Metody Komputerowe w Spalaniu
2025

Script for FEM analysis automation in ANSYS Mechanical

Krzysztof Niebylski 327623

May 24, 2025

Abstract

This project presents a scripting workflow to automate Finite Element Method (FEM) analysis in ANSYS Mechanical using IronPython and standard Python. The system simplifies repetitive analysis tasks by handling mesh generation, boundary condition assignment, and result extraction. A second Python script then reads the results and generates comparative visualizations. This solution reduces manual effort, improves repeatability, and allows for efficient post-processing.

Contents

1 Introduction **3**
1.1 Background 3
1.2 Motivation 3
1.3 Objectives 3

2 Requirements **3**
2.1 Software Tools 3
2.2 Hardware 3
2.3 Pre-Analysis Setup 3

3 Methodology **5**
3.1 Workflow Overview 5
3.2 Workflow Diagram 6
3.3 File Structure 7

4 Results **7**
4.1 Tabulated Results 7
4.2 Generated Graph 7

5 Conclusions **7**

6 References **8**

7 Appendices **9**

1 Introduction

1.1 Background

Finite Element Method (FEM) is widely used in structural and mechanical simulations. ANSYS Mechanical provides a powerful GUI-based platform for performing such analyses, but manual setup can be time-consuming and error-prone, especially in iterative workflows.

1.2 Motivation

Automating FEM simulations can significantly enhance productivity by reducing repetitive tasks and minimizing human errors. This project aims to build a scripting pipeline for conducting and analyzing multiple simulation scenarios with minimal manual input.

1.3 Objectives

This project is designed to allow quick analysis and comparison of different models under 3 different Static Structural load cases, based on maximum stress and deformation extracted from the results and visualized. The models undergo automatic meshing, application of boundary conditions and 3 progressive load cased: force, pressure and moment, which can easily be modified through code.

2 Requirements

2.1 Software Tools

- ANSYS Workbench (Version 2025 R1)
- IronPython 2.7 (embedded in ANSYS)
- Python 3.13 for post-processing
- `matplotlib` library

2.2 Hardware

Any workstation capable of running ANSYS simulations efficiently.

2.3 Pre-Analysis Setup

1) Import geometry manually into ANSYS Mechanical.

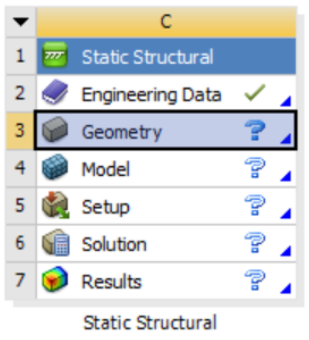


Figure 1: Importing geometry to ANSYS Mechanical

2) Create two named selections:

   - `fixed` — for the fixed constraint.
   - `load` — for applying the loads.

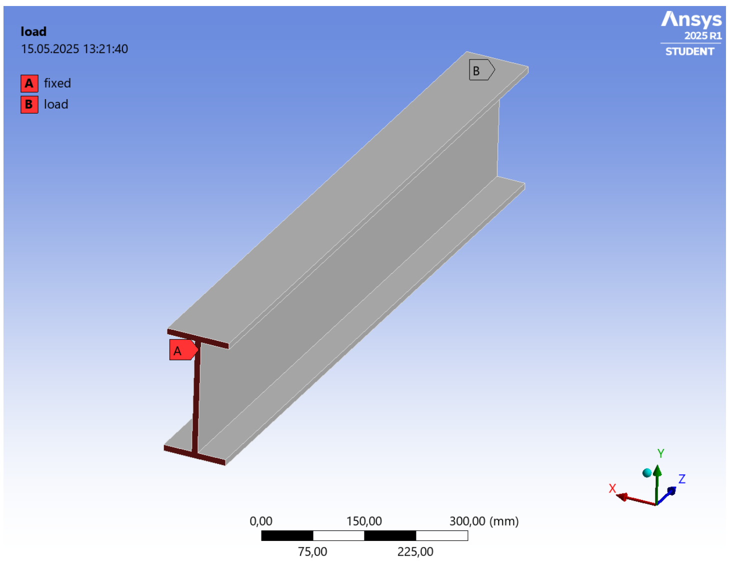


Figure 2: Created named selections

3) Check if loads match with defined coordinate system and modify if needed. The example analysis is set as: -10000 N force on Y axis, 100 MPa pressure on defined face and 1000Nm moment on Z axis to demonstrate the project potential.

4) Change paths in `automation.py` and `postprocess.py`. File paths are at the beginning of the scripts.

3 Methodology

3.1 Workflow Overview

The project consists of two scripts:

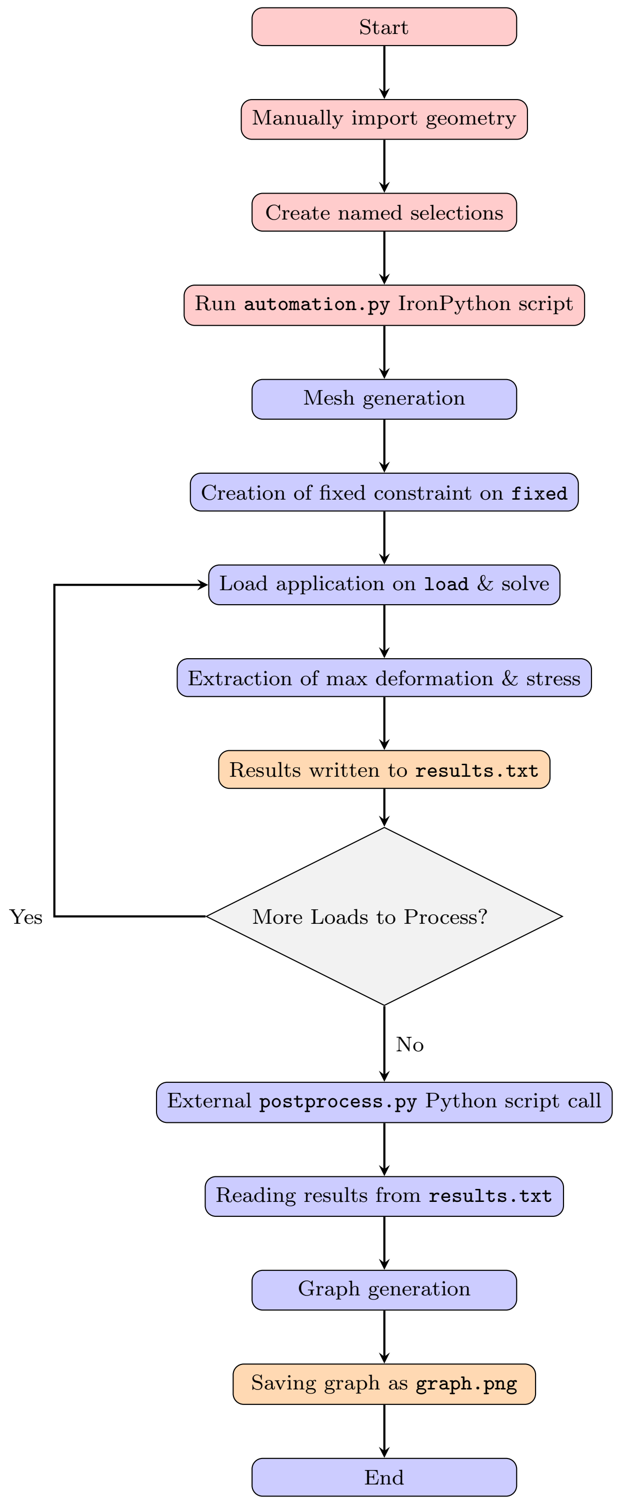
Script 1: IronPython (within ANSYS)

- Generate mesh
- Apply boundary conditions
- Define three load cases (force, pressure, moment) from pre-defined values in `automation.py`
- Solve simulations
- Extract maximum deformation and von Mises stress
- Write results to `results.txt` and save it in defined directory
- Call the second script

Script 2: Python 3.13

- Read `results.txt`.
- Generate and save a graph of results (`graph.png`).

3.2 Workflow Diagram


Start
Manually import geometry
Create named selections
Run automation.py IronPython script
Mesh generation
Creation of fixed constraint on fixed
Load application on load & solve
Extraction of max deformation & stress
Results written to results.txt
More Loads to Process?
Yes
No
External postprocess.py Python script call
Reading results from results.txt
Graph generation
Saving graph as graph.png
End

3.3 File Structure

```
/FEM_Automation_Project

 automation.py    # IronPython script for ANSYS
 postprocess.py      # Python 3 script for graphing
 results.txt          # Auto-generated results
 graph.png          # Auto-generated chart
```

4 Results

4.1 Tabulated Results

Load Case	Max stress [MPa]	Max deformation [mm]
Force Load	48.82	0.83
Pressure Load	119.48	0.50
Moment Load	172.64	5.47

Table 1: Example results extracted from simulations

4.2 Generated Graph

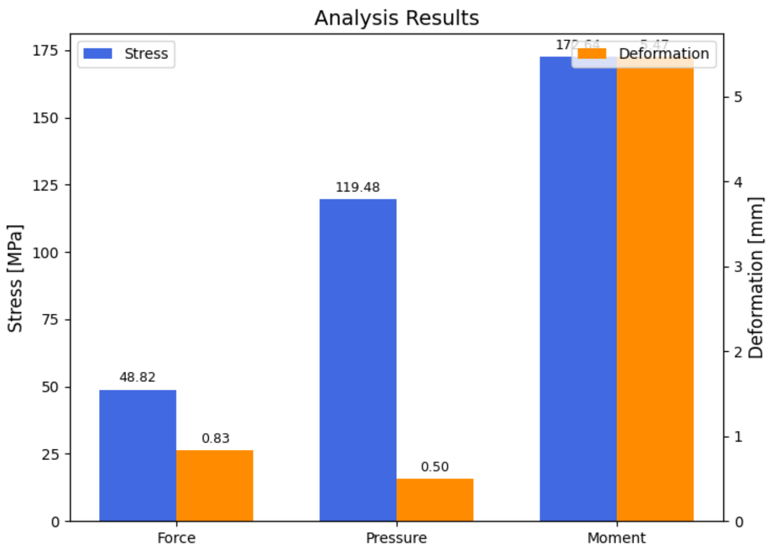


Figure 3: Bar chart showing max deformation and stress across load cases

5 Conclusions

This project demonstrated how to integrate IronPython and Python scripts to streamline FEM simulations in ANSYS. The automated process enables efficient batch simulations for simple moddels and result tracking with minimal user intervention.

6 References

- ANSYS Mechanical Scripting API Documentation.
- IronPython Official Documentation.
- Python `matplotlib` Documentation.
- ChatGPT

7 Appendices

Appendix A: automation.py

```
# -*- coding: utf-8 -*-

import subprocess

# input correct paths (output__file-results; script path-directory of
   second script; python_exe-location of python)
output_file_path = r'C:\Users\krzys\OneDrive\Pulpit\MKWS\results.txt'
python_exe = r'C:\Users\krzys\AppData\Local\Microsoft\WindowsApps\
   python.exe'
script_path = r'C:\Users\krzys\OneDrive\Pulpit\MKWS\postprocess.py'

#results file clear
with open(output_file_path, 'w') as file:
    file.write('')

analysis = Model.Analyses[0]
mesh = Model.Mesh
mesh.GenerateMesh()
solution = analysis.Solution

# adding fixed constraint
fixed_support = analysis.AddFixedSupport()
fixed_support.Location = ExtAPI.DataModel.GetObjectsByName("fixed")[0]

# list of loads
load_cases = [
    {"type": "force", "name": "load", "value": -10000},    # N
    {"type": "pressure", "name": "load", "value": -100},    # MPa
    {"type": "moment", "name": "load", "value": 1000}      # N m
]


# adding solutions
eq_stress = solution.AddEquivalentStress()
deformation = solution.AddTotalDeformation()

# main loop
for idx, case in enumerate(load_cases):

    # deleting previous load
    to_delete = []
    for obj in analysis.Children:
        typename = str(type(obj))
        if "Force" in typename or "Pressure" in typename or "Moment" in
            typename or "Result" in typename:
            to_delete.append(obj)
    for obj in to_delete:
        obj.Delete()

    # adding load
    load = None
    target = ExtAPI.DataModel.GetObjectsByName(case["name"])[0]
    if case["type"] == "force":
        load = analysis.AddForce()
        load.Location = target
        load.DefineBy = LoadDefineBy.Components
        load.YComponent.Output.SetDiscreteValue(0, Quantity(case["value
           "], "N"))

    elif case["type"] == "pressure":
        load = analysis.AddPressure()
        load.Location = target
        load.Magnitude.Output.SetDiscreteValue(0, Quantity(case["value"
           ], "MPa"))

    elif case["type"] == "moment":
        load = analysis.AddMoment()
        load.Location = target
        load.DefineBy = LoadDefineBy.Components
        load.ZComponent.Output.SetDiscreteValue(0, Quantity(case["value
           "], "N␣m"))

    # clear previous data and solve
    solution.ClearGeneratedData()
    solution.Solve()

    max_stress = float(eq_stress.Maximum.Value)
    max_deformation = float(deformation.Maximum.Value)

    # saving to file
    with open(output_file_path, 'a') as file:
        #file.write("Typ obciazenia: {}\n".format(case["type"]))
        file.write("{:.2f}\t".format(max_stress))
        file.write("{:.2f}\n".format(max_deformation))


# second script call
subprocess.Popen([python_exe, script_path])
```

Appendix B: postprocess.py

```
import matplotlib.pyplot as plt

# input correct directory of results file
input_file = r'C:\Users\krzys\OneDrive\Pulpit\MKWS\results.txt'

# reading data
case = ["Force", "Pressure", "Moment"]
stress = []
deformation = []

with open(input_file, 'r') as file:
    for linia in file:
        czesci = linia.strip().split('\t')
        if len(czesci) == 2:
            a, b = czesci
            stress.append(float(a))
            deformation.append(float(b))
```

```
print("x:", stress)
print("y:", deformation)

# graph settings
fig, ax1 = plt.subplots(figsize=(8, 6))

# position of bars
x_pos = range(len(stress))
bar_width = 0.35

# stress bars
bars1 = ax1.bar(x_pos, stress, bar_width, label="Stress", color='
   royalblue')
ax1.set_ylabel('Stress␣[MPa]', fontsize=12)
ax1.set_title('Analysis␣Results', fontsize=14)

# x axis labels
ax1.set_xticks([p + bar_width / 2 for p in x_pos])
ax1.set_xticklabels(case)

# y axis setup
ax2 = ax1.twinx()
bars2 = ax2.bar([p + bar_width for p in x_pos], deformation, bar_width,
    label="Deformation", color='darkorange')
ax2.set_ylabel('Deformation␣[mm]', fontsize=12)

# labels above bars (stress)
for bar in bars1:
    height = bar.get_height()
    ax1.annotate(f'{height:.2f}', xy=(bar.get_x() + bar.get_width() /
       2, height),
                 xytext=(0, 3), textcoords='offset␣points',
                 ha='center', va='bottom', fontsize=9)

# labels abve bars (deformation)
for bar in bars2:
    height = bar.get_height()
    ax2.annotate(f'{height:.2f}', xy=(bar.get_x() + bar.get_width() /
       2, height),
                 xytext=(0, 3), textcoords='offset␣points',
                 ha='center', va='bottom', fontsize=9)

# legend
ax1.legend(loc='upper␣left')
ax2.legend(loc='upper␣right')

# chart saving
chart_path = r'C:\Users\krzys\OneDrive\Pulpit\MKWS\graph.png'
plt.savefig(chart_path)

# graph plotting
plt.show()
```