# PROBLEM SET 1

You may also find the code here: https://github.com/helenxtian/math-for-ml.

1. Reading assignment. Read Boyd VMLS chapters 1 & 6.

2. **Matrix multiplication.**

   (a) The rows represent each of the years (e.g., 2021, 2022, 2023, and 2024). The columns represent the different rivers (e.g., A, B, and C).

$$\begin{bmatrix} 4 & 5 & 3 \\ 6 & 6 & 3 \\ 4 & 7 & 1 \\ 3 & 8 & 2 \end{bmatrix}$$

   (b) The total annual ecological impact score of all three rivers (with calculations) is shown below. Each of the resulting rows represent score per year of all three years (respective from 2021 to 2024).

$$\begin{bmatrix} 4 & 5 & 3 \\ 6 & 6 & 3 \\ 4 & 7 & 1 \\ 3 & 8 & 2 \end{bmatrix} \cdot \begin{bmatrix} 4 \\ 2 \\ 6 \end{bmatrix} = \begin{bmatrix} 4 \cdot 4 + 5 \cdot 2 + 3 \cdot 6 \\ 6 \cdot 4 + 6 \cdot 2 + 3 \cdot 6 \\ 4 \cdot 4 + 7 \cdot 2 + 1 \cdot 6 \\ 3 \cdot 4 + 8 \cdot 2 + 2 \cdot 6 \end{bmatrix} = \begin{bmatrix} 44 \\ 54 \\ 36 \\ 40 \end{bmatrix}$$

   (c) Using matrix multiplication, the 4-year weighted average concentration for each river is shown below. Each of the resulting rows represent the concentration per river (respective of order A, B, C).

$$\begin{bmatrix} 4 & 5 & 3 \\ 6 & 6 & 3 \\ 4 & 7 & 1 \\ 3 & 8 & 2 \end{bmatrix} \cdot \begin{bmatrix} 0.1 & 0.2 & 0.3 & 0.4 \end{bmatrix}^T = \begin{bmatrix} 4 & 5 & 3 \\ 6 & 6 & 3 \\ 4 & 7 & 1 \\ 3 & 8 & 2 \end{bmatrix}^T \cdot \begin{bmatrix} 0.1 \\ 0.2 \\ 0.3 \\ 0.4 \end{bmatrix}$$

$$= \begin{bmatrix} 4 & 6 & 4 & 3 \\ 5 & 6 & 7 & 7 \\ 3 & 3 & 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} 0.1 \\ 0.2 \\ 0.3 \\ 0.4 \end{bmatrix} = \begin{bmatrix} 4.0 \\ 7.0 \\ 2.0 \end{bmatrix}$$

   (d) Using the matrix we solved for in part 1b, we can calculate the weighted Total Annual Ecological Impact Score across all Years and Rivers. This is shown below.

$$\begin{bmatrix} 0.1 & 0.2 & 0.3 & 0.4 \end{bmatrix} \cdot \begin{bmatrix} 44 \\ 54 \\ 36 \\ 40 \end{bmatrix} = \begin{bmatrix} 42 \end{bmatrix}$$

   (e)

```python
import numpy as np

mat = np.array([
    [4, 5, 3],
    [6, 6, 3],
    [4, 7, 1],
    [3, 8, 2]
])
eco_index = np.array([[4], [2], [6]])
weights = np.array([0.1, 0.2, 0.3, 0.4])

print(mat)

annual_impact = np.dot(mat, eco_index)
print(annual_impact)

weighted_avg = np.dot(weights, mat)
print(weighted_avg)

weighted_total_impact = np.dot(weights,
    annual_impact)
print(weighted_total_impact)
```

```
helentian@Helens-MacBook-Air pset1_code % python3 q2
    .py
[[4 5 3]
 [6 6 3]
 [4 7 1]
 [3 8 2]]
[[44]
 [54]
 [36]
 [40]]
[4. 7. 2.]
[42.]
```

3. **Matrix dimensions.**

   (a) Yes, a solution exists. We see that all the dimensions of $cAv = y$ are satisfied with $\mathbf{v}^T$ having four rows to match $\mathbf{A}$'s four columns. This product then produces $\mathbf{y} \in \mathbb{R}^5$.

   (b) We put 1 in the $k$-th position and 0 elsewhere. Use the following dimensions and style to define $\mathbf{v}$ and then, remember to transpose it

when using $\mathbf{v}^T\mathbf{A}$. For example, when $k = 3$,

$$\mathbf{v} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

(c)

$$\mathbf{v} = \begin{bmatrix} 0 \\ 2 \\ 5 \\ 0 \\ -4 \end{bmatrix}$$

(d) We put 1 in the $k$-th position and 0 elsewhere. Use the following dimensions and style to define $\mathbf{v}$; we use it like $\mathbf{A}\mathbf{v}$. For example, when $k = 3$,

$$\mathbf{v} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

(e)

$$\mathbf{v} = \begin{bmatrix} -3 \\ -4 \\ 4 \\ 0 \end{bmatrix}$$

(f) For any $a, b \in \mathbb{R}$ and distinct $k, j \in \{1, 2, 3, 4\}$, to construct $\mathbf{v} \in \mathbb{R}^4$ such that

$$\mathbf{A}\mathbf{v} = a \cdot \text{col}_k(\mathbf{A}) + b \cdot \text{col}_j(\mathbf{A}),$$

define $\mathbf{v}$ as the following. Start with a 4-dimensional zero vector, and set the $k$-th entry of $\mathbf{v}$ to the value $a$. Then, set the $j$-th entry of $\mathbf{v}$ to the value $b$.

(g)

```
import numpy as np

A = np.array([
    [2, 1, 4, 0],
    [5, 3, 1, 7],
    [8, 0, 2, 6],
    [1, 9, 5, 3],
    [4, 2, 0, 8]
])
```

```
# a)
y = np.array([-2, 1, -1, 3, 0])
v_a = np.array([-1, -2, 2, 1])
Av = np.dot(A, v_a)
try:
    c = np.linalg.lstsq(Av.reshape(-1,1), y.reshape
        (-1,1), rcond=None)[0][0][0]
    residual = np.linalg.norm(c * Av - y)
    print(f"Solution exists with c = {c:.2f}")
except np.linalg.LinAlgError:
    print("No solution exists")

# b) trying k=3 row
v_b = np.array([0, 0, 1, 0, 0])
ans_b = np.dot(v_b, A)
print(ans_b)

# c)
v_c = np.array([0, 2, 5, 0, -4])
ans_c = np.dot(v_c, A)
print(ans_c)

# d) trying k=3 col
v_d = np.array([[0], [0], [1], [0]])
ans_d = np.dot(A, v_d)
print(ans_d)

# e)
v_e = np.array([-3, -4, 4, 0])
ans_e = np.dot(A, v_e)
print(ans_e)

# f)
a, b = 2, -1
k, j = 2, 0
v = np.zeros(4)
v[k] = a
v[j] = b
ans_f = np.dot(A, v)
print(ans_f)
```

```
helentian@Helens-MacBook-Air pset1_code % python3 q3
    .py
Solution exists with c = -0.50
[8 0 2 6]
[34 -2 12 12]
[[4]
 [1]
 [2]
```

```
 [5]
 [0]]
[  6 -23 -16 -19 -20]
[ 6. -3. -4.  9. -4.]
```

4. **Matrix Rank.** Given the rank of 2, we want to show that there are only two linearly independent columns. Let us denote the columns as:

$$\mathbf{c}_1 = \begin{bmatrix} 1 \\ -2 \\ 1 \\ 2 \end{bmatrix}, \quad \mathbf{c}_2 = \begin{bmatrix} 0 \\ 1 \\ 1 \\ -1 \end{bmatrix}, \quad \mathbf{c}_3 = \begin{bmatrix} 2 \\ -1 \\ 5 \\ 1 \end{bmatrix}$$

We will solve this problem as a linear combination of $\mathbf{c}_1$ and $\mathbf{c}_2$. Suppose

$$\alpha_1 \mathbf{c}_1 + \alpha_2 \mathbf{c}_2 = \mathbf{c}_3$$

. This gives the equation:

$$\alpha_1 \begin{bmatrix} 1 \\ -2 \\ 1 \\ 2 \end{bmatrix} + \alpha_2 \begin{bmatrix} 0 \\ 1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 2 \\ -1 \\ 5 \\ 1 \end{bmatrix}$$

From here, we can solve algebraically:

$\alpha_1 = 2$ (1st row)
$-2\alpha_1 + \alpha_2 = -1 \Rightarrow -2(2) + \alpha_2 = -1 \Rightarrow \alpha_2 = 3$ (2nd row)
(check, 3rd row): $\alpha_1 + \alpha_2 = 2 + 3 = 5$
(check, 4th row): $2\alpha_1 - \alpha_2 = 4 - 3 = 1$

Thus, the rank is 2.

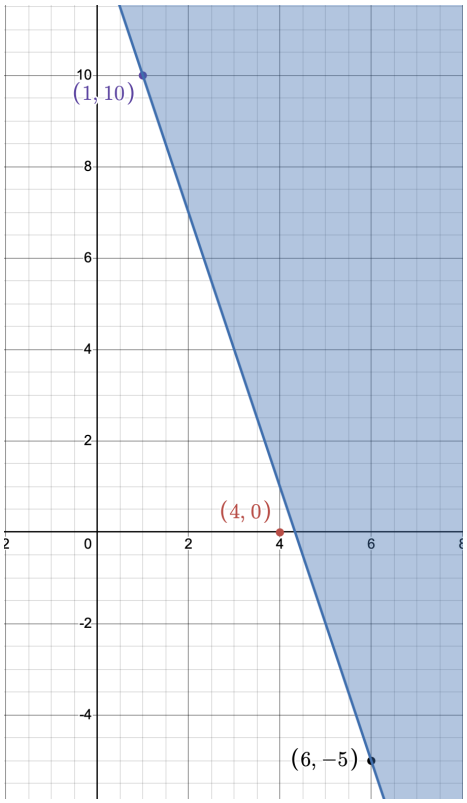
5. **Graph.** We can substitute the points we were given into the equations (note that the decision boundary means $= 0$ ).

$$w_1(1) + w_2(10) + w_3 = 0$$
$$w_1(6) + w_2(-5) + w_3 = 0$$

From here, we can solve $5w_1 - 15w_2 = 0 \Rightarrow w_1 = 3w_2$. And, then we can substitute it into the first equation such that $3w_2 + 10w_2 + w_3 = 0 \Rightarrow w_3 = -13w_2$. Thus, the general solution is

$$\mathbf{w} = (3w_2,\ w_2,\ -13w_2), \quad w_2 \neq 0$$

We can verify this with $\hat{y}$ for $(4, 0)$: $\hat{y} = 3(4) + 1(0) - 13 = -1 < 0 \Rightarrow$ label $= -1$ . Therefore, all weight vectors define the same decision boundary up to scaling: $\mathbf{w} = c \cdot (3,\ 1,\ -13)$.




6. **Polynomials using linear models.**

   (a) $p(z) = w_0 + w_1 z + w_2 z^2 + \cdots + w_d z^d$ where $w_0, w_1, \ldots, w_d$ are the coefficients of the polynomial.

   (b) The system of equations for $i = 1, \ldots, n$ can be written in matrix form as $Xw = y$, where

$$X = \begin{bmatrix} 1 & z_1 & z_1^2 & \cdots & z_1^d \\ 1 & z_2 & z_2^2 & \cdots & z_2^d \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & z_n & z_n^2 & \cdots & z_n^d \end{bmatrix}, \quad w = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

   (c)

```
import numpy as np
import matplotlib.pyplot as plt

# n = number of points
# z = points where polynomial is evaluated
# p = array to store the values of the interpolated
    polynomials
```

```
n = 100
z = np.linspace(-10, 10, n)

d = 3 # degree
rng = np.random.default_rng() # random number
    generator
w = rng.uniform(-1, 1, d+1)
X = np.zeros((n, d+1))

powers = np.arange(d+1)
X = z.reshape(-1,1) ** powers
p = np.dot(X, w)

plt.plot(z, p, linewidth=2)
plt.xlabel('z')
plt.ylabel('y')
plt.title('Polynomial with coefficients w = %s' % w)
plt.show()
```