BME Faculty of Mechanical Engineering	EXPERIMENTAL METHODS	Name: Kelle Gergő
Dep. of Applied Mechanics	1st HOMEWORK	Neptun code: GBBNUL
2024/25 I.	Deadline: see Moodle	Late submission ☐ Correction ☐
Declaration: With my signature I declare, that I did my homework myself, everything written in there is my knowledge.		Signature:

We only correct homeworks that correspond with formal requirements. Correction or late submission is only possible until the late submission deadline.

Problem

An open-cell polyurethane foam material has been tested using uniaxial relaxation and cyclic compression tests with an Instron 3345 Single Column Universal Testing System, while the load was measured by an Instron model 2519-107 5kN load cell. The displacement was measured directly from the cross-head position. The measurement layout is depicted in Figure 1, while the exact dimensions of the specimens are listed in the tables below. The initial tool separation was H_0. The deformation of the specimen is assumed to be homogeneous.

The link for the measurement data can be found at the bottom of the 2nd page below the table of test parameters!

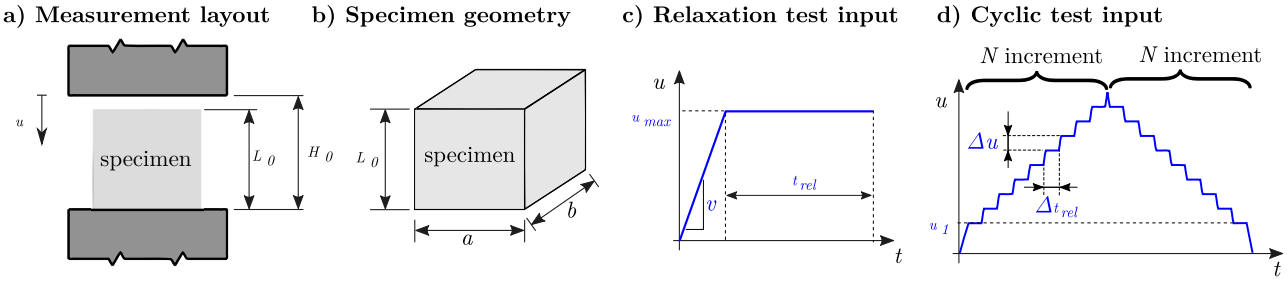

Figure 1: Measurement layout and displacement inputs

Task A

The relaxation test comprises of two phases (see Fig. 1/c): an unloading phase up to the maximal displacement u_{max} with cross-head speed v, and a relaxation phase for time t_{rel}. The measurement was also recorded with digital camera. The raw data of the measurement contains the measured time [s], extension [mm] and load [N] values and the video of the measurement in .mp4 format.

1. Determine the exact height of the relaxation test specimen (L_0^{rel}); obtain and plot the compensated force-displacement ($F-u$) and force-time ($F-t$) curves using slack correction method!
2. Determine and plot the measured engineering stress - engineering strain ($P-\varepsilon$), engineering stress – stretch ($P-\lambda$) and engineering stress-time ($P-t$) curves!
3. Determine and plot the cross-directional stretch characteristics ($\lambda-\lambda_T$) using a self-developed image processing method on the video recording with a suitable programming language (MatLab, Wolfram, Python, etc...)!
4. Determine the maximal $\lambda_{T,max}$ value!
5. Plot the true stress – true strain curve ($\sigma-\varepsilon^{true}$)!

Task B

After that, a cyclic compression test was performed on another specimen from the same material. The cyclic tests consist of N increments both in the uploading and unloading phases (see Fig. 1/d). The displacement of the first step in the uploading (and the last step of the unloading) was u_1, while in all other increments the displacement was Δu. The cross-head speed was v_{cyc}. After each increment (except the N^{th}), a relaxation phase for time Δt_{rel} was inserted. The raw data of the measurement contains the measured time [s], extension [mm] and load [N] values.

1. Determine the exact height of the cyclic test specimen (L_0^{cyc}); obtain and plot the compensated force-displacement ($F-u$) and force-time ($F-t$) curves using slack correction method!
2. Plot the measured engineering stress - engineering strain ($P-\varepsilon$), engineering stress – stretch ($P-\lambda$) and engineering stress-time ($P-t$) curves!
3. List and plot the data points corresponding to the long-term, pure elastic engineering-stress – stretch ($P-\lambda$) curve, by interpolating between endpoints of the relaxation phases in the uploading and unloading curves.
4. Illustrate on the same graph the engineering stress-stretch curves ($P-\lambda$) obtained from the relaxation test and pure elastic approximation. Determine the average of $\eta = P^{rel}(\lambda)/P^{long}(\lambda)$ ratio of the corresponding long-term and relaxation engineering stress values!

Formal requirements

Create a PDF report on the measurement results and the post-processing using document processor software (e.g. Word, LaTeX). The first page must be the signed cover sheet. Please also attach the commented program code of the image processing. The header and the numerical results have to be filled. Short comments and explanations must be given to each step, to such an extent that the work may be reproduced even by colleagues who are not specialized in the given topic. All questions must be answered! Do not upload pure program codes, it won't be accepted.

Data

Parameters of the relaxation test:

Specimen number	H_0 [mm]	u_{max} [mm]	a [mm]	b [mm]	v [mm/min]	t_{rel} [s]
6	75	56	76.24	74.91	120	30

Download relaxation test data: https://www.mm.bme.hu/edu/msc-en/expmeth/0hw1/relax6.zip

Parameters of the cyclic test:

Specimen number	H_0 [mm]	u_1 [mm]	Δu [mm]	Δt_{rel} [s]	N [-]	v_{cyc} [mm/min]	a [mm]	b [mm]
18	75	10.5	5.5	30	9	40	77.71	75.11

Download cyclic test raw data: https://www.mm.bme.hu/edu/msc-en/expmeth/0hw1/cyclic18.zip

Results

L_0^{rel} [mm]	$\lambda_{T,max}$ [-]	L_0^{cyc} [mm]	η_{avg} [-]
69.17	1.02376	68.5	1.425

Budapest University of Technology and Economics
Faculty of Mechanical Engineering

Experimental methods in solid mechanics
I. Homework

Kelle Gergő

November 2, 2024

Contents

1 Task A - Relaxation **5**
1.1 Task A1 5
1.2 Task A2 6
1.3 Task A3 7
1.4 Task A4 8
1.5 Task A5 8

2 Task B - Cyclic Compression **9**
2.1 Task B1 9
2.2 Task B2 10
2.3 Task B3 10
2.4 Task B4 11

A Code Appendix **13**

1 Task A - Relaxation

1.1 Task A1

Task description: Determine the exact height of the relaxation test specimen (L_0^{rel}); obtain and plot the compensated force - displacement (F - u) and force-time (F -t) curves using slack correction method!

The initial height of the relaxation specimen, L_0^{rel}, can be precisely calculated from the measured data. When the load begins to increase, it indicates initial contact between the foam and the compression head. Thus, the specimen height is determined as the initial tool separation minus the extension at this point:

$$L_0^{\text{rel}} = H_0 - u_0 \approx 69.17 \text{ mm} \tag{1}$$

To conduct slack correction, first the inflection point has to be located on the load-displacement curve and a tangent line at that point has to be drawn. Next, I've found the intersection of this tangent with the x-axis and eliminated the measurement data before this point. After that I've substituted the region with a linear segment. This method redefined the zero-displacement point as the following figures shows.

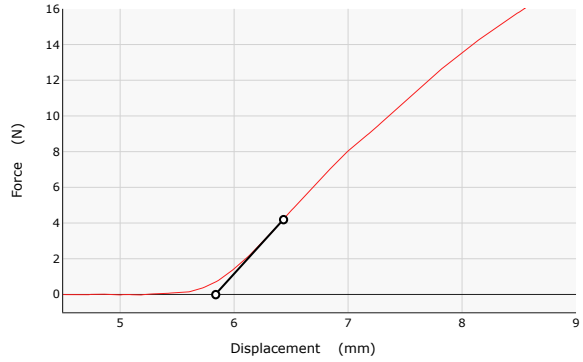

Figure 1: *Slack correction - tangential linear part*

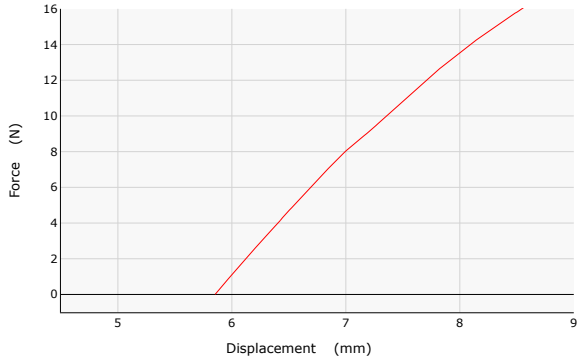

Figure 2: *Slack corrected part*

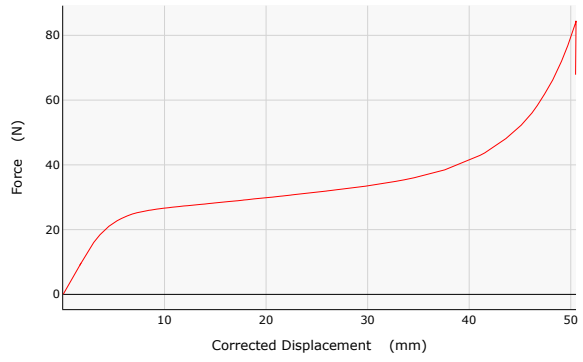

Figure 3: *Force - compensated displacement plot*

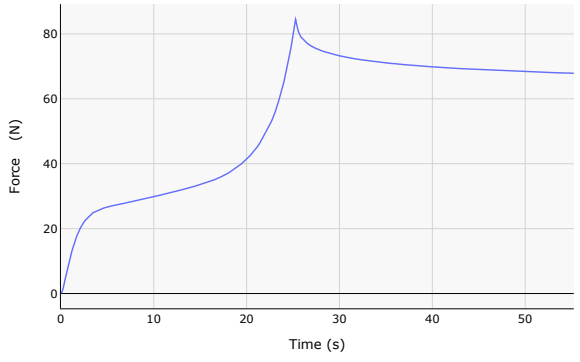

Figure 4: *Force - time plot*

1.2 Task A2

Task description: Determine and plot the measured engineering stress- engineering strain (P - ε), engineering stress - stretch (P - λ) and engineering stress-time (P - t) curves!

First let's clarify the used formulas. The cross-sectional area can be deterined as

$$A = a\ b = 5711.1384\ \left[\text{mm}^2\right]. \tag{2}$$

The engineering stress can be calculated as

$$P = \frac{F}{A}. \tag{3}$$

The stretch can be calculated as

$$\lambda = 1 - \frac{u}{L_0^{rel}}. \tag{4}$$

The engineering strain can be calculated as:

$$\varepsilon = \lambda - 1. \tag{5}$$

Using theese formulas I've created the neccessary plots.

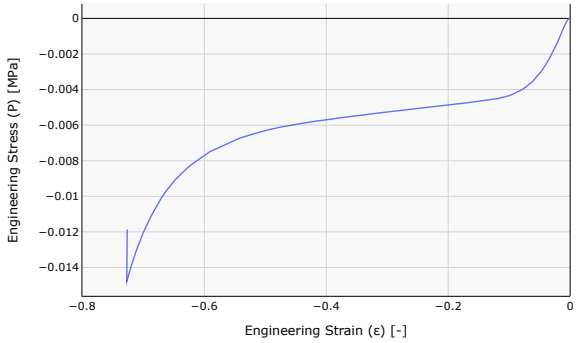

Figure 5: Engineering Stress - Engineering Strain Curve

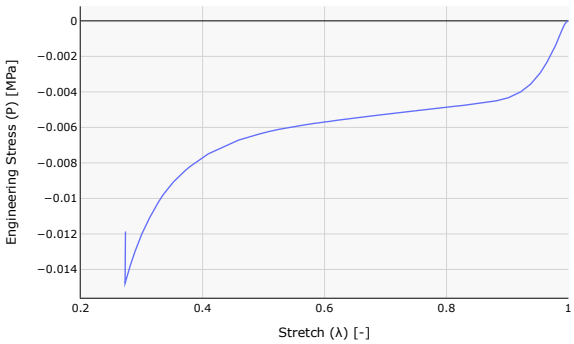

Figure 6: *Engineering Stress - Stretch Curve*

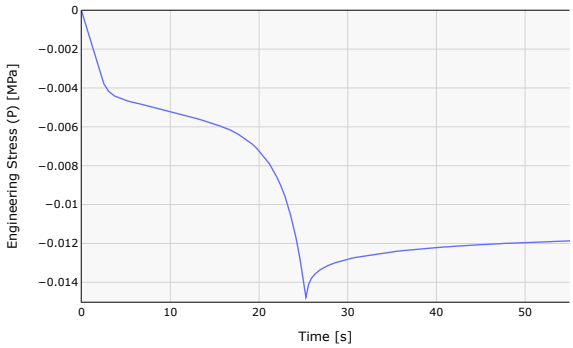

Figure 7: *Engineering Stress - Time Curve*

1.3 Task A3

Task description: Determine and plot the cross-directional stretch characteristics (λ - λ_T) using a self-developed image processing method on the video recording with a suitable programming language (MatLab, Wolfram, Python, etc...)!

The implementation of the image processing method was conducted using Python. The process began with video cropping to focus on the relevant area of interest. Next, each frame was converted to grayscale to simplify the data and enhance contrast. A thresholding technique was then applied to produce a binary image, transforming the grayscale frames into black and white representations. To quantify the cross-directional stretch, the sum of white pixels was calculated along the video duration at half-second intervals.

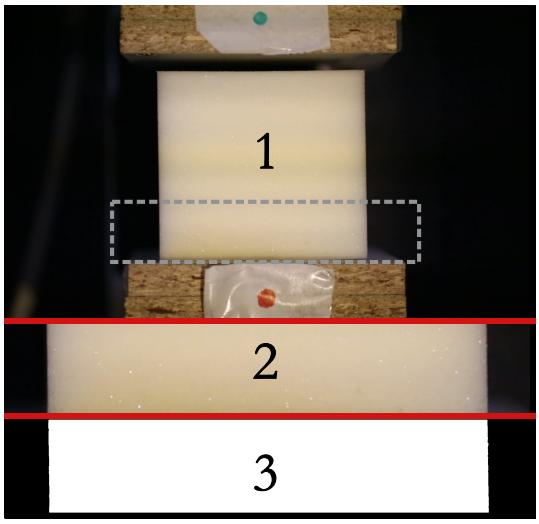

Figure 8: Video processing

After processing the video cross-directional stretch characteristics (λ - λ_T) plot can be done after evening out the two data set row numbers.

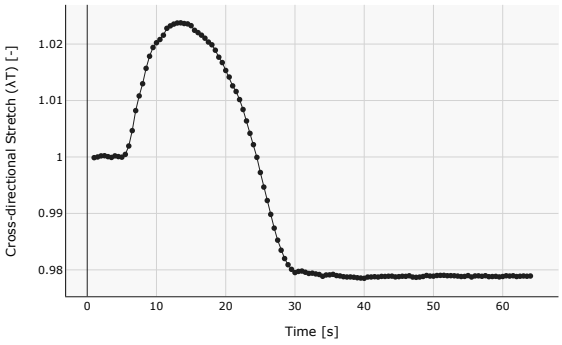

Figure 9: *Cross-dir. Stretch Characteristics*

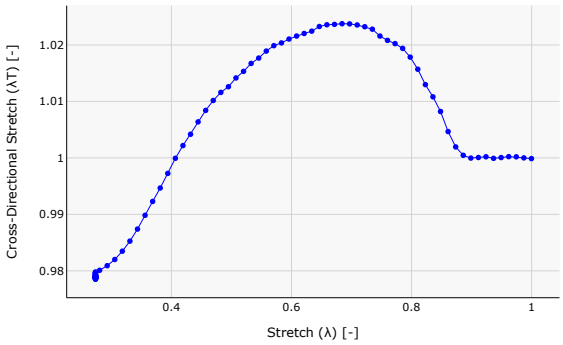

Figure 10: *Cross-dir. Stretch - Stretch plot*

The plots displayed anticipated outcomes, indicating that while foams are easily compressed, they exhibit minimal transverse stretch in the material.

1.4 Task A4

Task description: Determine the maximal λ_T,max value!

$$\max(\lambda_T) = 1.02376 \tag{6}$$

1.5 Task A5

Task description: Plot the true stress - true strain curve (σ - ε^{true})!

True stress we need the actual corss-section area that can be calculated as

$$\sigma_{\text{true}} = \frac{F}{\lambda_T^2 \; a \; b} \tag{7}$$

The true strain can be calculated with the following equation

$$\varepsilon_{\text{true}} = \ln(\lambda) \tag{8}$$

After the calculation the number of values in each dataset should be matched in order to plot properly, see the results on the Figure 11 below.

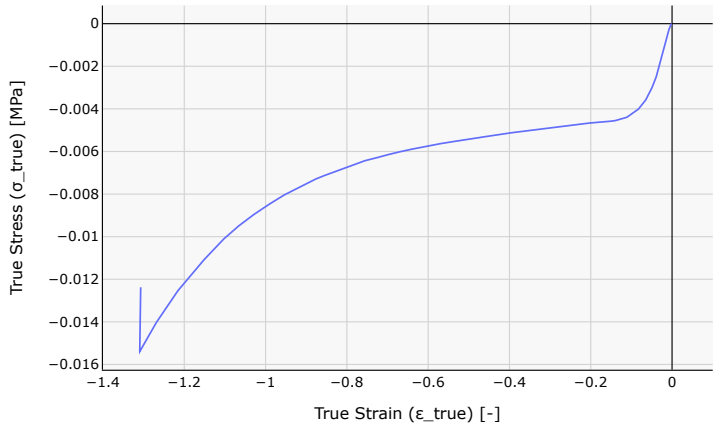

Figure 11: *True stress - True strain plot*

A true strain of -1.3 implies that the material's current length is only about $e^{-1.3} \approx 0.27$ of its original length. This suggests about a 73% reduction in length, indicating substantial compression.

2 Task B - Cyclic Compression

2.1 Task B1

Task description: Determine the exact height of the cyclic test specimen (L_0^{cyc}); obtain and plot the compensated force - displacement (F - u) and force - time (F - t) curves using slack correction method!

The initial height of the cyclic test specimen, L_0^{cyc}, can be calculated in the same manner as for the relaxation test. Furthermore, the required plots have been generated analogously to those for the relaxation test.

$$L_0^{\mathrm{cyc}} = H_0 - u_{0,cyc} = 68.5 \text{ mm} \tag{9}$$

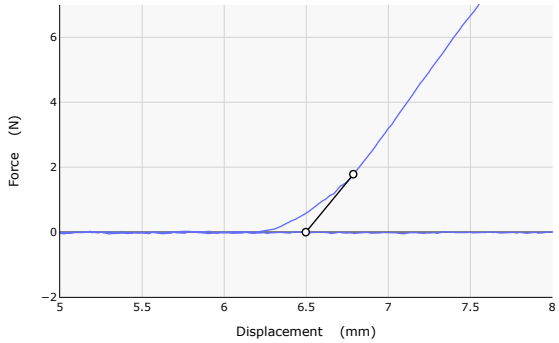

Figure 12: *Slack corr. - tangential linear part*

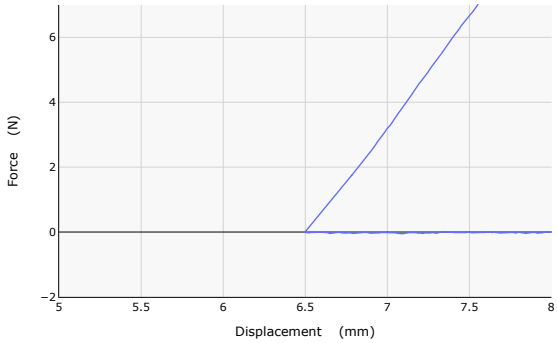

Figure 13: *Slack corrected part*

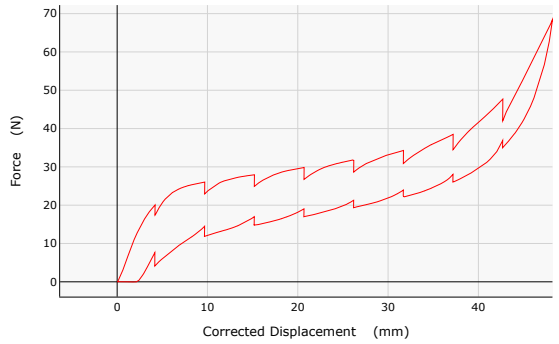

Figure 14: *Force - compensated disp. plot*

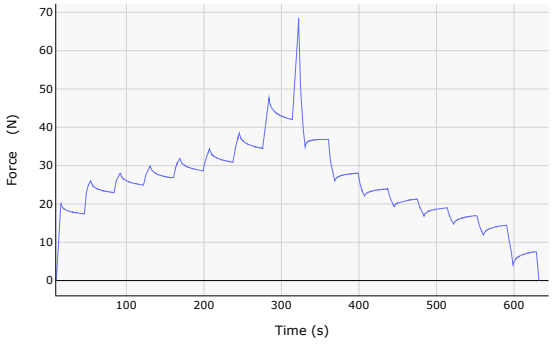

Figure 15: *Force - time plot*

2.2 Task B2

Task description: Plot the measured engineering stress- engineering strain (P - ε), engineering stress - stretch (P - λ) and engineering stress-time (P - t) curves!

To plot the required diagrams; engineering stress, engineering strain, and stretch must be calculated. This calculation is performed using the same equations established for the relaxation test; however, in this case, I utilize the results from the cyclic test.

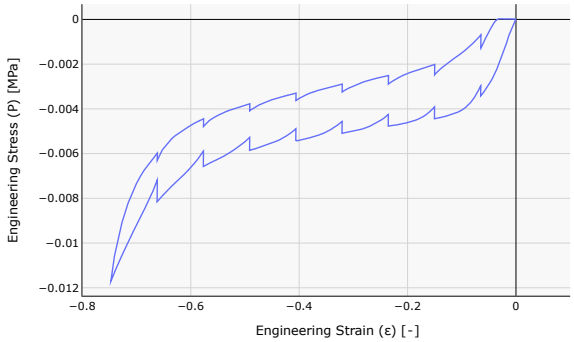

Figure 16: Engineering Stress - Engineering Strain Curve

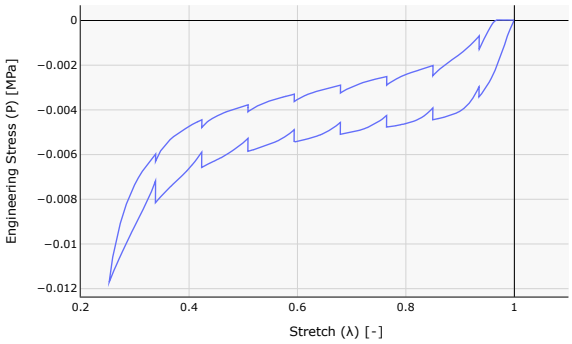

Figure 17: *Engineering Stress - Stretch Curve*

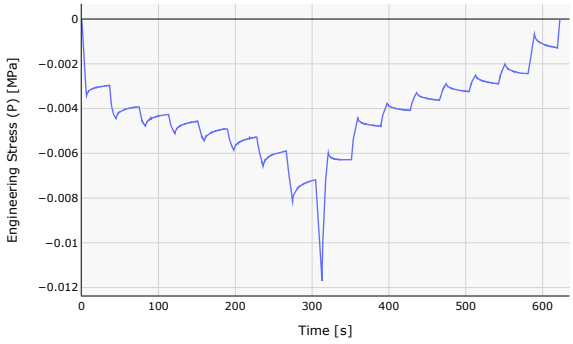

Figure 18: *Engineering Stress - Time Curve*

2.3 Task B3

Task description: List and plot the data points corresponding to the long-term, pure elastic engineering stress - stretch (P - λ) curve, by interpolating between endpoints of the relaxation phases in the uploading and unloading curves.

In order to generate the long-term, pure elastic engineering stress-stretch (P - λ) curve, I began by identifying the local minima and maxima of the uploading and unloading curves. After that, I've calculated the midpoints by averaging the corresponding values of these local extremas. Finally, I fitted a line through these midpoints to create the desired interpolation, resulting in the pure elastic approximation curve shown in Figure 19.

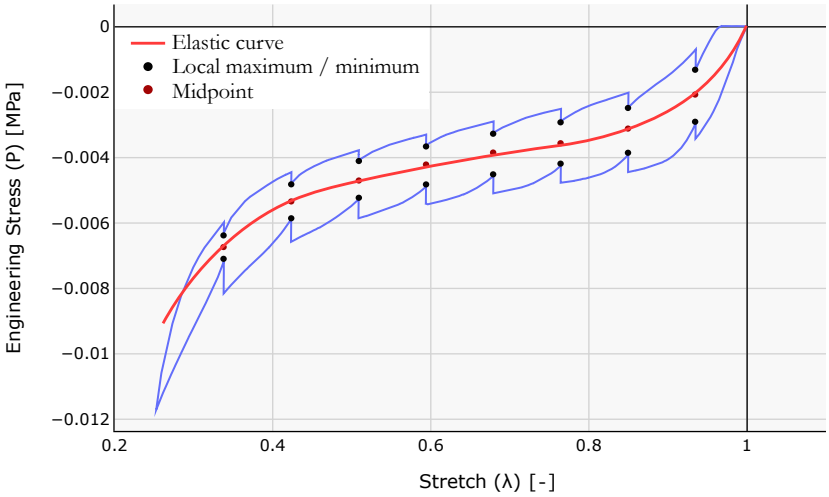

Figure 19: Engineering Stress - Stretch with fitted pure elastic approximation curve

2.4 Task B4

Task description: Illustrate on the same graph the engineering stress-stretch curves (P - λ) obtained from the relaxation test and pure elastic approximation. Determine the average of $\eta = P^{rel}(\lambda)/P^{long}(\lambda)$ ratio of the corresponding long-term and relaxation engineering stress values!!

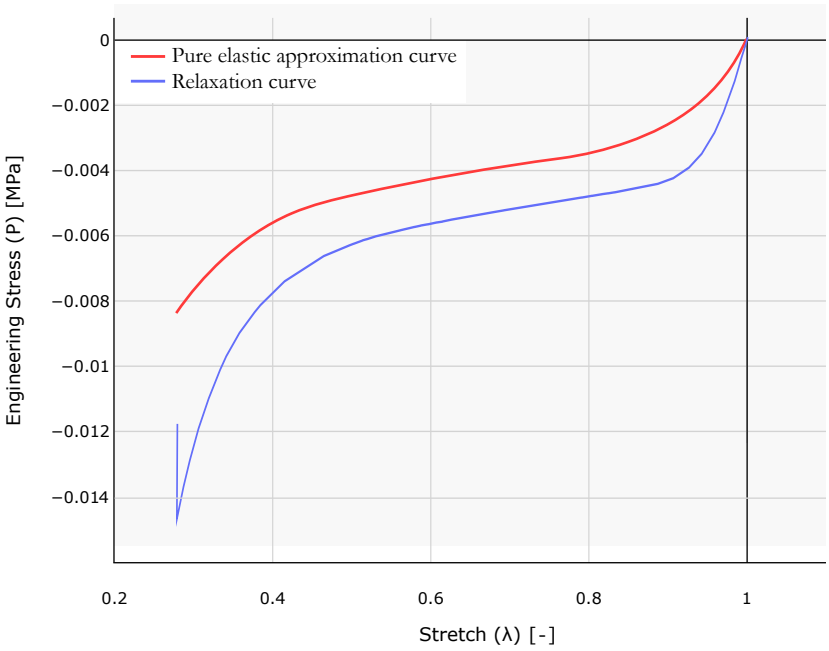

Figure 20: Pure Elastic approximation and relaxation test result

As for the ratio of the corresponding long-term and relaxation engineering stress values, I chose a straightforward approach using image processing. I converted Figure 20 into two separate binary

images, summed up the white pixels, and calculated the ratio as:

$$\eta = \frac{P^{\text{rel}}(\lambda)}{P^{\text{long}}(\lambda)} = 1.425 \tag{10}$$

Figure 21: *Binary picture of the pure elastic approximation*

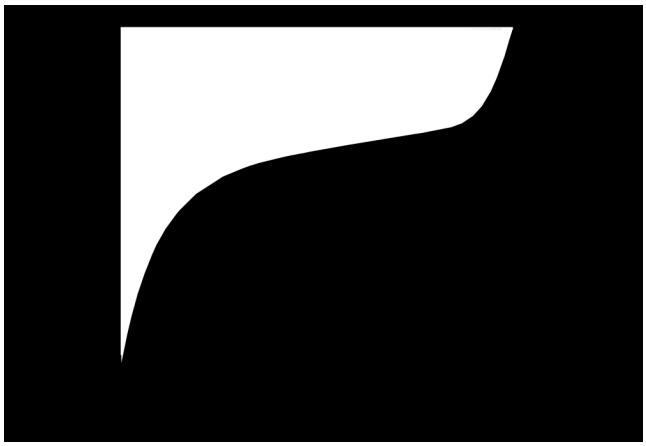

Figure 22: *Binary picture of the relaxation test result*

A Code Appendix

```
#!/usr/bin/env python
# coding: utf-8

# In[21]:


clear all


# In[22]:


def style_plot(fig, x_title, y_title, title):
    # Add gradient background using shapes
    fig.update_layout(
        title=title,
        plot_bgcolor='rgba(255, 255, 255, 0)',  # Transparent plot background
        shapes=[
            dict(
                type='rect',
                xref='paper', yref='paper',
                x0=0, x1=1, y0=0, y1=1,
                fillcolor='rgba(211, 211, 211, 0.15)',  # Grey with 15% opacity
                line=dict(width=0)
            ),
        ],
        xaxis=dict(
            title=x_title,
            gridcolor='lightgrey',  # Lighter grid color for more hollow effect
            gridwidth=1,
            zerolinecolor='black',  # Zero line color
            zerolinewidth=2,
            showgrid=True,
            linecolor='black'  # X-axis line color
        ),
        yaxis=dict(
            title=y_title,
            gridcolor='lightgrey',  # Lighter grid color for more hollow effect
            gridwidth=1,
            zerolinecolor='black',  # Zero line color
            zerolinewidth=2,
            showgrid=True,
            linecolor='black'  # Y-axis line color
        ),
        font=dict(color='black'),
    )

    # Set colors for traces
    if len(fig.data) >= 1:
        fig.update_traces(line=dict(color='red'), selector=dict(name='
   Compensated F-u'))  # Default color for the first trace
    if len(fig.data) >= 2:
```

```
        fig.data[1].line.color = 'green'  # Set the second trace color to green


# In[23]:


## TASK A - Preparation


# In[24]:


# Import necessary libraries
import pandas as pd
import plotly.graph_objs as go

# Define basic parameters
H_0 = 75  # mm, initial tool separation
u_max = 56  # mm, maximal displacement during unloading
a = 76.24  # mm, specimen width
b = 74.91  # mm, specimen length
v = 120  # mm/min, cross-head speed during unloading
t_rel = 30  # s, relaxation time

# Load the CSV file into a DataFrame
file_name = 'Relax_RawData_6.csv'
data = pd.read_csv(file_name)

# Convert columns to appropriate data types
data['Time'] = pd.to_numeric(data.iloc[:, 0], errors='coerce')  # Convert Time to float
data['Extension'] = pd.to_numeric(data.iloc[:, 1], errors='coerce')  # Convert Extension to float
data['Load'] = pd.to_numeric(data.iloc[:, 2], errors='coerce')  # Convert Load to float

# Extract the columns: Time, Extension, Load
time = data['Time']
extension = data['Extension']
load = data['Load']

# Plot Extension vs Time
ext_vs_time = go.Scatter(x=time, y=extension, mode='lines', name='Extension vs Time')
layout_ext = go.Layout(
    title='Extension vs Time',
    xaxis=dict(title='Time (s)', tickformat='.1f'),  # 1 decimal for Time
    yaxis=dict(title='Extension (mm)', tickformat='.2f'),  # 2 decimals for Extension
)
fig_ext = go.Figure(data=[ext_vs_time], layout=layout_ext)

# Plot Load vs Time
load_vs_time = go.Scatter(x=time, y=load, mode='lines', name='Load vs Time')
```

```
layout_load = go.Layout(
    title='Load vs Time',
    xaxis=dict(title='Time (s)', tickformat='.1f'),  # 1 decimal for Time
    yaxis=dict(title='Load (N)', tickformat='.2f'),  # 2 decimals for Load
)
fig_load = go.Figure(data=[load_vs_time], layout=layout_load)

# Show both plots
fig_ext.show()
fig_load.show()


# In[25]:


# Import necessary libraries
import pandas as pd
import plotly.graph_objs as go

# Define basic parameters
H_0 = 75  # mm, initial tool separation
u_max = 56  # mm, maximal displacement during unloading
a = 76.24  # mm, specimen width
b = 74.91  # mm, specimen length
v = 120  # mm/min, cross-head speed during unloading
t_rel = 30  # s, relaxation time

# Load the CSV file into a DataFrame
file_name = 'Relax_RawData_6.csv'
data = pd.read_csv(file_name)

# Convert columns to appropriate data types
data['Time'] = pd.to_numeric(data.iloc[:, 0], errors='coerce')  # Convert Time to float
data['Extension'] = pd.to_numeric(data.iloc[:, 1], errors='coerce')  # Convert Extension to float
data['Load'] = pd.to_numeric(data.iloc[:, 2], errors='coerce')  # Convert Load to float

# Extract the columns: Time, Extension, Load
time = data['Time']
extension = data['Extension']
load = data['Load']

# Create the plot for Extension vs Time
ext_vs_time = go.Scatter(x=time, y=extension, mode='lines', name='Extension vs Time')
layout_ext = go.Layout(
    title='Extension vs Time',
    xaxis=dict(title='Time (s)', tickformat='.1f'),  # 1 decimal for Time
    yaxis=dict(title='Extension (mm)', tickformat='.2f'),  # 2 decimals for Extension
)
fig_ext = go.Figure(data=[ext_vs_time], layout=layout_ext)
```

```
150
151 # Create the plot for Load vs Time
152 load_vs_time = go.Scatter(x=time, y=load, mode='lines', name='Load vs Time')
153 layout_load = go.Layout(
154     title='Load vs Time',
155     xaxis=dict(title='Time (s)', tickformat='.1f'),  # 1 decimal for Time
156     yaxis=dict(title='Load (N)', tickformat='.2f'),  # 2 decimals for Load
157 )
158 fig_load = go.Figure(data=[load_vs_time], layout=layout_load)
159
160 # Apply the styling to both figures
161 style_plot(fig_ext, 'Time (s)', 'Extension (mm)', 'Extension vs Time')
162 style_plot(fig_load, 'Time (s)', 'Load (N)', 'Load vs Time')
163
164 # Show both plots
165 fig_ext.show()
166 fig_load.show()
167
168
169 # TASK A - TASK 1
170
171 # In[157]:
172
173
174 # Identifying the slack region (first non-zero significant load)
175 slack_threshold = 0.1  # N, threshold for detecting when the load becomes
    significant
176 slack_index = (load > slack_threshold).idxmax()  # Get index of first
    significant load
177
178 # Slice the data from slack_index onwards
179 time_corrected = time[slack_index:].reset_index(drop=True)  # Time starting
    from slack point
180 u_corrected = (extension - extension[slack_index])[slack_index:].reset_index(
    drop=True)  # Corrected extension
181 load_corrected = load[slack_index:].reset_index(drop=True)  # Corresponding
    load values
182
183 # Plot compensated Force-Displacement (F-u) curve
184 compensated_force_displacement = go.Scatter(x=u_corrected, y=load_corrected,
    mode='lines', name='Compensated F-u')
185 layout_fu = go.Layout(
186     title='Compensated Force-Displacement (F-u) Curve',
187     xaxis=dict(title='Corrected Displacement u (mm)'),
188     yaxis=dict(title='Force F (N)'),
189 )
190 fig_fu = go.Figure(data=[compensated_force_displacement], layout=layout_fu)
191
192 # Plot Force-Time (F-t) curve using corrected time
193 force_time = go.Scatter(x=time_corrected, y=load_corrected, mode='lines', name=
    'F-t')
194 layout_ft = go.Layout(
195     title='Force-Time (F-t) Curve',
196     xaxis=dict(title='Time (s)'),
```

```
    yaxis=dict(title='Force F (N)'),
)
fig_ft = go.Figure(data=[force_time], layout=layout_ft)

# Show both plots
fig_fu.show()
fig_ft.show()


# In[158]:


import pandas as pd
import plotly.graph_objs as go

# Load the data
file_name = 'Relax_RawData_6.csv'
data = pd.read_csv(file_name)

# Ensure numeric conversion for relevant columns
data['Time'] = pd.to_numeric(data['Time'], errors='coerce')
data['Extension'] = pd.to_numeric(data['Extension'], errors='coerce')
data['Load'] = pd.to_numeric(data['Load'], errors='coerce')

# Drop rows with NaN values in essential columns
data = data.dropna(subset=['Time', 'Extension', 'Load'])

# Extract the columns
time = data['Time']
extension = data['Extension']
load = data['Load']

# Identifying the slack region (first non-zero significant load)
slack_threshold = 0.1  # N, threshold for detecting significant load
slack_index = (load > slack_threshold).idxmax()  # Get index of first
    significant load

# Slice the data from slack_index onwards
time_corrected = time[slack_index:].reset_index(drop=True)  # Time starting
    from slack point
u_corrected = (extension - extension[slack_index])[slack_index:].reset_index(
    drop=True)  # Corrected extension
load_corrected = load[slack_index:].reset_index(drop=True)  # Corresponding
    load values

# Plot compensated Force-Displacement (F-u) curve
compensated_force_displacement = go.Scatter(x=u_corrected, y=load_corrected,
    mode='lines', name='Compensated F-u')
fig_fu = go.Figure(data=[compensated_force_displacement])

# Apply styling to the F-u figure
style_plot(fig_fu, 'Corrected Displacement u (mm)', 'Force F (N)', 'Compensated
    Force-Displacement (F-u) Curve')

```

```
# Plot Force-Time (F-t) curve using corrected time
force_time = go.Scatter(x=time_corrected, y=load_corrected, mode='lines', name='F-t')
fig_ft = go.Figure(data=[force_time])

# Apply styling to the F-t figure
style_plot(fig_ft, 'Time (s)', 'Force F (N)', 'Force-Time (F-t) Curve')

# Show both plots
fig_fu.show()
fig_ft.show()


# In[166]:


import pandas as pd
import plotly.graph_objs as go

# Load the data
file_name = 'Relax_RawData_6.csv'
data = pd.read_csv(file_name)

# Ensure numeric conversion for relevant columns
data['Time'] = pd.to_numeric(data['Time'], errors='coerce')
data['Extension'] = pd.to_numeric(data['Extension'], errors='coerce')
data['Load'] = pd.to_numeric(data['Load'], errors='coerce')

# Drop rows with NaN values in essential columns
data = data.dropna(subset=['Time', 'Extension', 'Load'])

# Extract the columns
time = data['Time']
extension = data['Extension']
load = data['Load']

# Identifying the slack region (first non-zero significant load)
slack_threshold = 0.1  # N, threshold for detecting significant load
slack_index = (load > slack_threshold).idxmax()  # Get index of first significant load

# Slice the data from slack_index onwards
time_corrected = time[slack_index:].reset_index(drop=True)  # Time starting from slack point
u_corrected = (extension - extension[slack_index])[slack_index:].reset_index(drop=True)  # Corrected extension
load_corrected = load[slack_index:].reset_index(drop=True)  # Corresponding load values

# Shift time_corrected to start from 0
time_corrected_shifted = time_corrected - time_corrected.min()

# Plot compensated Force-Displacement (F-u) curve
compensated_force_displacement = go.Scatter(x=u_corrected, y=load_corrected,
```

```
    mode='lines', name='Compensated F-u')
fig_fu = go.Figure(data=[compensated_force_displacement])

# Apply styling to the F-u figure
style_plot(fig_fu, 'Corrected Displacement u (mm)', 'Force F (N)', 'Compensated Force-Displacement (F-u) Curve')

# Save the F-u figure as PDF
fig_fu.write_image("fig-1-v1.pdf")

# Plot Force-Time (F-t) curve using corrected time
force_time = go.Scatter(x=time_corrected_shifted, y=load_corrected, mode='lines', name='F-t')
fig_ft = go.Figure(data=[force_time])

# Apply styling to the F-t figure
style_plot(fig_ft, 'Time (s)', 'Force F (N)', 'Force-Time (F-t) Curve')

# Save the F-t figure as PDF
fig_ft.write_image("fig-2-v1.pdf")

# Show both plots
fig_fu.show()
fig_ft.show()


# In[180]:


L_0_rel = 69.17
L_0_rel


# In[162]:


# Identifying the slack region (first non-zero significant load)
slack_threshold = 0.1  # N, threshold for detecting when the load becomes significant
slack_index = (load > slack_threshold).idxmax()  # Get index of first significant load
# Slice the data from slack_index onwards
time_corrected = time[slack_index:].reset_index(drop=True)  # Time starting from slack point
u_corrected = (extension - extension[slack_index])[slack_index:].reset_index(drop=True)  # Corrected extension
load_corrected = load[slack_index:].reset_index(drop=True)  # Corresponding load values

# Plot compensated Force-Displacement (F-u) curve
compensated_force_displacement = go.Scatter(x=u_corrected, y=load_corrected, mode='lines', name='Compensated F-u')
layout_fu = go.Layout(
    title='Compensated Force-Displacement (F-u) Curve',
```

```
    xaxis=dict(title='Corrected Displacement u (mm)'),
    yaxis=dict(title='Force F (N)'),
)
fig_fu = go.Figure(data=[compensated_force_displacement], layout=layout_fu)

# Plot Force-Time (F-t) curve using corrected time
force_time = go.Scatter(x=time_corrected, y=load_corrected, mode='lines', name='F-t')
layout_ft = go.Layout(
    title='Force-Time (F-t) Curve',
    xaxis=dict(title='Time (s)'),
    yaxis=dict(title='Force F (N)'),
)
fig_ft = go.Figure(data=[force_time], layout=layout_ft)

# Apply the styling to the F-u and F-t figures
style_plot(fig_fu, 'Corrected Displacement u (mm)', 'Force F (N)', 'Compensated Force-Displacement (F-u) Curve')
style_plot(fig_ft, 'Time (s)', 'Force F (N)', 'Force-Time (F-t) Curve')

# Show both plots
fig_fu.show()
fig_ft.show()


# TASK A - TASK 2

# In[170]:


extension = extension[slack_index:] - extension[slack_index]
load = load[slack_index:] - load[slack_index]


# In[152]:


# Define cross-sectional area (A) in mm^2
A = a * b  # mm^2

# Calculate Engineering Stress (P)
stress = -load / A  # Engineering stress in N/mm^2 or MPa

# Calculate Stretch (lambda) using the updated formula
stretch = 1 - (extension) / L_0_rel  # Stretch ratio

# Calculate Engineering Strain (eps) as stretch - 1
strain = stretch - 1  # Engineering strain

# Plot Engineering Stress - Engineering Strain (P - eps) curve
fig_ps = go.Figure(data=[go.Scatter(x=strain, y=stress, mode='lines', name='Engineering Stress - Strain')])
style_plot(fig_ps, 'Engineering Strain (eps) [-]', 'Engineering Stress (P) [MPa]', 'Engineering Stress - Engineering Strain (P - eps) Curve')
```

```
389
390 # Plot Engineering Stress - Stretch (P - lambda) curve
391 fig_pl = go.Figure(data=[go.Scatter(x=stretch, y=stress, mode='lines', name='
    Engineering Stress - Stretch')])
392 style_plot(fig_pl, 'Stretch (lambda) [-]', 'Engineering Stress (P) [MPa]', '
    Engineering Stress - Stretch (P - lambda) Curve')
393
394 # Plot Engineering Stress - Time (P - t) curve
395 fig_pt = go.Figure(data=[go.Scatter(x=time_corrected, y=stress[slack_index:].
    reset_index(drop=True), mode='lines', name='Engineering Stress - Time')])
396 style_plot(fig_pt, 'Time [s]', 'Engineering Stress (P) [MPa]', 'Engineering
    Stress - Time (P - t) Curve')
397
398 # Show all plots
399 fig_ps.update_xaxes(range=[-0.8, 0])
400 fig_pl.update_xaxes(range=[0.2, 1])
401 fig_pt.update_xaxes(range=[0, 55])
402 fig_pt.update_yaxes(range=[-0.015, 0])
403
404 fig_ps.write_image("task2-1.pdf")
405 fig_pl.write_image("task2-2.pdf")
406 fig_pt.write_image("task2-3.pdf")
407
408 fig_ps.show()
409 fig_pl.show()
410 fig_pt.show()
411
412
413 # TASK A - TASK 3
414
415 # Video
416
417 # In[11]:
418
419
420 import cv2
421
422 # Load the video
423 input_video_path = 'Relax_video_V2.mp4'
424 output_video_path = 'Relax_video_V2_binary.mp4'
425 cap = cv2.VideoCapture(input_video_path)
426
427 # Get video properties
428 fps = cap.get(cv2.CAP_PROP_FPS)
429 frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
430 frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
431 fourcc = cv2.VideoWriter_fourcc(*'mp4v')
432
433 # Define the video writer for the output binary video
434 out = cv2.VideoWriter(output_video_path, fourcc, fps, (frame_width,
    frame_height), isColor=False)
435
436 # Process each frame
437 while cap.isOpened():
```

```
    ret, frame = cap.read()
    if not ret:
        break

    # Convert to grayscale
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Apply binary threshold
    _, binary_frame = cv2.threshold(gray_frame, 100, 255, cv2.THRESH_BINARY)

    # Write the binary frame to the output video
    out.write(binary_frame)

# Release video objects
cap.release()
out.release()

print(f"Binary video saved as '{output_video_path}'")


# In[57]:


import cv2
import numpy as np
import plotly.graph_objs as go

# Load the binary video
binary_video_path = 'Relax_video_V2_binary.mp4'
cap = cv2.VideoCapture(binary_video_path)

# Get video properties
fps = cap.get(cv2.CAP_PROP_FPS)
frames_per_half_second = int(fps / 2)
time_intervals = []  # Store the time intervals in seconds
white_pixel_sums = []  # Store the sum of white pixels for each half-second

current_sum = 0
frame_count = 0
time_point = 0.5  # Start with the first half-second interval

# Process each frame
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    # Count white pixels in the frame (255 is white in binary)
    white_pixel_count = np.sum(frame == 255)
    current_sum += white_pixel_count
    frame_count += 1

    # If half-second interval reached
    if frame_count >= frames_per_half_second:
```

```
        time_intervals.append(time_point)
        white_pixel_sums.append(current_sum)

        # Reset for the next half-second interval
        current_sum = 0
        frame_count = 0
        time_point += 0.5

# Release the video capture object
cap.release()

# Remove the first data point from both lists
time_intervals = time_intervals[1:]
white_pixel_sums = white_pixel_sums[1:]

# Create the plot with Plotly
fig = go.Figure()
fig.add_trace(go.Scatter(x=time_intervals, y=white_pixel_sums, mode='lines+markers', marker=dict(color='blue')))

# Update plot layout
fig.update_layout(
    title='Sum of White Pixels in Each Half-Second Interval (First Point Excluded)',
    xaxis_title='Time (s)',
    yaxis_title='Sum of White Pixels',
    template="plotly_white"
)

fig.show()


# In[64]:


import plotly.graph_objs as go

# Assume `white_pixel_sums` and `time_intervals` are available from previous code
# Calculate the stretch ratio
reference_sum = white_pixel_sums[1]  # Using the 1-second value as the reference
stretch_ratios = [pixel_sum / reference_sum for pixel_sum in white_pixel_sums]

# Create the plot for the stretch ratio
fig = go.Figure()
fig.add_trace(go.Scatter(x=time_intervals, y=stretch_ratios, mode='lines+markers', marker=dict(color='black')))

# Update plot layout
fig.update_layout(
    title='Cross-Directional Stretch (lambda)',
    xaxis_title='Time (s)',
    yaxis_title='Stretch (lambda)',
```

```
    template="plotly_white"
)

style_plot(fig, 'Time [s]', 'Stretch (lambdaT) [-]', 'Cross-Directional Stretch (lambda)')

fig.write_image("task3-2.pdf")

fig.show()


# In[65]:


import plotly.graph_objs as go

# Calculate vertical stretch based on u_corrected
vertical_stretch = [(L_0 - u) / L_0 for u in u_corrected]

# Plot with vertical stretch on x-axis and cross-directional stretch (stretch_ratios) on y-axis
fig = go.Figure()

fig.add_trace(go.Scatter(
    x=vertical_stretch,
    y=stretch_ratios,
    mode='lines+markers',
    marker=dict(color='blue')
))

# Update plot layout
fig.update_layout(
    title='Cross-Directional Stretch vs Vertical Stretch',
    xaxis_title='Stretch (lambda) [-]',
    yaxis_title='Cross-Directional Stretch (lambdaT) [-]',
    template="plotly_white"
)

style_plot(fig, 'Stretch (lambda) [-]', 'Cross-Directional Stretch (lambdaT) [-]', 'Cross-Directional Stretch vs Vertical Stretch')
fig.write_image("task3-3.pdf")

fig.show()


# In[68]:


import numpy as np
import plotly.graph_objs as go

# Step 1: Calculate vertical stretch based on u_corrected
vertical_stretch = [(L_0 - u) / L_0 for u in u_corrected]

```

```
# Step 2: Resample vertical stretch to match the length of stretch_ratios
num_vertical_stretch = len(vertical_stretch)
num_stretch_ratios = len(stretch_ratios)

# Generate original indices and new indices for resampling
original_indices = np.linspace(0, num_vertical_stretch - 1,
    num_vertical_stretch)
new_indices = np.linspace(0, num_vertical_stretch - 1, num_stretch_ratios)

# Interpolate vertical stretch to match the size of stretch_ratios
vertical_stretch_resampled = np.interp(new_indices, original_indices,
    vertical_stretch)

# Ensure the first and last values are the same
vertical_stretch_resampled[0] = vertical_stretch[0]  # Keep the first value
vertical_stretch_resampled[-1] = vertical_stretch[-1]  # Keep the last value

# Step 3: Plot with vertical stretch on x-axis and cross-directional stretch on
    y-axis
fig = go.Figure()

fig.add_trace(go.Scatter(
    x=vertical_stretch_resampled,
    y=stretch_ratios,
    mode='lines+markers',
    marker=dict(color='blue')
))

# Update plot layout
fig.update_layout(
    title='Cross-Directional Stretch vs Vertical Stretch',
    xaxis_title='Stretch (lambda) [-]',
    yaxis_title='Cross-Directional Stretch (lambdaT) [-]',
    template="plotly_white"
)

# Apply your styling function
style_plot(fig, 'Stretch (lambda) [-]', 'Cross-Directional Stretch (lambdaT)
    [-]', 'Cross-Directional Stretch vs Vertical Stretch')

# Save the plot as PDF
fig.write_image("task3-3.pdf")

# Show the plot
fig.show()


# TASK A - TASK 4

# In[69]:


lambda_T_max = max(stretch_ratios)
lambda_T_max
```

```
642
643
644 # TASK A - TASK 5
645
646 # In[179]:
647
648
649 import numpy as np
650 import plotly.graph_objects as go
651
652 # Define initial cross-sectional area (A) in mm^2
653 A = a * b  # initial cross-sectional area in mm^2
654
655 # Interpolate stretch_ratios and vertical_stretch to match the length of load
656 stretch_ratios_expanded = np.interp(np.arange(len(load)), np.linspace(0, len(
    load)-1, len(stretch_ratios)), stretch_ratios)
657
658 # Calculate Stretch (lambda) as 1 - (extension / L_0_rel)
659 stretch = 1 - (extension) / L_0_rel  # Expanded Stretch ratio
660
661 # Calculate True Stress (sig_true) as sig / (lambda^2)
662 true_stress = -load / (stretch_ratios_expanded ** 2 * A)  # True stress using
    actual cross-sectional area
663
664 # Calculate True Strain (eps_true) as ln(lambda)
665 true_strain = np.log(stretch)  # True strain as natural log of stretch
666
667 # Plot True Stress - True Strain (sig_true - eps_true) curve
668 fig_tt = go.Figure(data=[go.Scatter(x=true_strain, y=true_stress, mode='lines',
     name='True Stress - True Strain')])
669 style_plot(fig_tt, 'True Strain (eps_true) [-]', 'True Stress (sig_true) [MPa]'
    , 'True Stress - True Strain (sig_true - eps_true) Curve')
670
671 # Set axis ranges (adjust as needed)
672 fig_tt.update_xaxes(range=[-1.4, 0.1])
673
674 # Save and display the plot
675 fig_tt.write_image("task2-true_stress_true_strain.pdf")
676 fig_tt.show()
677
678
679 # TASK B - TASK 1
680
681 # In[92]:
682
683
684 def style_plot(fig, x_title, y_title, title):
685     # Add gradient background using shapes
686     fig.update_layout(
687         title=title,
688         plot_bgcolor='rgba(255, 255, 255, 0)',  # Transparent plot background
689         shapes=[
690             dict(
691                 type='rect',
```

```
                xref='paper', yref='paper',
                x0=0, x1=1, y0=0, y1=1,
                fillcolor='rgba(211, 211, 211, 0.15)',  # Grey with 15% opacity
                line=dict(width=0)
            ),
        ],
        xaxis=dict(
            title=x_title,
            gridcolor='lightgrey',  # Lighter grid color for more hollow effect
            gridwidth=1,
            zerolinecolor='black',  # Zero line color
            zerolinewidth=2,
            showgrid=True,
            linecolor='black'  # X-axis line color
        ),
        yaxis=dict(
            title=y_title,
            gridcolor='lightgrey',  # Lighter grid color for more hollow effect
            gridwidth=1,
            zerolinecolor='black',  # Zero line color
            zerolinewidth=2,
            showgrid=True,
            linecolor='black'  # Y-axis line color
        ),
        font=dict(color='black'),
    )

    # Set colors for traces
    if len(fig.data) >= 1:
        fig.update_traces(line=dict(color='red'), selector=dict(name='
    Compensated F-u'))  # Default color for the first trace
    if len(fig.data) >= 2:
        fig.data[1].line.color = 'green'  # Set the second trace color to green


# In[97]:


# Import necessary libraries
import pandas as pd
import plotly.graph_objs as go

# Define basic parameters
H_0_cyc = 75  # mm, initial tool separation
u_1 = 10.5  # mm, displacement of the first step in the uploading
delta_u = 5.5  # mm, incremental displacement
N = 9  # Number of increments
v_cyc = 40  # mm/min, cross-head speed during cyclic test
delta_t_rel = 30  # s, relaxation time
a_cyc = 77.71 # mm
b_cyc = 75.11 # mm
A_cyc = a_cyc * b_cyc

# Load the cyclic test CSV file into a DataFrame
```

```
file_name_cyc = 'CYCLIC_RawData_18.csv'
data_cyc = pd.read_csv(file_name_cyc)

# Convert columns to appropriate data types
data_cyc['Time'] = pd.to_numeric(data_cyc.iloc[:, 0], errors='coerce')  # Convert Time to float
data_cyc['Extension'] = pd.to_numeric(data_cyc.iloc[:, 1], errors='coerce')  # Convert Extension to float
data_cyc['Load'] = pd.to_numeric(data_cyc.iloc[:, 2], errors='coerce')  # Convert Load to float

# Extract the columns: Time, Extension, Load
time_cyc = data_cyc['Time']
extension_cyc = data_cyc['Extension']
load_cyc = data_cyc['Load']

# Define the exact height of the cyclic test specimen (L_0^(cyc))
L_0_cyc = H_0_cyc - u_1  # mm

# Perform slack correction for displacement (extension)
slack_threshold_cyc = 0.1  # N, threshold for detecting when the load becomes significant
slack_index_cyc = (load_cyc > slack_threshold_cyc).idxmax()  # Get index of first significant load

# Slice the data from slack_index onwards
time_corrected_cyc = time_cyc[slack_index_cyc:].reset_index(drop=True)  # Time starting from slack point
u_corrected_cyc = (extension_cyc - extension_cyc[slack_index_cyc])[slack_index_cyc:].reset_index(drop=True)  # Corrected extension
load_corrected_cyc = load_cyc[slack_index_cyc:].reset_index(drop=True)  # Corresponding load values

# Plot compensated Force-Displacement (F-u) curve
compensated_force_displacement_cyc = go.Scatter(x=u_corrected_cyc, y=load_corrected_cyc, mode='lines', name='Compensated F-u')
fig_fu_cyc = go.Figure(data=[compensated_force_displacement_cyc])
style_plot(fig_fu_cyc, 'Corrected Displacement u (mm)', 'Force F (N)', 'Compensated Force-Displacement (F-u) Curve')

# Plot Force-Time (F-t) curve using corrected time
force_time_cyc = go.Scatter(x=time_corrected_cyc, y=load_corrected_cyc, mode='lines', name='F-t')
fig_ft_cyc = go.Figure(data=[force_time_cyc])
style_plot(fig_ft_cyc, 'Time (s)', 'Force F (N)', 'Force-Time (F-t) Curve')

fig_fu_cyc.write_image("taskB1-1.pdf")

fig_ft_cyc.write_image("taskB1-2.pdf")

# Show both plots
fig_fu_cyc.show()
fig_ft_cyc.show()

```

```
788
789 # In[95]:
790
791
792 # Import necessary libraries
793 import pandas as pd
794 import plotly.graph_objs as go
795
796 # Define basic parameters
797 H_0_cyc = 75  # mm, initial tool separation
798 u_1 = 10.5  # mm, displacement of the first step in the uploading
799 delta_u = 5.5  # mm, incremental displacement
800 N = 9  # Number of increments
801 v_cyc = 40  # mm/min, cross-head speed during cyclic test
802 delta_t_rel = 30  # s, relaxation time
803
804 # Load the cyclic test CSV file into a DataFrame
805 file_name_cyc = 'CYCLIC_RawData_18.csv'
806 data_cyc = pd.read_csv(file_name_cyc)
807
808 # Convert columns to appropriate data types
809 data_cyc['Time'] = pd.to_numeric(data_cyc.iloc[:, 0], errors='coerce')  #
    Convert Time to float
810 data_cyc['Extension'] = pd.to_numeric(data_cyc.iloc[:, 1], errors='coerce')  #
    Convert Extension to float
811 data_cyc['Load'] = pd.to_numeric(data_cyc.iloc[:, 2], errors='coerce')  #
    Convert Load to float
812
813 # Extract the columns: Time, Extension, Load
814 time_cyc = data_cyc['Time']
815 extension_cyc = data_cyc['Extension']
816 load_cyc = data_cyc['Load']
817
818 # Define the exact height of the cyclic test specimen (L_0^(cyc))
819 L_0_cyc = H_0_cyc - 6.3  # mm
820
821 # Plot Force-Displacement (F-u) curve without slack correction
822 force_displacement_cyc = go.Scatter(x=extension_cyc, y=load_cyc, mode='lines',
    name='F-u')
823 fig_fu_cyc = go.Figure(data=[force_displacement_cyc])
824 style_plot(fig_fu_cyc, 'Displacement u (mm)', 'Force F (N)', 'Force-
    Displacement (F-u) Curve')
825
826 # Plot Force-Time (F-t) curve without slack correction
827 force_time_cyc = go.Scatter(x=time_cyc, y=load_cyc, mode='lines', name='F-t')
828 fig_ft_cyc = go.Figure(data=[force_time_cyc])
829 style_plot(fig_ft_cyc, 'Time (s)', 'Force F (N)', 'Force-Time (F-t) Curve')
830
831 # Save plots as PDFs
832 fig_fu_cyc.write_image("taskB1-1-noslack.pdf")
833 fig_ft_cyc.write_image("taskB1-2-noslack.pdf")
834
835 # Show both plots
836 fig_fu_cyc.show()
```

```
fig_ft_cyc.show()


# In[96]:


# Import necessary libraries
import pandas as pd
import plotly.graph_objs as go

# Define basic parameters
H_0_cyc = 75  # mm, initial tool separation
u_1 = 10.5  # mm, displacement of the first step in the uploading
delta_u = 5.5  # mm, incremental displacement
N = 9  # Number of increments
v_cyc = 40  # mm/min, cross-head speed during cyclic test
delta_t_rel = 30  # s, relaxation time

# Load the cyclic test CSV file into a DataFrame
file_name_cyc = 'CYCLIC_RawData_18.csv'
data_cyc = pd.read_csv(file_name_cyc)

# Convert columns to appropriate data types
data_cyc['Time'] = pd.to_numeric(data_cyc.iloc[:, 0], errors='coerce')  # Convert Time to float
data_cyc['Extension'] = pd.to_numeric(data_cyc.iloc[:, 1], errors='coerce')  # Convert Extension to float
data_cyc['Load'] = pd.to_numeric(data_cyc.iloc[:, 2], errors='coerce')  # Convert Load to float

# Extract the columns: Time, Extension, Load
time_cyc = data_cyc['Time']
extension_cyc = data_cyc['Extension']
load_cyc = data_cyc['Load']

# Define the exact height of the cyclic test specimen (L_0^(cyc))
L_0_cyc = H_0_cyc - 6.3  # mm

# Plot Force-Displacement (F-u) curve without slack correction
force_displacement_cyc = go.Scatter(x=extension_cyc, y=load_cyc, mode='lines', name='F-u')
fig_fu_cyc = go.Figure(data=[force_displacement_cyc])
style_plot(fig_fu_cyc, 'Displacement u (mm)', 'Force F (N)', 'Force-Displacement (F-u) Curve')

# Plot Force-Time (F-t) curve without slack correction
force_time_cyc = go.Scatter(x=time_cyc, y=load_cyc, mode='lines', name='F-t')
fig_ft_cyc = go.Figure(data=[force_time_cyc])
style_plot(fig_ft_cyc, 'Time (s)', 'Force F (N)', 'Force-Time (F-t) Curve')

fig_fu_cyc.update_xaxes(range=[5, 8])  # Set x-axis range
fig_fu_cyc.update_yaxes(range=[-2, 7])

# Save plots as PDFs
```

```
fig_fu_cyc.write_image("taskB1-1-noslack-close.pdf")
fig_ft_cyc.write_image("taskB1-2-noslack.pdf")

# Show both plots
fig_fu_cyc.show()
fig_ft_cyc.show()


# TASK B - TASK 2

# In[99]:


extension_cyc = extension_cyc[slack_index_cyc:] - extension_cyc[slack_index_cyc]
load_cyc = load_cyc[slack_index_cyc:] - load_cyc[slack_index_cyc]


# In[104]:


L_0_cyc = H_0_cyc - 6.5


# In[125]:


# Calculate engineering stress (P)
engineering_stress = load_cyc / A_cyc  *-1
stretch = 1 - extension_cyc/L_0_cyc
engineering_strain = stretch - 1 # Extension divided by initial length

# Plot Engineering Stress vs Engineering Strain (P - eps)
fig_stress_strain = go.Figure()
fig_stress_strain.add_trace(go.Scatter(x=engineering_strain, y=engineering_stress, mode='lines', name='P - eps'))
style_plot(fig_stress_strain, 'Engineering Strain (eps)', 'Engineering Stress (P) [MPa]', 'Engineering Stress vs Engineering Strain (P - eps)')
fig_stress_strain.update_xaxes(range=[-0.8, 0.1])  # Set x-axis range

fig_stress_strain.write_image("taskB2-1.pdf")
fig_stress_strain.show()

# Plot Engineering Stress vs Stretch (P - lambda)
fig_stress_stretch = go.Figure()
fig_stress_stretch.add_trace(go.Scatter(x=stretch, y=engineering_stress, mode='lines', name='P - lambda'))
style_plot(fig_stress_stretch, 'Stretch (lambda)', 'Engineering Stress (P) [MPa]', 'Engineering Stress vs Stretch (P - lambda)')
fig_stress_stretch.update_xaxes(range=[0.2, 1.1])  # Set x-axis range

fig_stress_stretch.write_image("taskB2-2.pdf")
fig_stress_stretch.show()

```

```
# Plot Engineering Stress vs Time (P - t)
fig_stress_time = go.Figure()
fig_stress_time.add_trace(go.Scatter(x=time_cyc, y=engineering_stress, mode='lines', name='P - t'))
style_plot(fig_stress_time, 'Time (s)', 'Engineering Stress (P) [MPa]', 'Engineering Stress vs Time (P - t)')

fig_stress_time.write_image("taskB2-3.pdf")
fig_stress_time.show()


# TASK B - TASK 3

# In[126]:


import pandas as pd
import numpy as np
import plotly.graph_objs as go
from scipy.signal import argrelextrema

# Load the cyclic test CSV file into a DataFrame
file_name_cyc = 'CYCLIC_RawData_18.csv'
data_cyc = pd.read_csv(file_name_cyc)

# Convert columns to appropriate data types
data_cyc['Time'] = pd.to_numeric(data_cyc.iloc[:, 0], errors='coerce')
data_cyc['Extension'] = pd.to_numeric(data_cyc.iloc[:, 1], errors='coerce')
data_cyc['Load'] = pd.to_numeric(data_cyc.iloc[:, 2], errors='coerce')

# Extract columns: Time, Extension, Load
time_cyc = data_cyc['Time']
extension_cyc = data_cyc['Extension']
load_cyc = data_cyc['Load']

# Define parameters
L_0_cyc = 68.5  # mm, original height of the specimen

# Perform slack correction for displacement (extension)
slack_threshold_cyc = 0.1  # N, threshold for detecting when load becomes significant
slack_index_cyc = (load_cyc > slack_threshold_cyc).idxmax()  # Index of first significant load

# Slice the data from slack_index onwards
time_corrected_cyc = time_cyc[slack_index_cyc:].reset_index(drop=True)
u_corrected_cyc = (extension_cyc - extension_cyc[slack_index_cyc])[slack_index_cyc:].reset_index(drop=True)
load_corrected_cyc = load_cyc[slack_index_cyc:].reset_index(drop=True)

# Smoothing load data to reduce noise for extrema detection
smoothed_load = load_corrected_cyc.rolling(window=300, center=True).mean()  # Adjust as necessary

```

```
# Plotting for visualization of engineering stress vs stretch
stress_vs_stretch = go.Scatter(x=stretch, y=engineering_stress, mode='lines',
    name='Engineering Stress vs Stretch')
fig_stress_stretch = go.Figure(data=[stress_vs_stretch])
fig_stress_stretch.update_layout(title='Engineering Stress vs Stretch',
                                 xaxis_title='Stretch (lambda)',
                                 yaxis_title='Engineering Stress (MPa)')
fig_stress_stretch.show()

# Print the lists of unloading maxima and uploading minima
print("Unloading Maxima:", unloading_max_values)
print("Uploading Minima:", uploading_min_values)


# In[181]:


import pandas as pd
import numpy as np
import plotly.graph_objs as go
from scipy.signal import argrelextrema

# Load the cyclic test CSV file into a DataFrame
file_name_cyc = 'CYCLIC_RawData_18.csv'
data_cyc = pd.read_csv(file_name_cyc)

# Convert columns to appropriate data types
data_cyc['Time'] = pd.to_numeric(data_cyc.iloc[:, 0], errors='coerce')
data_cyc['Extension'] = pd.to_numeric(data_cyc.iloc[:, 1], errors='coerce')
data_cyc['Load'] = pd.to_numeric(data_cyc.iloc[:, 2], errors='coerce')

# Extract columns: Time, Extension, Load
time_cyc = data_cyc['Time']
extension_cyc = data_cyc['Extension']
load_cyc = data_cyc['Load']

# Define the exact height of the cyclic test specimen (L_0^(cyc))
L_0_cyc = 68.5 # mm

# Perform slack correction for displacement (extension)
slack_threshold_cyc = 0.1  # N, threshold for detecting when load becomes
    significant
slack_index_cyc = (load_cyc > slack_threshold_cyc).idxmax()  # Index of first
    significant load

# Slice the data from slack_index onwards
time_corrected_cyc = time_cyc[slack_index_cyc:].reset_index(drop=True)
u_corrected_cyc = (extension_cyc - extension_cyc[slack_index_cyc])[
    slack_index_cyc:].reset_index(drop=True)
load_corrected_cyc = load_cyc[slack_index_cyc:].reset_index(drop=True)

# Smoothing load data to reduce noise for extrema detection
smoothed_load = load_corrected_cyc.rolling(window=300, center=True).mean()  #
    Adjust as necessary
```

```
1032
1033 # Find the global maximum index separating uploading and unloading phases
1034 global_max_index = smoothed_load.idxmax()
1035
1036 # Find approximate local maxima in the unloading phase (after the global
     maximum)
1037 unloading_approx_max_indices = argrelextrema(smoothed_load[global_max_index:].
     values, np.greater, order=70)[0] + global_max_index
1038
1039 # Refine local maxima by ensuring only one maximum per cycle
1040 unloading_max_indices = []
1041 for i in range(1, len(unloading_approx_max_indices)):
1042     start_idx = unloading_approx_max_indices[i-1]
1043     end_idx = unloading_approx_max_indices[i]
1044     exact_max_idx = load_corrected_cyc[start_idx:end_idx].idxmax()
1045     unloading_max_indices.append(exact_max_idx)
1046
1047 # Extract values of the exact unloading maxima
1048 unloading_max_values = load_corrected_cyc[unloading_max_indices].values
1049
1050 # Find approximate local minima in the uploading phase (before the global
     maximum)
1051 uploading_approx_min_indices = argrelextrema(smoothed_load[:global_max_index].
     values, np.less, order=50)[0]
1052
1053 # Add the last index to ensure we include it in the search for the final local
     minimum
1054 if global_max_index < len(smoothed_load):
1055     uploading_approx_min_indices = np.append(uploading_approx_min_indices,
     global_max_index)
1056
1057 # Refine local minima by ensuring only one minimum per cycle
1058 uploading_min_indices = []
1059 for i in range(1, len(uploading_approx_min_indices)):
1060     start_idx = uploading_approx_min_indices[i-1]
1061     end_idx = uploading_approx_min_indices[i]
1062     exact_min_idx = load_corrected_cyc[start_idx:end_idx].idxmin()
1063     uploading_min_indices.append(exact_min_idx)
1064
1065 # Check the last segment from the last detected minimum to the global maximum
1066 if uploading_min_indices:
1067     last_segment_min_idx = load_corrected_cyc[uploading_min_indices[-1]:
     global_max_index].idxmin()
1068     if last_segment_min_idx not in uploading_min_indices:
1069         uploading_min_indices.append(last_segment_min_idx)
1070
1071 # Extract values of the exact uploading minima
1072 uploading_min_values = load_corrected_cyc[uploading_min_indices].values
1073
1074 # Plotting for visualization
1075 compensated_force_displacement_cyc = go.Scatter(x=u_corrected_cyc, y=
     load_corrected_cyc, mode='lines', name='Compensated F-u')
1076 unloading_max_scatter = go.Scatter(x=u_corrected_cyc[unloading_max_indices], y=
     unloading_max_values, mode='markers', name='Unloading Local Maxima', marker=
```

```
    dict(color='red', size=8))
uploading_min_scatter = go.Scatter(x=u_corrected_cyc[uploading_min_indices], y=
    uploading_min_values, mode='markers', name='Uploading Local Minima', marker=
    dict(color='blue', size=8))

fig_fu_cyc = go.Figure(data=[compensated_force_displacement_cyc,
    unloading_max_scatter, uploading_min_scatter])
style_plot(fig_fu_cyc, 'Corrected Displacement u (mm)', 'Force F (N)', '
    Compensated Force-Displacement (F-u) Curve with Local Maxima and Minima')
fig_fu_cyc.show()

# Print the lists of unloading maxima and uploading minima
print("Unloading Maxima:", unloading_max_values)
print("Uploading Minima:", uploading_min_values)


# In[182]:


unloading_max_values = unloading_max_values[:-2]

# Print the updated unloading maxima list
print("Updated Unloading Maxima:", unloading_max_values)
print("Uploading Minima:", uploading_min_values)


# In[183]:


# Plotting for visualization
compensated_force_displacement_cyc = go.Scatter(x=u_corrected_cyc, y=
    load_corrected_cyc, mode='lines', name='Compensated F-u')
unloading_max_scatter = go.Scatter(x=u_corrected_cyc[unloading_max_indices], y=
    unloading_max_values, mode='markers', name='Unloading Local Maxima', marker=
    dict(color='red', size=8))
uploading_min_scatter = go.Scatter(x=u_corrected_cyc[uploading_min_indices], y=
    uploading_min_values, mode='markers', name='Uploading Local Minima', marker=
    dict(color='blue', size=8))

fig_fu_cyc = go.Figure(data=[compensated_force_displacement_cyc,
    unloading_max_scatter, uploading_min_scatter])
style_plot(fig_fu_cyc, 'Corrected Displacement u (mm)', 'Force F (N)', '
    Compensated Force-Displacement (F-u) Curve with Local Maxima and Minima')
fig_fu_cyc.show()


# In[184]:


# Assuming unloading_max_values and uploading_min_values are already defined
    and have the same length
midpoints = [(unloading_max_values[i] + uploading_min_values[i]) / 2 for i in
    range(len(unloading_max_values))]

```

```
# Print the midpoints list
print("Midpoints:", midpoints)


# In[185]:


# Assuming you have the following lists:
# unloading_max_values, uploading_min_values
# And you also have the corresponding extension values in `u_corrected_cyc`

# Ensure the lists have the same length
if len(unloading_max_values) == len(uploading_min_values):
    midpoints = []
    midpoint_x_values = []

    for i in range(len(unloading_max_values)):
        midpoint_value = (unloading_max_values[len(unloading_max_values)-i-1] + uploading_min_values[i]) / 2
        midpoint_x_value = u_corrected_cyc[uploading_min_indices[i]]  # Use the corresponding index for x value

        midpoints.append(midpoint_value)
        midpoint_x_values.append(midpoint_x_value)

    # Print the midpoints and their corresponding x-values
    for x, midpoint in zip(midpoint_x_values, midpoints):
        print(f"X: {x}, Midpoint: {midpoint}")
else:
    print("Error: unloading_max_values and uploading_min_values must have the same length.")


# In[186]:


# Plotting for visualization
compensated_force_displacement_cyc = go.Scatter(
    x=u_corrected_cyc,
    y=load_corrected_cyc,
    mode='lines',
    name='Compensated F-u'
)

unloading_max_scatter = go.Scatter(
    x=u_corrected_cyc[unloading_max_indices],
    y=unloading_max_values,
    mode='markers',
    name='Unloading Local Maxima',
    marker=dict(color='red', size=8)
)

uploading_min_scatter = go.Scatter(
    x=u_corrected_cyc[uploading_min_indices],
```

```
    y=uploading_min_values,
    mode='markers',
    name='Uploading Local Minima',
    marker=dict(color='blue', size=8)
)

# Create a scatter plot for midpoints
midpoints_scatter = go.Scatter(
    x=midpoint_x_values,  # Corresponding x-values for midpoints
    y=midpoints,          # Midpoint values
    mode='markers',
    name='Midpoints',
    marker=dict(color='green', size=8, symbol='diamond')
)

# Create the figure with all traces
fig_fu_cyc = go.Figure(data=[
    compensated_force_displacement_cyc,
    unloading_max_scatter,
    uploading_min_scatter,
    midpoints_scatter  # Add midpoints to the figure
])

# Style and show the plot
style_plot(fig_fu_cyc, 'Corrected Displacement u (mm)', 'Force F (N)', '
    Compensated Force-Displacement (F-u) Curve with Local Maxima, Minima, and
    Midpoints')
fig_fu_cyc.show()


# In[189]:


# Add (0, 0) to midpoints and corresponding x-values
midpoints = [0] + midpoints  # Add 0 at the start of midpoints list
midpoint_x_values = [0] + midpoint_x_values  # Add 0 at the start of x-values

# Specify the degree of the polynomial
degree = 5 # Change this if needed

# Fit a polynomial to the midpoints
coefficients = np.polyfit(midpoint_x_values, midpoints, degree)

# Generate x-values for the polynomial line (for smooth curve)
x_poly = np.linspace(min(midpoint_x_values), max(midpoint_x_values)*1.1, 200)

# Evaluate the polynomial at these x-values
y_poly = np.polyval(coefficients, x_poly)

# Plotting for visualization
compensated_force_displacement_cyc = go.Scatter(
    x=u_corrected_cyc,
    y=load_corrected_cyc,
    mode='lines',
```

```
    name='Compensated F-u'
)

unloading_max_scatter = go.Scatter(
    x=u_corrected_cyc[unloading_max_indices],
    y=unloading_max_values,
    mode='markers',
    name='Unloading Local Maxima',
    marker=dict(color='red', size=8)
)

uploading_min_scatter = go.Scatter(
    x=u_corrected_cyc[uploading_min_indices],
    y=uploading_min_values,
    mode='markers',
    name='Uploading Local Minima',
    marker=dict(color='blue', size=8)
)

# Midpoints scatter plot
midpoints_scatter = go.Scatter(
    x=midpoint_x_values,
    y=midpoints,
    mode='markers',
    name='Midpoints',
    marker=dict(color='green', size=8, symbol='diamond')
)

# Polynomial line for midpoints
polynomial_line = go.Scatter(
    x=x_poly,
    y=y_poly,
    mode='lines',
    name='Polynomial Fit',
    line=dict(color='black', width=2)
)

# Create the figure with all traces
fig_fu_cyc = go.Figure(data=[
    compensated_force_displacement_cyc,
    unloading_max_scatter,
    uploading_min_scatter,
    midpoints_scatter,
    polynomial_line  # Add polynomial line to the figure
])

# Style and show the plot
style_plot(fig_fu_cyc, 'Corrected Displacement u (mm)', 'Force F (N)', '
    Compensated Force-Displacement (F-u) Curve with Local Maxima, Minima,
    Midpoints, and Polynomial Fit')
fig_fu_cyc.show()


# TASK B - TASK 4
```

```
1272
1273 # In[190]:
1274
1275
1276 from PIL import Image
1277 import numpy as np
1278
1279 # Load the images
1280 image_kicsi = Image.open('taskB4-2_kicsi.png').convert('L')  # Convert to
     grayscale
1281 image_nagy = Image.open('taskB4-2_nagy.png').convert('L')    # Convert to
     grayscale
1282
1283 # Convert images to numpy arrays
1284 array_kicsi = np.array(image_kicsi)
1285 array_nagy = np.array(image_nagy)
1286
1287 # Count white pixels (assuming white is 255 in binary images)
1288 num_white_kicsi = np.sum(array_kicsi == 255)
1289 num_white_nagy = np.sum(array_nagy == 255)
1290
1291 # Calculate the ratio
1292 if num_white_nagy != 0:
1293     ratio = num_white_nagy / num_white_kicsi
1294 else:
1295     ratio = None  # Handle case where there are no white pixels in the large
     image
1296
1297 # Print results
1298 print(f"Number of white pixels in taskB4-2_kicsi: {num_white_kicsi}")
1299 print(f"Number of white pixels in taskB4-2_nagy: {num_white_nagy}")
1300 print(f"Ratio (taskB4-2_nagy / taskB4-2_kicsi): {ratio}")
```