MTH 602 Scientific Machine Learning

Homework 1

9/22/2025

S. M. Mahfuzul Hasan

02181922

UMass | Dartmouth

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)

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

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

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

Direction: along (-)ve x_1 axis.
Magnitude: $\|\vec{F}(1,0,0)\|_2 = \sqrt{(-1)^2} = 1$

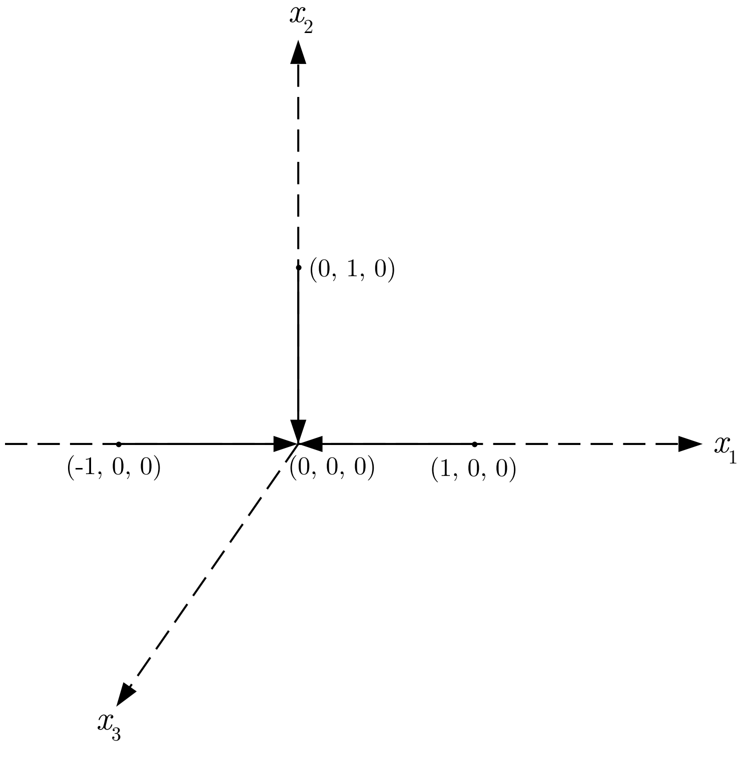

Figure 1: Sketch of $\vec{F}$ at (1, 0, 0), (-1, 0, 0) and (0, 1, 0).

At (-1, 0, 0),

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

Direction:	along (+)ve x_1 axis.
Magnitude:	$\|\vec{F}(-1,0,0)\|_2 = \sqrt{1^2} = 1$

At (0, 1, 0),

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

Direction:	along (-)ve x_2 axis.
Magnitude:	$\|\vec{F}(0,1,0)\|_2 = \sqrt{(1)^2} = 1$

2. Given,

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

(a)

$$\text{L.H.S} = \langle Q\vec{x}, Q\vec{y}\rangle = (Q\vec{x})^T(Q\vec{y}) = \left(\vec{x}^TQ^T\right)(Q\vec{y}) = \vec{x}^T\left(Q^TQ\right)\vec{y}$$
$$= \vec{x}^TI\vec{y} = \vec{x}^T\vec{y} = \langle \vec{x},\vec{y}\rangle = \text{R.H.S} \quad \text{(Showed)}$$

(b)

$$\|Q\vec{x}\|_2 = \sqrt{(Q\vec{x})^T(Q\vec{x})} = \sqrt{(\vec{x}^TQ^T)(Q\vec{x})} = \sqrt{\vec{x}^T(Q^TQ)\vec{x}}$$
$$= \sqrt{\vec{x}^TI\vec{x}} = \sqrt{\vec{x}^T\vec{x}} = \|\vec{x}\|_2$$

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,

$$\|\vec{x}\|_\infty \le \|\vec{x}\|_2$$
$$\|\vec{x}\|_2 \le \sqrt{N}\|\vec{x}\|_\infty$$

(a)

$$\|\vec{x}\|_\infty = \max_{1\le i\le N}|x_i| \tag{1}$$

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 (**??**) becomes-

$$\|\vec{x}\|_\infty = |x_N| \tag{2}$$

Now,

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

(b) We know,

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

Let's assume again that the maximum absolute valued element is x_N. Then equation (**??**) 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)

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

Note: X and Y are representative matrices.

(b)

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

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

So, equation (**??**) becomes-

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

5. Projection matrix is defined as,

$$P = A\left(A^TA\right)^{-1}A^T$$

(a)

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

(b)

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

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

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

Now,

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

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

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

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

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

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^TA 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 $\omega_j = \frac{a_j^T\vec{y}}{a_j^Ta_j}$ (where, a_j is j-th feature and ω_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 **??**.
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 $M = 9$,

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

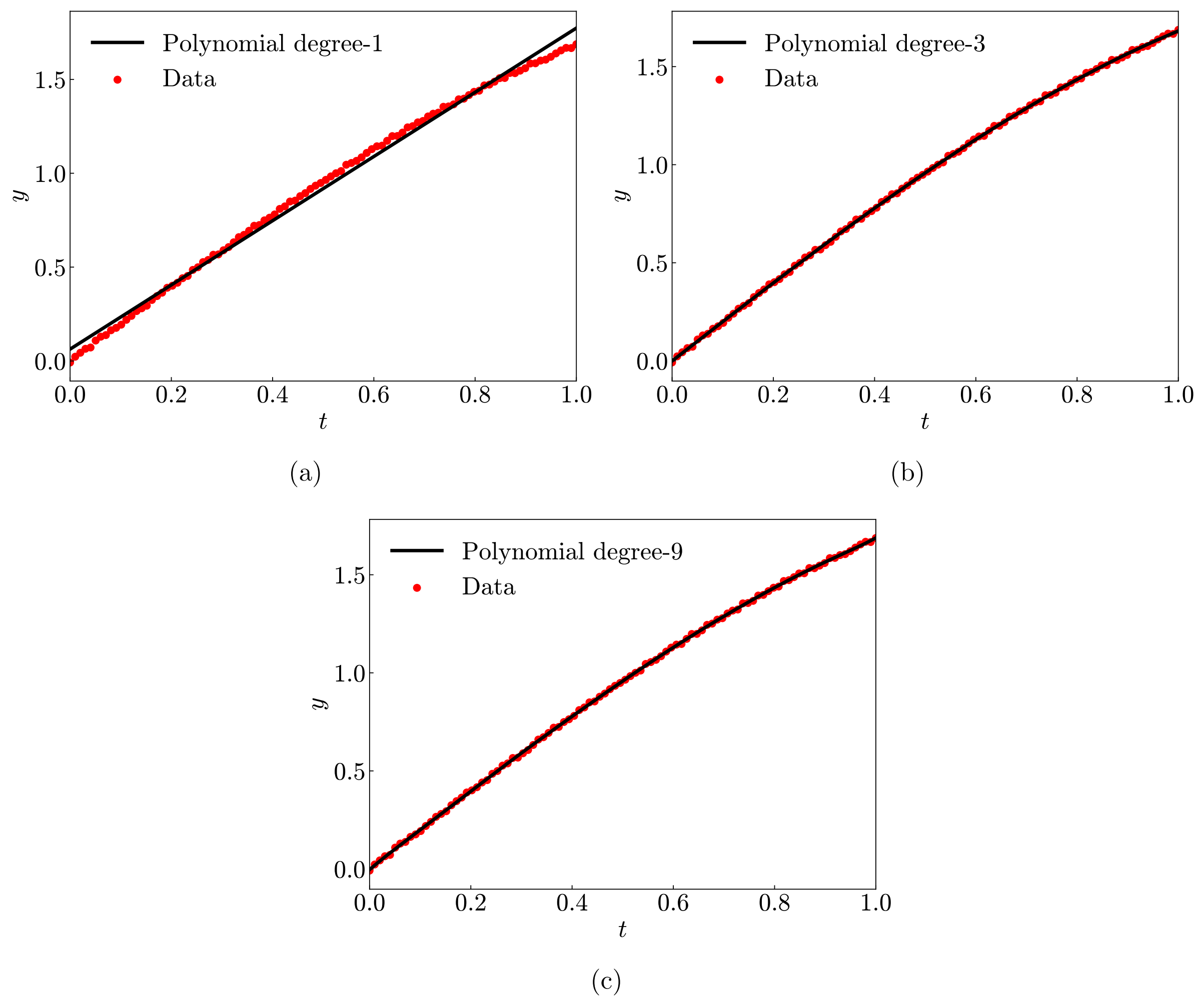

Figure 2: Polynomial regression fit for (a) $M = 1$, (b) $M = 3$, and (c) $M = 9$.

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

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

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

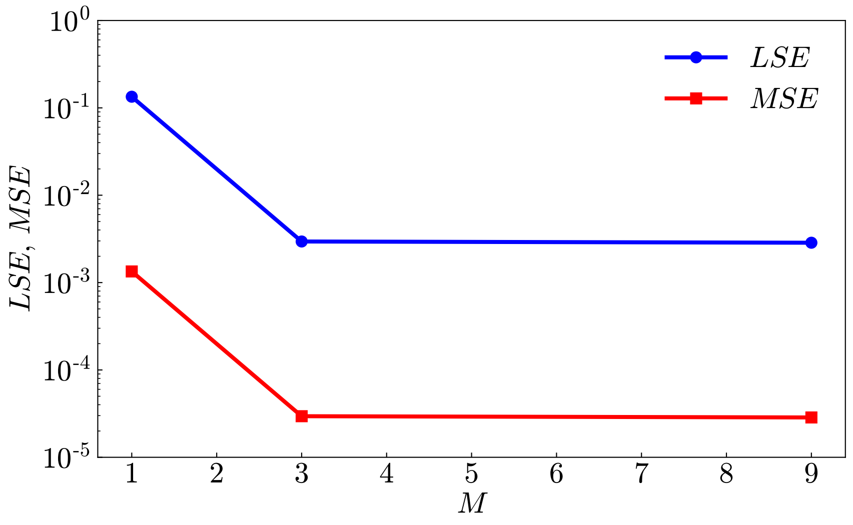

Figure 3: LSE and MSE for $M = \{1, 3, 9\}$.

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%),

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

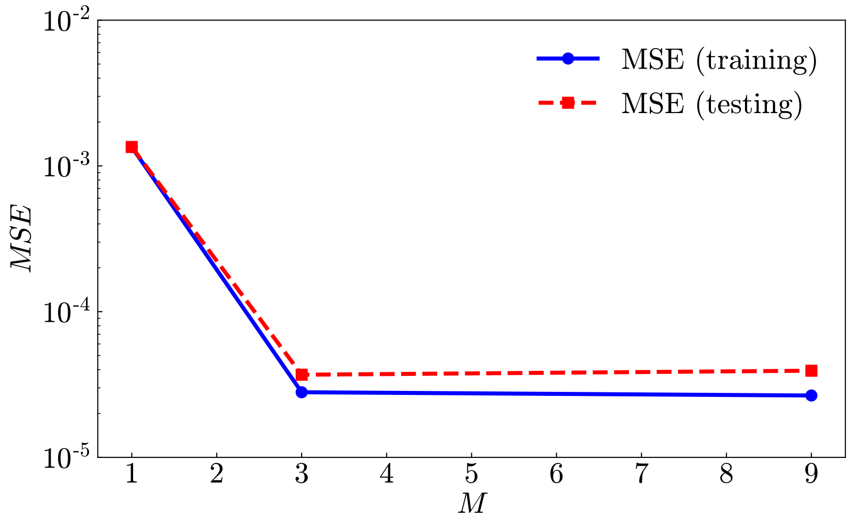

Figure 4: MSE for $M = \{1, 3, 9\}$.

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 $N = 100$,

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

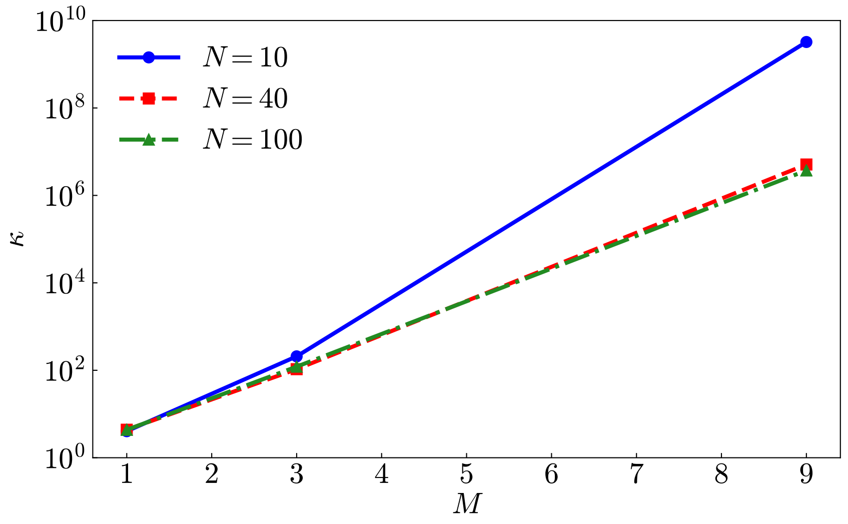

Figure 5: Condition number of least squares problem with different subsample sizes.

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. **??**, $M = 9$ 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 κ 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.

5. From fig. **??**, 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

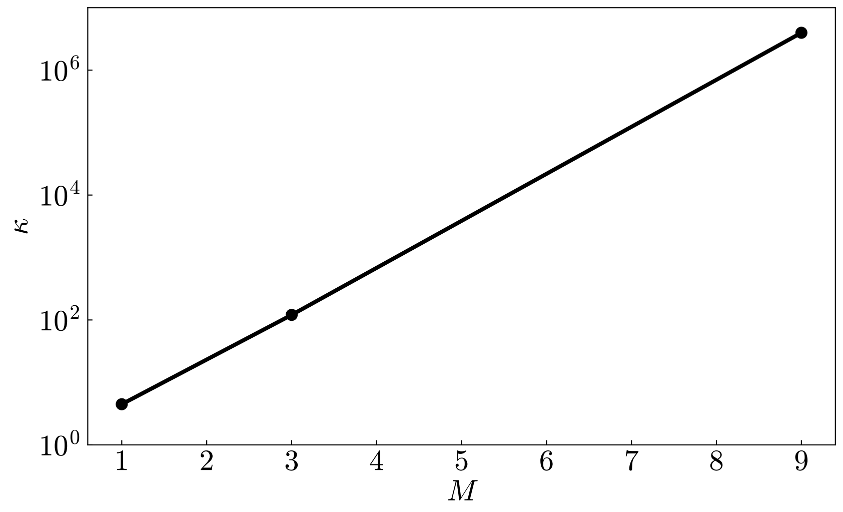

Figure 6: Condition number of least squares problem with 80% training data.

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. **??**, 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 accurcay 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):

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

To minimize this:

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

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

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

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

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

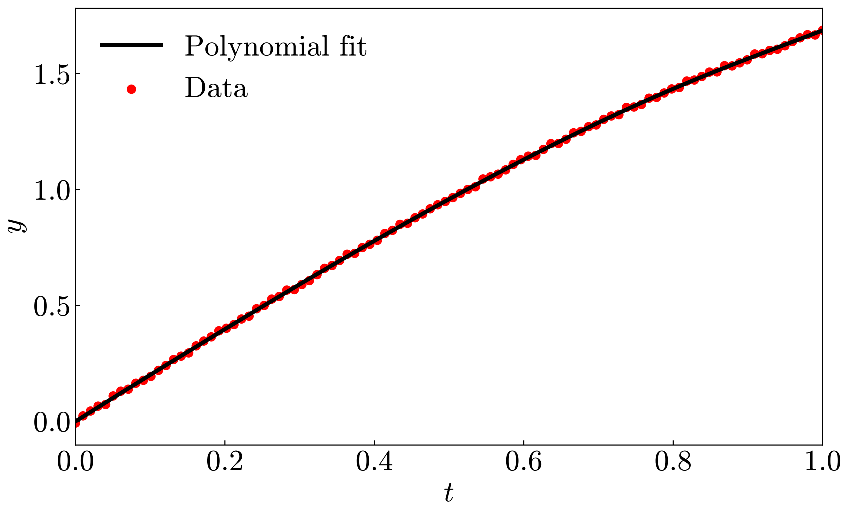

Figure 7: Estimated polynomial fit for true data.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
from matplotlib.ticker import MultipleLocator
from sympy import symbols, Eq, plot
#import statsmodels.api as sm
#from sklearn.linear_model import RANSACRegressor, LinearRegression

# ===== 1. Vandermonde matrix =====

#function to construct Vandermonde matrix
def vandermonde(time_sample, degree):
    matrix_col = []
    for m in range(degree+1):
        matrix_col.append(time_sample ** m)
    matrix = pd.concat(matrix_col, axis=1)
    return matrix

# ===== 2. Least Squared Regression =====

#input arguments
filename = "hw1.csv"
degree = [1, 3, 9]

x_list = {} #co-efficients
lse_list = {} #least squares error (LSE)
mse_list = {} #mean squared error (MSE)

col1 = pd.read_csv(filename, usecols=[0], header=None); #time samples
col2 = pd.read_csv(filename, usecols=[1], header=None); #target vector

for i in degree:
    vandermonde_ = vandermonde(col1, i)
    x = (np.linalg.inv(vandermonde_.T @ vandermonde_) @ vandermonde_.T @ col2).
        to_numpy().flatten() #$\omega = (A^TA)^{-1}A^T\vec{y}$
    y = col2.to_numpy().flatten()
    lse = np.sum((abs(y - vandermonde_ @ x))**2) #$\sum{{\left|\vec{y} - A\
        omega \right|}^2}$
    mse = np.mean((abs(y - vandermonde_ @ x))**2) #$\frac{1}{N}\sum{{\left|\vec
```

```
        {y} - A\omega \right|}^2}$
    x_list[i] = x #update co-efficient list
    lse_list[i] = lse #update LSE list
    mse_list[i] = mse #update MSE list
print('Co-efficients:', x_list)
print('LSE:', lse_list)
print('MSE:', mse_list)

#function to construct polynomial equation from co-efficients
def linear_reg(coeffs):
    x_ = symbols("x");
    y_ = 0;
    for i, a in enumerate(coeffs):
        y_ += a * x_ ** i;
    return Eq(symbols('y'), y_);

eq_list = {} #equations
for i in x_list:
    coeffs = x_list[i]
    print(f"Coefficients: {coeffs}")
    equation = linear_reg(coeffs)
    eq_list[i] = equation
print(eq_list)

#parameters for plotting
plt.rcParams['font.family'] = 'serif'
plt.rcParams['font.serif'] = 'cmr10'
plt.rcParams['mathtext.fontset'] = 'cm'
plt.rcParams['font.size'] = 22
mpl.rcParams['axes.unicode_minus'] = False

#plotting given data and least squares regression line for each polynomial
    degree
for i, equation in eq_list.items():
    fig, ax = plt.subplots(figsize=(8, 6))
    plt.scatter(col1, col2, color='red', label='Data')
    eq_plot = plot(equation.rhs, (symbols("x"), col1.to_numpy().flatten().min()
        ,
                                  col1.to_numpy().flatten().max()), show=False)

    eq_plot[0].line_color = 'black'
    eq_plot[0].line_width = 3
    eq_plot[0].label = f'Polynomial degree-{i}'

    for line in eq_plot:
        ax.plot(*line.get_points(), color=line.line_color,
                linewidth=line.line_width, linestyle='-',
                label=line.label)

    plt.xlabel('$t$')
    plt.ylabel('$y$')
    plt.xlim(col1.to_numpy().flatten().min(), col1.to_numpy().flatten().max())
    plt.legend(frameon=False)
    plt.tick_params(axis="both", which="both", direction="in")
    plt.savefig(f'regress_deg{i}.pdf', dpi=1080)
    plt.show()

#plotting LSE and MSE for each polynomial degree
lse = [lse_list[i] for i in degree]
mse = [mse_list[i] for i in degree]
fig, ax = plt.subplots(figsize=(10, 6))
ax.semilogy(degree, lse, color='blue', linestyle='-', linewidth=3, marker='o',
    markersize=8, label='$LSE$')
```

```python
ax.semilogy(degree, mse, color='red', linestyle='-', linewidth=3, marker = 's',
    markersize=8, label='$MSE$')
#plt.xlim(0, 10)
ax.xaxis.set_major_locator(MultipleLocator(1))
plt.ylim(10**(-5), 10**0)
plt.xlabel('$M$')
plt.ylabel('$LSE$, $MSE$')
plt.legend(frameon=False)
plt.tick_params(axis="both", which="both", direction="in")
plt.savefig('lse_mse.pdf', dpi=1080)
plt.show()

# ===== 3. Training & testing data MSE =====

#preparing data
data = np.column_stack((col1.to_numpy().flatten(), col2.to_numpy().flatten()))
np.random.seed(32) #for reproduciblity
np.random.shuffle(data) #shuffling positions
split = int(0.8 * len(data)) #80% training, 20% testing
train, test = data[:split], data[split:]
t_train, y_train = train[:,0], train[:,1]
t_test, y_test = test[:,0], test[:,1]

mse_train_list = {} #training MSE
mse_test_list = {} #testing MSE
cond_no_train_list = {} #training condition no.

for i in degree:
    vndrmnd_train = vandermonde(pd.DataFrame(t_train), i) #Vandermonde matrix
        for training data
    vndrmnd_test = vandermonde(pd.DataFrame(t_test), i) #Vandermonde matrix for
         testing data
    x_train = (np.linalg.inv(vndrmnd_train.T @ vndrmnd_train) @ vndrmnd_train.T
         @ pd.DataFrame(y_train)).to_numpy().flatten() #$\omega$ using training
        data
    y_train_predict = vndrmnd_train @ x_train #predicted traget vector for
        training data
    y_test_predict = vndrmnd_test @ x_train #predicted traget vector for
        testing data
    mse_train = np.mean((abs(y_train_predict - y_train))**2)
    mse_train_list[i] = mse_train #update traing MSE list
    mse_test = np.mean((abs(y_test_predict - y_test))**2)
    mse_test_list[i] = mse_test #update testing MSE list
    cond_no_train = np.linalg.cond(vndrmnd_train, 2) #$\|A\|_2 \|A^+\|_2$
    cond_no_train_list[i] = cond_no_train #update condition no. list

print('MSE (training):', mse_train_list)
print('MSE (testing):', mse_test_list)
print('Condition no. (training):', cond_no_train_list)

# ===== 4. Condition number =====

N = [10, 40, 100] #subsample size
cond_no_list = {} #condition no.

for i in degree:
    cond_no_list[i] = {}
    for j in N:
        if j > len(data):
            continue
        subsample = np.random.choice(len(data), size=j, replace=False)
        data_ = data[subsample]
        vndrmnd = vandermonde(pd.DataFrame(data_[:,0]), i)
        #print(np.linalg.inv(vndrmnd))
        cond_no = np.linalg.cond(vndrmnd, 2) #$\|A\|_2 \|A^+\|_2$
        cond_no_list[i][j] = cond_no #update condition no. list

print('Condition no.:')
for i in cond_no_list:
    print(f'degree-{i}:')
    for j in cond_no_list[i]:
        print(f'N = {j}: {cond_no_list[i][j]: .6e}')

#plotting condition no. for different subsample sizes
fig, ax = plt.subplots(figsize=(10, 6))
mrkr = ['o', 's', '^']
ls = ['-', '--', '-.']
clr = ['blue', 'red', 'forestgreen']
for sym, j in enumerate(N):
    x = []
    y = []
    for i in degree:
        if j in cond_no_list[i]:
            x.append(i)
            y.append(cond_no_list[i][j])
    ax.semilogy(x, y, color=clr[sym%len(clr)], linestyle=ls[sym%len(ls)],
       linewidth=3, marker=mrkr[sym%len(mrkr)], markersize=8, label=f'$N = {j}$
       ')
plt.xlabel(r'$M$')
ax.xaxis.set_major_locator(MultipleLocator(1))
plt.ylabel(r'$\kappa$')
plt.ylim(10**(0), 10**(10))
plt.legend(frameon=False)
plt.tick_params(axis="both", which="both", direction="in")
plt.savefig('cond_no.pdf', dpi=1080)
plt.show()

# ===== 5. MSE and condition number plots =====

#plotting MSE for training and testing data
mse_train = np.array([mse_train_list[i] for i in degree])
mse_test = np.array([mse_test_list[i] for i in degree])

fig, ax = plt.subplots(figsize=(10, 6))
ax.semilogy(degree, mse_train, color='blue', linestyle='-', linewidth=3, marker
    ='o', markersize=8, label='MSE (training)')
ax.semilogy(degree, mse_test, color='red', linestyle='--', linewidth=3, marker=
    's', markersize=8, label='MSE (testing)')
ax.xaxis.set_major_locator(MultipleLocator(1))
plt.ylim(10**(-5), 10**(-2))
plt.xlabel(r'$M$')
plt.ylabel(r'$MSE$')
plt.legend(frameon=False)
plt.tick_params(axis="both", which="both", direction="in")
plt.savefig('mse.pdf', dpi=1080)
plt.show()

#plotting condition no. for training data
cond_no_train = np.array([cond_no_train_list[i] for i in degree])

fig, ax = plt.subplots(figsize=(10, 6))
ax.semilogy(degree, cond_no_train, color='black', linestyle='-', linewidth=3,
    marker='o', markersize=8)
ax.xaxis.set_major_locator(MultipleLocator(1))
plt.ylim(10**(0), 10**(7))
plt.xlabel(r'$M$')
plt.ylabel(r'$\kappa$')
plt.legend(frameon=False)
plt.tick_params(axis="both", which="both", direction="in")
plt.savefig('cond_no_poly.pdf', dpi=1080)
plt.show()
# ===== Class competition =====

#residual computation
y_predict = vandermonde(col1, degree[2]).to_numpy() @ np.array(x_list[degree
    [2]])
#print(y_predict.shape)
residue = col2.to_numpy().flatten() - y_predict
print(f'Residual: ', residue)
print((col2.to_numpy().flatten()).shape)
#print(residue.shape)
plt.scatter(col2.to_numpy().flatten(), residue)
plt.ylim(-0.02, 0.02)

#closed form regularization (Ridge regression)
lam = 1e-7 #regularization parameter
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() #$\omega = (A^TA
     + \lambda I)^{-1}A^T\vec{y}$
x_reg[np.abs(x_reg) < 1e-6] = 0 #threshold for co-efficient
print(x_reg)
eq_reg = linear_reg(x_reg) #convert co-efficient to polynomial equation
print(eq_reg)
mse_reg = np.mean((abs(col2.to_numpy().flatten() - vandermonde(col1, degree[2])
    .to_numpy() @ x_reg))**2) #MSE
print(f'Regularized MSE: ', mse_reg)

#plotting noisy data and estimated actual polynomial
fig, ax = plt.subplots(figsize=(10, 6))
plt.scatter(col1, col2, color='red', label='Data')
eq_plot = plot(eq_reg.rhs, (symbols("x"), col1.to_numpy().flatten().min(), col1
    .to_numpy().flatten().max()), show=False)

eq_plot[0].line_color = 'black'
eq_plot[0].line_width = 3
eq_plot[0].label = 'Polynomial fit'

for line in eq_plot:
    ax.plot(*line.get_points(), color=line.line_color,
            linewidth=line.line_width, linestyle='-',
            label=line.label)
plt.xlabel('$t$')
plt.ylabel('$y$')
plt.xlim(col1.to_numpy().flatten().min(), col1.to_numpy().flatten().max())
plt.legend(frameon=False)
plt.tick_params(axis="both", which="both", direction="in")
plt.savefig(f'ploy_fit.pdf', dpi=1080)
plt.show()
```

Listing 1: *hw1.py*