Data Science

Yogesh Kulkarni

Linear Algebra

What's in a name?

- "Algebra" means, roughly, "relationship", between unknown numbers.
- Without knowing x and y, we can still work out that $(x+y)^2 = x^2 + 2xy + y^2$
- "Linear Algebra" means, roughly, "line-like relationships".
- Meaning, not curve like, ie quadratic, cubic, sinusoidal, etc, right?
- Nothing in "power" term!!
- An operation F is linear if scaling inputs scales the output, and adding inputs adds the outputs:

$$F(ax) = a.F(x)$$
$$F(x+y) = F(x) + F(y)$$

- When plotted, its a line!!

(Ref: An Intuitive Guide to Linear Algebra - Better Explained)

Linear Equations

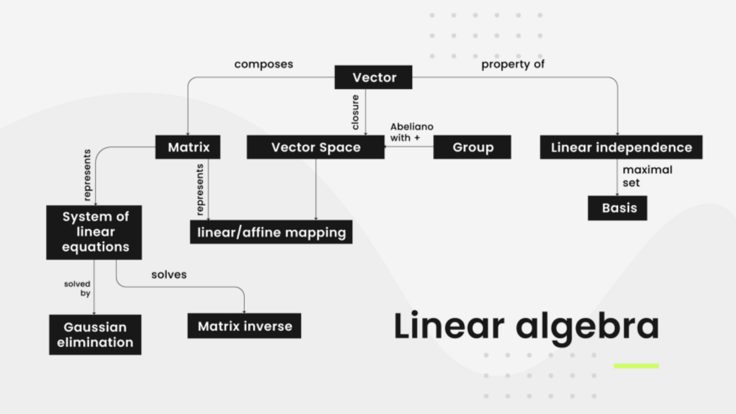
- $F(x, y, z) = 3x + 4y + 5z$
- Whats an example for this?
- Can you represent this by multiplication of two vectors?

(Ref: An Intuitive Guide to Linear Algebra - Better Explained)

Basic Entities

- Scalars?
- Vectors?
- Matrices?
- Next? (or Whats this called collectively?)
- Point in n-dimensional space is represented by?

Landscape: Linear Algebra



(Ref: The NOT definitive guide to learning math for machine learning - Favio Vazquez)

Vectors

Vectors

- At its simplest, a vector is an entity that has both magnitude and direction.
- The magnitude represents a distance (for example, "2 miles") and the direction indicates which way the vector is headed (for example,"East").
- One more way is $\bar{v} = 2\hat{i} + 3\hat{j}$; meaning?
- Is Magnitude-Direction form equivalent to i-j form?
- Inter-convertible? How?
- Can it have just two components?

Vectors

Two-dimensional example:

- A vector that is defined by a point in a two-dimensional plane
- A two dimensional coordinate consists of an x and a y value, and in this case we'll use 2 for x and 1 for y
- Its is written in matrix form as : $\vec{v} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$
- Describes the movements required to get to the end point (of head) of the vector
- So, it is not a point in space. It gives Direction, like a movement recipe.
- When added to a point, results into a transformed point.
- In this case, we need to move 2 units in the x dimension, and 1 unit in the y dimension

Vectors

Two-dimensional example:

- Note that we don't specify a starting point for the vector
- We're simply describing a destination coordinate that encapsulate the magnitude and direction of the vector.
- Think about it as the directions you need to follow to get to **there** from **here**, without specifying where **here** actually is!
- Generally using the point 0,0 as the starting point (or origin). Also called as Position Vector.
- Our vector of (2,1) is shown as an arrow that starts at 0,0 and moves 2 units along the x axis (to the right) and 1 unit along the y axis (up).

Vectors

Calculating Magnitude

- $\|\vec{v}\| = \sqrt{v_1{}^2 + v_2{}^2}$
- Double-bars are often used to avoid confusion with absolute values.
- Note that the components of the vector are indicated by subscript indices $(v_1, v_2, \ldots v_n)$
- In this case, the vector v has two components with values 2 and 1, so our magnitude calculation is:
- $\|\vec{v}\| = \sqrt{2^2 + 1^2} = \sqrt{5} \approx 2.24$

Vectors

Calculating Direction

- We can get the angle of the vector by calculating the inverse tangent; sometimes known as the arctan
- For our v vector (2,1):$tan(\theta) = \frac{1}{2}$
- $\theta = tan^{-1}(0.5) \approx 26.57^o$
- use the following rules:
  - Both x and y are positive: Use the tan-1 value.
  - x is negative, y is positive: Add 180 to the tan-1 value.
  - Both x and y are negative: Add 180 to the tan-1 value.
  - x is positive, y is negative: Add 360 to the tan-1 value.

Vectors

- Vectors are defined by an n-dimensional coordinate that describe a point in space that can be connected by a line from an arbitrary origin.
- Are n-dimensional Points and Vectors equivalent? How?
- $\|\vec{v}\| = \sqrt{{v_1}^2 + {v_2}^2 ... + {v_n}^2}$

Definition A *vector* is a matrix with one column.

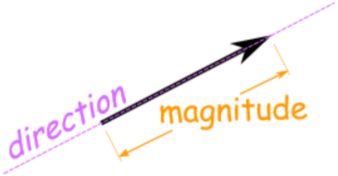
Example

$$\begin{bmatrix} 1 \\ 2 \\ -5 \\ 9 \end{bmatrix}$$

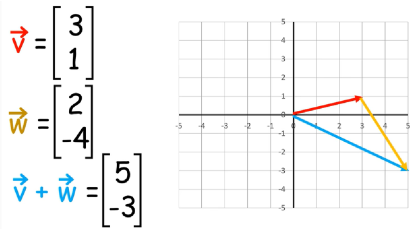
Note Two vectors are equal precisely when they have the same number of rows and all their corresponding entries are equal.

Vectors (Recap)

- A vector has magnitude (how long it is) and direction
- A point can be a vector (position vector, from Origin)
- A data row is a point in n-dimensions, thus a vector as well.



Vector Addition



- To add these vectors: We just add the individual components, so 3 plus 2 is 5; and 1 plus -4 is -3.
- It is simply adding another leg to the journey; so if we follow vector V along 3 and up 1, and then follow vector W along 2 and down 4, we end up at the head of the vector we calculated by adding V and W together.

Vector Addition

Definition We define the sum and of two vectors by

$$\begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} + \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} u_1 + v_1 \\ u_2 + v_2 \\ \vdots \\ u_n + v_n \end{bmatrix}$$

and the product of a scalar and a vector by

$$\alpha \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} = \begin{bmatrix} \alpha u_1 \\ \alpha u_2 \\ \vdots \\ \alpha u_n \end{bmatrix}$$

Example

$$\begin{bmatrix} 1 \\ 3 \\ -5 \end{bmatrix} + \begin{bmatrix} 2 \\ 2 \\ 7 \end{bmatrix} = \begin{bmatrix} 3 \\ 5 \\ 2 \end{bmatrix} \qquad \text{and} \qquad 3\begin{bmatrix} 5 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 15 \\ 6 \\ 3 \end{bmatrix}$$

Exercise

Let $\vec{u}$ and $\vec{v}$ be given by

$$\vec{u} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \qquad \text{and} \qquad \vec{v} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

Plot $\vec{u}$, $\vec{v}$, $2\vec{u}$ and $\vec{u}+\vec{v}$.
Parallelogram rule for vector addition Suppose $\vec{u}$ and $\vec{v} \in \mathbb{R}^2$. Then $\vec{u}+\vec{v}$ corresponds to the fourth vertex of the parallelogram whose opposite vertex is $\vec{0}$ and whose other two vertices are $\vec{u}$ and $\vec{v}$.

Exercise

Let $\vec{u} = \begin{bmatrix} 6 \\ 3 \end{bmatrix}$ and $\vec{v} = \begin{bmatrix} 5 \\ 2 \end{bmatrix}$. Display $\vec{u}$, $-2/3\vec{u}$, $\vec{v}$ and $-2/3\vec{u} + \vec{v}$ on a graph.

$\mathbb{R}^n$

In general we will consider vectors in $\mathbb{R}^n$, that is, having n real entries.

$$\vec{u} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} \in \mathbb{R}^n$$

The zero vector is $\vec{0} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$ having n entries, each equal to 0.

Properties of R^n

Theorem Suppose that $\vec{u}, \vec{v}, \vec{w} \in \mathbb{R}^n$ and $c, d \in \mathbb{R}$. Then,

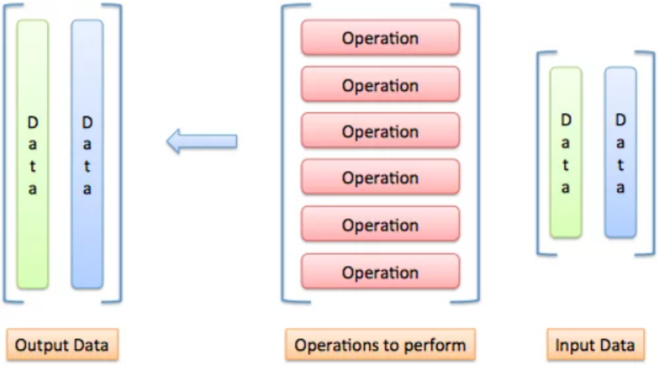
- $\vec{u} + \vec{v} = \vec{v} + \vec{u}$.
- $(\vec{u} + \vec{v}) + \vec{w} = \vec{u} + (\vec{v} + \vec{w})$
- $\vec{u} + \vec{0} = \vec{0} + \vec{u} = \vec{u}$
- $\vec{u} + -\vec{u} = -\vec{u} + \vec{u} = \vec{0}$ $\quad (-\vec{u} = (-1)\vec{u})$
- $c(\vec{u} + \vec{v}) = c\vec{u} + c\vec{v}$
- $(c + d)\vec{u} = c\vec{u} + d\vec{u}$
- $c(d\vec{u}) = (cd)\vec{u}$
- $1 \cdot \vec{u} = \vec{u}$

Matrix

Meaning of a Matrix

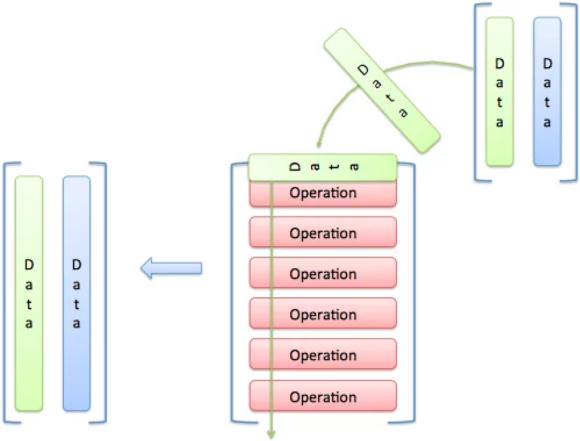
- Matrix is organization of data into rows and columns
- Example: columns can be various aspects of a person, such as height,weight, salary, etc, where as rows can represent different persons
- This Excel sheet like data can be thought of as a Matrix (especially in Data Science, Machine Learning)

Visualizing The Matrix



(Ref: An Intuitive Guide to Linear Algebra - Better Explained)

Visualizing The Matrix Application



(Ref: An Intuitive Guide to Linear Algebra - Better Explained)

Geometric Applications

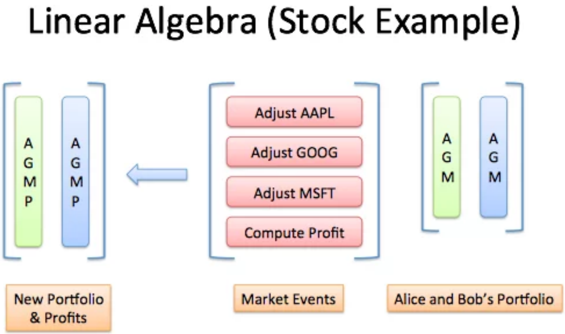
- Scale: make all inputs bigger/smaller
- Skew: make certain inputs bigger/smaller
- Flip: make inputs negative
- Rotate: make new coordinates based on old ones (East becomes North, North becomes West, etc.)

These are geometric interpretations of multiplication, and how to warp a vector space.

(Ref: An Intuitive Guide to Linear Algebra - Better Explained)

Non-Vector Applications

- Input data: stock portfolios with dollars in Apple, Google and Microsoft stock
- Operations: the changes in company values after a news event
- Output: updated portfolios



(Ref: An Intuitive Guide to Linear Algebra - Better Explained)

Solving Simultaneous equations

$$x + 2y + 3z = 3$$
$$2x + 3y + 1z = -10$$
$$5x + -y + 2z = 14$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 5 & -1 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 3 \\ -10 \\ 14 \end{bmatrix}$$

You can solve by ...? Some ... Elimination?

(Ref: An Intuitive Guide to Linear Algebra - Better Explained)

Matrix

A matrix is an array of numbers that can be arranged into rows and columns. We generally name matrices with a capital letter.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

```
 import numpy as np

A = np.array([[1,2,3],
              [4,5,6]])
print (A)

[[1 2 3]
 [4 5 6]]
```

Matrix

Definition A matrix with m rows and n columns is referred to as an $m \times n$ matrix. The number of rows always comes before the number of columns.

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \end{bmatrix}$$

```
import numpy as np

M = np.matrix([[1,2,3],
               [4,5,6]])
print (M)

[[1 2 3]
 [4 5 6]]
```

Matrix Addition

You can add or subtract matrices of the same size by simply adding or subtracting the corresponding elements in the two matrices.

$$A = \begin{bmatrix} 3 & 5 & 1 \\ 1 & 4 & 3 \end{bmatrix} \quad B = \begin{bmatrix} 2 & -2 & 4 \\ -1 & 3 & 1 \end{bmatrix}$$

$$A + B = \begin{bmatrix} 5 & 3 & 5 \\ 0 & 7 & 4 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 7 & 7 & 7 \\ 7 & 7 & 7 \end{bmatrix}$$

Matrix Addition

```
import numpy as np

A = np.array([[1,2,3],
              [4,5,6]])
B = np.array([[6,5,4],
              [3,2,1]])
print(A + B)

[[7 7 7]
 [7 7 7]]
```

Matrix Subtraction

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} - \begin{bmatrix} 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix} = \begin{bmatrix} -5 & -3 & -1 \\ 1 & 3 & 5 \end{bmatrix}$$

```
import numpy as np

A = np.array([[1,2,3],
              [4,5,6]])
B = np.array([[6,5,4],
              [3,2,1]])
print(A - B)

[[-5 -3 -1]
 [ 1  3  5]]
```

The Transpose of a Matrix

Definition The transpose of a $m \times n$ matrix A is the matrix A^T having (i, j)-entry a_{ji}. That is,

$$(A^T)_{ij} = a_{ji}.$$

Example For example, $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ has transpose $A^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$.

Note The rows of A become the columns of A^T and vice versa.

Meaning of a Matrix Multiplication

- Matrix is organization of data into rows and columns
- Example: columns can be various aspects of a person, such as height,weight, salary, etc, where as rows can represent different persons
- This Excel sheet like data can be thought of as a Matrix (especially in Data Science, Machine Learning)
- If you have another matrix like this, what is the meaning of their multiplication?
- Geometrically: say first matrix represents points of a shape, a polygon, where each row is a point, and each column represents X, Y, Z coordinates.
- Second matrix is typically a Homogeneous transformation matrix, such as rotation, when multiplied gets rotated shape.

Matrix Multiplication Rules

Theorem Let A and B be matrices whose sizes are appropriate for the following sums and products to be defined

- $(A^T)^T = A$
- $(A+B)^T = A^T + B^T$.
- For any scalar r, $(rA)^T = rA^T$.
- $(AB)^T = B^T A^T$

Example

$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, and $B = \begin{bmatrix} 5 & 1 & -1 \\ 1 & 2 & 2 \end{bmatrix}$ then

$$AB = \begin{bmatrix} 7 & 5 & 3 \\ 9 & 11 & 5 \end{bmatrix} \qquad (AB)^T = \begin{bmatrix} 7 & 9 \\ 5 & 11 \\ 3 & 5 \end{bmatrix} = B^T A^T$$

but A^T is 2×2 and B^T is 3×2, so $A^T B^T$ isn't even defined.

Matrix Transpose

Exchange rows and columns

$$A = \begin{bmatrix} 3 & 5 & 1 \\ 1 & 4 & 3 \end{bmatrix}$$

$$A^T = \begin{bmatrix} 3 & 1 \\ 5 & 4 \\ 1 & 3 \end{bmatrix}$$

Matrix Transpose

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

```
import numpy as np

A = np.array([[1,2,3],
              [4,5,6]])
print(A.T)

[[1 4]
 [2 5]
 [3 6]]
```

Matrix Multiplication

Here are the cases to consider:

- Scalar multiplication, which is multiplying a matrix by a single number
- Element wise matrix multiplication (rarely used, called Hadamard multiplication, shown with circle instead of dot)
- Dot product matrix multiplication, or multiplying a matrix by another matrix.

Matrix Scalar Multiplication

To multiply a matrix by a scalar value, you just multiply each element by the scalar to produce a new matrix:

$$2 \times \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 2 & 4 & 6 \\ 8 & 10 & 12 \end{bmatrix}$$

```
import numpy as np
import numpy as np

A = np.array([[1,2,3],
              [4,5,6]])
print(2 * A)

[[ 2  4  6]
 [ 8 10 12]]
```

Matrix Multiplication Defined

Definition If A is an $m \times n$ matrix, and if $B = [\vec{b}_1, \vec{b}_2 \ldots, \vec{b}_p]$ is a $n \times p$ matrix, then the matrix product AB is the following $m \times p$ matrix.

$$AB = [A\vec{b}_1 \quad A\vec{b}_2 \quad \ldots \quad A\vec{b}_p]$$

Example Let $A = \begin{bmatrix} 1 & 2 \\ -2 & 3 \end{bmatrix}$ and let $B = \begin{bmatrix} 3 & -1 & 6 \\ 7 & 5 & 3 \end{bmatrix}$. Compute AB.

Multiplying Matrices

Row-Column Rule If A is $m \times n$ and if B is $n \times p$ the (i, j)-entry of AB is given by

$$(AB)_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Note $\mathrm{Row}_i(AB) = \mathrm{Row}_i(A) \cdot B$.

Matrix Operations

Additions

- Commutative: $A + B = B + A$
- Associative: $A + (B + C) = (A + B) + C$

Multiplication

- Scalar : sA: multiplying all elements by s
- Commutative: $AB \neq BA$
- Associative: $A(BC) = (AB)C$
- Distributive: $A(B + C) = AB + AC$
- Identity: $I_m A_{mn} = A_{mn} I_n = A$

Linear Algebra with Python

(Ref: Linear Algebra and Python Basics - Rob Hicks)

Python Libraries

For numerical computing, useful libraries are:

- sympy: provides for symbolic computation (solving algebra problems)
- numpy: provides for linear algebra computations
- matplotlib.pyplot: provides for the ability to graph functions and draw figures
- scipy: scientific python provides a plethora of capabilities
- seaborn: makes matplotlib figures even pretties (another library like this is called bokeh).

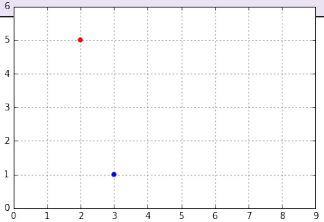
Vectors and Lists

To create a vector simply surround a python list ($[1, 2, 3]$) with the *np.array* function:

```
x_vector = np.array([1,2,3])
print(x_vector)

[1 2 3]

c_list = [1,2]
print("The list:",c_list)
print("Has length:", len(c_list))

c_vector = np.array(c_list)
print("The vector:", c_vector)
print("Has shape:",c_vector.shape)

The list: [1, 2]
Has length: 2
The vector: [1 2]
Has shape: (2,)
```

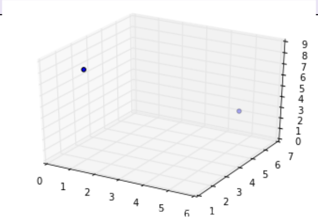
2D Vectors

```
u = np.array([2, 5])
v = np.array([3, 1])

x_coords, y_coords = zip(u, v)
plt.scatter(x_coords, y_coords, color=["r","b"])
plt.axis([0, 9, 0, 6])
plt.grid()
plt.show()
```



3D Vectors

```python
a = np.array([1, 2, 8])
b = np.array([5, 6, 3])

from mpl_toolkits.mplot3d import Axes3D

subplot3d = plt.subplot(111, projection='3d')
x_coords, y_coords, z_coords = zip(a,b)
subplot3d.scatter(x_coords, y_coords, z_coords)
subplot3d.set_zlim3d([0, 9])
plt.show()
```

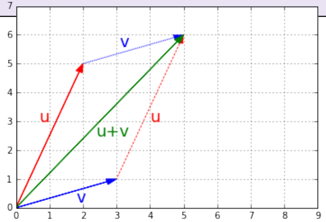


vector Norm

```
def vector_norm(vector):
    squares = [element**2 for element in vector]
    return sum(squares)**0.5

print(vector_norm(u))

5.385164807134504

import numpy.linalg as LA
print(LA.norm(u))

5.385164807134504
```

Vector Addition

```
print(u + v)

array([5, 6])

plot_vector2d(u, color="r")
plot_vector2d(v, color="b")
plot_vector2d(v, origin=u, color="b", linestyle="dotted")
plot_vector2d(u, origin=v, color="r", linestyle="dotted")
plot_vector2d(u+v, color="g")
plt.grid()
plt.show()
```



Matrices

```
b = list(zip(z,c_vector))
print(b)
print("Note that the length of our zipped list is 2 not (2 by 2):",len(b))

[(5, 1), (6, 2)]
Note that the length of our zipped list is 2 not (2 by 2): 2

D = np.matrix([[1.,2], [3,4], [5,6]])

matrix([[ 1.,  2.],
        [ 3.,  4.],
        [ 5.,  6.]])

E = = np.matrix("1.,2; 3,4; 5,6")

matrix([[ 1.,  2.],
        [ 3.,  4.],
        [ 5.,  6.]])
```

Matrices

```
F = np.ones((4,3))

array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])

np.rank(F)

2
```

Matrix Addition and Subtraction

Adding or subtracting a scalar value to a matrix

$$A + 3 = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} + 3 = \begin{bmatrix} a_{11} + 3 & a_{12} + 3 \\ a_{21} + 3 & a_{22} + 3 \end{bmatrix} \tag{1}$$

```
result = A + 3 #or result = 3 + A
print( result)

[[8 4]
 [9 5]]
```

Matrix Addition and Subtraction

Adding or subtracting two matrices

$$A_{2\times 2} + B_{2\times 2} = \begin{bmatrix} a_{11} + b_{11} & a_{12} + b_{12} \\ a_{21} + b_{21} & a_{22} + b_{22} \end{bmatrix}_{2\times 2} \tag{2}$$

```
B = np.random.randn(2,2)
print( B)

[[-0.9959588   1.11897568]
 [ 0.96218881 -1.10783668]]

result = A + B
print(result)

array([[4.0040412 , 2.11897568],
       [6.96218881, 0.89216332]])
```

Matrix Multiplication

Multiplying a scalar value times a matrix

$$3 \times A = 3 \times \begin{bmatrix} a11 & a12\ a21 & a22 \end{bmatrix} = \begin{bmatrix} 3a_{11} & 3a_{12} \\ 3a_{21} & 3a_{22} \end{bmatrix} \tag{3}$$

```
A * 3

array([[15,  3],
       [18,  6]])
```

Matrix Multiplication

Multiplying two matricies

$$A_{3\times 2} \times C_{2\times 3} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix}_{3\times 2} \times \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \end{bmatrix}_{2\times 3} \tag{4}$$

$$= \begin{bmatrix} a_{11}c_{11} + a_{12}c_{21} & a_{11}c_{12} + a_{12}c_{22} & a_{11}c_{13} + a_{12}c_{23} \\ a_{21}c_{11} + a_{22}c_{21} & a_{21}c_{12} + a_{22}c_{22} & a_{21}c_{13} + a_{22}c_{23} \\ a_{31}c_{11} + a_{32}c_{21} & a_{31}c_{12} + a_{32}c_{22} & a_{31}c_{13} + a_{32}c_{23} \end{bmatrix}_{3\times 3} \tag{5}$$

```
A = np.arange(6).reshape((3,2))
C = np.random.randn(2,2)

print( A.dot(C)) # or print( np.dot(A,C))

[[-1.19691566  1.08128294]
 [-2.47040472  1.00586034]
 [-3.74389379  0.93043773]]
```

Matrix Division

A misnomer. To divide in a matrix algebra world we first need to invert the matrix. It is useful to consider the analog case in a scalar work. Suppose we want to divide the f by g. We could do this in two different ways:

$$\frac{f}{g} = f \times g^{-1}. \tag{6}$$

Inverting a Matrix

$$A^{-1} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}^{-1} = \frac{1}{a_{11}a_{22} - a_{12}a_{21}} \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix} \tag{7}$$

```
C_inverse = np.linalg.inv(C)
print( C_inverse)

[[-1.47386391 -1.52526704]
 [-1.63147935 -0.76355223]]
```

Matrix Transpose

$$A_{3\times 2} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix}_{3\times 2} \tag{8}$$

The transpose of A (denoted as A') is

$$A' = \begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{32} \end{bmatrix}_{2\times 3} \tag{9}$$

```
A = np.arange(6).reshape((3,2))
print( A)
print( A.T)

[[0 1]
 [2 3]
 [4 5]]
[[0 2 4]
 [1 3 5]]
```

Matrix Eigen Values and Vectors

- Represent the "axes" of the transformation.
- Consider spinning a globe: every location faces a new direction, except the poles.
- Along "eigenvector", when it's run through the matrix, its points do not rotate (may scale though). The eigenvalue is the scaling factor.

(Ref: https://en.wikipedia.org/wiki/File:Eigenvectors.gif)

```
from numpy.linalg import eig
A = np.array([[1,2],[3,4]])
eigen_val, eigen_vec = eig(A)

print(eigen_val)
array([-0.37228132,  5.37228132])

print(eigen_vec)
array([[-0.82456484, -0.41597356],
       [ 0.56576746, -0.90937671]])
```

Singular Value Decomposition

Any $m \times n$ matrix M can be decomposed into the dot product of three simple matrices:

- a rotation matrix U (an $m \times m$ orthogonal matrix)
- a scaling & projecting matrix Σ (an $m \times n$ diagonal matrix)
- and another rotation matrix VT (an $n \times n$ orthogonal matrix)

$M = U \cdot \Sigma \cdot V^T$

```
U, S_diag, V_T = LA.svd(F)

print(U)
array([[ 0.89442719, -0.4472136 ],
       [ 0.4472136 ,  0.89442719]])

print(S_diag)
array([ 2. ,  0.5])
```

Linear Algebra - Python Implementation

Vectors

- If we have data of people with 3 attributes
- Heights, weights, and ages
- Each data point is in 3-D space with (height, weight, age) basis.
- In python, we can use list

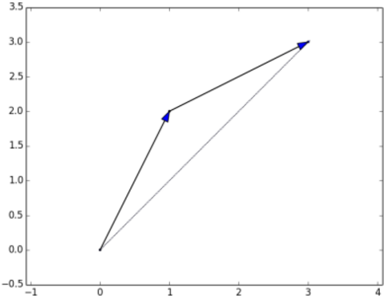
```
height_weight_age = [70,  # inches,
                      170, # pounds,
                      40 ] # years
```

Vectors

- lists are ok to represent vectors as storage, but not appropriate for operations
- Why?
- Whats list additions?
- Whats vector addition?

Vector Addition

- We'll frequently need to add two vectors.
- Vectors add component-wise.
- Resultant first element is $v[0] + w[0]$,
- Resultant second element is $v[1] + w[1]$, and so on.
- If they're not the same length, not allowed to add them.



Vector Addition

- Implement vector addition
- Hint: 'zip' two vectors and use List Comprehension

```python
def vector_add(v, w):
  :
  return [...]
vv = [ 1, 2,3]
ww = [3,2,1]
result = vector_add(vv,ww)
print("Vector Addition {}".format(result))
```

Vector Addition

Solution:

```
def vector_add(v, w):
    """adds corresponding elements"""
    return [v_i + w_i
            for v_i, w_i in zip(v, w)]
```

Vector Addition [4, 4, 4]

Vector Subtraction

- Implement vector subtraction
- Hint: its an addition with second vector negated

```
def vector_subtract(v, w):
  :
  return [...]
vv = [ 1, 2,3]
ww = [3,2,1]
result = vector_subtract(vv,ww)
print("Vector Subtraction {}".format(result))
```

Vector Subtraction

Solution:

```
def vector_subtract(v, w):
    """subtracts corresponding elements"""
    return [v_i - w_i
        for v_i, w_i in zip(v, w)]
```

Vector Subtraction [-2, 0, 2]

Vectors Summation

- Implement vector summation
- Component-wise sum a list of vectors
- Result is a new vector whose first element is the sum of all the first elements, and so on

```
def vector_sum(vectors):
  :
  return [...]
vecs = [[ 1, 2,3],[3,2,1],[3,2,-1]]
result = vector_sum(vecs)
print("Vectors Sum {}".format(result))
```

Vectors Summation

Solution:

```
def vector_sum(vectors):
    """sums all corresponding elements"""
    result = vectors[0]
    for vector in vectors[1:]:
        result = vector_add(result, vector)
    return result
```

Vectors Sum [7, 6, 3]

Scalar Multiplication

- Implement scalar multiplication of a vector
- Component-wise multiplication

```
def scalar_multiply(c, v):
  :
  return [...]
vv = [ 1, 2,3]
cc = 4
result =  scalar_multiply(cc,vv)
print("Scalar Multiply {}".format(result))
```

Scalar Multiplication

Solution:

```
def scalar_multiply(c, v):
    """c is a number, v is a vector"""
    return [c * v_i for v_i in v]
```

Scalar Multiply [4, 8, 12]

Vectors Mean

- Component-wise means of a list of (same-sized) vectors:
- Hint: use `vector_sum` to add all up, then use `scalar_multiply` to compute mean

```
def vector_mean(vectors):
  :
  return [...]
vecs = [[ 1, 2,3],[3,2,1],[3,2,-1]]
result = vector_mean(vecs)
print("Vectors Mean {}".format(result))
```

Vectors Mean

Solution:

```
def vector_mean(vectors):
    """compute the vector whose ith element is the mean of the ith elements of
     the input vectors"""
    n = len(vectors)
    return scalar_multiply(1/n, vector_sum(vectors))
```

Vectors Mean [2.333333333333333, 2.0, 1.0]

Vector Multiplication

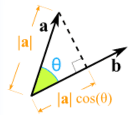
- Dot Product : Output? Meaning?
- Cross Product : Output? Meaning?

$$a.b = |a| \times |b| \times cos(\theta)$$

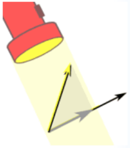
$$a \times b = |a| \times |b| \times sin(\theta)\hat{n}$$

Dot Product

- The Dot Product gives a number as an answer (a "scalar", not a vector).
- $a.b = a_x \times b_x + a_y \times b_y$
- So we multiply the x's, multiply the y's, then add.
- Cant multiply unless they are in same direction.
- So, one of them is projected over another using $cos(\theta)$



We take the component of **a** that lies alongside **b**



Like shining a light to see where the shadow lies

If vectors are at right angles? (Reference: https://www.mathsisfun.com/algebra/vectors-dot-product.html)

Dot Product

- The dot product of two vectors is the sum of their component-wise products.
- Hint: Similar to `vector_add` but with a difference in return type

```
def dot(v, w):
  :
  return ...
vv = [ 1, 2,3]
ww = [3,2,1]
result = dot(vv,ww)
print("Dot Product {}".format(result))
```

Dot Product

Solution:

```
def dot(v, w):
    """v_1 * w_1 + ... + v_n * w_n"""
    return sum(v_i * w_i
               for v_i, w_i in zip(v, w))
```

Dot Product 10:

Easy to compute a vector's sum of squares:

```
def sum_of_squares(v):
    """v_1 * v_1 + ... + v_n * v_n"""
    return dot(v, v)
```

Magnitude of a Vector

```python
import math
def magnitude(v):
    return math.sqrt(sum_of_squares(v))
```

Distance Between Vectors

- Formula: $\sqrt{(v_1 - w_1)^2 + \ldots (v_n - w_n)^2}$
- Hint: First use `vector_subtract` and then `sum_of_squares`
- For now, do not bother about normalizing it with product of their magnitudes.

```
def squared_distance(v, w):
  :
  return ...
vv = [ 1, 2,3]
ww = [3,2,1]
result = squared_distance(vv,ww)
print("Squared Distance {}".format(result))
```

Distance

```
def squared_distance(v, w):
    """(v_1 - w_1) ** 2 + ... + (v_n - w_n) ** 2"""
    return sum_of_squares(vector_subtract(v, w))

def distance(v, w):
   return math.sqrt(squared_distance(v, w))
```

or

```
def distance(v, w):
    return magnitude(vector_subtract(v, w))
```

Squared Distance 8

Matrices

- A matrix is a two-dimensional collection of numbers.
- list of lists, with each inner list having the same size and representing a row of the matrix.
- If A is a matrix, then $A[i][j]$ is the element in the ith row and the jth column.

```
A = [[1, 2, 3],  # A has 2 rows and 3 columns
     [4, 5, 6]]
B = [[1, 2],     # B has 3 rows and 2 columns
     [3, 4],
     [5, 6]]
```

Matrices

- Python lists, being '0' indexed, first row of a matrix "row 0" and the first column "column 0".
- matrix A has $len(A)$ rows and $len(A[0])$ columns, which we consider its shape

```
def shape(A):
    num_rows = len(A)
    num_cols = len(A[0]) if A else 0
    return num_rows, num_cols
```

Numpy and Pandas Dataframes have in built matrix functionality needed for Data Science

Linear Algebra Summary

(Ref: A Gentle Introduction to Linear Algebra - Json Brownlee)

Summary

- Linear algebra is about linear combinations.
- Linear algebra is the study of lines and planes, vector spaces and mappings that are required for linear transforms
- Applications of Linear Algebra
  - Matrices in Engineering, such as a line of springs.
  - Graphs and Networks, such as analyzing networks.
  - Computer Graphics, such as the various translation, rescaling and rotation of images.

Further Reading

Books:

- Introduction to Linear Algebra by Serge Lang
- Introduction to Linear Algebra, Gilbert Strang, 2016.
- Numerical Linear Algebra, Lloyd N. Trefethen, 1997.
- Linear Algebra and Matrix Analysis for Statistics, Sudipto Banerjee, Anindya Roy, 2014.

Courses:

- "Linear Algebra for machine learning" - Patrick van der Smagt
- "Machine Learning – 03. Linear Algebra Review". Playlist at Youtube
- Linear Algebra stream on Khan Academy.

Thanks . . .

- Feel free to follow me at:
  - Github (github.com/yogeshhk) for open-sourced Data Science training material, etc.
  - Kaggle (www.kaggle.com/yogeshkulkarni) for Data Science datasets and notebooks.
  - Medium (yogeshharibhaukulkarni.medium.com) and also my Publications:
    - Desi Stack https://medium.com/desi-stack
    - TL;DR,W,L https://medium.com/tl-dr-w-l
- Office Hours: Saturdays, 2 to 5pm (IST); Free-Open to all; email for appointment.
- Email: yogeshkulkarni at yahoo dot com