

# **MTH 602 Scientific Machine Learning**

Homework 1

9/22/2025

S. M. Mahfuzul Hasan

02181922



# I. MATH REFRESHER: LINEAR ALGEBRA & VECTOR CALCULUS

1. Given,

$$\phi(x_1, x_2, x_3) = \frac{1}{\sqrt{x_1^2 + x_2^2 + x_3^2}}$$

(a)

$$\begin{aligned}\vec{F} = \nabla\phi &= \left( \frac{\partial}{\partial x_1}\hat{i} + \frac{\partial}{\partial x_2}\hat{j} + \frac{\partial}{\partial x_3}\hat{k} \right) \frac{1}{\sqrt{x_1^2 + x_2^2 + x_3^2}} \\ &= \frac{-1}{2\sqrt{(x_1^2 + x_2^2 + x_3^2)^3}} (2x_1\hat{i} + 2x_2\hat{j} + 2x_3\hat{k}) \\ &= \frac{-1}{\sqrt{(x_1^2 + x_2^2 + x_3^2)^3}} (x_1\hat{i} + x_2\hat{j} + x_3\hat{k}) \quad (\text{Ans.})\end{aligned}$$

(b) At  $(1, 0, 0)$ ,

$$\begin{aligned}\vec{F}(1, 0, 0) &= \frac{-1}{\sqrt{(1^2 + 0^2 + 0^2)^3}} (1\hat{i} + 0\hat{j} + 0\hat{k}) \\ &= -\hat{i} \quad (\text{Ans.})\end{aligned}$$

Direction: along (-)ve  $x_1$  axis.

Magnitude:  $\|\vec{F}(1, 0, 0)\|_2 = \sqrt{(-1)^2} = 1$

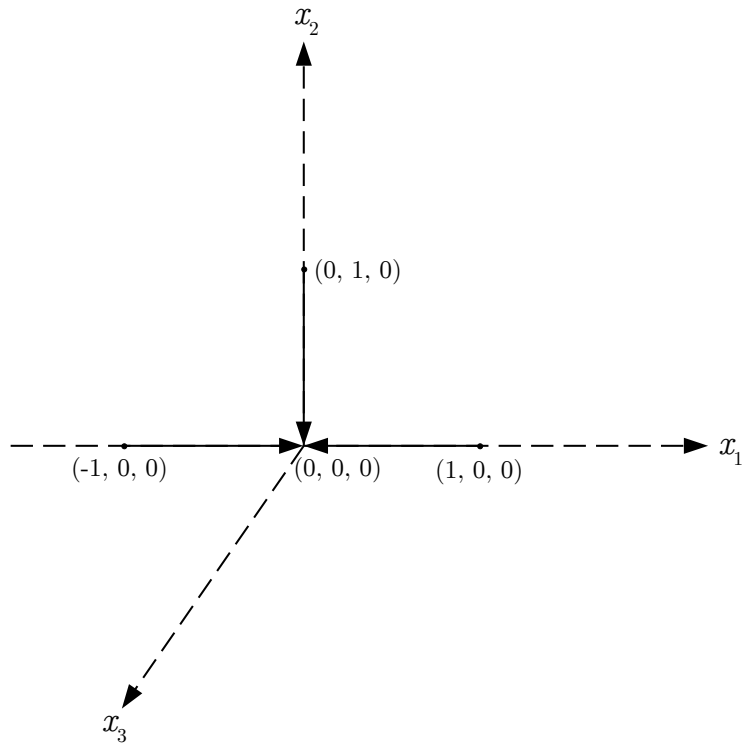


Figure 1: Sketch of  $\vec{F}$  at  $(1, 0, 0)$ ,  $(-1, 0, 0)$  and  $(0, 1, 0)$ .

At  $(-1, 0, 0)$ ,

$$\begin{aligned}\vec{F}(-1, 0, 0) &= \frac{-1}{\sqrt{((-1)^2 + 0^2 + 0^2)^3}} (-1\hat{i} + 0\hat{j} + 0\hat{k}) \\ &= \hat{i} \quad (\text{Ans.})\end{aligned}$$

Direction: along (+)ve  $x_1$  axis.

Magnitude:  $\|\vec{F}(-1, 0, 0)\|_2 = \sqrt{1^2} = 1$

At  $(0, 1, 0)$ ,

$$\begin{aligned}\vec{F}(0, 1, 0) &= \frac{-1}{\sqrt{(0^2 + 1^2 + 0^2)^3}} (0\hat{i} + 1\hat{j} + 0\hat{k}) \\ &= -\hat{j} \quad (\text{Ans.})\end{aligned}$$

Direction: along (-)ve  $x_2$  axis.

Magnitude:  $\|\vec{F}(0, 1, 0)\|_2 = \sqrt{(1)^2} = 1$

2. Given,

$$\begin{aligned}\langle \vec{x}, \vec{y} \rangle &= \vec{x}^T \vec{y}, \text{ where } \vec{x}, \vec{y} \in \mathbb{R}^N \\ Q^T Q &= I, \text{ where } Q \in \mathbb{R}^{N \times N}\end{aligned}$$

(a)

$$\begin{aligned}\text{L.H.S} &= \langle Q\vec{x}, Q\vec{y} \rangle = (Q\vec{x})^T (Q\vec{y}) = (\vec{x}^T Q^T) (Q\vec{y}) = \vec{x}^T (Q^T Q) \vec{y} \\ &= \vec{x}^T I \vec{y} = \vec{x}^T \vec{y} = \langle \vec{x}, \vec{y} \rangle = \text{R.H.S} \quad (\text{Showed})\end{aligned}$$

(b)

$$\begin{aligned}\|Q\vec{x}\|_2 &= \sqrt{(Q\vec{x})^T (Q\vec{x})} = \sqrt{(\vec{x}^T Q^T) (Q\vec{x})} = \sqrt{\vec{x}^T (Q^T Q) \vec{x}} \\ &= \sqrt{\vec{x}^T I \vec{x}} = \sqrt{\vec{x}^T \vec{x}} = \|\vec{x}\|_2\end{aligned}$$

**Optional:** PCA uses orthogonal matrix to transform (rotate or reflect) dataset from one co-ordinate system to another. Since, it preserves length and angle of the vectors it is applied to, the transformed dataset retains the original geometry (length and angle) along with the total variance. This gives numerical stability and efficiency in computations of PCA.

3. To verify,

$$\begin{aligned}\|\vec{x}\|_\infty &\leq \|\vec{x}\|_2 \\ \|\vec{x}\|_2 &\leq \sqrt{N} \|\vec{x}\|_\infty\end{aligned}$$

(a)

$$\|\vec{x}\|_\infty = \max_{1 \leq i \leq N} |x_i| \quad (1)$$

Let's arrange the elements of  $\vec{x}$  in such a way that the maximum absolute valued element sits at  $N^{th}$  position. Then equation (1) becomes-

$$\|\vec{x}\|_\infty = |x_N| \quad (2)$$

Now,

$$\begin{aligned}
\|\vec{x}\|_2 &= \sqrt{\sum_{i=1}^N |x_i|^2} = \sqrt{\sum_{i=1}^{N-1} |x_i|^2 + |x_N|^2} \\
\Rightarrow \|\vec{x}\|_2 &\geq \sqrt{|x_N|^2}, \quad \text{since, } \sum_{i=1}^{N-1} |x_i|^2 \geq 0 \\
\Rightarrow \|\vec{x}\|_2 &\geq \sqrt{(\|\vec{x}\|_\infty)^2}, \quad \text{from equation (2)} \\
\Rightarrow \|\vec{x}\|_\infty &\leq \|\vec{x}\|_2 \quad (\text{Verified})
\end{aligned}$$

(b) We know,

$$|x_i| \leq \max_{1 \leq i \leq N} |x_i|, \quad \text{for any } i \in [1, N] \quad (3)$$

Let's assume again that the maximum absolute valued element is  $x_N$ . Then equation (3) can be written as,

$$\begin{aligned}
|x_i|^2 &\leq |x_N|^2 \\
\Rightarrow \sum_{i=1}^N |x_i|^2 &\leq \sum_{i=1}^N |x_N|^2 = N|x_N|^2 \\
\Rightarrow \sqrt{\sum_{i=1}^N |x_i|^2} &\leq \sqrt{N|x_N|^2} \\
\Rightarrow \|\vec{x}\|_2 &\leq \sqrt{N}\|\vec{x}\|_\infty \quad (\text{Verified})
\end{aligned}$$

4. Let,

$$A \in \mathbb{R}^{N \times (M+1)}, \vec{y} \in \mathbb{R}^N, \vec{\omega} \in \mathbb{R}^{M+1}$$

(a)

$$\begin{aligned}
\|\vec{y} - A\vec{\omega}\|_2^2 &= (\vec{y} - A\vec{\omega})^T (\vec{y} - A\vec{\omega}) \\
&= \left( \vec{y}^T - (A\vec{\omega})^T \right) (\vec{y} - A\vec{\omega}), \quad \text{since, } (X - Y)^T = X^T - Y^T \\
&= \vec{y}^T \vec{y} - \vec{y}^T (A\vec{\omega}) - (A\vec{\omega})^T \vec{y} + (A\vec{\omega})^T (A\vec{\omega}) \\
&= \vec{y}^T \vec{y} - (\vec{y}^T (A\vec{\omega}))^T - (A\vec{\omega})^T \vec{y} + (A\vec{\omega})^T (A\vec{\omega}), \\
&\quad \text{since, } \vec{y}^T (A\vec{\omega}) \text{ is a scalar quantity} \\
&= \vec{y}^T \vec{y} - (A\vec{\omega})^T (\vec{y}^T)^T - (A\vec{\omega})^T \vec{y} + (A\vec{\omega})^T (A\vec{\omega}), \\
&\quad \text{since, } (XY)^T = Y^T X^T \\
&= \vec{y}^T \vec{y} - (A\vec{\omega})^T \vec{y} - (A\vec{\omega})^T \vec{y} + (A\vec{\omega})^T (A\vec{\omega}), \\
&\quad \text{since, } (X^T)^T = X \\
&= \vec{y}^T \vec{y} - 2(A\vec{\omega})^T \vec{y} + (A\vec{\omega})^T (A\vec{\omega}) \\
&= \vec{\omega}^T A^T A \vec{\omega} - 2\vec{\omega}^T A^T \vec{y} + \vec{y}^T \vec{y} \quad (\text{Showed})
\end{aligned}$$

Note:  $X$  and  $Y$  are representative matrices.

(b)

$$\nabla_{\vec{\omega}} \|\vec{y} - A\vec{\omega}\|_2^2 = \nabla_{\vec{\omega}} (\vec{\omega}^T A^T A \vec{\omega} - 2\vec{\omega}^T A^T \vec{y} + \vec{y}^T \vec{y}) \quad (4)$$

$$\begin{aligned} \nabla_{\vec{\omega}} (\vec{\omega}^T A^T A \vec{\omega}) &= (A^T A + (A^T A)^T) \vec{\omega}, \\ &\text{using } \nabla_{\vec{\omega}} (\vec{\omega}^T B \vec{\omega}) = (B + B^T) \vec{\omega}, \text{ where } B = A^T A \text{ here} \\ &= (A^T A + A^T A) \vec{\omega} = 2A^T A \vec{\omega} \\ \nabla_{\vec{\omega}} (\vec{\omega}^T A^T \vec{y}) &= A^T \vec{y}, \\ &\text{using } \nabla_{\vec{\omega}} (\vec{\omega}^T \vec{y}') = \vec{y}', \text{ where } \vec{y}' = A^T \vec{y} \text{ here} \\ \nabla_{\vec{\omega}} (\vec{y}^T \vec{y}) &= 0, \quad \text{since, } \vec{y} \text{ does not depend on } \vec{\omega} \end{aligned}$$

So, equation (4) becomes-

$$\nabla_{\vec{\omega}} \|\vec{y} - A\vec{\omega}\|_2^2 = 2A^T A \vec{\omega} - 2A^T \vec{y} + 0 = 2A^T A \vec{\omega} - 2A^T \vec{y} \quad (\text{Showed}) \quad (5)$$

5. Projection matrix is defined as,

$$P = A (A^T A)^{-1} A^T$$

(a)

$$\begin{aligned} L.H.S &= P(P\vec{y}) = A (A^T A)^{-1} A^T \left( A (A^T A)^{-1} A^T \vec{y} \right) \\ &= A \left( (A^T A)^{-1} A^T A \right) (A^T A)^{-1} A^T \vec{y} \\ &= AI (A^T A)^{-1} A^T \vec{y}, \\ &\quad \text{since, } A^T A \text{ is a square matrix } \left( \in \mathbb{R}^{(M+1) \times (M+1)} \right) \text{ and invertible} \\ &= A (A^T A)^{-1} A^T \vec{y} = P\vec{y} = R.H.S \quad (\text{Showed}) \end{aligned}$$

(b)

$$\begin{aligned} L.H.S &= P\vec{y}_M = A (A^T A)^{-1} A^T \vec{y}_M \\ &= A (A^T A)^{-1} A^T (A\vec{\omega}) \\ &= A \left( (A^T A)^{-1} A^T A \right) \vec{\omega} \\ &= AI\vec{\omega} = A\vec{\omega} = \vec{y}_M = R.H.S \quad (\text{Showed}) \end{aligned}$$

(c) The condition for least squares solution as derived in equation (5) is

$$\begin{aligned} \nabla_{\vec{\omega}} \|\vec{r}\|_2^2 &= \nabla_{\vec{\omega}} \|\vec{y} - \vec{y}_M\|_2^2 = 2A^T A \vec{\omega}_* - 2A^T \vec{y} = 0 \\ &\Rightarrow A^T A \vec{\omega}_* - A^T \vec{y} = 0 \end{aligned} \quad (6)$$

Now,

$$\begin{aligned} L.H.S &= P\vec{r} = A (A^T A)^{-1} A^T (\vec{y} - \vec{y}_M) \\ &= A (A^T A)^{-1} (A^T \vec{y} - A^T \vec{y}_M) \\ &= A (A^T A)^{-1} (A^T \vec{y} - A^T A \vec{\omega}_*), \quad \text{since, } \vec{y}_M = A\vec{\omega}_* \\ &= A (A^T A)^{-1} (0), \quad \text{from equation (6)} \\ &= 0 = R.H.S \quad (\text{Showed}) \end{aligned}$$

- (d) It has already been shown that  $P$  is idempotent ( $P^2 = P$ ) in (a). Now to have the orthogonality, it must satisfy  $P^T = P$ .

$$\begin{aligned}
P^T &= \left( A (A^T A)^{-1} A^T \right)^T \\
&= (A^T)^T \left( (A^T A)^{-1} \right)^T A^T \\
&= A \left( (A^T A)^T \right)^{-1} A^T \\
&= A (A^T A)^{-1} A^T = P
\end{aligned}$$

- (i) From equation (6) of (c), it is easy to see that the condition for least square solution  $\vec{y}_M = A\vec{\omega}_*$  satisfies the condition:

$$\begin{aligned}
A^T A \vec{\omega}_* &= A^T \vec{y} \\
\Rightarrow \vec{\omega}_* &= (A^T A)^{-1} A^T \vec{y} \\
\Rightarrow A \vec{\omega}_* &= A (A^T A)^{-1} A^T \vec{y} \\
\Rightarrow \vec{y}_M &= P \vec{y}
\end{aligned}$$

which shows that  $P$  maps  $\vec{y} \in \mathbb{R}^N$  to the optimal least squares solution.

- (ii) It has already been shown in (c) that  $P\vec{r} = 0$  which translates to  $P$  mapping residual vector  $\vec{r}$  to zero.

To conclude,  $P$  indeed is an orthogonal projection operator that maps  $\vec{y}$  to the optimal least squares solution and residual vector to zero.

**Optional:** If the features are orthogonal to each other, then  $A^T A$  becomes a diagonal matrix, and its inversion becomes simplified and easy to calculate. This also makes the computation of each coefficient independent of one another with each coefficient being calculated using  $\omega_j = \frac{a_j^T \vec{y}}{a_j^T a_j}$  (where,  $a_j$  is  $j$ -th feature and  $\omega_j$  is  $j$ -th coefficient) and makes the contribution of each coefficient clearly interpretable in the ML regression model.

## II. CODING COMPETITION: POLYNOMIAL REGRESSION, CONDITIONING, AND OVERFITTING

1. Please refer to 1.
2. For  $M = 1$ ,

$$\vec{\omega} = \begin{bmatrix} 0.06297434 \\ 1.71053715 \end{bmatrix}$$

For  $M = 3$ ,

$$\vec{\omega} = \begin{bmatrix} -7.10317894 \times 10^{-4} \\ 2.00931092 \\ -3.88421224 \times 10^{-2} \\ -2.87850274 \times 10^{-1} \end{bmatrix}$$

For  $M = 9$ ,

$$\vec{\omega} = \begin{bmatrix} -3.17365859 \times 10^{-3} \\ 2.24500920 \\ -4.91665701 \\ 42.6086166 \\ -199.415756 \\ 538.140270 \\ -872.045932 \\ 836.900160 \\ -438.527766 \\ 96.7029592 \end{bmatrix}$$

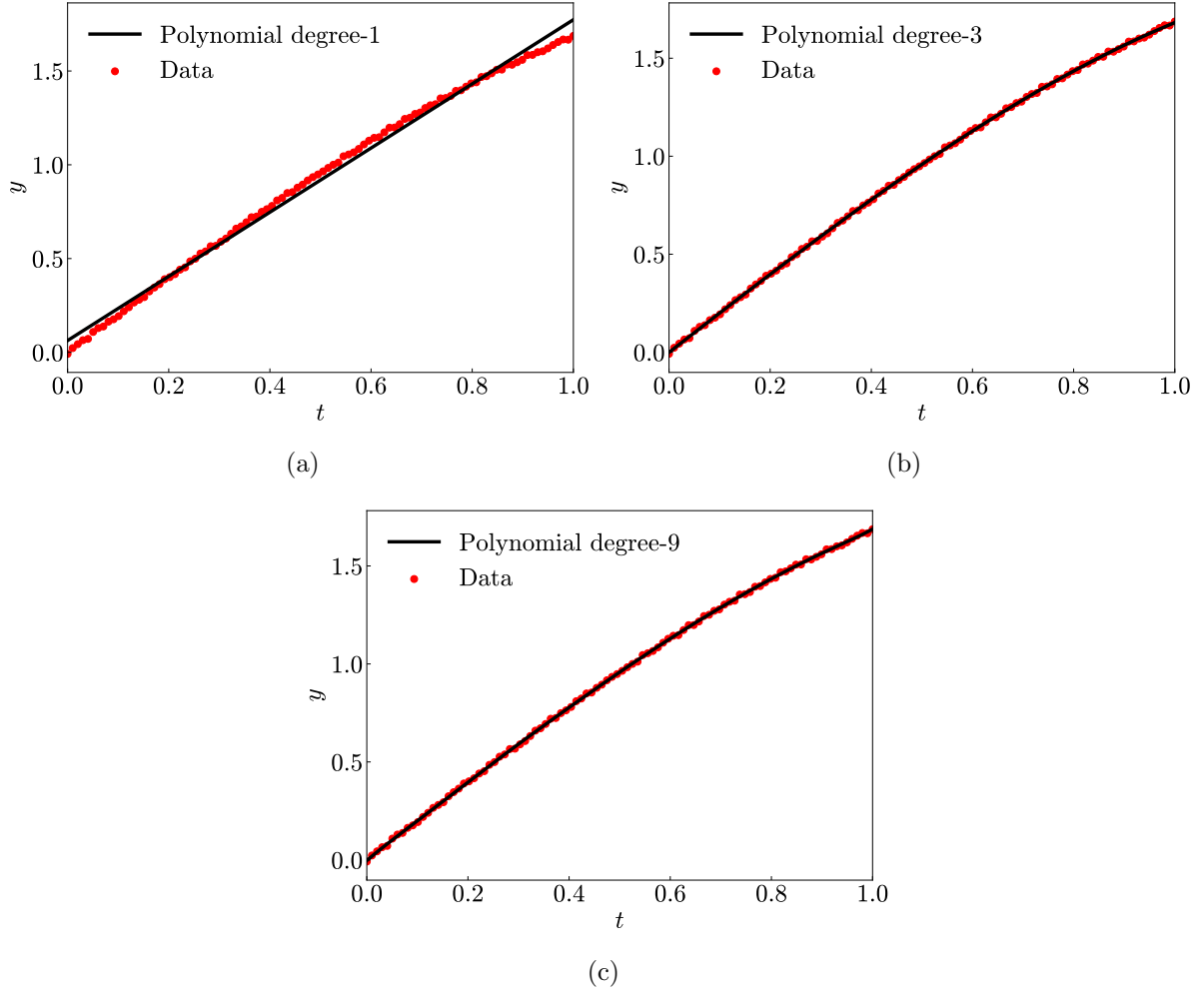


Figure 2: Polynomial regression fit for (a)  $M = 1$ , (b)  $M = 3$ , and (c)  $M = 9$ .

For  $M = \{1, 3, 9\}$ ,

$$LSE = \{0.134121, 0.002954, 0.002859\}$$

$$MSE = \{0.001341, 2.953798 \times 10^{-5}, 2.858653 \times 10^{-5}\}$$

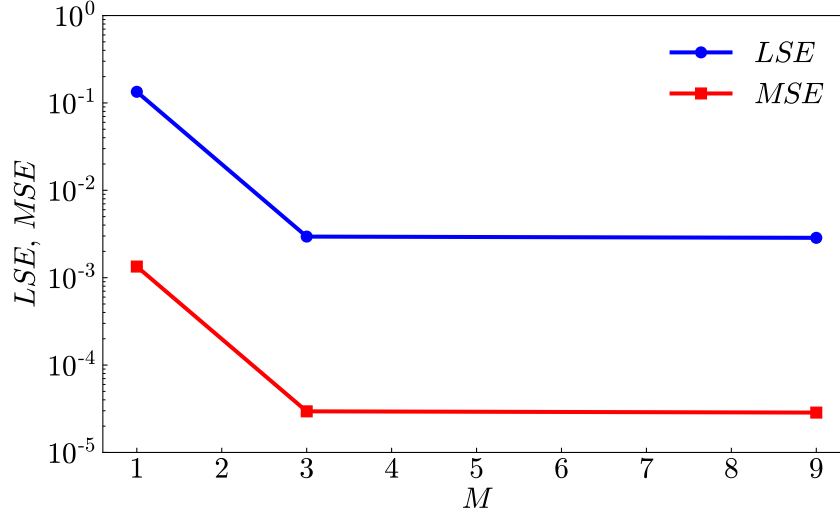


Figure 3: LSE and MSE for  $M = \{1, 3, 9\}$ .

3. Computed  $MSE$  for  $M = \{1, 3, 9\}$ :

For training data (80%),

$$MSE = \{0.001347, 2.798744 \times 10^{-5}, 2.658466 \times 10^{-5}\}$$

For testing data (20%),

$$MSE = \{0.001348, 3.691079 \times 10^{-5}, 3.933554 \times 10^{-5}\}$$

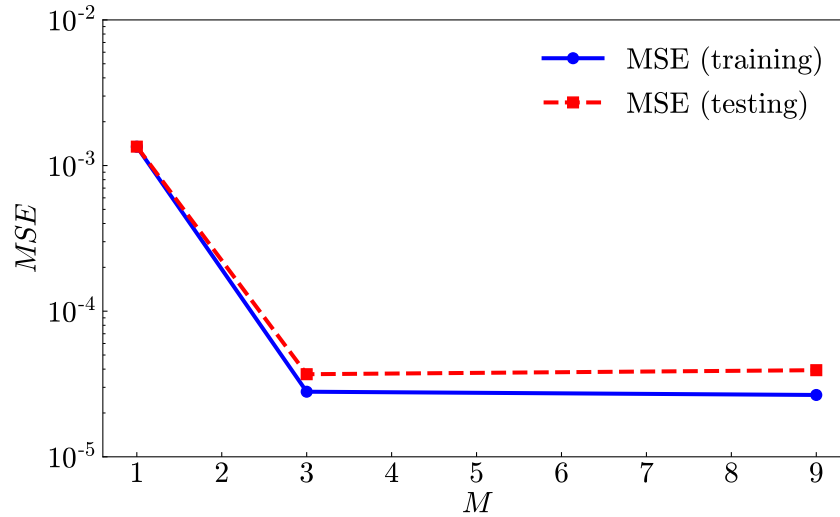


Figure 4:  $MSE$  for  $M = \{1, 3, 9\}$ .

4. Condition no. for  $M = \{1, 3, 9\}$ :

when  $N = 10$ ,

$$\{4.043212, 208.4386, 3.23 \times 10^9\}$$

when  $N = 40$ ,

$$\{4.377483, 106.0575, 5.09 \times 10^6\}$$

when  $N = 100$ ,

$$\{4.348661, 121.0929, 3.72 \times 10^6\}$$



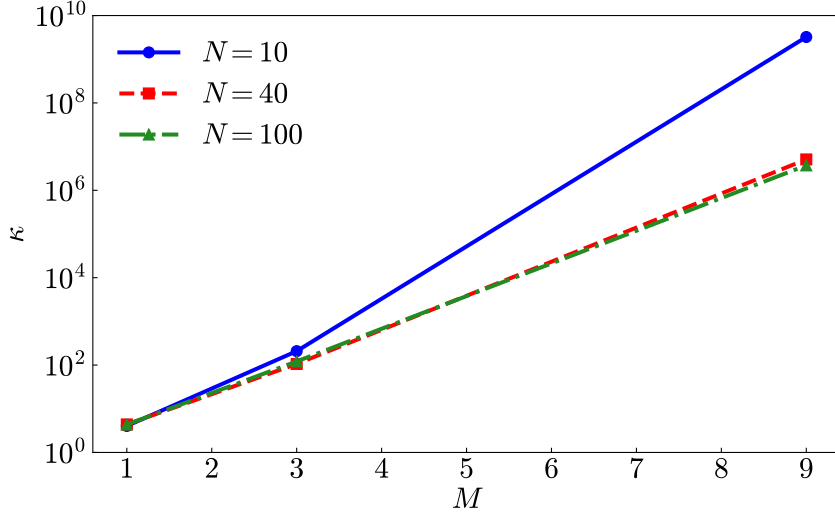


Figure 5: Condition number of least squares problem with different subsample sizes.

A large condition number means that the system is ill-conditioned, i.e., small perturbation in the "input" ( $A$  or  $\vec{y}$ ) can translate to a huge change in "output" ( $\vec{\omega}$ ). For a system to be stable, condition number should be as low as possible. Since,  $\kappa \geq 1$ , condition number closer to 1 is the most desirable.

From fig. 5,  $M = 9$  is ill-conditioned for all subsample sizes, where interpolation case ( $N = 10, M = 9$ ) is the worst.  $M = 1$  shows the best possible cases with  $\kappa$  being close to 1. This is a very well-conditioned system for all values of  $N$ .  $M = 3$  also shows that the system is moderate to somewhat well conditioned.  $N = 40$  shows the best condition number for  $M = 3$ . It is best to keep  $M$  as small as possible than  $N$  (interpolation case being an extreme example). As a result, purely from the point of view of conditioning, the reasonable value from these sets of  $M$  and  $N$  values should be  $M = \{1, 3\}$  and  $N = \{40, 100\}$ , since  $N = 10$  is a very small subsample size.

- From fig. 4, it is evident that  $M = 1$  has the largest  $MSE$  both in training and testing, while  $M = 9$  has slightly less training  $MSE$  than  $M = 3$ , but  $M = 3$  has slightly less

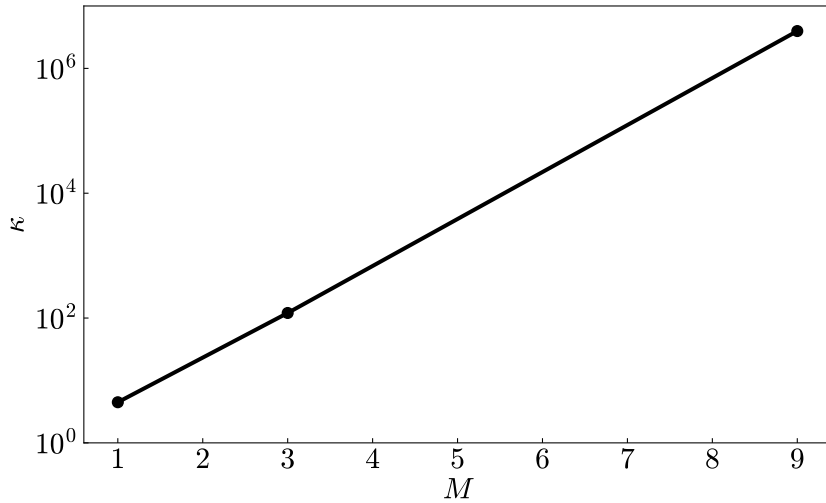


Figure 6: Condition number of least squares problem with 80% training data.

testing  $MSE$  than  $M = 9$ . This suggests that  $M = 9$  overfits the data compared to  $M = 3$  because of its higher testing  $MSE$  but slightly less training  $MSE$ . As a result, it can be concluded that  $M = 3$  shows the lowest test error, hence best accuracy of the three.

From fig. 6, it can be seen that  $M = 1, 3, 9$  account for well, moderate, ill-conditioned system respectively. As a result,  $M = 1$  has the best condition number but it underfits.  $M = 9$  has the worst condition number and it slightly overfits. Since,  $M = 3$  has moderate condition number and it shows the best accuracy, this is the best regression model of the three with good generalization overall.

From previous part, it is shown that  $N \geq 40$  shows good stability for the linear system. It can also be shown that  $MSE$  for 40% can show good accuracy with low testing  $MSE$  ( $O(\sim 10^{-5})$ ). Hence,  $M = 3$  and  $N \geq 40$  can be reasonable choices for balancing out the accuracy of the model and stability of the system.

### A. Class competition

Regularized error function (Ridge regression):

$$\tilde{E} = \|\vec{y} - A\vec{\omega}\|_2^2 + \lambda\|\vec{\omega}\|^2$$

To minimize this:

$$\begin{aligned}\nabla_{\vec{\omega}} \tilde{E} &= 0 \\ \Rightarrow \nabla_{\vec{\omega}} (\|\vec{y} - A\vec{\omega}\|_2^2 + \lambda\|\vec{\omega}\|^2) &= 0 \\ \Rightarrow 2A^T A\vec{\omega} - 2A^T \vec{y} + 2\lambda\vec{\omega} &= 0 \\ \Rightarrow \vec{\omega} &= (A^T A + \lambda I)^{-1} A^T \vec{y}\end{aligned}$$

where,  $\lambda (= 10^{-7})$  is a regularization parameter.

Estimated  $\vec{\omega}$  by regularization:

$$\vec{\omega} = \begin{bmatrix} -0.00100183858 \\ 2.02985641 \\ -0.284064707 \\ 0.807690825 \\ -2.09588975 \\ 1.08734780 \\ 1.33200525 \\ -0.613796398 \\ -1.81845028 \\ 1.24291928 \end{bmatrix}$$

$$MSE = 2.885477 \times 10^{-5}$$

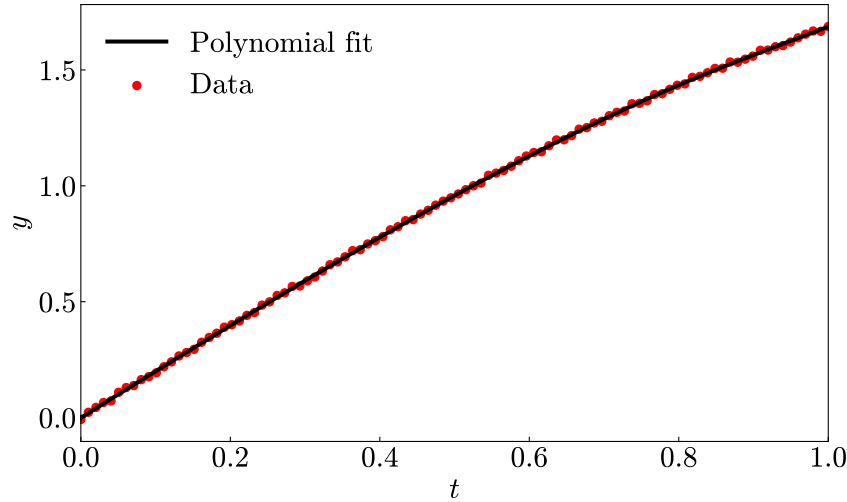


Figure 7: Estimated polynomial fit for true data.

```

1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import matplotlib as mpl
5 from matplotlib.ticker import MultipleLocator
6 from sympy import symbols, Eq, plot
7 #import statsmodels.api as sm
8 #from sklearn.linear_model import RANSACRegressor, LinearRegression
9
10 # ===== 1. Vandermonde matrix =====
11
12 #function to construct Vandermonde matrix
13 def vandermonde(time_sample, degree):
14     matrix_col = []
15     for m in range(degree+1):
16         matrix_col.append(time_sample ** m)
17     matrix = pd.concat(matrix_col, axis=1)
18     return matrix
19
20 # ===== 2. Least Squared Regression =====
21
22 #input arguments
23 filename = "hw1.csv"
24 degree = [1, 3, 9]
25
26 x_list = {} #co-efficients
27 lse_list = {} #least squares error (LSE)
28 mse_list = {} #mean squared error (MSE)
29
30 col1 = pd.read_csv(filename, usecols=[0], header=None); #time samples
31 col2 = pd.read_csv(filename, usecols=[1], header=None); #target vector
32
33 for i in degree:
34     vandermonde_ = vandermonde(col1, i)
35     x = (np.linalg.inv(vandermonde_.T @ vandermonde_) @ vandermonde_.T @ col2).
36         to_numpy().flatten() # $\omega = (A^T A)^{-1} A^T \text{vec}\{y\}$ 
37     y = col2.to_numpy().flatten()
38     lse = np.sum((abs(y - vandermonde_ @ x))**2) # $\sum\{\|\text{left}/\text{vec}\{y\} - A \backslash$ 
39         omega \right/\}^2\}
40     mse = np.mean((abs(y - vandermonde_ @ x))**2) # $\frac{1}{N} \sum\{\|\text{left}/\text{vec}\{y\} - A \backslash$ 

```

```

39     {y} - A\omega \right/}^2}$
40     x_list[i] = x #update co-efficient list
41     lse_list[i] = lse #update LSE list
42     mse_list[i] = mse #update MSE list
43 print('Co-efficients:', x_list)
44 print('LSE:', lse_list)
45 print('MSE:', mse_list)
46
47 #function to construct polynomial equation from co-efficients
48 def linear_reg(coeffs):
49     x_ = symbols("x");
50     y_ = 0;
51     for i, a in enumerate(coeffs):
52         y_ += a * x_ ** i;
53     return Eq(symbols('y'), y_);
54
55 eq_list = {} #equations
56 for i in x_list:
57     coeffs = x_list[i]
58     print(f"Coefficients: {coeffs}")
59     equation = linear_reg(coeffs)
60     eq_list[i] = equation
61 print(eq_list)
62
63 #parameters for plotting
64 plt.rcParams['font.family'] = 'serif'
65 plt.rcParams['font.serif'] = 'cmr10'
66 plt.rcParams['mathtext.fontset'] = 'cm'
67 plt.rcParams['font.size'] = 22
68 mpl.rcParams['axes.unicode_minus'] = False
69
70 #plotting given data and least squares regression line for each polynomial
71 #degree
72 for i, equation in eq_list.items():
73     fig, ax = plt.subplots(figsize=(8, 6))
74     plt.scatter(col1, col2, color='red', label='Data')
75     eq_plot = plot(equation.rhs, (symbols("x"), col1.to_numpy().flatten().min(),
76                                     col1.to_numpy().flatten().max()), show=False)
77
78     eq_plot[0].line_color = 'black'
79     eq_plot[0].line_width = 3
80     eq_plot[0].label = f'Polynomial degree-{i}'
81
82     for line in eq_plot:
83         ax.plot(*line.get_points(), color=line.line_color,
84                 linewidth=line.line_width, linestyle='-',
85                 label=line.label)
86
87     plt.xlabel('$t$')
88     plt.ylabel('$y$')
89     plt.xlim(col1.to_numpy().flatten().min(), col1.to_numpy().flatten().max())
90     plt.legend(frameon=False)
91     plt.tick_params(axis="both", which="both", direction="in")
92     plt.savefig(f'regress_deg{i}.pdf', dpi=1080)
93     plt.show()
94
95 #plotting LSE and MSE for each polynomial degree
96 lse = [lse_list[i] for i in degree]
97 mse = [mse_list[i] for i in degree]
98 fig, ax = plt.subplots(figsize=(10, 6))
99 ax.semilogy(degree, lse, color='blue', linestyle='-', linewidth=3, marker='o',
100             markersize=8, label='$LSE$')

```

```

98 ax.semilogy(degree, mse, color='red', linestyle='--', linewidth=3, marker = 's',
    markersize=8, label='$MSE$')
99 #plt.xlim(0, 10)
100 ax.xaxis.set_major_locator(MultipleLocator(1))
101 plt.ylim(10**(-5), 10**0)
102 plt.xlabel('$M$')
103 plt.ylabel('$LSE$, $MSE$')
104 plt.legend(frameon=False)
105 plt.tick_params(axis="both", which="both", direction="in")
106 plt.savefig('lse_mse.pdf', dpi=1080)
107 plt.show()
108
109 # ===== 3. Training & testing data MSE =====
110
111 #preparing data
112 data = np.column_stack((col1.to_numpy().flatten(), col2.to_numpy().flatten()))
113 np.random.seed(32) #for reproducibility
114 np.random.shuffle(data) #shuffling positions
115 split = int(0.8 * len(data)) #80% training, 20% testing
116 train, test = data[:split], data[split:]
117 t_train, y_train = train[:,0], train[:,1]
118 t_test, y_test = test[:,0], test[:,1]
119
120 mse_train_list = {} #training MSE
121 mse_test_list = {} #testing MSE
122 cond_no_train_list = {} #training condition no.
123
124 for i in degree:
125     vndrmnd_train = vandermonde(pd.DataFrame(t_train), i) #Vandermonde matrix
        for training data
126     vndrmnd_test = vandermonde(pd.DataFrame(t_test), i) #Vandermonde matrix for
        testing data
127     x_train = (np.linalg.inv(vndrmnd_train.T @ vndrmnd_train) @ vndrmnd_train.T
        @ pd.DataFrame(y_train)).to_numpy().flatten() #$\omega$ using training
        data
128     y_train_predict = vndrmnd_train @ x_train #predicted target vector for
        training data
129     y_test_predict = vndrmnd_test @ x_train #predicted target vector for
        testing data
130     mse_train = np.mean((abs(y_train_predict - y_train))**2)
131     mse_train_list[i] = mse_train #update traing MSE list
132     mse_test = np.mean((abs(y_test_predict - y_test))**2)
133     mse_test_list[i] = mse_test #update testing MSE list
134     cond_no_train = np.linalg.cond(vndrmnd_train, 2) #$\|A\|_2 \|A^+\|_2$
135     cond_no_train_list[i] = cond_no_train #update condition no. list
136
137 print('MSE (training):', mse_train_list)
138 print('MSE (testing):', mse_test_list)
139 print('Condition no. (training):', cond_no_train_list)
140
141 # ===== 4. Condition number =====
142
143 N = [10, 40, 100] #subsample size
144 cond_no_list = {} #condition no.
145
146 for i in degree:
147     cond_no_list[i] = {}
148     for j in N:
149         if j > len(data):
150             continue
151         subsample = np.random.choice(len(data), size=j, replace=False)
152         data_ = data[subsample]
153         vndrmnd = vandermonde(pd.DataFrame(data_[:,0]), i)

```

```

154         #print(np.linalg.inv(vndrmnd))
155         cond_no = np.linalg.cond(vndrmnd, 2) # $\|A\|_2 \|A^+\|_2$ 
156         cond_no_list[i][j] = cond_no #update condition no. list
157
158     print('Condition no.:')
159     for i in cond_no_list:
160         print(f'degree-{i}:')
161         for j in cond_no_list[i]:
162             print(f'N = {j}: {cond_no_list[i][j]: .6e}')
163
164     #plotting condition no. for different subsample sizes
165     fig, ax = plt.subplots(figsize=(10, 6))
166     mrkr = ['o', 's', '^']
167     ls = ['-', '--', '-.']
168     clr = ['blue', 'red', 'forestgreen']
169     for sym, j in enumerate(N):
170         x = []
171         y = []
172         for i in degree:
173             if j in cond_no_list[i]:
174                 x.append(i)
175                 y.append(cond_no_list[i][j])
176         ax.semilogy(x, y, color=clr[sym%len(clr)], linestyle=ls[sym%len(ls)],
177                    linewidth=3, marker=mrkr[sym%len(mrkr)], markersize=8, label=f'$N = {j}$')
178     plt.xlabel(r'$M$')
179     ax.xaxis.set_major_locator(MultipleLocator(1))
180     plt.ylabel(r'$\kappa$')
181     plt.ylim(10**(0), 10**(10))
182     plt.legend(frameon=False)
183     plt.tick_params(axis="both", which="both", direction="in")
184     plt.savefig('cond_no.pdf', dpi=1080)
185     plt.show()
186
187     # ===== 5. MSE and condition number plots =====
188
189     #plotting MSE for training and testing data
190     mse_train = np.array([mse_train_list[i] for i in degree])
191     mse_test = np.array([mse_test_list[i] for i in degree])
192
193     fig, ax = plt.subplots(figsize=(10, 6))
194     ax.semilogy(degree, mse_train, color='blue', linestyle='-', linewidth=3, marker='o', markersize=8, label='MSE (training)')
195     ax.semilogy(degree, mse_test, color='red', linestyle='--', linewidth=3, marker='s', markersize=8, label='MSE (testing)')
196     ax.xaxis.set_major_locator(MultipleLocator(1))
197     plt.ylim(10**(-5), 10**(-2))
198     plt.xlabel(r'$M$')
199     plt.ylabel(r'$MSE$')
200     plt.legend(frameon=False)
201     plt.tick_params(axis="both", which="both", direction="in")
202     plt.savefig('mse.pdf', dpi=1080)
203     plt.show()
204
205     #plotting condition no. for training data
206     cond_no_train = np.array([cond_no_train_list[i] for i in degree])
207
208     fig, ax = plt.subplots(figsize=(10, 6))
209     ax.semilogy(degree, cond_no_train, color='black', linestyle='-', linewidth=3, marker='o', markersize=8)
210     ax.xaxis.set_major_locator(MultipleLocator(1))
211     plt.ylim(10**(0), 10**(7))
212     plt.xlabel(r'$M$')

```

```

212 plt.ylabel(r'$\kappa$')
213 plt.legend(frameon=False)
214 plt.tick_params(axis="both", which="both", direction="in")
215 plt.savefig('cond_no_poly.pdf', dpi=1080)
216 plt.show()
217 # ===== Class competition =====
218
219 #residual computation
220 y_predict = vandermonde(col1, degree[2]).to_numpy() @ np.array(x_list[degree
    [2]])
221 #print(y_predict.shape)
222 residue = col2.to_numpy().flatten() - y_predict
223 print(f'Residual: ', residue)
224 print((col2.to_numpy().flatten()).shape)
225 #print(residue.shape)
226 plt.scatter(col2.to_numpy().flatten(), residue)
227 plt.ylim(-0.02, 0.02)
228
229 #closed form regularization (Ridge regression)
230 lam = 1e-7 #regularization parameter
231 x_reg = (np.linalg.inv(vandermonde(col1, degree[2]).T @ vandermonde(col1,
    degree[2]) + lam * np.eye(vandermonde(col1, degree[2]).shape[1])) @
    vandermonde(col1, degree[2]).T @ col2.to_numpy().flatten() ##\omega = (A^TA
    + \lambda I)^{-1}A^T\vec{y}$
232 x_reg[np.abs(x_reg) < 1e-6] = 0 #threshold for co-efficient
233 print(x_reg)
234 eq_reg = linear_reg(x_reg) #convert co-efficient to polynomial equation
235 print(eq_reg)
236 mse_reg = np.mean((abs(col2.to_numpy().flatten() - vandermonde(col1, degree[2])
    .to_numpy() @ x_reg)**2) #MSE
237 print(f'Regularized MSE: ', mse_reg)
238
239 #plotting noisy data and estimated actual polynomial
240 fig, ax = plt.subplots(figsize=(10, 6))
241 plt.scatter(col1, col2, color='red', label='Data')
242 eq_plot = plot(eq_reg.rhs, (symbols("x"), col1.to_numpy().flatten().min(), col1
    .to_numpy().flatten().max()), show=False)
243
244 eq_plot[0].line_color = 'black'
245 eq_plot[0].line_width = 3
246 eq_plot[0].label = 'Polynomial fit'
247
248 for line in eq_plot:
249     ax.plot(*line.get_points(), color=line.line_color,
250             linewidth=line.line_width, linestyle='--',
251             label=line.label)
252 plt.xlabel('$t$')
253 plt.ylabel('$y$')
254 plt.xlim(col1.to_numpy().flatten().min(), col1.to_numpy().flatten().max())
255 plt.legend(frameon=False)
256 plt.tick_params(axis="both", which="both", direction="in")
257 plt.savefig(f'ploy_fit.pdf', dpi=1080)
258 plt.show()

```

Listing 1: *hw1.py*