Week 1 Summary

> NAME: Mia Jerphagnon
>
> PID: A16821297

I certify that the following write-up is my own work, and have abided by the UCSD Academic Integrity Guidelines.

- [✓] Yes
- [ ] No

Key Takeaways from Week 1

The markdown cell is a versatile environment that allows for explanation, analysis, and math to be shown alongside code. Using markdown is intuitive, for example, when writing bullet points and lists. Markdown conveniently allows for LaTeX expressions, including those inline. From subscripts to sum functions, LaTeX has it all.

Vectors and matrices allow us to think about multidimensional space and data in a clean, mathematical way. Vectors are objects of both direction and magnitute. Matrices can represent the lefthand side of a system of equations. They are also functions that can input a vector and output another vector. You can also think of vectors as matrices, but with only one column.

Vectors and matrices can interact with each other algebraically. The dot product of two vectors is the sum of the element-wise products. A matrix can act on a vector which can be represented by matrix multiplication Av. In code, we can also write `A @ v`.

To solve a system of equations, you can initialize matrix A and vector b, and then use `numpy.linalg.solve(A, v)` in order to solve for x in $Ax = b$.

Thu, Apr 4th

Basics of markdown

Bullet Lists and Numbered Lists

- Main bullet point
    - Indented bullet point

1. First list item
2. Second list item

Math

This is an in-line math expression $x_1^2 + \omega = 2$.
These are outside math expressions:

$$\overline{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$$

$$\text{Var}(X) = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \overline{X})$$

Boxed & Colored Equations

Sample mean:

\color{blue} \boxed{\frac{1}{n}\sum_{i=1}^n X_i = \overline{X}}

Math Macros

$$\sum_{i=0}^{m} a_i$$

$$X_1, X_2, \cdots, X_n \text{is sampled iid from a distribution } H_0$$

Int, Float, Booleans, and Strings

In [1]:
```
print(int(7.1)) #convert float to integer
print(type(7)) #find the type
print((True, False)) #booleans start with capital letters
```

```
7
<class 'int'>
(True, False)
```

Arithmetic Operations (+, -, /, %, //, *)

In [2]:
```
print(13//2) #Integer division
print(13%4) #Modulo operator
print(13**4) #Exponential
```

```
6
1
28561
```

Arrays, Tuples, Dicts

In [3]:
```
# Arrays are dynamic
[1, 2, 3] + [4, 5, 6]
```

Out[3]:
```
[1, 2, 3, 4, 5, 6]
```

In [4]:
```
# Tuples have a fixed size
(1, 2, 3)
```

Out[4]:
```
(1, 2, 3)
```

In [5]:
```
# Dictionary
import math
for k, v in {"a":1, "b":2, "c":math.pi}.items():
    print(f'k={k}, v={v}')
```

```
k=a, v=1
k=b, v=2
k=c, v=3.141592653589793
```

In [6]:
```
# Set
a = {1, 2, 3, 4, 4, 5}
b = {4, 5, 7}
a.symmetric_difference(b)
```

Out[6]:
```
{1, 2, 3, 7}
```

Anonymous Functions

In [7]:
```
fib = lambda n: 0 if n==0 else 1 if n==1 else fib(n-1) + fib(n-2)
```

Vectors

A vector represents an object in multidimensional space with direction and magnitude. In other words, a vector $v \in \mathbb{R}^n$ represents a point

$$v = (v_1, v_2, \ldots, v_n)^\intercal$$

in n-dimensional space where each v_i is a number.

When you add two vectors you sum the elements in each position.

$$v + w = (v_1 + w_1, v_2 + w_2, \ldots, v_n + w_n)^\intercal$$

In [8]:
```
# %pip install numpy
import numpy as np
```

```
v = np.array([1, 2, 3])
w = np.array([1, 2, 3])
print(v + w)
```

```
[2 4 6]
```

When you multiply a vector by a scalar, you multiply the elements in each position by the scalar:

$$cv = (cv_1, cv_2 + cv_2, \dots, cv_n)^\intercal$$

In [9]:
```
v = np.array([1, 2, 3])
v*3
```

Out[9]:
```
array([3, 6, 9])
```

You can "multiply" two vectors in two different ways:

1. The **dot product** of two vectors $u, v \in \mathbb{R}^n$ is the sum of the element-wise product

   $$u \cdot v = \sum_{i=1}^{n} u_i v_i$$

   The geometric interpretation of the dot product is given by

   $$u \cdot v = \|u\| \|v\| \cos\Big(\angle\theta(u, v)\Big)$$

2. The **cross product** of two vectors $u, v \in \mathbb{R}^n$ the vector w which is perpendicular to the linear space *spanned* by u and v, i.e.,

   $$w \perp \operatorname{span}\{u, v\}$$

   and whose magnitude is

   $$\|w\| = \|u\| \|v\| \sin\big(\angle\theta(u, v)\big)$$

Matrices

Intuitively, a matrix is a way of expressing a system of equations where we take away the variables and put the entire system in square brackets. According to the professor, you can also think of a matrix as a function that takes a vector and produces another vector.

In [10]:
```
# Creating vectors and matrices
vector = np.array([1.0, -1.0])
matrix = np.array([
    [1.0, -1.0],
    [-1.0, 1.0],
    [-1.0, 1.0],
    ])
```

```
# Calculating the product
product = vector * matrix
print(product)
```

```
[[ 1.  1.]
 [-1. -1.]
 [-1. -1.]]
```

When a matrix acts on a vector, you get ...

In [11]:
```
# Action of a matrix on a vector
A = np.array([[1, -1], [-1, 1]])
v = np.array([2,3])
print(A @ v)

# Matrix-matrix product
print(A @ A)
```

```
[-1  1]
[[ 2 -2]
 [-2  2]]
```

Basic linear algebra operations:

1. Norm: Given a vector, its norm is a vector with the same direction, but a magnitute of 1.
2. Dot product: As explained earlier, it is the vector where each element is the sum of the elements in each position of the two vectors.
3. Matrix transpose: Given a matrix, rows become columns and vice versa.
4. Matrix inverse: Given a matrix A, its inverse is the matrix A^{-1} such that $AA^{-1} = I$.

Slightly more advanced linear algebra operations

If we're given a system of equations, for example,

$$\begin{aligned} 2x + 3y &= 5 \\ 4x + 5y &= 6 \end{aligned}$$

We can represent it in the matrix-vector form as

$$b = \begin{pmatrix} 5 \\ 6 \end{pmatrix} \quad A = \begin{pmatrix} 2 & 3 \\ 4 & 5 \end{pmatrix}$$

The solution for the unknown variables, (x, y) is given by

$$v = \begin{pmatrix} x \\ y \end{pmatrix}$$

In code, we can do this as follows: ...

In [12]:
```
# Solving a system of linear equations
A = np.array([
    [2.0, 3.0],
    [4.0, 5.0],
    ])
b = np.array([5, 6])
solution = np.linalg.solve(A,b)
print(solution)
```

```
[-3.5  4. ]
```

Eigenvalues and eigenvectors

Given an $n \times n$ matrix A, the eigenvalue of A is the scalar λ such that $Av = \lambda v$ where λ is a scalar.

The eigenvector of A is the vector v such that when A is applied to v, the result is a scalar multiple of v.

Intuitively, the eigenvector of a matrix describes a vector which is fixed in direction under a given linear transformation.

In code, we can compute the eigenvalues and eigenvectors of a matrix as follows:

In [13]:
```
# Eigenvalues
a = np.random.randn(2, 2)
A = a @ a.T

# Compute eigenvalues
eigenvalues, eigenvectors = np.linalg.eig(A)
print(f'Eigenvalues of A are \n{eigenvalues}\n\n')
print(f'Eigenvectors of A are \n{eigenvectors}\n\n')
```

```
Eigenvalues of A are
[0.05226596 2.7059382 ]


Eigenvectors of A are
[[-0.75006369  0.6613656 ]
 [-0.6613656  -0.75006369]]
```

We can verify that the eigenvalues and eigenvectors are correct by running the following code and ensuring that the output matches the expected values:

In [14]:
```
lambda1 = 4 #Largest eigenvalue
A = np.array([[2, 2], [1, 3]])
v1 = np.array([1, 1]) #Corresponding eigenvector

# Expected value of Av
```

```
expected_value = np.array([4, 4])

# Actual value of Av
Av = A @ v1

# Check if the expected value and actual value are equal
np.allclose(expected_value, Av)
```

```
Out[14]: True
```