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

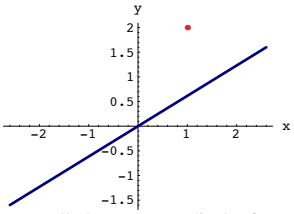
TUTORIALS

T.1) Making perpendicular projections onto lines

◻T.1.a) Projection matrices

Here are a line through {0,0} and a point not on the line plotted in true scale:

```
directionvector = 2 {1.3, 0.8};
givenpoint = {1.0, 2.0};
{tlow, thigh} = {-1, 1};
lineplot = ParametricPlot[t directionvector,
  {t, tlow, thigh}, PlotStyle → {{Thickness[0.01], NavyBlue}},
  AxesLabel → {"x", "y"}, PlotRange → All,
  AspectRatio → Automatic, DisplayFunction -> Identity];
pointplot = Graphics[{PointSize[0.02],
   AlizarinCrimson, Point[givenpoint]}];
setup = Show[lineplot, pointplot,
   DisplayFunction -> $DisplayFunction];
```



Use a well-chosen perpendicular frame to help come up with a matrix A so that

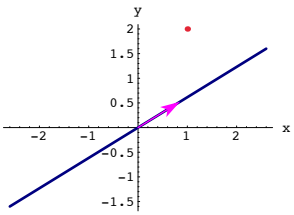
A.givenpoint

is the point on the line that is closest to the given point.

◻Answer:

Make the line's direction vector into a unit vector and plot:

```
Clear[perpframe];
perpframe[1] = directionvector / Sqrt[directionvector.directionvector];
Show[setup, Arrow[perpframe[1],
   Tail → {0, 0}, VectorColor → Magenta, HeadSize → 0.4]];
```



Look at:

```
perpframe[1]
```

```
{0.851658, 0.524097}
```

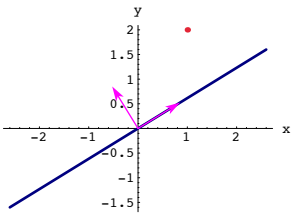
To get a perpendicular unit vector, just reverse the components of perpframe[1] and throw in a minus sign:

```
perpframe[2] = {-perpframe[1][[2]], perpframe[1][[1]]}
```

```
{-0.524097, 0.851658}
```

And plot:

```
frameplot = {Arrow[perpframe[1], Tail → {0, 0},
    VectorColor → Magenta, HeadSize → 0.3], Arrow[perpframe[2],
    Tail → {0, 0}, VectorColor → Magenta, HeadSize → 0.3]};
Show[setup, frameplot];
```



perpframe[1] points in the direction of the line.

To get the point on the line closest to the given point, make a matrix

A = hanger.stretcher.aligner

that preserves lengths in the direction of perpframe[1] but zeroes out all lengths in the direction of perpframe[2]:

This means you want

A.perpframe[1] = 1.0 perframe[1]

A.perpframe[2] = 0.0 perframe[2].

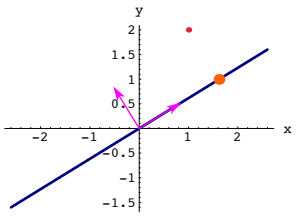
So you go with:

```
aligner = {perpframe[1], perpframe[2]};
{xstretch, ystretch} = {1, 0};
stretcher = ((xstretch 0) (0 ystretch));
hanger = Transpose[{perpframe[1], perpframe[2]}];
A = hanger.stretcher.aligner;
MatrixForm[A]
```

$$\begin{pmatrix} 0.725322 & 0.446352 \\ 0.446352 & 0.274678 \end{pmatrix}$$

Hit the given point with A and plot:

```
hitplot =
  Graphics[{CadmiumOrange, PointSize[0.04], Point[A.givenpoint]}];
Show[setup, frameplot, hitplot];
```



Nice. Give it the acid test:

```
(givenpoint - A.givenpoint).perpframe[1]
```

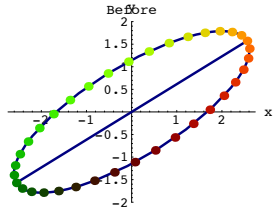
```
0
```

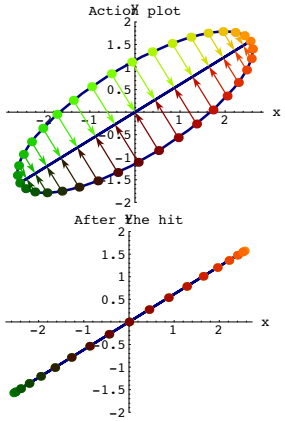
This confirms that the vector running from A.givenpoint to givenpoint is perpendicular to the line and, as you know, the shortest distance from the line to the given point is the perpendicular distance.

See this projection matrix do its work in this action movie:

```
Clear[x, y, t, hitplotter,
  hitpointplotter, pointcolor, actionarrows, matrix2D];
{tlow, thigh} = {0, 2 π};
ranger = 3;
```

```
{x[t_], y[t_]} = 3 Cos[t] perpframe[1]  + Sin[t] perpframe[2];

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

jump = (thigh - tlow) / 36;

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[lineplot, hitplotter[IdentityMatrix[2]],
   hitpointplotter[IdentityMatrix[2]], PlotLabel → "Before",
   PlotRange -> {-2, 2}, DisplayFunction → $DisplayFunction];

Show[lineplot, before, actionarrows[A], PlotLabel → "Action plot",
  PlotRange -> {-2, 2}, DisplayFunction → $DisplayFunction];

Show[lineplot, hitplotter[A], hitpointplotter[A],
  PlotRange -> {-2, 2}, PlotLabel → "After the hit",
  DisplayFunction → $DisplayFunction];
```





Grab the plots, align and then animate.

T.2) More on reflection matrices (flippers).

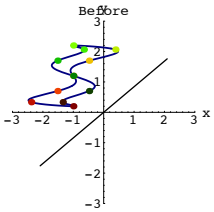
Making matrices for bouncing light rays off curves

□ T.2.a) Flipping a curve over a line with a matrix hit

Here's a curve shown together with a line through {0,0}:

```
Clear[x, y, t, line, s];
line[s_] = s {2.09, 1.76};
lineplot = Graphics[Line[{line[-1], line[1]}]];
{tlow, thigh} = {0, 2 π};
ranger = 3.0;
{x[t_], y[t_]} = {-1, 1.2} + {2 Sin[t] Cos[t] (0.6 - Cos[3 t]), Sin[t]};
Clear[hitplotter, hitpointplotter,
  pointcolor, actionarrows, matrix2D];
pointcolor[t_] = RGBColor[0.5 (Cos[t] + 1), 0.5 (Sin[t] + 1), 0];
jump = (thigh - tlow)/12;
hitplotter[matrix2D_] := ParametricPlot[matrix2D.{x[t], y[t]},
   {t, tlow, thigh}, PlotStyle → {{Thickness[0.01], 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}];
before = Show[hitplotter[IdentityMatrix[2]],
   hitpointplotter[IdentityMatrix[2]], lineplot,
   PlotLabel → "Before", DisplayFunction → $DisplayFunction];
```



Use a frame flipper matrix hit to pick up the curve and flip it over the plotted line to get its mirror image with respect to the plotted line.

□ Answer:

This is a job for a perpendicular frame. To get a good one, look at the parameterization of the line:

```
line[s]
```

```
{2.09 s, 1.76 s}
```

A unit vector pointing in the direction of the line is:

```
Clear[perpframe];
perpframe[1] = line[1.0]/Sqrt[line[1.0].line[1.0]]
```

```
{0.764911, 0.644136}
```

A unit vector perpendicular to perpframe[1] is:

```
perpframe[2] = {-perpframe[1][[2]], perpframe[1][[1]]}
```

```
{-0.644136, 0.764911}
```

Now use the matrix maker to define your frame flipper matrix:

```
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]
```

$$\begin{pmatrix} 0.170178 & 0.985413 \\ 0.985413 & -0.170178 \end{pmatrix}$$

Note the minus sign in the specification of the hanger frame.

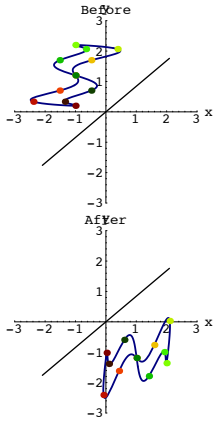
Check it out, remembering that perpframe[1] points in the direction of the line and perpframe[2] is perpendicular to the line:

```
A.perpframe[1] == perpframe[1]
A.perpframe[2] == -perpframe[2]
```

```
True
True
```

Now watch what a hit with A does to the curve:

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



Grab both plots and animate.

See that puppy flip.

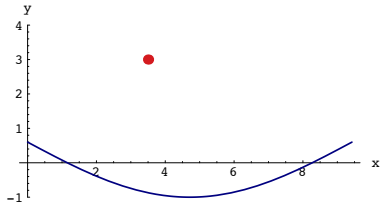
□ T.2.b.i) Bouncing light rays off a curve

For full understanding, you should be familiar with part a) above.

Here's a curve and a point above the curve all plotted in true scale:

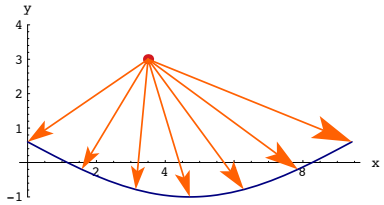
```
Clear[x, y, t];
{x[t_], y[t_]} = {0, 0.6} + {3 t, -1.6 Sin[t]};
```

```
point = {3.5, 3.0};
{tlow, thigh} = {0, π};
curveplot = ParametricPlot[{x[t], y[t]},
  {t, tlow, thigh}, PlotStyle → {{NavyBlue, Thickness[0.005]}},
  PlotRange → {-1, 4}, AspectRatio → Automatic,
  AxesLabel → {"x", "y"}, DisplayFunction → Identity];
pointplot = Graphics[{VenetianRed, PointSize[0.03], Point[point]}];
Show[curveplot, pointplot, DisplayFunction → $DisplayFunction];
```



Light rays emanate from the point and hit the curve like this:

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

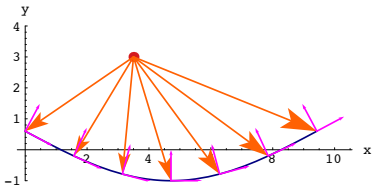


Your job is to plot the reflected light rays. Do it.

□ Answer:

Throw in these tangential (perpframe[1,t]) and normal (perpframe[2,t]) perpendicular frames:

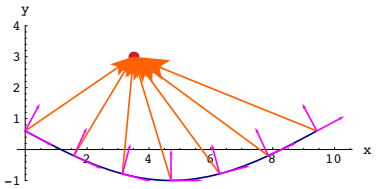
```
Clear[perpframe];
perpframe[1, t_] = {x'[t], y'[t]} / Sqrt[{x'[t], y'[t]}.{x'[t], y'[t]}];
perpframe[2, t_] = {-y'[t], x'[t]} / Sqrt[{y'[t], -x'[t]}.{y'[t], -x'[t]}];
frameplots = {Table[Arrow[perpframe[1, t], Tail → {x[t], y[t]},
    VectorColor → Magenta], {t, tlow, thigh, jump}],
  Table[Arrow[perpframe[2, t], Tail → {x[t], y[t]},
    VectorColor → Magenta], {t, tlow, thigh, jump}]};
step1 = Show[curveplot, pointplot, rayplots, frameplots,
  DisplayFunction → $DisplayFunction];
```



```
perpframe[1,t] is tangent to the curve at {x[t],y[t]}.
perpframe[1,t] is perpendicular to the curve at {x[t],y[t]}.
```

Now reverse the light vectors:

```
reversedrayplots = Table[Arrow[-ray[t], Tail → {x[t], y[t]},
  VectorColor → CadmiumOrange], {t, tlow, thigh, jump}];
step2 = Show[curveplot, pointplot, reversedrayplots,
  frameplots, DisplayFunction → $DisplayFunction];
```

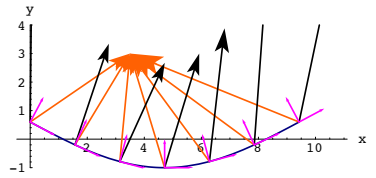


These are not the reflected rays, but the reflected rays do make the same angle with the plotted normals that these vectors make.

This tells you that to get the reflected rays, all you have to do is to hit the -ray[t] vectors with frame flipper matrices to flip -ray[t] about perpframe[2,t].

Here you go:

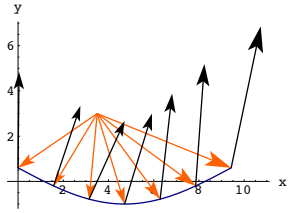
```
Clear[alignerframe, hangerframe, aligner, hanger, frameflipper];
{alignerframe[1, t_], alignerframe[2, t_]} =
  {perpframe[1, t], perpframe[2, t]};
aligner[t_] = {alignerframe[1, t], alignerframe[2, t]};
stretcher = {{1, 0}, {0, 1}};
{hangerframe[1, t_], hangerframe[2, t_]} =
  {-perpframe[1, t], perpframe[2, t]};
hanger[t_] = Transpose[{hangerframe[1, t], hangerframe[2, t]}];
frameflipper[t_] = hanger[t].stretcher.aligner[t];
reflectedrayplots =
 Table[Arrow[frameflipper[t].(-ray[t]), Tail → {x[t], y[t]},
   VectorColor → Black], {t, tlow, thigh, jump}];
step3 = Show[curveplot, reversedrayplots, reflectedrayplots,
  frameplots, DisplayFunction → $DisplayFunction];
```



```
Note the extra minus sign in the specification of the hangerframe.
That's what gives you the flip.
```

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

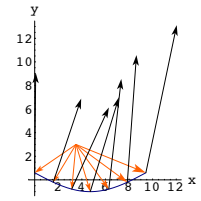
```
finalproduct = Show[curveplot, rayplots, reflectedrayplots,
  PlotRange → All, DisplayFunction → $DisplayFunction];
```



If you want lengthen the reflected rays, you can.

Just do this:

```
scalefactor = 2.0;
scaledreflectedrayplots =
  Table[Arrow[frameflipper[t].(-ray[t]), Tail → {x[t], y[t]},
    ScaleFactor → scalefactor, VectorColor → Black,
    HeadSize → 0.9], {t, tlow, thigh, jump}];
Show[curveplot, rayplots, scaledreflectedrayplots,
  PlotRange → All, DisplayFunction → $DisplayFunction];
```

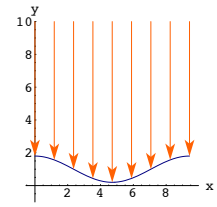


And you're out of here.

□T.2.b.ii) When the light source is far away

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

```
Clear[x, y, t, ray];
{x[t_], y[t_]} = {0, 1} + {3 t, 0.8 Cos[2 t]};
{tlow, thigh} = {0, π};
curveplot = ParametricPlot[{x[t], y[t]},
   {t, tlow, thigh}, PlotStyle → {{NavyBlue, Thickness[0.005]}},
   PlotRange → {-1, 10}, AspectRatio → Automatic,
   AxesLabel → {"x", "y"}, DisplayFunction → Identity];
ray[t_] = {x[t], y[t]} - {x[t], 10};
jump = (thigh - tlow) / 8;
rayplots =
  Table[Arrow[ray[t], Tail → {x[t], 10}, VectorColor → CadmiumOrange,
    HeadSize → 1], {t, tlow, thigh, jump}];
setup = Show[curveplot, rayplots,
  DisplayFunction → $DisplayFunction];
```



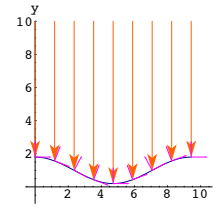
Your job is to plot the reflected light rays. Do it.

□**Answer:**

```
This is a copy, paste and edit job on part i).
```

Throw in these tangential (perpframe[1,t]) and normal (perpframe[2,t]) perpendicular frames:

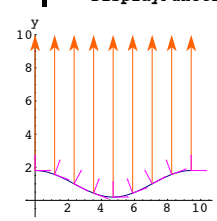
```
Clear[perpframe];
perpframe[1, t_] = {x'[t], y'[t]} / Sqrt[{x'[t], y'[t]}.{x'[t], y'[t]}];
perpframe[2, t_] = {-y'[t], x'[t]} / Sqrt[{y'[t], -x'[t]}.{y'[t], -x'[t]}];
frameplots = {Table[Arrow[perpframe[1, t], Tail → {x[t], y[t]},
    VectorColor → Magenta], {t, tlow, thigh, jump}],
  Table[Arrow[perpframe[2, t], Tail → {x[t], y[t]},
    VectorColor → Magenta], {t, tlow, thigh, jump}]};
step1 = Show[curveplot, rayplots, frameplots,
  DisplayFunction → $DisplayFunction];
```



```
perpframe[1,t] is tangent to the curve at {x[t],y[t]}.
```

Now reverse the light vectors:

```
reversedrayplots = Table[
  Arrow[-ray[t], Tail → {x[t], y[t]}, VectorColor → CadmiumOrange,
   HeadSize → 1], {t, tlow, thigh, jump}];
step2 = Show[curveplot, reversedrayplots, frameplots,
  DisplayFunction → $DisplayFunction];
```

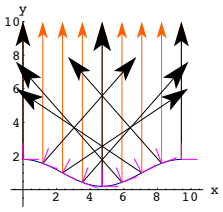


These are not the reflected rays, but the reflected rays do make the same angle with the plotted normals that these vectors make.

This tells you that to get the reflected rays, all you have to do is to hit the -ray[t] vectors with frame flipper matrices to flip -ray[t] about perpframe[2,t].

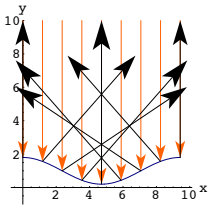
Here you go:

```
Clear[alignerframe, hangerframe, aligner, hanger, frameflipper];
{alignerframe[1, t_], alignerframe[2, t_]} =
  {perpframe[1, t], perpframe[2, t]};
aligner[t_] = {alignerframe[1, t], alignerframe[2, t]};
stretcher = ({{1, 0}, {0, 1}});
{hangerframe[1, t_], hangerframe[2, t_]} =
  {-perpframe[1, t], perpframe[2, t]};
hanger[t_] = Transpose[{hangerframe[1, t], hangerframe[2, t]}];
frameflipper[t_] = hanger[t].stretcher.aligner[t];
reflectedrayplots =
 Table[Arrow[frameflipper[t].(-ray[t]), Tail → {x[t], y[t]},
   VectorColor → Black], {t, tlow, thigh, jump}];
step3 = Show[curveplot, reversedrayplots, reflectedrayplots,
   frameplots, DisplayFunction → $DisplayFunction];
```



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

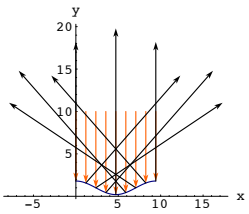
```
finalproduct = Show[curveplot, rayplots,
   reflectedrayplots, DisplayFunction → $DisplayFunction];
```



If you want lengthen the reflected rays, you can.

Just do this:

```
scalefactor = 2.0;
scaledreflectedrayplots =
  Table[Arrow[frameflipper[t].(-ray[t]), Tail → {x[t], y[t]},
    ScaleFactor → scalefactor, VectorColor → Black,
    HeadSize → 1], {t, tlow, thigh, jump}];
Show[curveplot, rayplots, scaledreflectedrayplots,
  PlotRange → All, DisplayFunction → $DisplayFunction];
```



And you're out of here.

T.3) Aligners and hangers made with right hand perpendicular frames are rotation matrices; Aligners and hangers made with left hand perpendicular frames are not rotation matrices; they are reflection matrices (flippers)

□ T.3.a.i) Hangers corresponding to left hand perpendicular frames are not rotations; they are pure flippers (reflection matrices)

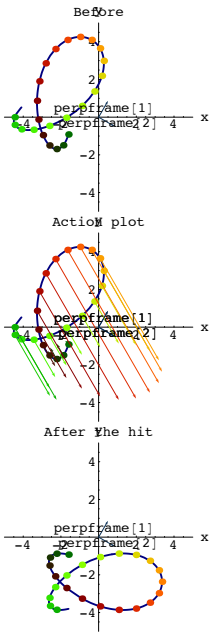
You already know that aligners and hangers based on right hand perpendicular frames are rotation matrices.

In this problem you are going to look into the math behind what happens when you hit with aligners and hangers based on left hand perpendicular frames.

To get the ball off the ground, here's a left hand perpendicular frame and an action movie showing what a hit with the hanger corresponding to this perpendicular frame does to a sample curve

```
s = π/3;
{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]]]};


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

Clear[x, y, t, hitplotter,
  hitpointplotter, pointcolor, actionarrows, matrix2D];
{tlow, thigh} = {-1, 1};
ranger = 5;
{x[t_], y[t_]} =
  {-1, 1} + 3 Cos[4 t] perpframe[1] + (1.5 Sin[5 t] - 1) perpframe[2];

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];
```



Grab all three plots, align and animate slowly.

The visual evidence is that hits with this hanger based on a left hand perpendicular frame is a flipper matrix (reflection matrix).

The goal here is to look into the math behind this in order to find out what line that matrix is flipping over.

The left hand frame used here is

$$\{\{\text{Cos}[s], \text{Sin}[s]\}, -\{\text{Cos}[s + \tfrac{\pi}{2}], \text{Sin}[s + \tfrac{\pi}{2}]\}\}$$

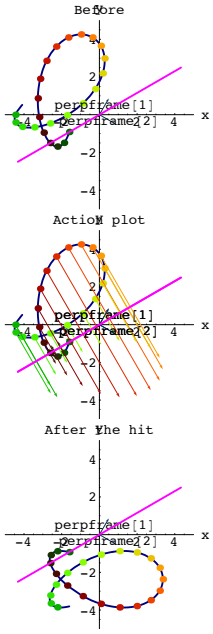
with $s = \frac{\pi}{3}$.

Throw into the action movie a plot of the line through {0,0} running in the direction of

$$\{\text{Cos}[\tfrac{s}{2}], \text{Sin}[\tfrac{s}{2}]\}:$$

```
Clear[t];
```

```
lineplot = ParametricPlot[t {Cos[s/2], Sin[s/2]},
  {t, -5, 5}, PlotStyle -> {{Thickness[0.01], Magenta}},
  DisplayFunction -> Identity];
before = Show[hitplotter[IdentityMatrix[2]],
   hitpointplotter[IdentityMatrix[2]], frameplot, lineplot,
   PlotLabel → "Before", DisplayFunction → $DisplayFunction];

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

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



Grab all three plots, align and animate slowly.

You betcha.

The hit with the hanger corresponding to this left hand frame

$$\{\{\text{Cos}[s], \text{Sin}[s]\}, -\{\text{Cos}[s+\tfrac{\pi}{2}], \text{Sin}[s+\tfrac{\pi}{2}]\}\}$$

seemed to flip that curve right over the line running in the direction of

$$\{\text{Cos}[\tfrac{s}{2}], \text{Sin}[\tfrac{s}{2}]\}.$$

Explain why this happens for any left hand perpendicular frame

$$\{\{\text{Cos}[s], \text{Sin}[s]\}, -\{\text{Cos}[s+\tfrac{\pi}{2}], \text{Sin}[s+\tfrac{\pi}{2}]\}\}.$$

□Answer:

Here's the hanger matrix corresponding to a cleared left hand perpendicular frame.

```
Clear[leftperpframe, s];
{leftperpframe[1], leftperpframe[2]} =
 {{Cos[s], Sin[s]}, -{Cos[s + π/2], Sin[s + π/2]}};
lefthanger = Transpose[{leftperpframe[1], leftperpframe[2]}];
MatrixForm[lefthanger]
```

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

Go with a cleared right hand perpendicular frame based on a cleared angle t:

```
Clear[rightperpframe, t];
{rightperpframe[1], rightperpframe[2]} =
 {{Cos[t], Sin[t]}, {Cos[t + π/2], Sin[t + π/2]}}
```

$$\{\{\text{Cos}[t], \text{Sin}[t]\}, \{-\text{Sin}[t], \text{Cos}[t]\}\}$$

Make the matrix that flips everything over the line through {0,0} defined by

rightperpframe[1]= {Cos[t],Sin[t]}:

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

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

Look at:

```
MatrixForm[lefthanger]
```

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

At this point, it should become vivid that when you set

$$t = \frac{s}{2},$$

then the flipper matrix becomes the same as the lefthanger matrix:

```
lefthanger == flipper /. t -> s/2
```

```
True
```

This signals that hanging on the left hand perpendicular frame

$$\{\{\text{Cos}[s], \text{Sin}[s]\}, -\{\text{Cos}[s+\tfrac{\pi}{2}], \text{Sin}[s+\tfrac{\pi}{2}]\}\}$$

is the same as reflecting (flipping) over the line through {0,0} defined by

$$\{\text{Cos}[\tfrac{s}{2}], \text{Sin}[\tfrac{s}{2}]\}.$$

Wrap up:

A hanger based on a left hand perpendicular frame is a reflection matrix , pure and simple.

□T.3.a.ii) Aligners corresponding to left hand perpendicular frames are not rotations; they are pure flippers (reflection matrices)

Explain this:

The aligner matrix based on a left hand frame

$$\{\{\text{Cos}[s], \text{Sin}[s]\}, -\{\text{Cos}[s+\tfrac{\pi}{2}], \text{Sin}[s+\tfrac{\pi}{2}]\}\}$$

is a reflection matrix

□Answer:

Here's the aligner matrix based on the left hand perpendicular frame.

```
Clear[leftperpframe, s];
{leftperpframe[1], leftperpframe[2]} =
 {{Cos[s], Sin[s]}, -{Cos[s + π/2], Sin[s + π/2]}};
leftaligner = {leftperpframe[1], leftperpframe[2]};
MatrixForm[leftaligner]
```

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

And here's the hanger matrix based on the same left hand perpendicular frame.

```
lefthanger = Transpose[{leftperpframe[1], leftperpframe[2]}];
MatrixForm[lefthanger]
```

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

They are the same.

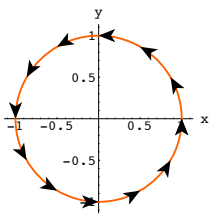
Because you already know that the hanger matrix based on a left hand frame is a reflection matrix, you now know why an aligner based on a left hand frame is also a reflection matrix.

T.4) Using left hand and right hand perpendicular frames to make matrices that preserve or reverse orientation

□T.4.a.i) Using a left hand frame and a right hand frame to make a matrix that hits counterclockwise orientation into clockwise orientation

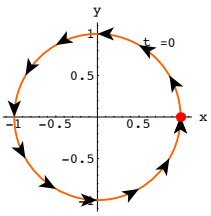
Here's the unit circle shown with tangent vectors that indicate the direction of the parameterization

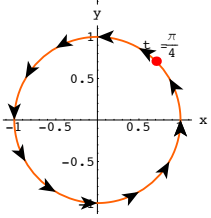
```
Clear[x, y, pointers, hitplot, matrix, t];
{tlow, thigh} = {0, 2π};
{x[t_], y[t_]} = {Cos[t], Sin[t]};
hitplot[matrix_] := ParametricPlot[matrix.{x[t], y[t]},
  {t, tlow, thigh}, PlotStyle -> {{Thickness[0.01], CadmiumOrange}},
  AxesLabel -> {"x", "y"}, DisplayFunction -> Identity];
tjump = (thigh - 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];
```

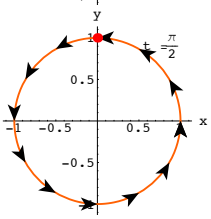


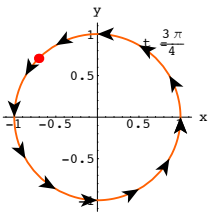
The parameterization is counterclockwise in the sense that as t increases the points {x[t],y[t]} on the curve advance in the counterclockwise
See this happen:

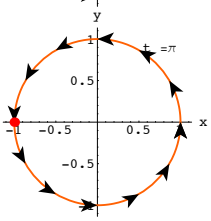
```
jump = π/4;
Table[Show[circleplot,
  Graphics[{Red, PointSize[0.05], Point[{x[t], y[t]}]}],
  Graphics[Text[t, {0.9, 0.9}]],
  Graphics[Text["t =", {0.7, 0.9}]]], {t, 0, 2 π - jump, jump}]
```

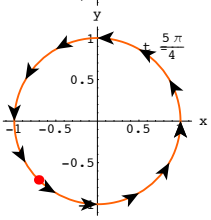


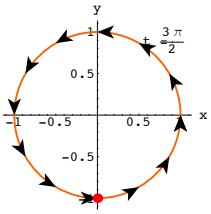


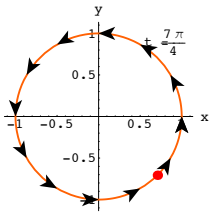












{- Graphics -, - Graphics -, - Graphics -, - Graphics -,
 - Graphics -, - Graphics -, - Graphics -, - Graphics -}

Grab and animate

Counterclockwise action.

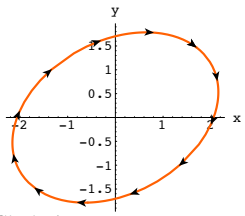
Now make a matrix
A= hanger.stretcher.aligner with a
- aligner based on a right hand perpendicular frame,
- hanger based on a left hand perpendicular frame, and
- positive stretch factors:

```
s = 0.3;
aligner = {{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}};

{xstretch, ystretch} = {2.3, 1.6};
stretcher = ( xstretch   0
              0          ystretch );

t = 0.5;
hanger =
 Transpose[ {{Cos[t], Sin[t]}, -{Cos[t + π/2], Sin[t + π/2]}}];
A = hanger.stretcher.aligner;
MatrixForm[A]
```

$$\begin{pmatrix} 1.7016 & 1.32931 \\ 1.46838 & -1.01555 \end{pmatrix}$$

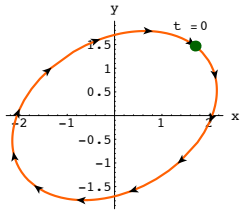
See what a hit with this matrix does to the circle and its parameterization:

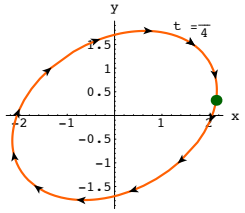
```
hitcircleplot =
 Show[hitplot[A], pointers[A], DisplayFunction -> $DisplayFunction];
```

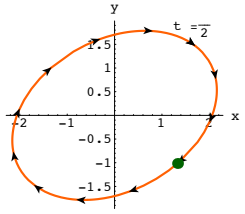


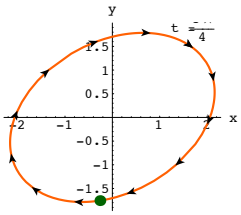
Clockwise.
See it happen:

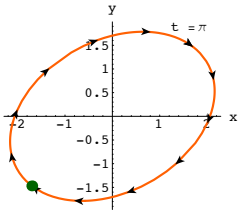
```
Table[Show[hitcircleplot,
  Graphics[{GosiaGreen, PointSize[0.05], Point[A.{x[t], y[t]}]}],
  Graphics[Text[t, {1.9, 1.9}]],
  Graphics[Text["t =", {1.5, 1.9}]]], {t, 0, 2 π - jump, jump}]
```

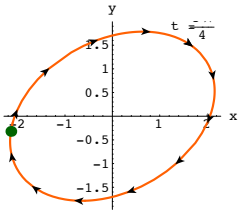


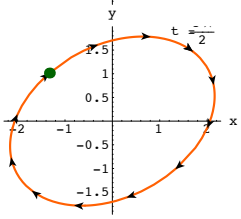


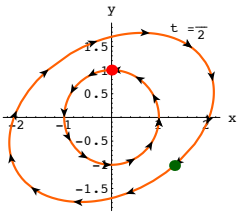


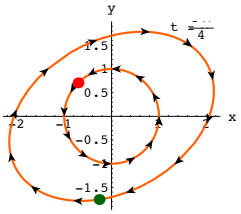


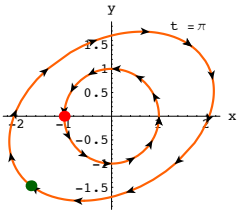


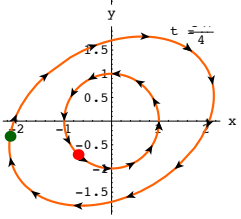


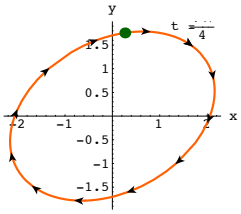






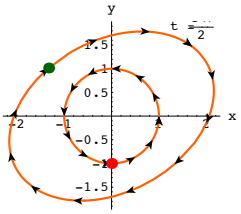


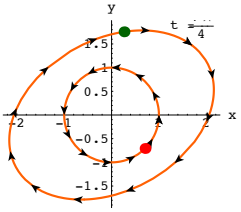




{- Graphics -, - Graphics -, - Graphics -, - Graphics -,
 - Graphics -, - Graphics -, - Graphics -, - Graphics -}

Grab and animate



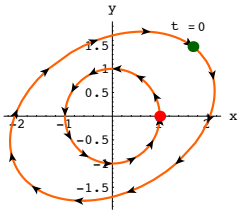


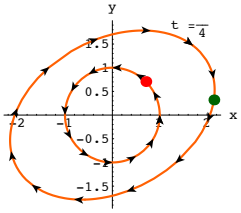
{- Graphics -, - Graphics -, - Graphics -, - Graphics -,
 - Graphics -, - Graphics -, - Graphics -, - Graphics -}

Grab and animate slowly.

See both:

```
Table[Show[circleplot, hitcircleplot,
  Graphics[{GosiaGreen, PointSize[0.05], Point[A.{x[t], y[t]}]}],
  Graphics[{Red, PointSize[0.05], Point[{x[t], y[t]}]}],
  Graphics[Text[t, {1.9, 1.9}]],
  Graphics[Text["t =", {1.5, 1.9}]]], {t, 0, 2 π - jump, jump}]
```



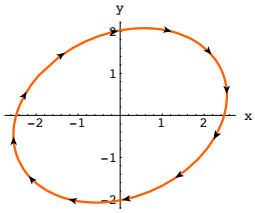


After the hit, the parameterization is clockwise.
In other words, a hit with the matrix A reversed orientation from counterclockwise to clockwise.

See what happens for some matrices A= hanger.stretcher.aligner with
- aligner based on a random right hand frame,
- hanger based on a random left hand frame, and
- random positive stretch factors:

```
s = Random[Real, {-π/2, π/2}];

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


{xstretch, ystretch} =
  {Random[Real, {1, 3}], Random[Real, {0.5, 2.0}]};
stretcher = ( xstretch      0
                0       ystretch );
s = Random[Real, {-π/2, π/2}];
hanger =
 Transpose[ {{Cos[s], Sin[s]}, -{Cos[s + π/2], Sin[s + π/2]}}];
A = hanger.stretcher.aligner;
Show[hitplot[A], pointers[A], DisplayFunction -> $DisplayFunction];
```

```
"A = " MatrixForm[A]
```



$$A = \begin{pmatrix} 2.22344 & -1.22445 \\ -0.565877 & -1.99489 \end{pmatrix}$$

Rerun many times noting that after each hit, the resulting parameterization is clockwise. Explain why it had to turn out that way.

Answer:

Go with a new matrix A= hanger.stretcher.aligner with

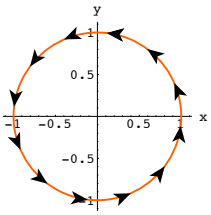
- aligner based on a random right hand frame,
- hanger based on a random left hand frame, and
- positive stretch factors:

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

{xstretch, ystretch} = {2.3, 1.5};
stretcher = ( xstretch   0
              0          ystretch );
t = Random[Real, {-π/2, π/2}];
hanger =
 Transpose[ {{Cos[t], Sin[t]}, -{Cos[t + π/2], Sin[t + π/2]}}];
A = hanger.stretcher.aligner;
MatrixForm[A]
```

$$\begin{pmatrix} 1.47483 & -0.73109 \\ -0.33811 & -2.17165 \end{pmatrix}$$

Here's what the hit with the aligner does:

```
afteralignerhit = Show[hitplot[aligner],
  pointers[aligner], DisplayFunction -> $DisplayFunction];
```



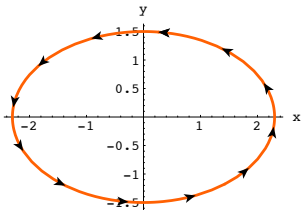
Still counterclockwise.

Reason: The aligner is based on a right hand frame and is therefore a rotation matrix.

Hits with rotation matrices cannot switch the direction of the parameterization.

Here's what the hit with the stretcher.aligner does:

```
afterstretcheralignerhit =
 Show[hitplot[stretcher.aligner], pointers[stretcher.aligner],
  DisplayFunction -> $DisplayFunction];
```



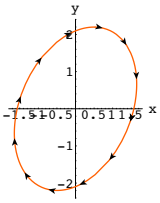
Still counterclockwise.

Reason:

Hits with stretcher matrices (coming from positive stretch factors) cannot switch the direction of the parameterization.

Here's what the hit with the A = hanger.stretcher.aligner does:

```
afterhangerstretcheralignerhit =
 Show[hitplot[hanger.stretcher.aligner],
  pointers[hanger.stretcher.aligner],
  DisplayFunction -> $DisplayFunction];
```



Now the parameterization is clockwise.

Reason: The hanger is based on a left hand perpendicular frame and is therefore a reflection (flipper) matrix that flips about some line through {0,0}..

Hits with reflection (flipper) matrices always switch the direction of the parameterization.

For exactly these reasons, hits with any matrix

A= hanger.stretcher.aligner

made with

- aligner based on a random right hand frame,
- hanger based on a random left hand frame, and
- positive stretch factors

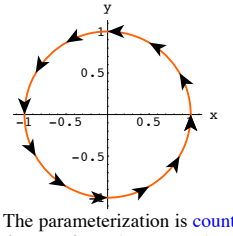
reverse orientation from counterclockwise to clockwise.

T.4.a.ii) Using two right hand frames to make a matrix that hits counterclockwise orientation into counterclockwise orientation

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];
```



The parameterization is counterclockwise in the sense that as t increases the points {x[t],y[t]} on the curve advance in the counterclockwise

Now make a matrix
A= hanger.stretcher.aligner with a

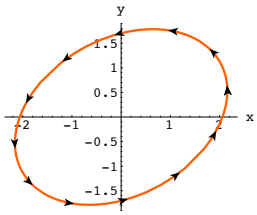
- aligner based on a right hand perpendicular frame,
- hanger based on a right hand perpendicular frame, and
- positive stretch factors:

```
s = 0.3;
aligner = {{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}};

{xstretch, ystretch} = {2.3, 1.6};
stretcher = ( xstretch   0
              0          ystretch );

t = 0.5;
hanger = Transpose[ {{Cos[t], Sin[t]}, {Cos[t + π/2], Sin[t + π/2]}}];
A = hanger.stretcher.aligner;
MatrixForm[A]
```

$$\begin{pmatrix} 2.15498 & -0.136331 \\ 0.63848 & 1.66728 \end{pmatrix}$$

See what a hit with this matrix does to the circle and its parameterization:

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

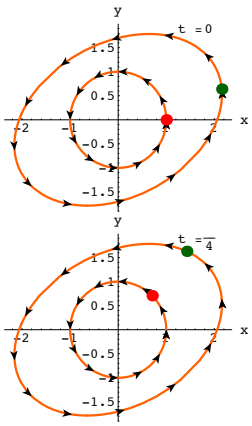


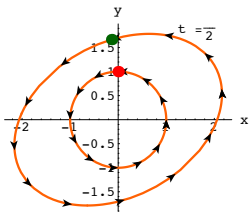
Still counterclockwise.
See it happen:

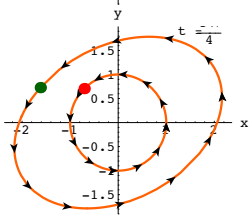
Grab and animate

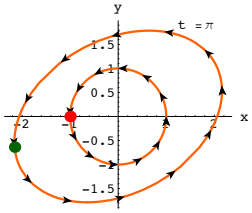
See both:

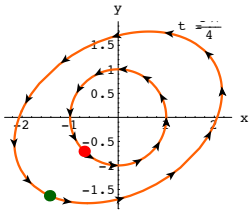
```
jump = π/4;
Table[Show[circleplot, hitcircleplot,
  Graphics[{GosiaGreen, PointSize[0.05], Point[A.{x[t], y[t]}]}],
  Graphics[{Red, PointSize[0.05], Point[{x[t], y[t]}]}],
  Graphics[Text[t, {1.9, 1.9}]],
  Graphics[Text["t =", {1.5, 1.9}]]], {t, 0, 2π - jump, jump}]
```

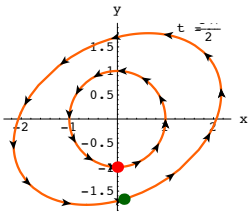


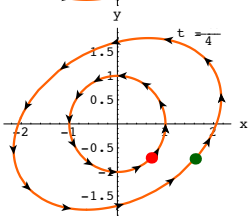












{- Graphics -, - Graphics -, - Graphics -, - Graphics -,
 - Graphics -, - Graphics -, - Graphics -, - Graphics -}

Grab and animate slowly.

After the hit, the parameterization is still counterclockwise.
In other words, a hit with the matrix A did not reverse orientation from counterclockwise to clockwise.

Explain why the same thing will happen for any matrices
A= hanger.stretcher.aligner with
- aligner based on any right hand frame,
- hanger based on any right hand frame, and
- any positive stretch factors.

□ Answer:

Go with a matrix A= hanger.stretcher.aligner with
- aligner based on a random right hand frame,
- hanger based on a random right hand frame, and
- positive stretch factors:

Because aligner is based on a right hand frame,the hit with aligner is a rotation. So the hit with aligner does not reverse orientation from counterclockwise to clockwise.

Stretcher matrices never reverse orientation from counterclockwise to clockwise.

Because hanger is based on a right hand frame,the hit with hanger is a rotation. So the hit with hanger does not reverse orientation from counterclockwise to clockwise.

The upshot:

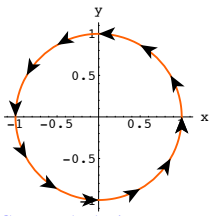
None of the hits with aligner, stretcher or hanger reverse orientation from counterclockwise to clockwise.

So a hit with A= hanger.stretcher.aligner cannot reverse orientation from counterclockwise to clockwise.

□ T.4.a.iii) Counting the flips

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];
```



Counterclockwise.

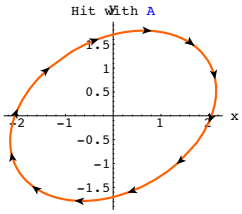
Now make a matrix
A = hanger.stretcher.aligner with a
- aligner based on a right hand perpendicular frame,
- hanger based on a left hand perpendicular frame, and
- positive stretch factors:

```
s = 0.3;
Aaligner = {{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}};

{Axstretch, Aystretch} = {2.3, 1.6};
Astretcher = ( Axstretch      0
                    0      Aystretch );

t = 0.5;
Ahanger =
 Transpose[ {{Cos[t], Sin[t]}, -{Cos[t + π/2], Sin[t + π/2]}}];
A = Ahanger.Astretcher.Aaligner;
MatrixForm[A]
```

$$\begin{pmatrix} 1.7016 & 1.32931 \\ 1.46838 & -1.01555 \end{pmatrix}$$

Because the hanger for A is based on a left hand frame and the aligner for A is based on a right hand frame, a hit with A on the counterclockwise unit circle above produces a clockwise parameterization:

```
Ahitcircleplot = Show[hitplot[A], pointers[A],
  PlotLabel -> "Hit with A", DisplayFunction -> $DisplayFunction];
```



Clockwise.

Now make a matrix

B = Bhanger.Bstretcher.Baligner with a

- aligner based on a left hand perpendicular frame,
- hanger based on a right hand perpendicular frame, and
- positive stretch factors:

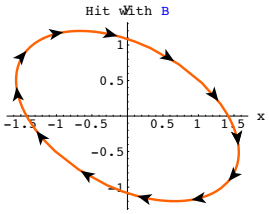
```
s = 0.4;
Baligner = {{Cos[s], Sin[s]}, -{Cos[s + π/2], Sin[s + π/2]}};

{Bxstretch, Bystretch} = {1.7, 1.0};
```

```
Bstretcher = ( Bxstretch      0
                    0      Bystretch );

t = -0.5;
Bhanger =
 Transpose[ {{Cos[t], Sin[t]}, {Cos[t + π/2], Sin[t + π/2]}}];
B = Bhanger.Bstretcher.Baligner;
MatrixForm[B]
```

$$\begin{pmatrix} 1.56082 & 0.139389 \\ -0.40894 & -1.12569 \end{pmatrix}$$

Because the hanger for B is based on a right hand frame and the aligner for B is based on a left hand frame, a hit with B on the counterclockwise unit circle above produces a clockwise parameterization:

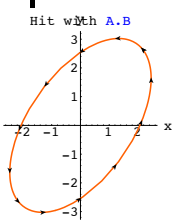
```
Bhitcircleplot = Show[hitplot[B], pointers[B],
  PlotLabel -> "Hit with B", DisplayFunction -> $DisplayFunction];
```



Clockwise.

Now see what happens when you hit the same unit circle with A.B:

```
ABhitcircleplot = Show[hitplot[A.B], pointers[A.B],
  PlotLabel -> "Hit with A.B", DisplayFunction -> $DisplayFunction];
```



Counterclockwise.

Explain how you could have predicted this in advance.

□ Answer:

A hit with B reverses orientation from counterclockwise to clockwise.. And a further hit with A reverses orientation again - this time from clockwise to counterclockwise. Consequently the hit with A.B does not reverse orientation at all.

If you feel you want a more detailed explanation, click on the right.

A = Ahanger.Astretcher.Aaligner

B = Bhanger.Bstretcher.Baligner

A. B = Ahanger.Astretcher.Aaligner.Bhanger.Bstretcher.Baligner

- Hits with the Aaligner and the Bhanger **do not** reverse orientation.

Reason: Both are based on right hand perpendicular frames, so they are rotation matrices.
Hits with rotation matrices do not reverse orientation.

- Hits with the Astretcher and the Bstretcher **do not** reverse orientation.

Reason: Stretches out the axes cannot reverse orientation.

- Hits with the Ahanger and the Baligner **do** reverse orientation.

Reason: Both are based on left hand perpendicular frames, so they are flipper matrices.
Hits with flipper matrices always reverse orientation.

So with regard to orientation switches, you can analyze hits with

A. B = Ahanger.Astretcher.Aaligner.Bhanger.Bstretcher.Baligner

in these terms:

(Orientation reverse).(No reverse).(No reverse).(No reverse)..(No reverse).(Orientation reverse) .

The first orientation reverse on the far right changes counterclockwise to clockwise. The second orientation reverse (on the far left) changes clockwise back to counterclockwise.

The upshot:

Hits with A.B hit counterclockwise into counterclockwise.