Intro Math Problem Solving
November 16

Matrix-Vector Multiplication

Linear Transforms

Linear Transforms in 2D

Transform Catalog

Snakes and Ladders

Homework #11

Reference

Chapter 4 of Cleve Moler's Experiments with Matlab discusses matrices and linear transforms.

The chapters of "Experiments with MATLAB" are available online at:

https://www.mathworks.com/moler/exm/chapters.html

The game of Snakes and Ladders is discussed by Nick Berry in an article "Analysis of Chutes and Ladders".

The article is available on his website:

http://datagenetics.com/blog/november12011/index.html

.

Matrix-Vector Multiplication

$$\begin{bmatrix} 1 & 0 & 2 & 0 \\ 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 0 \\ 6 & 0 & 0 & 7 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 5 \\ 1 \\ 8 \end{bmatrix} = \begin{bmatrix} 4 \\ 47 \\ 5 \\ 68 \end{bmatrix}$$

CA Transition Method #1

Let's look at the California population problem again. If we had the California and US populations from last year, we estimated the populations next year by this procedure:

```
ca_old = ca;
us_old = us;

us = 0.90 * us_old + 0.30 * ca_old;
ca = 0.10 * us_old + 0.70 * ca_old;
```

An Example of w=A*v

The California population calculation can be viewed as an example of matrix-vector multiplication, of the form:

pop = P * pop_old

Here pop and pop_old are column vectors (2 rows, 1 column), and P is a matrix (2 rows, 2 columns) containing the transition probabilities.

Matrix-Vector Multiplication

The interpretation of "pop = P * pop_old" is:

pop(1) combines row 1 of P with the entries of pop_old:

pop(1) = P(1,1) * pop_old(1) + P(1,2) * pop_old(2)

pop(2) combines row 2 of P with the entries of pop_old:

pop(2) = P(2,1) * pop_old(1) + P(2,2) * pop_old(2)

.

Numerical Example

P = [0.90, 0.30;
0.10, 0.70];

pop_old = [200;
100]; <- Column vector

pop = P * pop_old
= [0.90 * 200 + 0.30 * 100; = [180 + 30; = [210;
0.10 * 200 + 0.70 * 100] 20 + 70]; 90];

MxN * Nx1 Example

For w=A*v, we often have A an NxN or “square” matrix. However, if A is an MxN matrix and v is an Nx1 column vector, then the main difference is that the result w is now an Mx1 column vector:

A = [1, 2, 3; v = [10;
4, 5, 6]; 20;
30];

w = A * v = [1 * 10 + 2 * 20 + 3 * 30 = [140;
4 * 10 + 5 * 20 + 6 * 30] 320];

Write the dimensions of w=A*v:

2x1 = 2*3 * 3x1

We need the inner dimensions (3 * 3) to match. The dimension of the result is the outer pair of dimensions, 2x1.

MxN * NxL example

A = [2, 6, 5; V = [1, 10;
3, 2, 0; 0, 20;
4, 1, 7]; 1, 30];

W = A * V =

[2*1 + 6*0 +5*1, 2*10 + 6*20 + 5*30 = [7, 290;
3*1 + 2*0 + 0*1, 3*10 + 2*20 + 0*30 3, 70;
4*1 + 1*0 + 7*1 4*10 + 1*20 + 7*30]; 11, 270];

Dimension check:

3x2 = 3x3 * 3x2

MATLAB Details

MATLAB easily handles all these cases:

NxN * Nx1, square matrix times column vector;

MxN * Nx1, rectangular matrix times column vector;

MxN * NxL, matrix times matrix.

Difference between * and .* multiplication:

A .* B:

A and B must have the same MxN shape.

Multiply elements pairwise, result is MxN.

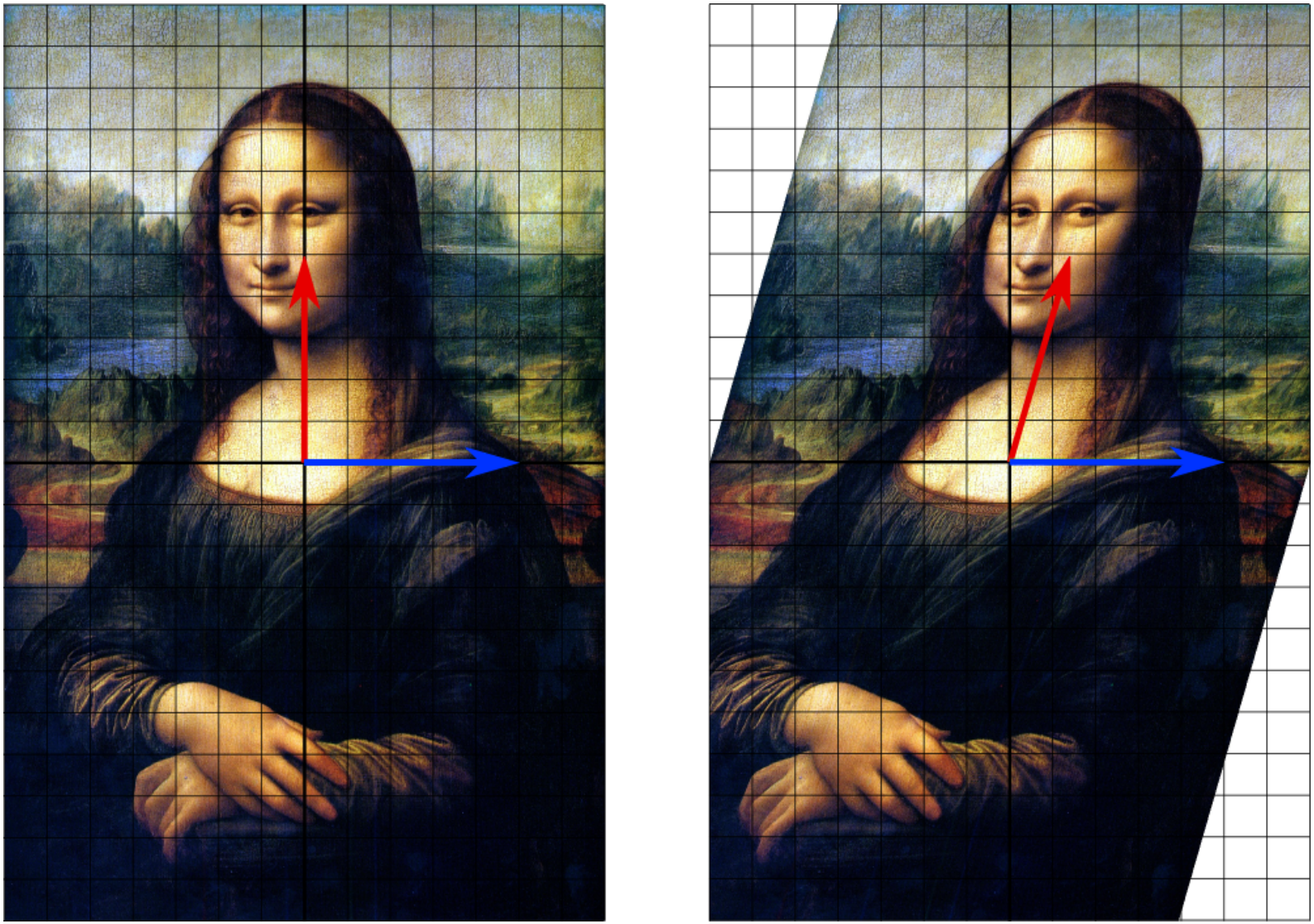
A * B:

Last dimension of A and first of B must match.

Rows of A and columns of B are multiplied and summed.

Shape of result is rows of A X columns of B.

Linear Transforms



Linear Algebra

Linear Algebra is the study of linear transforms, a topic which is the foundation for mathematical studies.

As math majors, you are likely to take a linear algebra course in sophomore year.

The algebra of matrices and vectors is the primary example of linear transforms, and so it's worth taking a brief, abstract look at this topic.

Linear Space

A linear space is a collection V of objects (often called "vectors") with typical name v, which can be added together, or scaled. In particular, for every scalar s, s1, and s2, and for every vector v, v1 and v2, it must be true that:

1) 0 + v = v; where 0 is in V
2) 1*v = v;
3) v1 + v2 = v2 + v1;
4) (v1+v2)+v3 = v1+(v2+v3);
5) There is an element v2 such that v1+v2 = 0
6) s1*(s2*v) = (s1*s2)*v;
7) s*(v1+v2) = s*v1 + s*v2;
8) (s1+s2) * v = s1*v + s2*v;

Linear transforms

A transform is a rule T which takes objects v from a linear space V and associates them with objects w in a linear space W. If v is one of the objects, then the transformed object can be written as T(v), and we sometimes write T: v -> T(v).

T is a linear transform if

a) T(sv) = s*T(v), for all scalars s, and objects v;

b) T(v1+v2) = T(v1) + T(v2), for all objects v1 and v2.

If A is an NxN matrix, and V is the space of column vectors of length N, then A: v -> A*v is a linear transform from V to V.

If A is an MxN matrix, then A: v-> A*v is a linear transform from V to W, where V and W are the spaces of column vectors of lengths N and M, respectively.

Inverse? Singular?

Suppose T is a linear transform, and S is another linear transform with the property that, if:

if $w = T(v)$,

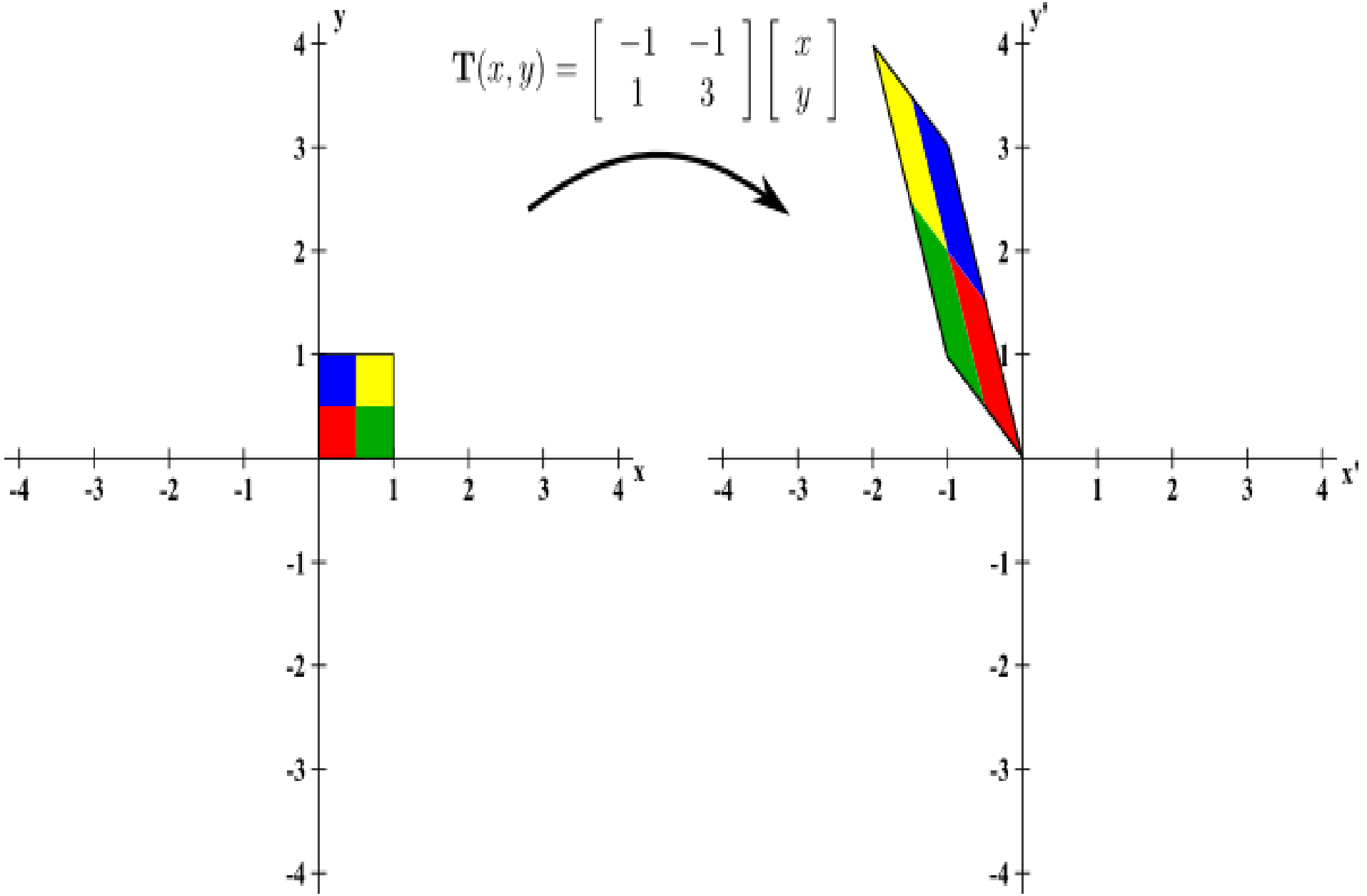
then $S(w) = v$.

We say that S is the inverse transform for T. It allows us to "undo" or "reverse" the transform.

If there is no inverse for T, we say that T is singular or noninvertible.

A singular transform "destroys information". A transform is singular if there are at least two values v1 and v2 which T transforms to the same value w. This makes it impossible for an inverse transform to exist, since if we gave it the value w, it cannot decide whether to return v1 or v2.

Linear Transforms in 2D



w = A*v in 2D geometry

Matrix multiplication is a kind of linear transform.

Let our linear space be the set of column vectors V of length 2, and our transforms be 2x2 matrices. Then w=A*v transforms one vector into another.

There are a number of linear transforms whose behavior can be understood by looking at before and after pictures of the data.

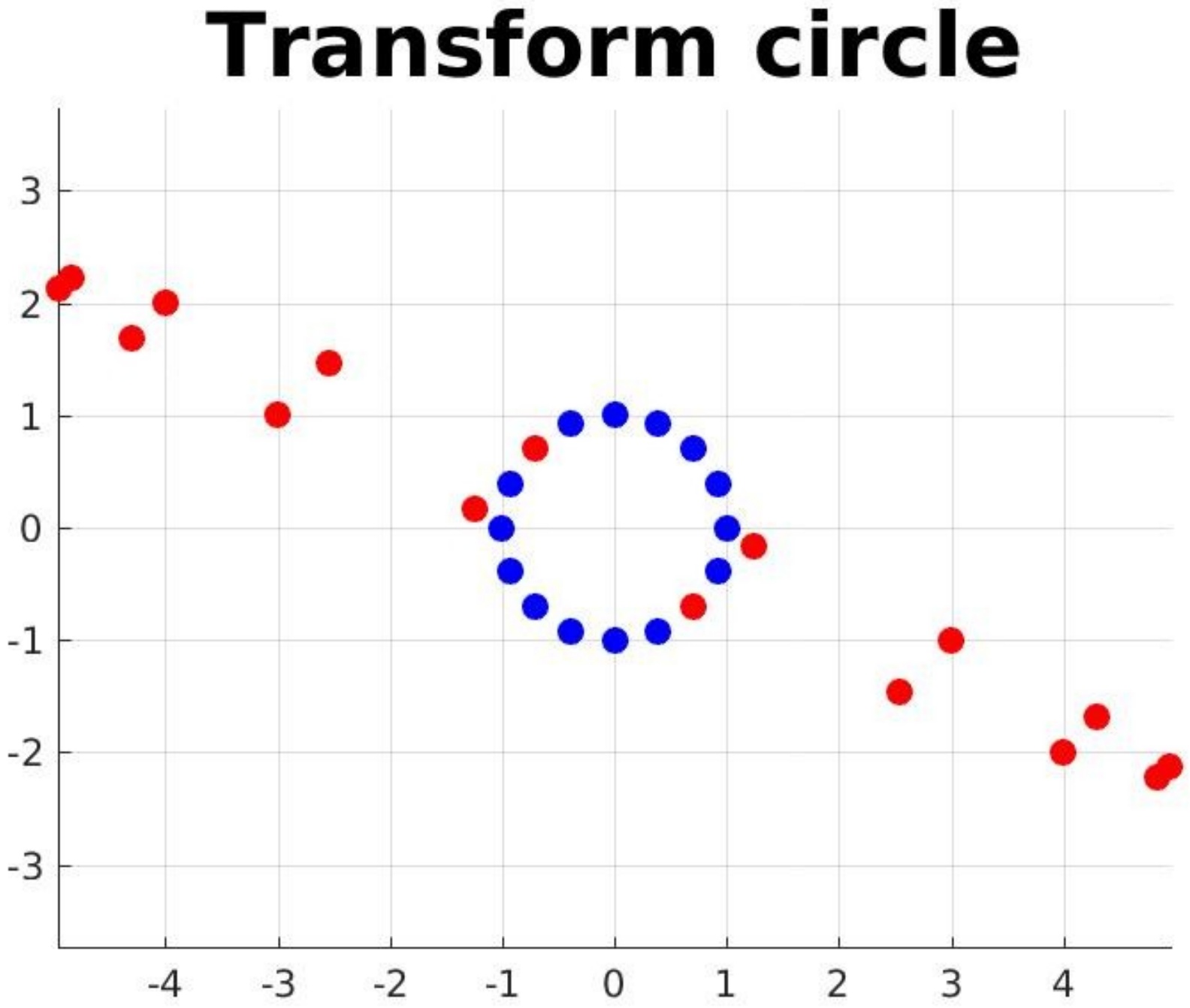
An arbitrary matrix A

To begin, let's consider the following matrix:

```
A = [ 4, -3;
      -2, 1];
```

This matrix A doesn't have any special properties, but we can still get some ideas about it by looking at its operation on data.

In particular, suppose we pick a collection of data vectors that lie on the unit circle, in other words, a typical vector v has v(1)^2 + v(2)^2 = 1. What happens to the w (result) vector?

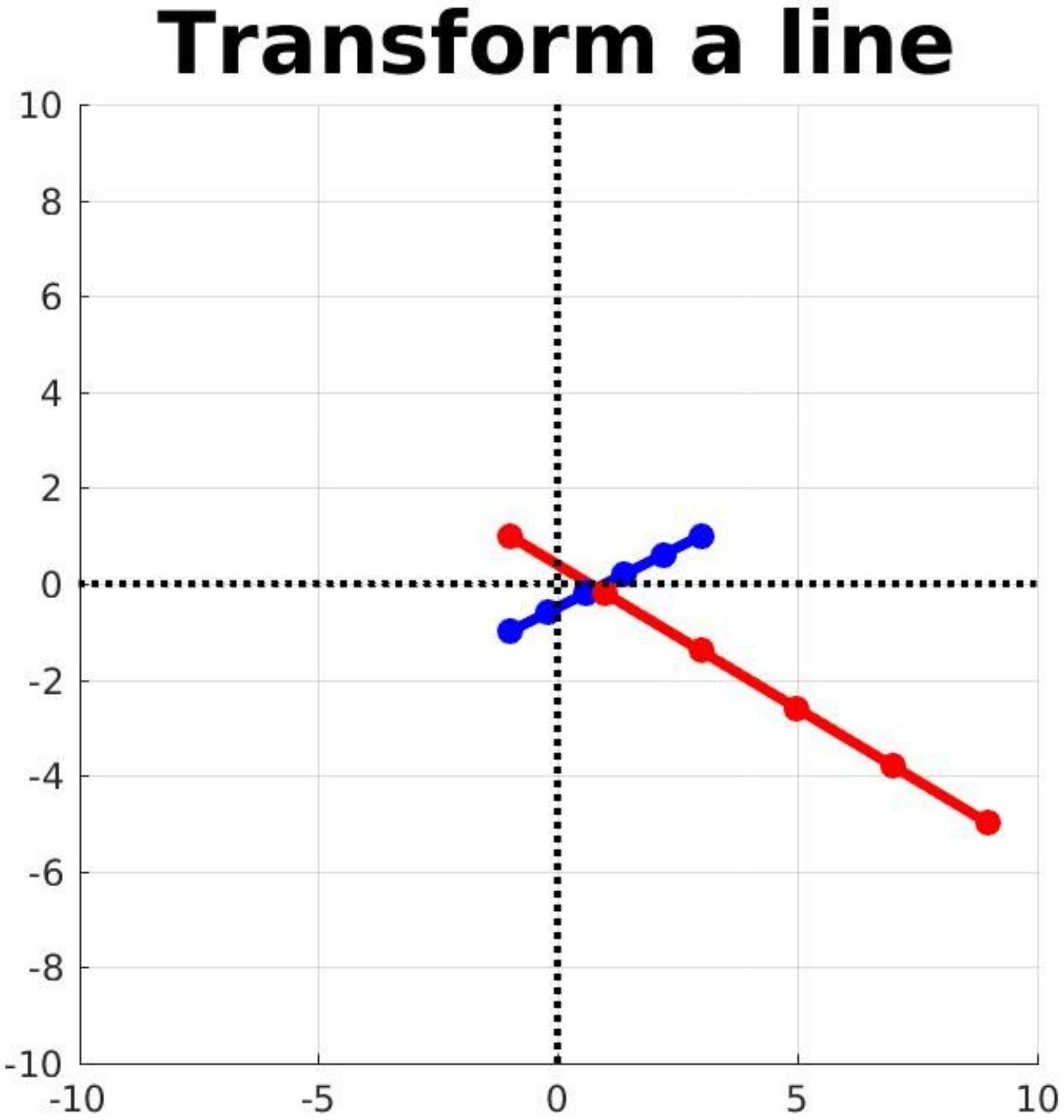

Transform circle
3
2
1
0
-1
-2
-3
-4
-3
-2
-1
0
1
2
3
4

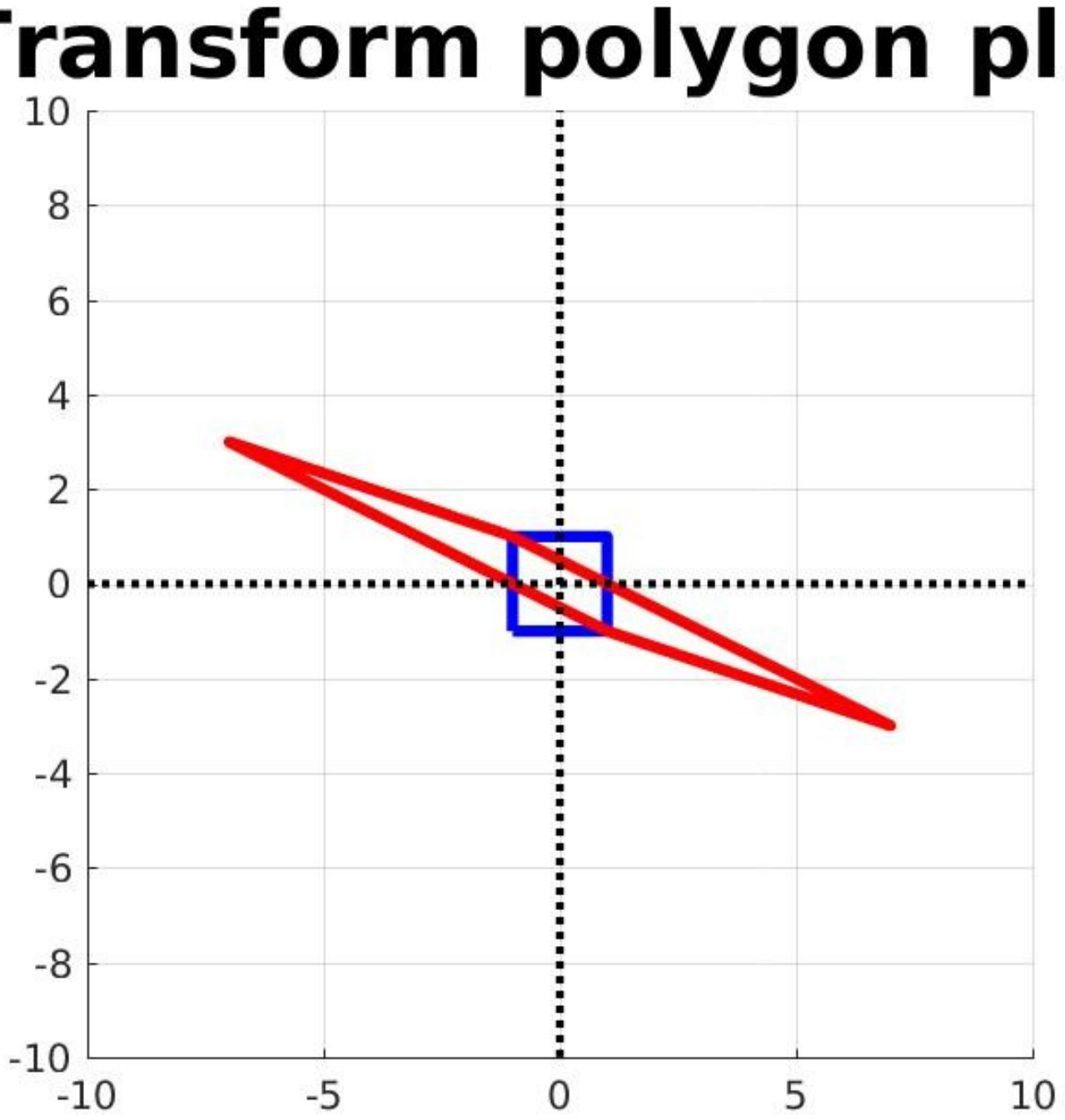
Circle -> Ellipse

The plot suggests that points on the circle are transformed into points on an ellipse.

The ellipse is stretched in one direction and thinned in another. The direction, and the amount of stretching and thinning are properties of the matrix that can be determined by linear algebra techniques.

What happens if we use A to transform points on a line instead, or points that form a polygon?


Transform a line
10
8
6
4
2
0
-2
-4
-6
-8
-10
-10
-5
0
5
10


Transform polygon plot
10
8
6
4
2
0
-2
-4
-6
-8
-10
-10
-5
0
5
10

Lines -> Lines

Both plots suggest that the matrix A maps straight lines to straight lines.

In fact, this is part of the definition of a linear transform.

If there is a linear relationship in the original data v1 and v2, then the same linear relationship will hold for the transformed data w1 and w2.

Singular Transform

We mentioned that some linear transforms have an inverse, but that singular transforms don't.

The matrix B = [1, 2; is a perfectly good matrix,

2, 4];

but it does not have an inverse. Numerically, we can predict this by finding two v's that map to the same w:

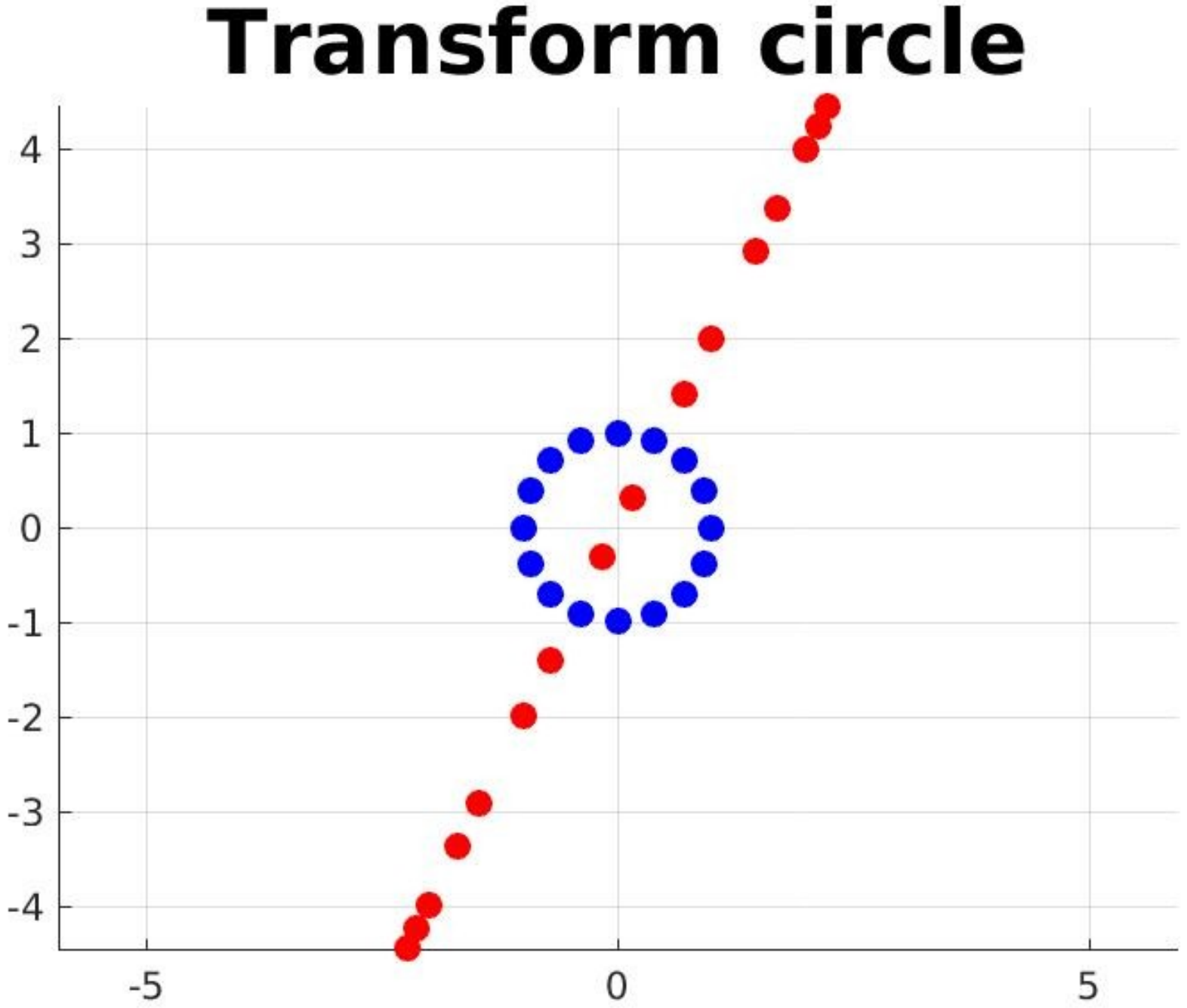
v1 = [1; v2 = [5;

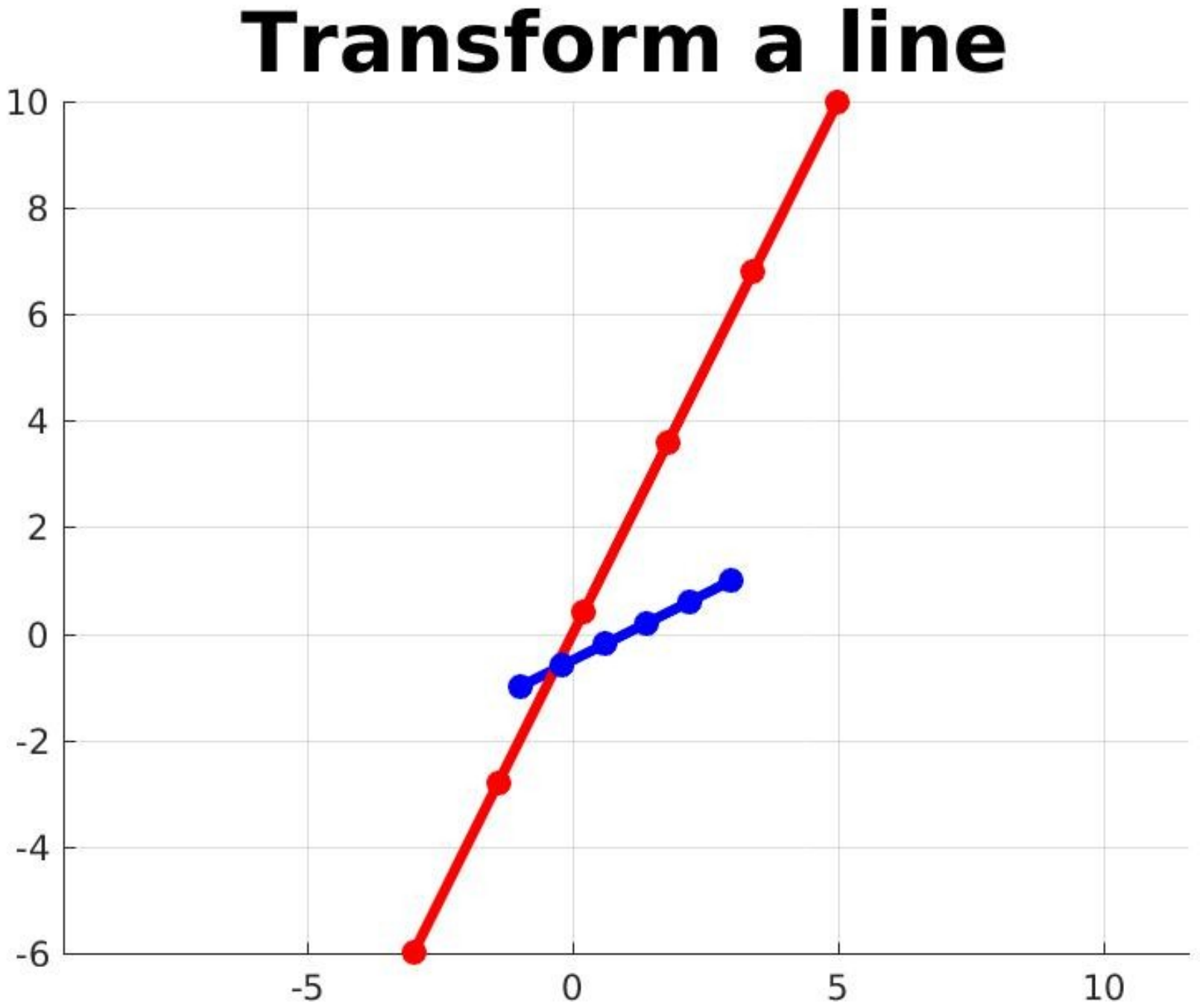
2]; 0];

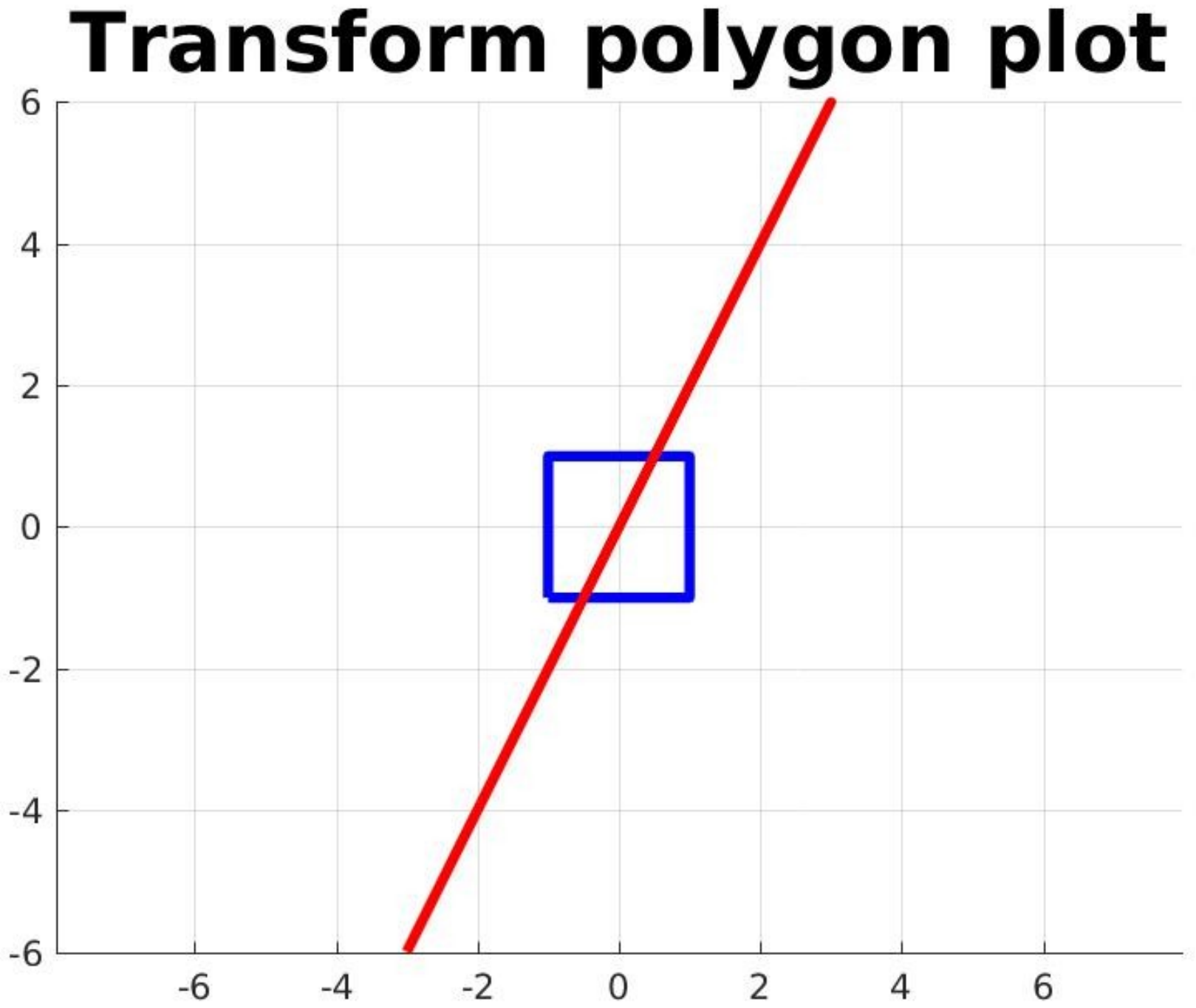
Then B*v1 = B*v2 = [5; and so B can't have an inverse.

10];

Let's look at what B does to circles, lines, and polygons:


Transform circle
4
3
2
1
0
-1
-2
-3
-4
-5
0
5


Transform a line
10
8
6
4
2
0
-2
-4
-6
-5
0
5
10


Transform polygon plot
6
4
2
0
-2
-4
-6
-6
-4
-2
0
2
4
6

Singular Matrix

For the singular matrix, all the results (circle, line, polygon) get squashed onto a single line.

The data v that “lives” in two dimensions gets flattened to a one dimensional space.

This is how a singular matrix “destroys information”, sends many objects v to the same result w, and can’t be inverted.

A “symptom” of a singular matrix is a small value for the “determinant”. If det(A) is very small, or zero, then A may be essentially singular.

The Inverse Matrix

If A = [4, -3; and v = [1; then w=A*v= [-2;
-2, 1]; 2]; 0];

If we know the result, w, and we know the transform A, can we figure out the starting point v?

If we can do this, the transform is said to be invertible or nonsingular.

If the transform w=A*v can be inverted, then there is actually a corresponding matrix, called the inverse of A, written inv(A), which starts with w and returns v, that is, v=inv(A)*w.

Inverse Example

If A = [4, -3; and v = [1; then w=A*v= [-2;

-2, 1]; 2]; 0];

Write w=A*v as follows:

-2 = 4 * v1 – 3 * v2

0 = -2 * v1 + 1 * v2

Equation 2 can be rewritten as:

v2 = 2 * v1

Then equation 1 can be simplified to

-2 = 4 * v1 – 3 * (2 * v1) = -2 * v1

Therefore:

v1 = 1, v2 = 2.

Inverse Transform

Given a matrix A, MATLAB returns the inverse with the function inv(A).

To store the inverse, we might write “B=inv(A);”

For A = [4, -3;
-2, -1]

inv(A) = [-1/2, -3/2;
-1, -2];

A*[1; = [-2;
2] 0]

inv(A) * [-2; = [1;
0] 2];

If B is the inverse of A, then A*B = B*A = I, where I is the identity matrix:

I = [1, 0;
0, 1];

Singular Matrix Example

A = [1, 2, 3;
4, 5, 6;
7, 8, 9];

v1 = [1;
2;
3]

A*v1 = [14;
32;
50]

v2 =[-0.8;
5.6;
1.2]

A*v2 = [14;
32;
50];

det(A) = -9.5162e-16

Transform Catalog

A Catalog of Special Transforms

0: the zero transform (singular)

I: the identity transform

R: rotation

D: dilation (squeeze or stretch)

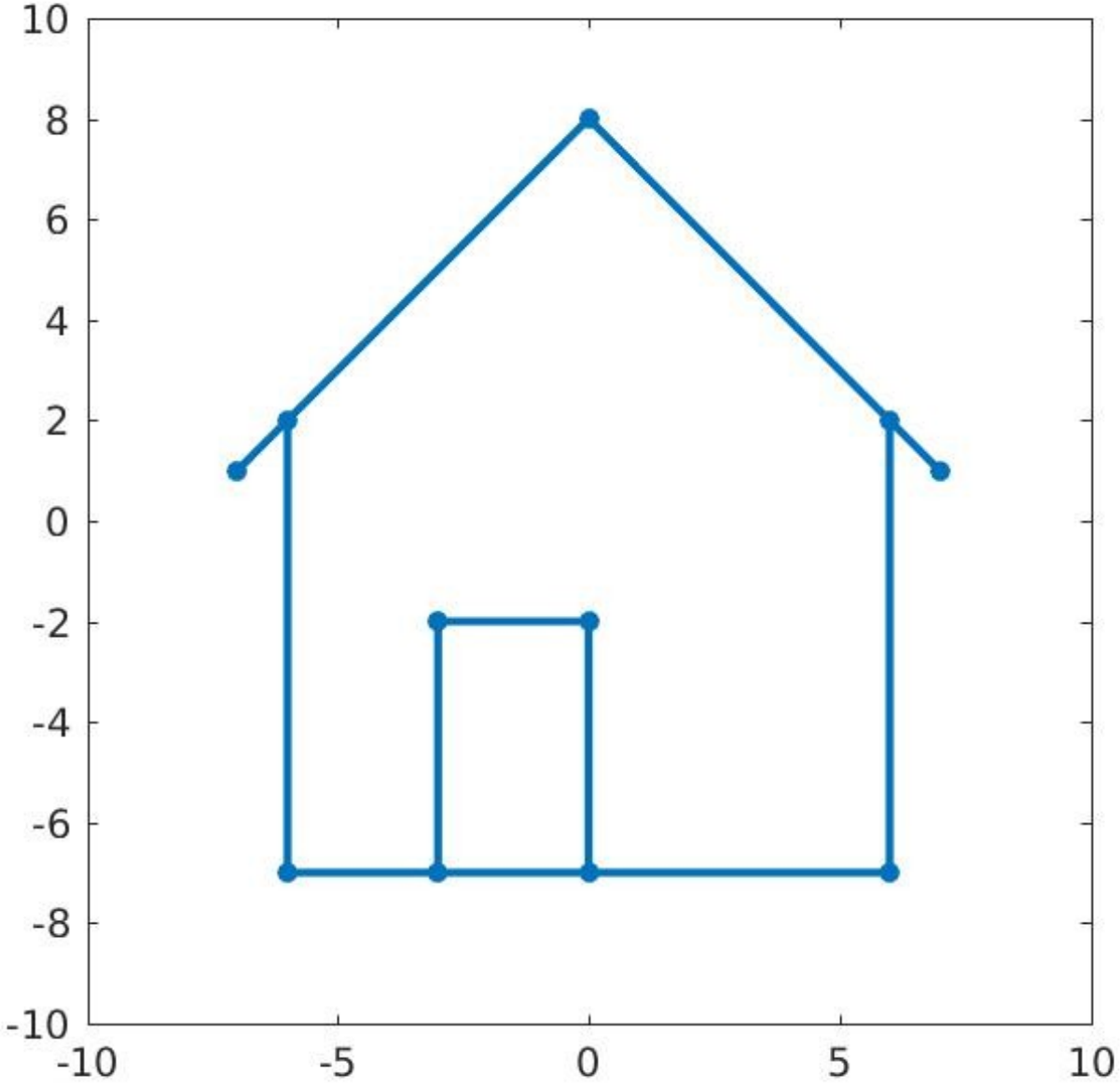
M: mirror reflection (up/down or left/right)

S: shear (vertical or horizontal)

P: projection (singular)

T: translation (not a linear transform!)

A Test Object: a House



Transformed House = A * House

Our test object is the image of a house, which is created by list of 11 points:

V = [-6, -6, -7, 0, 7, 6, 6, -3, -3, 0, 0;

-7, 2, 1, 8, 1, 2, -7, -7, -2, -2, -7];

We can apply any linear transform A to all 11 points using matrix-matrix multiplication:

W = A * V

2x11 = 2x2 * 2x11 (dimensions)

Then we can draw the transformed house W.

The 0 Transform

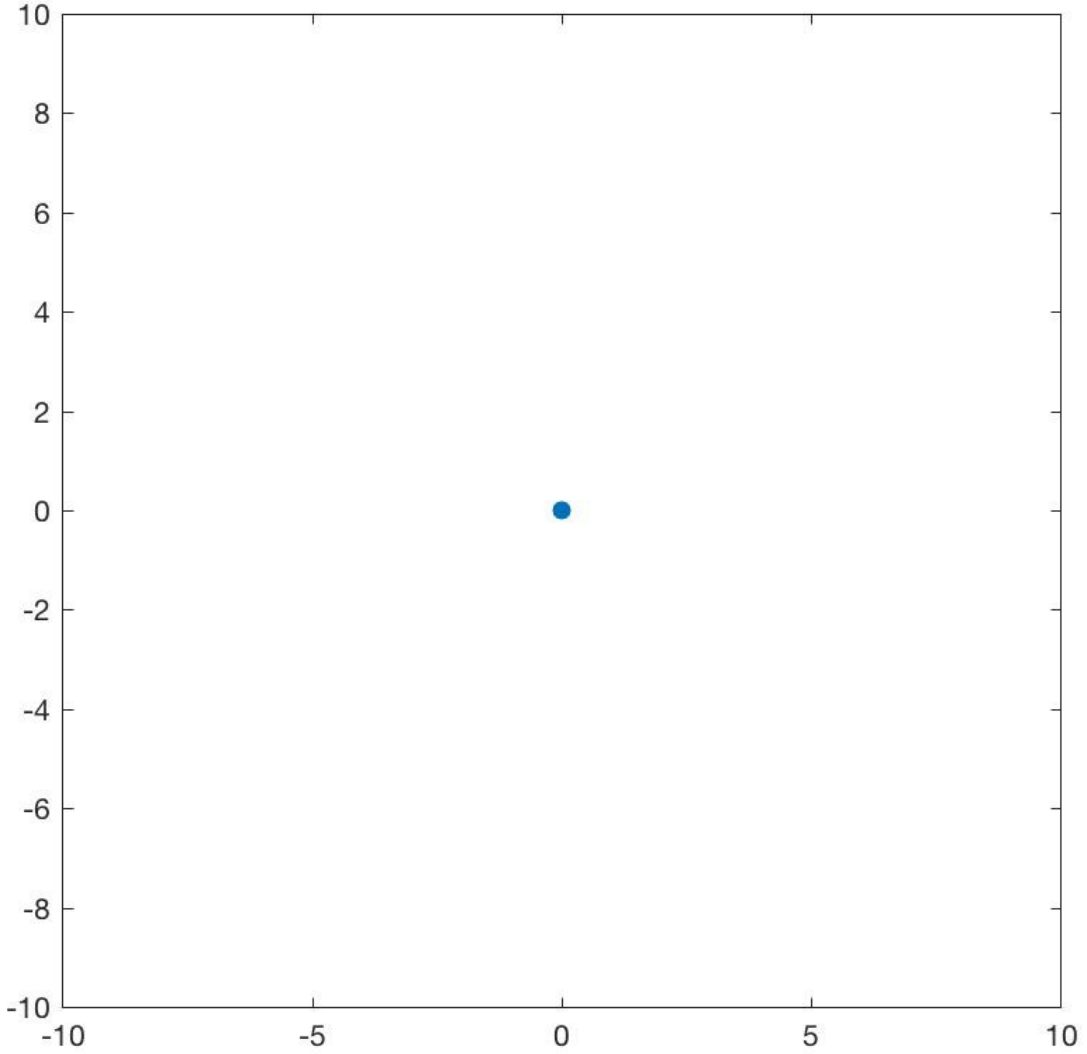
The zero transform maps every vector v to the 0 vector. In 2D, the matrix looks like this:

```
Z = [ 0, 0;
      0, 0];
```

```
Z = zeros ( 2, 2 );
```

Of course this matrix is singular!

House2 = Zero * House



The Identity Transform

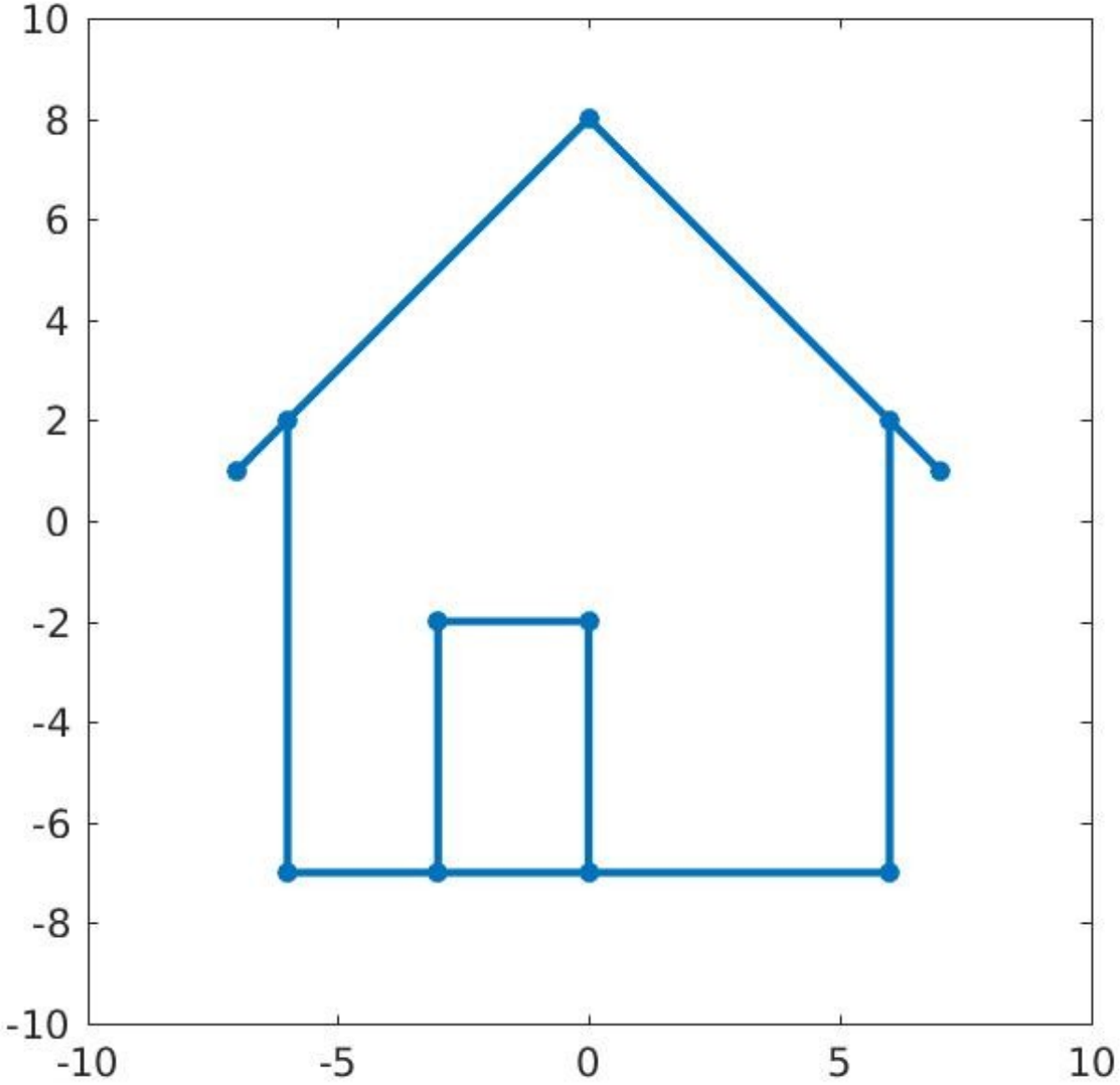
The identity transform maps every vector v to itself. In 2D, the matrix looks like this:

```
I = [ 1, 0;
      0, 1];
```

```
I = eye ( 2, 2 );
```

The inverse of the identity matrix is the identity matrix.

House2 = Identity * House



Rotation

A rotation matrix has the form:

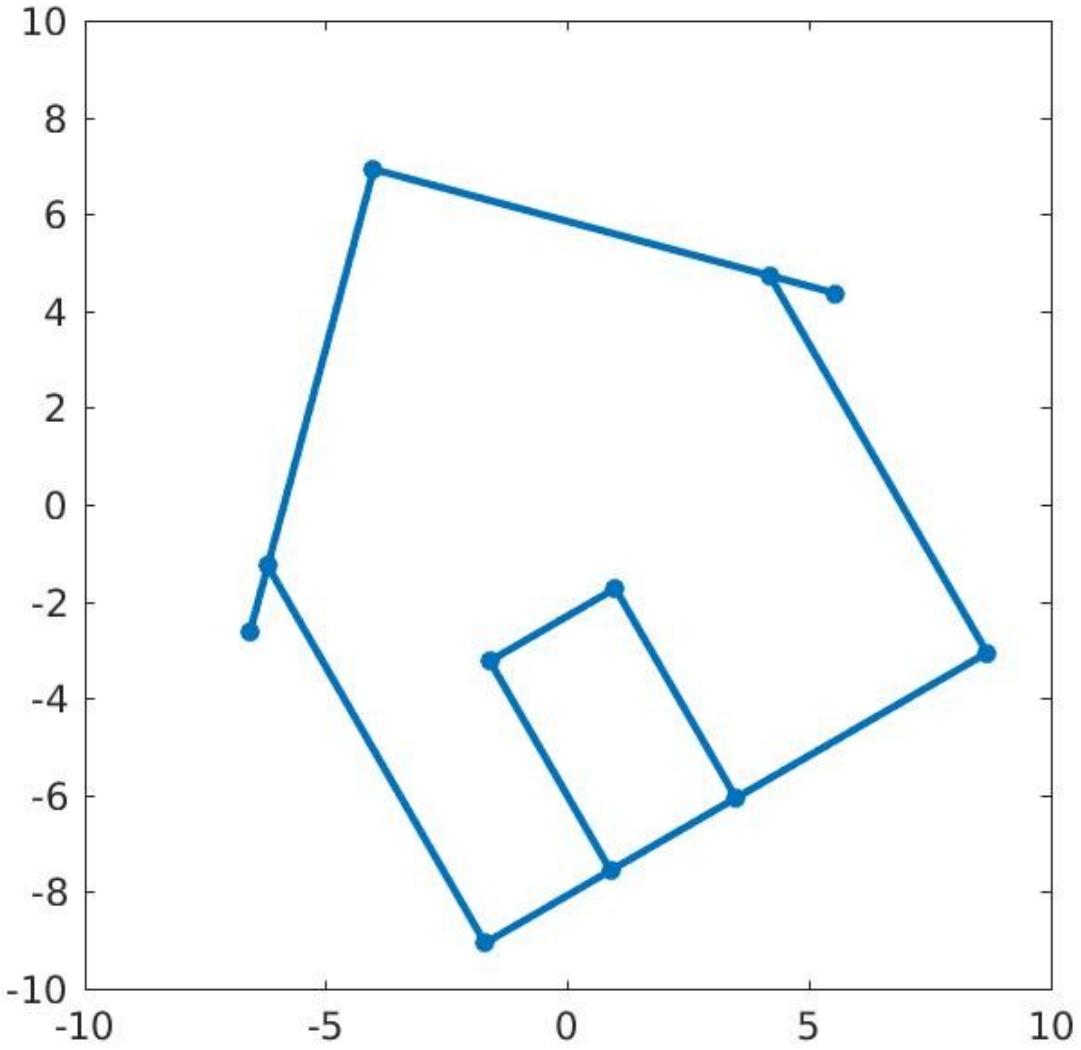
```
A = [ cos ( theta ), sin ( theta );
      -sin ( theta ), cos ( theta ) ];
```

For a rotation of 30 degrees counterclockwise:

```
A = [  0.866, -0.500;
       0.500, 0.866 ];
```

Let's apply A to the House:

House2 = Rotate30 * House1



Rotation Inverse

If a rotation matrix has the form:

A = [cos (theta), -sin (theta);
sin (theta), cos (theta)];

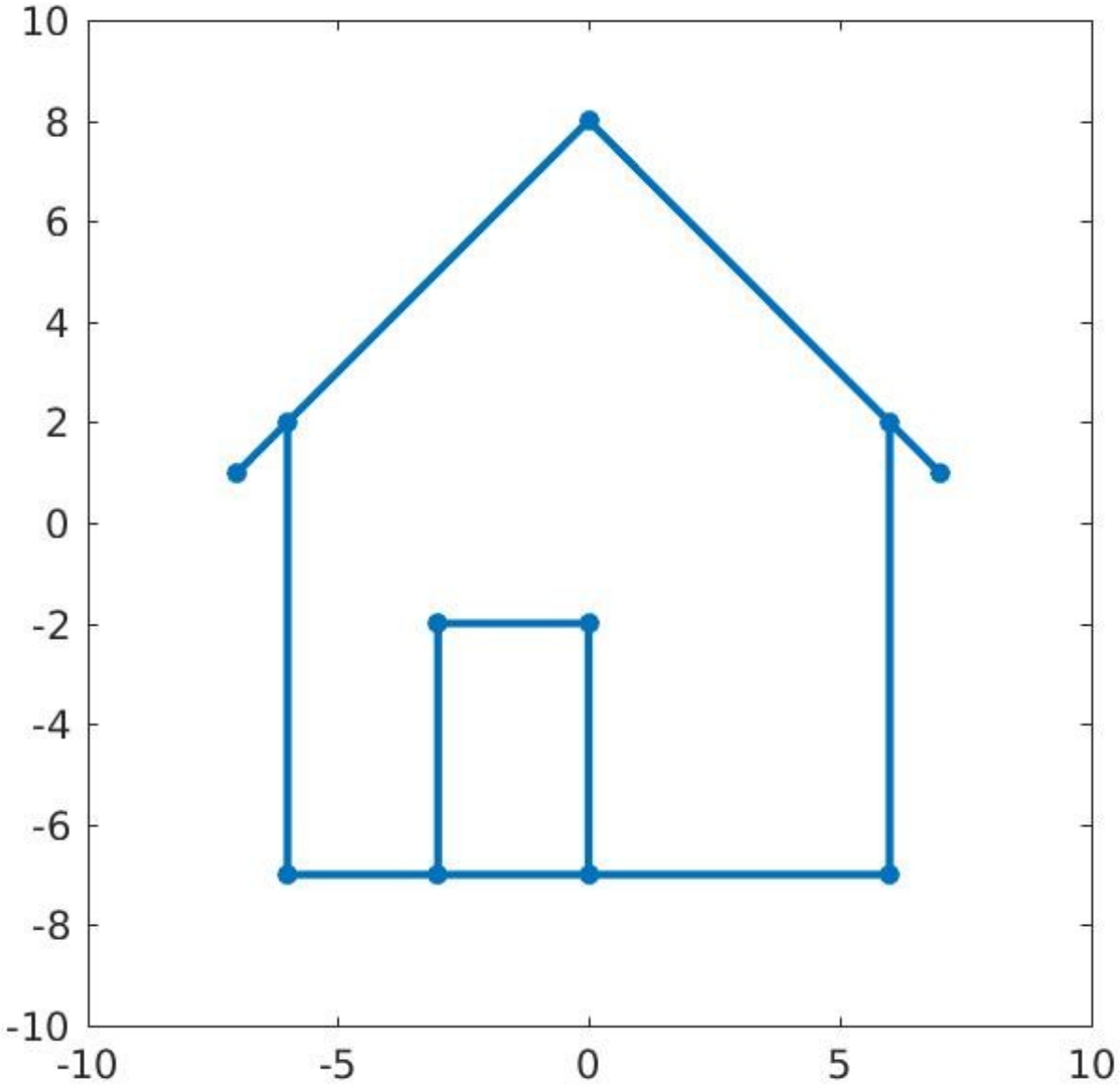
then the inverse has the form

inv(A) = [cos (theta), sin (theta);
-sin (theta), cos (theta)];

If A rotates by 30 degrees counterclockwise inv(A) is

inv(A) = [0.866, 0.500;
-0.500, 0.866];

House3 = inv(Rotate30) * House2



Dilation

A dilation matrix has the form:

A = [width factor, 0;

0, height factor];

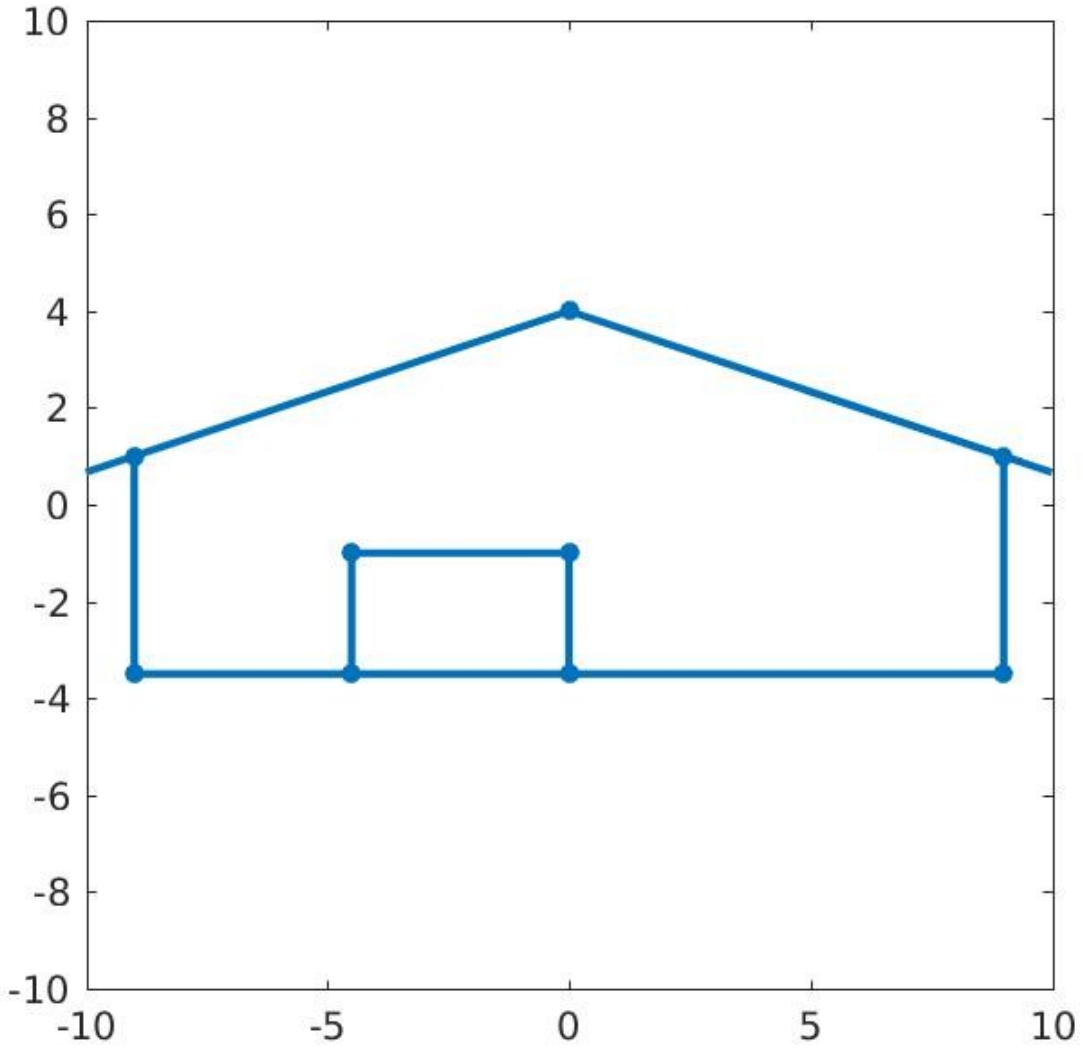
To make 1.5 wider, and half as tall:

A = [1.5, 0.0;

0.0, 0.5];

Let's apply A to the House:

House2 = Dilate(3/2,1/2) * House1



Dilation Inverse

If a dilation matrix has the form:

A = [width factor, 0;
0, height factor];

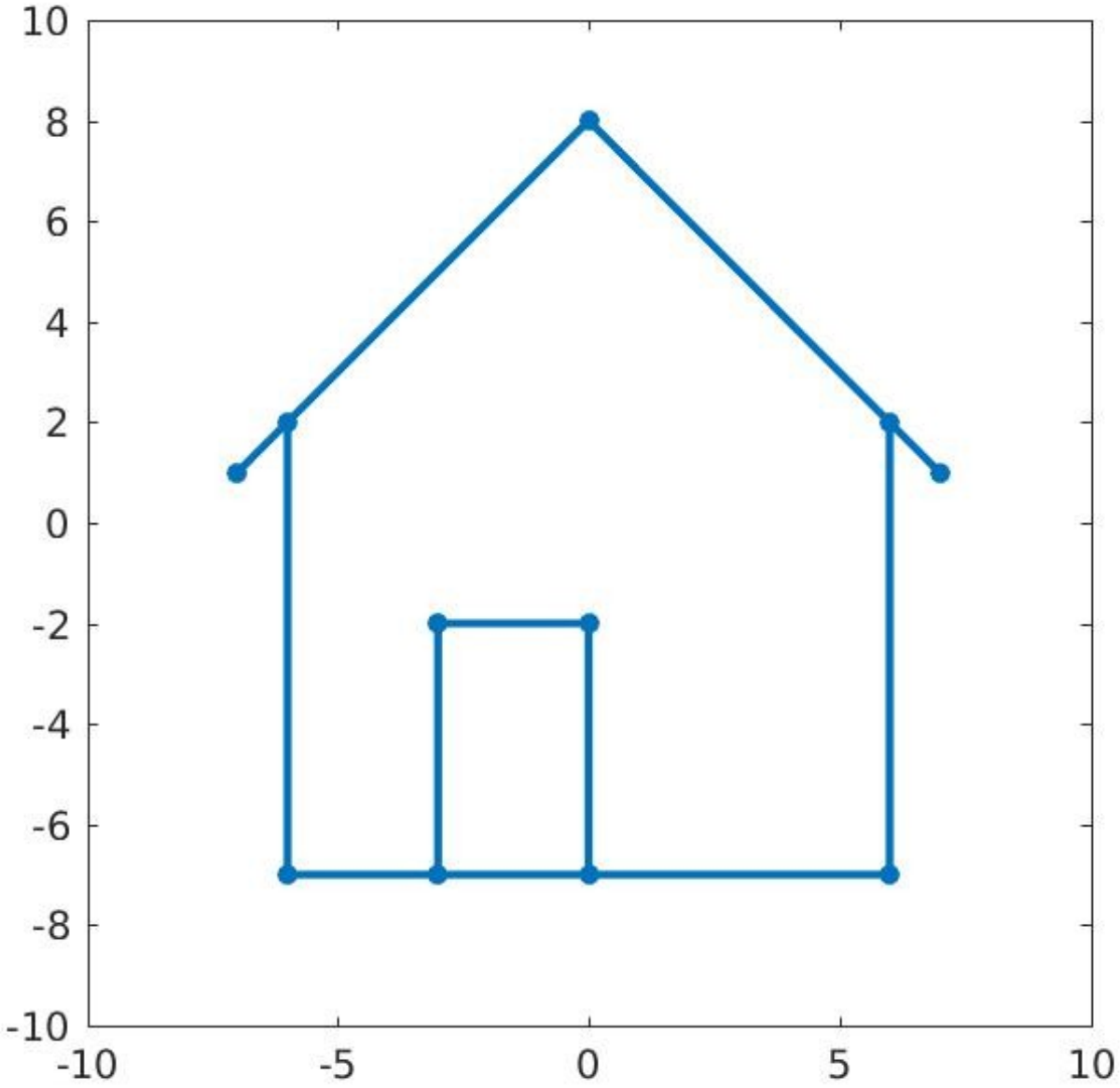
then the inverse is

inv(A) = [1/width factor, 0;
0, 1/height factor];

inv(A) = [2/3, 0.0;
0.0, 2];

If either factor was 0, A is singular, and has no inverse!

House3=inv(Dilate(3/2,1/2))*House2



Reflection

A reflection matrix has the form:

```
A = [ +1 or -1,      0;
            0,       +1 or -1 ];
```

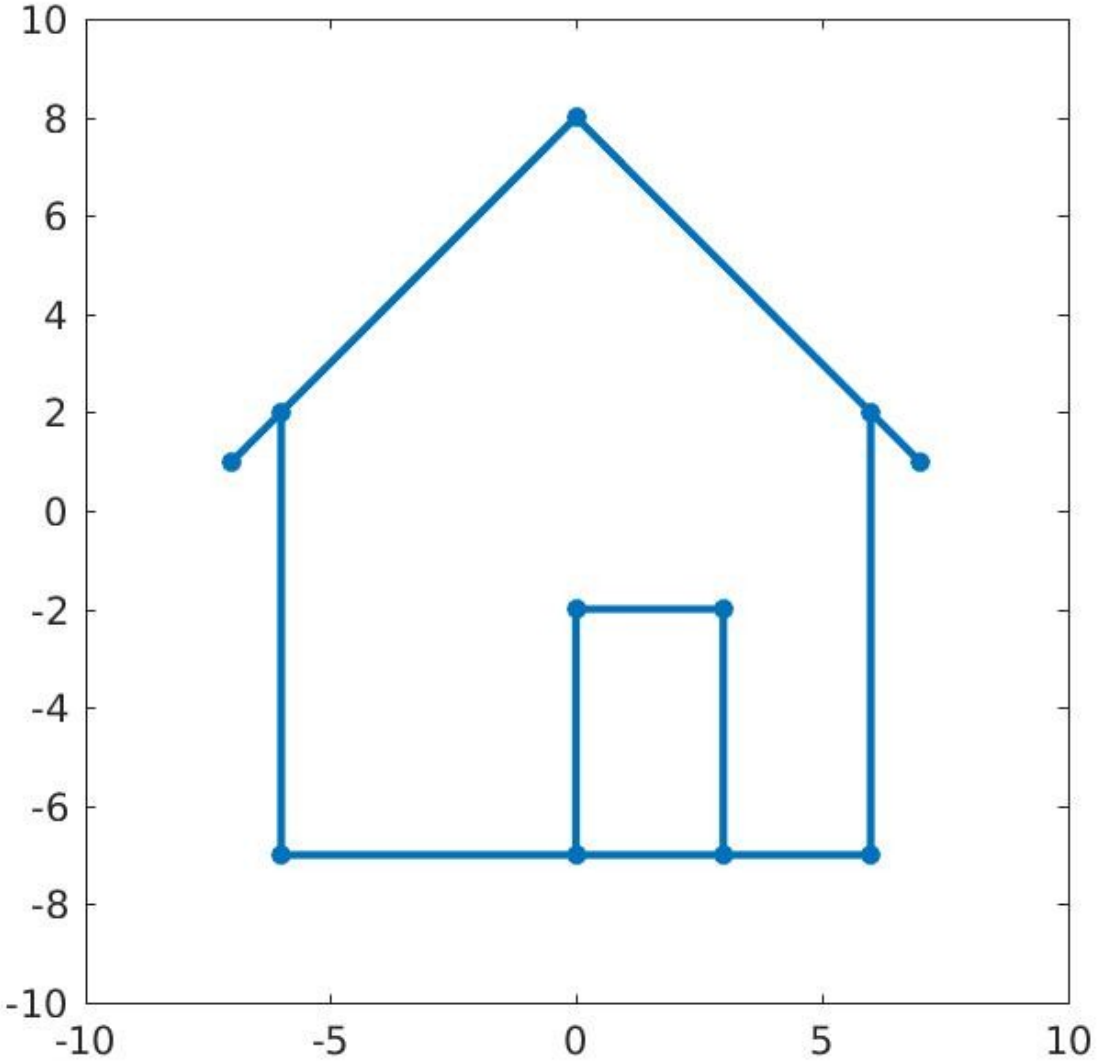
To reflect Left/Right

```
A = [  -1.0, 0.0;
          0.0, 1.0 ];
```

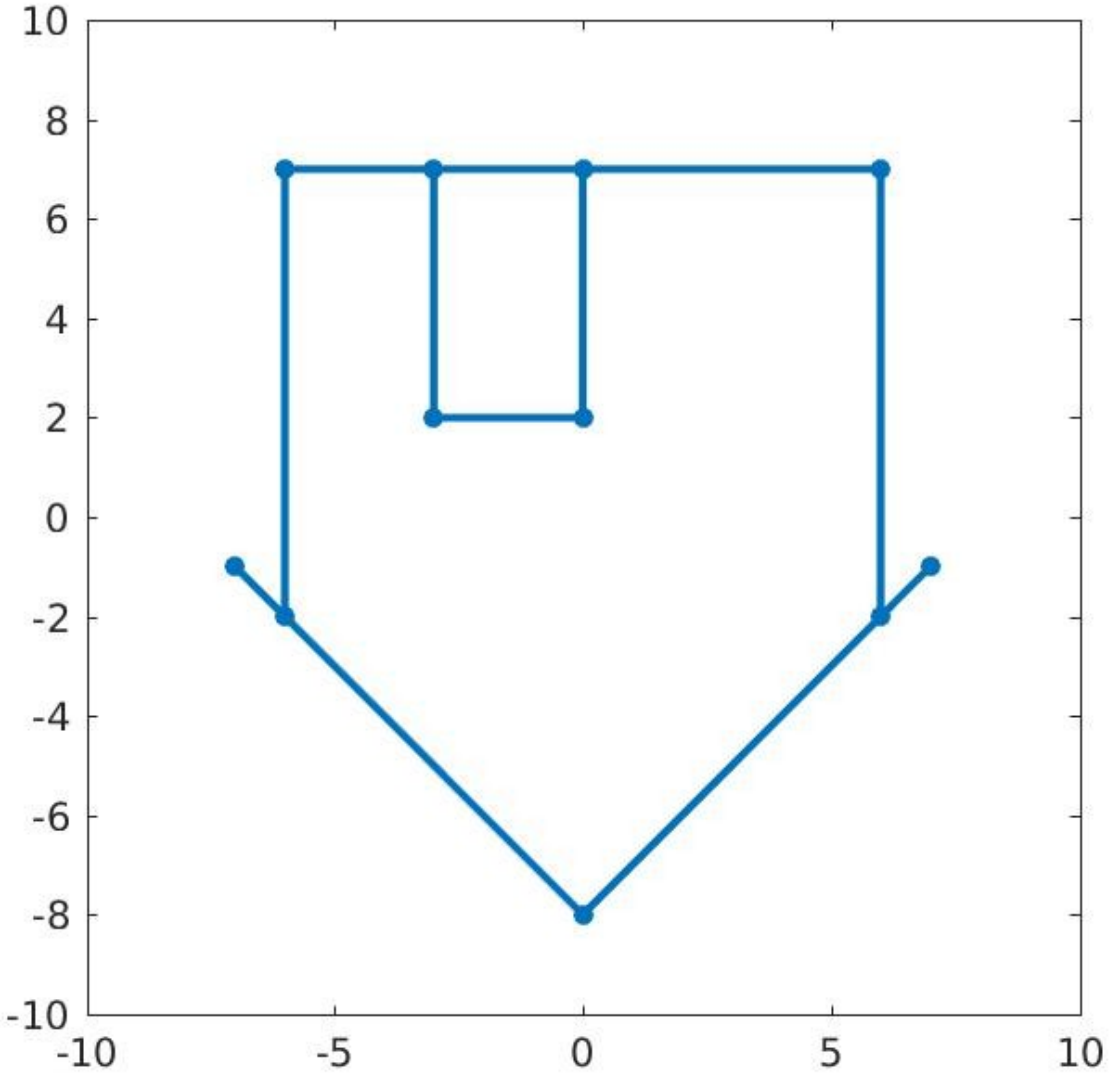
To reflect Up/Down

```
A = [   1.0, 0.0;
          0.0, -1.0 ];
```

House = ReflectLR * House



House = ReflectUD * House



Reflection Inverse

A reflection matrix is its own inverse.

To reflect Left/Right

A = [-1.0, 0.0; inv(A) = [-1.0, 0.0;
0.0, 1.0]; 0.0, 1.0];

To reflect Up/Down

A = [1.0, 0.0; inv(A) = [1.0, 0.0;
0.0, -1.0]; 0.0, -1.0];

Shear

A horizontal shear matrix:

A = [1, m;

0, 1];

A vertical shear matrix:

A = [1, 0;

m, 1];

Example:

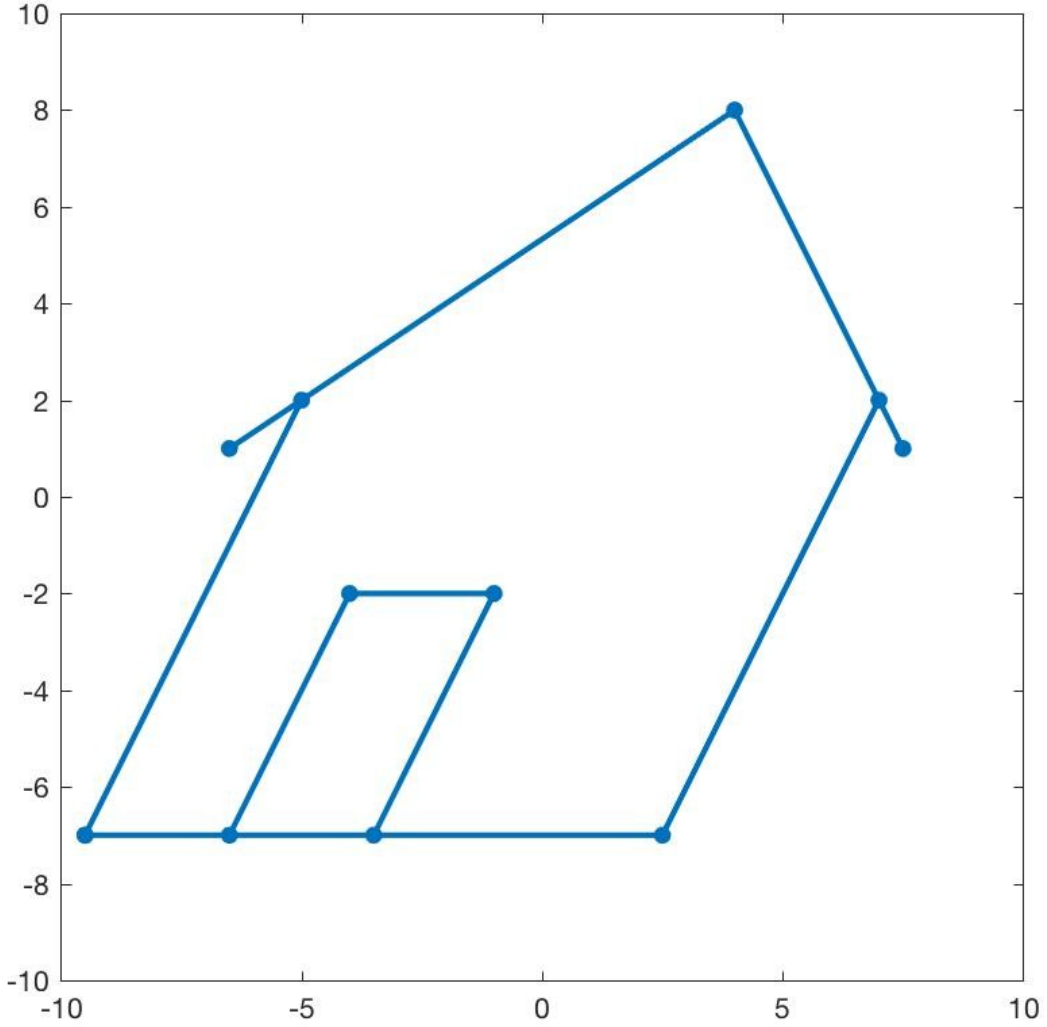
A_Hor = [1.0, 0.5;

0.0, 1.0];

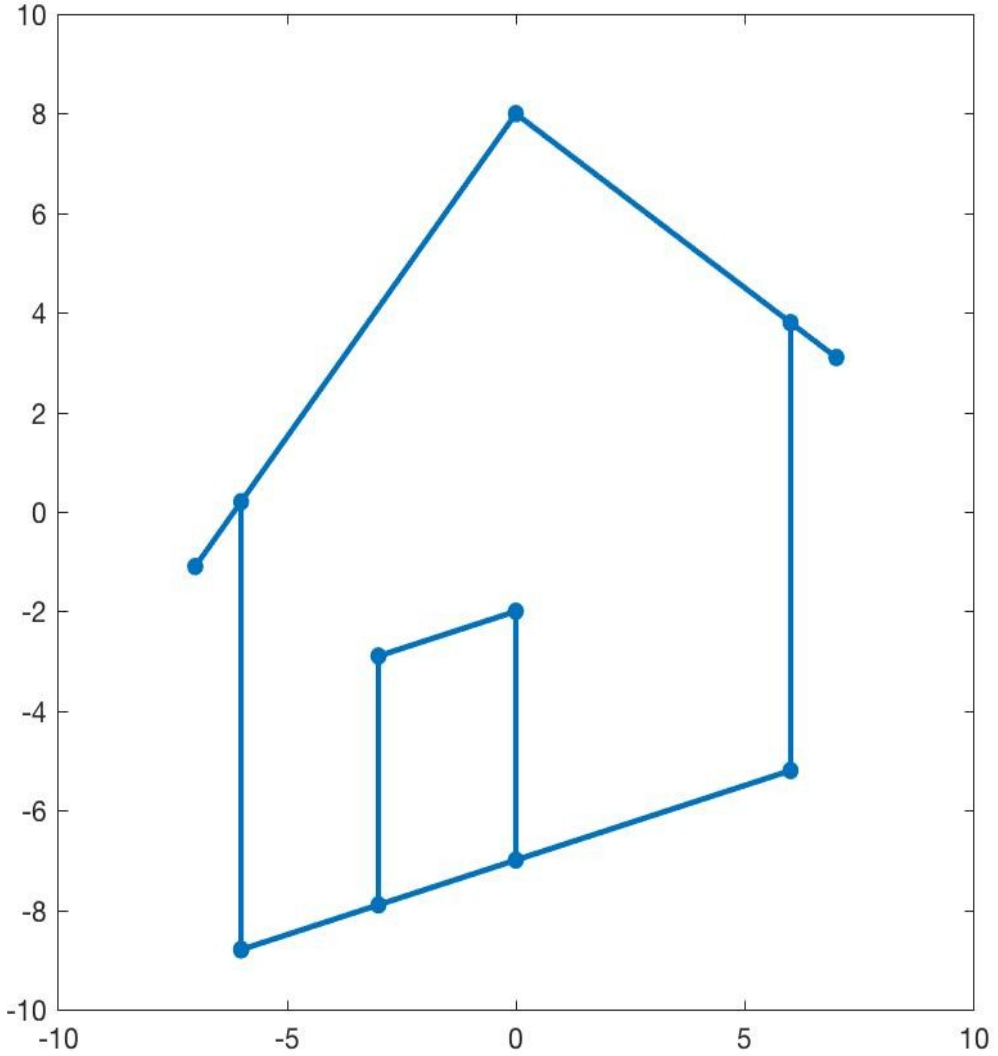
A_Vert = [1.0, 0.0;

0.3, 1.0];

House2 = ShearHOR * House1



House2 = ShearVERT * House1



Shear Inverse

Horizontal shear:

A = [1, m; inv(A) = [1, -m;

0, 1]; 0, 1];

Vertical shear;

A = [1, 0; inv(A) = [1, 0;

m, 1]; -m, 1];

Projection

A horizontal projection matrix:

A = [1,0;

0, 0];

A vertical projection matrix:

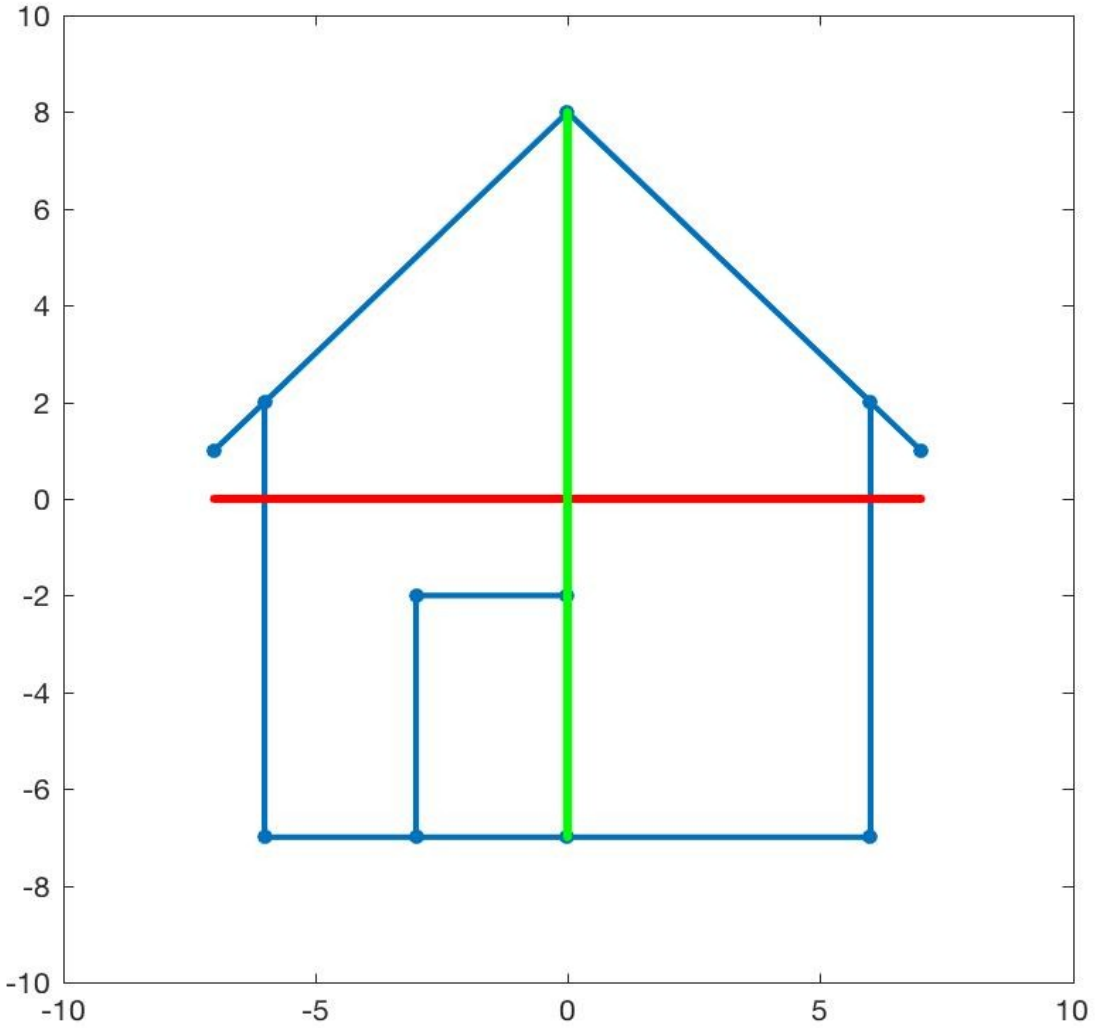
A = [0, 1;

0, 0];

Horizontal projection flattens a vector to the x direction, and vertical projection flattens it to the y direction.

A projection is singular, and has no inverse.

Projection: Horizontal (red) Vertical(green)



Translation

A translation transform has the form:

$w = T(v) = v + u;$

where u is the translation vector.

To shift everything right 2 units, and down 1 unit:

$u = \begin{bmatrix} +2; \\ -1]; \end{bmatrix}$

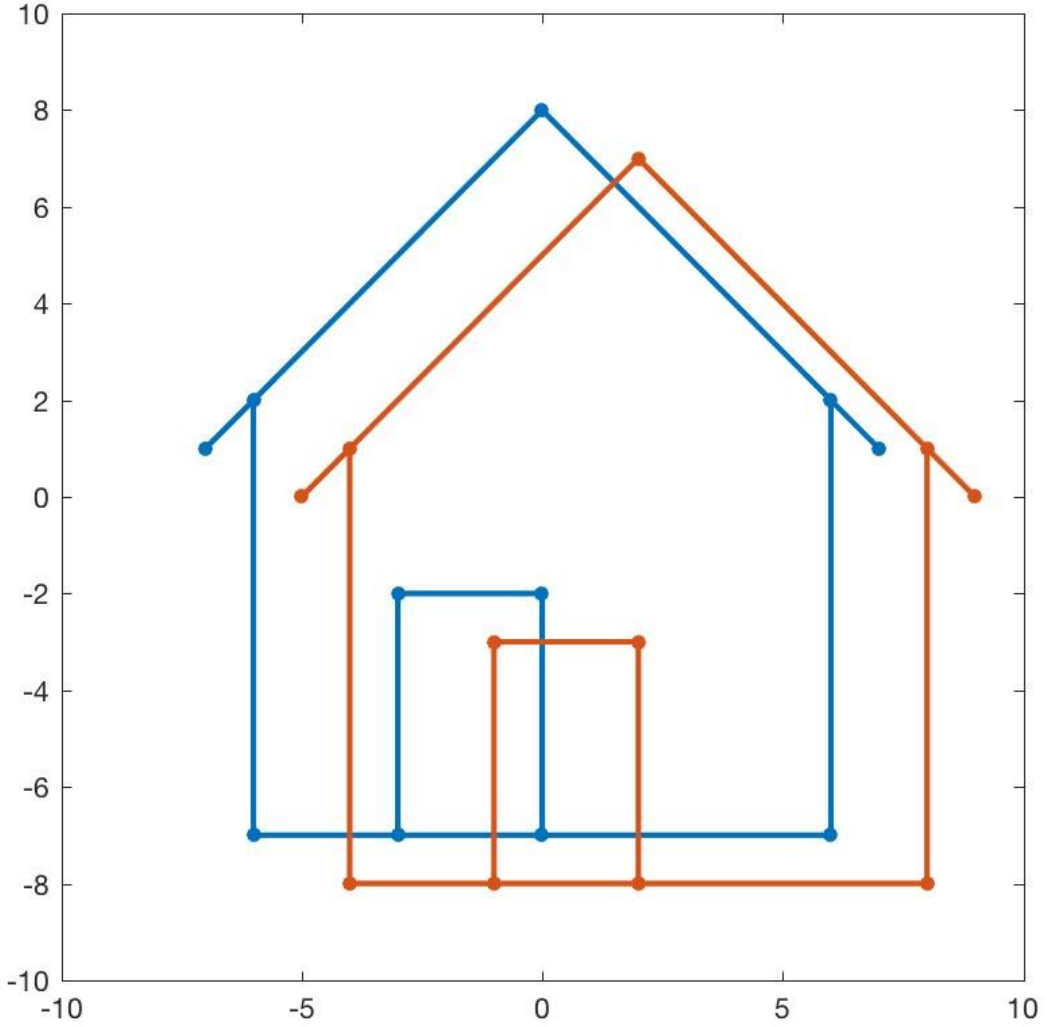
The inverse transform simply uses -u.

$v = inv(T)(w) = w - u$

Translation is a transform, but not a linear transform. We can't represent it as a matrix that multiplies the vector v.

Nonetheless, translation is a very useful transform.

House(blue), Translated House(red)



Summary

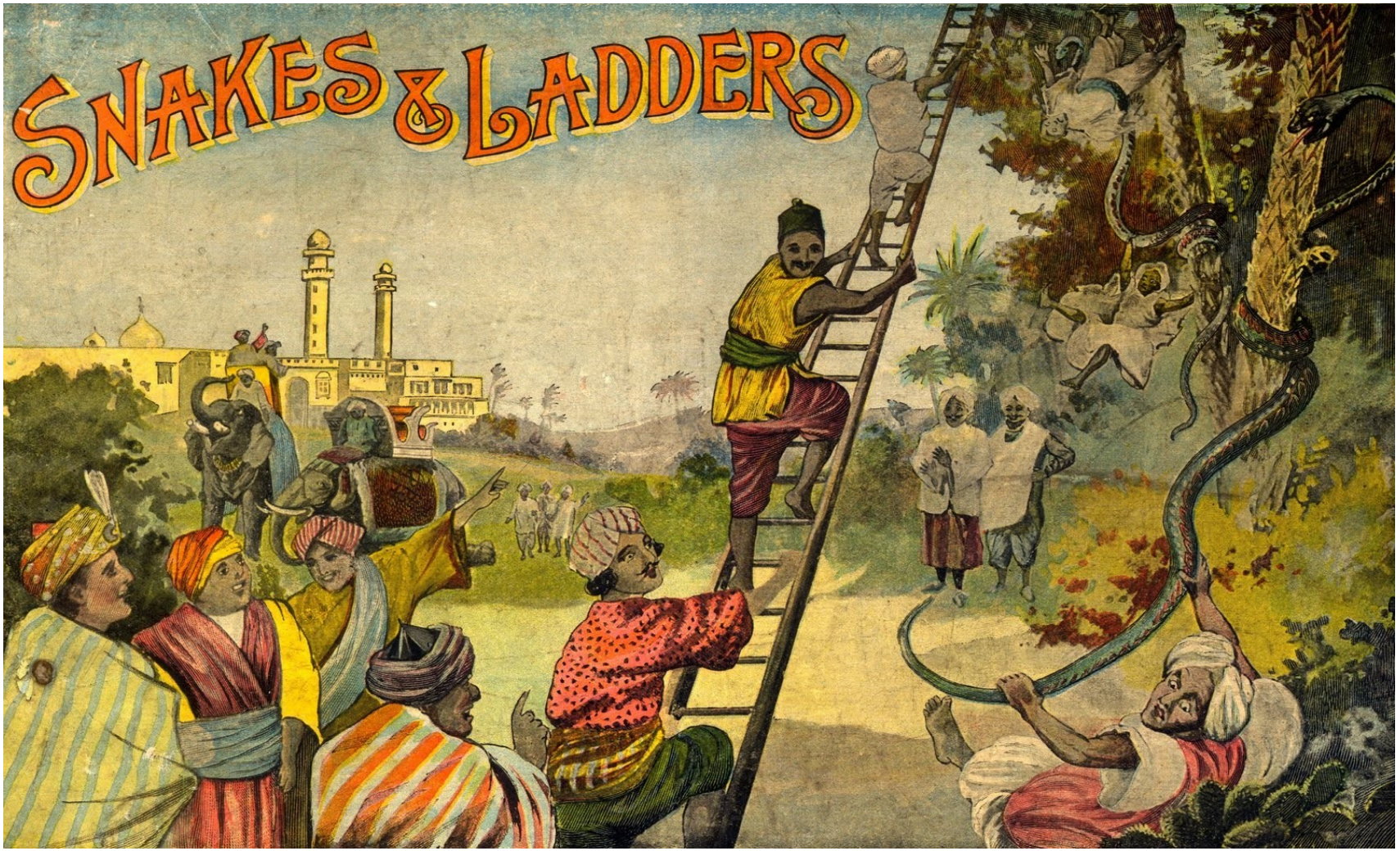
In two dimensions, a MATLAB column vector v=[x;y] is an “arrow” from the origin to (x,y).

In two dimensions, a MATLAB matrix A is a linear transform that changes v to w=A*v, using matrix multiplication.

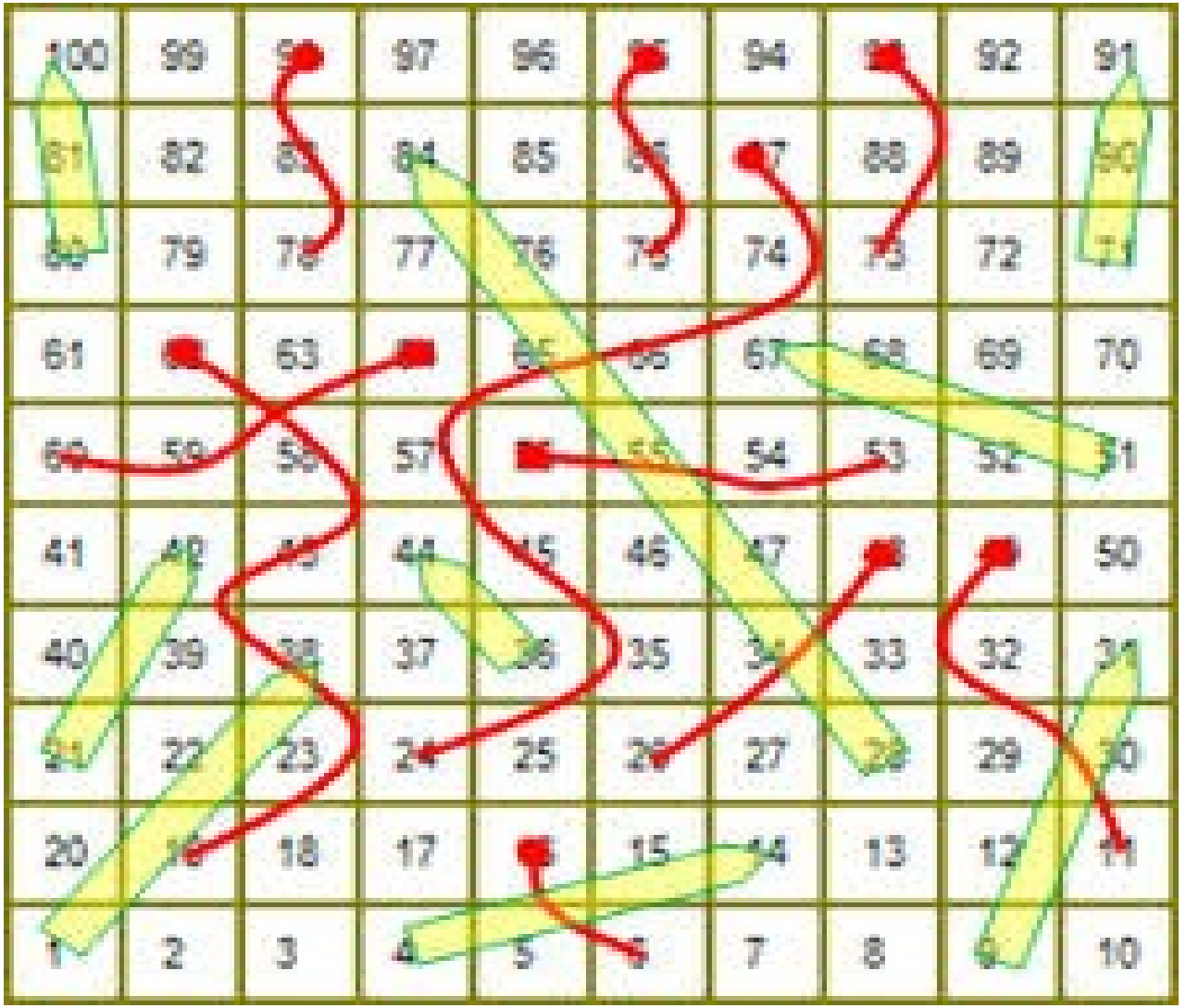
The matrix A may have an inverse. MATLAB computes it as inv(A). If so, v=inv(A)*w.

The determinant det(A) can warn whether A has an inverse.

Snakes and Ladders



A Sample Game Board



The Transition Matrix

This is another problem that can be described using a transition matrix P.

Our “states” are the numbered squares on the board. We have to add a state “0” which is where we are before the first roll of the die.

Our transitions are moves to new squares, and for (almost) every square, there are 6 new places to go to.

Initialize Transition Matrix P(:,0)

		Current Position
		0
Move to	0	.
	1	1/6
	2	1/6
	3	1/6
	4	1/6
	5	1/6
	6	1/6
	7	.

Modify Transition Matrix P(:,0)

		Current Position
		0
Move to	0	.
	1	0 <- Ladder to 38!
	2	1/6
	3	1/6
	4	0 <- Ladder to 14!
	5	1/6
	6	1/6
	7	.
	14	1/6
	38	1/6

Initialize Transition Matrix P(:,45:47)

		Current Position			
		45	46	47	
	45	.	.	.	
	46	1/6	.	.	
	47	1/6	1/6	.	
Move to	48	1/6	1/6	1/6	Snake to 26!
	49	1/6	1/6	1/6	Snake to 11!
	50	1/6	1/6	1/6	
	51	1/6	1/6	1/6	Ladder to 67!
	52	.	1/6	1/6	
	53	.	.	1/6	
	54	.	.	.	

Modify Transition Matrix P(:,45:47)

		Current Position			
		45	46	47	
	11	1/6	1/6	1/6	
	26	1/6	1/6	1/6	
	45	.	.	.	
	46	1/6	.	.	
	47	1/6	1/6	.	
Move to	48	0	0	0	Snake to 26!
	49	0	0	0	Snake to 11!
	50	1/6	1/6	1/6	
	51	0	0	0	Ladder to 67!
	52	.	1/6	1/6	
	53	.	.	1/6	
	54	.	.	.	
	67	1/6	1/6	1/6	

Initialize Transition Matrix P(:,94:100)

		Current Position						
		94	95	96	97	98	99	100
TO:	94	.	.	.	.	.	.	.
	95	1/6	.	.	.	.	.	.
	96	1/6	1/6	.	.	.	.	.
	97	1/6	1/6	1/6	.	.	.	.
	98	1/6	1/6	1/6	1/6	.	.	.
	99	1/6	1/6	1/6	1/6	1/6	.	.
	100	1/6	2/6	3/6	4/6	5/6	6/6	1

Modify Transition Matrix P(:,94:100)

		Current Position						
		94	95	96	97	98	99	100
TO:	75	1/6	.	.	.	.	.	.
	78	1/6	.	1/6	1/6	.	.	.
	94	.	.	.	.	.	.	.
	95	.	.	.	.	.	.	.
	96	1/6	.	.	.	.	.	.
	97	1/6	.	1/6	.	.	.	.
	98	.	.	.	.	.	.	.
	99	1/6	.	1/6	1/6	.	.	.
	100	1/6	.	3/6	4/6	.	6/6	1

Snake to 75!

Snake to 78!

Simulation using Transition Matrix

P contains the transition probabilities for one step of the game. If we create a column vector v of length 100, with a 1 at our current position, then w=P*v contains the probabilities for our next position.

But then P*P*v gives us probabilities after 2 rolls of the die, and P^n*v gives us the probabilities after n rolls.

Since the game is over when we reach square 100, we can compute keep multiplying P times v until the value of the value of the 100th entry is nonzero, which tells us the length of the shortest game, or when it is more than 1/2, which tells us that half of the games will have ended by step n.

To simulate a single game, need C

The P matrix deals with probabilities. As we have seen before with transition matrices, if we want to simulate a single case, we need to convert the P matrix (probability) into a C matrix (cumulative probability).

Then we can choose a random number R, and look at the column of the C matrix that describes our current state (location), and figure out where we move next.

Cumulative Matrix C(:,45:47)

	Current Position			
	45	46	47	
11	1/6	1/6	1/6	
26	2/6	2/6	2/6	
45	.	.	.	
46	3/6	.	.	
47	4/6	3/6	.	
Move to 48	0	0	0	Snake to 26!
49	0	0	0	Snake to 11!
50	5/6	4/6	3/6	
51	0	0	0	Ladder to 67!
52	.	5/6	4/6	
53	.	.	5/6	
54	.	.	.	
67	6/6	6/6	6/6	

Nick Berry's Article

Nick Berry used these ideas to determine that:

7 is the shortest possible game;

20 is the most common game length;

29 is the median game length;

36.2 is the mean game length.

He also determined which snakes and ladders are used most often.

Homework #11

hw053: Compute the cumulative probabilities for an event with outcomes that have unequal likelihoods.

hw054: Use a transition model to simulate how 1000 students shift between three dormitories.

hw055: Use a transition model to simulate how 1000 customers switch car insurance policies each year.

Homework #11 is due by midnight, Friday December 1st.

Homework #10 is due by tomorrow night.