



APL 104
AI-ML Project
Under the guidance of-Prof. Rajdip Nayek

November 15, 2024

1 References

Chatgpt,OpenAI

Gemini,Alphabet

2 Derivation of Governing Equation

We have derived the three governing equations from the 3 equations of stress equilibrium equation, 6 equation for stress-strain relation, 6 equations for strain-displacement conditions. The derivation is as follows-

Derivation

According to stress equilibrium equation:-

$$\frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{xy}}{\partial y} + \frac{\partial \sigma_{xz}}{\partial z} = 0$$

$$\frac{\partial \sigma_{yx}}{\partial x} + \frac{\partial \sigma_{yy}}{\partial y} + \frac{\partial \sigma_{yz}}{\partial z} = 0$$

$$\frac{\partial \sigma_{zx}}{\partial x} + \frac{\partial \sigma_{zy}}{\partial y} + \frac{\partial \sigma_{zz}}{\partial z} = 0$$

According to strain-displacement curve

$$\epsilon_{xx} = \frac{\partial u_x}{\partial x}, \epsilon_{yy} = \frac{\partial u_y}{\partial y}, \epsilon_{zz} = \frac{\partial u_z}{\partial z}$$

$$\epsilon_{xy} = \frac{1}{2} \left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \right), \epsilon_{xz} = \frac{1}{2} \left(\frac{\partial u_x}{\partial z} + \frac{\partial u_z}{\partial x} \right)$$

$$\epsilon_{yz} = \frac{1}{2} \left(\frac{\partial u_y}{\partial z} + \frac{\partial u_z}{\partial y} \right)$$

According to stress-strain relation :-

$$\sigma_{xx} = \lambda (E_{xx} + E_{yy} + E_{zz}) + 2\mu E_{xx}$$

$$\sigma_{yy} = \lambda (E_{xx} + E_{yy} + E_{zz}) + 2\mu E_{yy}$$

$$\sigma_{zz} = \lambda (E_{xx} + E_{yy} + E_{zz}) + 2\mu E_{zz}$$

$$\sigma_{xy} = 2\mu \epsilon_{xy}, \sigma_{xz} = 2\mu \epsilon_{xz}, \sigma_{yz} = 2\mu \epsilon_{yz}$$

Substituting stress-strain relation in stress equilibrium equation, we get.

$$\frac{\partial}{\partial x} (\lambda (E_{xx} + E_{yy} + E_{zz}) + 2\mu E_{xx}) + \frac{\partial}{\partial y} (2\mu \epsilon_{xy}) + \frac{\partial}{\partial z} (2\mu \epsilon_{xz}) = 0$$

Similarly for other 2 equations, we get

$$\frac{\partial}{\partial x} (2\mu \epsilon_{yx}) + \frac{\partial}{\partial y} (\lambda (E_{xx} + E_{yy} + E_{zz}) + 2\mu E_{yy}) + \frac{\partial}{\partial z} (2\mu \epsilon_{yz}) = 0$$

$$\frac{\partial}{\partial x} (2\mu \epsilon_{xz}) + \frac{\partial}{\partial y} (2\mu \epsilon_{yz}) + \frac{\partial}{\partial z} (\lambda (E_{xx} + E_{yy} + E_{zz}) + 2\mu E_{zz}) = 0$$

Substituting strain components, we get

$$(1+2\mu) \frac{\partial^2 u_x}{\partial x^2} + \mu \frac{\partial^2 u_x}{\partial y^2} + \mu \frac{\partial^2 u_x}{\partial z^2} + (\lambda + \mu) \left(\frac{\partial^2 u_y}{\partial x \partial y} + \frac{\partial^2 u_z}{\partial x \partial z} \right) = 0$$

$$(\lambda + \mu) \frac{\partial^2 u_y}{\partial x^2} + (1+2\mu) \frac{\partial^2 u_y}{\partial y^2} + \mu \frac{\partial^2 u_y}{\partial z^2} + (\lambda + \mu) \left(\frac{\partial^2 u_x}{\partial y \partial x} + \frac{\partial^2 u_z}{\partial y \partial z} \right) = 0$$

$$\mu \frac{\partial^2 u_z}{\partial x^2} + \mu \frac{\partial^2 u_z}{\partial y^2} + (1+2\mu) \frac{\partial^2 u_z}{\partial z^2} + (\lambda + \mu) \left(\frac{\partial^2 u_x}{\partial z \partial x} + \frac{\partial^2 u_y}{\partial z \partial y} \right) = 0$$

3 Parameter identification methodology

Objective Function: Find u_x and u_y using polynomial regression

$$\text{minimize} \sum_{i=1}^N [\text{residual}_{u_x}(x_i, y_i, \alpha, \mu)^2 + \text{residual}_{u_y}(x_i, y_i, \alpha, \mu)^2]$$

$$+ (x=0) [\lambda \left(\frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} \right) + 2\mu \frac{\partial u_x}{\partial x} - \text{abs}(R_1)]^2 \times 100$$

$$+ (y=0) [\lambda \left(\frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} \right) + 2\mu \frac{\partial u_y}{\partial y} - \text{abs}(R_2)]^2 \times 100 \quad (\text{penalty})$$

Physical Constraints

$$\alpha > 0 \text{ and } \mu > 0$$

- Algorithm for parameter identification
- Split the data from each file (60%40%)
 - Use Polynomial regression to find u_x and u_y Train set Test set
 - Find the further derivatives using differentiation.
 - Defining residuals from eqn calculated in point 1 (u_x, u_y)
 - Define Boundary conditions $(x=1, y=1)$
 - Defining the objective function.
 - Setting the bounds $1e-6$ to $1e12$ to ensure results to be valid for the given material
 - Using the 'Nelder-Mead' Numerical method to minimize the ~~cost~~ function
 - Evaluate by optimizing on training data and calculating the objective function and checking the fit

Logical decision for Nelder Mead Algorithm

If $f(R) < f(G)$, THEN perform case(i) [Reflect/Extend]
 ELSE perform case(ii) [Contract/shrink]

```

BEGIN {Case(i)}?
  IF f(B) < f(R) THEN
    replace W with R
  ELSE
    Compute E and f(E)
    IF f(E) < f(B) THEN
      replace W with E
    ELSE
      replace W with R
    ENDIF
  ENDIF
END {Case(i)}?
  
```

```

Begin {Case(ii)}?
  IF f(R) < f(W)
    replace W with R
    Compute C = (W+R)/2
    or C = (W+M)/2 and
    f(C)
    If f(C) < f(W)
    THEN replace W with C
  ELSE
    Compute S and
    f(S)
    replace W with S
    replace G with M
  ENDIF
END {Case(ii)}?
  
```

4 Results

Code to find out the polynomial equation for displacement

```

import numpy as np
import pandas as pd
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
# Load the CSV file
df1 = pd.read_csv(r'C:\Users\ragha\vscode\programming_cp\pythonn\data\1\displacement_data.csv')
# Step 1: Load the dataset
x = df1['x'].values
y = df1['y'].values
u_x = df1['u_x'].values
u_y = df1['u_y'].values
# Step 2: Split the dataset into 60% training and 40% testing
X = np.column_stack((x, y))
X_train, X_test, u_x_train, u_x_test, u_y_train, u_y_test =
train_test_split(X, u_x, u_y, test_size=0.4, random_state=42)
# Step 3: Prepare polynomial features (degree 3 for simplicity)
degree = 3
  
```

```

poly = PolynomialFeatures(degree=degree)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)
# Step 4: Fit the polynomial regression model for u_y
model_y = LinearRegression()
model_y.fit(X_train_poly, u_y_train)
# Step 4b: Fit the polynomial regression model for u_x
model_x = LinearRegression()
model_x.fit(X_train_poly, u_x_train)
# Step 5: Make predictions on the test set
u_y_pred = model_y.predict(X_test_poly)
u_x_pred = model_x.predict(X_test_poly)
# Step 6: Evaluate the models
mse_y = mean_squared_error(u_y_test, u_y_pred)
r2_y = r2_score(u_y_test, u_y_pred)
mse_x = mean_squared_error(u_x_test, u_x_pred)
r2_x = r2_score(u_x_test, u_x_pred)
print(f"u_y - Mean Squared Error (MSE): {mse_y:.5e}, R-squared (R2 Score): {r2_y:.5f}")
print(f"u_x - Mean Squared Error (MSE): {mse_x:.5e}, R-squared (R2 Score): {r2_x:.5f}")
# Step 7: Plot the predictions vs actual values for u_y
plt.figure(figsize=(10, 6))
plt.scatter(range(len(u_y_test)), u_y_test, color='blue', label='Actual u_y', alpha=0.6)
plt.scatter(range(len(u_y_pred)), u_y_pred, color='red', label='Predicted u_y', alpha=0.6)
plt.title("Comparison of Actual and Predicted Values for u_y")
plt.xlabel("Sample Index")
plt.ylabel("u_y")
plt.legend()
plt.show()
# Step 7b: Plot the predictions vs actual values for u_x
plt.figure(figsize=(10, 6))
plt.scatter(range(len(u_x_test)), u_x_test, color='green', label='Actual u_x', alpha=0.6)
plt.scatter(range(len(u_x_pred)), u_x_pred, color='orange', label='Predicted u_x', alpha=0.6)

```

```

plt.title("Comparison of Actual and Predicted Values for u_x")
plt.xlabel("Sample Index")
plt.ylabel("u_x")
plt.legend()
plt.show()
# Step 8: Display the polynomial equations
# For u_y
coefficients_y = model_y.coef_
intercept_y = model_y.intercept_
features = poly.get_feature_names_out(input_features=['x', 'y'])
equation_y = f"u_y = {intercept_y:.5f}"
for coef, feature in zip(coefficients_y, features):
    equation_y += f" + ({coef:.5e})*{feature}"
print("\nPolynomial Equation for u_y:")
print(equation_y)
# For u_x
coefficients_x = model_x.coef_
intercept_x = model_x.intercept_
equation_x = f"u_x = {intercept_x:.5f}"
for coef, feature in zip(coefficients_x, features):
    equation_x += f" + ({coef:.5e})*{feature}"
print("\nPolynomial Equation for u_x:")
print(equation_x)

```

The output is as follows:

u_y - Mean Squared Error (MSE): 5.48313e-11, R-squared (R2 Score):

0.95681

u_x - Mean Squared Error (MSE): 1.75159e-12, R-squared (R2 Score):

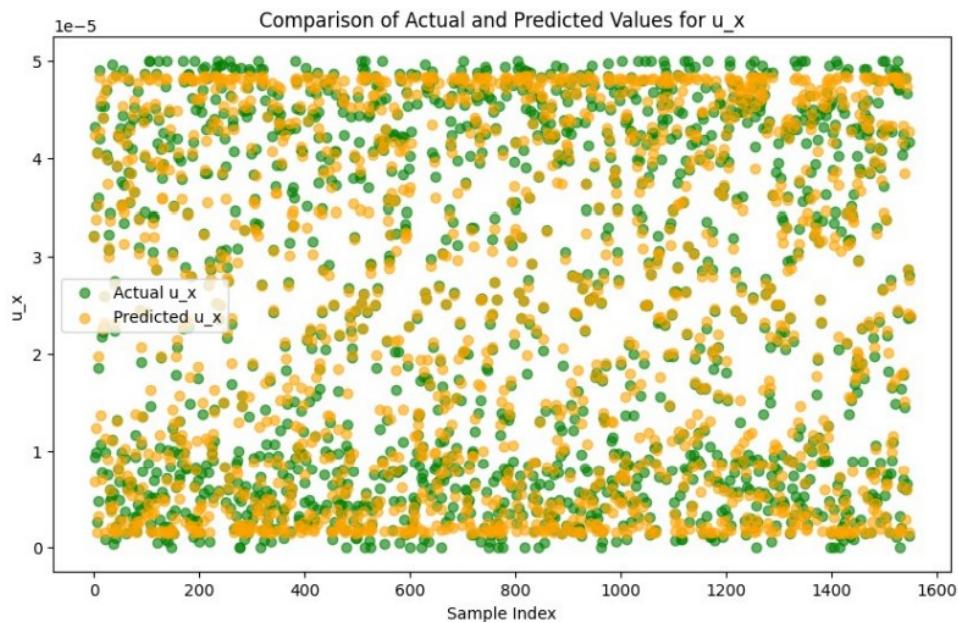
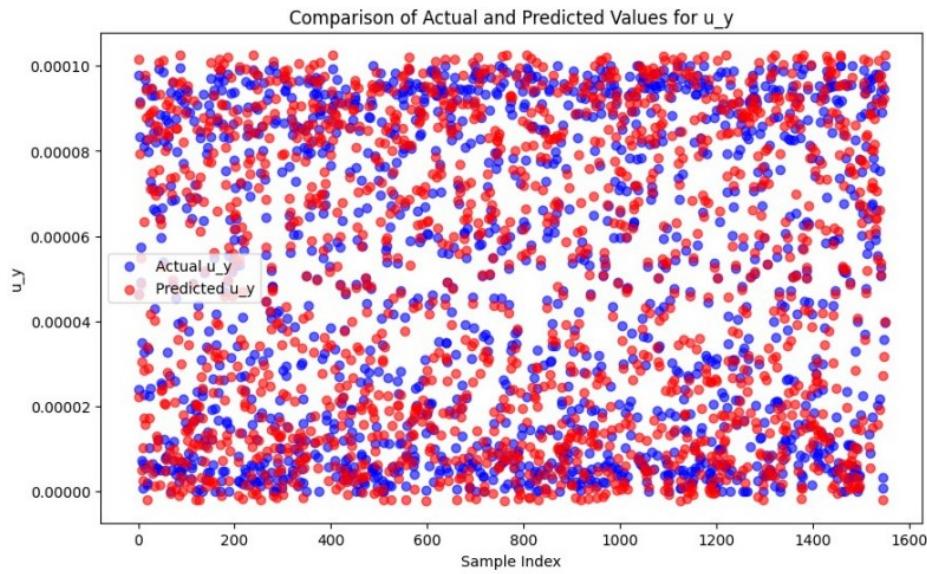
0.99425

Polynomial Equation for u_y:

$$\begin{aligned}
 u_y = & 0.00002 + (0.00000e+00)*1 + (-7.42272e-05)*x + (-5.29998e-05)*y \\
 & + (7.58230e-05)*x^2 + (1.50157e-04)*x y + (3.59174e-04)*y^2 + (- \\
 & 1.72507e-06)*x^3 + (-1.48567e-04)*x^2 y + (-2.41576e-06)*x y^2 + (- \\
 & 2.38374e-04)*y^3
 \end{aligned}$$

Polynomial Equation for u_x:

$$\begin{aligned}
 u_x = & 0.00000 + (0.00000e+00)*1 + (-1.37205e-05)*x + (5.20906e-07)*y + \\
 & (1.82015e-04)*x^2 + (-6.98688e-07)*x y + (-5.26740e-07)*y^2 + (- \\
 & 1.21757e-04)*x^3 + (-1.33234e-07)*x^2 y + (5.22233e-07)*x y^2 + \\
 & (1.79707e-07)*y^3
 \end{aligned}$$



The same step was repeated for all the data sets. **For data 2, the equations were:**

Polynomial Equation for u_y :

$$\begin{aligned}
 u_y = & 0.00003 + (0.00000e+00)*1 + (-1.48454e-04)*x + (-1.06000e-04)*y \\
 & + (1.51646e-04)*x^2 + (3.00314e-04)*x y + (7.18348e-04)*y^2 + (- \\
 & 3.45015e-06)*x^3 + (-2.97134e-04)*x^2 y + (-4.83151e-06)*x y^2 + (-
 \end{aligned}$$

4.76749e-04)*y^3

Polynomial Equation for u_x:

u_x = 0.00000 + (0.00000e+00)*1 + (-2.74409e-05)*x + (1.04181e-06)*y +
(3.64030e-04)*x^2 + (-1.39738e-06)*x y + (-1.05348e-06)*y^2 + (-
2.43514e-04)*x^3 + (-2.66467e-07)*x^2 y + (1.04447e-06)*x y^2 +
(3.59414e-07)*y^3

For data 3, the equations were:

Polynomial Equation for u_y:

u_y = 0.00005 + (0.00000e+00)*1 + (-2.22681e-04)*x + (-1.58999e-04)*y +
(2.27469e-04)*x^2 + (4.50471e-04)*x y + (1.07752e-03)*y^2 + (-
5.17522e-06)*x^3 + (-4.45701e-04)*x^2 y + (-7.24727e-06)*x y^2 + (-
7.15123e-04)*y^3

Polynomial Equation for u_x:

u_x = 0.00000 + (0.00000e+00)*1 + (-4.11614e-05)*x + (1.56272e-06)*y +
(5.46045e-04)*x^2 + (-2.09607e-06)*x y + (-1.58022e-06)*y^2 + (-
3.65271e-04)*x^3 + (-3.99701e-07)*x^2 y + (1.56670e-06)*x y^2 +
(5.39121e-07)*y^3

For data 4, the equations were:

Polynomial Equation for u_y:

u_y = 0.00007 + (0.00000e+00)*1 + (-2.96909e-04)*x + (-2.11999e-04)*y +
(3.03292e-04)*x^2 + (6.00628e-04)*x y + (1.43670e-03)*y^2 + (-
6.90029e-06)*x^3 + (-5.94268e-04)*x^2 y + (-9.66303e-06)*x y^2 + (-
9.53497e-04)*y^3

Polynomial Equation for u_x:

u_x = 0.00001 + (0.00000e+00)*1 + (-5.48818e-05)*x + (2.08362e-06)*y +
(7.28060e-04)*x^2 + (-2.79475e-06)*x y + (-2.10696e-06)*y^2 + (-
4.87028e-04)*x^3 + (-5.32934e-07)*x^2 y + (2.08893e-06)*x y^2 +
(7.18828e-07)*y^3

For data 5, the equations were:

Polynomial Equation for u_y:

u_y = 0.00008 + (0.00000e+00)*1 + (-3.71136e-04)*x + (-2.64999e-04)*y +
(3.79115e-04)*x^2 + (7.50785e-04)*x y + (1.79587e-03)*y^2 + (-
8.62537e-06)*x^3 + (-7.42835e-04)*x^2 y + (-1.20788e-05)*x y^2 + (-
1.19187e-03)*y^3

Polynomial Equation for u_x:

```

u_x = 0.00001 + (0.00000e+00)*1 + (-6.86023e-05)*x + (2.60453e-06)*y +
(9.10075e-04)*x^2 + (-3.49344e-06)*x y + (-2.63370e-06)*y^2 + (-
6.08785e-04)*x^3 + (-6.66168e-07)*x^2 y + (2.61117e-06)*x y^2 +
(8.98535e-07)*y^3

```

Now using the polynomial equation, we have found the Lame's constant.

```

import sympy as sp
import numpy as np
import pandas as pd
from scipy.optimize import minimize
from sklearn.model_selection import train_test_split
# Define symbols
df = pd.read_csv(r'C:\Users\ragha\vscode\programming_cp\pythonn\data\
1\displacement_data.csv')
x, y = sp.symbols('x y')
lambda_lame, mu_lame = sp.symbols('lambda mu')
# Define polynomial equations for u_x and u_y
u_y = (0.00002 - 7.42272e-05 * x - 5.29998e-05 * y +
7.58230e-05 * x**2 + 1.50157e-04 * x * y + 3.59174e-04 * y**2 -
1.72507e-06 * x**3 - 1.48567e-04 * x**2 * y - 2.41576e-06 * x *
y**2 - 2.38374e-04 * y**3)
u_x = (-1.37205e-05 * x + 5.20906e-07 * y + 1.82015e-04 * x**2 - 6.98688e-07 * x * y - 5.26740e
y**3)
# Compute first derivatives
dux_dx = sp.diff(u_x, x)
duy_dy = sp.diff(u_y, y)
# Compute second derivatives
d2ux_dx2 = sp.diff(u_x, x, 2)
d2ux_dy2 = sp.diff(u_x, y, 2)
d2uy_dx2 = sp.diff(u_y, x, 2)
d2uy_dy2 = sp.diff(u_y, y, 2)
d2ux_dxdy = sp.diff(u_x, x, y)
d2uy_dxdy = sp.diff(u_y, x, y)
# Laplacians of u_x and u_y
laplacian_ux = d2ux_dx2 + d2ux_dy2
laplacian_uy = d2uy_dy2 + d2uy_dx2
# Residuals based on Navier's equation
residual_ux = (lambda_lame + 2*mu_lame)d2ux_dx2 + mu_lame(d2ux_dy2) + (lambda_lame + mu_lame)*
residual_uy = (lambda_lame + 2*mu_lame)d2uy_dy2 + mu_lame(d2uy_dx2) + (lambda_lame + mu_lame)*

```

```

constraint_eq1 = lambda_lame * (dux_dx + duy_dy) + 2 *mu_lame*dux_dx - 21319457.47
constraint_eq2 = lambda_lame * (dux_dx + duy_dy) + 2 *mu_lame*duy_dy - 25377160.18
# Convert residuals and constraint to numerical functions
residual_ux_func = sp.lambdify((x, y, lambda_lame, mu_lame),
residual_ux)
residual_uy_func = sp.lambdify((x, y, lambda_lame, mu_lame),
residual_uy)
constraint_func1 = sp.lambdify((x, y, lambda_lame, mu_lame),
constraint_eq1)
constraint_func2 = sp.lambdify((x, y, lambda_lame, mu_lame),
constraint_eq2)
# Function to calculate the total cost
def calculate_cost(df, lambda_val, mu_val, penalty_scale=1e2):
    total_cost = 0
    x_vals, y_vals = df['x'].values, df['y'].values
    # Compute residuals for u_x and u_y at all points
    res_ux_vals = residual_ux_func(x_vals, y_vals, lambda_val, mu_val)
    res_uy_vals = residual_uy_func(x_vals, y_vals, lambda_val, mu_val) #
    Add squared residuals to cost
    total_cost += np.sum(res_ux_vals**2 + res_uy_vals**2)
    # Compute constraint violations only when x = 1 and y = 1
    respectively
    constraint1_violations = constraint_func1(x_vals[x_vals == 1],
y_vals[x_vals == 1], lambda_val, mu_val)
    constraint2_violations = constraint_func2(x_vals[y_vals == 1],
y_vals[y_vals == 1], lambda_val, mu_val)
    # Add penalties for constraint violations
    penalty_factor = 1e2# Adjusted penalty scale
    total_cost += penalty_factor *
    (np.sum(np.abs(constraint1_violations)**2) +
    np.sum(np.abs(constraint2_violations)**2))
    return total_cost
# Cost function wrapper for scipy optimizer
def cost_function(params, df):
    lambda_val, mu_val = params
    return calculate_cost(df, lambda_val, mu_val)
# Load data (replace with actual data file path)

```

```

df = pd.read_excel(r'C:\Sem3 IITD\hukka\proect1.xlsx')
# Split data: 60% for training, 40% for testing
df_train, df_test = train_test_split(df, test_size=0.4,
random_state=42)
# Initial guess for lambda and mu
initial_guess = [1e10, 1e10]
# Define bounds for lambda and mu (both are positive)
bounds = [(1e8, 1e12), (1e8, 1e12)] # Ensures positivity
# Optimize using Nelder-Mead method with bounds on the training data
result = minimize(cost_function, initial_guess, args=(df_train,),
method='Nelder-Mead', bounds=bounds)
# Extract optimized values
lambda_optimized, mu_optimized = result.x
# Print optimized values
print(f"Optimized Lamé's constants (Training Data):")
print(f"Lambda () = {lambda_optimized:.2e}")
print(f"Mu () = {mu_optimized:.2e}")
# Evaluate the cost on the test set
test_cost = calculate_cost(df_test, lambda_optimized, mu_optimized)

```

The output for data 1 is as follows:

Optimized Lamé's constants (Training Data):
Lambda () = 1.05e+11
Mu () = 5.61e+09

Similarly the code for data sets 2,3,4,5, was written. The output for them respectively would be:-

Optimized Lamé's constants (Training Data):
Lambda () = 1.24e+11
Mu () = 3.49e+09

Optimized Lamé's constants (Training Data):
Lambda () = 1.05e+11
Mu () = 5.61e+09

Optimized Lamé's constants (Training Data):

Lambda () = 1.05e+11
Mu () = 5.61e+09

Optimized Lamé's constants (Training Data):

Lambda () = 1.55e+11
Mu () = 4.36e+09

Now we will use Lame's constants(most repeating values) to find the value of stress and provide figures using

lambda=1.05e11
mu=5.61e9

For data 1, we will use the following code

```
import sympy as sp
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
# Load the data from CSV
df = pd.read_csv(r'C:\Users\ragha\vscode\programming_cp\pythonn\data\
1\displacement_data.csv')
# Extract x and y coordinates from the dataframe
x_points = df['x'].values
y_points = df['y'].values
# Define symbols
x, y = sp.symbols('x y')
lambda_lame, mu_lame = sp.symbols('lambda mu')
# Define polynomial equations for u_x and u_y
u_y = (0.00002 - 7.42272e-05 * x - 5.29998e-05 * y +
7.58230e-05 * x**2 + 1.50157e-04 * x * y + 3.59174e-04 * y**2 -
1.72507e-06 * x**3 - 1.48567e-04 * x**2 * y - 2.41576e-06 * x *
y**2 -
2.38374e-04 * y**3)
u_x = (-1.37205e-05 * x + 5.20906e-07 * y + 1.82015e-04 * x**2 -
6.98688e-07 * x * y - 5.26740e-07 * y**2 - 1.21757e-04 * x**3 -
1.33234e-07 * x**2 * y + 5.22233e-07 * x * y**2 + 1.79707e-07 *
```

```

y**3)

# Compute first derivatives
dux_dx = sp.diff(u_x, x)
duy_dy = sp.diff(u_y, y)
dux_dy = sp.diff(u_x, y)
duy_dx = sp.diff(u_y, x)
# Define stress components
sigma_xx = lambda_lame * (dux_dx + duy_dy) + 2 * mu_lame * dux_dx sigma_yy
= lambda_lame * (dux_dx + duy_dy) + 2 * mu_lame * duy_dy sigma_xy =
mu_lame * (dux_dy + duy_dx)
# Define values for lambda and mu
lambda_val = 1.05e11
mu_val = 5.61e9
# Lambdify the expressions for faster evaluation
sigma_xx_func = sp.lambdify((x, y, lambda_lame, mu_lame), sigma_xx,
modules="numpy")
sigma_yy_func = sp.lambdify((x, y, lambda_lame, mu_lame), sigma_yy,
modules="numpy")
sigma_xy_func = sp.lambdify((x, y, lambda_lame, mu_lame), sigma_xy,
modules="numpy")
# Evaluate the stress components for each point in the dataset
sigma_xx_values = sigma_xx_func(x_points, y_points, lambda_val,
mu_val)
sigma_yy_values = sigma_yy_func(x_points, y_points, lambda_val,
mu_val)
sigma_xy_values = sigma_xy_func(x_points, y_points, lambda_val,
mu_val)
# Create scatter plots for the stress components
fig, axes = plt.subplots(1, 3, figsize=(18, 6))
# Scatter plot for sigma_xx
scatter1 = axes[0].scatter(x_points, y_points, c=sigma_xx_values,
cmap='twilight', s=10)
axes[0].set_title(r'$\sigma_{xx}$ (Variation of $\sigma_{xx}$)')
axes[0].set_xlabel('x')
axes[0].set_ylabel('y')
cbar = fig.colorbar(scatter1, ax=axes[0])
scatter1.set_clim(0, 1.8 * 1e8)
# Scatter plot for sigma_yy

```

```

scatter2 = axes[1].scatter(x_points, y_points, c=sigma_yy_values,
cmap='twilight', s=10)
axes[1].set_title(r'$\sigma_{yy}$ (Variation of $\sigma_{yy}$)')
axes[1].set_xlabel('x')
axes[1].set_ylabel('y')
cbar = fig.colorbar(scatter2, ax=axes[1])
scatter2.set_clim(0, 1.8 * 1e8)
# Scatter plot for sigma_xy
scatter3 = axes[2].scatter(x_points, y_points, c=sigma_xy_values,
cmap='twilight', s=10)
axes[2].set_title(r'$\sigma_{xy}$ (Variation of $\sigma_{xy}$)')
axes[2].set_xlabel('x')
axes[2].set_ylabel('y')
cbar = fig.colorbar(scatter3, ax=axes[2])
scatter3.set_clim(0, 1.8 * 1e6)
# Display the plots
plt.tight_layout()
plt.show()

```

Similarly, we will write the code for data 2,3,4,5.

The graphs hence obtained are as follows:

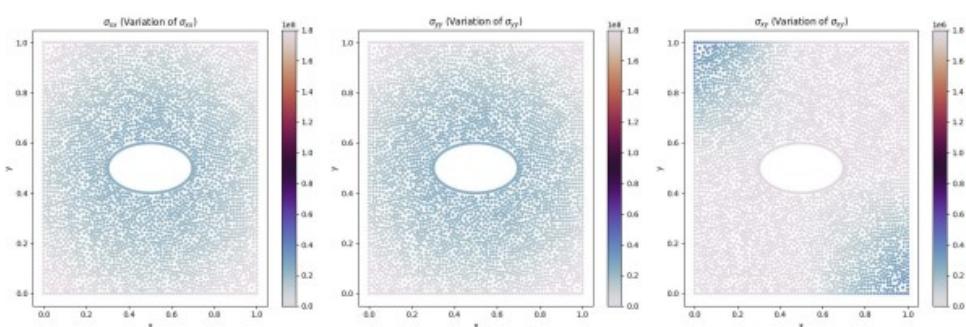


Figure 1: Data 1

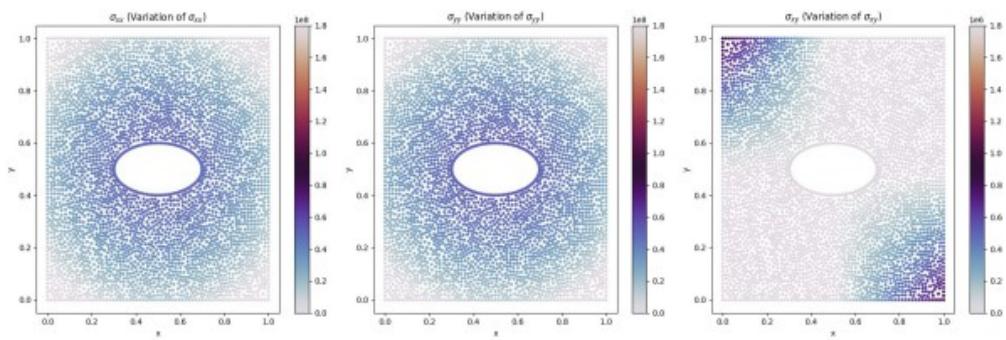


Figure 2: Data 2

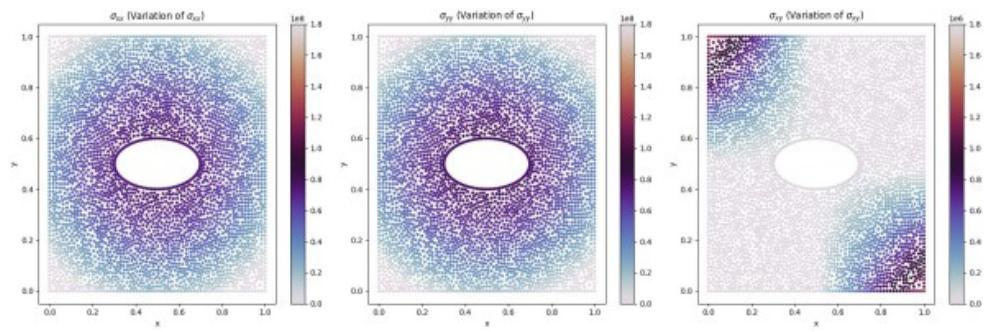


Figure 3: Data 3

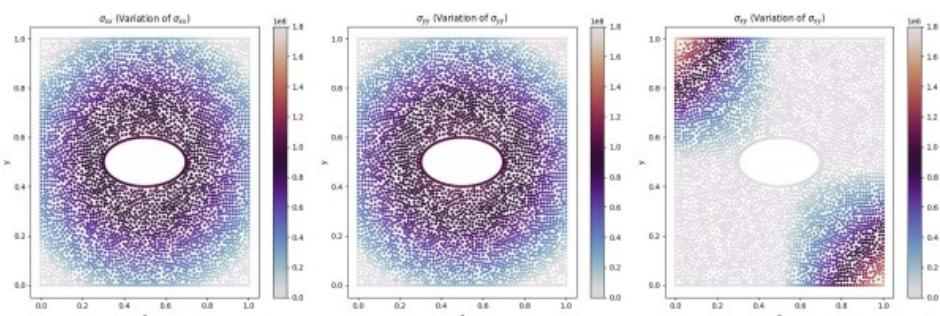


Figure 4: Data 4

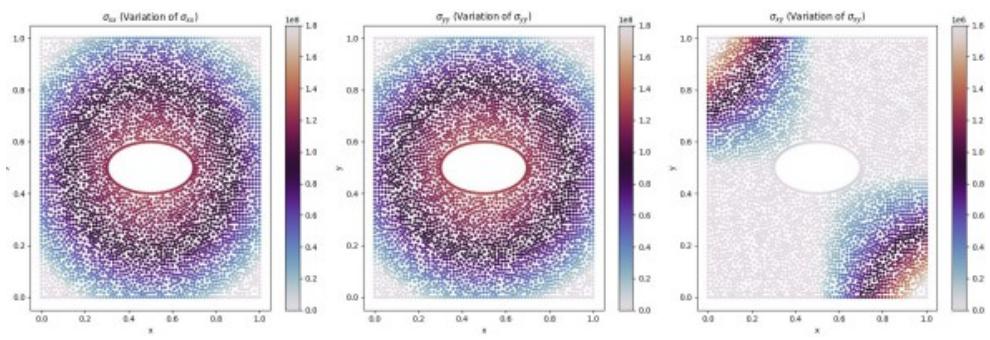


Figure 5: Data 5