Engineering Computation

A Brief Introduction to Computational Tools for Mechanics

Serhat Beyenir

3 September 2025

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).

Full text available at https://creativecommons.org/licenses/by-sa/4.0/.

Contents

List of Figures

List of Tables

Preface

A Brief Introduction to Computational Tools for Mechanics presents a collection of tutorials based on lecture notes from Applied Mechanics classes, designed to give learners clear and essential insights into key topics. No previous programming experience is required. Each tutorial guides you step by step through the concepts with hands-on examples and includes a problem set to reinforce understanding. You will learn to solve applied mechanics problems using computational tools, with an emphasis on practice and problem-solving rather than theory alone.

Chapter 1: Introduces the fundamentals of Python, including syntax, variables, data types, operations, control flow, and functions.

Chapter 2: Explains the International System of Units (SI), the globally accepted standard for measurement, and its role in ensuring consistency, clarity, and comparability in scientific and technical work.

Chapter 3: Covers advanced Python skills, including the use of virtual environments for project isolation, installing libraries such as Pint with pip, and developing a flexible unit conversion script.

Chapter 4: Demonstrates the application of Python to applied mechanics problems, using NumPy for computations, Matplotlib for visualization, and SciPy for numerical methods.

Dedication

To my students, whose inspiration and encouragement made this work possible.

Attributions

This book incorporates figures, data, and other resources from the following sources:

- Figures in the formulae, the centroids and second moments of area, are adapted from the Wikipedia pages on Centroids and Second moments of area.

Acknowledgements

I would like to thank the colleagues and friends who helped shape this project.

Special thanks to their valuable feedback and support.

Any errors or omissions are my own.

Part I

Beginning

Chapter 1

Python Tutorial

This tutorial introduces Python programming, covering basic concepts with examples to illustrate key points. We will start by using Python as a calculator, then explore variables, functions, and control flow.

1.1 Requirements

To follow this tutorial, you must have Python (version 3.10 or later) installed on your computer. Python is available for Windows, macOS, and Linux. Additionally, ensure you have a text editor or an Integrated Development Environment (IDE) to write Python code. We recommend Positron, a user-friendly IDE with a built-in terminal for running Python scripts, though other editors like VS Code or PyCharm are also suitable.

1.2 Basic Syntax

Python uses indentation (typically four spaces) to define code blocks. A colon (`:`) introduces a block, and statements within the block must be indented consistently. Python is case-sensitive, so `Variable` and `variable` are distinct identifiers. Statements typically end with a newline, but you can use a backslash (\) to continue a statement across multiple lines.

```
total = 1 + 2 + 3 + \
        4 + 5
print(total)  # Output: 15
```

Basic syntax rules:

- Comments start with `#` and extend to the end of the line.

- Strings can be enclosed in single quotes (`'`), double quotes (`"`), or triple quotes (`'''` or `"""`) for multi-line strings.
- Python is case-sensitive, so `Variable` and `variable` are different identifiers.

1.3 The `print()` Function

The `print()` function displays output in Python.

```
name = "Rudolf Diesel"
year = 1858
print(f"{name} was born in {year}.")
```

Output: `Rudolf Diesel was born in 1858.`

1.4 Formatting in `print()`

The following table illustrates common f-string formatting options for the `print()` function:

Format	Code	Example	Output
Round to 2 decimals	`f"{x:.2f}"`	`print(f"{3.14159:.2f}")`	`3.14`
Round to whole number	`f"{x:.0f}"`	`print(f"{3.9:.0f}")`	`4`
Thousands separator	`f"{x:,.2f}"`	`print(f"{1234567.89:,.2f}")`	`1,234,567.89`
Percentage	`f"{x:.1%}"`	`print(f"{0.756:.1%}")`	`75.6%`
Currency style	`f"${x:,.2f}"`	`print(f"${1234.5:,.2f}")`	`$1,234.50`

Note: The currency symbol (e.g., `$`) can be modified for other currencies (e.g., `€`, `£`) based on the desired locale.

1.5 Variables and Data Types

Variables store data and are assigned values using the `=` operator.

```
x = 10
y = 3.14
name = "Rudolph"
```

Python has several built-in data types, including:

- Integers (`int`): Whole numbers, e.g., `10`, `-5`
- Floating-point numbers (`float`): Decimal numbers, e.g., `3.14`, `-0.001`

- Strings (`str`): Text, e.g., `"Hello"`, `'World'`
- Booleans (`bool`): `True` or `False`

1.5.1 Arithmetic Operations

```
a = 10
b = 3
print(a + b)  # Addition: 13
print(a - b)  # Subtraction: 7
print(a * b)  # Multiplication: 30
print(a / b)  # Division: 3.3333...
print(a // b) # Integer Division: 3
print(a ** b) # Exponentiation: 1000
```

1.5.2 String Operations

```
first_name = "Rudolph"
last_name = "Diesel"
full_name = first_name + " " + last_name  # Concatenation using +
print(full_name)  # Output: Rudolph Diesel
print(f"{first_name} {last_name}")  # Concatenation using f-string
print(full_name * 2)  # Repetition: Rudolph DieselRudolph Diesel
print(full_name.upper())  # Uppercase: RUDOLPH DIESEL
```

Note: String repetition (`*`) concatenates the string multiple times without spaces. For example, `full_name * 2` produces `Rudolph DieselRudolph Diesel`.

1.6 Python as a Calculator in Interactive Mode

Python's interactive mode allows you to enter commands and see results immediately, ideal for quick calculations. To start, open a terminal (on macOS, Linux, or Windows) and type:

```
python3  # Use 'python' on Windows if 'python3' is not recognized
```

You should see the Python prompt:

```
>>>
```

Enter expressions and press **Enter** to see results:

```
2 + 3  # Output: 5
7 - 4  # Output: 3
6 * 9  # Output: 54
8 / 2  # Output: 4.0
8 // 2 # Output: 4
2 ** 3 # Output: 8
```

1.6.1 Parentheses for Grouping

```
(2 + 3) * 4  # Output: 20
2 + (3 * 4)  # Output: 14
```

1.6.2 Variables

```
x = 10
y = 3
x / y  # Output: 3.3333333333333335
```

1.6.3 Exiting Interactive Mode

To exit, type:

```
exit()
```

Alternatively, use: - **Ctrl+D** (macOS/Linux) - **Ctrl+Z** then Enter (Windows)

1.7 Control Flow

Control flow statements direct the execution of code based on conditions.

1.7.1 Conditional Statements

Conditional statements allow you to execute different code blocks based on specific conditions. Python provides three keywords for this purpose:

- **`if`**: Evaluates a condition and executes its code block if the condition is `True`.
- **`elif`**: Short for "else if," it checks an additional condition if the preceding `if` or `elif` conditions are `False`. You can use multiple `elif` statements to test multiple conditions sequentially, and Python will execute the first `True` condition's block, skipping the rest.
- **`else`**: Executes a code block if none of the preceding `if` or `elif` conditions are `True`. It serves as a fallback and does not require a condition.

The following example uses age to categorize a person as a Minor, Adult, or Senior, demonstrating how `if`, `elif`, and `else` work together.

```
# Categorize a person based on their age
age = 19
if age < 18:
    print("Minor")
elif age <= 64:
    print("Adult")
```

```
else:
    print("Senior")
```

Output: `Adult`

1.7.2 For Loop

A `for` loop iterates over a sequence (e.g., list or string).

```
components = ["piston", "liner", "connecting rod"]
for component in components:
    print(component)
```

Output:

```
piston
liner
connecting rod
```

1.7.3 While Loop

A `while` loop executes as long as a condition is true. Ensure the condition eventually becomes false to avoid infinite loops.

```
count = 0
while count <= 5:
    print(count)
    count += 1
```

Output:

```
0
1
2
3
4
5
```

1.8 Functions

1.8.1 The `def` Keyword

Functions are reusable code blocks defined using the `def` keyword. They can include default parameters for optional arguments.

```
def add(a, b=0):
    return a + b
print(add(5))      # Output: 5
```

```
print(add(5, 3))   # Output: 8

def multiply(*args):
    result = 1
    for num in args:
        result *= num
    return result
print(multiply(2, 3, 4))  # Output: 24
```

1.8.2 The `lambda` Keyword

The `lambda` keyword creates anonymous functions for short, one-off operations, often used in functional programming.

```
celsius_to_fahrenheit = lambda c: (c * 9 / 5) + 32
print(celsius_to_fahrenheit(25))  # Output: 77.0
```

1.9 The `math` Module

The `math` module provides mathematical functions and constants.

```
import math
print(math.sqrt(16))  # Output: 4.0
print(math.pi)        # Output: 3.141592653589793
```

```
import math
angle = math.pi / 4  # 45 degrees in radians
print(math.sin(angle))  # Output: 0.7071067811865475 (approximately √2/2)
print(math.cos(angle))  # Output: 0.7071067811865476 (approximately √2/2)
print(math.tan(angle))  # Output: 1.0
```

Note: Floating-point arithmetic may result in small precision differences, as seen in the `sin` and `cos` outputs.

```
import math
print(math.log(10))       # Natural logarithm of 10: 2.302585092994046
print(math.log(100, 10))  # Logarithm of 100 with base 10: 2.0
```

1.9.1 Converting Between Radians and Degrees

The `math` module provides `math.radians()` to convert degrees to radians and `math.degrees()` to convert radians to degrees, which is useful for trigonometric calculations.

```
import math
degrees = 180
radians = math.radians(degrees)
```

```
print(f"{degrees} degrees is {radians:.3f} radians")  # Output: 180 degrees is 3.142 radians

radians = math.pi / 2
degrees = math.degrees(radians)
print(f"{radians:.3f} radians is {degrees:.1f} degrees")  # Output: 1.571 radians is 90.0 degrees
```

1.10 Writing Python Scripts

Write Python code in a `.py` file and run it as a script. Create a file named `script.py`:

```
# script.py
import math
print("Square root of 16 is:", math.sqrt(16))
print("Value of pi is:", math.pi)
print("Sine of 90 degrees is:", math.sin(math.pi / 2))
print("Natural logarithm of 10 is:", math.log(10))
print("Logarithm of 100 with base 10 is:", math.log(100, 10))
```

To run the script, open a terminal, navigate to the directory containing `script.py` using the `cd` command (e.g., `cd /path/to/directory`), and type:

```
python3 script.py  # or python script.py on Windows
```

Output:

```
Square root of 16 is: 4.0
Value of pi is: 3.141592653589793
Sine of 90 degrees is: 1.0
Natural logarithm of 10 is: 2.302585092994046
Logarithm of 100 with base 10 is: 2.0
```

1.11 Summary

This tutorial covered Python basics, including syntax, variables, data types, operations, control flow, and functions. Python's rich ecosystem includes libraries like:

- **NumPy**: For numerical computations and array manipulations.
- **Matplotlib**: For data visualization and plotting.
- **Pandas**: For data manipulation and analysis with tabular data structures.
- **Pint**: For handling physical quantities and performing unit conversions.

You can explore these libraries to enhance your Python programming skills further. For example installing them can be done using `pip`:

```
pip install numpy matplotlib pandas pint
```

`pip` is Python's package manager for installing and managing additional libraries.

Chapter 2

International System of Units

2.1 SI Units

The International System of Units (SI) is the globally accepted standard for measurement. Established to provide a consistent framework for scientific and technical measurements, SI units facilitate clear communication and data comparison across various fields and countries. The system is based on seven fundamental units: the meter for length, the kilogram for mass, the second for time, the ampere for electric current, the kelvin for temperature, the mole for substance, and the candela for luminous intensity.

Table 2.1: Base SI units.

Physical Quantity	SI Base Unit	Symbol
Length	Meter	m
Mass	Kilogram	kg
Time	Second	s
Electric Current	Ampere	A
Temperature	Kelvin	K
Amount of Substance	Mole	mol
Luminous Intensity	Candela	cd

Table 2.2: Derived SI units.

Physical Quantity	Derived SI Unit	Symbol
Area	Square meter	m^2
Volume	Cubic meter	m^3
Speed	Meter per second	m/s
Acceleration	Meter per second squared	m/s^2
Force	Newton	N
Pressure	Pascal	Pa
Energy	Joule	J
Power	Watt	W
Electric Charge	Coulomb	C
Electric Potential	Volt	V
Resistance	Ohm	Ω
Capacitance	Farad	F
Frequency	Hertz	Hz
Luminous Flux	Lumen	lm
Illuminance	Lux	lx
Specific Energy	Joule per kilogram	J/kg
Specific Heat Capacity	Joule per kilogram Kelvin	$J/(kg \cdot K)$

Table 2.3: Common multiples and submultiples for SI units.

Factor	Prefix	Symbol
10^9	giga	G
10^6	mega	M
10^3	kilo	k
10^2	hecto	h
10^1	deca	da
10^{-1}	deci	d
10^{-2}	centi	c
10^{-3}	milli	m
10^{-6}	micro	µ

2.2 Unity Fraction

The **unity fraction** method, or **unit conversion using unity fractions**, is a systematic way to convert one unit of measurement into another. This method relies on multiplying by fractions that are equal to one, where the numerator and the denominator represent the same quantity in different units. Since any number multiplied by one remains the same, unity fractions allow for seamless conversion without changing the value.

The principle of unity fractions is based on:

1. **Setting up equal values**: Write a fraction where the numerator and denominator are equivalent values in different units, so the fraction equals one. For example, $\frac{1km}{1000m}$ is a unity fraction because 1 km equals 1000 m.
2. **Multiplying by unity fractions**: Multiply the initial quantity by the unity fraction(s) so that the undesired units cancel out, leaving only the desired units.

2.3 Classwork

Example 2.1. Suppose we want to convert 5 kilometers to meters.

1. Start with 5 kilometers:
$$5\,\mathrm{km}$$
2. Multiply by a unity fraction that cancels kilometers and introduces meters. We use $(\frac{1000\,\mathrm{m}}{1\,\mathrm{km}}), since\ 1\,\mathrm{km} = 1000\,\mathrm{m}$:
$$5\,\mathrm{km} \times \frac{1000\,\mathrm{m}}{1\,\mathrm{km}} = 5000\,\mathrm{m}$$
3. The kilometers km cancel out, leaving us with meters m:

$$5\,\mathrm{km} = 5000\,\mathrm{m}$$

This step-by-step approach illustrates how the unity fraction cancels the undesired units and achieves the correct result in meters.

Unity fractions can be extended by using multiple conversion steps. For example, converting hours to seconds would require two unity fractions: one to convert hours to minutes and another to convert minutes to seconds. This approach ensures accuracy and is widely used in science, engineering, and other fields that require precise unit conversions.

Example 2.2. Convert $15\,\mathrm{m/s}$ to km/h.

1. Start with $15\,\mathrm{m/s}$.
2. To convert meters to kilometers, multiply by $\frac{1\,\mathrm{km}}{1000\,\mathrm{m}}$.
3. To convert seconds to hours, multiply by $\frac{3600\,\mathrm{s}}{1\,\mathrm{h}}$.

$$15\,\mathrm{m/s} \times \frac{1\,\mathrm{km}}{1000\,\mathrm{m}} \times \frac{3600\,\mathrm{s}}{1\,\mathrm{h}} = 54\,\mathrm{km/h}$$

The meters and seconds cancel out, leaving kilometers per hour: $54\,\mathrm{km/h}$.

2.4 Problem Set

Instructions:

1. Use unity fraction to convert between derived SI units.
2. Show each step of your work to ensure accuracy.
3. Simplify your answers and include correct units.

---

1. **Speed**
   Convert $72\,\mathrm{km/h}$ to m/s.
2. **Force**
   Convert $980\,\mathrm{N}$ (newtons) to $\mathrm{kg} \cdot \mathrm{m/s^2}$.
3. **Energy**
   Convert $2500\,\mathrm{J}$ (joules) to kJ.
4. **Power**
   Convert $1500\,\mathrm{W}$ (watts) to kW.
5. **Pressure**
   Convert $101325\,\mathrm{Pa}$ (pascals) to kPa.
6. **Volume Flow Rate**
   Convert $3\,\mathrm{m^3/min}$ to L/s.

7. **Density**
   Convert $1000\,\mathrm{kg/m^3}$ to $\mathrm{g/cm^3}$.

8. **Acceleration**
   Convert $9.8\,\mathrm{m/s^2}$ to $\mathrm{cm/s^2}$.

9. **Torque**
   Convert $50\,\mathrm{N\cdot m}$ to $\mathrm{kN\cdot cm}$.

10. **Frequency**
    Convert $500\,\mathrm{Hz}$ (hertz) to kHz.

11. **Work to Energy Conversion**
    A force of $20\,\mathrm{N}$ moves an object $500\,\mathrm{cm}$. Convert the work done to joules.

12. **Kinetic Energy Conversion**
    Calculate the kinetic energy in kilojoules of a $1500\,\mathrm{kg}$ car moving at $72\,\mathrm{km/h}$.

13. **Power to Energy Conversion**
    A machine operates at $2\,\mathrm{kW}$ for 3 hours. Convert the energy used to megajoules.

14. **Pressure to Force Conversion**
    Convert a pressure of $200\,\mathrm{kPa}$ applied to an area of $0.5\,\mathrm{m^2}$ to force in newtons.

15. **Density to Mass Conversion**
    Convert $0.8\,\mathrm{g/cm^3}$ for an object with a volume of $250\,\mathrm{cm^3}$ to mass in grams.

2.4.1 Answer Key

1. $72\,\mathrm{km/h} = 20\,\mathrm{m/s}$
2. $980\,\mathrm{N} = 980\,\mathrm{kg\cdot m/s^2}$
3. $2500\,\mathrm{J} = 2.5\,\mathrm{kJ}$
4. $1500\,\mathrm{W} = 1.5\,\mathrm{kW}$
5. $101325\,\mathrm{Pa} = 101.325\,\mathrm{kPa}$
6. $3\,\mathrm{m^3/min} = 50\,\mathrm{L/s}$
7. $1000\,\mathrm{kg/m^3} = 1\,\mathrm{g/cm^3}$
8. $9.8\,\mathrm{m/s^2} = 980\,\mathrm{cm/s^2}$
9. $50\,\mathrm{N\cdot m} = 5\,\mathrm{kN\cdot cm}$
10. $500\,\mathrm{Hz} = 0.5\,\mathrm{kHz}$
11. $20\,\mathrm{N} \times 5\,\mathrm{m} = 100\,\mathrm{J}$
12. Kinetic energy $= 1500\,\mathrm{kg} \times (20\,\mathrm{m/s})^2\,/2 = 300\,\mathrm{kJ}$
13. $2\,\mathrm{kW} \times 3\,\mathrm{hours} = 21.6\,\mathrm{MJ}$
14. $200\,\mathrm{kPa} \times 0.5\,\mathrm{m^2} = 100,000\,\mathrm{N}$
15. $0.8\,\mathrm{g/cm^3} \times 250\,\mathrm{cm^3} = 200\,\mathrm{g}$

2.5 Further Reading

Introduction in Russell et al. (2021) and SI units in Bolton (2021) for additional information.

Chapter 3

Advanced Python Tutorial

This tutorial builds upon the foundational concepts introduced in the basic Python tutorial, focusing on more advanced topics. It covers virtual environments for project isolation, installing external libraries using `pip`, and applying these skills to build a unit conversion script with the Pint library. Examples are provided to demonstrate practical implementation, including conversions for speed and pressure units.

3.1 Requirements

To follow this tutorial, ensure you have Python (version 3.10 or later) installed on your computer, as detailed in the basic tutorial. You will also need access to a terminal or command prompt for creating virtual environments and installing libraries. No additional IDE is required beyond what was recommended previously, though Positron or VS Code remains suitable.

3.2 Virtual Environments (.venv)

Virtual environments in Python allow you to create isolated spaces for projects, ensuring that dependencies (libraries and versions) do not conflict across different projects. This is particularly useful when working on multiple applications that require different library versions.

3.2.1 Why Use Virtual Environments?

- **Isolation**: Each project can have its own set of installed packages without affecting the global Python installation.
- **Reproducibility**: Share your project's dependencies easily via a `requirements.txt` file.

- **Cleanliness**: Avoid cluttering your system Python with project-specific libraries.

3.2.2 Creating a Virtual Environment

To create a virtual environment named `.venv` in your project directory, open a terminal and navigate to the desired folder, then run:

```
python -m venv .venv
```

This command generates a `.venv` directory containing an isolated Python interpreter and `pip`.

3.2.3 Activating the Virtual Environment

Activation makes the virtual environment's Python and `pip` the default for your terminal session.

- On macOS/Linux:

  ```
  source .venv/bin/activate
  ```

- On Windows:

  ```
  .venv\Scripts\activate
  ```

Once activated, your terminal prompt will typically show `(.venv)` to indicate the active environment.

3.2.4 Deactivating the Virtual Environment

To exit the virtual environment and return to the global Python, simply run:

```
deactivate
```

3.3 Installing Libraries with pip

`pip` is Python's package installer, used to download and install libraries from the Python Package Index (PyPI). Within an activated virtual environment, installations are confined to that environment.

3.3.1 Installing Pint

Pint is a library for handling physical quantities and unit conversions, ensuring dimensional consistency in calculations.

To install Pint, activate your virtual environment (as described above) and run:

```
pip install pint
```

This command downloads and installs Pint and its dependencies. To verify the installation, start Python in interactive mode (e.g., `python`) and import Pint:

```
import pint
```

If no errors occur, the installation is successful.

3.3.2 Managing Dependencies

To save your project's dependencies (e.g., for sharing), generate a `requirements.txt` file:

```
pip freeze > requirements.txt
```

Others can recreate the environment by installing from this file:

```
pip install -r requirements.txt
```

3.3.3 Sample Requirements File

A `requirements.txt` file lists the libraries and their versions required for a project. Below is an example for a project using Pint:

```
pint>=0.23
```

Save this content in a file named `requirements.txt` in your project directory. You can install these dependencies in a new virtual environment using `pip install -r requirements.txt`. This ensures consistent library versions across different setups.

3.4 Building a Unit Conversion Script with Pint

Pint simplifies unit conversions by associating units with numerical values, automatically handling conversions and ensuring compatibility (e.g., preventing addition of length and mass).

3.4.1 Basic Usage of Pint

First, import Pint and create a `UnitRegistry` to manage units:

```
# Basic Usage of Pint
from pint import UnitRegistry

ureg = UnitRegistry()

# Define a quantity with units
length = 2.5 * ureg.meter

# Convert to another unit
```

```
length_in_feet = length.to(ureg.foot)
print(length_in_feet)  # Output: 8.202099737532808 foot
```

Pint supports a wide range of units, including length, mass, temperature, speed, and pressure. For temperature conversions, use the `.to()` method carefully, as some require delta considerations for differences versus absolute values.

3.4.2 Temperature Conversion

```
# Temperature Conversion

from pint import UnitRegistry

ureg = UnitRegistry()
Q_ = ureg.Quantity # shorthand

# === Absolute temperature conversions require .to() on a Quantity object ===
t_c = Q_(10, ureg.degC)
t_k = t_c.to(ureg.K)
t_f = t_c.to(ureg.degF)

print("Initial temperature:")
print(f"{t_c} = {t_k} = {t_f:.2f}")

t_c = Q_(5, ureg.degC)
t_k = t_c.to(ureg.K)
t_f = t_c.to(ureg.degF)

print("\nFinal temperature:")
print(f"{t_c} = {t_k} = {t_f:.2f}")

# === Delta temperatures (for differences) ===
delta_c = Q_(5, ureg.delta_degC)
delta_k = delta_c.to(ureg.kelvin)          # Use kelvin for differences
delta_f = delta_c.to(ureg.delta_degF)

print("\nTemperature differences:")
print(f"{delta_c} = {delta_k} = {delta_f}")

# === Difference between two absolute temperatures ===
t1_c = Q_(10, ureg.degC)
t2_c = Q_(5, ureg.degC)

delta_abs_c = t1_c - t2_c                 # 5 degC
delta_abs_k = delta_abs_c.to(ureg.kelvin) # 5 K
```

```
delta_abs_f = delta_abs_c.to(ureg.delta_degF) # 9 delta_degF

print("\nOr difference between 10 °C and 5 °C:")
print(f"{delta_abs_c} = {delta_abs_k} = {delta_abs_f}")
```

Output:

```
Initial temperature:
10 degree_Celsius = 283.15 kelvin = 50.00 degree_Fahrenheit

Final temperature:
5 degree_Celsius = 278.15 kelvin = 41.00 degree_Fahrenheit

Temperature differences:
5 delta_degree_Celsius = 5 kelvin = 9.0 delta_degree_Fahrenheit

Or difference between 10 °C and 5 °C:
5 delta_degree_Celsius = 5 kelvin = 9.0 delta_degree_Fahrenheit
```

Note that the conversion from Fahrenheit to Celsius is given:

$$t_f = (t_c \times \frac{9}{5}) + 32$$

and for temperature difference, the offset `+32` does not apply:

$$\Delta t_f = \Delta t_c \times \frac{9}{5}$$

3.4.3 Building the Script

Create a file named `unit_converter.py` in your project directory. The following script provides a command-line interface for converting various units (e.g., length, temperature, speed, pressure) using Pint. Activate your virtual environment, ensure Pint is installed, and add the code below:

```
# unit_converter.py
from pint import UnitRegistry, UndefinedUnitError, DimensionalityError
import sys

# Common unit aliases for applied mechanics
unit_aliases = {
    # Length / distance
    "m": "meter",
    "meter": "meter",
    "meters": "meter",
    "cm": "centimeter",
```

```
    "mm": "millimeter",
    "km": "kilometer",
    "ft": "foot",
    "inch": "inch",
    "in": "inch",
    "mi": "mile",

    # Force
    "n": "newton",
    "newton": "newton",
    "kgf": "kilogram_force",
    "kg_f": "kilogram_force",
    "lbf": "pound_force",
    "lb_f": "pound_force",

    # Pressure / stress
    "pa": "pascal",
    "kpa": "kPa",
    "mpa": "MPa",
    "bar": "bar",
    "psi": "psi",
    "atm": "atm",

    # Energy / work
    "j": "joule",
    "kj": "kilojoule",
    "cal": "calorie",
    "kcal": "kilocalorie",
    "ev": "electronvolt",
    "eV": "electronvolt",

    # Power
    "w": "watt",
    "kw": "kilowatt",
    "hp": "horsepower",

    # Mass
    "kg": "kilogram",
    "g": "gram",
    "lb": "pound",
    "lbs": "pound",
    "t": "ton",
    "slug": "slug",

    # Acceleration
```

```
    "m/s^2": "meter/second**2",
    "ft/s^2": "foot/second**2",
    "g": "9.80665*m/s**2",  # gravity acceleration

    # Torque / Moment
    "n*m": "newton*meter",
    "lbf*ft": "pound_force*foot",
    "kgf*m": "kilogram_force*meter",

    # Speed / velocity
    "m/s": "meter/second",
    "km/h": "km/h",
    "kph": "km/h",
    "mph": "mile/hour",
    "ft/s": "foot/second",
    "knot": "knot",

    # Temperature
    "degc": "degC",
    "degf": "degF",
    "celsius": "degC",
    "fahrenheit": "degF",
    "k": "kelvin",
    "delta_degC": "delta_degC",
    "delta_degF": "delta_degF"
}

def main():
    ureg = UnitRegistry()

    if len(sys.argv) != 4:
        print("Usage: python unit_converter.py <value> <from_unit> <to_unit>\n")
        print("Examples:")
        print("  python unit_converter.py 2.5 meter foot")
        print("  python unit_converter.py 100 degC degF")
        print("  python unit_converter.py 100 knot km/h")
        print("  python unit_converter.py 50 psi kPa")
        print("  python unit_converter.py 5000 hp kw")
        sys.exit(1)

    try:
        value = float(sys.argv[1])
        from_unit = sys.argv[2]
        to_unit = sys.argv[3]
```

```
        # Apply aliases if any
        from_unit = unit_aliases.get(from_unit.lower(), from_unit)
        to_unit = unit_aliases.get(to_unit.lower(), to_unit)

        # Use Quantity for proper temperature and unit handling
        quantity = ureg.Quantity(value, from_unit)
        converted = quantity.to(to_unit)

        # Rounded output
        print(f"{value} {from_unit} is {round(converted.magnitude, 4)} {to_unit}")

    except (UndefinedUnitError, DimensionalityError) as e:
        print(f"Error: {e}")
        print("Ensure units are valid and compatible (e.g., length to length, pressure
    except ValueError:
        print("Error: The value must be a number.")

if __name__ == "__main__":
    main()
```

3.4.4 Running the Script

Navigate to your project directory in the terminal, activate the virtual environment, and run:

```
python3 unit_converter.py
```

Output:

```
Usage: python unit_converter.py <value> <from_unit> <to_unit>

Examples:
  python unit_converter.py 100 km/h mph
  python unit_converter.py 16 knot km/h
  python unit_converter.py 3 m^3/min l/s
  python unit_converter.py 1000 kg/m^3 g/cm^3
  python unit_converter.py 7 bar psi
  python unit_converter.py 12 bar MPa
  python unit_converter.py 3000 kw hp
```

For temperature:

```
python3 unit_converter.py 100 degC degF
```

Output: `100.0 degC is 212.0 degF`

For speed:

```
python3 unit_converter.py 100 knot km_per_hour
```

Output: `100.0 knot is 185.2 km/h`

For pressure:

```
python3 unit_converter.py 50 psi kPa
```

Output: `50.0 psi is 344.7379 kPa`

```
python3 unit_converter.py 2 bar psi
```

Output: `2.0 bar is 29.0075 psi`

```
python3 unit_converter.py 3000 kw hp
```

Output: `3000.0 kilowatt is 4023.0663 horsepower`

This script handles errors for invalid units, incompatible conversions (e.g., meters to kilograms), and non-numeric inputs.

3.5 Summary

This advanced tutorial explored virtual environments for project isolation, installing libraries like Pint using `pip`, and constructing a versatile unit conversion script. The script supports conversions for length, temperature, speed (e.g., knots to km/h), and pressure (e.g., psi to kPa, bar to psi), making it useful for scientific and engineering applications. For further exploration, consult the official Pint documentation or experiment with additional units and quantities.

Part II

Making progress

Chapter 4

Python for Applied Mechanics

This tutorial extends the foundational and advanced Python concepts from previous tutorials, tailoring them to applied mechanics in engineering. It focuses on using NumPy for numerical computations involving vectors and matrices (e.g., forces, stresses), Matplotlib for visualizing mechanics data (e.g., stress-strain curves, motion plots), and integrates these with Pint for unit-aware calculations. Examples are drawn from statics, dynamics, and mechanics of materials. Assume Pint, NumPy, and Matplotlib are installed in your virtual environment (as covered in the advanced tutorial).

4.1 Requirements

Build on the advanced tutorial: Activate your virtual environment and ensure the following libraries are installed via `pip install numpy matplotlib scipy pint`. SciPy is included for numerical methods like solving differential equations.

4.2 NumPy for Vectors and Matrices

NumPy is essential for handling arrays and matrices in applied mechanics, such as representing force vectors, displacement arrays, or stiffness matrices.

4.2.1 Vector Operations

Vectors are used for forces, velocities, and moments. NumPy arrays enable efficient operations like addition (resultant forces) and dot/cross products (work or torque).

These operations are crucial in statics for equilibrium analysis or in dynamics for momentum calculations.

```
# Vector Operations using NumPy
import numpy as np

# Define two force vectors in 3D space (units: Newtons)
force1 = np.array([10, 20, 0])  # Force vector 1 along x and y directions
force2 = np.array([5, -10, 15]) # Force vector 2 along x, y, and z directions

# 1. Resultant Force
# Vector addition gives the combined effect of both forces
resultant = force1 + force2
print("Resultant Force:", resultant)
# Output: [15 10 15]

# 2. Dot Product
# Measures how much one vector acts in the direction of another
# Often interpreted as work done if one vector is force and the other is displacement
work = np.dot(force1, force2)
print("Dot Product (Work):", work)
# Output: -150

# 3. Cross Product
# Produces a vector perpendicular to both input vectors
# Often used to calculate torque (moment) vector
torque = np.cross(force1, force2)
print("Cross Product (Torque):", torque)
# Output: [ 300 -150 -200]
```

Output:

```
Resultant Force: [15 10 15]
Dot Product (Work): -150
Cross Product (Torque): [ 300 -150 -200]
```

4.3 Matplotlib for Visualizing Mechanics Data

Matplotlib allows plotting of mechanics results, such as stress-strain curves in materials testing or position-time graphs in kinematics.

4.3.1 Basic Plotting

Plot force vs. displacement for a linear spring (Hooke's law: F = kx).

```
# Plotting Force vs. Displacement for a Linear Spring (Hooke's Law)
```

```python
import numpy as np
import matplotlib.pyplot as plt

# 1. Define displacement array (from 0 to 0.1 meters)
x = np.linspace(0, 0.1, 50)  # 50 points for smooth curve

# 2. Define spring constant
k = 100  # Spring stiffness in N/m

# 3. Calculate force using Hooke's Law: F = k * x
F = k * x  # Force in Newtons corresponding to each displacement

# 4. Plotting
plt.plot(x, F, label='Force vs. Displacement', color='blue', linewidth=2)
plt.xlabel('Displacement (m)')       # x-axis label
plt.ylabel('Force (N)')              # y-axis label
plt.title('Hooke\'s Law for a Spring')  # Plot title
plt.legend()                         # Show legend
plt.grid(True)                        # Add grid lines for readability

# 5. Display the plot
plt.show()
```

This generates a line plot showing a linear relationship, useful for visualizing elastic behavior.

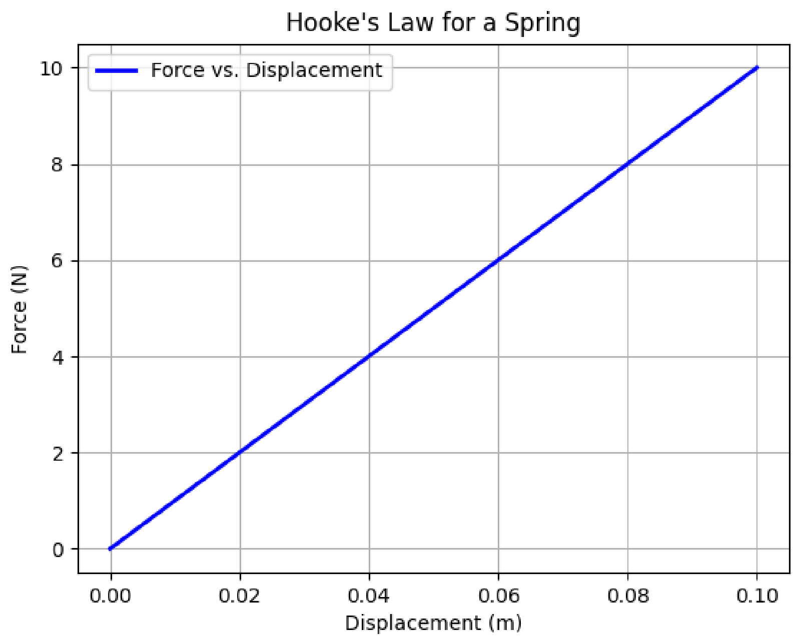


Figure 4.1: Hooke's Law

4.3.2 Advanced Plot: Stress-Strain Curve

For mechanics of materials, plot a typical stress-strain curve for steel.

```
# Plot a typical stress-strain curve for steel (elastic + plastic behavior)

import numpy as np
import matplotlib.pyplot as plt

# 1. Define strain array (dimensionless)
# Small values since steel deforms elastically up to ~0.2% strain
strain = np.linspace(0, 0.015, 100)  # 0 to 1.5% strain

# 2. Material properties
E = 200_000         # Young's modulus in MPa (elastic stiffness)
yield_strain = 0.002  # Approximate yield strain (0.2%)
yield_stress = E * yield_strain  # Stress at yield (MPa)

# 3. Stress-strain relationship
# Linear elastic region: stress = E * strain
# Plastic region: stress increases at lower slope (simplified hardening)
```

```
stress = np.where(
    strain <= yield_strain,
    E * strain,                             # Elastic region
    yield_stress + 10_000 * (strain - yield_strain)  # Plastic region with hardening
)

# 4. Plotting the curve
plt.plot(strain, stress, color='blue', linewidth=2, label='Steel Stress-Strain')
plt.xlabel('Strain (dimensionless)')
plt.ylabel('Stress (MPa)')
plt.title('Stress-Strain Curve for Steel')

# Mark the yield point
plt.axvline(x=yield_strain, color='red', linestyle='--', label='Yield Point')

plt.legend()
plt.grid(True)  # Show grid for better readability
plt.show()
```

This plot illustrates elastic and plastic regions, with a dashed line at the yield point.

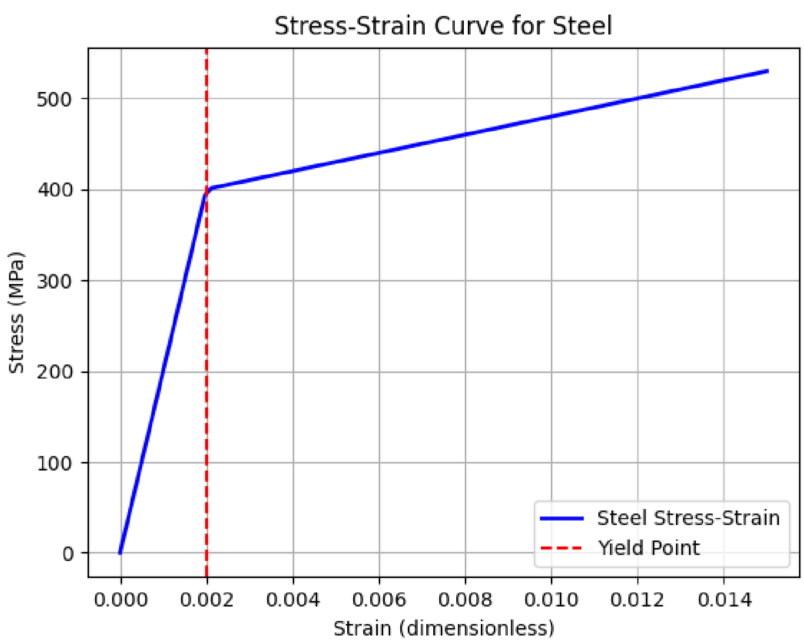


Figure 4.2: Stress-Strain Curve

4.4 Numerical Methods with SciPy

SciPy provides tools for numerical integration and solving differential equations, key in dynamics for motion simulation.

4.4.1 Solving ODEs: Simple Harmonic Motion

Model a mass-spring system: $d^2x/dt^2 + {}^2x = 0$.

```
# Simple Harmonic Motion (SHM) Simulation using ODE Integration

import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

# 1. Define the differential equation for SHM
def shm(y, t, omega):
    """
    y: [position, velocity]
    Returns derivatives [dy/dt, dv/dt]
    dv/dt = -omega^2 * position
    """
    return [y[1], -omega**2 * y[0]]

# 2. Initial conditions
y0 = [1, 0]           # Initial position = 1 m, initial velocity = 0 m/s

# 3. Time array for simulation
t = np.linspace(0, 10, 100)  # 0 to 10 seconds, 100 points

# 4. Angular frequency
omega = 1  # rad/s

# 5. Solve the ODE
sol = odeint(shm, y0, t, args=(omega,))

# 6. Plot position vs. time
plt.plot(t, sol[:, 0], label='Position (m)', color='blue', linewidth=2)
plt.xlabel('Time (s)')
plt.ylabel('Position (m)')
plt.title('Simple Harmonic Motion')
plt.grid(True)
plt.legend()
plt.show()
```

The plot shows a sinusoidal oscillation of position vs. time.

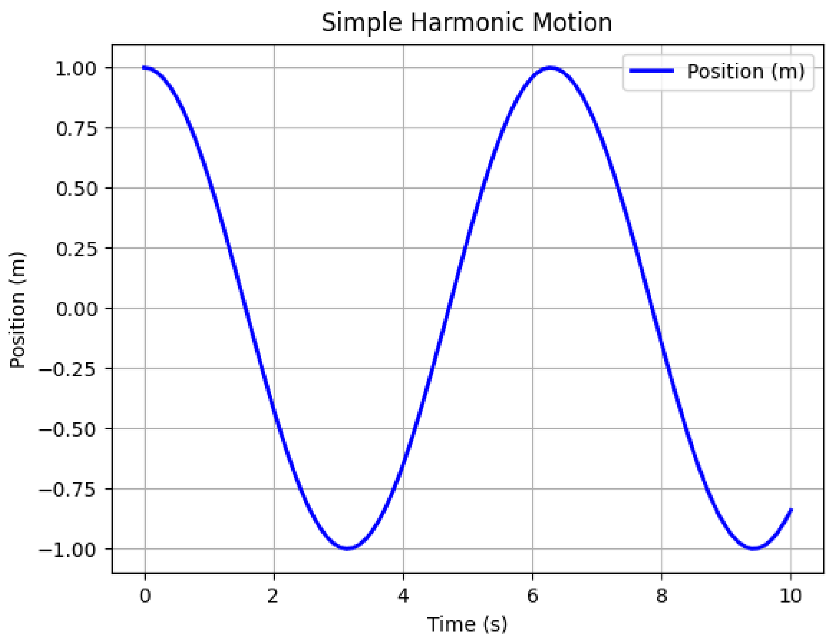


Figure 4.3: Sinusoidal oscillation of position vs. time.

4.5 Integration Example: Projectile Motion

Combine NumPy, SciPy, and Matplotlib for a dynamics problem: Simulate projectile trajectory under gravity.

```
# Simulate projectile motion under gravity using ODE integration

import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

# 1. Define the ODE system for projectile motion
def projectile(y, t, g, theta, v0):
    """
    y = [x, vx, y, vy]
    Returns derivatives [dx/dt, dvx/dt, dy/dt, dvy/dt]
    Assumes no air resistance, constant gravity.
    """
    return [y[1], 0, y[3], -g]  # dx/dt = vx, dvx/dt = 0, dy/dt = vy, dvy/dt = -g

# 2. Initial conditions
```

```
theta = np.radians(45)     # Launch angle in radians
v0 = 20                    # Initial speed in m/s
y0 = [0, v0 * np.cos(theta), 0, v0 * np.sin(theta)]  # [x0, vx0, y0, vy0]

# 3. Time array for simulation
t = np.linspace(0, 4, 100)  # 0 to 4 seconds

# 4. Gravity
g = 9.81  # m/s²

# 5. Solve the ODE
sol = odeint(projectile, y0, t, args=(g, theta, v0))

# 6. Plot trajectory (x vs. y)
plt.plot(sol[:, 0], sol[:, 2], color='blue', linewidth=2)
plt.xlabel('Horizontal Distance (m)')
plt.ylabel('Vertical Distance (m)')
plt.title('Projectile Trajectory')
plt.grid(True)
plt.show()
```

This simulates and plots the parabolic path, ignoring air resistance.

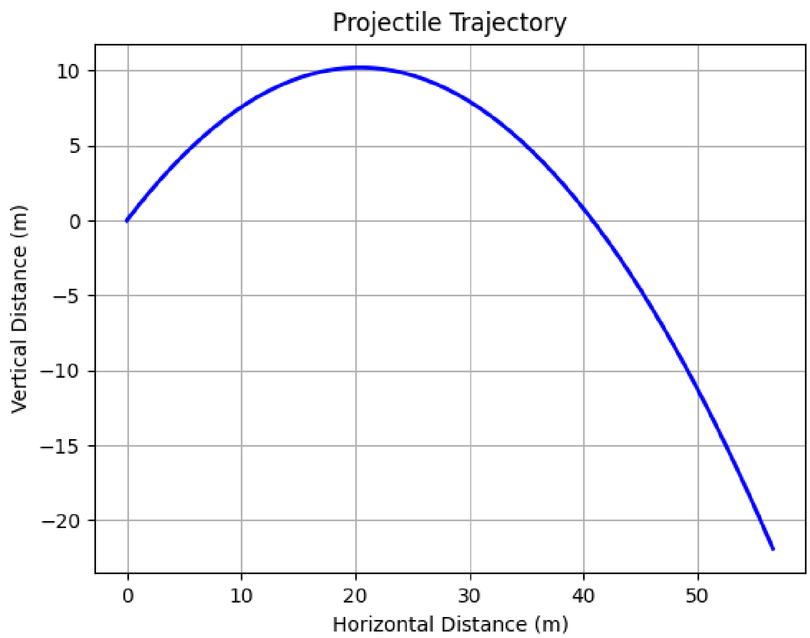


Figure 4.4: Projectile Trajectory

4.6 Summary

This tutorial applies Python tools to applied mechanics, using NumPy for computations, Matplotlib for visualization, SciPy for numerical methods, and Pint for unit handling. These techniques support analysis in statics, dynamics, and materials. Experiment with parameters or integrate with real data for class projects.

Part III

Going forward

Chapter 5

Review

We have used several books by Shaw (2017), Russell et al. (2021), Bolton (2021), Polya & Conway (2014), Bird & Ross (2020) and Bird (2021). These sources have helped you understand complex concepts.

Chapter 2:

- Purpose of SI Units: Provide a consistent framework for scientific and technical measurements.
- Advantages of SI Units: Facilitate clear communication and data comparison across various fields and countries.
- Fundamental Units of SI: Meter, kilogram, second, ampere, kelvin, mole, and candela.
- Method Name: Unity fraction method.
- Purpose: Converting one unit of measurement into another.
- Methodology: Multiplying by fractions equal to one, where the numerator and denominator represent the same quantity in different units.

References

Bird, J. O. (2021). *Bird's engineering mathematics* (Ninth edition). Routledge.

Bird, J. O., & Ross, C. T. F. (2020). *Mechanical engineering principles* (Fourth edition). Routledge.

Bolton, W. (2021). *Engineering science* (Seventh edition). Routledge.

Polya, G., & Conway, J. H. (2014). *How to solve it: A new aspect of mathematical method* (With a Foreword by John H. Con ed. edition). Princeton University Press.

Russell, P. A., Jackson, L., & Embleton, W. (2021). *Applied mechanics for marine engineers* (7th edition). Reeds.

Shaw, Z. (2017). *Learn python 3 the hard way: A very simple introduction to the terrifyingly beautiful world of computers and code* (4th edition). Addison-Wesley Professional.

Colophon

This book was created using Positron and typeset with Quarto v.1.8.22, using Pandoc for document conversion.

Appendix A

Formulae

A.1 Rules of Cosine and Sine

$$c^2 = a^2 + b^2 - 2ab\cos\gamma$$

$$\frac{a}{\sin\alpha} = \frac{b}{\sin\beta} = \frac{c}{\sin\gamma}$$

A.2 Linear Motion

$$\vec{v} = \vec{u} + \vec{a}t$$

$$\vec{s} = \frac{\vec{u} + \vec{v}}{2}t$$

$$\vec{s} = \vec{u}t + \frac{1}{2}\vec{a}t^2$$

$$\vec{v}^2 = \vec{u}^2 + 2\vec{a}\cdot\vec{s}$$

where:

- $\vec{u}$: Initial velocity
- $\vec{v}$: Final velocity
- $\vec{s}$: Displacement
- $\vec{a}$: Acceleration
- t: Time

A.3 Angular Motion

$$\omega_2 = \omega_1 \mp \alpha t$$

$$\theta = \frac{\omega_1 + \omega_2}{2} t$$

$$\theta = \omega_1 t \mp \frac{1}{2} \alpha t^2$$

$$\omega_2^2 = \omega_1^2 \mp 2\alpha\theta$$

where:

- ω_1: Initial angular velocity (rad/s)
- ω_2: Final angular velocity (rad/s)
- θ Angular displacement (rad)
- α: Angular acceleration (rad/s^2)
- t: Time (s)

A.4 Relation Between Linear and Angular Motion

The relationship between linear and angular motion is described by the following equations:

$s = r\theta$ (linear displacement s and angular displacement θ).

$v = r\omega$ (linear velocity v and angular velocity ω),

$a = r\alpha$ (linear acceleration a and angular acceleration α).

A.5 Centre of Gravity

$$\bar{x} = \frac{\sum Moments\ of\ Weights}{\sum Weights} \qquad \bar{y} = \frac{\sum Moments\ of\ Weights}{\sum Weights}$$

A.6 Centroid

$$\bar{x} = \frac{\sum \bar{x}_i\ A_i}{\sum A_i} \qquad \bar{y} = \frac{\sum \bar{y}_i\ A_i}{\sum A_i}$$

A.7 Parallel Axis Theorem

To find the moment of inertia about an axis parallel to the centroidal axis:

$$I = I_c + Ad^2$$

A.8 Radius of Gyration

$$k = \sqrt{\frac{I}{A}} \quad \text{(for area)} \quad \text{or} \quad k = \sqrt{\frac{I}{m}} \quad \text{(for mass)},$$

where:

- I: Moment of inertia about the axis
- A: Area of the cross-section (for area calculations)
- m: Mass of the body (for mass calculations)

A.8.1 Rectangle (about its centroidal axis)

- Dimensions: (b) (breadth), (h) (height)
- Radius of gyration about the centroidal x-axis:

$$k_x = \sqrt{\frac{I_x}{A}} = \sqrt{\frac{\frac{1}{12}bh^3}{bh}} = \frac{h}{\sqrt{12}}$$

A.8.2 Circle (about its centroidal axis)

- Radius: (r)
- Radius of gyration:

$$k = \sqrt{\frac{I}{A}} = \sqrt{\frac{\frac{\pi r^4}{4}}{\pi r^2}} = \frac{r}{\sqrt{2}}$$

A.9 Beam Calculations

Sum of Horizontal Forces	Sum of Vertical Force	Sum of Moments
$\sum F_x = 0$	$\sum F_y = 0$	$\sum M = 0$

Load Type	Shear Diagram Shape	Moment Diagram Shape
Point Load	Rectangular (constant)	Triangular
Uniformly Distributed Load (UDL)	Triangular	Parabolas (second degree)

A.10 Dynamics

A.10.1 Linear momentum

$$Linear\ momentum = m\vec{v}$$

Where:

- Linear momentum is in $\mathrm{kg} \cdot \mathrm{m/s}$.
- m is the mass of the object in kilograms.
- $\vec{v}$ is the velocity of the object in meters per second.

A.10.2 Angular momentum

$$\text{Angular momentum} = I\omega$$

Where:

- I is the moment of inertia in m^4.
- ω is the angular velocity in rad/s.

A.10.3 Moment of inertia

$$I = mk^2$$

Where:

- I is the moment of inertia in m^4.
- m is the mass in kg.
- k is the radius of gyration in m.

A.10.4 Turning moment

$$\tau = I\alpha$$

Where:

- τ is the torque in Nm.
- I is the moment of inertia in m^4.
- α: Angular acceleration in rad/s^2.

A.10.5 Power by Torque

$$P = \tau \cdot \omega$$

Where:

P is the power in watts (W),

τ is the torque in newton-meters (Nm), and

ω is the angular velocity in radians per second (rad/s).

A.10.6 Kinetic Energy of Rotation

$$\text{Rotational K.E.} = \frac{1}{2} I\omega^2$$

Where:

- I is the moment of inertia in m^4.
- ω is the angular velocity in radians per second (rad/s).

A.11 Stress and Strain

A.11.1 Stress

$$\sigma = \frac{F}{A}$$

$$\tau = \frac{F}{A}$$

Where:

- σ is the stress (Pa),
- τ is the shearing stress (Pa),
- F is the shearing force (N),
- A is the cross-sectional area (m^2).

A.11.2 Strain

$$\varepsilon = \frac{\Delta L}{L_0}$$

Where:

- ε is the strain (unitless),
- ΔL is the change in length,
- L_0 is the original length.

A.11.3 Hooke's Law

$$E = \frac{\sigma}{\varepsilon}$$

Where:

- σ is the stress (Pa).
- ε is the strain (unitless).
- E: Young's modulus (Pa), a material property (modulus of elasticity).

A.11.4 Factor of Safety (FOS)

$$\text{FOS} = \frac{\text{Breaking Stress}}{\text{Working Stress}}$$

A.12 Hydrodynamics

A.12.1 Volume Flow

$$\dot{v} = A \cdot C$$

Where:

$\dot{v}$: Volume flow rate, m^3/s

A: Cross-sectional area of the flow, m^2

C: Mean (average) velocity of the fluid, m/s

A.12.2 Mass Flow

$$\dot{m} = \rho \cdot \dot{v}$$

Where:

$\dot{m}$: mass flow rate, kg/s

ρ: density, kg/m^3

$\dot{v}$: volume flow, m^3/s

A.12.3 Specific Weight

$$\gamma = g \cdot \rho$$

Where:

γ: specific weight, N/m^3

g: gravitational acceleration, m/s^2

ρ: density, kg/m^3

A.12.4 Continuity Equation

$$A_1 \cdot C_1 = A_2 \cdot C_2$$

A.12.5 Energy Equation

$$Z_1 + \frac{C_1^2}{2g} + \frac{P_1}{g\rho_1} = Z_2 + \frac{C_2^2}{2g} + \frac{P_2}{g\rho_2}$$

$$Z_1 + \frac{C_1^2}{2g} + \frac{P_1}{\gamma_1} = Z_2 + \frac{C_2^2}{2g} + \frac{P_2}{\gamma_2}$$

Each term has units of m, therefore:

- Potential energy Z is known as the elevation head.
- Kinetic energy $\frac{c^2}{2g}$ is known as the velocity head.
- Pressure energy $\frac{P}{\gamma}$ is known as the pressure head.

$$Total\ Head = Elevation\ Head + Velocity\ Head + Pressure\ Head$$

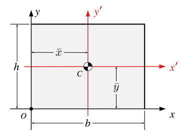
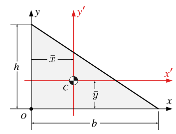
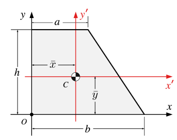
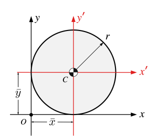
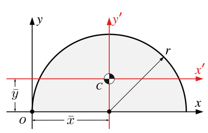
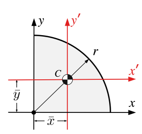
A.12.6 Bernoulli's Equation

$$P_1 + \frac{1}{2}\rho C_1^2 + \rho g h_1 = P_2 + \frac{1}{2}\rho C_2^2 + \rho g h_2$$

Where:

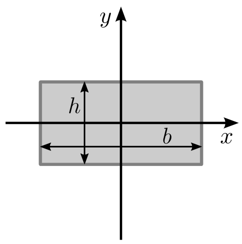
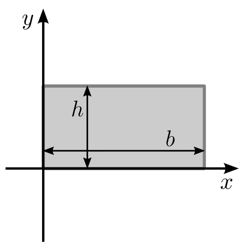
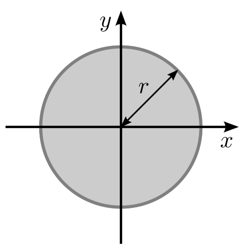
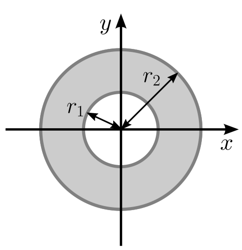
- P_1 and P_2 are the pressures at points 1 and 2, respectively.
- ρ is the density of the fluid.
- C_1 and C_2 are the velocities of the fluid at points 1 and 2, respectively.
- g is the acceleration due to gravity.
- h_1 and h_2 are the heights of the fluid at points 1 and 2, respectively.

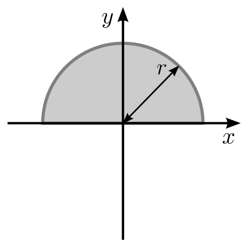
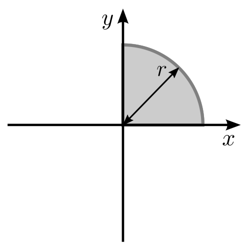
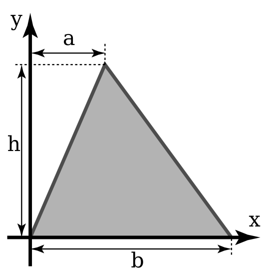
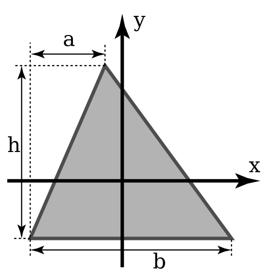
Table A.3: Centroids of Common Shapes

Shape	Area	$\bar{x}$	$\bar{y}$
	$A = bh$	$b/2$	$h/2$
	$\dfrac{bh}{2}$	$b/3$	$h/3$
	$\dfrac{(a+b)h}{2}$	$\dfrac{a^2+ab+b^2}{3(a+b)}$	$\dfrac{h(2a+b)}{3(a+b)}$
	πr^2	r	r
	$\dfrac{\pi r^2}{2}$	r	$\dfrac{4r}{3\pi}$
	$\dfrac{\pi r^2}{4}$	$\dfrac{4r}{3\pi}$	$\dfrac{4r}{3\pi}$

A.13 Second Moments of Common Shapes

Table A.4: Second moments

Shape	Second moment (I_x)	Second moment (I_y)
	$I_x = \frac{1}{12}bh^3$	$I_y = \frac{1}{12}b^3h$
	$I_x = \frac{1}{3}bh^3$	$I_y = \frac{1}{3}b^3h$
	$I_x = \frac{\pi}{4}r^4$	$I_y = \frac{\pi}{4}r^4$
	$I_x = \frac{\pi}{4}(r_2^4 - r_1^4)$	$I_y = \frac{\pi}{4}(r_2^4 - r_1^4)$

Shape	Second moment (I_x)	Second moment (I_y)
	$I_x = \frac{\pi}{8} r^4$	$I_y = \frac{\pi}{8} r^4$
	$I_x = \frac{\pi}{16} r^4$	$I_y = \frac{\pi}{16} r^4$
	$I_x = \frac{1}{12} b h^3$	
	$I_x = \frac{1}{36} b h^3$	

Appendix B

SI System Common Mistakes

Using the SI system correctly is crucial for clear communication in science and engineering. Below are common mistakes in using the SI system, examples of incorrect usage, and how to correct them.

Table B.1: SI system rules and common mistakes

Concept	Mistake	Correct Usage	Notes
Use of SI Unit Symbols	`m./s`	`m/s`	Use the correct format without additional punctuation.
Spacing Between Value & Unit	`10kg`	`10 kg`	Always leave a space between the number and the unit symbol.
Incorrect Unit Symbols	`sec`, `hrs`, `°K`	`s`, `h`, `K`	Use the proper SI symbols; symbols are case-sensitive.
Abbreviations for Units	`5 kilograms (kgs)`	`5 kilograms (kg)`	Avoid informal abbreviations like "kgs"; adhere to standard symbols.

Concept	Mistake	Correct Usage	Notes
Multiple Units in Expressions	5 m/s/s, 5 kg/meter^2	5 m/s^2, 5 kg/m^2	Use compact, standardized formats for derived units.
Incorrect Use of Prefixes	0.0001 km	100 mm	Choose prefixes to keep numbers in the range (0.1 x < 1000).
Misplaced Unit Symbols	5/s, kg10	5 s^1, 10 kg	Symbols must follow numerical values, not precede them.
Degrees Celsius vs. Kelvin	300°K	300 K	Kelvin is written without "degree"
Singular vs. Plural Units	5 kgs, 1 meters	5 kg, 1 meter	Symbols do not pluralize; full unit names follow grammar rules.
Capitalization of Symbols	Kg, S, Km, MA	kg, s, km, mA	Symbols are case-sensitive; use uppercase only where specified (e.g., N, Pa).
Capitalization of Unit Names	Newton, Pascal, Watt	newton, pascal, watt	Unit names are lowercase, even if derived from a person's name, unless starting a sentence.
Prefix Capitalization	MilliMeter, MegaWatt	millimeter, megawatt	Prefixes are lowercase for (10^{-1}) to (10^{-9}), uppercase for (10^6) and larger (except k for kilo).
Formatting in Reports	5, Temperature: 300	5 kg, Temperature: 300 K	Always specify units explicitly.

Appendix C

Greek Letters

The following tables present the names of Greek letters and selected symbols commonly used in engineering courses, ensuring precise reference and avoiding reliance on informal descriptors such as "squiggle."

Table C.1: Greek letters.

Lower Case	Upper Case	Name
α	A	alpha
β	B	beta
γ	Γ	gamma
δ	Δ	delta
ϵ	E	epsilon
ζ	Z	zeta
η	E	eta
θ	Θ	theta
ι	I	iota
κ	K	kappa
λ	Λ	lambda
μ	M	mu
ν	N	nu
ξ	Ξ	xi
o	O	omicron
π	Π	pi
ρ	P	rho
σ	Σ	sigma
τ	T	tau
υ	Υ	upsilon
ϕ	Φ	phi
χ	X	chi

Lower Case	Upper Case	Name
ψ	Ψ	psi
ω	Ω	omega

Table C.2: Commonly used symbols in engineering courses.

Symbol	Name	Use	Course
Δ	Delta	Change	Thermodynamics
Δ	Delta	Displacement	Naval Architecture
∇	Nabla	Volume	Naval Architecture
Σ	Sigma	Sum	Thermodynamics, Naval Architecture, Applied Mechanics
σ	Sigma	Stress	Thermodynamics, Applied Mechanics
ϵ	Epsilon	Modulus of elasticity	Thermodynamics, Applied Mechanics
η	Eta	Efficiency	Thermodynamics
ω	Omega	Angular velocity	Thermodynamics, Applied Mechanics
ρ	Rho	Density	Thermodynamics, Naval Architecture
τ	Tau	Torque	Thermodynamics, Applied Mechanics

Appendix D

Revision History

Table D.1: Changelog.

Version	Date	Description
2.0	2026-01-	Major revision with new chapters
1.0	2025-09-	Updated SI units section
0.2	2025-09-03	Added formulae section
0.1	2025-09-01	Initial development

Index