PHYTON DRIVING LICENSE

Exam project

Supervisors of the course:

Prof. Raos, Guido
Prof. Miglio, Edie
Prof. Bruschi, Francesco

Students ID:

Piccagli, Federico [10801916]
Leonardis, Giacomo [10575811]

Academic year: 2022 - 2023

School of Industrial and Information Engineering
PhD XXXVIIIth cycle – Materials Engineering

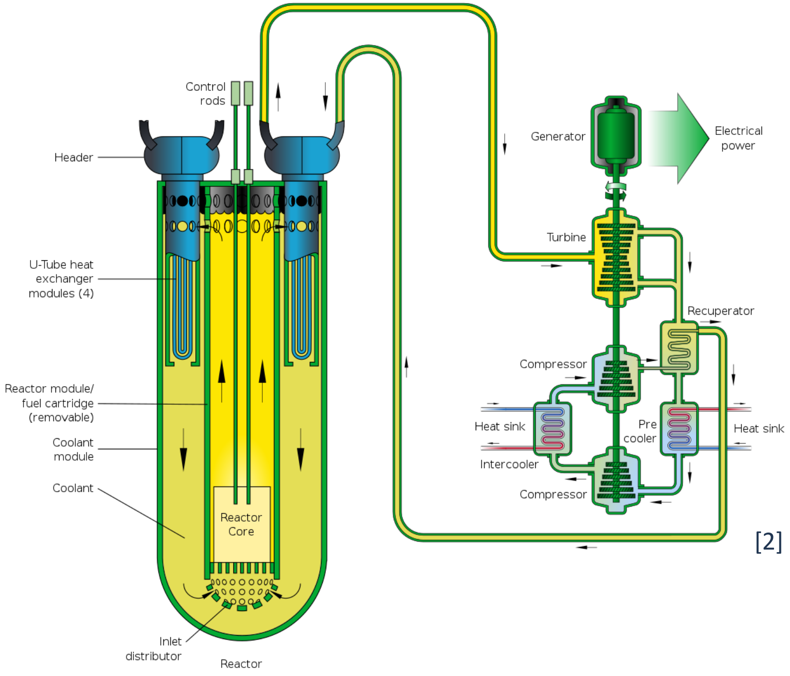
POLITECNICO MILANO 1863 X_nano invisible matters

IVth generation nuclear reactors

Next generation nuclear reactors | Materials challenges | Troubleshooting solutions | Materials requirements | Aim of the current PhD project

Fast Breeder Reactor (FBR) concepts

Lead-cooled fast reactor (LFR) technology



[2]

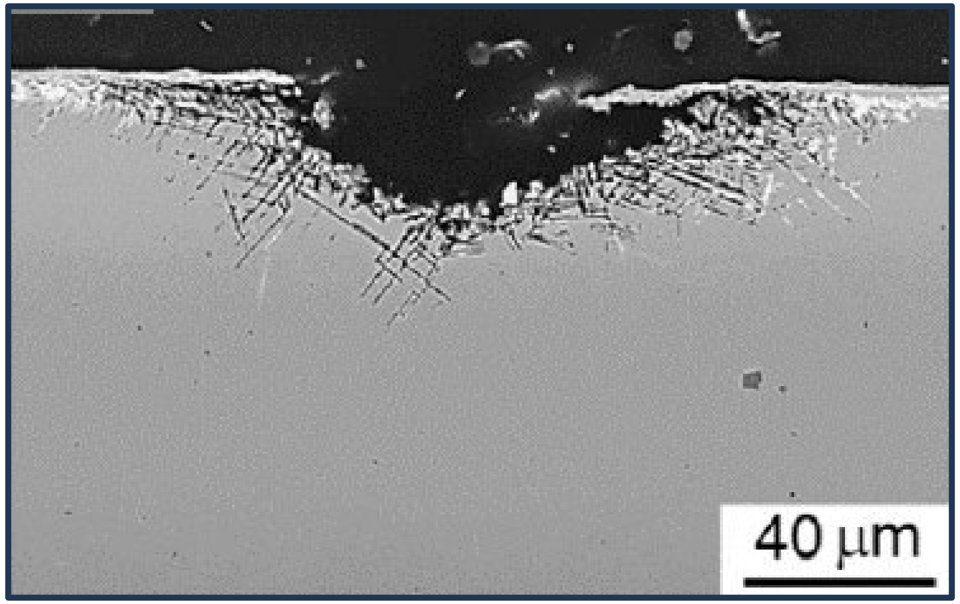
[1] *M. Vanazzi, PhD Thesis, Politecnico di Milano, 2019.*
[2] *https://en.wikipedia.org/wiki/Lead-cooled_fast_reactor.*

IVth generation nuclear reactors

Next generation nuclear reactors → **Materials challenges** → Troubleshooting solutions → Materials requirements → Aim of the current PhD project

Heavy Liquid Metal (HLM) dissolution

Integrity of reactor's in-core steel parts



[1]

[1] *M. Vanazzi, PhD Thesis, Politecnico di Milano, 2019.*

POLITECNICO MILANO 1863

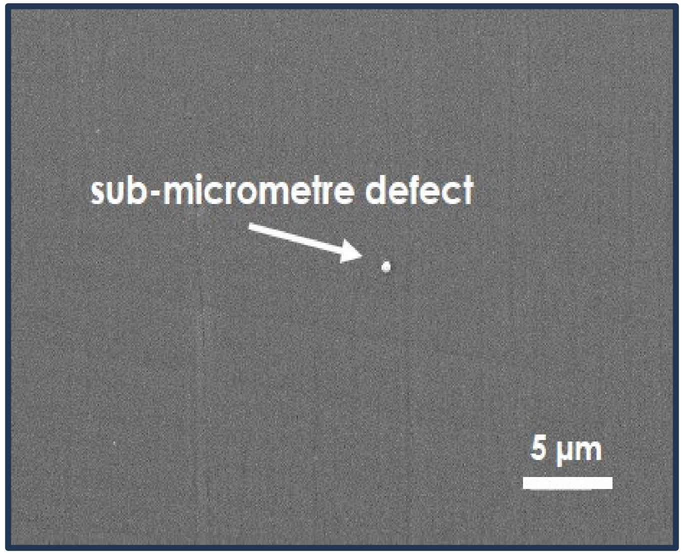
X_nano invisible matters

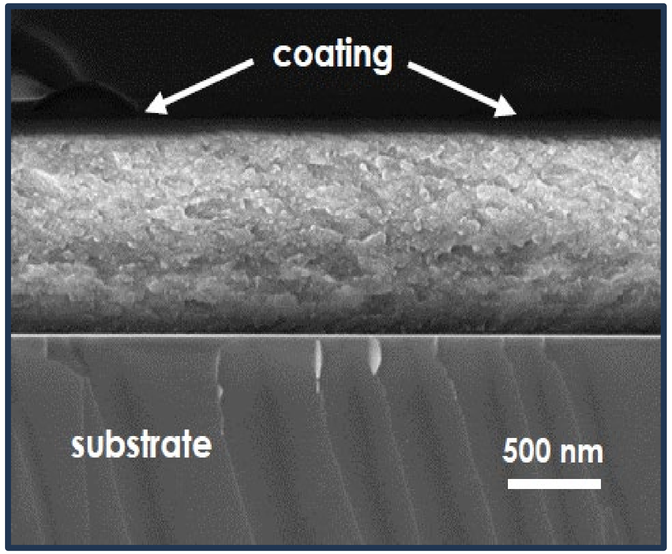
IVth generation nuclear reactors

Next generation nuclear reactors → Materials challenges → **Troubleshooting solutions** → Materials requirements → Aim of the current PhD project

Ceramic protective barriers

PLD-grown a-Al_2O_3 coatings





[1]

[1] M. Vanazzi, PhD Thesis, Politecnico di Milano, 2019.

POLITECNICO MILANO 1863

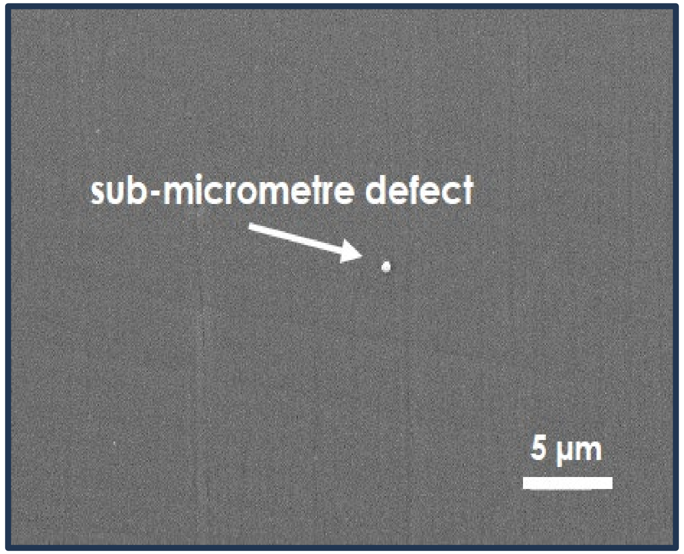
IV^th generation nuclear reactors

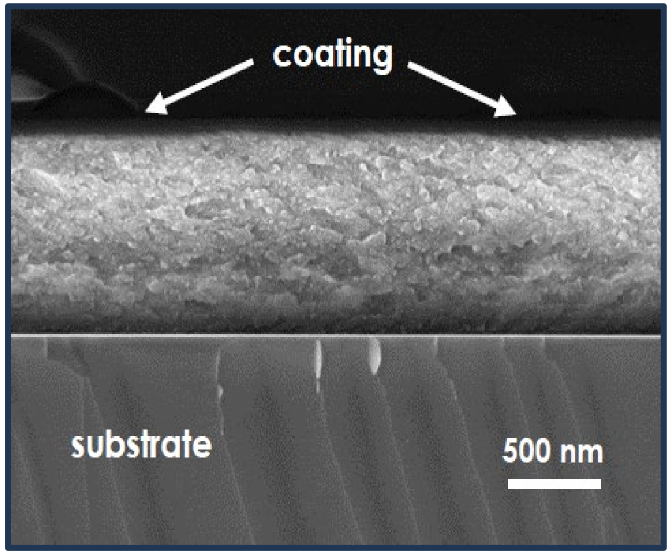
Next generation nuclear reactors → Materials challenges → Troubleshooting solutions → **Materials requirements** → Aim of the current PhD project

Preservation of the film's integrity

- *Corrosion-damage exposure*
- *Radiation-damage exposure*
- *Mechanical deformation*

Focus of the project hereby presented





[1]

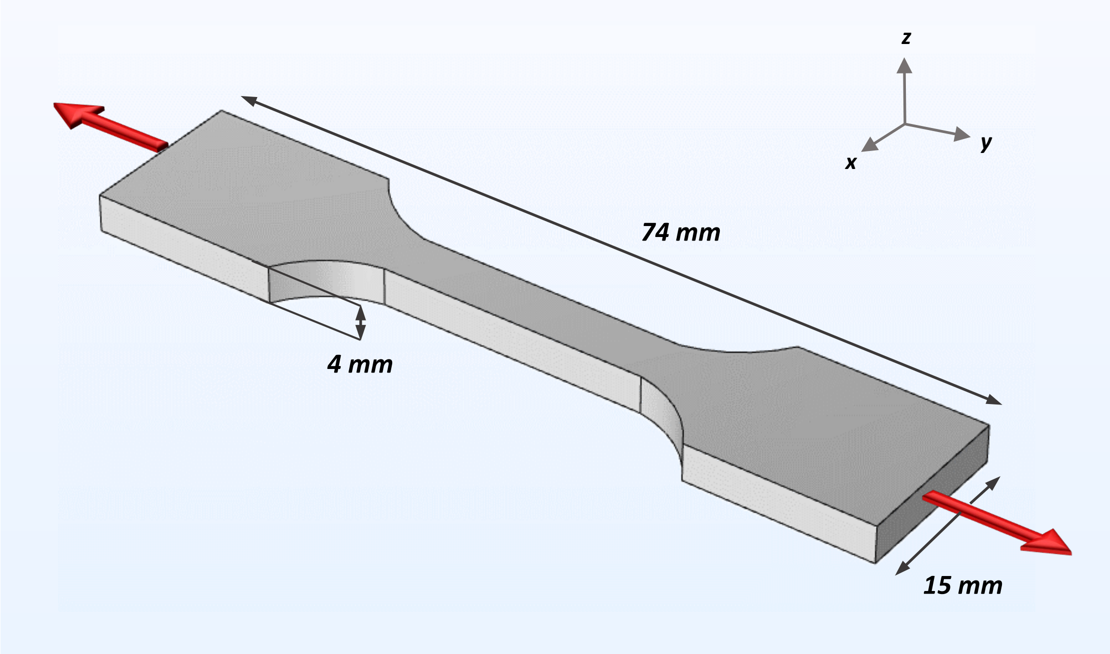
[1] *M. Vanazzi, PhD Thesis, Politecnico di Milano, 2019.*

IVth generation nuclear reactors

Next generation nuclear reactors → Materials challenges → Troubleshooting solutions → Materials requirements → **Aim of the current PhD project**

TENSILE TESTS experimental campaign

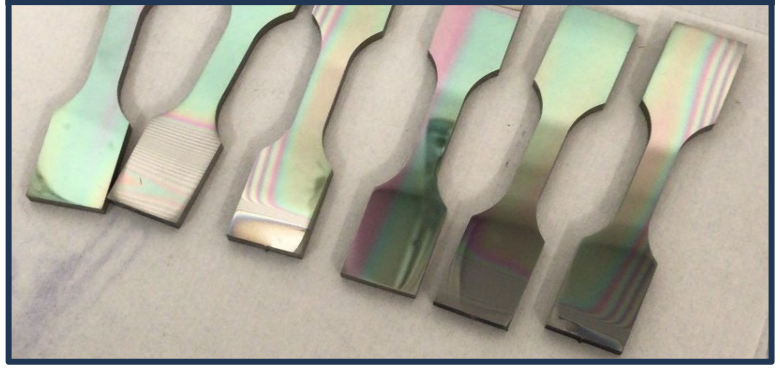
- **Planar** dog-bone **geometry**
- Externally **imposed strain/stress**
- Evaluate the **coating's response** upon mechanical deformations



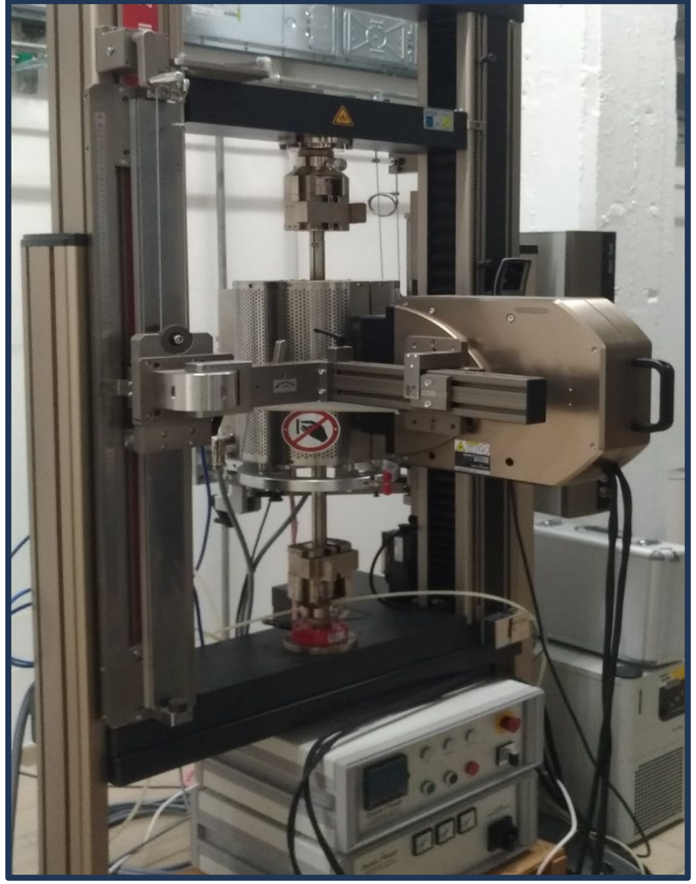
Experimental analysis

1. **Generation** of **dog-bone substrates** from a cold-rolled AISI316 metal sheet
2. **Pre-deposition treatments** (grinding, polishing, ultrasonic cleaning, ion gun cleaning)

3. Pulsed Laser Deposition of a-Al_2O_3 coatings with different thickness



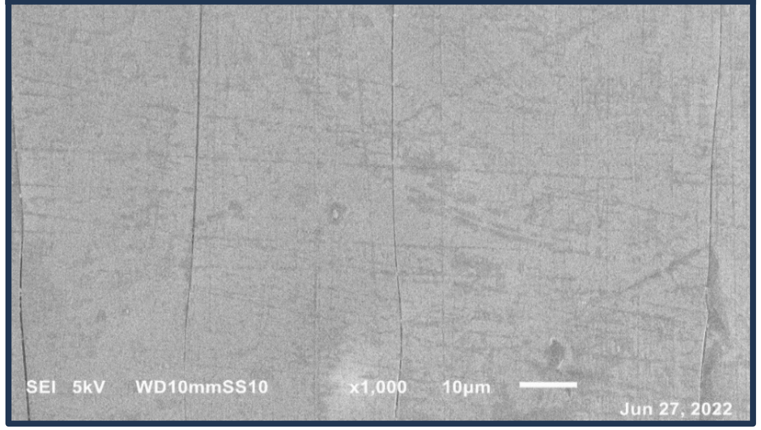
4. Mechanical tests with diverse tensile loads



5. Ex-situ characterization and analysis

Retrive the conditions (stress or strain) upon which the coating failed *i.e.*, cracked or delaminated

SEM image of a cracked coating



How Phyton is useful for?

Raw data as resulting from each tensile test

Stress-strain characteristics

Linear elastic approximation

Study of the ***fracture mechanics*** of the ***coating***

Incremental modulus of the composite

Study of the ***mechanics*** of the whole ***composite***

General overview of the assignment

Sequence of the operations leading to the expected result:

- Creation of a **test script** for a random case study
- **Adjustment** of the previous script to all case studies
- Generation of **re-elaborated scripts**

Focus on:

1. **DATABASE ANALYSIS**
2. **MANIPULATION OF RAW DATA**
3. **PLOTS AND DIAGRAMS**

Phyton libraries adopted for the purpose:

MatPlotLib	**NumPy/SciPy**	**Scikit-Learn/Statistics**	**Pandas**
Graphics	*Mathematics*	*Data analysis*	*Data manipulation*

Test script

- Definition of useful **functions**

```
def engineering_stress(load):
    A0 = 24
    return load*1000/A0
```

```
def engineering_strain(extension):
    L0 = 15
    return extension/L0
```

```
def median_value(array):
    sum_val = 0
    for i in range (0, array.shape[0]):
        sum_val += array[i]
    return sum_val/array.shape[0]
```

```
def r2regression(valori_reali, valori_attesi):
    RSS = float(0)
    TSS = float(0)
    media = median_value(valori_reali)
    for i in range (0, valori_reali.shape[0]):
        RSS += (valori_reali[i] - valori_attesi[i])**2
        TSS += (valori_reali[i] - media)**2
    return round(float(1-RSS/TSS), 4)
```

```
def regression_reliability(r2):
    if(r2>=0.95):
        return 'VERY GOOD'
    elif(r2<0.95 and r2>=0.90):
        return 'ADMISSIBLE'
    else:
        return 'NOT GOOD'
```

R^2 analytical definition: [3]

$$
\left\{
\begin{aligned}
R^2 &= 1 - \frac{RSS}{TSS} \\
RSS &= \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \\
TSS &= \sum_{i=1}^{n} (y_i - \bar{y})^2
\end{aligned}
\right.
$$

Test script

- Definition of useful **functions**
- **Raw data import** from an excel file
- Creation of a **DataFrame** containing the data to be analyzed

```
df = pd.DataFrame(pd.read_excel(r'C:\Users\Federico\Desktop\PhD\EXAMS\PHYTON DRIVING LICENSE\PROJECT_EXAM\RTdegC(2,0).xlsx',
                                sheet_name='8.3kN', usecols=[0,1,2,3]))
df = df.rename(columns={'time\n[s]':'Time (s)', 'crosshead\n[mm]':'Crosshead (mm)', 'extensometer\n[mm]':'Extensometer (mm)',
                        'load\n[kN]':'Load (kN)'})
df['Engineering strain (abs.)'] =  engineering_strain( df['Extensometer (mm)'] )
df['Engineering stress (MPa)'] =  engineering_stress( df['Load (kN)'] )
del df["Crosshead (mm)"] # crosshead data are useless and can be removed from the DataFrame
df.style.set_properties(**{'text-align': 'center'}) #centering the text
display(df)
```

	Time (s)	Extensometer (mm)	Load (kN)	Engineering strain (abs.)	Engineering stress (MPa)
0	0.0000	0.000000e+00	0.364120	0.000000e+00	15.171654
1	0.0200	-4.717588e-07	0.364118	-3.145059e-08	15.171579
2	0.0400	-9.435175e-07	0.364116	-6.290117e-08	15.171508
3	0.0600	-1.415276e-06	0.364114	-9.435173e-08	15.171433
4	0.0800	-1.887035e-06	0.364113	-1.258023e-07	15.171363
...	...	...	...	...	...
1954	39.0800	2.212456e-01	7.747406	1.474971e-02	322.808583
1955	39.1000	2.212485e-01	7.747242	1.474990e-02	322.801750
1956	39.1200	2.212513e-01	7.747077	1.475009e-02	322.794875
1957	39.1400	2.212542e-01	7.746913	1.475028e-02	322.788042
1958	39.1402	2.212542e-01	7.746912	1.475028e-02	322.788000

Raw data ***Processed data***

Test script

- Definition of useful **functions**
- **Raw data import** from an excel file
- Creation of a **DataFrame** containing the data to be analyzed
- **Linear interpolation**

```
X = df['Engineering strain (abs.)']
Y = df['Engineering stress (MPa)']
interpolation_function = interp1d(X, Y, kind = 'linear')
interpolation_points = np.linspace(0, df.iloc[-1]['Engineering strain (abs.)'], Y.shape[0])
interpolated_value = interpolation_function(interpolation_points)
```

The entire strain and stress columns of the DataFrame are collected into an array-like parameter (X and Y, respectively)

*The **"interp1d" class** generates the **interpolation function***

*The "**linspace**" **class** creates the **interpolation points** by specifying evenly spaced numbers over a specified interval (the entire strain column of the DataFrame)*

*The **interpolated value** is obtained by computing the interpolation function all over the prescribed interpolation points*

Test script

- Definition of useful **functions**
- **Raw data import** from an excel file
- Creation of a **DataFrame** containing the data to be analyzed
- **Linear interpolation**
- **Linear regression**

```
X1 = X[:1100].values.reshape(-1,1)
Y1 = Y[:1100].values.reshape(-1,1)
regressor = LinearRegression(fit_intercept = False).fit(X1, Y1)
# print(regressor.intercept_) --> to verify that the straight line effectively passes through the origin
y_estimated = regressor.predict(X1)
incremental_modulus = int(regressor.coef_/1000)
```

The entire strain and stress columns of the DataFrame are collected into a reshaped matrix-like parameter (X1 and Y1 respectively)

*The "**LinearRegression**" **class** fits the x- and y- data (X1 and Y1, respectively) generating a straight line passing through the origin (specified by the "fit_intercept" parameter)*

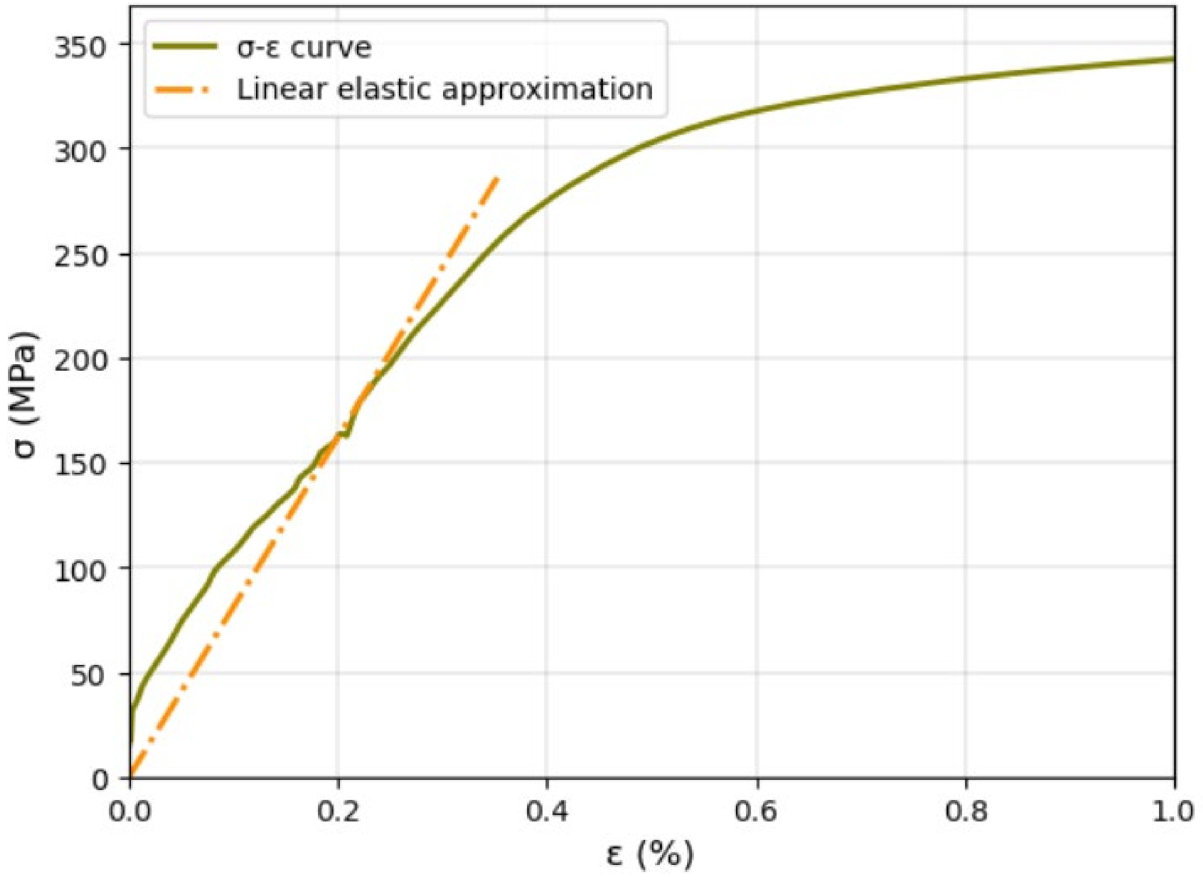
*The **estimated y-value** as resulting from the linear regression is obtained by way of the "**predict**" **method**, applied to all the x-values (X1, in this case)*

*The **incremental modulus** is the slope of the linear regression straight line passing through the origin (divided by 1000 to have the result in GPa)*

Test script

*"**pyplot**" **class** is used to display the stress-strain curve and the linear elastic approximation*

- Definition of useful **functions**
- **Raw data import** from an excel file
- Creation of a **DataFrame** containing the data to be analyzed
- **Linear interpolation**
- **Linear regression**
- **Graphical analysis** of the results



Test script

- Definition of useful **functions**
- **Raw data import** from an excel file
- Creation of a **DataFrame** containing the data to be analyzed
- **Linear interpolation**
- **Linear regression**
- **Graphical analysis** of the results
- Determination of the **numerical results**

```
--> The R² value of the linear regression computed analytically is: ~ 0.9111
--> The R² value of the linear regression computed with r2_score is: ~ 0.9111
--> The regression reliability is: ADMISSIBLE
--> The elastic modulus in the linear elastic region is: ~ 81 GPa
```

The R^2 value of the linear regression has been computed analytically by calling the "r2regression" function and with a specific class belonging to the scikit-learn library

A function determines the goodness of the linear elastic approximation performed so far

*The **incremental** (elastic) **modulus** is returned as an integer, whose magnitude is in GPa*

*An additional library ("**colorama**") is used to display colored text*

Analysis_RTdegC(2,0)

→ **Adjustment of the test script** to analyse **2 µm–thick coatings** tensile tested at 25°C

- Definition of useful **functions**
- **Raw data import** from an excel file
- Creation of a **DataFrame** containing the data to be analyzed

```python
# create the general dictionary
data = pd.read_excel(r'C:\Users\Federico\Desktop\PhD\EXAMS\PHYTON DRIVING LICENSE\PROJECT_EXAM\RTdegC(2,0).xlsx',
                     sheet_name=[1,2,3,4], usecols=[0,1,2,3])

# create the 1st DataFrame from the general dictionary
df1 = pd.DataFrame.from_dict(data[1], orient='columns')
df1 = df1.rename(columns={'time\n[s]':'Time (s)', 'crosshead\n[mm]':'Crosshead (mm)',
                          'extensometer\n[mm]':'Extensometer (mm)','load\n[kN]':'Load (kN)'})
df1['Engineering strain (abs.)'] =  engineering_strain( df1['Extensometer (mm)'] )
df1['Engineering stress (MPa)'] =  engineering_stress( df1['Load (kN)'] )
del df1["Crosshead (mm)"]
display(df1)
```

If multiple sheets are imported contemporarily, the Pandas library creates a general ***dictionary***

Each sheet corresponds to a diverse tensile load (experimental condition)

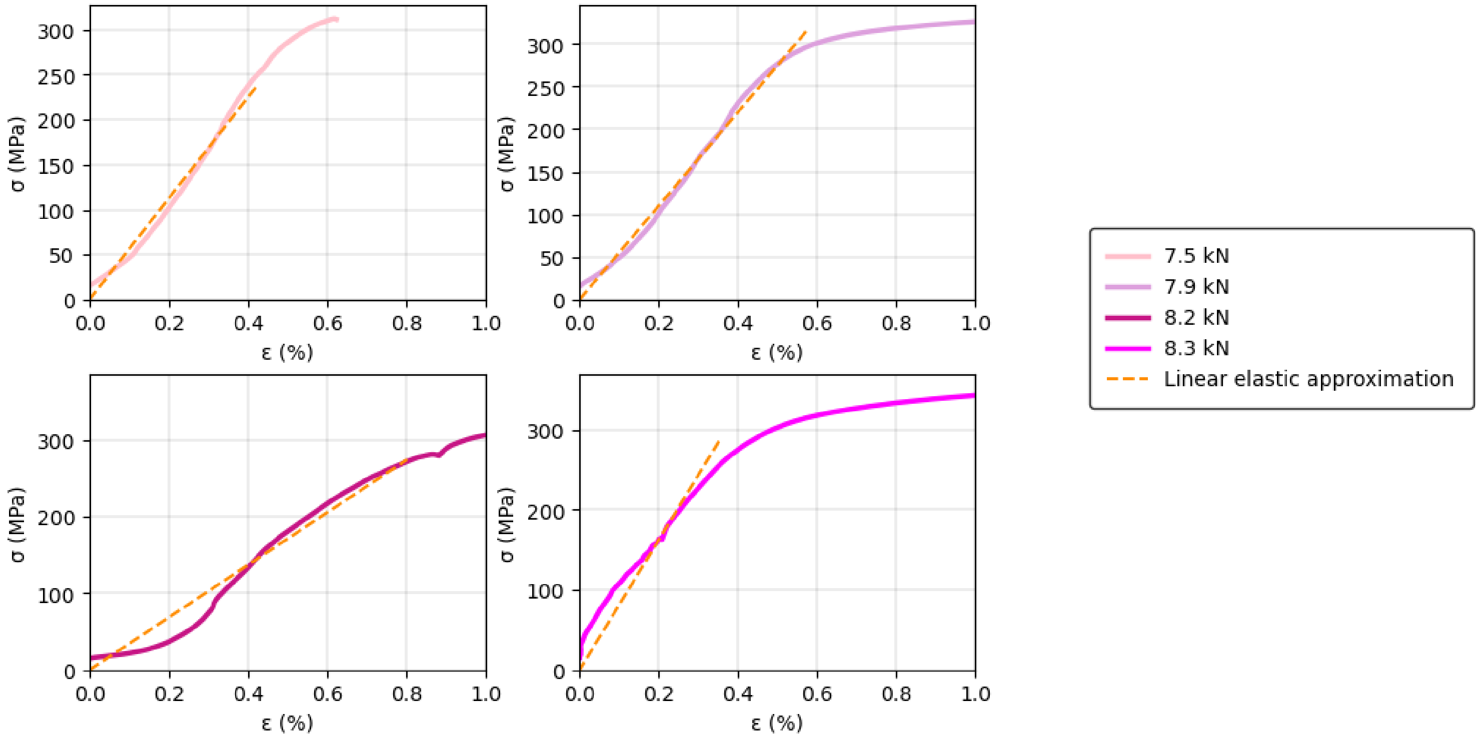
Every DataFrame must be extracted from the dictionary singularly

Analysis_RTdegC(2,0)

Adjustment of the test script to analyse **2 μm–thick coatings** tensile tested at 25°C

- Definition of useful **functions**
- **Raw data import** from an excel file
- Creation of a **DataFrame** containing the data to be analyzed
- **Linear interpolation**
- **Linear regression**
- **Graphical analysis** of the results

*"**pyplot**" **class** (**subplot method**) is used to display the stress-strain curve and the linear elastic approximation for every tensile load applied by the tensile machinery*



Analysis_RTdegC(2,0)

Adjustment of the test script to analyse **2 µm–thick coatings** tensile tested at 25°C

- Definition of useful **functions**
- **Raw data import** from an excel file
- Creation of a **DataFrame** containing the data to be analyzed
- **Linear interpolation**
- **Linear regression**
- **Graphical analysis** of the results
- Determination of the **numerical results**

```
# generation of 4 different lists of results
pulling_force = ['7.5', '7.9', '8.2', '8.3']
r2_results = [r2_score(Y1_1, y_estimated1), r2_score(Y2_2, y_estimated2), r2_score(Y3_3, y_estimated3),
              r2_score(Y4_4, y_estimated4)]
r2_reliability = [regression_reliability(r2_score(Y1_1, y_estimated1)),
                  regression_reliability(r2_score(Y2_2, y_estimated2)),
                  regression_reliability(r2_score(Y3_3, y_estimated3)),
                  regression_reliability(r2_score(Y4_4, y_estimated4))]
incremental_moduli = [incremental_modulus1, incremental_modulus2, incremental_modulus3, incremental_modulus4]

# creation of the corresponding DataFrame (N.B: each list is added to the DataFrame by way of the "zip" function)
df_results = pd.DataFrame(list(zip(pulling_force, r2_results, r2_reliability, incremental_moduli)),
                          columns =['Load (kN)', 'R\u00b2 value', 'Linear regression reliability',
                                    'Approximate incremental modulus (GPa)'])

# visualization of the final results
display(df_results.style.set_properties(**{'text-align': 'center'}))
```

Numerical results are saved into appropriate lists (arrays)

A DataFrame collecting the results is then generated by adding each list one next the other

Analysis_RTdegC(2,0)

Adjustment of the test script to analyse **2 µm–thick coatings** tensile tested at 25°C

- Definition of useful **functions**
- **Raw data import** from an excel file
- Creation of a **DataFrame** containing the data to be analyzed
- **Linear interpolation**
- **Linear regression**
- **Graphical analysis** of the results
- Determination of the **numerical results**

```
# generation of 4 different lists of results
pulling_force = ['7.5', '7.9', '8.2', '8.3']
r2_results = [r2_score(Y1_1, y_estimated1), r2_score(Y2_2, y_estimated2), r2_score(Y3_3, y_estimated3),
              r2_score(Y4_4, y_estimated4)]
r2_reliability = [regression_reliability(r2_score(Y1_1, y_estimated1)),
                  regression_reliability(r2_score(Y2_2, y_estimated2)),
                  regression_reliability(r2_score(Y3_3, y_estimated3)),
                  regression_reliability(r2_score(Y4_4, y_estimated4))]
incremental_moduli = [incremental_modulus1, incremental_modulus2, incremental_modulus3, incremental_modulus4]

# creation of the corresponding DataFrame (N.B: each list is added to the DataFrame by way of the "zip" function)
df_results = pd.DataFrame(list(zip(pulling_force, r2_results, r2_reliability, incremental_moduli)),
                          columns =['Load (kN)', 'R\u00b2 value', 'Linear regression reliability',
                                    'Approximate incremental modulus (GPa)'])

# visualization of the final results
display(df_results.style.set_properties(**{'text-align': 'center'}))
```

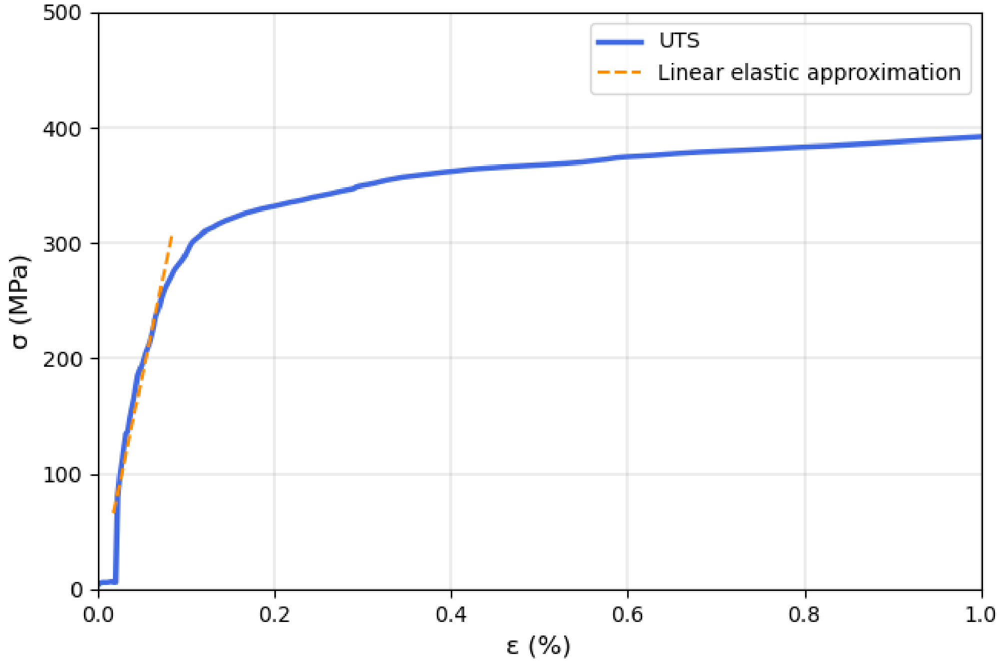
	Load (kN)	R^2 value	Linear regression reliability	Approximate incremental modulus (GPa)
0	7.5	0.982522	VERY GOOD	56
1	7.9	0.991223	VERY GOOD	54
2	8.2	0.963955	VERY GOOD	34
3	8.3	0.911062	ADMISSIBLE	81

Analysis_RTdegC(3,0)

→ **Adjustment of the test script** to analyse **3 µm–thick coatings** tensile tested at 25°C

- Definition of useful **functions**
- **Raw data import** from an excel file
- Creation of a **DataFrame** containing the data to be analyzed
- **Linear interpolation**
- **Linear regression**
- **Graphical analysis** of the results
- Determination of the **numerical results**

"pyplot" class is used to display the stress-strain curve and the linear elastic approximation



	Load (kN)	R^2 value	Linear regression reliability	Approximate incremental modulus (GPa)
0	UTS	0.959396	VERY GOOD	365

Analysis_RTdegC(0,25)

Adjustment of the test script to analyse **250 nm–thick coatings** tensile tested at 25°C

- Definition of useful **functions**
- **Raw data import** from an excel file
- Creation of a **DataFrame** containing the data to be analyzed

```
# create the general dictionary
data = pd.read_excel(r'C:\Users\Federico\Desktop\PhD\EXAMS\PHYTON DRIVING LICENSE\PROJECT_EXAM\RTdegC(0,25).xlsx',
                     sheet_name=[1,2], usecols=[0,1,2,3])

# create the 1st DataFrame from the general dictionary
df1 = pd.DataFrame.from_dict(data[1], orient='columns')
df1 = df1.rename(columns={'time\n[s]':'Time (s)', 'crosshead\n[mm]':'Crosshead (mm)',
                          'extensometer\n[mm]':'Extensometer (mm)','load\n[kN]':'Load (kN)'})
df1['Engineering strain (abs.)'] =  engineering_strain( df1['Extensometer (mm)'] )
df1['Engineering stress (MPa)'] =  engineering_stress( df1['Load (kN)'] )
del df1["Crosshead (mm)"]
display(df1)
```

If multiple sheets are imported contemporarily, the Pandas library creates a general ***dictionary***

Each sheet corresponds to a diverse tensile load (experimental condition)

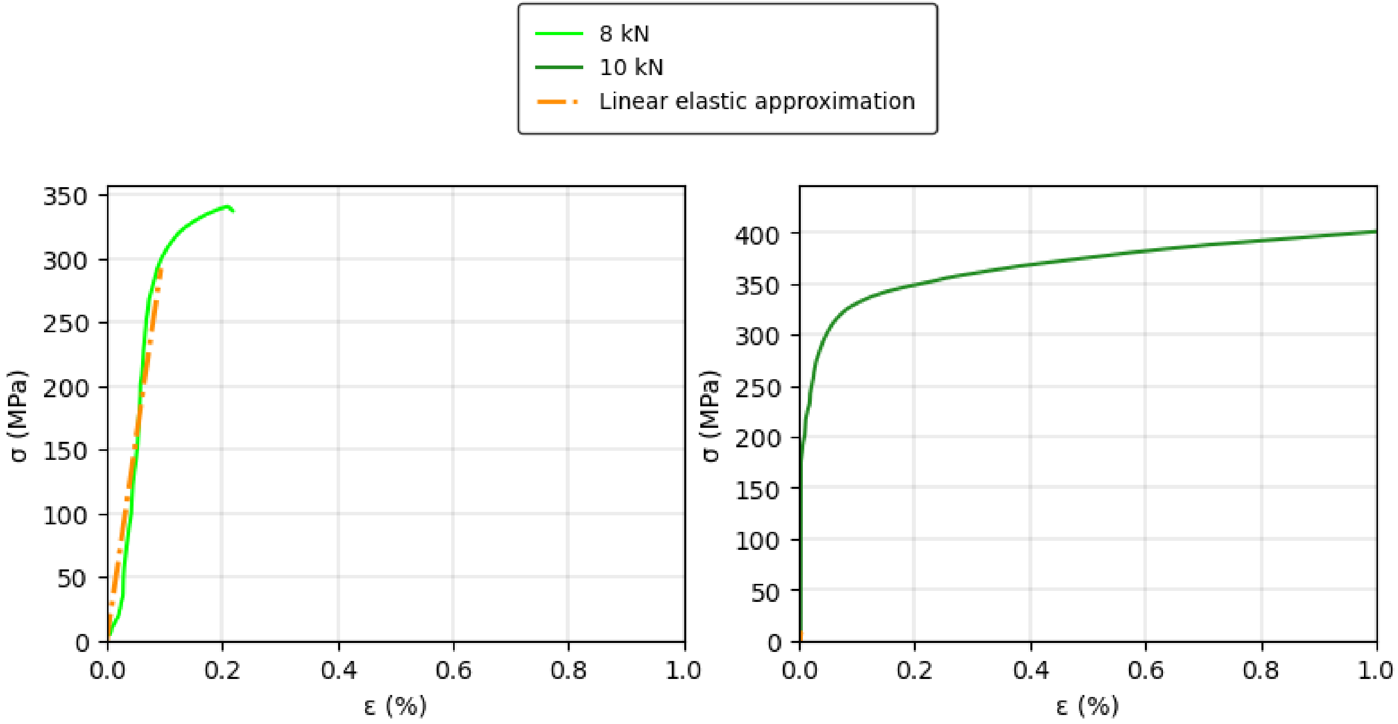
Every DataFrame must be extracted from the dictionary singularly

Analysis_RTdegC(0,25)

Adjustment of the test script to analyse **250 nm–thick coatings** tensile tested at 25°C

- Definition of useful **functions**
- **Raw data import** from an excel file
- Creation of a **DataFrame** containing the data to be analyzed
- **Linear interpolation**
- **Linear regression**
- **Graphical analysis** of the results

*“**pyplot**” **class** (**subplot method**) is used to display the stress-strain curve and the linear elastic approximation for every tensile load applied by the tensile machinery*



Analysis_RTdegC(0,25)

→ **Adjustment of the test script** to analyse **250 nm–thick coatings** tensile tested at 25°C

- Definition of useful **functions**
- **Raw data import** from an excel file
- Creation of a **DataFrame** containing the data to be analyzed
- **Linear interpolation**
- **Linear regression**
- **Graphical analysis** of the results
- Determination of the **numerical results**

```
# generation of 4 different lists of results
pulling_force = ['8', '10']
r2_results = [r2_score(Y1_1, y_estimated1), r2_score(Y2_2, y_estimated2), r2_score(Y3_3, y_estimated3),
              r2_score(Y4_4, y_estimated4)]
r2_reliability = [regression_reliability(r2_score(Y1_1, y_estimated1)),
                  regression_reliability(r2_score(Y2_2, y_estimated2)),
                  regression_reliability(r2_score(Y3_3, y_estimated3)),
                  regression_reliability(r2_score(Y4_4, y_estimated4))]
incremental_moduli = [incremental_modulus1, incremental_modulus2, incremental_modulus3, incremental_modulus4]

# creation of the corresponding DataFrame (N.B: each list is added to the DataFrame by way of the "zip" function)
df_results = pd.DataFrame(list(zip(pulling_force, r2_results, r2_reliability, incremental_moduli)),
                          columns =['Load (kN)', 'R\u00b2 value', 'Linear regression reliability',
                                    'Approximate incremental modulus (GPa)'])

# visualization of the final results
display(df_results.style.set_properties(**{'text-align': 'center'}))
```

Numerical results are saved into appropriate lists (arrays)

A DataFrame collecting the results is then generated by adding each list one next the other

Analysis_RTdegC(0,25)

Adjustment of the test script to analyse **250 nm–thick coatings** tensile tested at 25°C

- Definition of useful **functions**
- **Raw data import** from an excel file
- Creation of a **DataFrame** containing the data to be analyzed
- **Linear interpolation**
- **Linear regression**
- **Graphical analysis** of the results
- Determination of the **numerical results**

```
# generation of 4 different lists of results
pulling_force = ['8', '10']
r2_results = [r2_score(Y1_1, y_estimated1), r2_score(Y2_2, y_estimated2), r2_score(Y3_3, y_estimated3),
              r2_score(Y4_4, y_estimated4)]
r2_reliability = [regression_reliability(r2_score(Y1_1, y_estimated1)),
                  regression_reliability(r2_score(Y2_2, y_estimated2)),
                  regression_reliability(r2_score(Y3_3, y_estimated3)),
                  regression_reliability(r2_score(Y4_4, y_estimated4))]
incremental_moduli = [incremental_modulus1, incremental_modulus2, incremental_modulus3, incremental_modulus4]

# creation of the corresponding DataFrame (N.B: each list is added to the DataFrame by way of the "zip" function)
df_results = pd.DataFrame(list(zip(pulling_force, r2_results, r2_reliability, incremental_moduli)),
                          columns =['Load (kN)', 'R\u00b2 value', 'Linear regression reliability',
                                    'Approximate incremental modulus (GPa)'])

# visualization of the final results
display(df_results.style.set_properties(**{'text-align': 'center'}))
```

	Load (kN)	R² value	Linear regression reliability	Approximate incremental modulus (GPa)
0	8	0.932999	ADMISSIBLE	311
1	10	-138.525595	NOT GOOD	289

!!!

The linear regression is not always possible and may run into ***errors in fitting the linear elastic regime***

Analysis_RTdegC(0,5)

Adjustment of the test script to analyse **500 nm–thick coatings** tensile tested at 25°C

- Definition of useful **functions**
- **Raw data import** from an excel file
- Creation of a **DataFrame** containing the data to be analyzed

```
# create the general dictionary
data = pd.read_excel(r'C:\Users\Federico\Desktop\PhD\EXAMS\PHYTON DRIVING LICENSE\PROJECT_EXAM\RTdegC(0,50).xlsx',
                     sheet_name=[1,2,3,4], usecols=[0,1,2,3])

# create the 1st DataFrame from the general dictionary
df1 = pd.DataFrame.from_dict(data[1], orient='columns')
df1 = df1.rename(columns={'time\n[s]':'Time (s)', 'crosshead\n[mm]':'Crosshead (mm)',
                          'extensometer\n[mm]':'Extensometer (mm)','load\n[kN]':'Load (kN)'})
df1['Engineering strain (abs.)'] =  engineering_strain( df1['Extensometer (mm)'] )
df1['Engineering stress (MPa)'] =  engineering_stress( df1['Load (kN)'] )
del df1["Crosshead (mm)"]
display(df1)
```

*If multiple sheets are imported contemporarily, the Pandas library creates a general **dictionary***

Each sheet corresponds to a diverse tensile load (experimental condition)

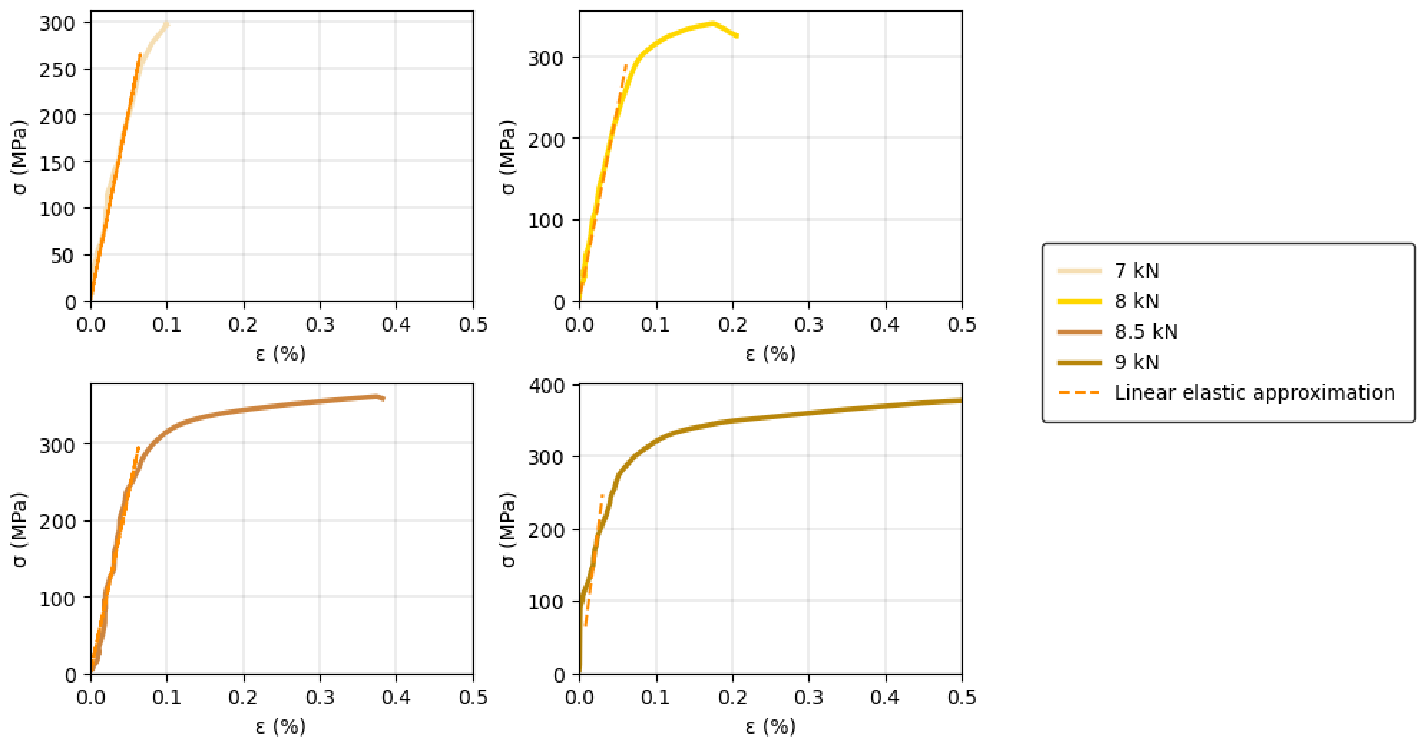
Every DataFrame must be extracted from the dictionary singularly

Analysis_RTdegC(0,5)

Adjustment of the test script to analyse **500 nm–thick coatings** tensile tested at 25°C

- Definition of useful **functions**
- **Raw data import** from an excel file
- Creation of a **DataFrame** containing the data to be analyzed
- **Linear interpolation**
- **Linear regression**
- **Graphical analysis** of the results

*"**pyplot**" **class** (**subplot method**) is used to display the stress-strain curve and the linear elastic approximation for every tensile load applied by the tensile machinery*



Analysis_RTdegC(0,5)

Adjustment of the test script to analyse **500 nm–thick coatings** tensile tested at 25°C

- Definition of useful **functions**
- **Raw data import** from an excel file
- Creation of a **DataFrame** containing the data to be analyzed
- **Linear interpolation**
- **Linear regression**
- **Graphical analysis** of the results
- Determination of the **numerical results**

```python
# generation of 4 different lists of results
pulling_force = ['7', '8', '8.5', '9']
r2_results = [r2_score(Y1_1, y_estimated1), r2_score(Y2_2, y_estimated2), r2_score(Y3_3, y_estimated3),
              r2_score(Y4_4, y_estimated4)]
r2_reliability = [regression_reliability(r2_score(Y1_1, y_estimated1)),
                  regression_reliability(r2_score(Y2_2, y_estimated2)),
                  regression_reliability(r2_score(Y3_3, y_estimated3)),
                  regression_reliability(r2_score(Y4_4, y_estimated4))]
incremental_moduli = [incremental_modulus1, incremental_modulus2, incremental_modulus3, incremental_modulus4]

# creation of the corresponding DataFrame (N.B: each list is added to the DataFrame by way of the "zip" function)
df_results = pd.DataFrame(list(zip(pulling_force, r2_results, r2_reliability, incremental_moduli)),
                          columns =['Load (kN)', 'R\u00b2 value', 'Linear regression reliability',
                                    'Approximate incremental modulus (GPa)'])

# visualization of the final results
display(df_results.style.set_properties(**{'text-align': 'center'}))
```

Numerical results are saved into appropriate lists (arrays)

A DataFrame collecting the results is then generated by adding each list one next the other

Analysis_RTdegC(0,5)

Adjustment of the test script to analyse **500 nm–thick coatings** tensile tested at 25°C

- Definition of useful **functions**
- **Raw data import** from an excel file
- Creation of a **DataFrame** containing the data to be analyzed
- **Linear interpolation**
- **Linear regression**
- **Graphical analysis** of the results
- Determination of the **numerical results**

```
# generation of 4 different lists of results
pulling_force = ['7', '8', '8.5', '9']
r2_results = [r2_score(Y1_1, y_estimated1), r2_score(Y2_2, y_estimated2), r2_score(Y3_3, y_estimated3),
              r2_score(Y4_4, y_estimated4)]
r2_reliability = [regression_reliability(r2_score(Y1_1, y_estimated1)),
                  regression_reliability(r2_score(Y2_2, y_estimated2)),
                  regression_reliability(r2_score(Y3_3, y_estimated3)),
                  regression_reliability(r2_score(Y4_4, y_estimated4))]
incremental_moduli = [incremental_modulus1, incremental_modulus2, incremental_modulus3, incremental_modulus4]

# creation of the corresponding DataFrame (N.B: each list is added to the DataFrame by way of the "zip" function)
df_results = pd.DataFrame(list(zip(pulling_force, r2_results, r2_reliability, incremental_moduli)),
                          columns =['Load (kN)', 'R\u00b2 value', 'Linear regression reliability',
                                    'Approximate incremental modulus (GPa)'])

# visualization of the final results
display(df_results.style.set_properties(**{'text-align': 'center'}))
```

	Load (kN)	R² value	Linear regression reliability	Approximate incremental modulus (GPa)
0	7	0.984558	VERY GOOD	403
1	8	0.983680	VERY GOOD	472
2	8.5	0.968531	VERY GOOD	466
3	9	0.369303	NOT GOOD	804

!!!

Conclusions of the project

- Results obtained with Phyton are in line with the MS Excel™ ones computed in a previous analysis
- Phyton is an **efficient** and **ready-to-use** programming language
- The broad range of libraries permits to cover every aspect of **data processing** and **graphical analysis**
- Phyton permits to manage huge quantity of data with negligible **computational times**
- A large community and a thorough online documentation can easily **support** programmers

THANK YOU FOR YOUR KIND ATTENTION

School of Industrial and Information Engineering
PhD XXXVIIIth cycle – Materials Engineering