Matrices, Geometry&*Mathematica*

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

Producer: Bruce Carpenter

Publisher: Math Everywhere, Inc.

MGM.04 SVD Analysis of 2D Matrices

BASICS

B.1) SVD Matrix analysis:

Putting any 2D matrix A in the form A = hanger. stretcher. aligner

SVD analysis tells you that every matrix can be duplicated with matrix maker ingredients: aligner, stretcher and hanger. If you want to find out how to do this for any 2D matrix, jump in right now.

□B.1.a.i) Taking a random matrix and duplicating it with matrix maker ingredients:

aligner, stretcher and hanger

Here's a random 2D matrix

```
A = {{Random[Real, {-2, 2}]  Random[Real, {-2, 2}]},
     {Random[Real, {-2, 2}]  Random[Real, {-2, 2}]}};
MatrixForm[A]
```

$$\begin{pmatrix} -0.669767 & -1.9008 \\ -1.91958 & -0.650124 \end{pmatrix}$$

Duplicate this matrix by coming up by with an aligner frame, stretch factors and a hanger frame so that

A = hanger. stretcher.aligner.

□Answer:

Thanks go to Professor Todd Will of Davidson College for suggesting this..

First go with a cleared perpendicular frame:

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

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

Now go after an s that makes

A.perpframe[1] and A.perpframe[2]

into perpendicular vectors

```
schoices = Solve[(A.perpframe[1]).(A.perpframe[2]) == 0, s]
```

```
{{s → -2.36588}, {s → -0.795081}, {s → 0.775715}, {s → 2.34651}}
```

Take any of these choices

```
s = s /. schoices[[2]]
```

```
-0.795081
```

Plug this s into the cleared perpendicular frame to get your alignerframe and aligner::

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

$$\begin{pmatrix} 0.700227 & -0.71392 \\ 0.71392 & 0.700227 \end{pmatrix}$$

The stretch factors are:

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

and

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

so your stretcher is:

```
xstretch = Norm[A.alignerframe[1]];
ystretch = Norm[A.alignerframe[2]];
stretcher = {{xstretch, 0}, {0, ystretch}};
MatrixForm[stretcher]
```

$$\begin{pmatrix} 1.2502 & 0 \\ 0 & 2.57023 \end{pmatrix}$$

Your hangerframe is:

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

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

This makes it automatic that the hanger frame vectors are perpendicular unit vectors.
And this makes it automatic that
A.alignerframe[1] = xstretch hangerframe[1]

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

So your hanger is:

```
Clear[hangerframe];
    {hangerframe[1], hangerframe[2]} =
  {1/xstretch A.alignerframe[1], 1/ystretch A.alignerframe[2]};
hanger = Transpose[{hangerframe[1], hangerframe[2]}];
MatrixForm[hanger]
```

$$\begin{pmatrix} 0.710311 & -0.703888 \\ -0.703888 & -0.710311 \end{pmatrix}$$

Try them out by comparing A with:

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

$$\begin{pmatrix} -0.669767 & -1.9008 \\ -1.91958 & -0.650124 \end{pmatrix}$$

$$\begin{pmatrix} -0.669767 & -1.9008 \\ -1.91958 & -0.650124 \end{pmatrix}$$

Bingo!

Try it for more random 2D matrices A.

```
Clear[perpframe, s];
    {perpframe[1], perpframe[2]} =
  {{Cos[s], Sin[s]}, {Cos[s + π/2], Sin[s + π/2]}};
schoices = Solve[(A.perpframe[1]).(A.perpframe[2]) == 0, s];
s = s /. schoices[[2]];

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

xstretch = Norm[A.alignerframe[1]];
ystretch = Norm[A.alignerframe[2]];
stretcher = {{xstretch, 0}, {0, ystretch}};

Clear[hangerframe];
    {hangerframe[1], hangerframe[2]} =
  {1/xstretch A.alignerframe[1], 1/ystretch A.alignerframe[2]};
hanger = Transpose[{hangerframe[1], hangerframe[2]}];

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

$$\begin{pmatrix} -0.669767 & -1.9008 \\ -1.91958 & -0.650124 \end{pmatrix}$$

$$\begin{pmatrix} -0.669767 & -1.9008 \\ -1.91958 & -0.650124 \end{pmatrix}$$

Rerun until you feel good.

□B.1.a.ii) What is SVD analysis?

What is SVD analysis of a 2D matrix A?

□Answer:

You just did it.

SVD analysis of a given matrix A is the act of duplicating A with matrix maker ingredients

A = hanger.stretcher.aligner.

□B.1.a.iii) Why SVD analysis is always possible

What guarantees that SVD analysis works?

□Answer:

The discussion above hinges on going with

perpframe[1] = {Cos[s], Sin[s]}
perpframe[2] = {Cos[s + $\frac{\pi}{2}$], Sin[s + $\frac{\pi}{2}$]}

and coming up with an s that makes

(A.perpframe[1]).(A.perpframe[2]) = 0.

To see why this is possible for any 2D matrix A, go with a cleared matrix A:

```
Clear[a, b, c, d];
A = {{a, b}, {c, d}};
MatrixForm[A]
```

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

Go with a cleared candidate for your aligner frame:

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

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

The job is to explain why at least one solution s of

(A.perpframe[1]).(A.perpframe[2]) = 0

is guaranteed.

To do this, look at:

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

```
(b Cos[s] - a Sin[s]) (a Cos[s] + b Sin[s]) +
 (d Cos[s] - c Sin[s]) (c Cos[s] + d Sin[s])
```

This is the same as:

```
Expand[(A.perpframe[1]).(A.perpframe[2])]
```

```
a b Cos[s]^2 + c d Cos[s]^2 - a^2 Cos[s] Sin[s] + b^2 Cos[s] Sin[s] -
 c^2 Cos[s] Sin[s] + d^2 Cos[s] Sin[s] - a b Sin[s]^2 - c d Sin[s]^2
```

```
Simplify[Expand[(A.perpframe[1]).(A.perpframe[2])]]
```

$$\frac{1}{2}\,(2\,(a\,b+c\,d)\,\text{Cos}[2\,s]+(-a^2+b^2-c^2+d^2)\,\text{Sin}[2\,s])$$

This comes from applications of the identities

$$\text{Sin}[2\,x] = 2\,\text{Sin}[x]\,\text{Cos}[x]$$
$$\text{Cos}[2\,x] = \text{Cos}[x]^2 - \text{Sin}[x]^2.$$

This signals that saying that

$$(\text{A.perpframe[1]}).(\text{A.perpframe[2]}) = 0$$

is the same as saying that

$$2\,(a\,b+c\,d)\,\text{Cos}[2\,s] + (-a^2+b^2-c^2+d^2)\,\text{Sin}[2\,s] = 0.$$

So saying that

$$(\text{A.perpframe[1]}).(\text{A.perpframe[2]}) = 0$$

is the same as saying that

$$2\,(a\,b+c\,d)\,\text{Cos}[2\,s] = (a^2-b^2+c^2-d^2)\,\text{Sin}[2\,s]\,.$$

And this the same as saying that

$$\frac{2\,(a\,b+c\,d)}{a^2-b^2+c^2-d^2} = \frac{\text{Sin}[2\,s]}{\text{Cos}[2\,s]}$$

And this the same as saying that

$$\frac{2\,(a\,b+c\,d)}{a^2-b^2+c^2-d^2} = \text{Tan}[2\,s]\,.$$

And because Tan[2 s] takes on all values from -∞ to ∞,

your s is guaranteed.

This formula is not worth memorizing.

□B.1.a.iv) More than one aligner frame

The discussion above tells you that it is always possible to come up with a right hand perpendicular frame that serves as the aligner frame for a given matrix.
Can you get more than one aligner frame for a given matrix?
Is it an issue?

□Answer:

The answer depends on how anal you want to be.

```
A = (Random[Real, {-2, 2}]  Random[Real, {-2, 2}]
     Random[Real, {-2, 2}]  Random[Real, {-2, 2}]);
MatrixForm[A]
```

```
(0.356779  -1.42277
 -1.72738   0.473681)
```

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

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

Now go after the s's that make

$$(\text{A.perpframe[1]}).(\text{A.perpframe[2]}) = 0:$$

```
schoices = Solve[(A.perpframe[1]).(A.perpframe[2]) == 0, s]
```

```
{{s → -2.19896}, {s → -0.628161}, {s → 0.942635}, {s → 2.51343}}
```

This gives you four choices for s:

```
s1 = s /. schoices[[1]]
```

```
-2.19896
```

```
s2 = s /. schoices[[2]]
```

```
-0.628161
```

```
s3 = s /. schoices[[3]]
```

```
0.942635
```

```
s4 = s /. schoices[[4]]
```

```
2.51343
```

See four perpendicular frames all of which will work as aligner frame for this matrix:

```
{perpframe[1], perpframe[2]} /. s -> s1
```

```
{{-0.587658, -0.809109}, {0.809109, -0.587658}}
```

```
{perpframe[1], perpframe[2]} /. s -> s2
```

```
{{0.809109, -0.587658}, {0.587658, 0.809109}}
```

```
{perpframe[1], perpframe[2]} /. s -> s3
```

```
{{0.587658, 0.809109}, {-0.809109, 0.587658}}
```

```
{perpframe[1], perpframe[2]} /. s -> s4
```

```
{{-0.809109, 0.587658}, {-0.587658, -0.809109}}
```

Four right hand aligner frames - but they are just slightly repackaged versions of the first one.

Any of these will work very satisfactorily. And if you want to go with left hand aligner frames you can go with these:

```
{perpframe[1], -perpframe[2]} /. s -> s1
```

```
{{-0.587658, -0.809109}, {-0.809109, 0.587658}}
```

```
{perpframe[1], -perpframe[2]} /. s -> s2
```

```
{{0.809109, -0.587658}, {-0.587658, -0.809109}}
```

```
{perpframe[1], -perpframe[2]} /. s -> s3
```

```
{{0.587658, 0.809109}, {0.809109, -0.587658}}
```

```
{perpframe[1], -perpframe[2]} /. s -> s4
```

```
{{-0.809109, 0.587658}, {0.587658, 0.809109}}
```

Now you've got eight aligner frames..

But one is enough.

B.2) *Mathematica*'s SVD analysis instruction

□B.2.a) Doing SVD analysis with a *Mathematica* instruction

Here's random 2D matrix A:

```
A = (Random[Real, {-2, 2}]  Random[Real, {-2, 2}]
     Random[Real, {-2, 2}]  Random[Real, {-2, 2}]);
MatrixForm[A]
```

```
(-0.365221  -1.65379
 0.0175302  -1.14529)
```

According to B.1) above,you can use SVD analysis to duplicate
A = hanger.stretcher.aligner
this way:

```
Clear[perpframe, s];
    {perpframe[1], perpframe[2]} =
  {{Cos[s], Sin[s]}, {Cos[s + Pi/2], Sin[s + Pi/2]}};
schoices = Solve[(A.perpframe[1]).(A.perpframe[2]) == 0, s];
s = s /. schoices[[2]];

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

xstretch = Norm[A.alignerframe[1]];
ystretch = Norm[A.alignerframe[2]];
stretcher = (xstretch      0
                0      ystretch);
```

```
Clear[hangerframe];
    {hangerframe[1], hangerframe[2]} =
  {1/xstretch A.alignerframe[1], 1/ystretch A.alignerframe[2]};
hanger = Transpose[{hangerframe[1], hangerframe[2]}];

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

```
(-0.365221  -1.65379
 0.0175302  -1.14529)
```

```
(-0.365221  -1.65379
 0.0175302  -1.14529)
```

That's a lot of typing.
To eliminate the need for all this typing, *Mathematica* has an instruction that will help you do this SVD analysis quickly.
How do you use it?

□Answer:

To get an aligner matrix for A you do this:

```
aligner = SingularValues[A][[3]];
MatrixForm[aligner]
```

```
(0.14451   0.989503
 -0.989503  0.14451)
```

To get the corresponding hanger matrix for A you do this:

```
hanger = Transpose[SingularValues[A][[1]]];
MatrixForm[hanger]
```

```
(-0.831007  0.556262
 -0.556262  -0.831007)
```

To get the stretcher matrix for A, you do this:

```
stretcher = DiagonalMatrix[SingularValues[A][[2]]];
MatrixForm[stretcher]
```

```
(2.03273     0
    0     0.220036)
```

It's as easy as 1-2-3.
1 <-----> hanger 2 <-----> stretches 3 <-----> aligner

Check it out:

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

$$\begin{pmatrix} -0.365221 & -1.65379 \\ 0.0175302 & -1.14529 \end{pmatrix}$$

$$\begin{pmatrix} -0.365221 & -1.65379 \\ 0.0175302 & -1.14529 \end{pmatrix}$$

Yep. It's as easy as 1-2-3.

If you want the individual vectors in the calculated aligner frame, do this:

```
Clear[alignerframe, k];
alignerframe[k_] := SingularValues[A][[3, k]];
Table[alignerframe[k], {k, 1, 2}]
```

```
{{0.14451, 0.989503}, {-0.989503, 0.14451}}
```

If you want xstretch and ystretch here they are:

```
{xstretch, ystretch} = SingularValues[A][[2]]
```

```
{2.03273, 0.220036}
```

If you want the individual vectors in the calculated hanger frame, do this:

```
Clear[hangerframe, k];
hangerframe[k_] := SingularValues[A][[1, k]];
Table[hangerframe[k], {k, 1, 2}]
```

```
{{-0.831007, -0.556262}, {0.556262, -0.831007}}
```

Check everything out:

```
MatrixForm[A]
```

$$\begin{pmatrix} -0.365221 & -1.65379 \\ 0.0175302 & -1.14529 \end{pmatrix}$$

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

$$\begin{pmatrix} -0.365221 & -1.65379 \\ 0.0175302 & -1.14529 \end{pmatrix}$$

```
{A.alignerframe[1], xstretch hangerframe[1]}
```

```
{{-1.68921, -1.13073}, {-1.68921, -1.13073}}
```

```
{A.alignerframe[2], ystretch hangerframe[2]}
```

```
{{0.122398, -0.182851}, {0.122398, -0.182851}}
```

Everything checks.

Another way for the winning team of mathematics and *Mathematica* to serve you.

B.3) Using SVD analysis and the fact that if

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

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

to invert 2D matrices.

Using SVD analysis to recognize and deal with non-invertible matrices

☐B.3.a.i) Using SVD analysis to determine whether a matrix is invertible

Here's a random 2D matrix A:

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

$$\begin{pmatrix} 3.48851 & 2.97982 \\ 1.10982 & -1.51059 \end{pmatrix}$$

Use SVD analysis to determine whether this matrix is invertible.

☐Answer:

Check the SVD stretch factors:

```
{xstretch, ystretch} = SingularValues[A][[2]]
```

```
{4.59038, 1.86842}
```

```
If by some fluke,one of the stretch factors turns out to 0,
        then rerun all code until neither is 0.
```

Neither stretch factor is 0.

This tells you that is matrix is invertible.

☐B.3.a.ii) Using SVD analysis and the fact that

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

to construct the inverse

Stay with the same matrix A as in part i)
Here's *Mathematica*'s calculation of A^{-1}:

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

$$\begin{pmatrix} 0.176126 & 0.347429 \\ 0.129398 & -0.40674 \end{pmatrix}$$

Use SVD analysis and matrix maker ingredients to explain where this calculation comes from.

☐Answer:

Do SVD analysis of A by writing A in the form

A = hanger. stretcher. aligner:

```
hanger = Transpose[SingularValues[A][[1]]];
MatrixForm[hanger]
```

$$\begin{pmatrix} -0.999358 & 0.0358396 \\ 0.0358396 & 0.999358 \end{pmatrix}$$

```
stretcher = DiagonalMatrix[SingularValues[A][[2]]];
MatrixForm[stretcher]
```

$$\begin{pmatrix} 4.59038 & 0 \\ 0 & 1.86842 \end{pmatrix}$$

```
aligner = SingularValues[A][[3]];
MatrixForm[aligner]
```

$$\begin{pmatrix} -0.750808 & -0.660521 \\ 0.660521 & -0.750808 \end{pmatrix}$$

Now remembering that

->the hanger for A^{-1} is the transpose of the aligner for A

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

->the aligner for A^{-1} is the transpose of the hanger for A,

you build

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

```
unstretcher = {{1/xstretch, 0}, {0, 1/ystretch}};
SVDinverse = Transpose[aligner].unstretcher.Transpose[hanger];
MatrixForm[SVDinverse]
```

$$\begin{pmatrix} 0.176126 & 0.347429 \\ 0.129398 & -0.40674 \end{pmatrix}$$

Compare:

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

$$\begin{pmatrix} 0.176126 & 0.347429 \\ 0.129398 & -0.40674 \end{pmatrix}$$

Not much to it when you understand what to do with the matrix maker ingredients.

☐B.3.a.iii) The moral

What's the moral?

☐Answer:

You can use SVD analysis to build the inverse of any invertible matrix.

☐B.3.b.i) Using SVD analysis to recognize noninvertible matrices

Here's another sample 2D matrix A:

```
A = {{1.2, -3.6}, {0.4, -1.2}};
MatrixForm[A]
```

$$\begin{pmatrix} 1.2 & -3.6 \\ 0.4 & -1.2 \end{pmatrix}$$

Here's *Mathematica*'s attempt at a calculation of A^{-1}:

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

```
Inverse::luc : Result for Inverse of badly conditioned matrix
  {{1.2, -3.6}, {0.4, -1.2}} may contain significant numerical errors.
```

$$\begin{pmatrix} -4.82529 \times 10^{15} & 1.44759 \times 10^{16} \\ -1.60843 \times 10^{15} & 4.82529 \times 10^{15} \end{pmatrix}$$

Garbage.

Use SVD analysis to explain why that happened.

☐Answer:

Look at the SVD stretch factors of A:

```
SingularValues[A][[2]]
```

```
{4.}
```

Mathematica reports only one non-zero stretch factor.

This is *Mathematica*'s way of telling you that one of the stretch factors is 0.

And this tells you that A is not invertible.

□B.3.b.ii) Using SVD analysis to determine the line on which a noninvertible matrix hangs all its hits

Stay with the same noninvertible matrix A as in part i).
Use SVD analysis to determine the line on which A hangs all its hits. Back up your answer with a convincing plot.

□**Answer:**

Look at:

```
alignerframe = SingularValues[A][[3]]
```
```
{{-0.316228, 0.948683}}
```

Mathematica delivers only the alignerframe vector whose corresponding stretch factor is not 0.

The matrix A hangs all its hits on the line with direction unit vector

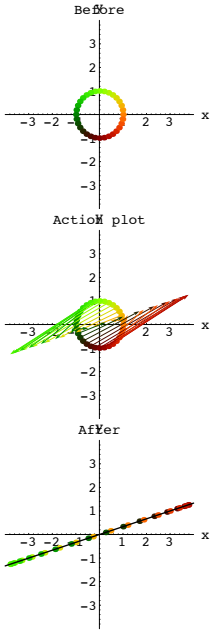
```
linedirection = A. alignerframe[[1]] / Norm[A. alignerframe[[1]]]
```
```
{-0.948683, -0.316228}
```

This is the same as hangerframe[1].

```
SingularValues[A][[1]][[1]]
```
```
{-0.948683, -0.316228}
```

Watch it happen in this action movie:

```
Clear[hitpointplotter,
  pointcolor, actionarrows, matrix2D, r, x, y, t];
ranger = Max[Append[SingularValues[A][[2]], 1.5]];
{x[t_], y[t_]} = {Cos[t], Sin[t]};

pointcolor[t_] = RGBColor[0.5 (Cos[t] + 1), 0.5 (Sin[t] + 1), 0];
{tlow, thigh} = { 0, 2 π};
tjump = (thigh - tlow) / 32;


hitpointplotter[matrix2D_] :=
  Table[Graphics[{pointcolor[t], PointSize[0.03],
     Point[matrix2D.{x[t], y[t]}]}], {t, tlow, thigh - tjump, tjump}];
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 - tjump, tjump}];

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

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


after = Show[hitpointplotter[A],
   Graphics[Line[{-2 ranger linedirection, 2 ranger linedirection}]],
   PlotLabel → "After", Axes → True, AxesLabel → {"x", "y"},
   PlotRange → {{-ranger, ranger}, {-ranger, ranger}},
   DisplayFunction → $DisplayFunction];
```



Grab and animate.

That black line is the line through {0,0} whose direction is defined by:

```
hangerframe[1]
```
```
{-0.948683, -0.316228}
```

The noninvertible matrix A hangs all its hits in this line.

B.4) Linear Algebra: If the coefficient matrix A is invertible , then given {u,v} you can solve

$$A.\{x,y\} = \{u,v\}$$

for {x,y} by putting

$$\{x,y\} = A^{-1}.\{u,v\}.$$

If the coefficient matrix A is not invertible , saying you can solve

$$A.\{x,y\} = \{u,v\}$$

for {x,y} is the same as saying {u.v} is on the line through {0,0} running in the direction of hangerframe[1].

□B.4.a) When the coefficient matrix A is invertible , you can solve

$$A.\{x,y\} = \{u,v\}$$

for {x,y} by putting

$$\{x,y\} = A^{-1}.\{u,v\}$$

Use what you know about matrices to try come up with the x and the y that solve the simultaneous linear equations:
2.1 x - 1.7 y = 9.8
1.8 x + 0.6 y = - 1.5.

□**Answer:**

Go with this matrix:

```
Fancy folks like to call this the "coefficient matrix."
```
```
A = {{2.1, -1.7}, {1.8, 0.6}};
MatrixForm[A]
```
$$\begin{pmatrix} 2.1 & -1.7 \\ 1.8 & 0.6 \end{pmatrix}$$

The linear system of equations is

2.1 x - 1.7 y = 9.8
1.8 x + 0.6 y = - 1.5.

This is the same as

A.{x,y} = {9.8 , -1.5}:

```
Clear[x, y];
A.{x, y} == {9.8, -1.5}
```
```
{2.1 x - 1.7 y, 1.8 x + 0.6 y} == {9.8, -1.5}
```

To come up with the x and the y that solve this system, just start with

A.{x,y} = {9.8 , -1.5}:

and hit both sides with A^{-1} to get

$\{x,y\} = A^{-1}A.\{x,y\} = A^{-1}.\{9.8,-1.5\}$:

```
{xsol, ysol} = Inverse[A].{9.8, -1.5}
```
```
{0.770833, -4.8125}
```

This is the solution.

Check:

```
A.{xsol, ysol} == {9.8, -1.5}
```
```
True
```

Got it.

This approach works anytime the coefficient matrix is invertible.

□B.4.b.i) When the coefficient matrix A is not invertible , you can solve

$$A.\{x,y\} = \{u,v\}$$

only if {u,v} is on the line through {0,0} in the direction of hangerframe [1]

Here's a matrix which is not invertible:

```
A = {{0.6, -0.3}, {0.8, -0.4}};
MatrixForm[A]
```
$$\begin{pmatrix} 0.6 & -0.3 \\ 0.8 & -0.4 \end{pmatrix}$$

```
Inverse[A]
```
```
Inverse::luc : Result for Inverse of badly conditioned matrix
  {{0.6, -0.3}, {0.8, -0.4}} may contain significant numerical errors.
```
$$\{\{1.80144 \times 10^{16}, -1.35108 \times 10^{16}\}, \{3.60288 \times 10^{16}, -2.70216 \times 10^{16}\}\}$$

That's garbage.
In spite of this, the corresponding linear system, for given u and v,
3.0 x - 1.5 y = u
1.0 x - 0.5 y = v
might have many or no solutions for x and y, depending on where the point {u,v} is

located.
How do you make this determination?

□ Answer:

Begin your SVD analysis of A:

```
stretches = SingularValues[A][[2]]
```
```
{1.11803}
```

Only one non-zero stretch factor. (As expected because A is not invertible.)

Let *Mathematica* give you the hangerframe vector corresponding to this lone stretch factor:

```
hangerframe[1] = SingularValues[A][[1]][[1]]
```
```
{-0.6, -0.8}
```

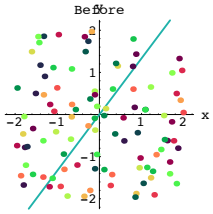
```
Remember 1-2-3:
Hanger-Stretcher-Aligner
```

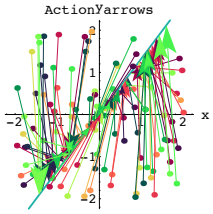
Hitting a point {x,y} with A will result in a point hung on the line through {0,0} determined by hangerframe[1].

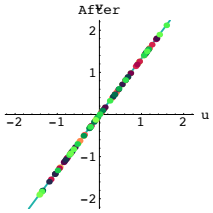
What it happen in this action movie:

```
ranger
```
```
4.
```
```
a Max[SingularValues[A][[2]]]
```
```
1.11803 a
```
```
a = 2;
points =
  Table[{Random[Real, {-a, a}], Random[Real, {-a, a}]}, {k, 1, 100}];

ranger = a Max[SingularValues[A][[2]]];

pointcolor[k_] = RGBColor[0.5 (Sin[2 π k / Length[points]] + 1),
   0.5 (Cos[2 π k / Length[points]] + 1), 0.3];

pointplot = Table[
   Graphics[{PointSize[0.03], pointcolor[k], Point[points[[k]]]}],
   {k, 1, Length[points]}];

hitpointplot = Table[
   Graphics[{PointSize[0.03], pointcolor[k], Point[A.points[[k]]]}],
   {k, 1, Length[points]}];

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

```
b = 3;
lineplot = Graphics[{LightSeaGreen, Thickness[0.01],
    Line[{-b hangerframe[1], b hangerframe[1]}]}];

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

Show[before, actionarrows,
  PlotRange → {{-ranger, ranger}, {-ranger, ranger}},
  PlotLabel → "Action arrows"];

after =
  Show[lineplot, hitpointplot, Axes → True, AxesLabel → {"u", "v"},
   PlotRange → {{-ranger, ranger}, {-ranger, ranger}},
   PlotLabel → "After"];
```








```
Grab and animate.
```

The plotted line is the line through {0,0} defined by hangerframe[1].

The upshot:

Given {u,v}, if you are going to have any chance of solving

A.{x,y} = {u,v}

for {x,y}, then {u,v} **MUST** be on the line through {0,0} defined by hangerframe[1].

This is the same as saying that {u,v} is its own component in the direction of hangerframe[1].

And this is the same as saying

{u,v} = ({u,v}.hangerframe[1]) hangerframe[1]

□ B.4.b.ii) Failure because {u,v} is not on the line through {0,0} in the direction of hangerframe [1]

Stay with the same matrix A as in part i) above.
Go with
{u,v} = {1.00,1.45}
Does the linear system
A.{x,y} = {u,v},
(which is the same as
3.0 x - 1.5 y = u
1.0 x - 0.5 y = v)
have a solution for x and y?

□ Answer:

Remember that if you are going to have any chance of solving

A.{x,y} = {u,v}

for {x,y}, then {u,v} MUST be on the line through {0,0} defined by hangerframe[1].

This is the same as saying

{u,v} = ({u,v}.hangerframe[1]) hangerframe[1].

Check this out:

```
{u, v} = {1.00, 1.45}
```
```
{1., 1.45}
```
```
({u, v}.hangerframe[1]) hangerframe[1]
```
```
{1.056, 1.408}
```

Close, but no cigar.

The linear system

A.{x,y} = {1.00,1.45}

has NO solutions for x and y.

□ B.4.b.iii) Success because {u,v} is on the line through {0,0} in the direction of hangerframe [1]

Stay with the same matrix A as in part i) above.
Go with
{u,v} = {2.46, 3.28}
Does the linear system
A.{x,y} = {u,v},
(which is the same as
3.0 x - 1.5 y = u
1.0 x - 0.5 y = v)
have a solution for x and y?

□ Answer:

Check:

```
{u, v} = {2.46, 3.28}
```
```
{2.46, 3.28}
```
```
({u, v}.hangerframe[1]) hangerframe[1]
```
```
{2.46, 3.28}
```

These agree.

This tells you that the system

A.{x,y} = {u,v} = ({u,v}.hangerframe[1]) hangerframe[1]

DOES have a solution {x,y}.

□B.4.b.iv) Coming up with a solution {x,y} through the formula

$$\{x, y\} = \frac{(\{u,v\}.\text{hangerframe}[1])\ \text{alignerframe}[1]}{\text{xstretch}}$$

Stay with the same matrix A as in part i) above.
At this point you know that if

{u,v} = {2.46, 3.28},

the linear system

A.{x,y} = {u,v}

does have a solution {x,y}.
Here's how you can get one .

```
Clear[hangerframe];
hangerframe[1] = SingularValues[A][[1]][[1]]
```
```
{-0.6, -0.8}
```
```
Clear[alignerframe];
alignerframe[1] = SingularValues[A][[3]][[1]]
```
```
{-0.894427, 0.447214}
```
```
xstretch = SingularValues[A][[2]][[1]]
```
```
1.11803
```

You build a solution {x,y} this way:

```
Xsol = (({u, v}.hangerframe[1]) alignerframe[1]) / xstretch
```
```
{3.28, -1.64}
```

Try it out:

```
A.Xsol
```
```
{2.46, 3.28}
```
```
{u, v}
```
```
{2.46, 3.28}
```

Explain why that worked.

□**Answer:**

You know

{u, v} = ({u, v}.hangerframe[1]) hangerframe[1].

And you know

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

This is the same as

$$A.\left(\frac{\text{alignerframe}[1]}{\text{xstretch}}\right) = \text{hangerframe}[1].$$

And this is the same as

$$A.\left(\frac{(\{u,v\}.\text{hangerframe}[1])\ \text{alignerframe}[1]}{\text{xstretch}}\right) = (\{u, v\}.\text{hangerframe}[1])\ \text{hangerframe}[1].$$

And this is the same as

$$A.\left(\frac{(\{u,v\}.\text{hangerframe}[1])\ \text{alignerframe}[1]}{\text{xstretch}}\right) = \{u, v\}$$

because .

{u, v} = ({u, v}.hangerframe[1]) hangerframe[1].

The upshot:

When you hit A on

$$\frac{(\{u,v\}.\text{hangerframe}[1])\ \text{alignerframe}[1]}{\text{xstretch}},$$

you are guaranteed to get {u, v}.

That's where the formula:

```
Xsol = ({u, v}.hangerframe[1] alignerframe[1]) / xstretch
```
```
{3.28, -1.64}
```

comes from.

□B.4.b.v) Success comes in spades

Stay with the same matrix A as in part iii) above. And stay with the same {x,y} and {u,v} as in part iii) so that

A.{x,y} = {u,v}

Are there solutions other than the {x,y} you found in part iii)?
If so, indicate with a plot where they are.

□**Answer:**

You bet there are!

There are infinitely many.

To see where they come from, remember that ystretch = 0. This tells you that hits with A collapse everything perpendicular to alignerframe[1] to {0,0}.

Look at:

```
alignerframe[1]
```
```
{-0.894427, 0.447214}
```

A vector perpendicular to alignerframe[1] is:

```
alignerframe[2] = {alignerframe[1][[2]], -alignerframe[1][[1]]}
```
```
{0.447214, 0.894427}
```

Check:

```
A.alignerframe[2]
```
```
{0, 0}
```

Consequently when you go with the Xsol you got in the last part and add to it any multiple of alignerframe[2], you get another solution:

```
Clear[t];
Chop[Expand[A.(Xsol + t alignerframe[2])]]
```
```
{2.46, 3.28}
```
```
{u, v}
```
```
{2.46, 3.28}
```

The upshot:

All the solutions of the linear system

A.{x,y} = {u,v}

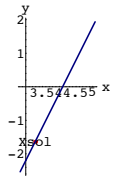
are located on the line through {x,y} with direction vector alignerframe[2].

These are the points parameterized by:

```
Clear[solutionline, t];
solutionline[t_] = Xsol + t alignerframe[2]
```
```
{3.28 + 0.447214 t, -1.64 + 0.894427 t}
```

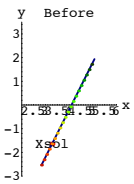
Here's a look at part of this line:

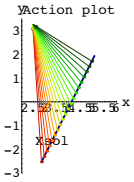
```
basepoint = Xsol;
basepointplot =
  Graphics[{Red, PointSize[0.05], Point[basepoint]}];


solutionlineplot = Graphics[{NavyBlue, Thickness[0.02],
    Line[{solutionline[-1], solutionline[4]}]}];

Xsollabel = Graphics[Text["Xsol", basepoint]];

solutionlineplot = Show[basepointplot, Xsollabel, solutionlineplot,
   Axes → True, PlotRange → All, AxesLabel → {"x", "y"}];
```

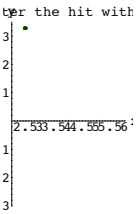


See an action movie showing A hit all points on the line smack-dab onto {u,v}:

```
Clear[hitpointplotter,
  pointcolor, actionarrows, matrix2D, r, x, y, t];
{x[t_], y[t_]} = solutionline[t];

pointcolor[t_] = RGBColor[0.5 (Cos[t] + 1), 0.5 (Sin[t] + 1), 0];
{tlow, thigh} = {-1, 4};
tjump = (thigh - tlow) / 20;


hitpointplotter[matrix2D_] :=
  Table[Graphics[{pointcolor[t], PointSize[0.03],
     Point[matrix2D.{x[t], y[t]}]}], {t, tlow, thigh - tjump, tjump}];
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 - tjump, tjump}];

before = Show[solutionlineplot, hitpointplotter[IdentityMatrix[2]],
   PlotLabel → "Before", Axes → True,
   AxesLabel → {"x", "y"}, PlotRange → {{2, 6}, {-3, 3.5}},
   DisplayFunction → $DisplayFunction];

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


after = Show[hitpointplotter[A],
   PlotLabel → "After the hit with A", Axes → True,
   AxesLabel → {"x", "y"}, PlotRange → {{2, 6}, {-3, 3.5}},
   DisplayFunction → $DisplayFunction];
```







Grab and animate.

Think of it:

All points {x,y} on the plotted line are solutions of

A.{x,y} = {u,v}.

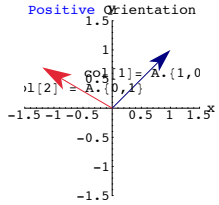
The moral:

If A is not invertible and

A.{x,y} = {u,v}

has any one solution then it has bundles of solutions.

It's enough to make you want to freak out.

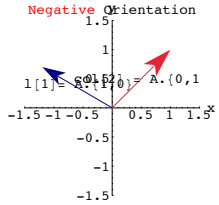


$$A = \begin{pmatrix} 1. & -1.2 \\ 1. & 0.7 \end{pmatrix}$$

Matrix folks say that the orientation of the columns of A is positive.

Now look at the columns of $A = \begin{pmatrix} -1.2 & 1.0 \\ 0.7 & 1.0 \end{pmatrix}$

```
A = {{-1.2, 1.0}, {0.7, 1.0}};
Show[columnplotter[A], DisplayFunction -> $DisplayFunction];
"A =" MatrixForm[A]
```



$$A = \begin{pmatrix} -1.2 & 1. \\ 0.7 & 1. \end{pmatrix}$$

Look at the same thing for random matrices A:

```
Clear[a];
a[i_, j_] := ((-1)^Random[Integer,{0,1}]) Random[Real, {0.5, 1.5}]

A = {{a[1, 1], a[1, 2]}, {a[2, 1], a[2, 2]}};
Show[columnplotter[A], DisplayFunction -> $DisplayFunction];
"A = " MatrixForm[A]
```

B.5) Positive versus negative orientation of the columns of a 2D matrix.

If the columns of A are positively oriented, then Det[A] = + xstretch ystretch.

If the columns of A are negatively oriented, then Det[A] = — xstretch ystretch.

So you can use |Det[A]| = xstretch ystretch as the area conversion factor.

If you interchange the vertical columns of a 2D matrix, you change the sign but not the absolute value of the determinant.

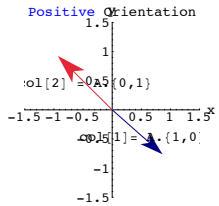
The short, sweet formula Det[$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$] = a d - b c

□B.5.a) Positive versus negative orientation of the columns of a 2D matrix

Here is a 2D matrix A together with a plot of the vertical columns of A:

col[1] = A.{1,0} and col[2] = A.{0,1}

```
Clear[columnplotter, matrix];
columnplotter[matrix_] :=
 Show[Arrow[matrix.{1, 0}, Tail → {0, 0}, VectorColor → NavyBlue,
   HeadSize → 0.4], Arrow[matrix.{0, 1}, Tail → {0, 0},
   VectorColor → AlizarinCrimson, HeadSize → 0.5],
  Graphics[{Text["col[1]= A.{1,0}", 0.6 matrix.{1, 0}]}],
  Graphics[{Text["col[2] = A.{0,1}", 0.5 matrix.{0, 1}]}],
  Axes → True, AxesLabel → {"x", "y"},
  PlotRange -> {{-1.5, 1.5}, {-1.5, 1.5}},
  PlotLabel -> If[ Det[A] > 0, "Positive Orientation",
    If[Det[A] < 0, "Negative Orientation", "No Orientation"]],
  DisplayFunction -> Identity];
A = {{1.0, -1.2}, {1.0, 0.7}};


Show[columnplotter[A], DisplayFunction -> $DisplayFunction];
"A =" MatrixForm[A]
```



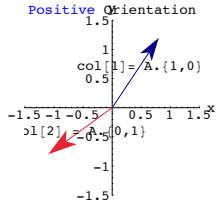
$$A = \begin{pmatrix} 0.865405 & -0.933637 \\ -0.763728 & 0.926544 \end{pmatrix}$$

Rerun many times and then say what positive and negative orientation means.

□Answer:

Take a look at this one:

```
A = {{0.8, -1.1}, {1.2, -0.8}};
Show[columnplotter[A], DisplayFunction -> $DisplayFunction];
"A = " MatrixForm[A]
```

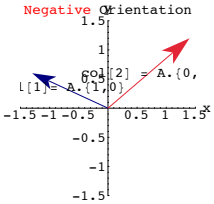


$$A = \begin{pmatrix} 0.8 & -1.1 \\ 1.2 & -0.8 \end{pmatrix}$$

This is **positive** orientation and the **shorter angle** from col[1] to col[2] is **counterclockwise**.

And take a look at this one:

```
A = {{-1.3, 1.4}, {0.6, 1.2}};
Show[columnplotter[A], DisplayFunction -> $DisplayFunction];
"A = " MatrixForm[A]
```



$$A = \begin{pmatrix} -1.3 & 1.4 \\ 0.6 & 1.2 \end{pmatrix}$$

This is **negative** orientation and the **shorter angle** from col[1] to col[2] is **clockwise**.

That's all there is to it.

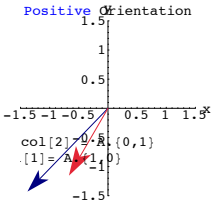
The columns of a 2D matrix A are **positively oriented** if the **shorter angle** from col[1] to col[2] is **counterclockwise**.

The columns of a 2D matrix A are **negatively oriented** if the **shorter angle** from col[1] to col[2] is **clockwise**.

If col[1] and col[2] point in opposite or the same directions, folks say that they are not oriented at all.

Try it again for random matrices A:

```
Clear[a];
a[i_, j_] := ((-1)^Random[Integer,{0,1}]) Random[Real, {0.5, 1.5}]

A = ( a[1, 1]  a[1, 2] );
    ( a[2, 1]  a[2, 2] )
Show[columnplotter[A], DisplayFunction -> $DisplayFunction];
"A = " MatrixForm[A]
```



$$A = \begin{pmatrix} -1.38211 & -0.67017 \\ -1.42922 & -1.15602 \end{pmatrix}$$

Rerun until the idea crystallizes.

□B.5.b.i) Definition of determinant first part:

If the columns of A are positively oriented, then Det[A] = + xstretch ystretch

Here's a 2D matrix A together with *Mathematica*'s calculation of the determinant of A:

```
A = ( 2.2  0.9 );
    ( 0.5  1.7 )
Det[A]
```

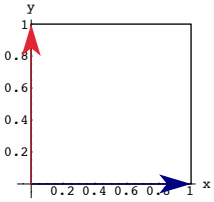
3.29

What does this calculation of Det[$\begin{pmatrix} 2.2 & 0.9 \\ 0.5 & 1.7 \end{pmatrix}$] measure?

□Answer:

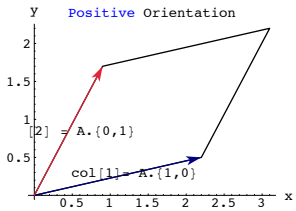
Go with the unit square with corners at {0,0},{1,0},{1,1} and {0,1}:

```
unitsquare =
 Show[Graphics[Line[{{0, 0}, {1, 0}, {1, 1}, {0, 1}, {0, 0}}]], Arrow[
   {1, 0}, Tail → {0, 0}, VectorColor → NavyBlue, HeadSize → 0.2],
  Arrow[{0, 1}, Tail → {0, 0}, VectorColor → AlizarinCrimson,
   HeadSize → 0.2], Axes → True, AxesLabel → {"x", "y"}];
```



Here's what you get when you hit everything with $A = \begin{pmatrix} 2.2 & 0.9 \\ 0.5 & 1.7 \end{pmatrix}$:

```
A = ( 2.2  0.9 );
    ( 0.5  1.7 )
Clear[hitplotter, matrix];
hitplotter[matrix_] :=
 Show[Graphics[Line[{matrix.{0, 0}, matrix.{1, 0},
     matrix.{1, 1}, matrix.{0, 1}, matrix.{0, 0}}]],
  Arrow[matrix.{1, 0}, Tail → {0, 0}, VectorColor → NavyBlue,
   HeadSize → 0.2], Arrow[matrix.{0, 1}, Tail → {0, 0},
   VectorColor → AlizarinCrimson, HeadSize → 0.2],
  Graphics[{Text["col[1]= A.{1,0}", 0.6 matrix.{1, 0}]}],
  Graphics[{Text["col[2] = A.{0,1}", 0.5 matrix.{0, 1}]}],
  PlotLabel -> If[ Det[matrix] > 0, "Positive Orientation",
    If[Det[matrix] < 0, "Negative Orientation", "No Orientation"]],
  Axes → True, AxesLabel → {"x", "y"}];
hitplotter[A];
```



That's the parallelogram defined by the columns of A.

As you can see the columns of A have positive orientation.

If any given 2D matrix A has positively oriented columns,

Det[A] is defined to be the area measurement of the parallelogram defined by the columns of A.

Because the area of the square measures out to 1 and you hit A on the square to get the parallelogram,

Det[A] is given by:

```
{xstretch, ystretch} = SingularValues[A][[2]];
detA = xstretch ystretch
```

3.29

Check:

```
Det[A]
```

3.29

On the money.

□B.5.b.ii) Definition of determinant second part:

If the columns of A are negatively oriented, then Det[A] = −xstretch ystretch

Here's another 2D matrix A together with Mathematica's calculation of the determinant of A:

```
A = ( 0.9  2.2 );
    ( 1.7  0.5 )
Det[A]
```

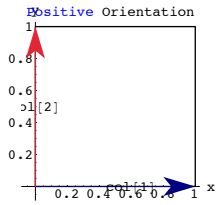
-3.29

What does this calculation of Det[$\begin{pmatrix} 0.9 & 2.2 \\ 1.7 & 0.5 \end{pmatrix}$] measure?

□Answer:

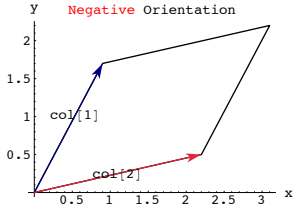
Go with the unit square with corners at {0,0},{1,0},{1,1} and {0,1}:

```
Clear[hitplotter, matrix];
hitplotter[matrix_] :=
 Show[Graphics[Line[{matrix.{0, 0}, matrix.{1, 0},
     matrix.{1, 1}, matrix.{0, 1}, matrix.{0, 0}}]],
  Arrow[matrix.{1, 0}, Tail → {0, 0}, VectorColor → NavyBlue,
   HeadSize → 0.2], Arrow[matrix.{0, 1}, Tail → {0, 0},
   VectorColor → AlizarinCrimson, HeadSize → 0.2],
  Graphics[{Text["col[1]", 0.6 matrix.{1, 0}]}],
  Graphics[{Text["col[2]", 0.5 matrix.{0, 1}]}],
  PlotLabel -> If[ Det[matrix] > 0, "Positive Orientation",
    If[Det[matrix] < 0, "Negative Orientation", "No Orientation"]],
  Axes → True, AxesLabel → {"x", "y"}];
hitplotter[IdentityMatrix[2]];
```



Here's the parallelogram you get when you hit this square with $A = \begin{pmatrix} 0.9 & 2.2 \\ 1.7 & 0.5 \end{pmatrix}$:

```
A = ( 0.9  2.2 );
    ( 1.7  0.5 )
hitplotter[A];
```



That's the parallelogram defined by the columns of A.

As you can see, the columns of A have negative orientation.

If any given 2D matrix A has negatively oriented columns,

-Det[A] is defined to be the area measurement of the parallelogram defined by the columns of A.

Because the area of the square measures out to 1 and you hit A on the square to get the parallelogram,

Det[A] is given by:

```
{xstretch, ystretch} = SingularValues[A][[2]];
minusdetA = xstretch ystretch
```

```
3.29
```

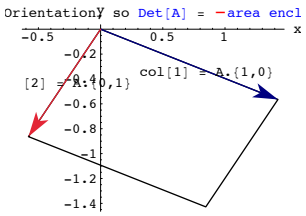
Check:

```
-Det[A]
```

```
3.29
```

Try some more for random matrices A:

```
Clear[a];
a[i_, j_] := ((-1)^Random[Integer,{0,1}]) Random[Real, {0.5, 1.5}]

A = (a[1, 1]  a[1, 2]
     a[2, 1]  a[2, 2]);
Clear[hitplotter, matrix];
hitplotter[matrix_] :=
 Show[Graphics[Line[{matrix.{0, 0}, matrix.{1, 0},
     matrix.{1, 1}, matrix.{0, 1}, matrix.{0, 0}}]],
  Arrow[A.{1, 0}, Tail → {0, 0}, VectorColor → NavyBlue,
   HeadSize → 0.2],  Arrow[A.{0, 1}, Tail → {0, 0},
   VectorColor → AlizarinCrimson, HeadSize → 0.2],
  Graphics[{Text["col[1] = A.{1,0}", 0.6 matrix.{1, 0}]}],
  Graphics[{Text["col[2] = A.{0,1}", 0.5 matrix.{0, 1}]}],
  PlotLabel -> If[ Det[A] > 0,
    "Pos Orientation, so Det[A] = area enclosed",
    If[Det[A] < 0, "Neg Orientation, so Det[A] = -area enclosed",
     "No Orientation"]], Axes → True, AxesLabel → {"x", "y"}];
hitplotter[A];
```

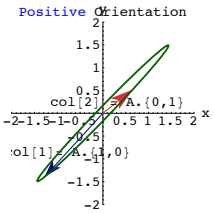


Rerun until you feel good.

□B.5.b.iii) Using the determinant to measure area

Here's a random 2D matrix A together with the ellipse you get when you hit the unit circle with A:

```
Clear[a];
a[i_, j_] := ((-1)^Random[Integer,{0,1}]) Random[Real, {0.5, 1.5}]

A = (a[1, 1]  a[1, 2]
     a[2, 1]  a[2, 2]);
ellipseplot = ParametricPlot[A.{Cos[t], Sin[t]},
   {t, 0, 2 Pi}, PlotStyle -> {{GosiaGreen, Thickness[0.01]}},
   DisplayFunction -> Identity];
Clear[columnplotter, matrix];

columnplotter[matrix_] :=
 Show[Arrow[matrix.{1, 0}, Tail → {0, 0}, VectorColor → NavyBlue,
   HeadSize → 0.4], Arrow[matrix.{0, 1}, Tail → {0, 0},
   VectorColor → AlizarinCrimson, HeadSize → 0.5],
  Graphics[{Text["col[1]= A.{1,0}", 0.6 matrix.{1, 0}]}],
  Graphics[{Text["col[2] = A.{0,1}", 0.5 matrix.{0, 1}]}],
  Axes → True, AxesLabel → {"x", "y"}, PlotRange -> {{-2, 2}, {-2, 2}},
  PlotLabel -> If[ Det[A] > 0, "Positive Orientation",
    If[Det[A] < 0, "Negative Orientation", "No Orientation"]],
  DisplayFunction -> Identity];



Show[columnplotter[A], ellipseplot,
  DisplayFunction -> $DisplayFunction];
"A =" MatrixForm[A]
```



$$A = \begin{pmatrix} -1.28453 & 0.653318 \\ -1.39979 & 0.518926 \end{pmatrix}$$

Rerun a couple of times until you get a nice inviting ellipse.

The area enclosed by this ellipse measures out to:

```
{xstretch, ystretch} = SingularValues[A][[2]];
xstretch ystretch π
```

```
0.778901
```

Use the determinant of A to duplicate this calculation.

□Answer:

Just go with |Det[A]| π:

```
Abs[Det[A]] π
```

```
0.778901
```

The reasons:

If the columns of A are positively oriented,then Det[A] = + xstretch ystretch.

If the columns of A are negatively oriented,then Det[A] = - xstretch ystretch.

Either way: |Det[A]| = xstretch ystretch.

□B.5.b.iv) If you interchange the vertical columns of a 2D matrix, you change the sign but not the absolute value of the determinant

Here's a random 2D matrix A:

```
Clear[a];
a[i_, j_] := ((-1)^Random[Integer,{0,1}]) Random[Real, {0.5, 1.5}]

A = (a[1, 1]  a[1, 2]
     a[2, 1]  a[2, 2]);
MatrixForm[A]
```

$$\begin{pmatrix} -1.36806 & 0.504842 \\ -1.33828 & -1.31946 \end{pmatrix}$$

Here's the matrix you get when you interchange the two columns of A:

```
interchangedA = A.(0   1.0
                   1.0 0   );
MatrixForm[interchangedA]
```

$$\begin{pmatrix} 0.504842 & -1.36806 \\ -1.31946 & -1.33828 \end{pmatrix}$$

Calculate the determinants of each:

```
{Det[A], Det[interchangedA]}
```

```
{2.48072, -2.48072}
```

The result:

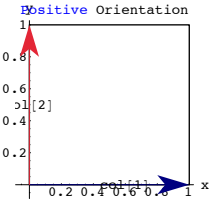
Det[interchangedA] = - Det[A].

Explain why it had to turn out this way

□Answer:

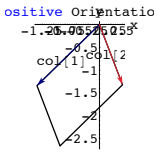
Go with the unit square with corners at {0,0},{1,0},{1,1} and {0,1}:

```
Clear[hitplotter, matrix];
hitplotter[matrix_] :=
 Show[Graphics[Line[{matrix.{0, 0}, matrix.{1, 0},
     matrix.{1, 1}, matrix.{0, 1}, matrix.{0, 0}}]],
  Arrow[matrix.{1, 0}, Tail → {0, 0}, VectorColor → NavyBlue,
   HeadSize → 0.2],  Arrow[matrix.{0, 1}, Tail → {0, 0},
   VectorColor → AlizarinCrimson, HeadSize → 0.2],
  Graphics[{Text["col[1]", 0.6 matrix.{1, 0}]}],
  Graphics[{Text["col[2]", 0.5 matrix.{0, 1}]}],
  PlotLabel -> If[ Det[matrix] > 0, "Positive Orientation",
    If[Det[matrix] < 0, "Negative Orientation", "No Orientation"]],
  Axes → True, AxesLabel → {"x", "y"}];
hitplotter[IdentityMatrix[2]];
```



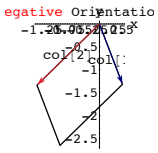
Here's the parallelogram you get when you hit everything with A:

```
Ahit = hitplotter[A];
```



Here's the parallelogram you get when you hit everything with interchangedA:

```
interchangedAhit = hitplotter[interchangedA];
```

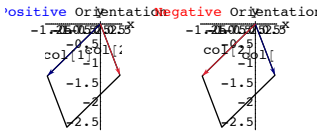


Grab and animate both plots.

That's the parallelogram defined by the columns of interchangedA.

Notice that both parallelograms are the same physical parallelograms.

```
Show[GraphicsArray[{Ahit, interchangedAhit}]];
```



Reason: They are both defined by the same vectors.

So

enclosed area = |Det[A]| = |Det[interchangedA]|.

But the orientation of the columns of interchangedA is opposite to the orientation of the columns of A.

So Det[A] and Det[interchangedA] have opposite signs.

□) **B.5.c) The short, sweet formula Det[$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$] = a d - b c**

Look at *Mathematica*'s calculation of the determinant of a cleared matrix A = $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$:

```
Clear[a, b, c, d];
A = {{a, b}, {c, d}};
Det[A]
```

```
-b c + a d
```

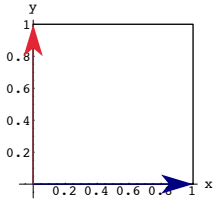
Explain where this short, sweet formula comes from.

□ **Answer:**

Case 1: The columns of A are positively oriented (so Det[A] = xstretch ystretch).

Hit the matrix on:

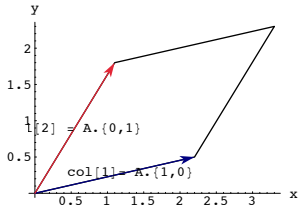
```
unitsquare =
 Show[Graphics[Line[{{0, 0}, {1, 0}, {1, 1}, {0, 1}, {0, 0}}]], Arrow[
   {1, 0}, Tail → {0, 0}, VectorColor → NavyBlue, HeadSize → 0.2],
  Arrow[{0, 1}, Tail → {0, 0}, VectorColor → AlizarinCrimson,
   HeadSize → 0.2], Axes → True, AxesLabel → {"x", "y"}];
```



to get:

The graphics here go with the sample case in the code below.

```
{a, b, c, d} = {2.2, 1.1, 0.5, 1.8};

A = {{a, b}, {c, d}};
hitplot = Show[
  Graphics[Line[{A.{0, 0}, A.{1, 0}, A.{1, 1}, A.{0, 1}, A.{0, 0}}]],
  Arrow[A.{1, 0}, Tail → {0, 0}, VectorColor → NavyBlue,
   HeadSize → 0.2], Arrow[A.{0, 1}, Tail → {0, 0},
   VectorColor → AlizarinCrimson, HeadSize → 0.2],
  Graphics[{Text["col[1]= A.{1,0}", 0.6 A.{1, 0}]}],
  Graphics[{Text["col[2] = A.{0,1}", 0.5 A.{0, 1}]}],
  Axes → True, AxesLabel → {"x", "y"}];
```



In this sample, the smaller angle running from col[1] to col[2] is counterclockwise signalling columns of A are in positive orientation. So Det[A] = xstretch ystretch.

And because the area of the square measures out to 1, the area of the parallelogram measures out to

area = Det[A].

Now measure the area a different way.

Look again at A = $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$:

The left column of A is

```
Clear[col];
col[1] = {a, c}
```

```
{2.2, 0.5}
```

The right column of A is:

```
col[2] = {b, d}
```

```
{1.1, 1.8}
```

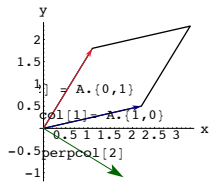
Make a vector perpendicular to col[2]

```
Clear[perpcol];
perpcol[2] = {d, -b}
```

```
{1.8, -1.1}
```

Throw it into the plot:

```
Show[hitplot,
 Arrow[perpcol[2], Tail -> {0, 0}, VectorColor -> GosiaGreen],
    Graphics[Text["perpcol[2]", 0.5 perpcol[2]]]];
```



The area of the parallelogram measures out to

area = ||col[1]|| ||col[2]|| **Sin**[short angle from col[1] to col[2]] .

So

Det[A] = area = ||col[1]|| ||col[2]|| **Sin**[short angle from col[1] to col[2]] . .

This is the same as

Det[A] = ||col[1]|| ||col[2]|| **Cos**[short angle from col[1] to perpcol[2]] .

And because

$\|col[2]\| = \sqrt{b^2 + d^2} = \sqrt{d^2 + (-b)^2} = \|perpcol[2]\|$,

Det[A] = ||col[1]|| ||perpcol[2]|| **Cos**[short angle from col[1] to perpcol[2]] .

And this is the same as the dot product

Det[A] = col[1].perpcol[2]

= {a,c}.{d,-b} = a d - b c.

Reason: col[1] = {a,c} and perpcol[2] = {d,-b}.

So in the case that the columns of A are positively oriented, the formula

Det[$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$] = a d - b c

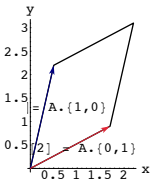
is explained.

Case 2: The columns of A are negatively oriented (so Det[A] = -xstretch ystretch).

Now see what happens when you hit a matrix whose columns are negatively oriented on the same unit square.

```
{a, b, c, d} = {0.5, 1.7, 2.2, 0.9};
```

```
A = {{a, b}, {c, d}};
hitplot = Show[
  Graphics[Line[{A.{0, 0}, A.{1, 0}, A.{1, 1}, A.{0, 1}, A.{0, 0}}]],
  Arrow[A.{1, 0}, Tail → {0, 0}, VectorColor → NavyBlue,
   HeadSize → 0.2],  Arrow[A.{0, 1}, Tail → {0, 0},
   VectorColor → AlizarinCrimson, HeadSize → 0.2],
  Graphics[{Text["col[1]= A.{1,0}", 0.6 A.{1, 0}]}],
  Graphics[{Text["col[2] = A.{0,1}", 0.5 A.{0, 1}]}],
  Axes → True, AxesLabel → {"x", "y"}];
```



The smaller angle running from col[1] to col[2] is clockwise, signalling that the columns of A are negatively oriented.

So the area of the parallelogram measures out to

area = —Det[A].

Now measure the area a different way.

Look again at A = $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$:

The left column of A is

```
Clear[col];
col[1] = {a, c}
```

```
{0.5, 2.2}
```

The right column of A is:

```
col[2] = {b, d}
```

```
{1.7, 0.9}
```

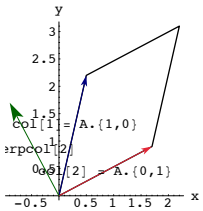
Make a vector perpendicular to col[2]

```
Clear[perpcol];
perpcol[2] = {-d, b}
```

```
{-0.9, 1.7}
```

Throw it into the plot:

```
Show[hitplot,
 Arrow[perpcol[2], Tail -> {0, 0}, VectorColor -> GosiaGreen],
    Graphics[Text["perpcol[2]", 0.5 perpcol[2]]]];
```



The area of the parallelogram measures out to

area = ||col[1]|| ||col[2]|| **Sin**[short angle from col[1] to col[2]] .

So

-Det[A] = area = ||col[1]|| ||col[2]|| **Sin**[short angle from col[1] to col[2]] . .

This is the same as

-Det[A] = ||col[1]|| ||col[2]|| **Cos**[short angle from col[1] to perpcol[2]] .

And because

$$\|col[2]\| = \sqrt{b^2 + d^2} = \sqrt{(-d)^2 + b^2} = \|perpcol[2]\|,$$

-Det[A] = ||col[1]|| ||perpcol[2]|| **Cos**[short angle from col[1] to perpcol[2]] .

And this is the same as the dot product

- Det[A] = col[1].perpcol[2]

= {a,c}.{-d,b} = -a d + b c.

Reason: col[1] = {a,c} and perpcol[2] = {-d,b}.

Multiply through by - 1 to get

Det[A] = a d - b c

Reason: col[1] = {a,c} and perpcol[2] = {-d,b}.

So in the case that the columns of A are negatively oriented, the formula

Det[$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$] = a d - b c

is explained.

Case3: The columns of A are neither positively or negatively oriented.

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

In this case the columns point in the same or opposite direction.

In this case the parallelogram collapses to a straight line. So the area of the parallelogram measures out to

0 .

Because area= |Det[A]|, this tells you, Det[A] = 0.

Also in this case col[1] = u col[2]

So unless col[2] = {0,0}, you can be sure col[1] = u col[2] a number u.

This gives {a,c} = {u b, u d} and

a d - b c = u b d - u d b = 0.

And if col[2] = {0,0}, then a d - b c = a 0 - b 0 = 0.

The upshot: In this case

Det[A] = a d - b c as well.

Now you are at the point where you know why

$$\mathbf{Det}\left[\begin{pmatrix} a & b \\ c & d \end{pmatrix}\right] = a\,d - b\,c$$