Kalman Filter Equations Fx versus Bu

Consider this specific Kalman filter equation: $\mathbf{x' = Fx + Bu}$
This equation is the move function that updates your beliefs in between sensor measurements. Fx models motion based on velocity, acceleration, angular velocity, etc of the object you are tracking.
B is called the control matrix and u is the control vector. Bu measures extra forces on the object you are tracking. An example would be if a robot was receiving direct commands to move in a specific direction, and you knew what those commands were and when they occurred. Like if you told your robot to move backwards 10 feet, you could model this with the Bu term.
When you take the self-driving car engineer nanodegree, you'll use Kalman filters to track objects that are moving around your vehicle like other cars, pedestrians, bicyclists, etc. In those cases, you would ignore

Bu

Bu because you do not have control over the movement of other objects. The Kalman filter equation becomes $\mathbf{x' = Fx}$

Representing State with Matrices

The State Vector

You just learned how to represent a self-driving car's state using a motion model. It turns out that matrices provide a very convenient and compact form for representing a vehicle's state.
Let's go back to the constant velocity motion model:

$distance = velocity \times time$

The vehicle's state is represented by the two variables distance and velocity. If you were going to store these two variables in Python, you'd probably use a list like this:

```
state = [distance, velocity]
```

That Python code looks a lot like a mathematical concept called a vector. A vector is essentially a list where each element in the list contains some information.
You could imagine that a state vector could have even more information. In a two-dimensional world, the state could have a

Distance_x Distance_y Velocity_x and a Velocity_y

In Python, the list would look like this:

```
state = [distance_x, distance_y, velocity_x, velocity_y]
```

Or an even more complex model could include information about the turning angle of the vehicle and the turning rate:

```
state = [distance_x, distance_y, velocity_x, velocity_y, angle,
angle_rate]
```

State Vector in a One-Dimensional World

For now, consider the one-dimensional model with just distance and velocity.

```
state = [distance, velocity]
```

How did you calculate the distance and velocity of the vehicle over time when the velocity was constant? There were two different equations.

$$distance = velocity \times time$$

$$velocity = velocity$$

Now, think about these equations in terms of state. For convenience, you can represent distance as x, velocity as v, and time as t.

Initial State

When the vehicle first starts moving, you can consider that $t = t_0$, $x = x_0$ and $v = v_0$. So the state vector is

$$state_0 = [x_0, v_0]$$

First Time Step

What about after a certain amount of time has passed and now you are at a time t_1? At t_1, the state vector is:

$$state_1 = [x_1, v_1]$$

According to the model formulas,

$$x_1 = x_0 + v_0 * (t_1 - t_0)$$

and since velocity is constant:

$$v_1 = v_0$$

Second Time Step

Then after the next time step t_2, the state vector is:

$$state_2 = [x_2, v_2]$$

where

$$x_2 = x_1 + v_0 * (t_2 - t_1)$$

and since velocity is constant

$$v_2 = v_0$$

A Better Way

The math so far is not too hard, right? You have a distance equation and a velocity equation. You plug in the previous state into each equation, the time lapse, and you get the new velocity and the new distance.

But imagine what will happen as your self-driving car model gets more complex. What happens when you have to take into account an x-direction, y-direction, x and y velocities, and steering angle and angular velocity? Or what about an even more complex model like a drone or helicopter that also has a z-direction?

Instead of updating your equations one by one, you can actually use vectors and matrices to do all of the calculations in just one step.

Updating State with Matrix Algebra

If you look back at the equation that updates the distance, you'll notice that distance depends on the previous distance, the initial velocity, and how much time has elapsed since the distance formula was updated. You end up with a generic function:

$$x_{t+1} = x_t + v_0 * (t_{t+1} - t_t)$$

You can also write

$t_{t+1} - t_t$ As: Δt

For a constant velocity model, the generic velocity equation becomes:

$$t + 1 = v_t$$

How could you combine the x and v equations into one matrix algebra expression? The matrix algebra would look like this:

$$\begin{bmatrix} x_{t+1} \\ v_{t+1} \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} x_t \\ v_t \end{bmatrix}$$

Don't worry if you're not sure what this expression means or how to multiply these matrices. You will learn how in this lesson.

Notation

The matrix algebra equation you just saw is actually one part of the Kalman filter update equation.

Traditionally, the matrix operation:

$$\begin{bmatrix} x_{t+1} \\ v_{t+1} \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} x_t \\ v_t \end{bmatrix}$$

is represented by this notation for Kalman filters:

$$\mathbf{\hat{x}_{k|k-1}} = \mathbf{F}\mathbf{\hat{x}_{k-1|k-1}}$$

where

$\mathbf{\hat{x}}$ is the state vector and $\mathbf{F}$

$$\begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}$$

$\mathbf{k-1}$ is the previous step and $\mathbf{k}$ is the current step.

You will see in the next part of the lesson why the notation contains $\mathbf{k-1|k-1}$ and $\mathbf{k|k-1}$.

This notation can get a little bit confusing. For example, what is the difference between x and $\mathbf{\hat{x}}$?

The regular x would usually represent distance along the x-axis; on the other hand, the bold $\mathbf{\hat{x}}$ indicates a vector. In the one-dimensional case being discussed here, the $\mathbf{\hat{x}}$ vector contains two variables: distance along the x-axis and velocity; hence

$$\mathbf{\hat{x}} = \begin{bmatrix} x \\ v \end{bmatrix}$$

Why is there a capitalized bold $\mathbf{F}$ instead of $\mathbf{f}$? The capitalized, bold $\mathbf{F}$ tells you that this variable is a matrix.

Kalman Equation Reference

We're just including this here in case you want to refer back to the Kalman Filter equations at any time. Feel free to move along :)

Variable Definitions

$\hat{\mathbf{x}}$ - state vector
$\mathbf{F}$ - state transition matrix
$\mathbf{P}$ - error covariance matrix
$\mathbf{Q}$ - process noise covariance matrix
$\mathbf{R}$ - measurement noise covariance matrix
$\mathbf{S}$ - intermediate matrix for calculating Kalman gain
$\mathbf{H}$ - observation matrix
$\mathbf{K}$ - Kalman gain
$\tilde{\mathbf{y}}$ - difference between predicted state and measured state
$\mathbf{z}$ - measurement vector (lidar data or radar data, etc.)
$\mathbf{I}$ - Identity matrix

Prediction Step Equations

PREDICT STATE VECTOR AND ERROR COVARIANCE MATRIX

$$\hat{\mathbf{x}}_{\mathbf{k|k-1}} = \mathbf{F_k}\hat{\mathbf{x}}_{\mathbf{k-1|k-1}} \; \mathbf{P_{k|k-1}} = \mathbf{F_k}\mathbf{P_{k-1|k-1}}\mathbf{F_k^T} + \mathbf{Q_k}$$

Update Step Equations

KALMAN GAIN

$$\mathbf{S_k} = \mathbf{H_k}\mathbf{P_{k|k-1}}\mathbf{H_k^T} + \mathbf{R_k}$$
$$\mathbf{K_k} = \mathbf{P_{k|k-1}}\mathbf{H_k^T}\mathbf{S_k}^{-1}$$

UPDATE STATE VECTOR AND ERROR COVARIANCE MATRIX

$$\tilde{\mathbf{y_k}} = \mathbf{z_k} - \mathbf{H_k}\hat{\mathbf{x}}_{\mathbf{k|k-1}}$$
$$\hat{\mathbf{x}}_{\mathbf{k|k}} = \hat{\mathbf{x}}_{\mathbf{k|k-1}} + \mathbf{K_k}\tilde{\mathbf{y_k}}$$
$$\mathbf{P_{k|k}} = (\mathbf{I} - \mathbf{K_k}\mathbf{H_k})\mathbf{P_{k|k-1}}$$

What is a vector? Physics versus Computer Programming

You might have learned at some point that a vector is a measurement or quantity that has both a **magnitude** and a **direction**. Examples might be distance along a y-axis or velocity towards the north-west.

But in computer programming, when we say "vector" we are just referring a **list of values**.

These two ways of thinking about vectors are actually deeply related, but for this Nanodegree we're going to look at vectors from a computer science point of view.

Vectors, Motion Models and Kalman Filters in Self-Driving Cars

In a physics class, you might have one vector for position and then a separate vector for velocity. But in computer programming, a vector is really just a list of values.

When using the Kalman filter equations, the bold, lower-case variable $\mathbf{x}$ represents a vector in the computer programming sense of the word. The $\mathbf{x}$ vector holds the values that represent your motion model such as position and velocity.

In a basic motion model, the vector $\mathbf{x}$ will contain information about position and linear velocity like:

$$\mathbf{x} = [x, y, v_x, v_y].$$

In a physics class, these might be represented with two different vectors: a position vector and a velocity vector.

A more complex motion model might take into account yaw rate, which considers a rotational angle and angular velocity around the center of the vehicle like

$$\mathbf{x} = [x, y, v_x, v_y, \psi, \dot{\psi}]$$

So in order to use the Kalman filter equations and execute object tracking, you have to be familiar with vectors and how to write programs with them.

Transpose of a Matrix

There were a few Kalman filter equations that required the transpose of a matrix. You can identify these matrices because they have a T superscript. For example the transpose of matrix $\mathbf{A}$ is written $\mathbf{A^T}$. These were the three equations that contained the transpose of a matrix:

$$\mathbf{P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k}$$

$$\mathbf{S_k = H_k P_{k|k-1} H_k^T + R_k}$$

$$\mathbf{K_k = P_{k|k-1} H_k^T S_k}^{-1}$$

What exactly is the transpose?

You can think of the tranpose as switching rows and columns. The matrix rows become the columns or alternatively you can consider the columns become the rows.

Here is an example. If you start with this matrix,

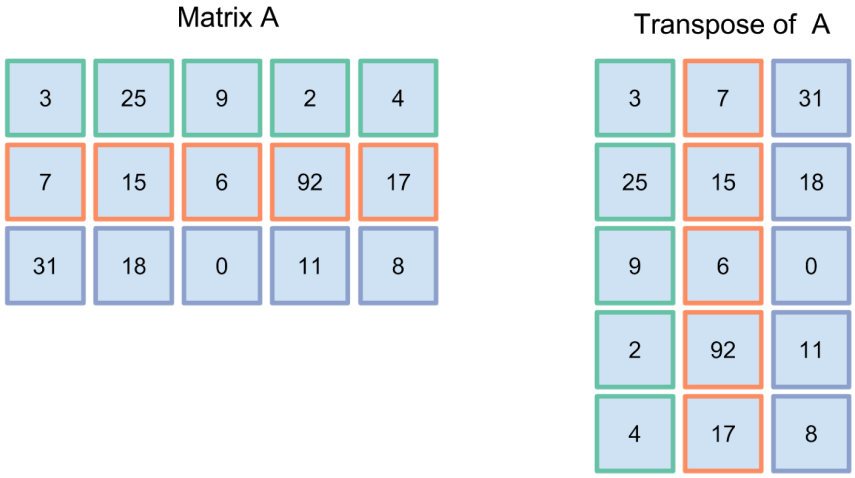
$$\begin{bmatrix} 3 & 25 & 9 & 2 & 4 \\ 7 & 15 & 6 & 92 & 17 \\ 31 & 18 & 0 & 11 & 8 \end{bmatrix}$$

the transpose would be

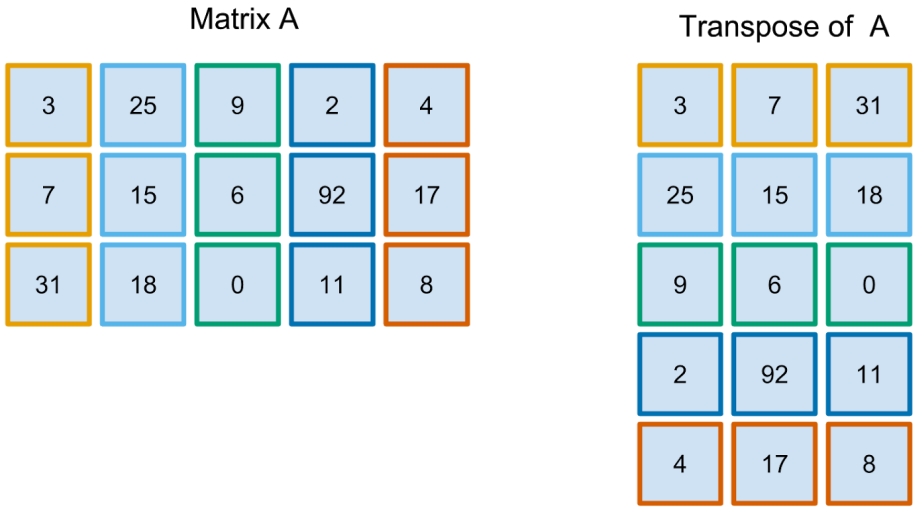
$$\begin{bmatrix} 3 & 7 & 31 \\ 25 & 15 & 18 \\ 9 & 6 & 0 \\ 2 & 92 & 11 \\ 4 & 17 & 8 \end{bmatrix}$$

The original matrix was size 3x5. The transpose is 5x3.

To get a better understanding of what is happening, this image is color coded to match values from the original matrix and the transpose of the matrix. If you think of switching the rows and making them into columns, the matrix operation looks like this:



But you could also think of transposing the columns into rows:



Mathematically, you are switching around the i and j values for every element in the matrix. For example, the element in the 3rd row, 4th column is 11. For the transpose of the matrix, 11 is now in the 4th row, 3rd column. The formal mathematical definition of the transpose of a matrix is

$$[\mathbf{A}^\mathbf{T}]_{ij} = [\mathbf{A}]_{ji}$$

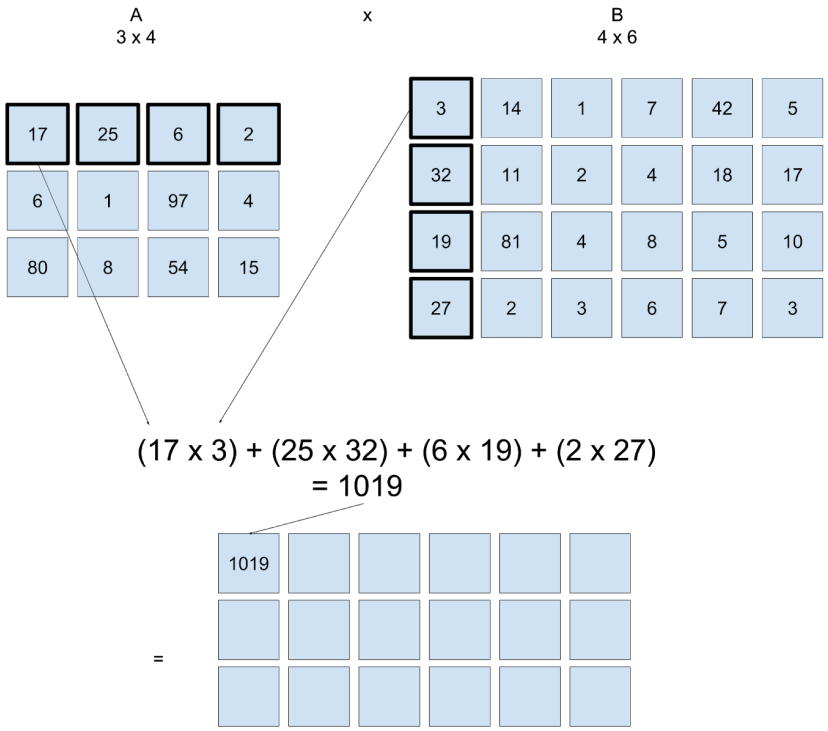
Motivation for Calculating the Transpose

In order to use the Kalman filter equations, you need to calculate the transpose of both the

$\mathbf{F}$ and $\mathbf{H}$ matrices.

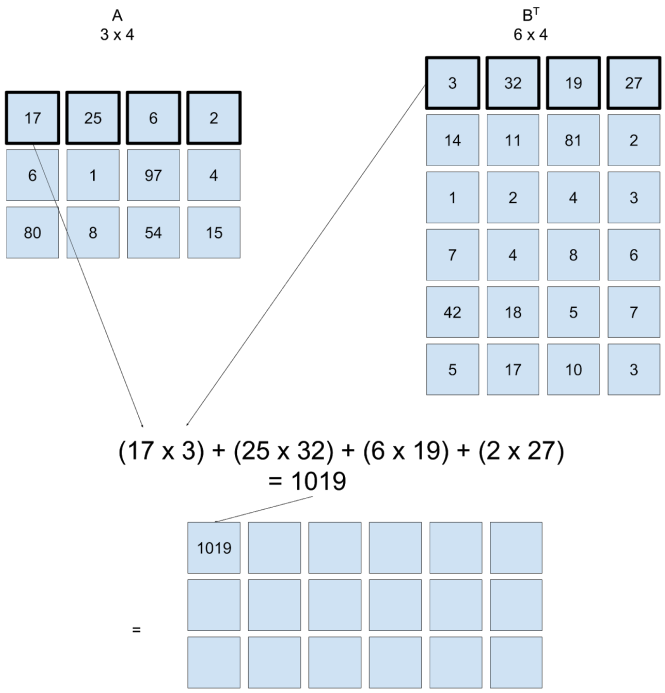
But there is also another place where you could use a matrix transposition: when carrying out **matrix multiplication**. In the previous exercises for matrix multiplication,

you wrote a function that returned a column of a matrix. You needed matrix columns in order to find the dot product of a row from matrix **A** and a column from matrix **B**. Here is a reminder of what that looked like:



But what happens if you take the transpose of matrix **B**? All of the columns in **B** become rows. Your matrix multiplication function then involves finding the dot product

between rows of $\mathbf{A}$ and rows of $\mathbf{B^T}$



Dot product between rows of A and transpose of matrix B

You are not calculating the product of AB^T. Instead, you are taking advantage of matrix transposition to make matrix multiplication easier to code.

In the previous coding exercises, the get_column function you built to change a matrix column into a horizontal vector was essentially a transpose.

In the coding exercises for this part of the lesson, you will not only write a transpose function but also write a new multiplication function that takes advantage of the matrix transpose.

Coding the Transpose of a Matrix

This is a similar problem to what you have already seen; you'll need to use nested for loops. But what exactly will this nested for loop look like?

You could iterate through the matrix like you've done already with the rows in the outer loop and the columns in the inner loop:

```
for i in range(len(matrixA)):
    for j in range(len(matrixA[0])):
        print(matrixA[i][j])
```

The transpose of the matrix would need to store each i, j element inside a new matrix with position j, i. But how are you going to populate this new matrix? You would probably first need to create an n x m list within a list and populate this nested list with empty values. That sounds complicated.

Is there a more efficient way to code matrix transposition? Think about how you could start iterating through the columns in the outside for loop and the rows on the inside for loop.

Go to the next part of the lesson to write your code.

The Identity Matrix

The identity matrix is a special matrix in linear algebra that shows up in quite a few applications. For the purposes of this lesson, gaining insight into the identity matrix will help you understand matrix inversion. The identity matrix is represented by the symbol $\mathbf{I}$. $\mathbf{I}$ is an n x n square matrix with 1 across the main diagonal and 0 for all other elements.

For a 1x1 matrix, the identity matrix looks like this:

$$\begin{bmatrix} 1 \end{bmatrix}$$

A 2x2 identity matrix looks like this:

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

The 3x3 identity matrix is the following:

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

The 4x4 identity matrix is:

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

and so on.

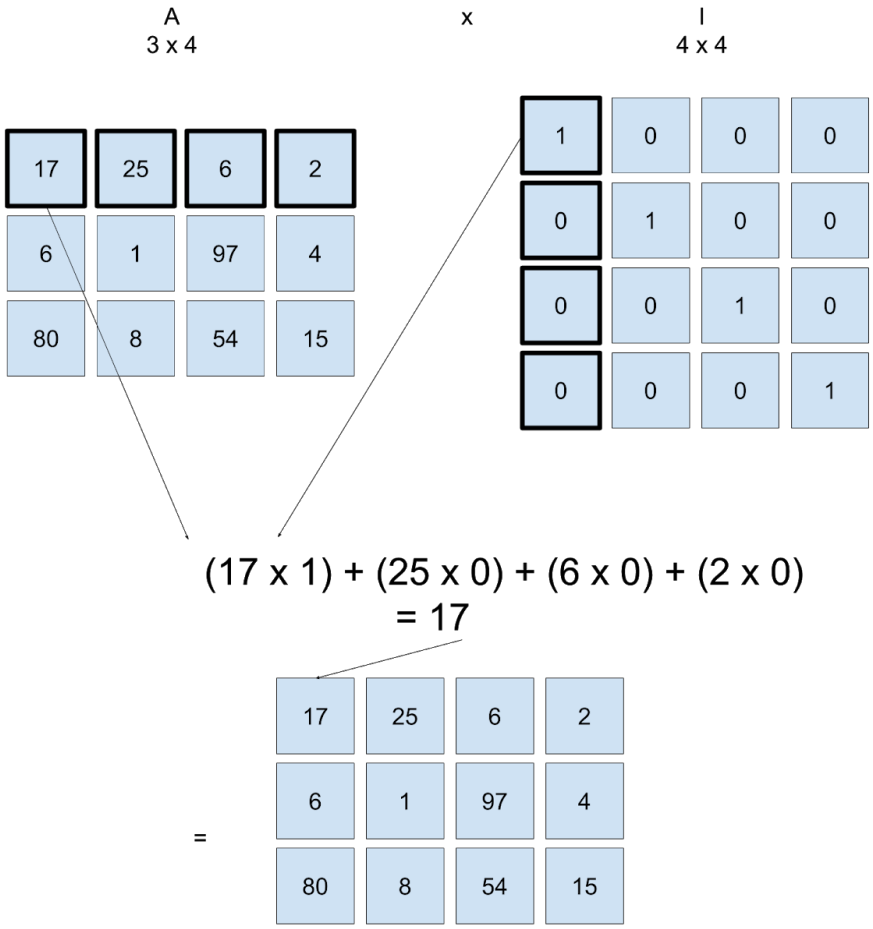
Identity Matrix is like the Number One

In scalar multiplication, the number one has a special property: $1 \times a = a$. Likewise, $a \times 1 = a$.

It turns out the Identity matrix has the same property:

$\mathbf{AI} = \mathbf{IA} = \mathbf{A}$. And although the identity matrix is always square, matrix $\mathbf{A}$ does not have to be square.

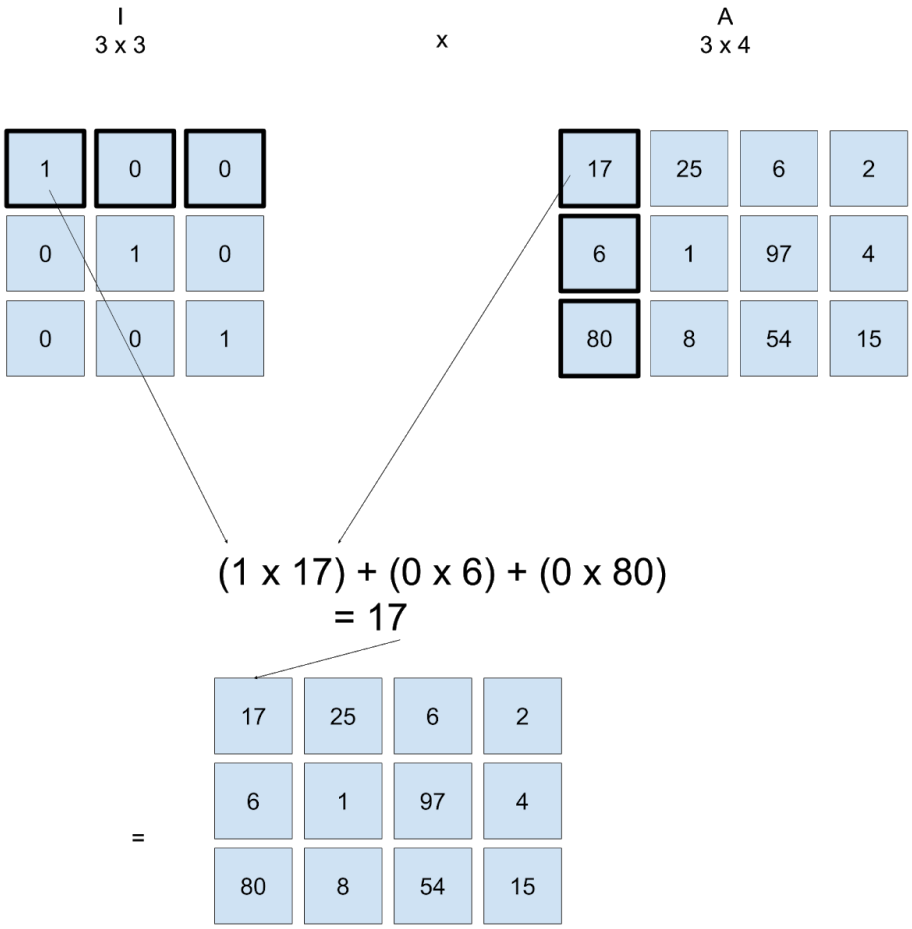
Here is an example of multiplying $\mathbf{AI}$



What about the other case of multiplying $\mathbf{IA}$? You'll need to take into account the dimensions of the I and A matrix so that the multiplication works.

How do you figure out the size of $\mathbf{I}$?

The output matrix has to be 3x4 just like $\mathbf{A}$. So a 3x3 matrix multiplied by a 3x4 matrix will give a 3x4 matrix.



Initializing an Identity Matrix with Code

In the coding exercise, you will write a function that receives a size and outputs the identity matrix for that size.

Think about how you might go about coding this starting from an empty Python list. The ones will always be at indicies

- [0][0]
- [1][1]
- [2][2]
- [3][3]
- etc.

Everywhere else in the matrix will be zero. So you will need to not only use nested for loops but also if else statements.

Matrix Inverse

There is one more matrix operation that you will need in order to use the Kalman Filter equations: the inverse of a matrix.

Specifically when calculating the Kalman filter gain matrix $\mathbf{K}$, you will need to take the inverse of the $\mathbf{S}$ matrix. The superscript -1 represents the inverse of a matrix. Here is a reminder of the Kalman Filter gain equations where you can see the need for the inverse of S.

$$\mathbf{S_k} = \mathbf{H_k P_{k|k-1} H_k^T} + \mathbf{R_k}$$

$$\mathbf{K_k} = \mathbf{P_{k|k-1} H_k^T S_k}^{-1}$$

Formal Definition of Inverse of a Matrix

As mentioned, the inverse of a matrix $\mathbf{A}$ would be denoted by $\mathbf{A^{-1}}$

Formally, if matrix $\mathbf{A}$ has an inverse, then

$$\mathbf{A \times A^{-1} = A^{-1} \times A = I},$$

where $\mathbf{I}$ is an identity matrix.

Only square matrices, or in other words matrices with the same number of columns as rows, have inverses. You can see that this must be true based on the definition of the inverse and the identity matrix. The identity matrix is always a square matrix, so if $\mathbf{A}$ is m x n, then $\mathbf{A^{-1}}$ has to be n x m to get a square identity matrix of m x m. Multiplying $\mathbf{A^{-1}A}$ gives (n x m)(n x m) = n x m, which is not a square matrix unless n = m.

So in order for a matrix to have an inverse, the matrix must be square. At the same time, not all square matrices have inverses.

Relationship between a Scalar Inverse and a Matrix Inverse

In scalar math, the inverse of a number x is $1/x$.

If you multiply a scalar by its inverse, you get 1:

$$x \times \frac{1}{x} = 1$$

In linear algebra, the inverse of a matrix is analogous to the scalar inverse:

$$\mathbf{A \times A^{-1} = I}$$

As you saw in the previous part of the lesson, the identity matrix has a similar role to the number 1.

Calculating the Inverse of a Matrix

For the purposes of this class, you will learn to calculate the inverse of a 1x1 matrix and a 2x2 matrix.

For matrices with more dimensions, the calculations become more complicated. Both Python and C++ have libraries that can calculate the inverse of a matrix such as the NumPy Library and the Eigen Library.

Inverse of a 1 x 1 matrix

For a 1×1 matrix with a single element with value a, the inverse is simply

$$\frac{1}{a}$$

So the inverse of

$$\begin{bmatrix} 25 \end{bmatrix}$$

is

$\left[\frac{1}{25}\right]$

Inverse of a 2 x 2 matrix

Say you have a matrix

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

The inverse of this 2 x 2 matrix is

$$\frac{1}{ad-bc}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

And you can see that if *ad=bc*, then the matrix does not have an inverse.

Another formula for 2x2 inverse matrix

Here is a more formal formula for the 2x2 inverse matrix.

$$\mathbf{A}^{-1} = \frac{1}{\det\ \mathbf{A}}\left[(\mathrm{tr}\ \mathbf{A})\,\mathbf{I} - \mathbf{A}\right]$$

Where $\det\ \mathbf{A}$ is called the determinant of a matrix. For a 2x2 matrix, the determinant is ad - bc. $\mathrm{tr}\ \mathbf{A}$ is called the trace of a matrix. The trace is the sum across the major diagonal, which in this case would be a + d.

If you multiply everything, you end up with the same equation already presented, namely:

$$\mathbf{A^{-1}} = \frac{1}{ad-bc}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

Inverse of a 3 x 3 or larger matrix

Calculating the inverse of a larger matrix involves relatively complex matrix algebra theory. In this course, you will only need to calculate the inverse of a 2 x 2 matrix.

Coding the Inverse of a Matrix

You are going to write a function that calculates the inverse of a matrix. Remember that you will need to check the size of the matrix because a 1x1 matrix and a 2x2 matrix have different formulas.