2)

a) $X = \begin{array}{c} \\ \\ \\ \\ \end{array}\begin{array}{ccc} A & B & C \end{array}$

$$X = \begin{array}{cc} & \begin{array}{ccc} A & B & C \end{array} \\ & \begin{pmatrix} 2350 & 500 & 200 \\ 2000 & 405 & 250 \\ 2000 & 350 & 400 \\ 2150 & 210 & 450 \end{pmatrix} \begin{array}{l} \text{Jan} \\ \text{Feb} \\ \text{Mar} \\ \text{Apr} \end{array} \end{array}$$

X is the profit matrix

rows: January, February, March, April

columns: country A, country B, country C

X_{ij} = profit in month i from country j measured in country j's dollars where $i \in \{1,2,3,4\}$, $j \in \{1,2,3\}$

b) $w = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$ w is the exchange rate vector

$$Xw = \begin{pmatrix} 2350 & 500 & 200 \\ 2000 & 405 & 250 \\ 2000 & 350 & 400 \\ 2150 & 210 & 450 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = \begin{pmatrix} 3950 \\ 3560 \\ 3900 \\ 3920 \end{pmatrix}$$

total profit per month measured in country A's dollars

profit of month measured in country A's dollars

$$\boxed{\begin{array}{l} \text{January: } 3950 \\ \text{February: } 3560 \\ \text{March: } 3900 \\ \text{April: } 3920 \end{array}}$$

c) $v = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$

$$v^T X = \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 2350 & 500 & 200 \\ 2000 & 405 & 250 \\ 2000 & 350 & 400 \\ 2150 & 210 & 450 \end{pmatrix} = \begin{pmatrix} 8500 & 1465 & 1300 \end{pmatrix}$$

total profit from each country measured in respective country's dollars

profits from country j measured in country j's dollars

$$\boxed{\begin{array}{l} \text{A: 8500 A dollars} \\ \text{B: 1465 B dollars} \\ \text{C: 1300 C dollars} \end{array}}$$

d) $$(v^T X)w = \begin{pmatrix} 8500 & 1465 & 1300 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = \begin{pmatrix} 15330 \end{pmatrix} = 15330$$

total profit measured in country A's dollars: $\boxed{\text{15330 A dollars}}$

e) code at the end

3)

a) $y = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$

b) $y = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{pmatrix}$ where $y_i = 1$ if $i = k$
$y_i = 0$ if $i \neq k$

c) $y = \begin{pmatrix} 3 \\ 0 \\ -2 \\ 0 \\ 0 \end{pmatrix}$

d) $w = \begin{pmatrix} -1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$

e) $w = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{pmatrix}$ where $w_i = 1$ if $i = k$
$w_i = 0$ if $i \neq k$

f) $w = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{pmatrix}$ where $w_i = a$ if $i = k$
$w_i = b$ if $i = j$
$w_i = 0$ otherwise

Assume $k \neq j$

g) code at the end

4) call the matrix B. let B_j denote the jth column of B.

$B_3 = B_1 + 2B_2$, so the last column is a linear combination of the first 2. So, the last column is redundant and makes the columns linearly dependent.

This means the $\boxed{\text{rank is } 2}$ as it is the number of linearly independent columns in the matrix.

5) Since this is a linear model, the decision boundary must be a line. Since $(-1,1)$ and $(4,-2)$ lie on the decision boundary, we find the equation of this line.

$$\text{slope} = \frac{-2-1}{4-(-1)} = \frac{-3}{5}$$

$$x_2-(-2) = \frac{-3}{5}(x_1-4)$$

$$x_2+2 = \frac{-3}{5}x_1 + \frac{12}{5}$$

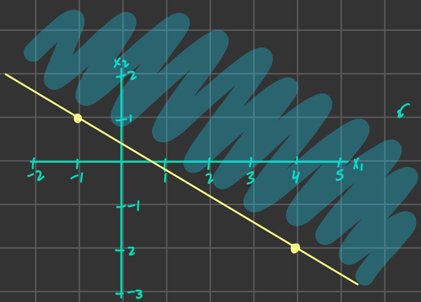
$$x_2 = \frac{-3}{5}x_1 + \frac{2}{5}$$

$$3x_1+5x_2-2=0$$

The same line can be represented as $3(3x_1+5x_2-2)=0$, but since $(4,0)$ is $+1$ and we get: $3(4)+5(0)-2=10>0$, then $c>0$ so that $\hat{y}_i$ is >0 for $(4,0)$.

Thus, the set of all weight vectors w that satisfy this problem are: $\boxed{w = c\begin{pmatrix} 3 \\ 5 \\ -2 \end{pmatrix} \text{ where } c>0}$

Different w correspond to the same boundary as $c(3x_1+5x_2-2)=0$ corresponds to the same decision boundary $3x_1+5x_2-2=0$ that we calculated, as they are the same line so $c(3x_1+5x_2-2)$ output $y_i>0$, $y_i<0$, or $y_i=0$ identically for a given (x_1,x_2).



predicts +1 for points above the decision boundary.

predicts -1 for points below the decision boundary.

6)

a) $y_i = p(z_i) = \boxed{w_0 + w_1 z_i + w_2 z_i^2 + \dots + w_{d-1} z_i^{d-1}}$

b) $$\boxed{X = \begin{pmatrix} 1 & z_1 & z_1^2 & \dots & z_1^{d-1} \\ 1 & z_2 & z_2^2 & \dots & z_2^{d-1} \\ \vdots & & & & \\ 1 & z_n & z_n^2 & \dots & z_n^{d-1} \end{pmatrix}} \quad w = \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_{d-1} \end{pmatrix}$$

$$Xw = \begin{pmatrix} 1 & z_1 & z_1^2 & \dots & z_1^{d-1} \\ 1 & z_2 & z_2^2 & \dots & z_2^{d-1} \\ \vdots & & & & \\ 1 & z_n & z_n^2 & \dots & z_n^{d-1} \end{pmatrix} \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_{d-1} \end{pmatrix} = \begin{pmatrix} w_0 + w_1 z_1 + w_2 z_1^2 + \dots + w_{d-1} z_1^{d-1} \\ w_0 + w_1 z_2 + w_2 z_2^2 + \dots + w_{d-1} z_2^{d-1} \\ \vdots \\ w_0 + w_1 z_n + w_2 z_n^2 + \dots + w_{d-1} z_n^{d-1} \end{pmatrix} = y$$

c) code at the end

In [1]:
```
import numpy as np
import scipy.io as scio
import matplotlib.pyplot as plt
```

2e

In [2]:
```
# 2a
# rows: January, February, March, April
# columns: A, B, C
A = np.array([[2350, 500, 200], [2000, 405, 250], [2000, 350, 400], [2150, 210, 450]])
A
```

Out[2]:
```
array([[2350,  500,  200],
       [2000,  405,  250],
       [2000,  350,  400],
       [2150,  210,  450]])
```

As expected, we get our profit matrix. The ith row and jth column represent the profit from country j in month i, measured in country j's dollars.

In [3]:
```
# 2b
exchange_rates = np.array([[1], [2], [3]])
profit_by_month = np.dot(A, exchange_rates)
profit_by_month
```

Out[3]:
```
array([[3950],
       [3560],
       [3900],
       [3920]])
```

As expected, we get the total profit of each month, measured via country A's dollars after accounting for exchange rates. In country A's dollars, January has 3950 dollars of profit, February has 3560 dollars of profit, March has 3900 dollars of profit, and April has 3920 dollars of profit.

In [4]:
```
# 2c
v = np.array([[1], [1], [1], [1]])
vT = v.transpose()

profit_by_country = np.dot(vT, A)
profit_by_country
```

Out[4]:
```
array([[8500, 1465, 1300]])
```

As expected, we get the total profit in each country across all the four months measured in the respective country's dollars. The total profit from country A is 8500 A dollars, total profit from country B is 1465 B dollars, and total profit from country C is 1300 C dollars.

In [5]:
```
# 2d
total_profit = np.dot(profit_by_country, exchange_rates)
total_profit
```

Out[5]:
```
array([[15330]])
```

As expected, we get the total profit across all months and countries measured in country A's dollars. The total profit is 15330 A dollars.

3g

In [6]:
```
X = np.array([[8, 0, 1, 1], [9, 2, 9, 4], [1, 5, 9, 9], [9, 9, 4, 7], [6, 9, 8, 9]])
X
```

Out[6]:
```
array([[8, 0, 1, 1],
       [9, 2, 9, 4],
       [1, 5, 9, 9],
       [9, 9, 4, 7],
       [6, 9, 8, 9]])
```

In [7]:
```
# 3a
y = np.array([[0], [0], [1], [0], [0]])
yT = y.transpose()
np.dot(yT, X)
```

Out[7]:
```
array([[1, 5, 9, 9]])
```

As expected, we get the 3rd row of X with the correct elements.

In [8]:

```
# 3b
for k in [1, 2, 3, 4, 5]:
    y_k = np.array([[0]] * (k - 1) + [[1]] + [[0]] * (5 - k))
    y_kT = y_k.transpose()
    print("y when k = {}\n".format(k), y_k)
    print("row {}\n".format(k), np.dot(y_kT, X))
```

```
y when k = 1
 [[1]
 [0]
 [0]
 [0]
 [0]]
row 1
 [[8 0 1 1]]
y when k = 2
 [[0]
 [1]
 [0]
 [0]
 [0]]
row 2
 [[9 2 9 4]]
y when k = 3
 [[0]
 [0]
 [1]
 [0]
 [0]]
row 3
 [[1 5 9 9]]
y when k = 4
 [[0]
 [0]
 [0]
 [1]
 [0]]
row 4
 [[9 9 4 7]]
y when k = 5
 [[0]
 [0]
 [0]
 [0]
 [1]]
row 5
 [[6 9 8 9]]
```

As expected, we get the 1st, 2nd, 3rd, 4th, and 5th row of X with the correct elements.

In [9]:

```
# 3c
y = np.array([[3], [0], [-2], [0], [0]])
yT = y.transpose()
np.dot(yT, X)
```

Out[9]:

```
array([[ 22, -10, -15, -15]])
```

As expected, we get 3 times the 1st row of X minus 2 times the 3rd row of X with the correct elements.

In [10]:

```
# 3d
w = [[-1], [0], [1], [0]]
np.dot(X, w)
```

Out[10]:

```
array([[-7],
       [ 0],
       [ 8],
       [-5],
       [ 2]])
```

As expected, we get the 3rd column of X minus the 1st column of X with the correct elements.

In [11]:

```
# 3e
for k in [1, 2, 3, 4]:
    w_k = np.array([[0]] * (k - 1) + [[1]] + [[0]] * (4 - k))
    print("w when k = {}\n".format(k), w_k)
    print("column {}\n".format(k), np.dot(X, w_k))
```

```
w when k = 1
 [[1]
 [0]
 [0]
 [0]]
column 1
 [[8]
 [9]
 [1]
 [9]
 [6]]
w when k = 2
 [[0]
 [1]
 [0]
 [0]]
column 2
 [[0]
 [2]
 [5]
 [9]
 [9]]
w when k = 3
 [[0]
 [0]
 [1]
 [0]]
column 3
 [[1]
 [9]
 [9]
 [4]
 [8]]
w when k = 4
 [[0]
 [0]
 [0]
 [1]]
column 4
 [[1]
 [4]
 [9]
 [7]
 [9]]
```

As expected, we get the 1st, 2nd, 3rd, and 4th column of X with the correct elements.

In [12]:

```python
# 3f
a, b = 1.5, 2
for k in [1, 2, 3, 4]:
    for j in [1, 2, 3, 4]:
        # assumption
        if k != j:
            w_kj = np.zeros((4, 1))
            w_kj[k - 1] = a
            w_kj[j - 1] = b
            print("w when k = {} and j = {}\n".format(k, j), w_kj)
            print("result\n", np.dot(X, w_kj))
```

```
w when k = 1 and j = 2
 [[1.5]
 [2. ]
 [0. ]
 [0. ]]
result
 [[12. ]
 [17.5]
 [11.5]
 [31.5]
 [27. ]]
w when k = 1 and j = 3
 [[1.5]
 [0. ]
 [2. ]
 [0. ]]
result
 [[14. ]
 [31.5]
 [19.5]
 [21.5]
 [25. ]]
w when k = 1 and j = 4
 [[1.5]
 [0. ]
 [0. ]
 [2. ]]
result
```

```
 [[14. ]
 [21.5]
 [19.5]
 [27.5]
 [27. ]]
w when k = 2 and j = 1
 [[2. ]
 [1.5]
 [0. ]
 [0. ]]
result
 [[16. ]
 [21. ]
 [ 9.5]
 [31.5]
 [25.5]]
w when k = 2 and j = 3
 [[0. ]
 [1.5]
 [2. ]
 [0. ]]
result
 [[ 2. ]
 [21. ]
 [25.5]
 [21.5]
 [29.5]]
w when k = 2 and j = 4
 [[0. ]
 [1.5]
 [0. ]
 [2. ]]
result
 [[ 2. ]
 [11. ]
 [25.5]
 [27.5]
 [31.5]]
w when k = 3 and j = 1
 [[2. ]
 [0. ]
 [1.5]
 [0. ]]
result
 [[17.5]
 [31.5]
 [15.5]
 [24. ]
 [24. ]]
w when k = 3 and j = 2
 [[0. ]
 [2. ]
 [1.5]
 [0. ]]
result
 [[ 1.5]
 [17.5]
 [23.5]
 [24. ]
 [30. ]]
w when k = 3 and j = 4
 [[0. ]
 [0. ]
 [1.5]
 [2. ]]
result
 [[ 3.5]
 [21.5]
 [31.5]
 [20. ]
 [30. ]]
w when k = 4 and j = 1
 [[2. ]
 [0. ]
 [0. ]
 [1.5]]
result
 [[17.5]
 [24. ]
 [15.5]
 [28.5]
 [25.5]]
w when k = 4 and j = 2
 [[0. ]
 [2. ]
 [0. ]
 [1.5]]
result
 [[ 1.5]
```

```
 [10. ]
 [23.5]
 [28.5]
 [31.5]]
w when k = 4 and j = 3
 [[0. ]
 [0. ]
 [2. ]
 [1.5]]
result
 [[ 3.5]
 [24. ]
 [31.5]
 [18.5]
 [29.5]]
```

Here, we get a times the kth column of X + b times the jth column of X, where a = 1.5 and b = 2. We also assume that k is not equal to j.

In [14]:

```
# 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(-1, 1, n)

d = 3  # degree
w = np.random.rand(d)

# TODO : generate X - matrix
X = np.vander(z, d)

# TODO : evaluate polynomial at all points z,
# and store the result in p
# do NOT use a loop for this
p = np.dot(X, w)

# plot the datapoints and the best - fit polynomials
plt.plot(z, p, linewidth=2)
plt.xlabel("z")
plt.ylabel("y")
plt.title("polynomial with coefficients w = % s" % w)
plt.show()
```