,s + d\,v \end{pmatrix}$$

Here's what you get when you hit A on the second column of B:

```
A.Bcol[2]
```

```
{a s + b v, c s + d v}
```

That's the same as the vector that is in the second column of A.B:

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

$$\begin{pmatrix} a\,r + b\,t & a\,s + b\,v \\ c\,r + d\,t & c\,s + d\,v \end{pmatrix}$$

This confirms this rule:

To get the first column of A.B, you hit A on the first column of B.

To get the second column of A.B, you hit A on the second column of B.

□B.5.a.iv) The action of A.B in terms of the action of B and the action of A

Here's a hanger matrix:

```
s = 0.6;
{perpframe[1], perpframe[2]} =
  N[{{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}}];
hanger = Transpose[{perpframe[1], perpframe[2]}];
MatrixForm[A]
```

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

```
                Hits with this hanger hang on the chosen perpendicular frame.
```

Here's an xystretcher matrix

```
xstretch = 2.0;
ystretch = 0.8;
xystretcher = DiagonalMatrix[{xstretch, ystretch}];
MatrixForm[xystretcher]
```

$$\begin{pmatrix} 2. & 0 \\ 0 & 0.8 \end{pmatrix}$$

```
                         Hits with this xystretcher
 multiply all measurements in the direction of the x axis by 2 and
  multiply all measurements in the direction of the y axis by 0.8
```

Put A = hanger and B = xystretcher and calculate A.B:

```
A = hanger;
B = xystretcher;
MatrixForm[A.B]
```

$$\begin{pmatrix} 1.65067 & -0.451714 \\ 1.12928 & 0.660268 \end{pmatrix}$$

Predict what a hit with A.B does to the unit circle centered at {0,0} and test out your prediction with an action movie.

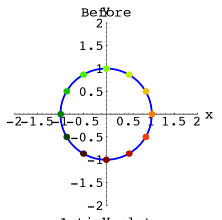
□Answer:

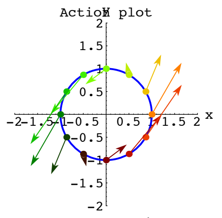
Because hits with B stretch along the x and y axes, a hit with B will stretch the unit circle into and ellipse.

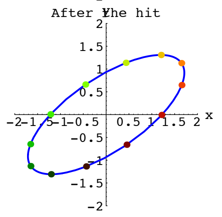
When you follow the hit by B with a hit with the hanger matrix A, you gotta get a tilted ellipse.

Try it out:

```
Clear[x, y, t, hitplotter,
  hitpointplotter, pointcolor, actionarrows, matrix2D];
{tlow, thigh} = {0, 2 π};
ranger = Max[{xstretch, ystretch}];
{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], Blue}},
    PlotRange → {{-ranger, ranger}, {-ranger, ranger}},
    AxesLabel → {"x", "y"}, DisplayFunction → Identity];

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

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

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

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

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



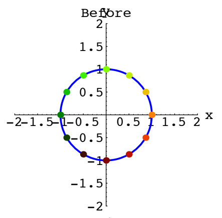




Exactly as predicted.

You can view this in three stages:

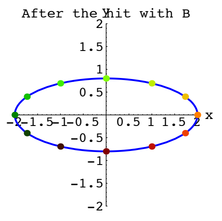
```
stage1 = Show[before]
```



```
- Graphics -
```

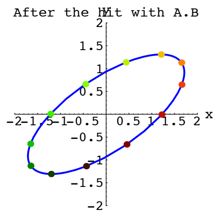
Hitting this circle with A.B is the same as hitting the circle with B and then hitting the result with A. Here's what you get when you hit the circle with B:

```
stage2 = Show[hitplotter[B],
    hitpointplotter[B], PlotLabel → "After the hit with B",
    DisplayFunction → $DisplayFunction];
```



Here's what you get when you hit the ellipse with A:

```
stage3 = Show[hitplotter[A.B],
    hitpointplotter[A.B], PlotLabel → "After the hit with A.B",
    DisplayFunction → $DisplayFunction];
```



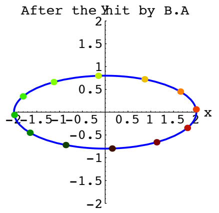
```
Grab the last three plots and animate slowly.
```

□B.5.a.v) Why you can't expect that A.B = B.A

```
Make sure all the code from part iii) above is live.
```

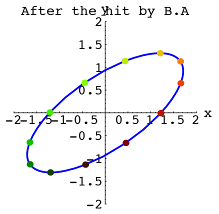
Stay with the same matrices A and B as in the last part and look at the result of hitting the unit circle with B.A

```
BAhitplot = Show[hitplotter[B.A],
   hitpointplotter[B.A], PlotLabel → "After the hit by B.A",
   DisplayFunction → $DisplayFunction];
```



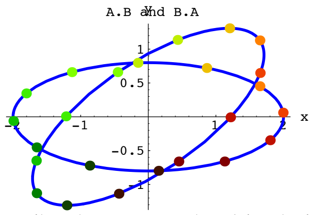
Compare what you get when you hit the unit circle with A.B

```
ABhitplot = Show[hitplotter[A.B],
   hitpointplotter[A.B], PlotLabel → "After the hit by B.A",
   DisplayFunction → $DisplayFunction];
```



See them both:

```
Show[ABhitplot, BAhitplot,
  PlotLabel → " A.B and B.A", PlotRange -> All];
```

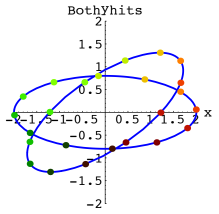


Describe what you see and explain why it happened and explain what it means.

□Answer:

Take another look:

```
Show[ABhitplot, BAhitplot, PlotLabel → "Both hits"];
```



One of these ellipses results from hitting the unit circle with A.B; the other results from hitting the unit circle with B.A.

Because these ellipse are not the same, you are left with the vivid conclusion that

A.B ≠ B.A:

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

$$\begin{pmatrix} 1.65067 & -0.451714 \\ 1.12928 & 0.660268 \end{pmatrix}$$

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

$$\begin{pmatrix} 1.65067 & -1.12928 \\ 0.451714 & 0.660268 \end{pmatrix}$$

This is natural because:

-> The hit with B stretches out along the x and y -axes.

-> The hit with A rotates.

Consequently:

-> The hit with A.B stretches and then rotates,

but

-> The hit with B.A rotates and then stretches.

And the results are not the same.