Understanding matrices as transformations

Felipe Orihuela-Espina

```
%File: MatricesAsTransformations.mlx
%

%% Log
%
% 23-Oct-2024: FOE
%    + File created for module Intelligent Data Analysis (Autumn 2024)
%
```

Table of Contents

Deal with options

```
opt.destinationDir = ['..' filesep 'media' filesep];
opt.fontSize = 12;
opt.lineWidth = 2;
opt.saveFigures = true;
```

Matrices as transformations

Matrices are defined as rank 2 **tensors**, and you are used to see them as a rectangular array of elements, whether simple numbers or functions.

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

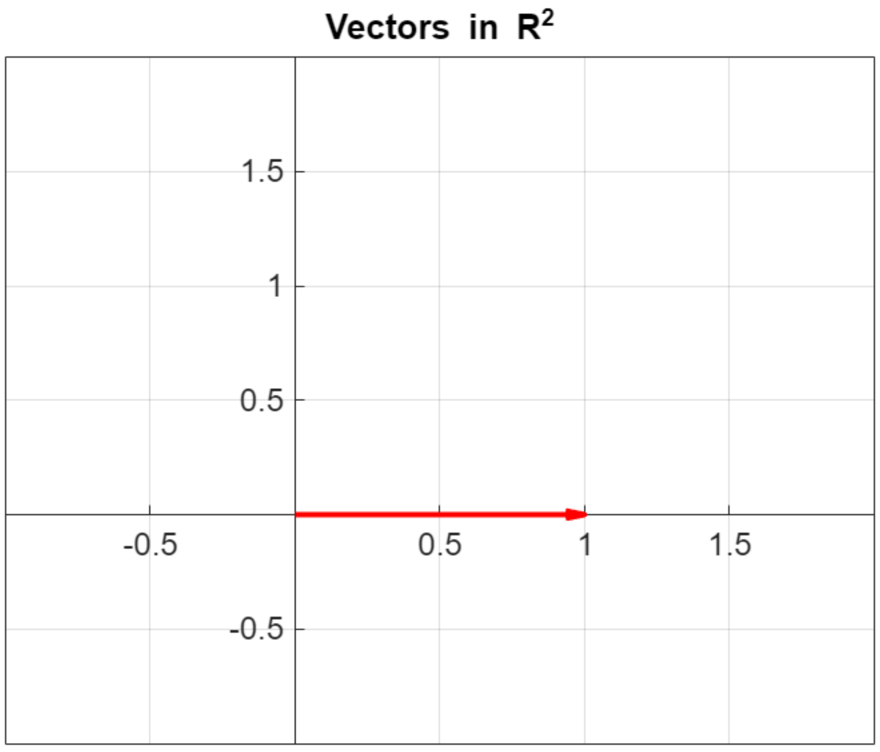
They also happen to represent (geometric) linear transformations. A **transformation** is a specific type of operation; in particular, a (linear) transformation is a mapping $V \to W$ between two vector spaces V and W that preserves the operations of vector addition and scalar multiplication. In other words, *we can use matrices to apply linear operations to vectors*. And because a point cloud in a vector space is nothing but a discrete collection of vectors, we can use matrices to apply linear operation to point clouds.

But let's go one step at a time. Let's start with perhaps the simplest case; 1 single vector $\mathbf{p}$ in the Euclidean space $\mathbb{R}^2$.

```
p = [1; 0] %Note that this is a column vector
```

```
p = 2×1
     1
     0
```

```
figure
quiver(0,0,p(1),p(2),0,'LineWidth',opt.lineWidth,'Color','r') %The "last" 0
after p(2) is to ignore the "scaling" of the vectors that the function quiver
uses otherwise.
set(gca,'XLim',[-1 2]);
set(gca,'YLim',[-1 2]);
set(gca,'XAxisLocation','origin');
set(gca,'YAxisLocation','origin');
title('Vectors in R^2','FontSize',opt.fontSize);
set(gca,'FontSize',opt.fontSize)
box on, grid on
```



```
if opt.saveFigures
    mySaveFig(gcf,[opt.destinationDir 'MatricesAsTransformations0001_Vectors']);
end
```

Now, let's apply some arbitrary transformation in the form of a matrix A; Let.

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

```
A = [1 1; 1 0];
```

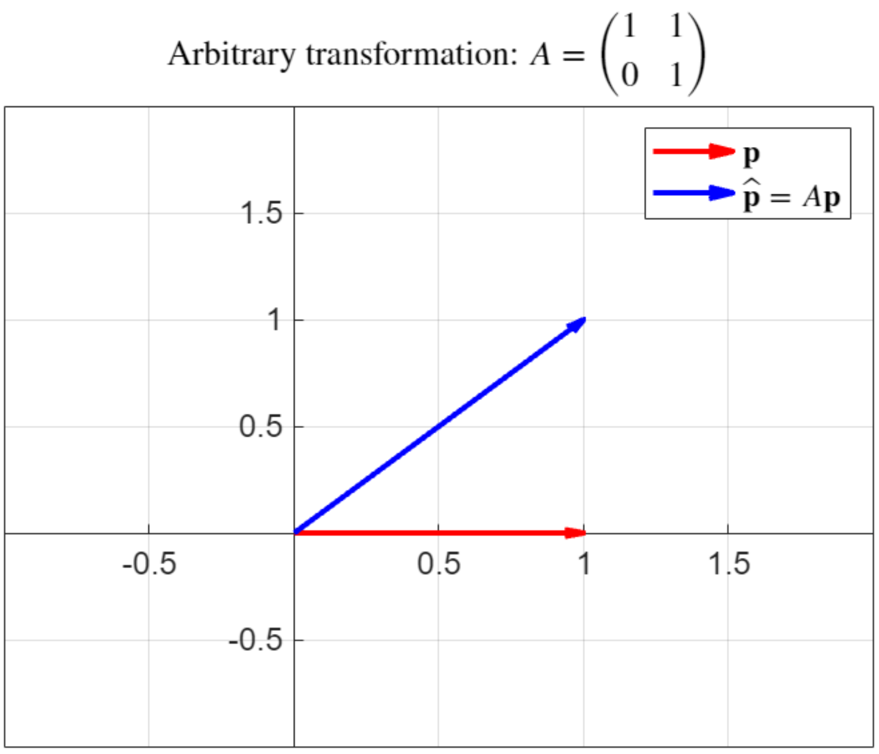
...and by applying the transformation we just mean multiplying by A. Then, our vector $\mathbf{p}$ becomes vector $\hat{\mathbf{p}}$;

$\hat{\mathbf{p}} = A\mathbf{p}$

```
hatp = A*p
```

```
hatp = 2×1
     1
     1
```

```

figure, hold on
h(1) = quiver(0,0,p(1),p(2),0,'LineWidth',opt.lineWidth,'Color','r');
h(2) = quiver(0,0,hatp(1),hatp(2),0,'LineWidth',opt.lineWidth,'Color','b');
set(gca,'XLim',[-1 2]);
set(gca,'YLim',[-1 2]);
set(gca,'XAxisLocation','origin');
set(gca,'YAxisLocation','origin');
title('Arbitrary transformation: $A=\left(\begin{array}{cc}1&1\\ 0&1\end{array}
\right)$','FontSize',opt.fontSize, 'Interpreter','latex');
set(gca,'FontSize',opt.fontSize)
box on, grid on
legend(h,{'$\mathbf{p}$','$\hat{\mathbf{p}}=A\mathbf{p}
$'},'FontSize',opt.fontSize, 'Interpreter','latex');
clear h
if opt.saveFigures
    mySaveFig(gcf,[opt.destinationDir
'MatricesAsTransformations0002_ArbtraryTransformation']);
end
```

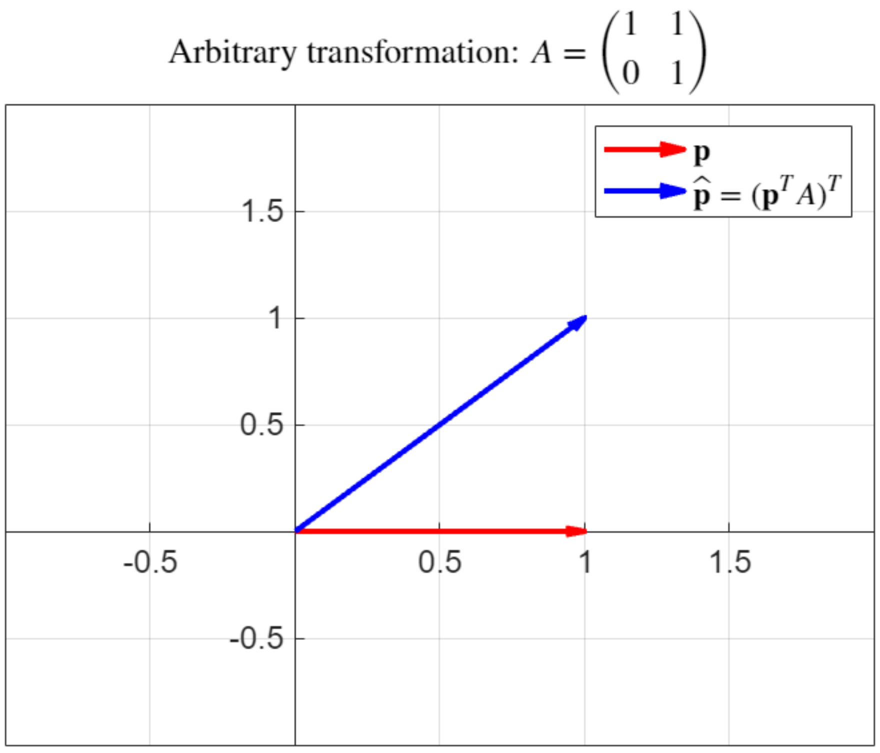


As this is an arbitrary transformation, it does not matter much where did the transformed vector $\widehat{\mathbf{p}}$ lands; all that matters right now is that this is the result of the transformation induced by the matrix A.

Remember that matrix multiplication is not commutative, hence it is important that you note that in this case, matrix A was pre-multiplying vector $\mathbf{p}$, that is , we apply $A\mathbf{p}$ rather than $\mathbf{p}A$ which would not have been possible to compute becuase, as you may remember, being a column vector sized 2x1 it would not have been comformable for the product with a matrix sized 2x2, i.e. (2x1)x(2x2). However, it is of course possible to obtain the same outcome post-multiplying but then we need to use the transpose of $\mathbf{p}$, $\mathbf{p}^T$ (now sized 1x2 and hence (1x2)x(2x2)).

$\widehat{\mathbf{p}} = A\mathbf{p} = (\mathbf{p}^T A)^T$

```
hatp_postmultiplied = (p'*A)';

figure, hold on
h(1) = quiver(0,0,p(1),p(2),0,'LineWidth',opt.lineWidth,'Color','r');
h(2) =
quiver(0,0,hatp_postmultiplied(1),hatp_postmultiplied(2),0,'LineWidth',opt.lineWi
dth,'Color','b');
set(gca,'XLim',[-1 2]);
set(gca,'YLim',[-1 2]);
set(gca,'XAxisLocation','origin');
set(gca,'YAxisLocation','origin');
title('Arbitrary transformation: $A=\left(\begin{array}{cc}1&1\\ 0&1\end{array}
\right)$','FontSize',opt.fontSize, 'Interpreter','latex');
set(gca,'FontSize',opt.fontSize)
box on, grid on
legend(h,{'$\mathbf{p}$','$\hat{\mathbf{p}}=(\mathbf{p}^T
A)^T$'},'FontSize',opt.fontSize, 'Interpreter','latex');
```



```
clear h
if opt.saveFigures
    mySaveFig(gcf,[opt.destinationDir
'MatricesAsTransformations0003_ArbtraryTransformationPostMultipled']);
end
```

So by now, let's stay on the "pre-multiplication", $\hat{\mathbf{p}} = A\mathbf{p}$ for the sake of simplicity.

The Identity matrix is the neutral element transformation

Let's A be the identity matrix $I_{2\times2} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$. The identity matrix is the neutral element transformation, i.e. your vectors will land on the same position where they started. It's like multiplying by 1 in your regular natural numbers, e.g. $1 \cdot 3 = 3$

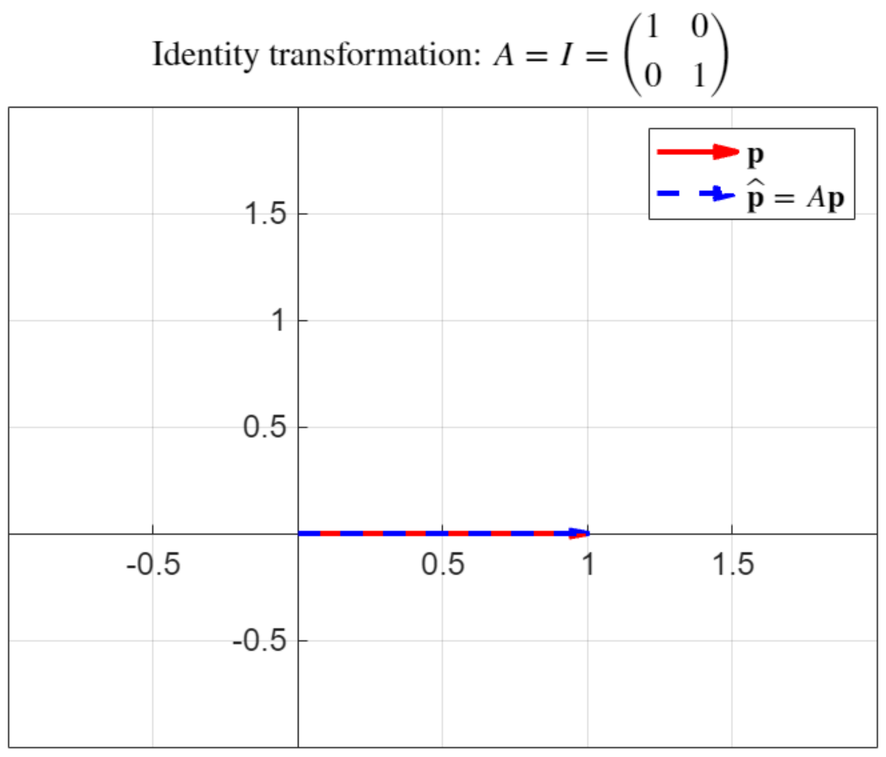
$$\hat{\mathbf{p}} = A\mathbf{p} \overset{A=I}{=} I\mathbf{p} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\mathbf{p} \quad \Rightarrow \quad \hat{\mathbf{p}} = \mathbf{p}$$

```
A = eye(2)
```

```
A = 2×2
```

```
     1     0
     0     1
```

```
hatp_byIdentity = A*p;

figure, hold on
h(1) = quiver(0,0,p(1),p(2),0,'LineWidth',opt.lineWidth,'Color','r');
h(2) =
quiver(0,0,hatp_byIdentity(1),hatp_byIdentity(2),0,'LineWidth',opt.lineWidth,'Col
or','b','LineStyle','--');
set(gca,'XLim',[-1 2]);
set(gca,'YLim',[-1 2]);
set(gca,'XAxisLocation','origin');
set(gca,'YAxisLocation','origin');
title('Identity transformation: $A=I=\left(\begin{array}{cc}1&0\\ 0&1\end{array}
\right)$','FontSize',opt.fontSize, 'Interpreter','latex');
set(gca,'FontSize',opt.fontSize)
box on, grid on
legend(h,{'$\mathbf{p}$','$\hat{\mathbf{p}}=A\mathbf{p}
$'},'FontSize',opt.fontSize, 'Interpreter','latex');
```



```
clear h
if opt.saveFigures
    mySaveFig(gcf,[opt.destinationDir
'MatricesAsTransformations0004_TransformationByIdentity']);
end
```

Focus on the relation between the elements of A rather than the values

In order to understand how a matrix produces a transformation, the specific values of the elements a_{ij} of the matrix are less relevant that the relative values among them. That is the arithmetics are only secondary to the algebraics when trying to understand how matrices work.

Let's take two matrices; of the form:

$$A=\begin{pmatrix} a & 0 \\ 0 & 1 \end{pmatrix}$$

```
A1 = [2 0; 0 1]
```

```
A1 = 2×2
     2     0
     0     1
```

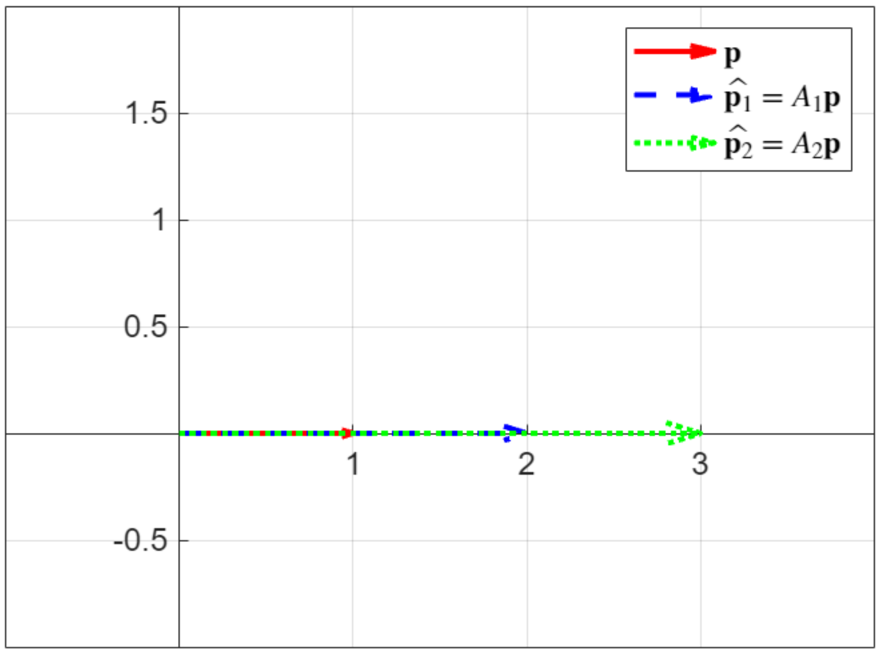
```
A2 = [3 0; 0 1]
```

```
A2 = 2×2
     3     0
     0     1
```

Regardless of the value of a, these two matrices are going to scale your vectors along the abcissa axes by a factor of 2 and 3 respectively, but the critical thing is that the transformation itself, i.e. the scaling over the abcissa axes, is fundamentally the same.

```
hatp_byScalingA1 = A1*p;
hatp_byScalingA2 = A2*p;

figure, hold on
h(1) = quiver(0,0,p(1),p(2),0,'LineWidth',opt.lineWidth,'Color','r');
h(2) =
quiver(0,0,hatp_byScalingA1(1),hatp_byScalingA1(2),0,'LineWidth',opt.lineWidth,'C
olor','b','LineStyle','--');
h(3) =
quiver(0,0,hatp_byScalingA2(1),hatp_byScalingA2(2),0,'LineWidth',opt.lineWidth,'C
olor','g','LineStyle',':');
set(gca,'XLim',[-1 4]);
set(gca,'YLim',[-1 2]);
set(gca,'XAxisLocation','origin');
set(gca,'YAxisLocation','origin');
title('Scaling transformation: $A=\left(\begin{array}{cc}a&0\\ 0&1\end{array}
\right)$','FontSize',opt.fontSize, 'Interpreter','latex');
set(gca,'FontSize',opt.fontSize)
box on, grid on
legend(h,{'$\mathbf{p}$','$\hat{\mathbf{p}_1}=A_1\mathbf{p}$','$
\hat{\mathbf{p}_2}=A_2\mathbf{p}$'},'FontSize',opt.fontSize,
'Interpreter','latex');
```

Scaling transformation: $A = \begin{pmatrix} a & 0 \\ 0 & 1 \end{pmatrix}$



```
clear h
if opt.saveFigures
    mySaveFig(gcf,[opt.destinationDir
'MatricesAsTransformations0005_TransformationByScaling']);
end
```

Indeed, no matter the value of a, any matris of the form

$$A = \begin{pmatrix} a & 0 \\ 0 & 1 \end{pmatrix}$$

...will invariably produce the same effect, a scaling over the abcissa axes.

What if I want to scale over the ordinate axis instead? Easy peasy!

$$A = \begin{pmatrix} 1 & 0 \\ 0 & a \end{pmatrix}$$

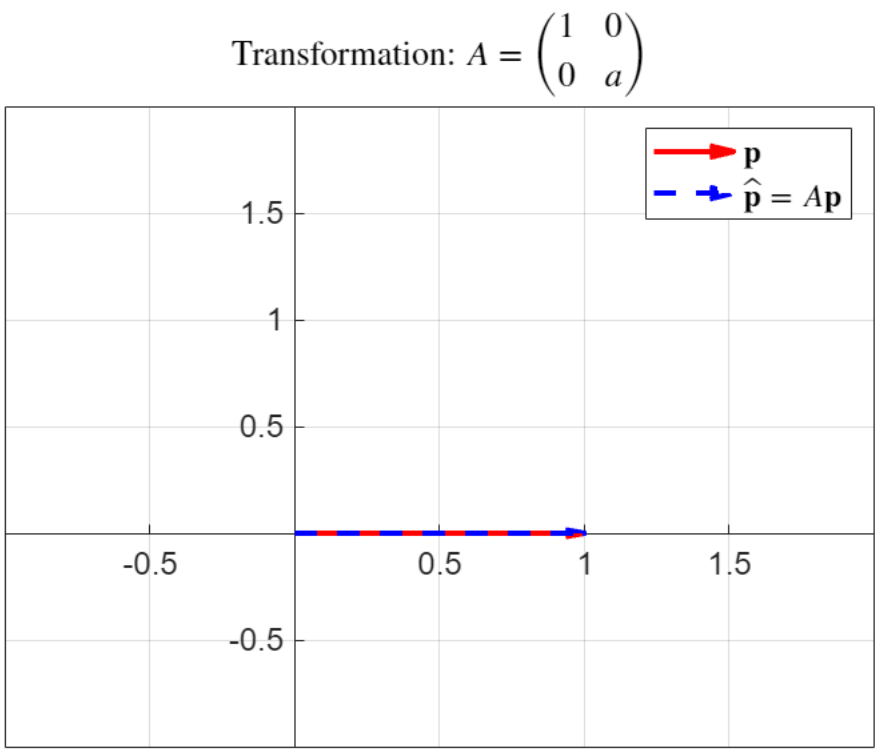
Of course, if you try this on $\mathbf{p}$, you are not going to *"see"* much, because $\mathbf{p}$ second coordinate is 0, and hence $a \cdot 0 = 0$

```
A3 = [1 0; 0 2]
```

```
A3 = 2×2
     1     0
     0     2
```

```
hatp_byScalingA3 = A3*p;

figure, hold on
```

```
h(1) = quiver(0,0,p(1),p(2),0,'LineWidth',opt.lineWidth,'Color','r');
h(2) =
quiver(0,0,hatp_byScalingA3(1),hatp_byScalingA3(2),0,'LineWidth',opt.lineWidth,'C
olor','b','LineStyle','--');
set(gca,'XLim',[-1 2]);
set(gca,'YLim',[-1 2]);
set(gca,'XAxisLocation','origin');
set(gca,'YAxisLocation','origin');
title('Transformation: $A=\left(\begin{array}{cc}1&0\\ 0&a\end{array}\right)
$','FontSize',opt.fontSize, 'Interpreter','latex');
set(gca,'FontSize',opt.fontSize)
box on, grid on
legend(h,{'$\mathbf{p}$','$\hat{\mathbf{p}}=A\mathbf{p}
$'},'FontSize',opt.fontSize, 'Interpreter','latex');
```



```
clear h
if opt.saveFigures
    mySaveFig(gcf,[opt.destinationDir
'MatricesAsTransformations0006_TransformationByScaling']);
end
```

..but if we choose, any other vector, e.g.

```
p2 = [1;1]
```

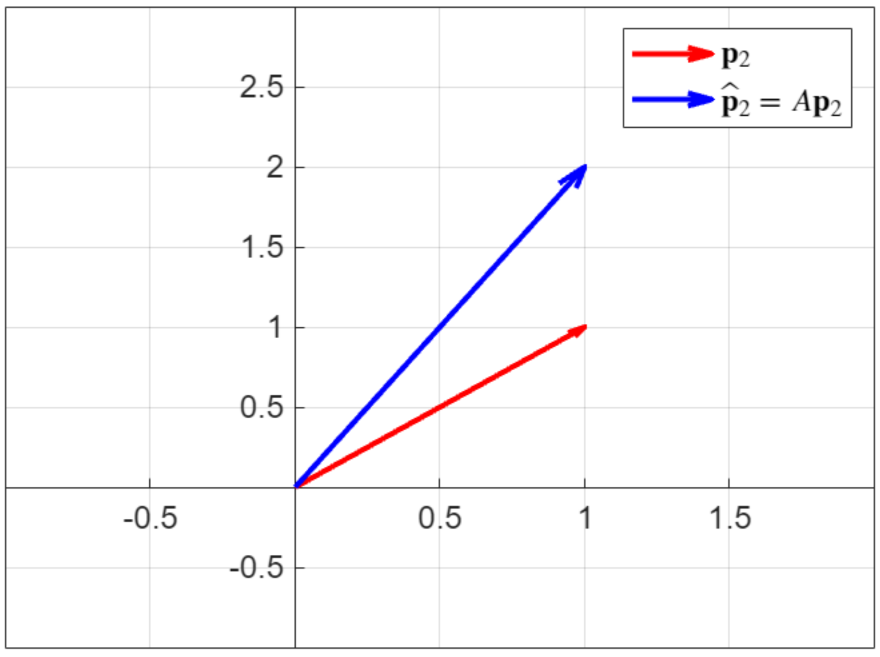
```
p2 = 2×1
     1
     1
```

then, you can now see the effect of A clearly;

```
hatp2_byScalingA3 = A3*p2;
```

```
figure, hold on
h(1) = quiver(0,0,p2(1),p2(2),0,'LineWidth',opt.lineWidth,'Color','r');
h(2) =
quiver(0,0,hatp2_byScalingA3(1),hatp2_byScalingA3(2),0,'LineWidth',opt.lineWidth,
'Color','b');
set(gca,'XLim',[-1 2]);
set(gca,'YLim',[-1 3]);
set(gca,'XAxisLocation','origin');
set(gca,'YAxisLocation','origin');
title('Transformation: $A=\left(\begin{array}{cc}1&0\\ 0&a\end{array}\right)
$','FontSize',opt.fontSize, 'Interpreter','latex');
set(gca,'FontSize',opt.fontSize)
box on, grid on
legend(h,{'$\mathbf{p}_2$','$
\hat{\mathbf{p}}_2=A\mathbf{p}_2$'},'FontSize',opt.fontSize,
'Interpreter','latex');
```

Transformation: $A = \left(\begin{array}{cc}1&0\\ 0&a\end{array}\right)$



```
clear h
if opt.saveFigures
    mySaveFig(gcf,[opt.destinationDir
'MatricesAsTransformations0007_TransformationByScaling']);
end
```

Transforming a point cloud

We can apply the same transformation to a point cloud, and all the vectors will get the same transformation. So if in our example, we concatenate the vectors we have been playing with; $\mathbf{p}_1 = \mathbf{p}$ and $\mathbf{p}_2$;

$$P = [\mathbf{p}_1 \quad \mathbf{p}_2] = \begin{bmatrix} \mathbf{p}_1^1 & \mathbf{p}_2^1 \\ \mathbf{p}_1^2 & \mathbf{p}_2^2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

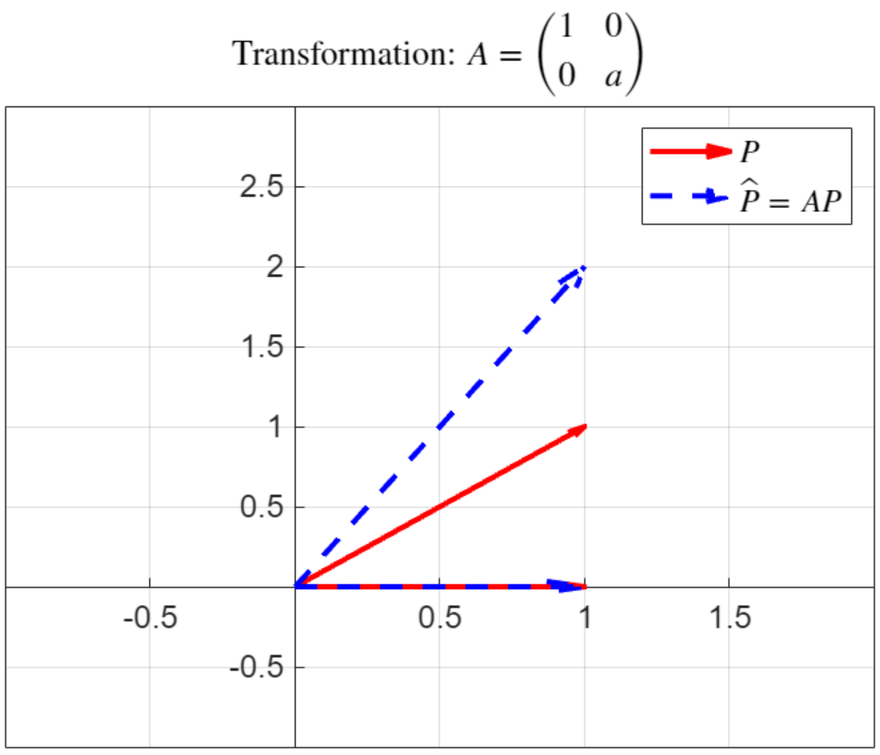
```
p1 = p;
P= [p1 p2]
```

```
P = 2×2
     1     1
     0     1
```

Now P is a **point cloud**.

And we now pre-multiply our point cloud P by A, that is, AP :

```
hatP_byScalingA3 = A3*P;
nPoints = size(P,2);
OO = zeros(2,nPoints); %One cop of the origin for each vector in the point
cloud. Helps to keep the code clean, but otherwise is just a matrix of 0s.

figure, hold on
h(1) =
quiver(OO(1,:),OO(2,:),P(1,:),P(2,:),0,'LineWidth',opt.lineWidth,'Color','r');
h(2) =
quiver(OO(1,:),OO(2,:),hatP_byScalingA3(1,:),hatP_byScalingA3(2,:),0,'LineWidth',
opt.lineWidth,'Color','b','LineStyle','--');
set(gca,'XLim',[-1 2]);
set(gca,'YLim',[-1 3]);
set(gca,'XAxisLocation','origin');
set(gca,'YAxisLocation','origin');
title('Transformation: $A=\left(\begin{array}{cc}1&0\\ 0&a\end{array}\right)
$','FontSize',opt.fontSize, 'Interpreter','latex');
set(gca,'FontSize',opt.fontSize)
box on, grid on
legend(h,{'$P$','$\hat{P}=AP$'},'FontSize',opt.fontSize, 'Interpreter','latex');
```



```
clear h
if opt.saveFigures
    mySaveFig(gcf,[opt.destinationDir
'MatricesAsTransformations0008_TransformationByScaling_PointCloud']);
end
```

General scaling

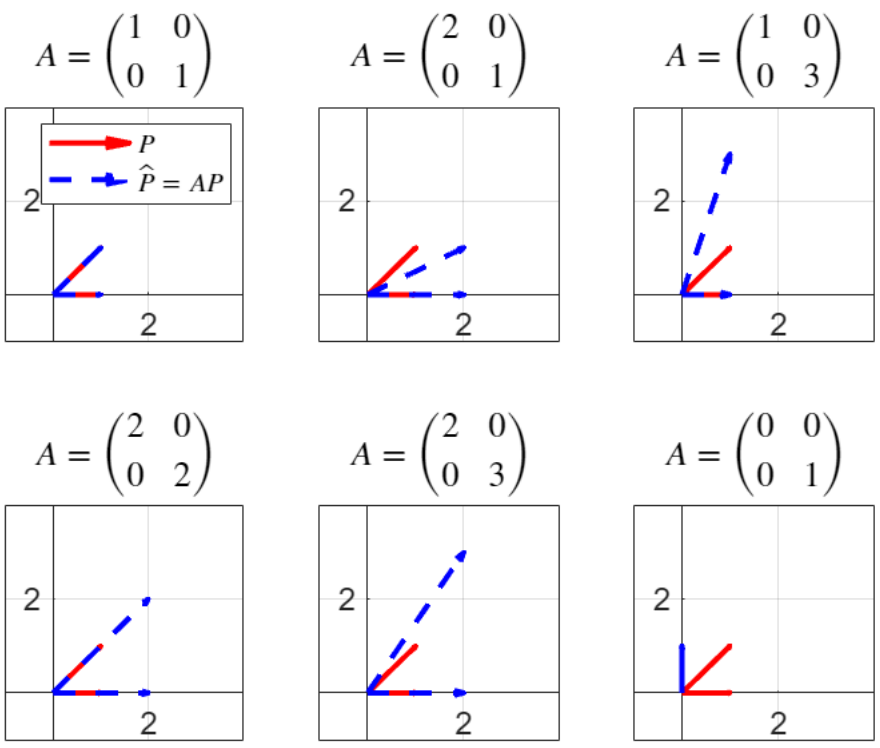
As long as outside the main diagonal of A everything is 0, then A is just a simple scaling, and how much do we scale obviously depends on the specific coefficients. Let

$$A = \begin{pmatrix} a_1 & 0 \\ 0 & a_2 \end{pmatrix}$$

Here are a couple of examples; put attention to the "relation" between a_1 and a_2.

```
A(:,:,1) = eye(2); %Identity matrix
A(:,:,2) = [2 0; 0 1]; % Scale along abcissa
A(:,:,3) = [1 0; 0 3]; % Scale along ordinate
A(:,:,4) = [2 0; 0 2]; % General scaling but same scaling across both axes
A(:,:,5) = [2 0; 0 3]; % General scaling but different scaling for each axes;
ordinate axes is 3/2 = 1.5 more stretched than abscissa.
A(:,:,6) = [0 0; 0 1]; % Reducing dimensionality by nullifying the first
coordinate
nExamples=size(A,3);

figure
```

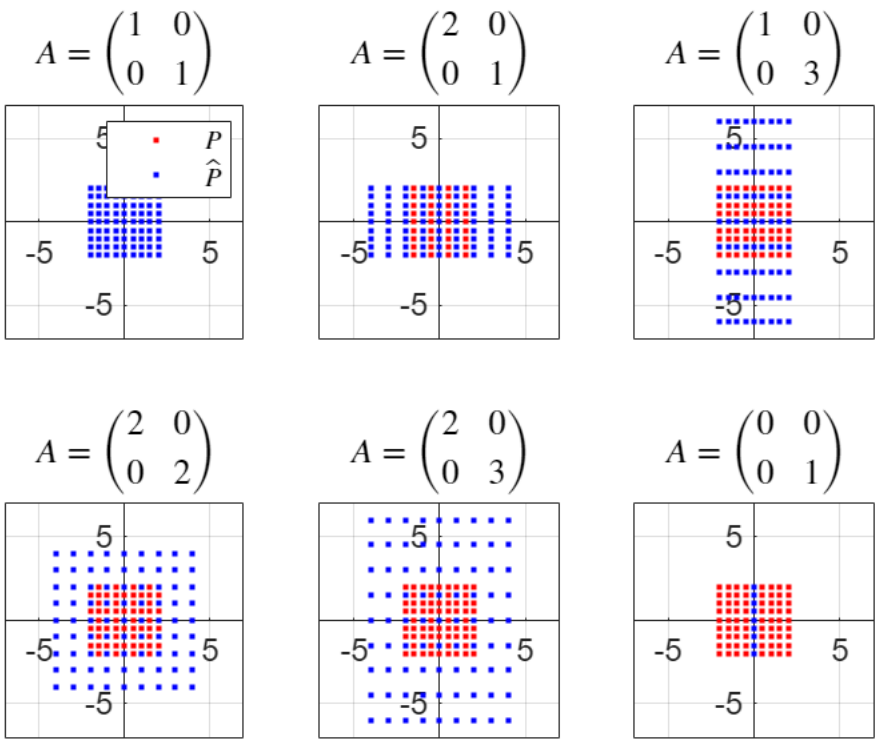
```
for iExample = 1:nExamples
    tmpA = squeeze(A(:,:,iExample));
    hatPi = tmpA*P;

    subplot(2,3,iExample), hold on
    h(1) =
quiver(OO(1,:),OO(2,:),P(1,:),P(2,:),0,'LineWidth',opt.lineWidth,'Color','r');
    h(2) =
quiver(OO(1,:),OO(2,:),hatPi(1,:),hatPi(2,:),0,'LineWidth',opt.lineWidth,'Color',
'b','LineStyle','--');

    set(gca,'XLim',[-1 4]);
    set(gca,'YLim',[-1 4]);
    set(gca,'XAxisLocation','origin');
    set(gca,'YAxisLocation','origin');
    tmpTitle = ['$A=\left(\begin{array}{cc}' num2str(tmpA(1,1)) '&'
num2str(tmpA(1,2)) '\\ ' ...
                         num2str(tmpA(2,1)) '&' num2str(tmpA(2,2)) '\end{array}
\right)$'];
    title(tmpTitle,'FontSize',opt.fontSize, 'Interpreter','latex');
    set(gca,'FontSize',opt.fontSize)
    box on, grid on
    if iExample == 1
        legend(h,{'$P$','$\hat{P}=AP$'},'FontSize',opt.fontSize,
'Interpreter','latex','FontSize',opt.fontSize-2);
    end
    clear h
end
if opt.saveFigures
    mySaveFig(gcf,[opt.destinationDir
'MatricesAsTransformations0009_GeneralScaling_CloudAsVectors']);
end
```



Perhaps the effect it is more clear if we "abandon" now the vector pictorial depiction and we only keep the point cloud per se and we use a grid like cloud of points. Note how the "center" of the scaling is the origin.

```
X1=[-2:0.5:2];
X2=[-2:0.5:2];
[XX1,XX2]=ndgrid(X1,X2);
P = [XX1(:)'; XX2(:)'];

figure
for iExample = 1:nExamples
    tmpA = squeeze(A(:,:,iExample));
    hatPi = tmpA*P;

    subplot(2,3,iExample),
    hold on
    h(1) = plot(P(1,:),P(2,:),'LineWidth',opt.lineWidth-0.5,'Color','r',...
        'Marker','.','LineStyle','none');
    h(2) =
plot(hatPi(1,:),hatPi(2,:),'LineWidth',opt.lineWidth-0.5,'Color','b',...
        'Marker','.','LineStyle','none');

    set(gca,'XLim',[-7 7]);
    set(gca,'YLim',[-7 7]);
    set(gca,'XAxisLocation','origin');
    set(gca,'YAxisLocation','origin');
    tmpTitle = ['$A=\left(\begin{array}{cc}' num2str(tmpA(1,1)) '&'
num2str(tmpA(1,2)) '\\ ' ...
```

```
                          num2str(tmpA(2,1)) '&' num2str(tmpA(2,2)) '\end{array}
\right)$'];
    title(tmpTitle,'FontSize',opt.fontSize, 'Interpreter','latex');
    set(gca,'FontSize',opt.fontSize)
    box on, grid on
    if iExample == 1
        legend(h,{'$P$','$\hat{P}$'},
'Interpreter','latex','FontSize',opt.fontSize-2);
    end
    clear h
end
if opt.saveFigures
    mySaveFig(gcf,[opt.destinationDir
'MatricesAsTransformations0010_GeneralScaling_CloudAsPoints']);
end
```



Can you guess what happens when the relation between a_1 and a_2 is $a_1 = \frac{1}{a_2}$?

Hint: Think squeezing.

Shearing

Plain scaling is not the only transformation that you can achieve using matrices as transformations. If instead of perturbing the main diagonal, you perturb outside the main diagonal, you get a shearing effect. For instance, let $A_{shearing}$ be

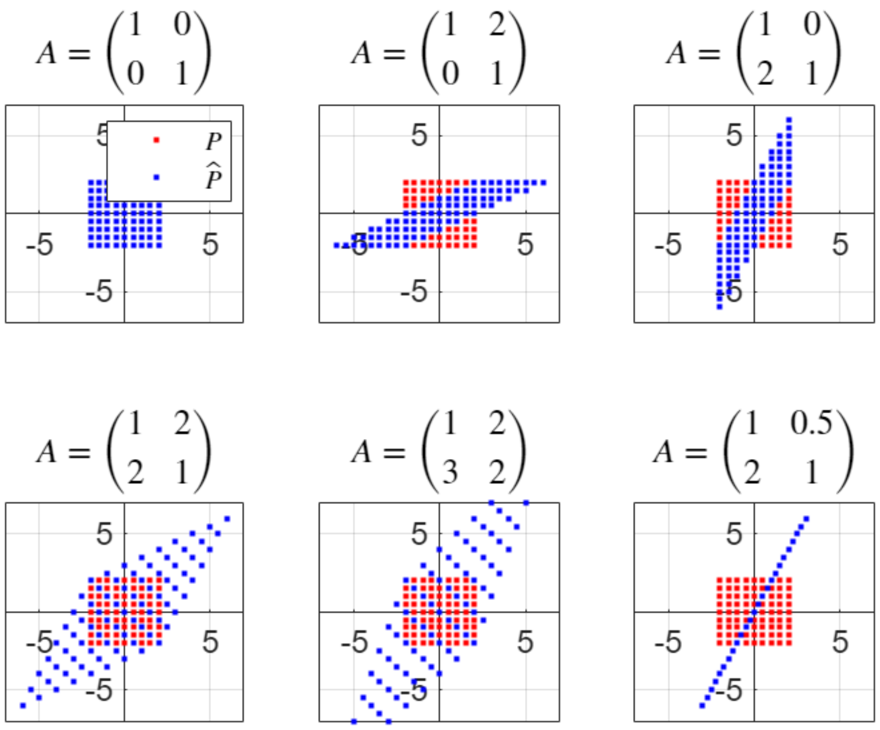
$$A_{shearing} = \begin{pmatrix} 1 & a \\ 0 & 1 \end{pmatrix}$$

Note how in this example we are keeping the main diagonal all to 1 so we do not get other spurious scaling.

Now we can transform our point cloud as usual; $\widehat{P} = AP$

```
A_shearing(:,:,1) = eye(2); %Identity matrix. No shear
A_shearing(:,:,2) = [1 2; 0 1]; % Shear along abcissa
A_shearing(:,:,3) = [1 0; 2 1]; % Shear along ordinate
A_shearing(:,:,4) = [1 2; 2 1]; % General shearing but same shear across both
axes
A_shearing(:,:,5) = [1 2; 3 2]; % General shearing but different shearing for
each axes; ordinate axes is 3/2 = 1.5 more sheared than abscissa.
A_shearing(:,:,6) = [1 0.5; 2 1]; % Shearing and squeezing
nExamples=size(A,3);

X1=[-2:0.5:2];
X2=[-2:0.5:2];
[XX1,XX2]=ndgrid(X1,X2);
P = [XX1(:)'; XX2(:)'];

figure
for iExample = 1:nExamples
    tmpA = squeeze(A_shearing(:,:,iExample));
    hatPi = tmpA*P;

    subplot(2,3,iExample),
    hold on
    h(1) = plot(P(1,:),P(2,:),'LineWidth',opt.lineWidth-0.5,'Color','r',...
        'Marker','.','LineStyle','none');
    h(2) =
plot(hatPi(1,:),hatPi(2,:),'LineWidth',opt.lineWidth-0.5,'Color','b',...
        'Marker','.','LineStyle','none');

    set(gca,'XLim',[-7 7]);
    set(gca,'YLim',[-7 7]);
    set(gca,'XAxisLocation','origin');
    set(gca,'YAxisLocation','origin');
    tmpTitle = ['$A=\left(\begin{array}{cc}' num2str(tmpA(1,1)) '&'
num2str(tmpA(1,2)) '\\ ' ...
                        num2str(tmpA(2,1)) '&' num2str(tmpA(2,2)) '\end{array}
\right)$'];
    title(tmpTitle,'FontSize',opt.fontSize, 'Interpreter','latex');
    set(gca,'FontSize',opt.fontSize)
    box on, grid on
    if iExample == 1
        legend(h,{'$P$','$\hat{P}$'},
'Interpreter','latex','FontSize',opt.fontSize-2);
    end
    clear h
end
if opt.saveFigures
```

```
    mySaveFig(gcf,[opt.destinationDir
'MatricesAsTransformations0011_Shearing_CloudAsPoints']);
end
```



Again, note that the center of the operation is the origin $(0, 0)$.

Rotation matrices

Many of the most common transformations with matrices are very well understood by now; scaling and shearing are but two examples. Other examples of matrix transformations that are well understood are for instance reflection or projections. But let's stop in one that it is particulary important for us: **rotations**.

A **rotation matrix** is a transformation matrix that is used to perform a rotation in the Euclidean space.

In $\mathbb{R}^2$, for a given rotation angle θ, the rotation matrix looks like:

$$A_\theta = \begin{pmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{pmatrix}$$

For instance, let's start by rotating our vector $\mathbf{p}$ by $\theta = 45^\circ = \frac{\pi}{4}$. We can define A as:

$$A_\theta = \begin{pmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{pmatrix} = \begin{pmatrix} cos(\pi/4) & -sin(\pi/4) \\ sin(\pi/4) & cos(\pi/4) \end{pmatrix} = \begin{pmatrix} 0.7071 & -0.7071 \\ 0.7071 & 0.7071 \end{pmatrix}$$

```
theta = pi/4;
```

```
Arot = [cos(theta) -sin(theta); sin(theta) cos(theta)]
```

```
Arot = 2×2
    0.7071   -0.7071
    0.7071    0.7071
```

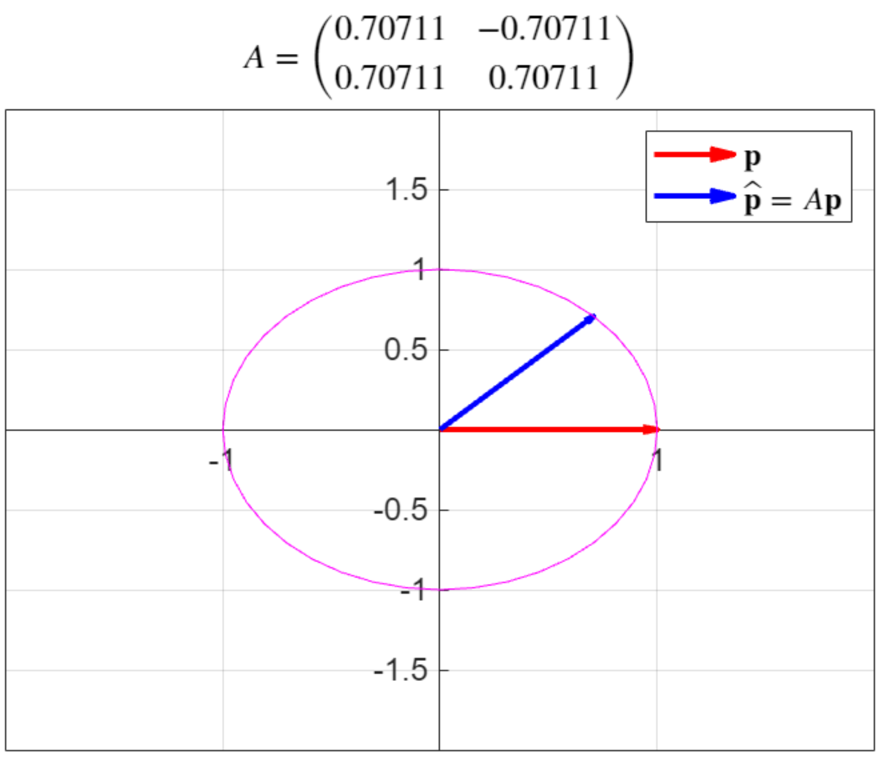
An proceed to multiply our vector $\mathbf{p}$ by A just like we did before;

$\hat{\mathbf{p}} = A\mathbf{p}$

```
hatp_byRotation = Arot*p;
```

Rendering the unit circle facilitates verifying that the vector has not been scaled in magnitude.

```

figure, hold on
h(1) = quiver(0,0,p(1),p(2),0,'LineWidth',opt.lineWidth,'Color','r');
h(2) =
quiver(0,0,hatp_byRotation(1),hatp_byRotation(2),0,'LineWidth',opt.lineWidth,'Col
or','b');
tmpTheta = [0:pi/20:2*pi];
plot(cos(tmpTheta),sin(tmpTheta),'m'); %Display the unit circle

set(gca,'XLim',[-2 2]);
set(gca,'YLim',[-2 2]);
set(gca,'XAxisLocation','origin');
set(gca,'YAxisLocation','origin');
tmpTitle = ['$A=\left(\begin{array}{cc}' num2str(Arot(1,1)) '&'
num2str(Arot(1,2)) '\\ ' ...
                              num2str(Arot(2,1)) '&' num2str(Arot(2,2)) '\end{array}
\right)$'];
title(tmpTitle,'FontSize',opt.fontSize, 'Interpreter','latex');
set(gca,'FontSize',opt.fontSize)
box on, grid on
legend(h,{'$\mathbf{p}$','$\hat{\mathbf{p}}=A\mathbf{p}
$'},'FontSize',opt.fontSize, 'Interpreter','latex');
clear h
if opt.saveFigures
    mySaveFig(gcf,[opt.destinationDir
'MatricesAsTransformations0012_TransformationByRotation_Vector']);
end
```

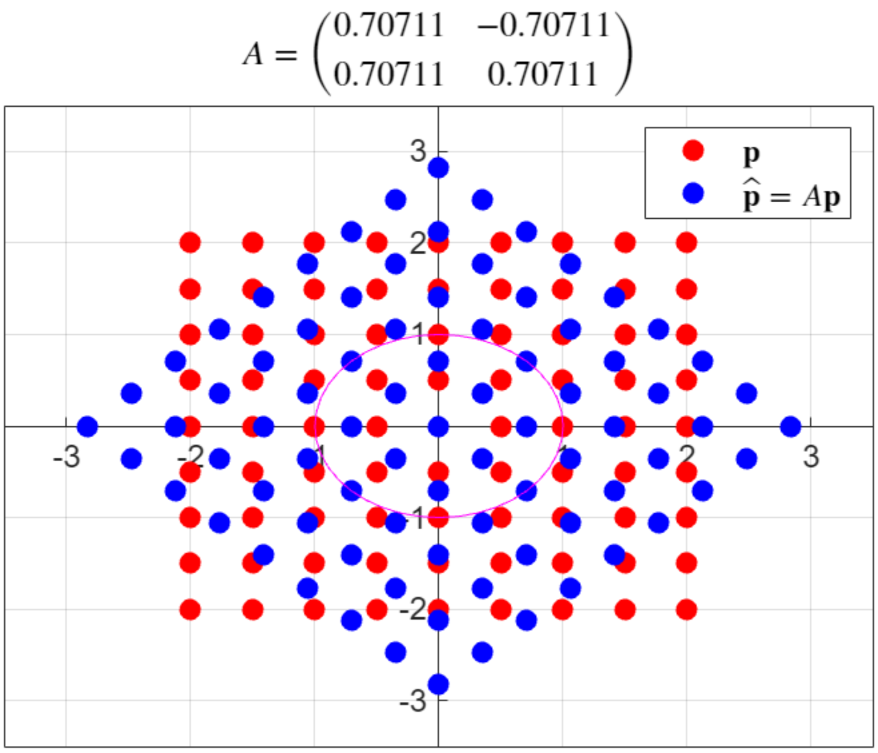


For a full point cloud, literally nothing changes other than the data itself!

$\hat{\mathbf{P}} = A\mathbf{P}$

```
hatP_byRotation = (Arot)*P;

figure, hold on
h(1) = plot(P(1,:),P(2,:),'LineWidth',opt.lineWidth,'Color','r',...
        'Marker','o','LineStyle','none','MarkerFaceColor','r','MarkerSize',6);
h(2) =
plot(hatP_byRotation(1,:),hatP_byRotation(2,:),'LineWidth',opt.lineWidth,'Color',
'b',...
        'Marker','o','LineStyle','none','MarkerFaceColor','b','MarkerSize',6);
tmpTheta = [0:pi/20:2*pi];
plot(cos(tmpTheta),sin(tmpTheta),'m'); %Display the unit circle

set(gca,'XLim',[-3.5 3.5]);
set(gca,'YLim',[-3.5 3.5]);
set(gca,'XAxisLocation','origin');
set(gca,'YAxisLocation','origin');
tmpTitle = ['$A=\left(\begin{array}{cc}' num2str(Arot(1,1)) '&'
num2str(Arot(1,2)) '\\ ' ...
                              num2str(Arot(2,1)) '&' num2str(Arot(2,2)) '\end{array}
\right)$'];
title(tmpTitle,'FontSize',opt.fontSize, 'Interpreter','latex');
set(gca,'FontSize',opt.fontSize)
box on, grid on
```

```
legend(h,{'$\mathbf{p}$','$\hat{\mathbf{p}}=A\mathbf{p}
$'},'FontSize',opt.fontSize, 'Interpreter','latex');
clear h
if opt.saveFigures
    mySaveFig(gcf,[opt.destinationDir
'MatricesAsTransformations0013_TransformationByRotation_PointCloud']);
end
```



...and yet again, the operation is performed around the origin $(0, 0)$, a.k.a. the *zero vector*. Note as well that the rotation is *clockwise*.

IMPORTANT:

- Rotation matrices are orthogonal, that is $A^T A = AA^T = I$, from where it follows that
- a matrix A is orthogonal if its transpose is equal to its inverse.
- a rotation matrix A ought to be square (otherwise it has no inverse).
- Rotation matrices have determinant 1, i.e. $det(A) = 1$
- If the determinant is not 1, a rotation effect may still happen, but you also get a scaling at the same time. It is customary in mathematics NOT to call the matrices that produce these unnormalized rotations a rotation matrix (as far as I know they usually do not receive a particular name).

Have a look at Wikipedia:Rotation_matrix if you are curious to see how the fundamental rotation matrices look in 3D.

PCA as a rotation

As we already know, PCA is the eigendecomposition of the covariance matrix, and this is equivalent to a rotation

```
X1=[-5:0.5:5];
X2=[-1:0.5:1];
[XX1,XX2]=ndgrid(X1,X2);
P2 = [XX1(:)'; XX2(:)'];
%Rotate P2 by some arbitrary \theta
theta = pi/8;
Arot = [cos(theta) -sin(theta); sin(theta) cos(theta)];
P2 = (Arot)*P2;
P2 = P2';
size(P2) %sanity check
```

```
ans = 1×2
   105     2
```

```

%Mean center
P2centered = P2-mean(P2);
size(P2centered) %sanity check
```

```
ans = 1×2
   105     2
```

```

%Now calculate the covariance matrix and PCA
P2cov = cov(P2centered)
```

```
P2cov = 2×2
    7.9734    3.0936
    3.0936    1.7862
```

```
[Q,D,W] = eig(P2cov) %Columns of Q contains the right-eigenvectors of P2, D the
eigenvalues along the main diagonal, and W the left eigenvectors
```

```
Q = 2×2
    0.3827   -0.9239
   -0.9239   -0.3827
D = 2×2
    0.5048         0
         0    9.2548
W = 2×2
    0.3827   -0.9239
   -0.9239   -0.3827
```

```

%We can see that the eigenvalues are not sorted, so let's sort them:
tmpD = diag(D);
[tmpD,ridx]=sortrows(tmpD,'descend');
D(eye(2)==1) = tmpD
```

```
D = 2×2
    9.2548         0
         0    0.5048
```

```
%...and also sort our eigenvectors accordingly.
Q = Q(:,ridx)
```

```
Q = 2×2
   -0.9239    0.3827
   -0.3827   -0.9239
```

```
W = W(ridx,:)
```

```
W = 2×2
   -0.9239   -0.3827
    0.3827   -0.9239
```

```
%Show that Q is a rotation matrix;
% 1) It is orthogonal i.e. QQ^T = I
% 2) Its det(Q)=1
Q*Q'
```

```
ans = 2×2
     1     0
     0     1
```

```
W*W'
```

```
ans = 2×2
     1     0
     0     1
```

```
det(Q)
```

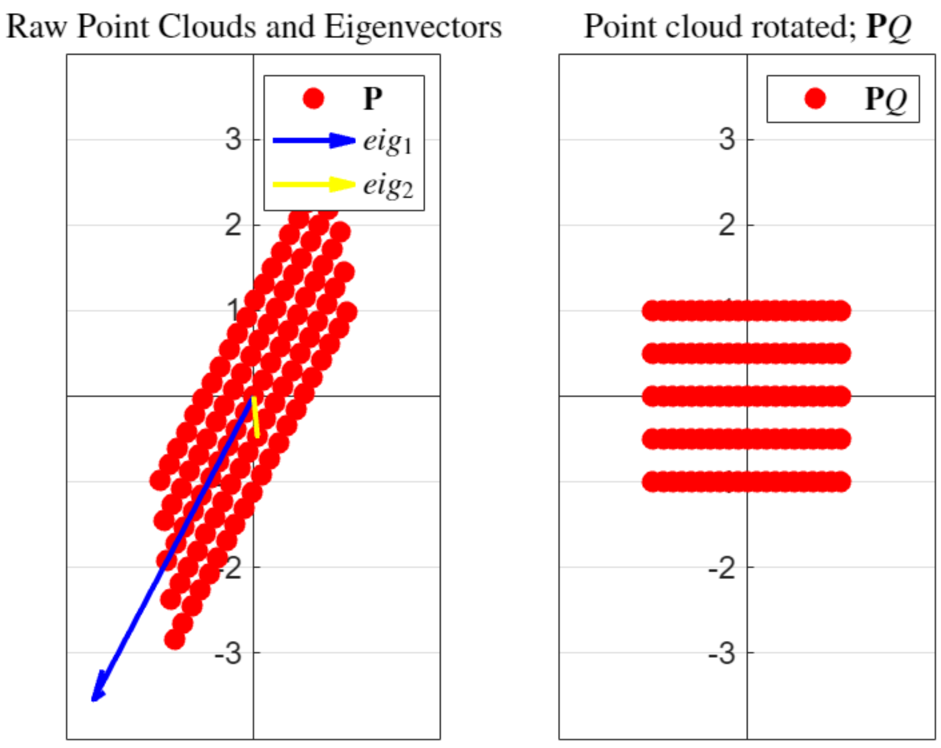
```
ans = 1
```

```
det(W)
```

```
ans = 1
```

```
figure,
subplot(1,2,1), hold on
h(1) =
plot(P2centered(:,1),P2centered(:,2),'LineWidth',opt.lineWidth,'Color','r',...
        'Marker','o','LineStyle','none','MarkerFaceColor','r','MarkerSize',6);
h(2) =
quiver(0,0,D(1,1)*Q(1,1),D(1,1)*Q(2,1),0,'LineWidth',opt.lineWidth,'Color','b');
h(3) =
quiver(0,0,D(2,2)*Q(1,2),D(2,2)*Q(2,2),0,'LineWidth',opt.lineWidth,'Color','y');

set(gca,'XLim',[-10 10]);
set(gca,'YLim',[-4 4]);
set(gca,'XAxisLocation','origin');
set(gca,'YAxisLocation','origin');
tmpTitle = ['Raw Point Clouds and Eigenvectors'];
title(tmpTitle,'FontSize',opt.fontSize, 'Interpreter','latex');
set(gca,'FontSize',opt.fontSize)
box on, grid on
legend(h,{'$\mathbf{P}$','$eig_1$','$eig_2$'},'FontSize',opt.fontSize,
'Interpreter','latex');
clear h
```

```
subplot(1,2,2), hold on
%Remember:
%   cov(P2)*Q = Q*D
% or alternatively:
%   W'*cov(P2) = D*W'
%
%Rotate the point cloud by simply multiplying by the eigenvectors Q
hatP2 = P2centered*Q; %Using the right eigenvectors
%hatP2 = (W*P2centered')'; %Using the left eigenvectors
h(1) = plot(hatP2(:,1),hatP2(:,2),'LineWidth',opt.lineWidth,'Color','r',...
        'Marker','o','LineStyle','none','MarkerFaceColor','r','MarkerSize',6);

set(gca,'XLim',[-10 10]);
set(gca,'YLim',[-4 4]);
set(gca,'XAxisLocation','origin');
set(gca,'YAxisLocation','origin');
tmpTitle = ['Point cloud rotated; $\mathbf{P}Q$'];
title(tmpTitle,'FontSize',opt.fontSize, 'Interpreter','latex');
set(gca,'FontSize',opt.fontSize)
box on, grid on
legend(h,{'$\mathbf{P}Q$'},'FontSize',opt.fontSize, 'Interpreter','latex');
clear h
if opt.saveFigures
    mySaveFig(gcf,[opt.destinationDir
'MatricesAsTransformations0014_RotationByPCA']);
end
```



Translation

Note that thus far, all of our operations are centered around the origin $(0,0)$. If you want to operate using the center of the operation different from the origin, one thing you can do is;

1. Translate your points so that the center is where you want to be
2. Do whatever operation you need
3. Translate back.

This is why you have to mean center your data when applying PCA or ICA!

But:

1. **how do we translate our cloud of points?**
2. **Is there a matrix transformation that do this?**

The answer to the second question is yes, there is such matrix, but the *how* is not that simple.

Again, let's start with what is likely the simplest case. Remember; the three major operations;

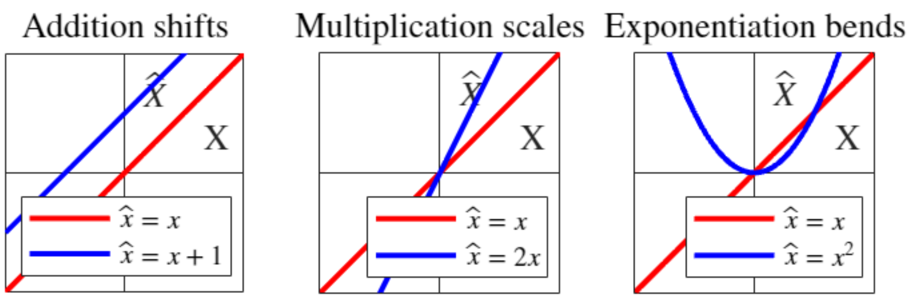
- Addition results in shifing or moving things i.e. translation
- Multiplication results in scaling
- Exponentiation results in bending

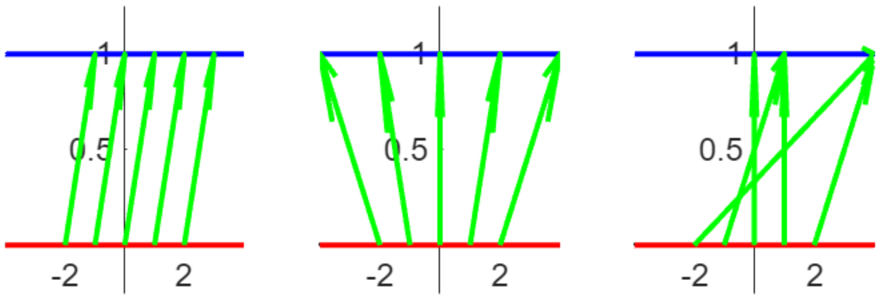
This is easy to see over the $y = x$ line;

```
x = [-2:.1:2];

figure
subplot(2,3,1), hold on
h(1)=plot(x,x,'r-','LineWidth',opt.lineWidth);
h(2)=plot(x,x+1,'b-','LineWidth',opt.lineWidth);

daspect([1 1 1])
set(gca,'XLim',[-2 2]);
set(gca,'YLim',[-2 2]);
set(gca,'XAxisLocation','origin');
set(gca,'YAxisLocation','origin');
xlabel('X','FontSize',opt.fontSize, 'Interpreter','latex');
ylabel('$\hat{X}$','FontSize',opt.fontSize, 'Interpreter','latex');
title('Addition shifts','FontSize',opt.fontSize, 'Interpreter','latex');
set(gca,'FontSize',opt.fontSize)
box on, grid on
legend(h,{'$\hat{x}=x$','$\hat{x}=x+1$'},'FontSize',opt.fontSize-2,
'Interpreter','latex','Location','SouthEast');
clear h

subplot(2,3,4), hold on
h(1)=plot([-100 100],[0 0],'r-','LineWidth',opt.lineWidth);
h(2)=plot([-100 100],[1 1],'b-','LineWidth',opt.lineWidth);
%"subsample" x for the sake of visualization
tmpx = [-2:1:2];
```

```
diffx = (tmpx+1) - tmpx;
tmpy = ones(size(tmpx));
quiver(tmpx,0*tmpy,diffx,tmpy,0,'g-','LineWidth',opt.lineWidth);

set(gca,'XLim',[-4 4]);
set(gca,'YLim',[-0.25 1.25]);
set(gca,'XAxisLocation','origin');
set(gca,'YAxisLocation','origin');
set(gca,'FontSize',opt.fontSize)
clear h







subplot(2,3,2), hold on
h(1)=plot(x,x,'r-','LineWidth',opt.lineWidth);
h(2)=plot(x,2*x,'b-','LineWidth',opt.lineWidth);

daspect([1 1 1])
set(gca,'XLim',[-2 2]);
set(gca,'YLim',[-2 2]);
set(gca,'XAxisLocation','origin');
set(gca,'YAxisLocation','origin');
xlabel('X','FontSize',opt.fontSize, 'Interpreter','latex');
ylabel('$\hat{X}$','FontSize',opt.fontSize, 'Interpreter','latex');
title('Multiplication scales','FontSize',opt.fontSize, 'Interpreter','latex');
set(gca,'FontSize',opt.fontSize)
box on, grid on
legend(h,{'$\hat{x}=x$','$\hat{x}=2x$'},'FontSize',opt.fontSize-2,
'Interpreter','latex','Location','SouthEast');
clear h


subplot(2,3,5), hold on
h(1)=plot([-100 100],[0 0],'r-','LineWidth',opt.lineWidth);
h(2)=plot([-100 100],[1 1],'b-','LineWidth',opt.lineWidth);
%"subsample" x for the sake of visualization
tmpx = [-2:1:2];
diffx = (2*tmpx) - tmpx;
tmpy = ones(size(tmpx));
quiver(tmpx,0*tmpy,diffx,tmpy,0,'g-','LineWidth',opt.lineWidth);

set(gca,'XLim',[-4 4]);
set(gca,'YLim',[-0.25 1.25]);
set(gca,'XAxisLocation','origin');
set(gca,'YAxisLocation','origin');
set(gca,'FontSize',opt.fontSize)
clear h



subplot(2,3,3), hold on
```

```
h(1)=plot(x,x,'r-','LineWidth',opt.lineWidth);
h(2)=plot(x,x.^2,'b-','LineWidth',opt.lineWidth);

daspect([1 1 1])
set(gca,'XLim',[-2 2]);
set(gca,'YLim',[-2 2]);
set(gca,'XAxisLocation','origin');
set(gca,'YAxisLocation','origin');
xlabel('X','FontSize',opt.fontSize, 'Interpreter','latex');
ylabel('$\hat{X}$','FontSize',opt.fontSize, 'Interpreter','latex');
title('Exponentiation bends','FontSize',opt.fontSize, 'Interpreter','latex');
set(gca,'FontSize',opt.fontSize)
box on, grid on
legend(h,{'$\hat{x}=x$','$\hat{x}=x^2$'},'FontSize',opt.fontSize-2,
'Interpreter','latex','Location','SouthEast');
clear h



subplot(2,3,6), hold on
h(1)=plot([-100 100],[0 0],'r-','LineWidth',opt.lineWidth);
h(2)=plot([-100 100],[1 1],'b-','LineWidth',opt.lineWidth);
%"subsample" x for the sake of visualization
tmpx = [-2:1:2];
diffx = (tmpx.^2) - tmpx;
tmpy = ones(size(tmpx));
quiver(tmpx,0*tmpy,diffx,tmpy,0,'g-','LineWidth',opt.lineWidth);

set(gca,'XLim',[-4 4]);
set(gca,'YLim',[-0.25 1.25]);
set(gca,'XAxisLocation','origin');
set(gca,'YAxisLocation','origin');
set(gca,'FontSize',opt.fontSize)
clear h





if opt.saveFigures
    mySaveFig(gcf,[opt.destinationDir
'MatricesAsTransformations0015_EffectOfMajorOperations']);
end
```





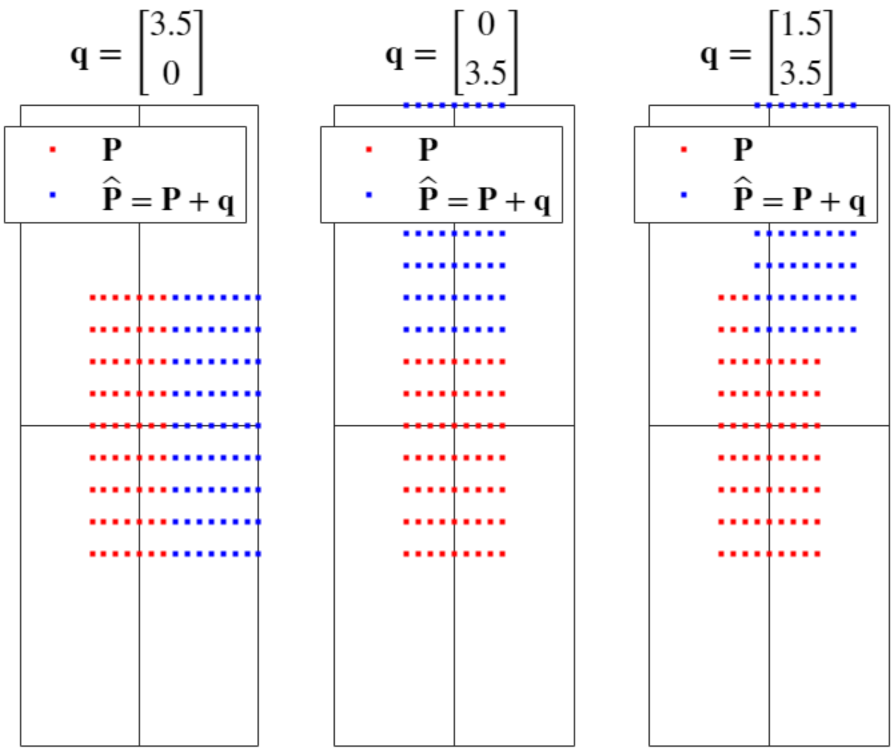
Translating or shifting points is therefore not naturally a multiplication but an addition. So if we want to translate our point cloud, we can simply add a vector $\mathbf{q} = \begin{bmatrix} q^1 \\ q^2 \end{bmatrix}$, e.g.

$$\hat{\mathbf{P}} = \mathbf{P} + \mathbf{q}$$

For instance, for shifting along the abcissa axis we may use $\mathbf{q} = \begin{bmatrix} a \\ 0 \end{bmatrix}$, for shifting along the ordinate axis we may use $\mathbf{q} = \begin{bmatrix} 0 \\ a \end{bmatrix}$, or for a general translation over $\mathbb{R}^2$, we may use $\mathbf{q} = \begin{bmatrix} a \\ b \end{bmatrix}$

```
figure
subplot(1,3,1), hold on
q = [3.5; 0];
hatP_shiftedRight = P + q;
h(1) = plot(P(1,:),P(2,:),'LineWidth',opt.lineWidth,'Color','r',...
        'Marker','.','LineStyle','none','MarkerFaceColor','r','MarkerSize',6);
h(2) =
plot(hatP_shiftedRight(1,:),hatP_shiftedRight(2,:),'LineWidth',opt.lineWidth,'Col
or','b',...
        'Marker','.','LineStyle','none','MarkerFaceColor','b','MarkerSize',6);

set(gca,'XLim',[-5 5]);
set(gca,'YLim',[-5 5]);
set(gca,'XAxisLocation','origin');
set(gca,'YAxisLocation','origin');
title('$\mathbf{q} = \left[\begin{array}{c}3.5\\0\end{array}\right]
$','FontSize',opt.fontSize, 'Interpreter','latex');
set(gca,'FontSize',opt.fontSize)
box on, grid on
legend(h,{'$\mathbf{P}$','$\hat{\mathbf{P}}=\mathbf{P}+\mathbf{q}
$'},'FontSize',opt.fontSize, 'Interpreter','latex');
clear h

subplot(1,3,2), hold on
q = [0; 3.5];
hatP_shiftedUp = P + q;
h(1) = plot(P(1,:),P(2,:),'LineWidth',opt.lineWidth,'Color','r',...
        'Marker','.','LineStyle','none','MarkerFaceColor','r','MarkerSize',6);
h(2) =
plot(hatP_shiftedUp(1,:),hatP_shiftedUp(2,:),'LineWidth',opt.lineWidth,'Color','b
',...
        'Marker','.','LineStyle','none','MarkerFaceColor','b','MarkerSize',6);

set(gca,'XLim',[-5 5]);
set(gca,'YLim',[-5 5]);
set(gca,'XAxisLocation','origin');
set(gca,'YAxisLocation','origin');
title('$\mathbf{q} = \left[\begin{array}{c}0\\3.5\end{array}\right]
$','FontSize',opt.fontSize, 'Interpreter','latex');
set(gca,'FontSize',opt.fontSize)
box on, grid on
legend(h,{'$\mathbf{P}$','$\hat{\mathbf{P}}=\mathbf{P}+\mathbf{q}
$'},'FontSize',opt.fontSize, 'Interpreter','latex');
clear h

subplot(1,3,3), hold on
q = [1.5; 3.5];
hatP_shiftedGeneral = P + q;
h(1) = plot(P(1,:),P(2,:),'LineWidth',opt.lineWidth,'Color','r',...
        'Marker','.','LineStyle','none','MarkerFaceColor','r','MarkerSize',6);
h(2) =
plot(hatP_shiftedGeneral(1,:),hatP_shiftedGeneral(2,:),'LineWidth',opt.lineWidth,
'Color','b',...
        'Marker','.','LineStyle','none','MarkerFaceColor','b','MarkerSize',6);

set(gca,'XLim',[-5 5]);
set(gca,'YLim',[-5 5]);
set(gca,'XAxisLocation','origin');
set(gca,'YAxisLocation','origin');
title('$\mathbf{q} = \left[\begin{array}{c}1.5\\3.5\end{array}\right]
$','FontSize',opt.fontSize, 'Interpreter','latex');
set(gca,'FontSize',opt.fontSize)
box on, grid on
legend(h,{'$\mathbf{P}$','$\hat{\mathbf{P}}=\mathbf{P}+\mathbf{q}
$'},'FontSize',opt.fontSize, 'Interpreter','latex');
clear h
```

```
if opt.saveFigures
    mySaveFig(gcf,[opt.destinationDir
'MatricesAsTransformations0016_EffectOfShifting_PointCloud']);
end
```



But correct as this may be, it is not convenient for concatenating operations (formally *composing* operations), e.g.

$f_n \circ f_{n-1} \circ \ldots f_2 \circ f_1(\mathbf{P}) = A_n A_{n-1} \ldots A_2 A_1(\mathbf{P})$

In this sense, it would be ideal if we could mimick shifting (addition) but by using scaling (multiplication). Therefore, recalling the second question above, **Is there a matrix transformation that can achieve translation?** We already anticipated above that the answer was yes, but the question pending to be answer was **How?**.

For this we need to do a trick; instead of thinking of the operation at hand as a translation in $\mathbb{R}^n$, you can think of it as a shearing in $\mathbb{R}^{n+1}$!

Sorry, what??? This:

First, project your data to a higher dimensional space by adding an additional dummy coordinate of 1 (neutral element of the product), e.g.

$$\mathbf{P}_{\mathbb{R}^{n+1}} = \begin{bmatrix} \mathbf{P}_{\mathbb{R}^n} \\ 1 \end{bmatrix}$$

Now shear the cloud of point along the new dimension;

$$\widehat{\mathbf{P}}_{\mathbb{R}^{n+1}} = A\mathbf{P}_{\mathbb{R}^{n+1}} \qquad A = \begin{pmatrix} I_n & \mathbf{q} \\ 0 & 1 \end{pmatrix}$$

And finally project back $\mathbb{R}^n$ by dropping the dummy dimension;

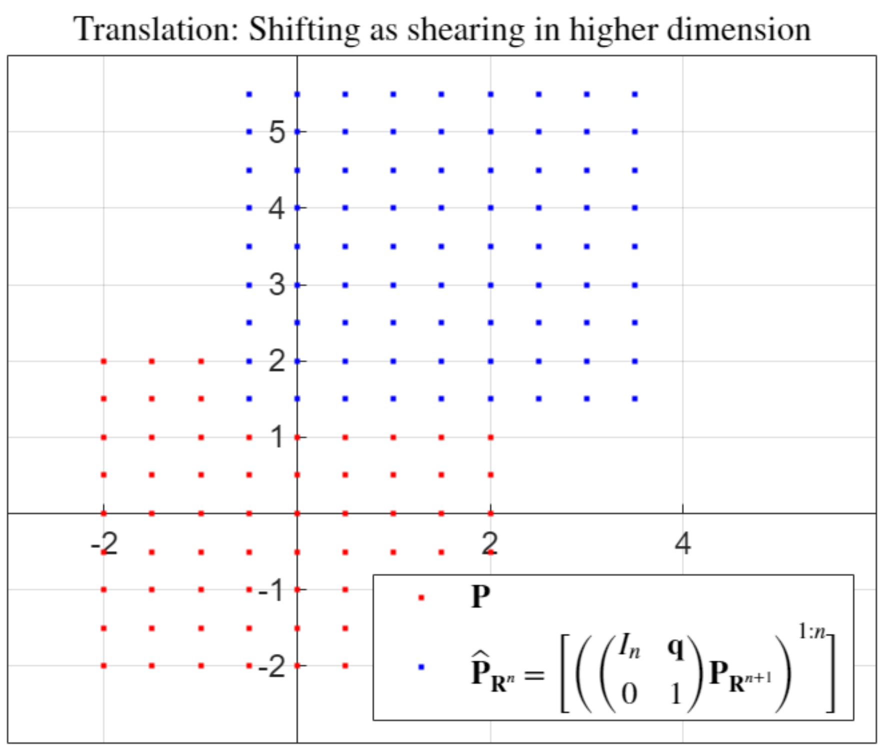
$$\widehat{\mathbf{P}}_{\mathbb{R}^n} = \left[\widehat{\mathbf{P}}^{1:n}_{\mathbb{R}^{n+1}} \right]$$

For instance, for shifting by $\mathbf{q} = \begin{bmatrix} 1.5 \\ 3.5 \end{bmatrix}$, we can use:

$$A = \begin{pmatrix} 1 & 0 & 1.5 \\ 0 & 1 & 3.5 \\ 0 & 0 & 1 \end{pmatrix}$$

```
figure, hold on
q = [1.5; 3.5];
A = [eye(2) q; 0 0 1];
hatP_shiftedByProjectingAndShearing = A*[P;ones(1,size(P,2))] ;
h(1) = plot(P(1,:),P(2,:),'LineWidth',opt.lineWidth,'Color','r',...
        'Marker','.','LineStyle','none','MarkerFaceColor','r','MarkerSize',6);
h(2) =
plot(hatP_shiftedByProjectingAndShearing(1,:),hatP_shiftedByProjectingAndShearin
g(2,:),'LineWidth',opt.lineWidth,'Color','b',...
        'Marker','.','LineStyle','none','MarkerFaceColor','b','MarkerSize',6);

set(gca,'XLim',[-3 6]);
set(gca,'YLim',[-3 6]);
set(gca,'XAxisLocation','origin');
set(gca,'YAxisLocation','origin');
title('Translation: Shifting as shearing in higher
dimension','FontSize',opt.fontSize, 'Interpreter','latex');
set(gca,'FontSize',opt.fontSize)
box on, grid on
legend(h,{'$\mathbf{P}$',['$\hat{\mathbf{P}}_{\mathbf{R}^{n}}='...
        '\left[\begin{array}{c}'...
        '\left(\left(\begin{array}{cc}'...
        'I_n & \mathbf{q} \\'...
        '0 & 1'...
        '\end{array}'...
        '\right)\mathbf{P}_{\mathbf{R}^{n+1}}\right)^{1:n}'...
        '\end{array}\right]$']},'FontSize',opt.fontSize,
'Interpreter','latex','Location','SouthEast');
clear h

if opt.saveFigures
    mySaveFig(gcf,[opt.destinationDir
'MatricesAsTransformations0017_ShiftingAsShearingInHigherDimension']);
```

```
  end
```



Translation: Shifting as shearing in higher dimension
5
4
3
2
1
-1
-2
-2
2
4
$\mathbf{P}$
$\widehat{\mathbf{P}}_{\mathbf{R}^n} = \left[\left(\begin{pmatrix} I_n & \mathbf{q} \\ 0 & 1 \end{pmatrix}\mathbf{P}_{\mathbf{R}^{n+1}}\right)^{1:n}\right]$