



University of Illinois Urbana-Champaign

Theoretical and Applied Mechanics

Class: TAM 251

Assignment: D4

Originated by: TFG

Signed: TK

Date: 9/19/26

Checked by: Taewoo Kim

Signed: TK

Date: 9/19/26

1 Load modules

```
[45]: import plotly.graph_objects as go          # for interactive plots
      #!pip install xlrd                        # for reading excel
      import pandas as pd                     # to store data in pandas dataframes
      import os                               # to access the files
      from scipy.stats import linregress       # find slope with linear regression
      from plotly.subplots import make_subplots # for subplots
      import numpy as np                      # numpy commands
      from numpy import trapezoid              # numerical integration
```

```
[46]: import plotly.io as pio
      pio.renderers.default = "notebook+pdf"
```

2 Parts 1 & 2. Import the test data

```
[47]: folder_path = 'C:\\Users\\tfg\\Downloads\\TestData'

      # Dictionary to store DataFrames
      excel_dataframes = {}

      # Iterate over files in the specified folder
      for filename in os.listdir(folder_path):
          if filename.endswith('.xlsx'):
              file_path = os.path.join(folder_path, filename)
              df_name = os.path.splitext(filename)[0] # Use filename without
              ↪extension as DataFrame name
              excel_dataframes[df_name] = pd.read_excel(file_path)
              print(f"Successfully loaded '{filename}' into DataFrame '{df_name}'.")

      # Define a mapping for column renaming
      column_rename_map = {
          'Horário [s]': 'Time (s)',
          'Extensão [mm]': 'Extension (mm)',
```

```

        'Carga [N]': 'Load (N)'
    }

    # Iterate through each DataFrame in the dictionary and rename columns
    for df_name, df in excel_dataframes.items():
        columns_to_rename = {old_name: new_name for old_name,
                             new_name in column_rename_map.items() if old_name in_
        df.columns}
        df.rename(columns=columns_to_rename, inplace=True)
        print(f"Renamed columns in DataFrame '{df_name}'")

```

```

Successfully loaded '0_1.xlsx' into DataFrame '0_1'.
Successfully loaded '0_2.xlsx' into DataFrame '0_2'.
Successfully loaded '0_3.xlsx' into DataFrame '0_3'.
Successfully loaded '0_4.xlsx' into DataFrame '0_4'.
Successfully loaded '0_5.xlsx' into DataFrame '0_5'.
Successfully loaded '45_1.xlsx' into DataFrame '45_1'.
Successfully loaded '45_2.xlsx' into DataFrame '45_2'.
Successfully loaded '45_3.xlsx' into DataFrame '45_3'.
Successfully loaded '45_4.xlsx' into DataFrame '45_4'.
Successfully loaded '45_5.xlsx' into DataFrame '45_5'.
Successfully loaded '90_1.xlsx' into DataFrame '90_1'.
Successfully loaded '90_2.xlsx' into DataFrame '90_2'.
Successfully loaded '90_3.xlsx' into DataFrame '90_3'.
Successfully loaded '90_4.xlsx' into DataFrame '90_4'.
Successfully loaded '90_5.xlsx' into DataFrame '90_5'.
Renamed columns in DataFrame '0_1'.
Renamed columns in DataFrame '0_2'.
Renamed columns in DataFrame '0_3'.
Renamed columns in DataFrame '0_4'.
Renamed columns in DataFrame '0_5'.
Renamed columns in DataFrame '45_1'.
Renamed columns in DataFrame '45_2'.
Renamed columns in DataFrame '45_3'.
Renamed columns in DataFrame '45_4'.
Renamed columns in DataFrame '45_5'.
Renamed columns in DataFrame '90_1'.
Renamed columns in DataFrame '90_2'.
Renamed columns in DataFrame '90_3'.
Renamed columns in DataFrame '90_4'.
Renamed columns in DataFrame '90_5'.

```

```

[48]: # make sure it worked by viewing some of the data
display(excel_dataframes['0_1'].head())

```

	Time (s)	Extension (mm)	Load (N)
0 1	0	0	-0.21
1 2	0.1	0.00582	1.372

2	3	0.2	0.01481	1.652
3	4	0.3	0.02311	1.041
4	5	0.4	0.0315	0.7279

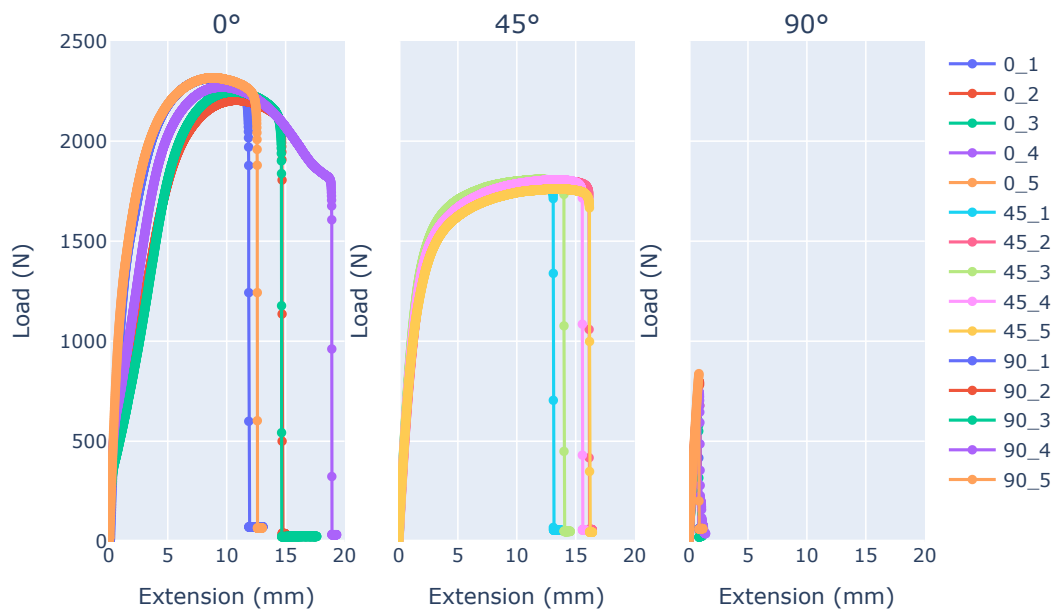
3 Define a Plotting Function

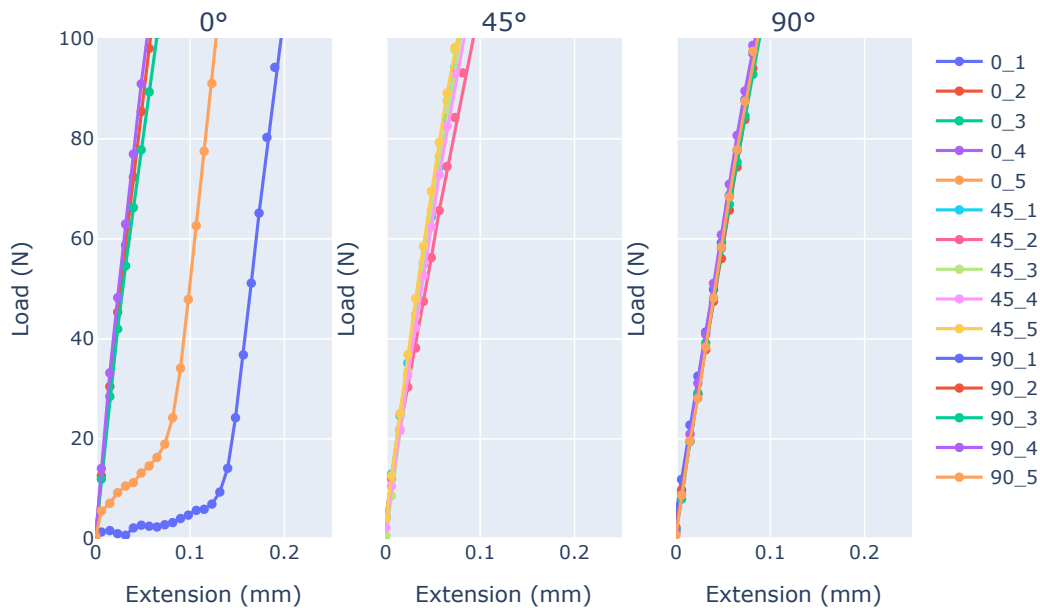
```
[49]: # make a function to plot all the load-deformation curves
def plotter(excel_dataframes,xcol,ycol,xlims,ylims):
    # pass in the dictionary of dataframes and specify what to plot
    prefixes = ['0_', '45_', '90_']
    fig = make_subplots(
        rows=1,
        cols=3,
        subplot_titles=['0°', '45°', '90°'],
        shared_yaxes=True
    )
    for col, prefix in enumerate(prefixes, start=1):
        # Filter dataframes for the current prefix
        group_dfs = {
            name: df
            for name, df in excel_dataframes.items()
            if name.startswith(prefix)
        }
        sorted_group_dfs = sorted(group_dfs.items(), key=lambda item: item[0])
        for df_name, df in sorted_group_dfs:
            if {xcol, ycol}.issubset(df.columns):
                fig.add_trace(
                    go.Scatter(
                        x=df[xcol],
                        y=df[ycol],
                        mode='lines+markers',
                        name=df_name,
                        showlegend=True
                    ),
                    row=1,
                    col=col
                )
    fig.update_layout(hovermode='x unified')
    fig.update_xaxes(title_text=xcol)
    fig.update_yaxes(title_text=ycol)
    fig.update_xaxes(range=xlims)
    fig.update_yaxes(range=ylims)
    fig.show()
```

4 Part 3. Plot Force vs Deflection

```
[50]: # Plot the entire curve (up to 20mm and 2500N)
      plotter(excel_dataframes,'Extension (mm)', 'Load (N)', [0,20],[0,2500])

      # show just the initial region (up to 0.25mm and 100N)
      plotter(excel_dataframes,'Extension (mm)', 'Load (N)', [0,0.25],[0,100])
```





5 Part 4. Data Cleaning

0_1 and 0_5 have a zero load extension, this is likely taking up some 'slack' in the system and is not representative of the actual material behavior. These data points should be removed, and the curves shifted in both axes to the new origin.

```
[51]: # row 17 of '0_1' is the should be the new 0,0 point
      origin = 17

      excel_dataframes['0_1']["Extension (mm)"] -= excel_dataframes['0_1'].
      ↪loc[origin, "Extension (mm)"]
      excel_dataframes['0_1']["Load (N)"] -= excel_dataframes['0_1'].loc[origin,
      ↪"Load (N)"]

      # Keep only rows from the origin onward
      excel_dataframes['0_1'] = excel_dataframes['0_1'].loc[origin:].
      ↪reset_index(drop=True)
      excel_dataframes['0_1'].head(20)

      # (this code is not great, running this cell again will trim the data again, so
      ↪make sure it only runs once)
```

```
[51]:
```

		Time (s)	Extension (mm)	Load (N)
0	18	1.7	0	0
1	19	1.8	0.0083	10.09
2	20	1.9	0.01663	22.68
3	21	2	0.02503	37.01
4	22	2.1	0.03331	51.02
5	23	2.2	0.04163	66.15
6	24	2.3	0.05002	80.17
7	25	2.4	0.05832	95.65
8	26	2.5	0.06662	110.6
9	27	2.6	0.07501	125.7
10	28	2.7	0.08335	140.6
11	29	2.8	0.09162	154.9
12	30	2.9	0.1	169.1
13	31	3	0.1084	184.7
14	32	3.1	0.1166	198.8
15	33	3.2	0.125	213.7
16	34	3.3	0.1334	227.9
17	35	3.4	0.1416	242.8
18	36	3.5	0.15	256.9
19	37	3.6	0.1584	271.1

```
[52]: # row 11 of '0_5' is the should be the new 0,0 point
origin = 11

excel_dataframes['0_5']["Extension (mm)"] -= excel_dataframes['0_5'].
↳loc[origin, "Extension (mm)"]
excel_dataframes['0_5']["Load (N)"] -= excel_dataframes['0_5'].loc[origin,
↳"Load (N)"]

# Keep only rows from the origin onward
excel_dataframes['0_5'] = excel_dataframes['0_5'].loc[origin:].
↳reset_index(drop=True)
excel_dataframes['0_5'].head(20)

# (this code is not great, running this cell again will trim the data again, so
↳make sure it only runs once)
```

```
[52]:
```

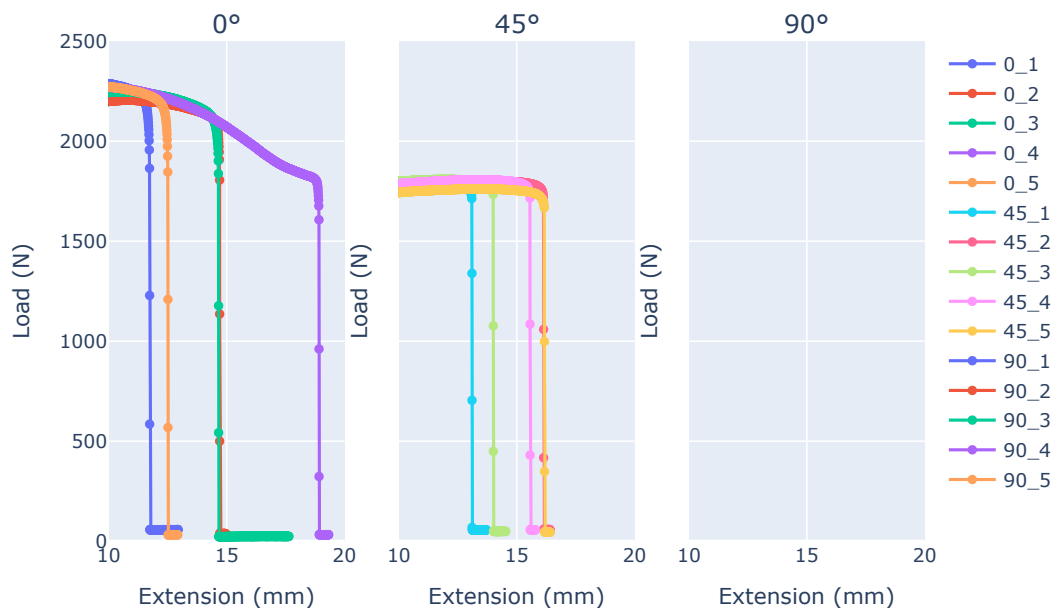
		Time (s)	Extension (mm)	Load (N)
0	12	1.1	0	0
1	13	1.2	0.00829	13.75
2	14	1.3	0.0167	28.49
3	15	1.4	0.02504	43.38
4	16	1.5	0.03331	56.94
5	17	1.6	0.04167	71.81
6	18	1.7	0.05001	87.47
7	19	1.8	0.05831	102.1

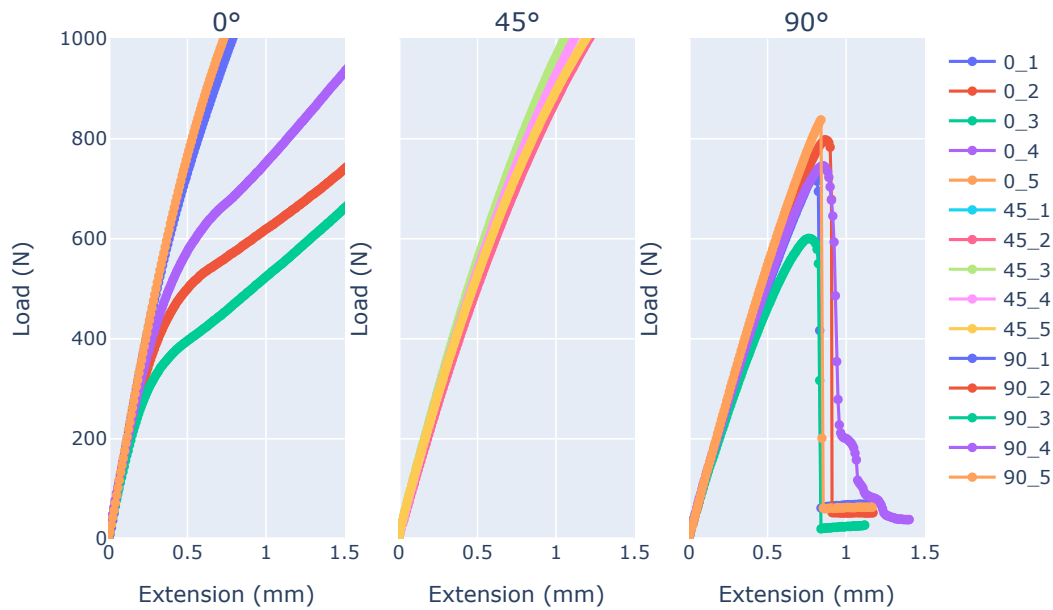
8	20	1.9	0.06664	117.2
9	21	2	0.07504	133.3
10	22	2.1	0.08332	147.3
11	23	2.2	0.09164	161.4
12	24	2.3	0.1	175.9
13	25	2.4	0.1083	191
14	26	2.5	0.1166	203.9
15	27	2.6	0.125	219.4
16	28	2.7	0.1334	233.5
17	29	2.8	0.1416	248.1
18	30	2.9	0.15	262.3
19	31	3	0.1584	276.1

Plot the load vs extension data again, with the updated 0_ data, and with markers to facilitate finding the failure points.

```
[53]: # Plot the entire curve (up to 20mm and 2500N)
plotter(excel_dataframes, 'Extension (mm)', 'Load (N)', [10,20], [0,2500])

# show just the initial region (up to 0.25mm and 100N)
plotter(excel_dataframes, 'Extension (mm)', 'Load (N)', [0,1.5], [0,1000])
```





Define the row that corresponds to rupture in each test

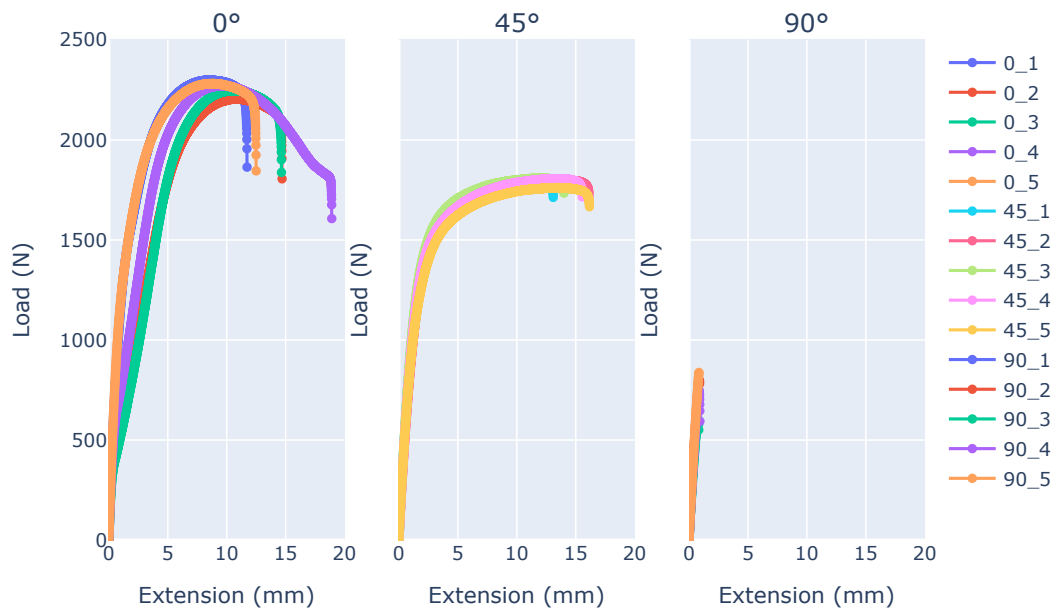
```
[54]: end_rows = {
    "0": [11.72503, 14.69816, 14.64815, 18.91472, 12.49997],
    "45": [13.08976, 16.13147, 13.99813, 15.5564, 16.16428],
    "90": [0.82312, 0.8918, 0.82312, 0.92308, 0.83967],
}

for group, cutoffs in end_rows.items():
    for i, max_extension in enumerate(end_rows[group], start=1):
        key = f"{group}_{i}"
        print(key)
        cutoff = (excel_dataframes[key]["Extension (mm)"] >= max_extension).
        ↪idxmax()
        excel_dataframes[key] = excel_dataframes[key].loc[:cutoff].copy()
```

```
0_1
0_2
0_3
0_4
0_5
45_1
45_2
```

45_3
45_4
45_5
90_1
90_2
90_3
90_4
90_5

```
[55]: # plot again after removing the post-failure data
      plotter(excel_dataframes, 'Extension (mm)', 'Load (N)', [0,20], [0,2500])
```



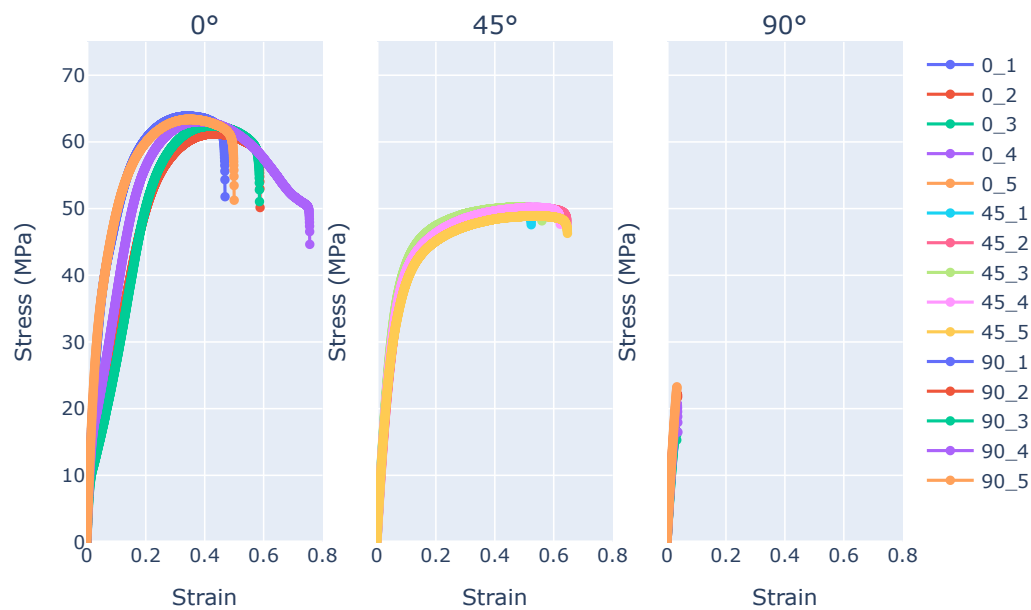
6 Part 5. Compute Stress and Strain

Use ASTM E8 FIG 1 to find the gauge length of the specimens. Assume cross sectional area is nearly 36mm^2 . (Actual specimen measurements found in properties.csv are nearly 6mm by 6 mm.)

```
[56]: area = 36          # mm2
gauge_length = 25.0    # mm

for df in excel_dataframes.values():
    df["Stress (MPa)"] = df["Load (N)"] / area
    df["Strain"] = df["Extension (mm)"] / gauge_length

[57]: # Plot stress vs strain data
plotter(excel_dataframes, 'Strain', 'Stress (MPa)', [0, 0.8], [0, 75])
```



Determine if the first 20 data points are nearly linear

```
[58]: for df_name, df in excel_dataframes.items():
    df_subset = df.iloc[:20]
    result = linregress(df_subset['Strain'], df_subset['Stress (MPa)'])
    print(f"{df_name}: R^2 = {result.rvalue**2:.5f}")
```

```
0_1: R^2 = 0.99969
0_2: R^2 = 0.99741
0_3: R^2 = 0.99729
0_4: R^2 = 0.99752
0_5: R^2 = 0.99989
45_1: R^2 = 0.99975
```

```

45_2: R^2 = 0.99989
45_3: R^2 = 0.99865
45_4: R^2 = 0.99959
45_5: R^2 = 0.99793
90_1: R^2 = 0.99965
90_2: R^2 = 0.99994
90_3: R^2 = 0.99828
90_4: R^2 = 0.99858
90_5: R^2 = 0.99972

```

7 Part 6. Compute Material Properties

The first 20 points are linear ($R^2=1$) so use these to compute Elastic modulus.

```

[59]: def tensile_properties(df, elastic_points=20):
    # Young's modulus (average the slope over the elastic range)
    elastic = df.iloc[1:elastic_points]
    E = np.mean(elastic["Stress (MPa)"] / elastic["Strain"])

    # 0.2% offset line
    offset_line = E * (df["Strain"] - 0.002)
    diff = df["Stress (MPa)"] - offset_line

    # Find first crossing
    crossing = np.where(np.diff(np.sign(diff)))[0]

    if len(crossing) > 0:
        idx = crossing[0]
        yield_stress = df.iloc[idx]["Stress (MPa)"]
        yield_strain = df.iloc[idx]["Strain"]
    else:
        yield_stress = np.nan
        yield_strain = np.nan

    return {
        "Young's Modulus (MPa)": E,
        "Yield Stress (MPa)": yield_stress,
        "Yield Strain": yield_strain,
        "Ultimate Stress (MPa)": df["Stress (MPa)"].max(),
        "Rupture Stress (MPa)": df["Stress (MPa)"].iloc[-1],
        "Rupture Strain": df["Strain"].iloc[-1],
        "Modulus of Toughness (MJ/m³)": trapezoid(df["Stress (MPa)"], df["Strain"]),
    }

```

```

[60]: results = {}

```

```

for name, df in excel_dataframes.items():
    results[name] = tensile_properties(df, elastic_points=20)

results_df = pd.DataFrame(results).T
# Show 4 sig figs
pd.set_option("display.float_format", lambda x: "{:.4g}".format(x))
display(results_df)

```

	Young's Modulus (MPa)	Yield Stress (MPa)	Yield Strain \
0_1	1126	20.32	0.02
0_2	1193	9.484	0.009926
0_3	1095	7.74	0.008926
0_4	1264	10.53	0.01026
0_5	1210	20.19	0.01867
45_1	947.2	10.08	0.01259
45_2	846.6	10.39	0.01426
45_3	929.9	11.78	0.01459
45_4	901.8	11.1	0.01426
45_5	971.7	8.684	0.01093
90_1	874.4	9.281	0.01259
90_2	829	13.5	0.01826
90_3	808.9	8.89	0.01293
90_4	864.4	10.32	0.01393
90_5	843.4	15.43	0.02026

	Ultimate Stress (MPa)	Rupture Stress (MPa)	Rupture Strain \
0_1	63.87	51.78	0.469
0_2	61.27	50.15	0.5879
0_3	62.4	51.05	0.5859
0_4	63.09	44.64	0.7566
0_5	63.39	51.26	0.5
45_1	49.69	47.58	0.5236
45_2	50.04	47.15	0.6453
45_3	50.24	48.16	0.5599
45_4	50.17	47.66	0.6223
45_5	48.94	46.29	0.6466
90_1	20.29	19.3	0.03292
90_2	22.17	21.75	0.03592
90_3	16.69	15.29	0.03292
90_4	20.73	16.48	0.03692
90_5	23.28	23.28	0.03359

	Modulus of Toughness (MJ/m ³)
0_1	25.77
0_2	28.7
0_3	28.67
0_4	39.91
0_5	27.6

45_1	23.08
45_2	28.75
45_3	25.11
45_4	27.86
45_5	28.26
90_1	0.3743
90_2	0.4567
90_3	0.3362
90_4	0.4582
90_5	0.4226

8 Part 7. Average Material Properties

```
[61]: averages = results_df.groupby(results_df.index.str.split('_').str[0]).mean()
display(averages)
```

	Young's Modulus (MPa)	Yield Stress (MPa)	Yield Strain \
0	1177	13.65	0.01356
45	919.4	10.4	0.01332
90	844	11.49	0.01559

	Ultimate Stress (MPa)	Rupture Stress (MPa)	Rupture Strain \
0	62.8	49.77	0.5799
45	49.82	47.37	0.5995
90	20.63	19.22	0.03446

	Modulus of Toughness (MJ/m ³)
0	30.13
45	26.61
90	0.4096

9 Part 8 Material Characterization

The 0° and 45° test data exhibit a large plastic deformation before failure, these are ductile. The 90° test data shows a failure at very low strain, while the material is still nearly linear, this is brittle. For this 3D printed material, the orientation dictates the behavior. If the manufactured part were treated as a “material” it would be considered anisotropic.