# HW 1- CMSC 25300

Bayard Walsh

March 2023

# 1

## 1.1 1 a)

*Note: following convention where rows indicate data instances and columns are features*

$$M = \begin{array}{c} \begin{matrix} Housing & Food & Recreation + Transportation \end{matrix} \\ \begin{pmatrix} 2350 & 500 & 200 \\ 2000 & 405 & 250 \\ 2000 & 350 & 400 \\ 2150 & 210 & 450 \end{pmatrix} \end{array} \begin{array}{c} \\ January \\ February \\ March \\ April \end{array}$$

Rows represent data instances/ training samples, which is specific months in this case. Columns represent features of a given piece of data, which are various categories of monthly expenses in this matrix.

## 1.2 1 b)

Considering we have three columns we want to collapse into one index per row we will use the following column vector: $x$, with degree 3, so we can have a $4x1$ column vector as an output. Therefore, we have $x$ as defined below

$$x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \begin{matrix} (Housing) \\ (Food) \\ (Recreation + Transportation) \end{matrix}$$

From this, we have

$$M \cdot x = \begin{pmatrix} 2350 * x_1 + 500 * x_2 + 200 * x_3 & (January) \\ 2000 * x_1 + 405 * x_2 + 250 * x_3 & (February) \\ 2000 * x_1 + 350 * x_2 + 400 * x_3 & (March) \\ 2150 * x_1 + 210 * x_2 + 450 * x_3 & (April) \end{pmatrix}$$

Now we will set $1 = x_1 = x_2 = x_3$, and then we will obtain our desired output, which is this matrix :

$$M \cdot x = \begin{pmatrix} 3050 & (January) \\ 2655 & (February) \\ 2750 & (March) \\ 2810 & (April) \end{pmatrix}$$

## 1.3 1 c)

Considering we have four months we want to collapse into one row per column we will use a column $y$ vector to represent each of the 4 months. However, in order to fit the dimensions we will transpose the vector so that it is a row vector. Therefore we have:

$$y^T = \begin{pmatrix} y_1 & y_2 & y_3 & y_4 \end{pmatrix}$$

From this, we have **(NOTE: representing as column matrix so the equations are visible, but actually will be 1x3 row vector, described below**

$$y^T \cdot M = \begin{pmatrix} 2350 * y_1 + 2000 * y_2 + 2000 * y_3 + 2150 * y_4 & (Housing) \\ 500 * y_1 + 405 * y_2 + 350 * y_3 + 210 * y_4 & (Food) \\ 200 * y_1 + 250 * y_2 + 400 * y_3 + 450 * y_4 & (Recreation + Transportation) \end{pmatrix}$$

If we set $1 = y_1 = y_2 = y_3 = y_4$, we obtain our desired output, which is this matrix :

$$y^T \cdot M = \begin{pmatrix} 8500(Housing) & 1465(Food) & 1300(Recreation + Transportation) \end{pmatrix}$$

## 1.4 1 d)

Using the previous defined matrices, we will compute total expenses through this equation $y^T \cdot M \cdot x$. Note that $y$ and $x$ will have $1 = \forall x_n, y_n$, because we want to sum up our values so we can scale by 1 and sum. First we will begin with $y^T \cdot M$ as computed previously

$$y^T \cdot M = \begin{pmatrix} 8500(Housing) & 1465(Food) & 1300(Recreation + Transportation) \end{pmatrix}$$

Now we want $x \cdot y^T \cdot M$, which we can obtain below (again we will have $1 = x_1 = x_2 = x_3$):

$$x \cdot y^T \cdot M = \begin{pmatrix} 8500 * x_1 + 1465 * x_2 + 1300 * x_3 & (Total) \end{pmatrix} = 11265$$

This is our total expenses across all categories and months.

### 1.5 1 e)

I will write code in python corresponding to each of the 4 previous sections, with brief comment descriptions before. I also will reference matrices computed in previous steps in my calculation:

```
# 1 a)
 #  create 4x3 matrix with array of arrays in python, and
 #  use np.array function from numpy.
 # Will print the function as a way of returning the variables
matrix = np.array([[2350, 500 , 200],
[2000, 405, 250],
[2000, 350, 400],
[2150, 210, 450]])
print(matrix)

# 1 b)
# create vector X initilized to all 1s
#  create outcome multiplication vector mx as dot product of matrix and x
# and then print mx. Note while mx is 4 degree column vector, vectors are
# represented in an array
x=[1,1,1]
mx=np.dot(matrix,x)
print(mx)

# 1 c)
# create vector y initilized to all 1s, then yt as transposed vector y
# create outcome multiplication vector ytm as dot product of y and matrix
# and then print ytm.
y=[1,1,1,1]
yt=np.transpose(y)
ytm=np.dot(yt,matrix)
print(ytm)

# 1 d)
# use ytm and take dot product with x as defined previously, print xytm as
# final matrix (1,1) size
xytm=np.dot(x,ytm)
print(xytm)
```

# 2

**2 a)**
First we identify the second row of $X$ as $x_2$,

$$x_2 = \begin{pmatrix} 9 & 2 & 9 & 4 \end{pmatrix}$$

As we want to solve $y^T X = x_2$ with respect to $y$, we want to find $y$ such that in the dot product only values from the second row are return in a given column. Therefore, we will define $y$ column vector as follows:

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

We also have ($y^T$ is a row vector)

$$y^T = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

**(NOTE: representing as column matrix so the equations are visible, but actually will be 1x4 row vector, described below**

$$y^T X = \begin{pmatrix} 0*8+1*9+0*1+0*9+0*6 \\ 0*0+1*2+0*5+0*9+0*9 \\ 0*1+1*9+0*9+0*4+0*8 \\ 0*1+1*4+0*9+0*7+0*9 \end{pmatrix} = \begin{pmatrix} 9 & 2 & 9 & 4 \end{pmatrix}$$

The only nonzero value is 1 in the second column of $y^T$, as we want to have the matrix $y^T X = x_2$, so we will only keep the values in the second row of $X$ as we compute $y^T \cdot X = x_2$, because every equation will be a given column of $X$ where every value $\notin$ row 2 is set to 0. As we have $y^T \cdot X = x_2$ (the only values which are nonzero in each of the systems of equations will be the computation in the second row), so our solution is correct

**2 b)**
We will follow the approach above to compute the $y$ vector which corresponds to the $kth$ row of $X$ (or $x_k$) for $y^T X$:

1) where $y_k$ is the $kth$ row index in vector $y$, let $y_k = 1$
2) let all other values of $y = 0$

Next we take $y^T$, and we know that the only non-zero value of $y^T$ is in the $kth$ column, and as this column equals 1, we know that $y^T X = x_k$, or we will

return the $kth$ row of $X$ as all other values will be multiplied by 0, except for the given value in the $kth$ column, which only results in the $k$ row. Therefore we have generalized the solution from the same approach as above.

**2 c)**
First, we will define the process completed in **2 b)** as the function $R(X, r)$, where $X$ is a given matrix and $r$ is a given row within $X$ ( $r$ must reference a legal $X$ row). We will repeat this computation in our answer for $y$.

Next, we will define vectors $y' = R(X, k)$ (so that $y'^T X = x_k$ row of $X$) and $y'' = R(X, j)$ (so that $y''^T X = x_j$ row of $X$) *note: as* $k, j \in \{1 \cdots 5\}$ *they are legal inputs for function* $R$

Next, we will scale $y'$ and $y''$, so that $y' = y' \cdot a$ and $y'' = y'' \cdot b$.

Finally, we will define $y = y' + y''$.

As $y'^T X = x_k$ row of $X$ scaled by $a$, and $y''^T X = x_j$ row of $X$ scaled by $b$, we know that from $y = y' + y''$, $y$ is the sum of these two computations, which is our desired answer.

**2 d)**
First we identify the third columns of $X$ as $x_3$,

$$x_3 = \begin{pmatrix} 1 \\ 9 \\ 9 \\ 4 \\ 8 \end{pmatrix}$$

As we want to solve $Xw = x_3$ with respect to $w$, we want to find $w$ such that in the dot product only values from the third column are returned based on $w$. Therefore, we will define column vector $w$ as follows:

$$w = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

$$Xw = \begin{pmatrix} 0*8+0*0+1*1+0*1 \\ 0*9+0*2+1*9+0*4 \\ 0*1+0*5+1*9+0*9 \\ 0*9+0*9+1*4+0*7 \\ 0*6+0*9+1*8+0*9 \end{pmatrix} = \begin{pmatrix} 1 \\ 9 \\ 9 \\ 4 \\ 8 \end{pmatrix}$$

The only nonzero value is 1 in the third row of $w$, and as we want to have the matrix $Xw = x_3$, we will only keep the values in the third column of $X$. As we have $Xw = x_3$ (the only values which are nonzero in the computation are the third column values which are multiplied by 1), our solution is correct

**2 e)**

We will follow the approach above to compute the $w$ vector which corresponds to the $kth$ column of $X$ (or $x_k$) from $Xw$:

1) where $w_k$ is the $kth$ row index in vector $w$, let $w_k = 1$
2) let all other values of $w = 0$

We know that the only non-zero value of $w$ is in the $kth$ row, and as this row equals 1, we know that $Xw = x_k$, or we will return the $kth$ column of $X$ as all other values will be 0 and each value in $X$ is multiplied by 1 for the row in $w$. Therefore we have generalized the solution from the same approach as above.

**2 f)**

First, we will define the process completed in **2 e)** as the function $V(X, r)$, where $X$ is a given matrix and $r$ is a given column within $X$ (that is $r$ is within the bounds of legal $X$ columns).
Next, we will define vector $w' = V(X, k)$ ($Xw' = kth$ column of $X$) and $w'' = V(X, j)$ ($Xw'' = jth$ column of $X$) *note: as* $k, j \in \{1 \cdots 4\}$ *they are legal inputs for function* $V$
Next, we will scale $w'$ and $w''$, so that $w' = w' \cdot a$ and $w'' = w'' \cdot b$.
Finally, we will define $w = w' + w''$.
As $w'$ is initially designed so $Xw' = kth$ column of $X$ and we scaled $w'$ by $a$, and $Xw'' = jth$ column of $X$ and we scaled $w''$ by $b$, we know that from $w = w' + w''$, $w$ is the sum of these two computations, which is our desired answer.

**2 g)**

```python
import numpy as np

#  input matrix X
X = np.array([[8,0,1,1], [9,2,9,4], [1,5,9,9], [9,9,4,7], [6,9,8,9]])

#Note, designing functions to be based on global matrix X, following bounds from
# the problem statement. This function is 2 c)
# assuming no invalid inputs for j,k,a,b (so all reals for a,b)
# and j,k are valid row index
# 2 c)
def scale_row(a,b,j,k):
    j=j-1
    k=k-1 #offset 0 index for columns, assume input range 1-4

    y_1=[0,0,0,0,0] #initilize vectors
    y_1[j]=1
    y_2=[0,0,0,0,0]
    y_2[k]=1

    y_1=np.transpose(y_1) # transposing
    y_2=np.transpose(y_2)

    y_1=np.dot(y_1,X) # vector math
    y_1 = [i * a for i in y_1] #scale

    y_2=np.dot(y_2,X)
    y_2 = [i * b for i in y_2]

    y=[0,0,0,0] # sum function
    y= [y_1[i]+y_2[i] for i in range(4)]
    return y

# now we solve a and b
#2a - scale set to 1, only looking for one row, so other row set to 0
# and other scaler 0, row index j=2 for second row
scale_row(1,0,2,0)

#2b - define general function for any row k
# only variable chosen input is K (set to j), not scaled vector so a=1
# only one row so b=k=0
def generalized_row(K):
    return scale_row(1,0,K,0)

```

```
43
44  #Note, designing functions to be based on global matrix X, following bounds from
45  # the problem statement. This function is 2 f)
46  # assuming no invalid inputs for j,k,a,b (so all reals for a,b)
47  # and j,k are valid column index
48  # 2 f)
49  def scale_column(a,b,j,k):
50      j=j-1
51      k=k-1 #offset 0 index for rows, assume input range 1-4
52
53      y_1=[0,0,0,0] #initilize vectors
54      y_1[j]=1
55      y_2=[0,0,0,0]
56      y_2[k]=1
57
58
59      y_1=np.dot(X,y_1) # vector math
60      y_1 = [i * a for i in y_1] #scale
61
62      y_2=np.dot(X,y_2)
63      y_2 = [i * b for i in y_2]
64
65      y=[0,0,0,0,0] # sum function
66      y= [y_1[i]+y_2[i] for i in range(5)]
67      return y
68
69  # now we solve d and e
70  #2d - scale set to 1, only looking for one column, so other column set to 0
71  # and other scaler 0, column index j=3 for third row
72  print(scale_column(1,0,3,0))
73
74  #2e - define general function for any column e
75  # only variable chosen input is K (set to j), not scaled vector so a=1
76  # only one row so b=k=0
77  def generalized_column(K):
78      return scale_column(1,0,K,0)
79
```

## 3

The given matrix, lets call it, $X \in R^{n \cdot p}$ has rank 1 because the smallest $r$ such that we can find $U \in R^{n \cdot r}$ and $V \in R^{r \cdot p}$, where $X = UV$ is 1, or $r = 1$.

We will use the following values for $U$ and $V$ to show that $r = 1$ is possible:

$$U^{4\cdot 1} = \begin{bmatrix} 2 \\ 10 \\ 6 \\ 14 \end{bmatrix} \tag{1}$$

$$V^{1\cdot 3} = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \tag{2}$$

As $UV = X$ and $r = 1$ for these given $U$, $V$, we have $X$ is a matrix of rank 1

# 4

First we make the following substitution from our weight vector **w** for the model equation:

$\hat{y} = 3 \cdot x_{i,1} + 5 \cdot x_{i,2} - 2$

From our predicted label formula, we can substitute in $\hat{y} > 0$ classifying as $+1$ for $\hat{y}$, which gives us the following formula for points which are classified as $+1$:

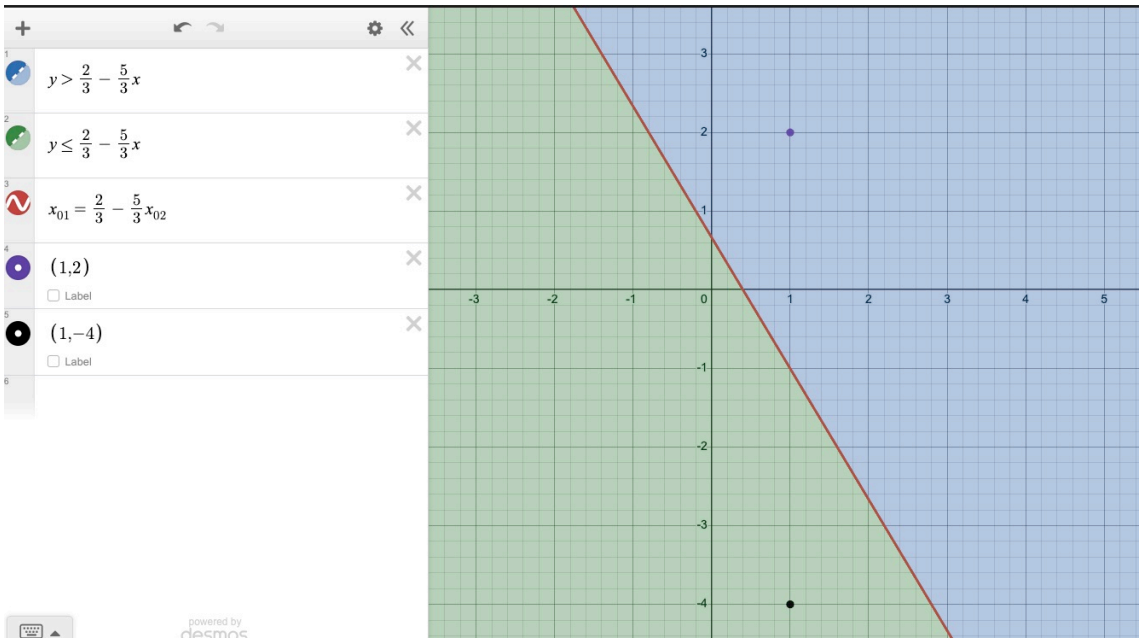
$(3 \cdot x_{i,1}) + (5 \cdot x_{i,2}) - 2 > 0$

After some simple algebra, we have the following equation for classifying $+1$ based on $x_{i,1}$ and $x_{i,2}$ :

$x_{i,1} > \frac{2}{3} - (\frac{5}{3} \cdot x_{i,2})$

From here we have the following graph, which is a direct graph of the above equation in the $x_{i,1}, x_{i,2}$ space: *(see next page)*

**4) cont.**




Here, we have the classification equation as the red line. **In order to classify the space, we have $y = x_{01}$ and $x = x_{02}$ so we can use Desmos graphing equations to demonstrate the space where our model would predict a $+1$ or $-1$.** Therefore, the blue area will indicate places which are labeled $+1$ by the model and the green area represents places which are labeled $-1$ by the model. We also have two arbitrary black and a purple points, which are annotated below:

Black point: $(x_{01} = 1, x_{02} = -4)$. Therefore we have $\hat{y} = (3 \cdot 1) + (5 \cdot -4) - 2 = -19$, so $\hat{y} \not> 0$, meaning the predicted label should be $-1$ for the model, which it is on the graph (the point is in the green section)

Purple point: $(x_{01} = 1, x_{02} = 2)$. Therefore we have $\hat{y} = (3 \cdot 1) + (5 \cdot 2) - 2 = 11$, so $\hat{y} > 0$, meaning the predicted label should be $+1$ for the model, which it is on the graph (the point is in the blue section)

Therefore for the given weight function and predicted labeling, the plot is correct (as shown through two arbitrary points)

# 5

**5 a)**
A general expression of any polynomial $p$ of degree $d$, is the form

$$p(z) = a_0 + a_1 \cdot z + a_2 \cdot z^2 + \cdots + a_d \cdot z^d = y$$

where $a_0, a_1 \cdots a_d$ are coefficients of the polynomial and $z$ is an independent variable.

**5 b)**

For each equation, we have the given form for all $z_i$ based on the generalized equation:

$$\begin{aligned}
&p(z_0) = a_0 + a_1 \cdot z_0 + a_2 \cdot z_0^2 + \cdots + a_d \cdot z_0^d = y_0 \\
&p(z_1) = a_0 + a_1 \cdot z_1 + a_2 \cdot z_1^2 + \cdots + a_d \cdot z_1^d = y_1 \\
&\vdots \\
&p(z_n) = a_0 + a_1 \cdot z_n + a_2 \cdot z_n^2 + \cdots + a_d \cdot z_n^d = y_n
\end{aligned}$$

From this structure of equations based on a given $z_i$, we will now transfer this into the Vandermonde matrix and a scalar vector, observed below

$$X = \begin{vmatrix} 1 & z_0 & z_0^2 & \cdots & z_0^n \\ 1 & z_1 & z_1^2 & \cdots & z_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & z_n & z_n^2 & \cdots & z_n^n \end{vmatrix}$$

$$w = \begin{vmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{vmatrix}$$

$$y = \begin{vmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{vmatrix}$$

From these matrices and the system of equations above, we have $Xw = y$, such that the matrix relation depicts the system of equations for all $z_i$

**5 c)**

```
# 5 C)
# NOTE: IMPORTED FILES (numpy, scipy,matplotlib.pyplot) are all included
# earlier in file
# 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  )
X = np . zeros (( n , d  ))

# TODO : generate X- matrix

#create Vandermonde matrix of degree 3 from z with vander function
# note that d is used to limit number of columns, and we only want 3 degree
# polynomials. We use array z so we hav  z^0, z^1, ..., z^(d-1). We have
# true to ensure that function is increasing ( z^0, z^1, ..., z^(d-1)
# instead of  z^(d-1) ..., z^1, z^0)
X=np.vander(z,d,True)

# TODO : evaluate polynomial at all points z, and store the result in p
# do NOT use a loop for this
# plot the datapoints and the best -fit polynomials

# use np.polynomial.polynomial.polyval, a function which
# will return the value p(x)= w0 + w1*x+... wn+x^n (where w is
# length n+1). As this matches our formula of Xw=y, this value of p
# will evaluate polynomial at all points z in p
p=np.polynomial.polynomial.polyval(X,w)

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