

# Applied Mechanics Tutorial

A Concise Introduction to Computational Tools

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

<b>Preface</b>	<b>1</b>
<b>Attributions</b>	<b>3</b>
<b>Study Guide</b>	<b>5</b>
<b>Python Tutorial</b>	<b>7</b>
Requirements . . . . .	7
Basic Syntax . . . . .	7
The <code>print()</code> Function . . . . .	8
Variables and Data Types . . . . .	8
Arithmetic Operations . . . . .	9
String Operations . . . . .	9
Python as a Calculator in Interactive Mode . . . . .	9
Parentheses for Grouping . . . . .	10
Variables . . . . .	10
Exiting Interactive Mode . . . . .	10
Control Flow . . . . .	10
Conditional Statements . . . . .	10
For Loop . . . . .	11
While Loop . . . . .	11
Functions . . . . .	11
The <code>def</code> Keyword . . . . .	11
The <code>lambda</code> Keyword . . . . .	12
The <code>math</code> Module . . . . .	12
Converting Between Radians and Degrees . . . . .	12
Writing Python Scripts . . . . .	13
Python Style Guide . . . . .	13
Summary . . . . .	15
<b>1 Statics</b>	<b>17</b>
1.1 Space Diagrams . . . . .	17
1.2 Vector Diagrams . . . . .	17

1.2.1	Resultant . . . . .	17
1.2.2	Equilibrium . . . . .	17
1.3	Conditions of Equilibrium . . . . .	19
1.4	Cosine Rule . . . . .	19
1.5	Sine Rule . . . . .	21
1.6	Problem Set . . . . .	21
<b>2</b>	<b>Dynamics</b>	<b>29</b>
2.1	Inertia . . . . .	29
2.2	Momentum . . . . .	30
2.3	Newton's Laws of Motion . . . . .	30
2.4	Linear momentum . . . . .	32
2.4.1	Conservation of Linear Momentum . . . . .	32
2.5	Angular Momentum . . . . .	34
2.6	The Radius of Gyration . . . . .	35
2.7	Turning Moment . . . . .	37
2.8	Power by Torque . . . . .	40
2.9	Kinetic Energy of Rotation . . . . .	43
<b>3</b>	<b>Hydrostatics</b>	<b>47</b>
3.1	Pressure Head . . . . .	47
3.2	Load On Immersed Surfaces . . . . .	47
3.3	Hydraulic Jacks . . . . .	58
<b>4</b>	<b>Hydrodynamics</b>	<b>61</b>
4.1	Volume Flow Rate . . . . .	61
4.2	Mass Flow Rate . . . . .	62
4.3	Discharge Through an Orifice . . . . .	64
4.4	Continuity Equation . . . . .	67
4.5	The Energy Equation for an Ideal Fluid . . . . .	70
4.6	Bernoulli's Equation . . . . .	74
4.7	Venturi Meter . . . . .	74
	<b>References</b>	<b>79</b>
	<b>Author</b>	<b>81</b>
	<b>Appendices</b>	<b>83</b>
<b>A</b>	<b>Applied Mechanics Formulae</b>	<b>83</b>
A.1	Rules of Cosine and Sine . . . . .	83
A.2	Linear Motion . . . . .	83
A.3	Angular Motion . . . . .	84
A.4	Relation Between Linear and Angular Motion . . . . .	84
A.5	Centre of Gravity . . . . .	84
A.6	Centroid . . . . .	84

A.7	Parallel Axis Theorem . . . . .	85
A.8	Radius of Gyration . . . . .	85
A.8.1	Rectangle (about its centroidal axis) . . . . .	85
A.8.2	Circle (about its centroidal axis) . . . . .	85
A.9	Beam Calculations . . . . .	85
A.10	Dynamics . . . . .	86
A.10.1	Linear momentum . . . . .	86
A.10.2	Angular momentum . . . . .	86
A.10.3	Moment of inertia . . . . .	86
A.10.4	Turning moment . . . . .	86
A.10.5	Power by Torque . . . . .	87
A.10.6	Kinetic Energy of Rotation . . . . .	87
A.11	Stress and Strain . . . . .	87
A.11.1	Stress . . . . .	87
A.11.2	Strain . . . . .	87
A.11.3	Hooke's Law . . . . .	88
A.11.4	Factor of Safety (FOS) . . . . .	88
A.12	Hydrodynamics . . . . .	88
A.12.1	Volume Flow . . . . .	88
A.12.2	Mass Flow . . . . .	88
A.12.3	Specific Weight . . . . .	88
A.12.4	Continuity Equation . . . . .	89
A.12.5	Energy Equation . . . . .	89
A.12.6	Bernoulli's Equation . . . . .	89
A.13	Second Moments of Common Shapes . . . . .	91



# List of Figures

1	Input parameters . . . . .	5
1.1	Space Diagram . . . . .	18
1.2	Vector Diagram . . . . .	18
1.3	Equilibrant . . . . .	19
1.4	Rules of Cosine and Sine . . . . .	20
3.1	Barometer . . . . .	48
3.2	U-tube manometer . . . . .	49
3.3	Hydraulic lift . . . . .	59
4.1	Orifice . . . . .	64
4.2	Vena contracta . . . . .	65
4.3	Continuity equation . . . . .	67
4.4	Energy equation . . . . .	71
4.5	Venturi meter . . . . .	75



# List of Tables

A.3	Centroids of Common Shapes . . . . .	90
A.4	Second moments . . . . .	91
B.1	Greek letters. . . . .	93
B.2	Commonly used symbols in engineering courses. . . . .	94



# Preface

A Concise Introduction to Computational Tools provides a series of tutorials derived from lecture notes, offering clear and focused guidance on essential topics. No prior programming experience is required.

Students are expected to be comfortable with college-level mathematics and science and are strongly encouraged to consult reliable reference texts. For example, foundational and practical texts in mathematics, applied mechanics and computational tools, including Bird (2021), Hannah & Hillier (1995), Russell et al. (2021), Bolton (2021), Shaw (2017), Sweigart (2021), and Sweigart (2020).

The tutorials also assume working familiarity with either macOS or Microsoft Windows. For best results, these materials should be used on a laptop or desktop computer rather than a tablet or smartphone.



# Attributions

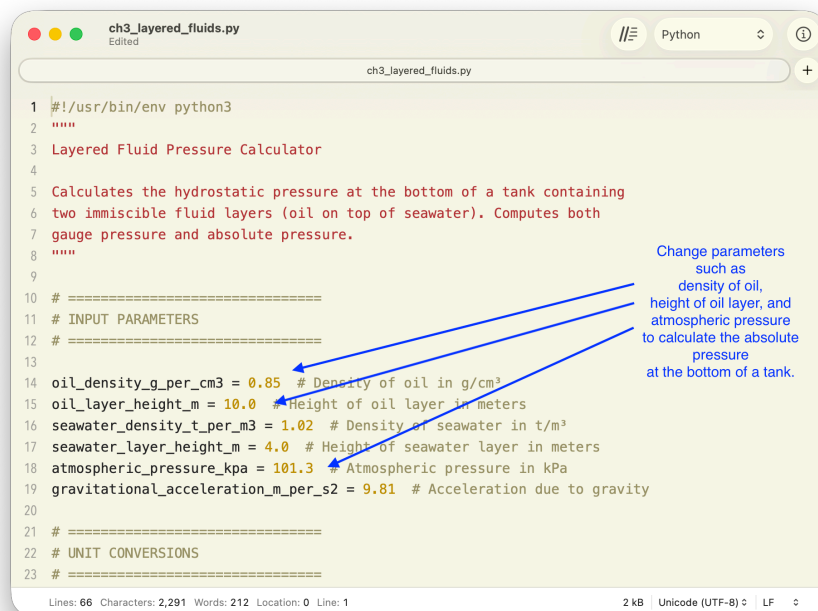
This book incorporates figures from the following sources:

- Figures in the Chapter 3 and Chapter 4 chapters are sourced from the Applied Fluid Mechanics Online Textbook.
- Figures in the Appendix A section, including centroids and second moments of area, are adapted from the Wikipedia pages on Centroids and Second moments of area.



# Study Guide

First and foremost, solve the problem sets using only pen, paper, and a calculator. After completing a problem by hand, slightly modify the conditions or variables (Figure 1) and solve it again. Then, input the new values into the Python script and compare the results with your hand-calculated solution.



```
1 #!/usr/bin/env python3
2 """
3 Layered Fluid Pressure Calculator
4
5 Calculates the hydrostatic pressure at the bottom of a tank containing
6 two immiscible fluid layers (oil on top of seawater). Computes both
7 gauge pressure and absolute pressure.
8 """
9
10 # =====
11 # INPUT PARAMETERS
12 # =====
13
14 oil_density_g_per_cm3 = 0.85 # Density of oil in g/cm³
15 oil_layer_height_m = 10.0 # Height of oil layer in meters
16 seawater_density_t_per_m3 = 1.02 # Density of seawater in t/m³
17 seawater_layer_height_m = 4.0 # Height of seawater layer in meters
18 atmospheric_pressure_kpa = 101.3 # Atmospheric pressure in kPa
19 gravitational_acceleration_m_per_s2 = 9.81 # Acceleration due to gravity
20
21 # =====
22 # UNIT CONVERSIONS
23 # =====
```

Change parameters such as density of oil, height of oil layer, and atmospheric pressure to calculate the absolute pressure at the bottom of a tank.

Lines: 66 Characters: 2,291 Words: 212 Location: 0 Line: 1 2 kB Unicode (UTF-8) LF

Figure 1: Input parameters

- Buddy system: Study with a classmate. Teaching and helping one another significantly deepens understanding of the material. Students are strongly encouraged to work through problem sets collaboratively. For example, one student solves the odd-numbered problems while the other solves the

even-numbered ones; then each explains their solutions to the other.

- Practice, practice, practice: As the saying goes, “practice makes perfect.” More accurately in this context, “good practice makes perfect.” That means pen-and-paper-first practice. Use Python as a verification tool, not a crutch. Always derive the solution analytically on paper first. Only after obtaining a complete hand-calculated result should you run the corresponding Python script. This approach echoes a proverb commonly attributed to Confucius:

I hear and I forget

I see and I remember

I do and I understand

- Maintain a solution notebook: For each major problem, keep a single document (physical or digital) containing:
  - The full hand derivation
  - Key numerical results with units
  - A brief paragraph summarizing insights or reconciling any intermediate discrepancies

This record becomes an invaluable review resource before exams.

# 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.

## Requirements

To follow this tutorial, the easiest way to get started is by using a web-based Python environment. This lets you write and run Python code right in your browser; no downloads or setup needed. I recommend [python-fiddle.com](http://python-fiddle.com), an easy-to-use online editor that lets you experiment with Python instantly and solve your problem sets effortlessly.

If you prefer working on your own computer, make sure you have Python (version 3.14 or later) installed. Python works on Windows, macOS, and Linux. You'll also need a text editor or an Integrated Development Environment (IDE) to write your code. I recommend Positron, a beginner-friendly IDE with a built-in terminal, though other editors like VS Code or PyCharm are also good options.

## 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.

## 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.

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

Format	Code	Example	Output
<b>Round to 2 decimals</b>	<code>f"{x:.2f}"</code>	<code>print(f"{3.1415:.2f}")</code>	3.14
<b>Round to whole number</b>	<code>f"{x:.0f}"</code>	<code>print(f"{3.9:.0f}")</code>	4
<b>Thousands separator</b>	<code>f"{x:, .2f}"</code>	<code>print(f"{1234.567:.2f}")</code>	1,234.57
<b>Percentage</b>	<code>f"{x:.1%}"</code>	<code>print(f"{0.756:.1%}")</code>	75.6%
<b>Currency (JPY)</b>	<code>f"¥{x:,.0f}"</code>	<code>print(f"¥{1234:,.0f}")</code>	¥1,234

## 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

## 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
```

## 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`.

## 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
```

## Parentheses for Grouping

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

## Variables

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

## Exiting Interactive Mode

To exit, type:

```
exit()
```

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

## Control Flow

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

### 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

## 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
```

## 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
```

## Functions

### 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
```

## 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
```

## 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
print(math.cos(angle))  # Output: 0.7071067811865476
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: 2.302585092
print(math.log(100, 10)) # Logarithm of 100 with base 10: 2.0
```

## 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.

 Warning

In Python, the trigonometric functions in the `math` module (`sin`, `cos`, `tan`, etc.) expect angles in radians, not degrees. Always convert degrees to radians before using these functions.

```
import math
degrees = 180
radians = math.radians(degrees)
print(f"{degrees} degrees is {radians:.3f} radians")
radians = math.pi
degrees = math.degrees(radians)
print(f"{radians:.3f} radians is {degrees:.1f} degrees")
```

## 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
```

## Python Style Guide

Code is read far more often than it is written. — Guido van Rossum,  
creator of the Python programming language.

Adhere to PEP 8 as the primary style guide. This document outlines the coding

conventions for Python code in the core standard library of the main Python distribution.

- Employ descriptive names for variables and functions. Avoid abbreviations or single-letter names unless they are widely conventional (e.g., `i` as a loop index).
- Organize code with consistent sectioning and clear headers or separators to enhance readability.
- Restrict line length to a maximum of 79 characters to ensure compatibility with print media and prevent horizontal overflow.
- Include meaningful inline comments where necessary to explain the purpose and intent of the code, particularly for complex logic.

Many modern Python projects enhance PEP 8 compliance by employing Black, the uncompromising code formatter. Black applies a deterministic style that is a strict subset of PEP 8, with a default line length of 88 characters for improved readability on contemporary displays. It can be configured to use 79 characters if required.

The following scripts illustrate the earlier examples in accordance with PEP 8 guidelines:

```
"""
Degrees-Radians Conversion
"""

import math

# Conversion from degrees to radians
input_degrees = 180.0
converted_radians = math.radians(input_degrees)

# Conversion from radians to degrees
input_radians = math.pi
converted_degrees = math.degrees(input_radians)

# Display results
print("Degrees to Radians Conversion:")
print(f"{input_degrees:.0f} degrees = {converted_radians:.3f} radians")
print()
print("Radians to Degrees Conversion:")
print(f"{input_radians:.3f} radians = {converted_degrees:.1f} degrees")

"""
Math Module Examples
```

```
Demonstrates common functions from the Python math module:  
square root, pi constant, sine, natural logarithm,  
and base-10 logarithm.  
"""  
  
import math  
  
# Compute and display mathematical operations  
sqrt_result = math.sqrt(16)  
pi_value = math.pi  
sine_90_degrees = math.sin(math.pi / 2) # 90 degrees = /2 radians  
natural_log_10 = math.log(10)  
log_base10_100 = math.log(100, 10)  
  
# Display results with clear descriptions  
print("Square root of 16 is:", sqrt_result)  
print("Value of pi is:", pi_value)  
print("Sine of 90 degrees is:", sine_90_degrees)  
print("Natural logarithm of 10 is:", natural_log_10)  
print("Logarithm of 100 with base 10 is:", log_base10_100)
```

## Summary

You have now learned Python basics: syntax, variables, control flow, functions, and the math module. Practice all examples instantly at [python-fiddle.com](https://python-fiddle.com).



# Chapter 1

## Statics

Mechanics is the branch of physics and engineering concerned with the behavior of bodies under the action of forces, encompassing statics and dynamics. This chapter introduces statics, the study of bodies in equilibrium where net forces and moments are zero. We examine balanced force systems, including space and vector diagrams, conditions of equilibrium, and the calculation of resultant forces and moments.

### 1.1 Space Diagrams

A space diagram illustrates the geometry of the system and the applied forces.

### 1.2 Vector Diagrams

A vector diagram (or force polygon) is a **scaled** graphical construction in which force vectors are placed head-to-tail in succession. It is used to determine the resultant of a system of forces through vector addition, as well as to resolve equilibrium problems when the polygon closes.

#### 1.2.1 Resultant

The resultant is a single force that has exactly the same effect on an object as all the original forces acting together. It is found by adding the forces vectorially taking into account both their magnitude and direction. The resultant gives the overall direction and magnitude of the combined forces as shown in Figure 1.2.

#### 1.2.2 Equilibrium

Equilibrium of an object occurs when all the forces acting on it are balanced, so the object remains at rest or moves at a constant speed in a straight line.

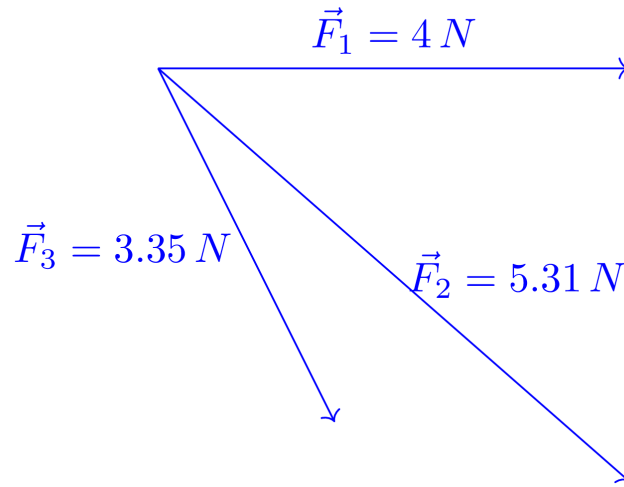


Figure 1.1: Space Diagram

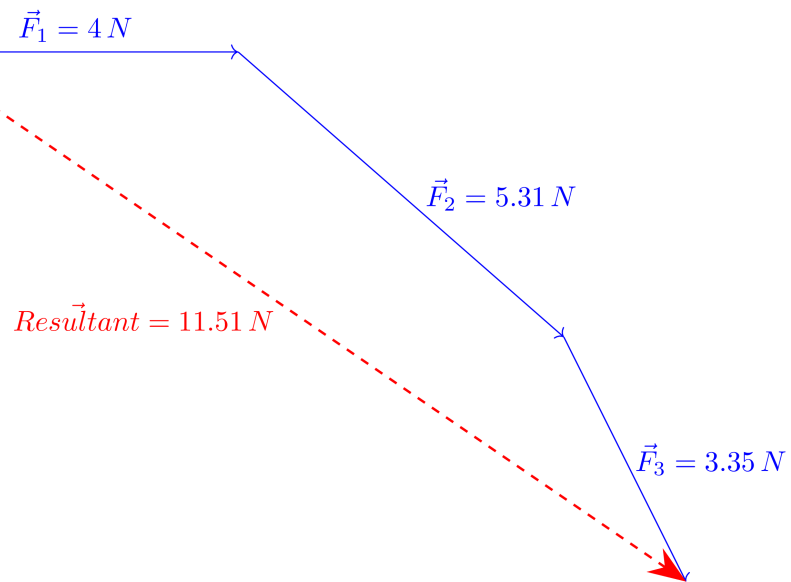


Figure 1.2: Vector Diagram

In equilibrium, as shown in Figure 1.3, there is no net force or acceleration, meaning the object is in a stable state without any change in its motion.

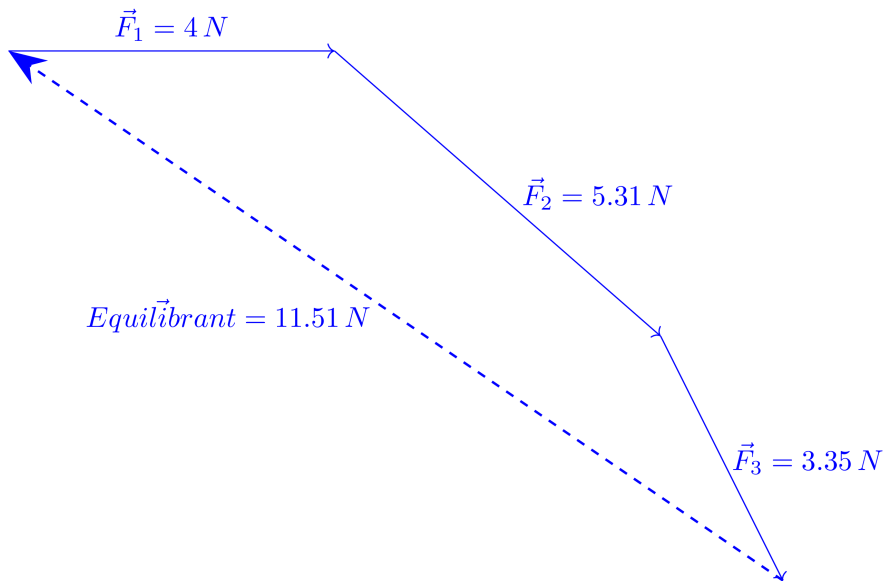


Figure 1.3: Equilibrant

### 1.3 Conditions of Equilibrium

1. Net force must be zero:

$$\sum_k \vec{F}_k = \vec{0} \quad (1.1)$$

2. Net torque must be zero:

$$\sum_k \vec{\tau}_k = \vec{0} \quad (1.2)$$

### 1.4 Cosine Rule

The Cosine Rule is used to relate the lengths of the sides of a triangle as shown in Figure 1.4 to the cosine of one of its angles:

$$c^2 = a^2 + b^2 - 2ab \cos \gamma \quad (1.3)$$

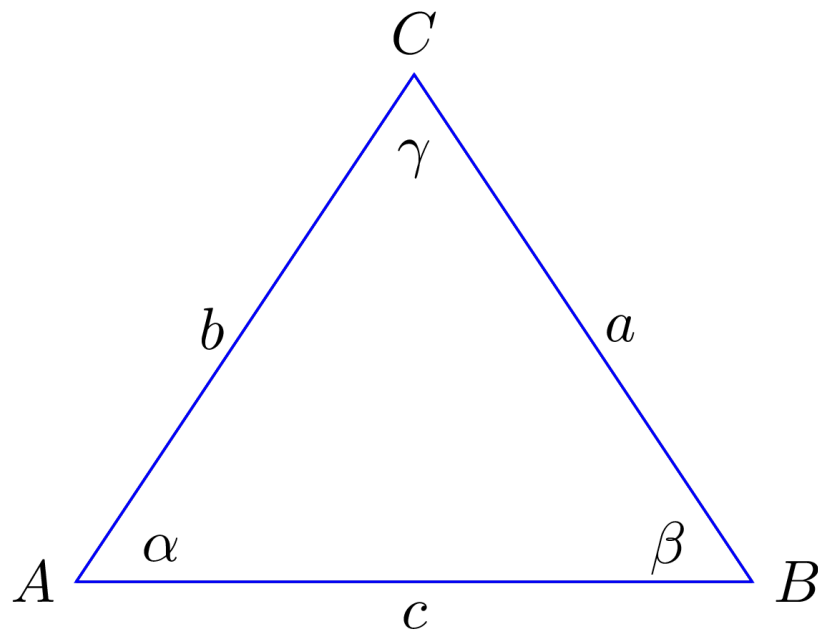


Figure 1.4: Rules of Cosine and Sine

Where:

- a, b, c are the sides of the triangle.
- $\gamma$  is the angle opposite side c.

## 1.5 Sine Rule

The Sine Rule relates the sides and angles of a triangle in Figure 1.4:

$$\frac{a}{\sin \alpha} = \frac{b}{\sin \beta} = \frac{c}{\sin \gamma} \quad (1.4)$$

Where:

- $\alpha, \beta, \gamma$  are the angles of the triangle.
- a, b, c are the sides of the triangle opposite to angles  $\alpha, \beta, \gamma$  respectively.

## 1.6 Problem Set

**Example 1.1.** Two slings of equal length are slung from a horizontal beam and connected to a ring at their lower ends, the slings and beam forming an equilateral triangle. Find the force in each sling when a load of 30 kN hangs from the ring.

### Warning

In Python, the trigonometric functions in the `math` module (`sin`, `cos`, `tan`, etc.) expect angles in radians, not degrees. Always convert degrees to radians before using these functions.

```
"""
Sling Tension Calculator

Calculates the tension in each sling of a two-sling system
supporting a vertical load, the slings and beam forming
an equilateral triangle.
"""

import math

# =====
# INPUT PARAMETERS
# =====
```

```

vertical_load_kn = 30 # Total vertical load in kilonewtons
sling_angle_degrees = 60 # Angle each sling makes with horizontal

# =====
# CALCULATIONS
# =====

# Convert angle from degrees to radians for trigonometric functions
sling_angle_radians = math.radians(sling_angle_degrees)

# Apply vertical equilibrium equation:
# Sum of vertical forces = 0
# 2 * T * sin() = W
# where T is tension in each sling, is angle, W is vertical load
tension_per_sling_kn = vertical_load_kn / (2 * math.sin(sling_angle_radians))

# =====
# OUTPUT RESULTS
# =====

print(f"Tension in each sling: {tension_per_sling_kn:.4f} kN")

```

**Example 1.2.** Two identical slings of equal length are attached to a horizontal beam and meet at a ring from which a 45 kN load is suspended. The slings and the beam form an isosceles triangle in which each sling makes an angle of  $50^\circ$  with the horizontal. Find the force (tension) in each sling.

```

"""
Sling Tension Calculator

Calculates the tension in each sling of a two-sling system
supporting a vertical load, the slings and the beam form
an isosceles triangle.
"""

import math

# =====
# INPUT PARAMETERS
# =====

vertical_load_kn = 45 # Total vertical load in kilonewtons
sling_angle_degrees = 50 # Angle each sling makes with horizontal

# =====
# CALCULATIONS

```

```
# =====

# Convert angle from degrees to radians for trigonometric functions
sling_angle_radians = math.radians(sling_angle_degrees)

# Apply vertical equilibrium equation:
# Sum of vertical forces = 0
# 2 * T * sin() = W
# where T is tension in each sling, is angle, W is vertical load
tension_per_sling_kn = vertical_load_kn / (2 * math.sin(sling_angle_radians))

# =====
# OUTPUT RESULTS
# =====

print(f"Tension in each sling: {tension_per_sling_kn:.4f} kN")
```

**Example 1.3.** Two lifting ropes are connected at their lower ends to a common shackle from which a load of 25 kN hangs. If the ropes make angles of 32° and 42° respectively to the vertical, find the tension in each rope.

```
"""
Rope Tension Calculator (Law of Sines)

Calculates tensions in two ropes supporting a load using the law of sines.
The system forms a triangle where the angles and load are known.
"""

import math

# =====
# INPUT PARAMETERS
# =====

vertical_load_kn = 25 # Total vertical load in kilonewtons
angle_opposite_rope2_deg = 32 # Angle opposite to tension T2 (degrees)
angle_opposite_rope1_deg = 42 # Angle opposite to tension T1 (degrees)

# Calculate the third angle of the triangle (angles sum to 180°)
angle_opposite_load_deg = 180 - (angle_opposite_rope2_deg + angle_opposite_rope1_deg)

# =====
# CALCULATIONS
# =====

# Convert all angles from degrees to radians for trigonometric functions
```

```

angle_opposite_rope2_rad = math.radians(angle_opposite_rope2_deg)
angle_opposite_rope1_rad = math.radians(angle_opposite_rope1_deg)
angle_opposite_load_rad = math.radians(angle_opposite_load_deg)

# Apply the law of sines to find tensions:
# T1 / sin(1) = T2 / sin(2) = W / sin(3)
# where 1 is opposite T1, 2 is opposite T2, 3 is opposite W

# Tension in rope 1 (opposite to angle_opposite_rope1)
tension_rope1_kn = (
    vertical_load_kn
    * math.sin(angle_opposite_rope2_rad)
    / math.sin(angle_opposite_load_rad)
)

# Tension in rope 2 (opposite to angle_opposite_rope2)
tension_rope2_kn = (
    vertical_load_kn
    * math.sin(angle_opposite_rope1_rad)
    / math.sin(angle_opposite_load_rad)
)

# =====
# OUTPUT RESULTS
# =====

print(f"Tension in rope 1: {tension_rope1_kn:.4f} kN")
print(f"Tension in rope 2: {tension_rope2_kn:.4f} kN")

```

**Example 1.4.** The angle between the jib and vertical post of a jib crane is  $40^\circ$ , and the angle between the jib and tie is  $45^\circ$ . Find the force in the jib and tie when a load of 15 kN hangs from the crane head.

```

"""
Jib and Tie Force Calculator

Calculates forces in a jib and tie member supporting a vertical load.
Uses the law of sines to solve the force triangle.
"""

import math

# =====
# INPUT PARAMETERS
# =====

```

```

vertical_load_kn = 15 # Total vertical load in kilonewtons
angle_jib_to_vertical_deg = 40 # Angle between jib and vertical (degrees)
angle_jib_to_tie_deg = 45 # Angle between jib and tie (degrees)

# Calculate the third angle in the force triangle (angles sum to 180°)
angle_tie_to_load_deg = 180 - angle_jib_to_vertical_deg - angle_jib_to_tie_deg

# =====
# CALCULATIONS
# =====

# Convert all angles from degrees to radians for trigonometric functions
angle_jib_to_vertical_rad = math.radians(angle_jib_to_vertical_deg)
angle_jib_to_tie_rad = math.radians(angle_jib_to_tie_deg)
angle_tie_to_load_rad = math.radians(angle_tie_to_load_deg)

# Apply the law of sines to find forces in jib and tie:
#  $F_{\text{jib}} / \sin(\text{\_opposite\_jib}) = F_{\text{tie}} / \sin(\text{\_opposite\_tie}) = W / \sin(\text{\_ref})$ 
# where the reference angle is the angle between jib and tie

# Force in jib (compression or tension depending on configuration)
force_jib_kn = (
    vertical_load_kn * math.sin(angle_tie_to_load_rad) / math.sin(angle_jib_to_tie_rad)
)

# Force in tie (tension)
force_tie_kn = (
    vertical_load_kn
    * math.sin(angle_jib_to_vertical_rad)
    / math.sin(angle_jib_to_tie_rad)
)

# =====
# OUTPUT RESULTS
# =====

print(f"Force in jib: {force_jib_kn:.4f} kN")
print(f"Force in tie: {force_tie_kn:.4f} kN")

```

**Example 1.5.** The lengths of the vertical post, jib, and tie of a jib crane are 8, 13, and 9 m, respectively. Find the forces in the jib and tie when a load of 20 kN hangs from the crane head.

```

"""
Structural Force Calculator

```

```

Calculates forces in a triangular structure (post, jib, tie) given the
member lengths and applied load. Uses law of cosines to find angles,
then law of sines to determine member forces.
"""

import math

# =====
# INPUT PARAMETERS
# =====

post_length_m = 8 # Length of post member in meters
jib_length_m = 13 # Length of jib member in meters
tie_length_m = 9 # Length of tie member in meters
applied_load_kn = 20 # Applied vertical load in kilonewtons

# =====
# ANGLE CALCULATIONS (LAW OF COSINES)
# =====

# Calculate angle opposite to post using law of cosines:
#  $\cos(\theta_p) = (J^2 + T^2 - P^2) / (2 \cdot J \cdot T)$ 
angle_opposite_post_rad = math.acos(
    (jib_length_m**2 + tie_length_m**2 - post_length_m**2)
    / (2 * jib_length_m * tie_length_m)
)

# Calculate angle opposite to tie using law of cosines:
#  $\cos(\theta_t) = (J^2 + P^2 - T^2) / (2 \cdot J \cdot P)$ 
angle_opposite_tie_rad = math.acos(
    (jib_length_m**2 + post_length_m**2 - tie_length_m**2)
    / (2 * jib_length_m * post_length_m)
)

# Calculate angle opposite to jib (angles in triangle sum to radians)
angle_opposite_jib_rad = math.pi - angle_opposite_post_rad - angle_opposite_tie_rad

# =====
# FORCE CALCULATIONS (LAW OF SINES)
# =====

# Apply law of sines to find member forces:
#  $F_{jib} / \sin(\theta_{jib}) = F_{tie} / \sin(\theta_{tie}) = W / \sin(\theta_{post})$ 

# Force in jib member

```

```
force_jib_kn = (  
    applied_load_kn  
    * math.sin(angle_opposite_jib_rad)  
    / math.sin(angle_opposite_post_rad)  
)  
  
# Force in tie member  
force_tie_kn = (  
    applied_load_kn  
    * math.sin(angle_opposite_tie_rad)  
    / math.sin(angle_opposite_post_rad)  
)  
  
# =====  
# CONVERT ANGLES TO DEGREES FOR OUTPUT  
# =====  
  
angle_opposite_post_deg = math.degrees(angle_opposite_post_rad)  
angle_opposite_tie_deg = math.degrees(angle_opposite_tie_rad)  
angle_opposite_jib_deg = math.degrees(angle_opposite_jib_rad)  
  
# =====  
# OUTPUT RESULTS  
# =====  
  
print(f"theta_p = {angle_opposite_post_deg:.4f} degrees")  
print(f"theta_t = {angle_opposite_tie_deg:.4f} degrees")  
print(f"theta_j = {angle_opposite_jib_deg:.4f} degrees")  
print()  
print(f"Force in jib (F_J): {force_jib_kn:.4f} kN")  
print(f"Force in tie (F_T): {force_tie_kn:.4f} kN")
```



## Chapter 2

# Dynamics

Dynamics is the branch of mechanics that studies the motion of bodies under the action of forces. This chapter examines situations where forces produce acceleration, and it presents methods for analyzing the resulting motion, including the application of Newton's laws, energy principles, and momentum conservation.

### 2.1 Inertia

Inertia is the property of a body that resists any change in its state of motion and is directly proportional to its mass. It can be understood as a form of “sluggishness” inherent to matter. For an object at rest, a net force is required to set it in motion; the greater the mass, the greater the force needed. Similarly, if an object is already moving, a net force is required to change its speed or direction, and the magnitude of this force is again proportional to the mass.

#### **i** Note

The tendency of an object to resist any change in its motion. An object with greater mass has greater inertia (it is “harder to start or stop”).

- In **Naval Architecture**, the *moment of inertia* (units:  $\text{m}^4$ ) is a measure of resistance to bending (related to stiffness/deflection of beams and hull girders).
- In **angular motion and physics**, the *moment of inertia* (units:  $\text{kg} \cdot \text{m}^2$ ) is a measure of resistance to angular acceleration (rotational inertia).

## 2.2 Momentum

Momentum is the product of an object's mass and its velocity and quantifies the amount of motion a moving body possesses. This concept is particularly important when considering a large vessel approaching a dock. Once the vessel is underway, its considerable momentum means that a substantial force, applied over sufficient time, is required to bring it to a stop or significantly alter its course. Without such intervention, the vessel will not stop quickly on its own and may collide with the dock.

## 2.3 Newton's Laws of Motion

### 1. First Law (Law of Inertia):

An object remains at rest, or in uniform motion in a straight line, unless acted upon by a net external force.

$$\Sigma F = 0 \implies v = \text{constant}$$

### 2. Second Law (Law of Acceleration):

The acceleration of an object is directly proportional to the net force acting on it and inversely proportional to its mass.

$$\vec{F} = m\vec{a}$$

### 3. Third Law (Action-Reaction Law):

For every action, there is an equal and opposite reaction.

$$\vec{F}_{\text{action}} = -\vec{F}_{\text{reaction}}$$

**Example 2.1.** A lift is supported by a steel wire rope, the total mass of the lift and contents is 750 kg. Find the tension in the wire rope, in newtons, when the lift is (i) moving at constant velocity, (ii) moving upwards and accelerating at  $1.2 \text{ m/s}^2$ , (iii) moving upwards and retarding at  $1.2 \text{ m/s}^2$

```
"""
Elevator Cable Tension Calculator

Calculates the tension in an elevator cable under different motion
conditions: constant velocity, upward acceleration, and upward
deceleration. Uses Newton's second law (F = ma).
"""

# =====
# INPUT PARAMETERS
# =====
```

```

elevator_mass_kg = 750 # Mass of elevator in kilograms
gravitational_acceleration_m_per_s2 = 9.81 # Acceleration due to gravity

# Accelerations for different motion conditions
acceleration_constant_velocity_m_per_s2 = 0 # No acceleration
acceleration_upward_m_per_s2 = 1.2 # Upward acceleration
acceleration_deceleration_m_per_s2 = -1.2 # Upward deceleration (negative)

# =====
# TENSION CALCULATIONS USING NEWTON'S SECOND LAW
# =====

# For an elevator, the cable tension is given by:
#  $T = m(g + a)$ 
# where:
# - m is the mass
# - g is gravitational acceleration
# - a is the net acceleration (positive upward, negative downward)

# Case 1: Constant velocity (a = 0)
# Cable tension equals the weight of the elevator
tension_constant_velocity_n = elevator_mass_kg * (
    gravitational_acceleration_m_per_s2 + acceleration_constant_velocity_m_per_s2
)

# Case 2: Upward acceleration (a > 0)
# Cable tension is greater than weight due to upward acceleration
tension_upward_acceleration_n = elevator_mass_kg * (
    gravitational_acceleration_m_per_s2 + acceleration_upward_m_per_s2
)

# Case 3: Upward deceleration (a < 0)
# Cable tension is less than weight due to deceleration
tension_upward_deceleration_n = elevator_mass_kg * (
    gravitational_acceleration_m_per_s2 + acceleration_deceleration_m_per_s2
)

# =====
# OUTPUT RESULTS
# =====

print(f"Tension at constant velocity: {tension_constant_velocity_n:.4f} N")
print(f"Tension during upward acceleration: " f"{tension_upward_acceleration_n:.4f} N")
print(f"Tension during upward deceleration: " f"{tension_upward_deceleration_n:.4f} N")

```

## 2.4 Linear momentum

Linear momentum is a fundamental concept in physics that quantifies the motion of an object. It is defined as the product of the object's mass and its velocity. Because momentum is a vector quantity, it possesses both magnitude and direction.

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

Where:

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

### 2.4.1 Conservation of Linear Momentum

The law of conservation of linear momentum states that, in a closed system with no external forces (or when the net external force is zero), the total linear momentum remains constant over time. This principle is particularly evident in collisions between two bodies. During a collision, the force exerted by the first body on the second is equal in magnitude and opposite in direction to the force exerted by the second on the first (Newton's third law). These equal and opposite forces act for the same duration, producing changes in momentum that are equal in magnitude but opposite in direction. Consequently, the momentum gained by one body exactly equals the momentum lost by the other. Therefore, the total momentum of the system before the collision is equal to the total momentum after the collision.

In the absence of external forces, linear momentum is neither created nor destroyed; it is conserved. This is known as the law of conservation of linear momentum.

**Example 2.2.** A jet of fresh water, 20 mm in diameter, emerges horizontally from a nozzle at a speed of 21 m/s and strikes a stationary vertical plate normally. Assuming no splashback (i.e., the water comes to rest upon impact with no velocity component normal to the plate after striking), calculate: (a) the mass flow rate of water leaving the nozzle (in kg/s), and (b) the force exerted by the jet on the plate. The density of fresh water is  $1000 \text{ kg/m}^3$ .

"""

Jet Impact Force Calculator

Calculates the force exerted by a water jet impacting a flat plate. Uses the momentum equation to determine the force based on mass flow rate and jet velocity.

```

"""

import math

# =====
# INPUT PARAMETERS
# =====

jet_diameter_m = 0.02 # Diameter of water jet in meters
jet_velocity_m_per_s = 21 # Velocity of water jet in m/s
water_density_kg_per_m3 = 1000 # Density of water in kg/m³

# =====
# CROSS-SECTIONAL AREA AND FLOW RATE CALCULATIONS
# =====

# Calculate cross-sectional area of the jet
jet_cross_sectional_area_m2 = math.pi * (jet_diameter_m**2) / 4

# Calculate volumetric flow rate using  $Q = A \times v$ 
volumetric_flow_rate_m3_per_s = jet_cross_sectional_area_m2 * jet_velocity_m_per_s

# Calculate mass flow rate using  $\dot{m} = \rho \times Q$ 
mass_flow_rate_kg_per_s = water_density_kg_per_m3 * volumetric_flow_rate_m3_per_s

# =====
# FORCE CALCULATION USING MOMENTUM EQUATION
# =====

# Calculate force on the plate using the momentum equation:
#  $F = \dot{m} \times \Delta v$ 
# For a jet hitting a flat plate normally and deflecting at 90°,
# the change in velocity in the normal direction equals the jet velocity:
#  $F = \dot{m} \times v$ 
force_on_plate_n = mass_flow_rate_kg_per_s * jet_velocity_m_per_s

# =====
# OUTPUT RESULTS
# =====

print(f"Mass flow rate (kg/s): {mass_flow_rate_kg_per_s:.4f}")
print(f"Force on plate (N): {force_on_plate_n:.4f}")

```

## 2.5 Angular Momentum

Angular momentum is the moment of linear momentum about a chosen point or axis. For a single particle, it is given by the cross product of its position vector (measured from that point) and its linear momentum.

More broadly, angular momentum measures the rotational motion of a system. For extended bodies, its evaluation depends on both the distance of mass elements from the axis (the first moment of position) and how the mass is distributed throughout the object—the moment of inertia, or second moment of mass.

Given **linear velocity**  $\vec{v}$  and **angular velocity**  $\omega$ :

$$\vec{v} = r\omega$$

The **linear momentum** is determined by:

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

Substituting for  $\vec{v}$ :

$$\text{Linear momentum} = mr\omega$$

The **moment of linear momentum** can then be written as:

$$\text{Moment of linear momentum} = mr\omega r$$

The moment of linear momentum is referred to as the **angular momentum**. Simplifying the above equation:

$$\text{Angular momentum} = mr^2\omega$$

Here,  $mr^2$  is the **moment of inertia** (second moment of mass) of the object about its axis of rotation, denoted as  $I$ . Thus, the angular momentum can be expressed as:

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

Where:

- $I$  is the moment of inertia in  $kg \cdot m^2$ .
- $\omega$  is the angular velocity in  $rad/s$ .

Additionally, the moment of inertia  $I$  is given by:

$$I = mk^2$$

Where:

- $I$  is the moment of inertia in  $kg \cdot m^2$ .
- $m$  is the mass in  $kg$ .
- $k$  is the radius of gyration in  $m$ .

#### **i** Note

In structural engineering, the moment of inertia ( $m$ ) is a measure of resistance to bending (deflection or stiffness).

In angular motion and physics, the moment of inertia ( $kg \cdot m^2$ ) is a measure of resistance to angular acceleration (rotational inertia).

## 2.6 The Radius of Gyration

A geometric property of a rigid body that indicates how the mass is distributed relative to a specified axis of rotation. It is defined such that:

$$I = mk^2$$

where

- $I$  = moment of inertia ( $kg \cdot m^2$ )
- $m$  = total mass ( $kg$ )
- $k$  = radius of gyration ( $m$ )

#### **i** Note

- **Smaller**  $k \rightarrow$  mass is concentrated closer to the axis  $\rightarrow$  **lower** rotational inertia  
 $\rightarrow$  easier/faster to speed up or slow down rotation  
 (e.g., engine crankshafts, turbine rotors, figure skater pulling arms in).
- **Larger**  $k \rightarrow$  mass is distributed farther from the axis  $\rightarrow$  **higher** rotational inertia  
 $\rightarrow$  better for storing rotational energy  
 (e.g., flywheels; a hollow cylinder has a larger  $k$  than a solid cylinder of the same mass and outer radius).

**Example 2.3.** A flywheel of mass 500 kg and radius of gyration 1.2 m is running at 300 rev/min. By means of a clutch, this flywheel is suddenly connected to

another flywheel, mass 2000 kg and radius of gyration 0.6 m, initially at rest. Calculate their common speed of rotation after engagement.

```

"""
Angular Momentum Conservation Calculator

Calculates the final angular velocity when two rotating bodies are coupled
together. Uses conservation of angular momentum to determine the combined
system's rotation speed.
"""

import math

# =====
# INPUT PARAMETERS
# =====

# Body 1 (initially rotating)
mass_body1_kg = 500 # Mass of body 1 in kilograms
radius_of_gyration_body1_m = 1.2 # Radius of gyration in meters
initial_angular_velocity_body1_rpm = 300 # Initial speed in rpm

# Body 2 (initially stationary)
mass_body2_kg = 2000 # Mass of body 2 in kilograms
radius_of_gyration_body2_m = 0.6 # Radius of gyration in meters

# =====
# MOMENT OF INERTIA CALCULATIONS
# =====

# Calculate moment of inertia for body 1 using  $I = m \times k^2$ 
# where k is the radius of gyration
moment_of_inertia_body1_kg_m2 = mass_body1_kg * radius_of_gyration_body1_m**2

# Calculate moment of inertia for body 2
moment_of_inertia_body2_kg_m2 = mass_body2_kg * radius_of_gyration_body2_m**2

# =====
# UNIT CONVERSION
# =====

# Convert initial angular velocity from rpm to rad/s
# (rad/s) = (rpm / 60) * 2
initial_angular_velocity_body1_rad_per_s = (
    (initial_angular_velocity_body1_rpm / 60) * 2 * math.pi
)

```

```

# =====
# ANGULAR MOMENTUM CONSERVATION
# =====

# Apply conservation of angular momentum:
# L_initial = L_final
# I  = (I  + I ) _f
# Solving for final angular velocity:
# _f = (I  ) / (I  + I )

final_angular_velocity_rad_per_s = (
    moment_of_inertia_body1_kg_m2 * initial_angular_velocity_body1_rad_per_s
) / (moment_of_inertia_body1_kg_m2 + moment_of_inertia_body2_kg_m2)

# =====
# UNIT CONVERSION FOR OUTPUT
# =====

# Convert final angular velocity from rad/s to rpm
# rpm = ( / 2 ) × 60
final_angular_velocity_rpm = (final_angular_velocity_rad_per_s / (2 * math.pi)) * 60

# =====
# OUTPUT RESULTS
# =====

print(f"Final angular velocity (rad/s): " f"{final_angular_velocity_rad_per_s:.4f}")
print(f"Final angular velocity (rpm): {final_angular_velocity_rpm:.4f}")

```

## 2.7 Turning Moment

We know the relation between linear acceleration  $a$  and angular acceleration  $\alpha$ :

$$a = r\alpha$$

And

$$F = ma$$

Therefore,

$$F = m\alpha r$$

If the force is not applied directly on the rim but at a greater or lesser leverage, say  $L$  from the centre, the effective force on the rim causing it to accelerate will be greater or lesser accordingly, in the ratio of  $L$  to  $r$ . Thus:

$$F \frac{L}{r} = m\alpha r$$

Multiplying both sides by  $r$ ,

$$FL = m\alpha r^2$$

Now,  $FL$  is the torque applied. Therefore, the accelerating torque is:

$$\tau = mr^2\alpha$$

Or,

$$\tau = mk^2\alpha$$

$\tau$  is also given by:

$$\tau = I\alpha$$

Where:

- $\tau$  is the torque in  $Nm$ .
- $I$  is the moment of inertia in  $kg \cdot m^2$ .
- $\alpha$ : Angular acceleration in  $rad/s^2$ .

**Example 2.4.** The torque to overcome frictional and other resistances of a turbine is 317 N m and may be considered constant for all speeds. The mass of the rotating parts is 1.59 t and the radius of gyration is 0.686 m. If the gas is cut off when the turbine is running free of load at 1920 rev/min, find the time it will take to come to rest and the number of revolutions turned during that time.

```
"""
Rotational Deceleration Calculator

Calculates the time and number of revolutions required to bring a
rotating body to rest under a constant braking torque. Uses rotational
kinematics and dynamics equations.
"""

import math
```

```

# =====
# INPUT PARAMETERS
# =====

braking_torque_n_m = 317.0 # Braking torque in Newton-meters
rotating_mass_kg = 1.59 * 1000 # Mass of rotating body in kilograms
radius_of_gyration_m = 0.686 # Radius of gyration in meters
initialAngular_velocity_rpm = 1920.0 # Initial rotational speed in rpm

# =====
# MOMENT OF INERTIA CALCULATION
# =====

# Calculate moment of inertia using  $I = m \times k^2$ 
# where k is the radius of gyration
moment_of_inertia_kg_m2 = rotating_mass_kg * radius_of_gyration_m**2

# =====
# UNIT CONVERSION
# =====

# Convert initial angular velocity from rpm to rad/s
# (rad/s) = (rpm / 60) * 2
initialAngular_velocity_rad_per_s = (
    (initialAngular_velocity_rpm / 60.0) * 2.0 * math.pi
)

# =====
# ANGULAR DECELERATION CALCULATION
# =====

# Calculate angular deceleration using  $\alpha = I \times$ 
# Solving for  $\alpha = \tau / I$ 
# (negative because it's deceleration, but using magnitude here)
angularDeceleration_rad_per_s2 = braking_torque_n_m / moment_of_inertia_kg_m2

# =====
# TIME TO STOP CALCULATION
# =====

# Calculate time to stop using  $\omega_f = \omega_0 - \alpha t$ 
# With  $\omega_f = 0$  (comes to rest):  $t = \omega_0 / \alpha$ 
time_to_stop_s = initialAngular_velocity_rad_per_s / angularDeceleration_rad_per_s2

# Convert time to minutes

```

```

time_to_stop_min = time_to_stop_s / 60.0

# =====
# ANGULAR DISPLACEMENT CALCULATION
# =====

# Calculate total angular displacement using  $\theta = \omega_0 t - \frac{1}{2} \alpha t^2$ 
# Or equivalently:  $\theta = \frac{1}{2} \omega_0 t$  (when final velocity is zero)
angular_displacement_rad = 0.5 * initial_angular_velocity_rad_per_s * time_to_stop_s

# Convert angular displacement from radians to revolutions
number_of_revolutions = angular_displacement_rad / (2.0 * math.pi)

# =====
# OUTPUT RESULTS
# =====

print(f"I (kg·m²):           {moment_of_inertia_kg_m2:.4f}")
print(f"omega0 (rad/s):       {initial_angular_velocity_rad_per_s:.4f}")
print(f"alpha (rad/s²):          {angular_deceleration_rad_per_s2:.6f}")
print(f"time to stop (s):        {time_to_stop_s:.4f}")
print(f"time to stop (min):      {time_to_stop_min:.4f}")
print(f"revolutions:              {number_of_revolutions:.4f}")

```

## 2.8 Power by Torque

Consider a force  $F$  applied at a radius  $r$  on a rotating mechanism. The work done in one revolution is the product of the force and the circumference. Therefore:

$$W = F \cdot 2\pi r$$

If the mechanism is running at  $n$  revolutions per second:

$$\text{Power} = F \cdot 2\pi r n$$

Since torque  $\tau = Fr$ , we can rewrite the equation as:

$$P = \tau \cdot 2\pi n$$

Given that  $\omega = 2\pi n$ :

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

Where:

- $P$  is the power in watts (W),
- $\tau$  is the torque in newton-meters (Nm), and
- $\omega$  is the angular velocity in radians per second (rad/s).

**Example 2.5.** A vessel has a displacement of 1000 tonnes. It is being pulled up the horizontal dry dock by a steel cable, which is being wound onto a power-driven capstan. The capstan drum has a mass of 4.5 tonnes, an effective diameter of 3.1 m, and a radius of gyration of 1 m. The coefficient of friction between the vessel and the dock can be assumed constant at 0.6. The ship velocity increases uniformly from 0.1 m/s to 0.3 m/s in 30 seconds during the operation. Calculate:

- The angular acceleration of the capstan drum.
- The torque required at the capstan drum.

```

"""
Ship Mooring Drum Torque Calculator

Calculates the torque required at a mooring drum to accelerate a ship
from initial to final velocity. Accounts for friction forces, ship
acceleration, and drum rotational inertia.
"""

# =====
# INPUT PARAMETERS
# =====

ship_mass_tonnes = 1000 # Mass of ship in tonnes
drum_mass_tonnes = 4.5 # Mass of drum in tonnes
drum_diameter_m = 3.1 # Diameter of drum in meters
drum_radius_of_gyration_m = 1.0 # Radius of gyration in meters
friction_coefficient = 0.6 # Coefficient of friction (dimensionless)
gravitational_acceleration_m_per_s2 = 9.81 # Acceleration due to gravity
ship_initial_velocity_m_per_s = 0.1 # Initial velocity in m/s
ship_final_velocity_m_per_s = 0.3 # Final velocity in m/s
acceleration_time_s = 30 # Time duration in seconds

# =====
# UNIT CONVERSIONS
# =====

# Convert masses from tonnes to kilograms
ship_mass_kg = ship_mass_tonnes * 1000
drum_mass_kg = drum_mass_tonnes * 1000

```

```

# Calculate effective radius of drum (rope wraps at outer surface)
drum_effective_radius_m = drum_diameter_m / 2

# =====
# ACCELERATION CALCULATIONS
# =====

# Calculate linear acceleration of the ship using  $a = \Delta v / \Delta t$ 
ship_linear_acceleration_m_per_s2 = (
    ship_final_velocity_m_per_s - ship_initial_velocity_m_per_s
) / acceleration_time_s

# Calculate angular acceleration of the drum using  $\alpha = a / r$ 
drum_angular_acceleration_rad_per_s2 = (
    ship_linear_acceleration_m_per_s2 / drum_effective_radius_m
)

# =====
# MOMENT OF INERTIA CALCULATION
# =====

# Calculate drum moment of inertia using  $I = m \times k^2$ 
drum_moment_of_inertia_kg_m2 = drum_mass_kg * drum_radius_of_gyration_m**2

# =====
# FORCE CALCULATIONS
# =====

# Calculate friction force between ship and water:  $F_f = \mu \times m \times g$ 
friction_force_n = (
    friction_coefficient * ship_mass_kg * gravitational_acceleration_m_per_s2
)

# Calculate force required to accelerate the ship:  $F = m \times a$ 
ship_acceleration_force_n = ship_mass_kg * ship_linear_acceleration_m_per_s2

# =====
# TORQUE CALCULATIONS
# =====

# Torque required to overcome friction:  $T_f = F_f \times r$ 
torque_friction_n_m = friction_force_n * drum_effective_radius_m

# Torque required to accelerate the ship linearly:  $T_s = F_s \times r$ 
torque_ship_acceleration_n_m = ship_acceleration_force_n * drum_effective_radius_m

```

```

# Torque required to accelerate the drum rotationally: T_d = I ×
torque_drum_acceleration_n_m = (
    drum_moment_of_inertia_kg_m2 * drum_angular_acceleration_rad_per_s2
)

# Total torque required at the drum
total_torque_n_m = (
    torque_friction_n_m + torque_ship_acceleration_n_m + torque_drum_acceleration_n_m
)

# Convert total torque to meganewton-meters for output
total_torque_mn_m = total_torque_n_m / 1e6

# =====
# OUTPUT RESULTS
# =====

print(
    f"Angular acceleration of drum (rad/s²): "
    f"{drum_angular_acceleration_rad_per_s2:.4f}"
)
print(f"Torque required at the drum (MN·m): {total_torque_mn_m:.4f}")

```

## 2.9 Kinetic Energy of Rotation

We know that  $K.E. = \frac{1}{2}mv^2$ , where  $v$  is the linear velocity of the body. For a rotating body, the effective linear velocity is at the radius of gyration, as this is the radius at which the entire mass of the rotating body can be considered to act.

Let  $k$  = radius of gyration (in meters),  
 Let  $\omega$  = angular velocity (in radians per second).

The relationship between linear and angular velocity is:

$$v = \omega k$$

Substituting  $v^2 = \omega^2 k^2$  into  $K.E. = \frac{1}{2}mv^2$  gives:

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

Since  $I = mk^2$ , where  $I$  is the moment of inertia:

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

**Example 2.6.** The radius of gyration of the flywheel of a shearing machine is 0.46 m and its mass is 750 kg. Find the kinetic energy stored in it when running at 120 rev/min. If the speed falls to 100 rev/min during the cutting stroke, find the kinetic energy given out by the wheel.

```

"""
Rotational Kinetic Energy Calculator

Calculates the rotational kinetic energy of a rotating body at different
angular velocities and determines the energy released during deceleration.
"""

import math

# =====
# INPUT PARAMETERS
# =====

rotating_mass_kg = 750 # Mass of rotating body in kilograms
radius_of_gyration_m = 0.46 # Radius of gyration in meters
initialAngular_velocity_rpm = 120 # Initial rotational speed in rpm
finalAngular_velocity_rpm = 100 # Final rotational speed in rpm

# =====
# MOMENT OF INERTIA CALCULATION
# =====

# Calculate moment of inertia using I = m × k2
# where k is the radius of gyration
moment_of_inertia_kg_m2 = rotating_mass_kg * radius_of_gyration_m**2

# =====
# UNIT CONVERSIONS
# =====

# Convert initial angular velocity from rpm to rad/s
# (rad/s) = (rpm / 60) × 2
initialAngular_velocity_rad_per_s = initialAngular_velocity_rpm * 2 * math.pi / 60

# Convert final angular velocity from rpm to rad/s
finalAngular_velocity_rad_per_s = finalAngular_velocity_rpm * 2 * math.pi / 60

# =====

```

```

# KINETIC ENERGY CALCULATIONS
# =====

# Calculate initial rotational kinetic energy using  $KE = \frac{1}{2}I \omega^2$ 
initial_kinetic_energy_j = (
    0.5 * moment_of_inertia_kg_m2 * initial_angular_velocity_rad_per_s**2
)

# Calculate final rotational kinetic energy
final_kinetic_energy_j = (
    0.5 * moment_of_inertia_kg_m2 * final_angular_velocity_rad_per_s**2
)

# Calculate energy released (change in kinetic energy)
energy_released_j = initial_kinetic_energy_j - final_kinetic_energy_j

# Convert energies from Joules to kilojoules
initial_kinetic_energy_kj = initial_kinetic_energy_j / 1e3
final_kinetic_energy_kj = final_kinetic_energy_j / 1e3
energy_released_kj = energy_released_j / 1e3

# =====
# OUTPUT RESULTS
# =====

print(f"I: {moment_of_inertia_kg_m2:.4f} kg·m2")
print(f"KE at 120 rpm: {initial_kinetic_energy_kj:.4f} kJ")
print(f"KE at 100 rpm: {final_kinetic_energy_kj:.4f} kJ")
print(f"Energy given out: {energy_released_kj:.4f} kJ")

```



## Chapter 3

# Hydrostatics

Hydrostatics is the branch of fluid mechanics that studies fluids at rest and the forces and pressures they exert on immersed or containing surfaces. Key principles include Pascal's law (pressure applied to an enclosed fluid is transmitted undiminished in all directions), Archimedes' principle (buoyant force on a submerged body equals the weight of displaced fluid), and the hydrostatic pressure distribution in a fluid column, expressed as  $p = \rho gh$ , where  $p$  is gauge pressure,  $\rho$  is fluid density,  $g$  is gravitational acceleration, and  $h$  is depth below the free surface.

### 3.1 Pressure Head

In fluid mechanics, pressure head is the height of a fluid column that corresponds to a specific pressure at a given point, assuming the fluid is static. It's expressed as  $h_p = \frac{p}{\rho g}$ , where  $p$  is the pressure above atmospheric,  $\rho$  is the fluid density, and  $g$  is the acceleration due to gravity. The unit is typically meters (or feet) of the fluid column. Pressure head represents the potential energy per unit weight due to pressure and is a component of the total hydraulic head, which is the sum of pressure head and elevation head.

### 3.2 Load On Immersed Surfaces

The load on an immersed surface is the total hydrostatic force exerted by a fluid on a surface submerged in a static fluid. This force arises from the pressure variation with depth, governed by the hydrostatic pressure equation  $p = \rho gh$ , where  $\rho$  is fluid density,  $g$  is gravitational acceleration, and  $h$  is the depth below the free surface.

The pressure acts perpendicular to the surface, resulting in a distributed load.

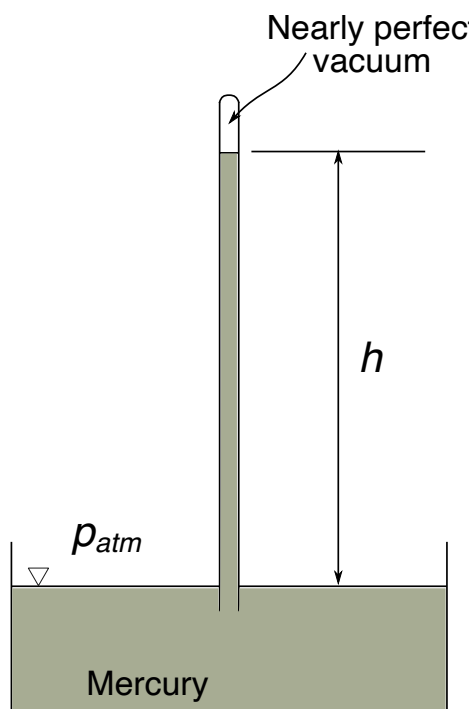


Figure 3.1: Barometer

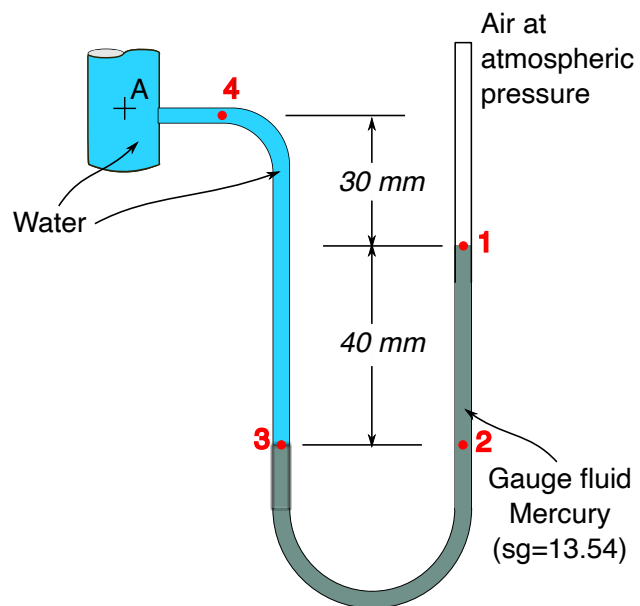


Figure 3.2: U-tube manometer

The total force  $F$  is calculated as  $F = \rho g A \bar{h}$ , where  $A$  is the surface area and  $\bar{h}$  is the depth of the centroid of the area.

**Example 3.1.** A vertical rectangular bulkhead is 7 m wide and extends over the full height of the water column. Fresh water stands 6 m deep on one side and is assumed to be at zero level on the other side. Calculate the total hydrostatic load (thrust) on the bulkhead. Density of fresh water = 1000 kg/m<sup>3</sup>.

```

"""
Hydrostatic Load Calculator

Calculates the total hydrostatic force on a vertical rectangular bulkhead
submerged in water. The force acts at the centroid of the pressure
distribution.
"""

# =====
# INPUT PARAMETERS
# =====

bulkhead_width_m = 7.0 # Horizontal width of bulkhead in meters
water_depth_m = 6.0 # Depth of water on one side in meters
water_density_kg_per_m3 = 1000.0 # Density of fresh water (kg/m³)
gravitational_acceleration_m_per_s2 = 9.81 # Acceleration due to gravity

# =====
# CALCULATIONS
# =====

# Calculate submerged area of bulkhead
submerged_area_m2 = bulkhead_width_m * water_depth_m

# Calculate depth to centroid of pressure distribution
# (for rectangular area, centroid is at half the water depth)
centroid_depth_m = water_depth_m / 2

# Calculate total hydrostatic force using:
# F = ρ · g · h_c · A
# where ρ = water density, g = gravity, h_c = centroid depth, A = area
hydrostatic_force_n = (
    water_density_kg_per_m3
    * gravitational_acceleration_m_per_s2
    * centroid_depth_m
    * submerged_area_m2
)

```

```

# Convert force from Newtons to kilonewtons
hydrostatic_force_kn = hydrostatic_force_n / 1000

# Convert force from kilonewtons to meganewtons for additional output
hydrostatic_force_mn = hydrostatic_force_kn / 1000

# =====
# OUTPUT RESULTS
# =====

print("=== Hydrostatic Load on Bulkhead ===")
print(f"Bulkhead width      : {bulkhead_width_m} m")
print(f"Water depth          : {water_depth_m} m")
print(f"Submerged area          : {submerged_area_m2:.1f} m²")
print(f"Depth of centroid       : {centroid_depth_m} m")
print(
    f"Total water load       : {hydrostatic_force_kn:,.0f} kN  "
    f"({hydrostatic_force_mn:.3f} MN)"
)

```

**Example 3.2.** A 10-meter-long, 4-meter-wide, and 6-meter-high tank is filled with oil (specific gravity = 0.9). The oil rises 5 meters up a vent pipe above the tank's top. Calculate the load on one end plate and the bottom of the tank.

```

"""
Oil Tank Hydrostatic Load Calculator

Calculates hydrostatic forces on the end plates and bottom plate of a
rectangular oil tank with a sounding pipe. The oil rises above the tank
top in the sounding pipe, increasing the pressure on all surfaces.
"""

# =====
# INPUT PARAMETERS
# =====

tank_length_m = 10.0 # Length of tank in meters
tank_width_m = 4.0 # Width of tank in meters
tank_height_m = 6.0 # Height of tank in meters
sounding_pipe_height_m = 5.0 # Oil rise above tank top in meters
oil_specific_gravity = 0.9 # Specific gravity of oil (unitless)
oil_density_kg_per_m3 = oil_specific_gravity * 1000 # Density in kg/m³
gravitational_acceleration_m_per_s2 = 9.81 # Acceleration due to gravity

# =====
# CALCULATED VALUES

```

```

# =====

# Total head from tank bottom to oil surface in sounding pipe
total_head_m = tank_height_m + sounding_pipe_height_m

# End plate dimensions (vertical wall at short end of tank)
end_plate_height_m = tank_height_m
end_plate_width_m = tank_width_m
end_plate_area_m2 = end_plate_height_m * end_plate_width_m

# Depth of centroid of end plate below oil surface
# (centroid is at mid-height of plate, plus sounding pipe height)
end_plate_centroid_depth_m = sounding_pipe_height_m + (tank_height_m / 2)

# Hydrostatic force on one end plate using  $F = \rho \cdot g \cdot h_c \cdot A$ 
force_end_plate_n = (
    oil_density_kg_per_m3
    * gravitational_acceleration_m_per_s2
    * end_plate_centroid_depth_m
    * end_plate_area_m2
)

# Bottom plate dimensions and force
bottom_plate_area_m2 = tank_length_m * tank_width_m

# Hydrostatic force on bottom plate using  $F = \rho \cdot g \cdot h \cdot A$ 
# (pressure at bottom is based on total head)
force_bottom_plate_n = (
    oil_density_kg_per_m3
    * gravitational_acceleration_m_per_s2
    * total_head_m
    * bottom_plate_area_m2
)

# Convert forces from Newtons to kilonewtons
force_end_plate_kn = force_end_plate_n / 1000
force_bottom_plate_kn = force_bottom_plate_n / 1000

# =====
# OUTPUT RESULTS
# =====

print("=== Hydrostatic Loads on Oil Tank ===")
print(
    f"Tank dimensions      : {tank_length_m} m × {tank_width_m} m × "

```

```

    f"{tank_height_m} m (L×W×H)"
)
print(f"Oil rise in sounding pipe : {sounding_pipe_height_m} m")
print(f"Oil density          : {oil_density_kg_per_m3} kg/m³")
print()
print(
    f"End plate (one wall)  : {end_plate_height_m} m high × "
    f"{end_plate_width_m} m wide = {end_plate_area_m2} m²"
)
print(
    f"Horizontal load on one end plate : {force_end_plate_n:,.0f} N = "
    f"{force_end_plate_kn:,.0f} kN"
)
print()
print(
    f"Bottom plate          : {tank_length_m} m × {tank_width_m} m = "
    f"{bottom_plate_area_m2} m²"
)
print(
    f"Vertical load on bottom          : {force_bottom_plate_n:,.0f} N = "
    f"{force_bottom_plate_kn:,.0f} kN"
)

```

**Example 3.3.** A rectangular dock gate measures 4 meters wide. If the water level is 5.5 meters high on one side and 3.5 meters high on the other, what horizontal thrust will act on the gate? Assume the water density is 1000 kg/m<sup>3</sup>.

```

"""
Hydrostatic Thrust Calculator for Gate

Calculates the net horizontal thrust on a vertical gate separating two
different water levels. The net thrust is the difference between forces
from the high and low sides.
"""

# =====
# INPUT PARAMETERS
# =====

gate_width_m = 4.0 # Width of gate in meters
water_depth_high_side_m = 5.5 # Water depth on higher side in meters
water_depth_low_side_m = 3.5 # Water depth on lower side in meters
water_density_kg_per_m3 = 1000.0 # Density of water (kg/m³)
gravitational_acceleration_m_per_s2 = 9.81 # Acceleration due to gravity

# =====

```

```

# CALCULATIONS - HIGH SIDE
# =====

# Area of gate submerged on high side
area_high_side_m2 = water_depth_high_side_m * gate_width_m

# Depth to centroid of pressure distribution on high side
# (for rectangular area, centroid is at half the water depth)
centroid_depth_high_side_m = water_depth_high_side_m / 2

# Hydrostatic force from high side using  $F = \rho \cdot g \cdot h_c \cdot A$ 
force_high_side_n = (
    water_density_kg_per_m3
    * gravitational_acceleration_m_per_s2
    * centroid_depth_high_side_m
    * area_high_side_m2
)

# =====
# CALCULATIONS - LOW SIDE
# =====

# Area of gate submerged on low side
area_low_side_m2 = water_depth_low_side_m * gate_width_m

# Depth to centroid of pressure distribution on low side
centroid_depth_low_side_m = water_depth_low_side_m / 2

# Hydrostatic force from low side using  $F = \rho \cdot g \cdot h_c \cdot A$ 
force_low_side_n = (
    water_density_kg_per_m3
    * gravitational_acceleration_m_per_s2
    * centroid_depth_low_side_m
    * area_low_side_m2
)

# =====
# NET THRUST CALCULATION
# =====

# Net horizontal thrust (force from high side minus force from low side)
net_thrust_n = force_high_side_n - force_low_side_n

# =====
# OUTPUT RESULTS

```

```
# =====

print("=== Hydrostatic Thrust Calculation ===")
print(
    f"Force from higher side ({water_depth_high_side_m} m): "
    f"{force_high_side_n:.0f} N"
)
print(
    f"Force from lower side ({water_depth_low_side_m} m): " f"{force_low_side_n:.0f} N"
)
print(f"Net horizontal thrust on gate: {net_thrust_n:.0f} N")
```

**Example 3.4.** A rectangular dock gate measures 12 meters wide. When the sea is 9 meters deep on one side and 4.5 meters deep on the other, what horizontal thrust will act on the gate? Assume the density of seawater is 1025 kg/m<sup>3</sup>.

```
"""
Dock Gate Hydrostatic Thrust Calculator

Calculates the net horizontal thrust on a vertical dock gate separating
two different water levels. Accounts for seawater density and provides
results in multiple units (N, kN, MN).
"""

# =====
# INPUT PARAMETERS
# =====

gate_width_m = 12.0 # Width of dock gate in meters
water_depth_deep_side_m = 9.0 # Water depth on deeper side in meters
water_depth_shallow_side_m = 4.5 # Water depth on shallower side in meters
seawater_density_kg_per_m3 = 1025.0 # Density of seawater (kg/m³)
gravitational_acceleration_m_per_s2 = 9.81 # Acceleration due to gravity

# =====
# CALCULATIONS - DEEPER SIDE
# =====

# Area of gate submerged on deeper side
area_deep_side_m2 = water_depth_deep_side_m * gate_width_m

# Depth to centroid of pressure distribution on deeper side
# (for rectangular area, centroid is at half the water depth)
centroid_depth_deep_side_m = water_depth_deep_side_m / 2

# Hydrostatic force from deeper side using  $F = \rho \cdot g \cdot h_c \cdot A$ 
```

```

force_deep_side_n = (
    seawater_density_kg_per_m3
    * gravitational_acceleration_m_per_s2
    * centroid_depth_deep_side_m
    * area_deep_side_m2
)

# =====
# CALCULATIONS - SHALLOWER SIDE
# =====

# Area of gate submerged on shallower side
area_shallow_side_m2 = water_depth_shallow_side_m * gate_width_m

# Depth to centroid of pressure distribution on shallower side
centroid_depth_shallow_side_m = water_depth_shallow_side_m / 2

# Hydrostatic force from shallower side using  $F = \rho \cdot g \cdot h_c \cdot A$ 
force_shallow_side_n = (
    seawater_density_kg_per_m3
    * gravitational_acceleration_m_per_s2
    * centroid_depth_shallow_side_m
    * area_shallow_side_m2
)

# =====
# NET THRUST CALCULATION
# =====

# Net horizontal thrust (force from deep side minus force from shallow side)
net_thrust_n = force_deep_side_n - force_shallow_side_n

# Convert forces to additional units for output
force_deep_side_mn = force_deep_side_n / 1e6
force_shallow_side_mn = force_shallow_side_n / 1e6
net_thrust_kn = net_thrust_n / 1000
net_thrust_mn = net_thrust_n / 1e6

# =====
# OUTPUT RESULTS
# =====

print("=== Dock Gate Hydrostatic Thrust Calculation ===")
print(
    f"Force from {water_depth_deep_side_m} m side : "

```

```

    f"{force_deep_side_n:,.0f} N  ({force_deep_side_mn:.3f} MN)"
)
print(
    f"Force from {water_depth_shallow_side_m} m side : "
    f"{force_shallow_side_n:,.0f} N  ({force_shallow_side_mn:.3f} MN)"
)
print(
    f"Resultant horizontal thrust: {net_thrust_n:,.0f} N  "
    f"({net_thrust_kn:.1f} kN or {net_thrust_mn:.3f} MN)"
)

```

**Example 3.5.** Seawater has flooded an oil tanker's oil tank to a depth of 4 meters, and a 10-meter layer of oil sits atop the seawater. Calculate the pressure at the bottom of the tank. Note: The density of oil is 0.85 g/cm<sup>3</sup>, the density of seawater is 1.02 t/m<sup>3</sup>, and the atmospheric pressure is 101.3 kPa.

```

"""
Layered Fluid Pressure Calculator

Calculates the hydrostatic pressure at the bottom of a tank containing
two immiscible fluid layers (oil on top of seawater). Computes both
gauge pressure and absolute pressure.
"""

# =====
# INPUT PARAMETERS
# =====

oil_density_g_per_cm3 = 0.85 # Density of oil in g/cm3
oil_layer_height_m = 10.0 # Height of oil layer in meters
seawater_density_t_per_m3 = 1.02 # Density of seawater in t/m3
seawater_layer_height_m = 4.0 # Height of seawater layer in meters
atmospheric_pressure_kpa = 101.3 # Atmospheric pressure in kPa
gravitational_acceleration_m_per_s2 = 9.81 # Acceleration due to gravity

# =====
# UNIT CONVERSIONS
# =====

# Convert oil density from g/cm3 to kg/m3
oil_density_kg_per_m3 = oil_density_g_per_cm3 * 1000

# Convert seawater density from t/m3 to kg/m3
seawater_density_kg_per_m3 = seawater_density_t_per_m3 * 1000

# =====

```

```

# HYDROSTATIC PRESSURE CALCULATIONS
# =====

# Pressure contribution from oil layer using  $P = \rho \cdot g \cdot h$ 
pressure_from_oil_kpa = (
    oil_density_kg_per_m3 * gravitational_acceleration_m_per_s2 * oil_layer_height_m
) / 1000 # Convert Pa to kPa

# Pressure contribution from seawater layer using  $P = \rho \cdot g \cdot h$ 
pressure_from_seawater_kpa = (
    seawater_density_kg_per_m3
    * gravitational_acceleration_m_per_s2
    * seawater_layer_height_m
) / 1000 # Convert Pa to kPa

# Total gauge pressure (pressure above atmospheric)
total_gauge_pressure_kpa = pressure_from_oil_kpa + pressure_from_seawater_kpa

# Absolute pressure (gauge pressure plus atmospheric pressure)
absolute_pressure_kpa = atmospheric_pressure_kpa + total_gauge_pressure_kpa

# =====
# OUTPUT RESULTS
# =====

print(f"Pressure due to oil layer          : {pressure_from_oil_kpa:.4f} kPa")
print(f"Pressure due to seawater layer     : " f"{pressure_from_seawater_kpa:.4f} kPa")
print(f"Total gauge pressure                 : {total_gauge_pressure_kpa:.4f} kPa")
print(f"Absolute pressure at the bottom    : {absolute_pressure_kpa:.4f} kPa")

```

### 3.3 Hydraulic Jacks

Hydraulic jacks use Pascal's law to lift heavy loads with minimal input force. Pascal's law states that pressure applied to an enclosed fluid is transmitted equally in all directions. In a hydraulic jack as shown in Figure 3.3, two pistons of different sizes are connected by a confined fluid. Applying an effort force to the smaller piston generates a pressure that is transmitted to the larger piston, producing an output force. Calculations involving hydraulic jacks, such as determining system pressure, required input force, or lifted load, rely on this principle under ideal conditions, neglecting friction and fluid compressibility.

**Example 3.6.** A force of 150 N is transmitted from a piston of 25 mm<sup>2</sup> to one of 100 mm<sup>2</sup>. Determine the system pressure and the load carried by the larger piston.

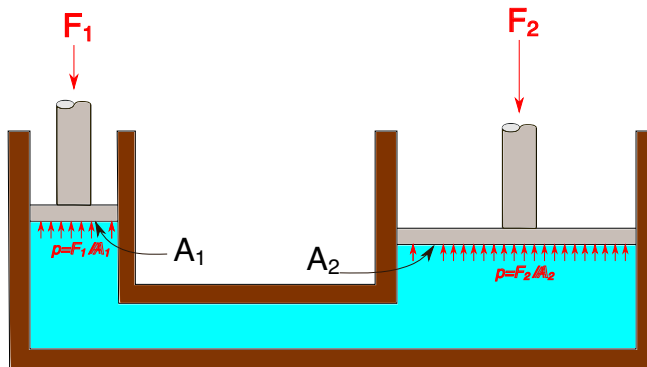


Figure 3.3: Hydraulic lift

```

"""
Hydraulic Press Calculator

Calculates the load capacity of a hydraulic press using Pascal's principle.
Given the force applied to a small piston and the areas of both pistons,
determines the system pressure and the load that can be lifted by the
larger piston.
"""

# =====
# INPUT PARAMETERS
# =====

small_piston_area_mm2 = 25.0 # Area of small piston in mm²
large_piston_area_mm2 = 100.0 # Area of large piston in mm²
force_on_small_piston_n = 150.0 # Force applied to small piston in N

# =====
# UNIT CONVERSIONS
# =====

# Convert piston areas from mm² to m² (1 mm² = 10⁻⁶ m²)
small_piston_area_m2 = small_piston_area_mm2 * 1e-6
large_piston_area_m2 = large_piston_area_mm2 * 1e-6

# =====
# HYDRAULIC SYSTEM CALCULATIONS
# =====

```

```
# Calculate system pressure using  $P = F / A$  (Pascal's principle)
# Pressure is uniform throughout the hydraulic fluid
system_pressure_pa = force_on_small_piston_n / small_piston_area_m2

# Calculate load carried by large piston using  $F = P \cdot A$ 
load_on_large_piston_n = system_pressure_pa * large_piston_area_m2

# Convert pressure to MPa for additional output
system_pressure_mpa = system_pressure_pa / 1e6

# =====
# OUTPUT RESULTS
# =====

print(
    f"System pressure: {system_pressure_pa:.2f} Pa "
    f"(or {system_pressure_mpa:.1f} MPa)"
)
print(f"Load carried by larger piston: {load_on_large_piston_n:.1f} N")
```

## Chapter 4

# Hydrodynamics

Hydrodynamics, a branch of fluid mechanics, studies the motion of liquids and the forces acting on them. Unlike aerodynamics, which concerns compressible gases such as air, hydrodynamics primarily deals with incompressible fluids like water. It is based on classical physics principles, including Newton's laws of motion, the conservation of mass and momentum. These principles are used to describe the velocity, pressure and viscosity of a fluid in motion.

### 4.1 Volume Flow Rate

Volume rate of flow refers to the rate at which a fluid volume passes a given section in a flow stream. Volume rate of flow may also be referred to as capacity of flow, flow rate or discharge. It is generally given in units of volume per unit of time, cubic meters per second ( $\text{m}^3/\text{s}$ ) or liters per second (L/s).

Consider an ideal fluid flow in a pipe of cross-section 'A' ( $\text{m}^2$ ). For an ideal fluid all the particles move to the right with a velocity of 'C' (m/s). As the fluid flow to the right with a velocity of C m/s, 'C' m of fluid pass section 1 every second. i.e. the volume of fluid passing section-1 every second is A x C.

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

Where:

- $\dot{v}$ : Volume flow rate,  $\text{m}^3/\text{s}$
- A: Cross-sectional area of the flow,  $\text{m}^2$
- C: Mean (average) velocity of the fluid, m/s

Consider a real fluid flowing in the same pipe. Here particle velocity will be a variable across the flow stream.

If the mean velocity  $\bar{c}_m$  of all the fluid particles could be found, then similar to the ideal fluid.

$$\text{Volume flow} = \text{area} \times \text{mean velocity}$$

Unless otherwise indicated, the velocity of flow is understood to refer to the average or mean velocity of all the particles in a flowing fluid.

## 4.2 Mass Flow Rate

Mass flow rate refers to the rate at which a fluid mass passes a given section in a flow stream. It is given in units of mass per unit of time, kilogram per second (kg/s). Mass flow rate is easily calculated from volume flow rate as follows.

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

Where:

- $\dot{m}$ : mass flow rate, kg/s
- $\rho$ : density, kg/m<sup>3</sup>
- $\dot{v}$ : volume flow, m<sup>3</sup>/s

Or

$$\dot{m} = \rho \cdot A \cdot C$$

Where:

- $\dot{m}$ : mass flow rate, kg/s
- $\rho$ : density, kg/m<sup>3</sup>
- $A$ : Cross-sectional area of the flow, m<sup>2</sup>
- $C$ : Mean (average) velocity of the fluid, m/s

**Example 4.1.** Oil of relative density 0.9 flows at full bore through a pipe with an internal diameter of 75 mm at a velocity of 1.2 m/s. Calculate the volume flow rate in cubic metres per second and the mass flow rate in tonnes per hour.

"""

Pipe Flow Rate Calculator for Oil

Determines the volumetric and mass flow rates in a pipe carrying oil, given the internal diameter, fluid velocity, and relative density.

"""

```

import math

# =====
# INPUT PARAMETERS
# =====

pipe_internal_diameter_mm = 75.0 # Pipe internal diameter in millimeters
oil_velocity_m_per_s = 1.2 # Average velocity of oil in m/s
oil_relative_density = 0.9 # Specific gravity of oil (relative to water)
water_density_kg_per_m3 = 1000.0 # Density of water in kg/m³

# =====
# UNIT CONVERSIONS AND GEOMETRY
# =====

# Convert pipe diameter from millimeters to meters
pipe_internal_diameter_m = pipe_internal_diameter_mm / 1000.0

# Calculate pipe radius
pipe_radius_m = pipe_internal_diameter_m / 2.0

# Calculate cross-sectional area of the pipe (circular)
pipe_cross_sectional_area_m2 = math.pi * pipe_radius_m**2

# =====
# FLOW RATE CALCULATIONS
# =====

# Calculate volumetric flow rate using  $Q = A \cdot v$ 
volumetric_flow_rate_m3_per_s = pipe_cross_sectional_area_m2 * oil_velocity_m_per_s

# Calculate oil density from relative density
oil_density_kg_per_m3 = oil_relative_density * water_density_kg_per_m3

# Calculate mass flow rate using  $\dot{m} = \rho \cdot Q$ 
mass_flow_rate_kg_per_s = oil_density_kg_per_m3 * volumetric_flow_rate_m3_per_s

# Convert mass flow rate to tonnes per hour
mass_flow_rate_tonnes_per_hour = mass_flow_rate_kg_per_s * 3600 / 1000

# =====
# OUTPUT RESULTS
# =====

print("Pipe Flow Analysis Results:")

```

```
print(f" Volumetric flow rate : {volumetric_flow_rate_m3_per_s:.4f} m³/s")
print(f" Mass flow rate       : {mass_flow_rate_tonnes_per_hour:.4f} " f"tonnes/hour")
```

### 4.3 Discharge Through an Orifice

When water discharges through an orifice (Figure 4.1) in the side of a tank, the potential energy associated with the height of the water surface above the orifice is converted into kinetic energy of the efflux.

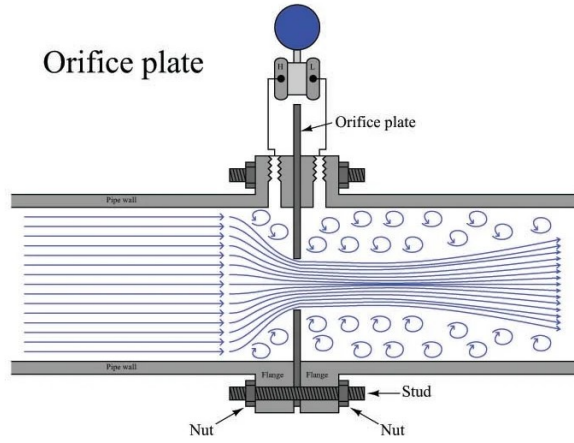


Figure 4.1: Orifice

This yields the theoretical velocity of the jet:

$$C = \sqrt{2gh}$$

where  $g$  is the acceleration due to gravity and  $h$  is the height of the water surface above the orifice.

The theoretical volume flow rate is then:

$$\dot{v}_{\text{theoretical}} = AC = A\sqrt{2gh}$$

where  $A$  is the area of the orifice.

Due to viscous effects, the actual velocity is less than the theoretical value. The coefficient of velocity,  $C_v$ , is defined as the ratio of actual velocity to theoretical velocity. Thus, the velocity-corrected flow rate is:

$$\dot{v}_{\text{velocity-corrected}} = C_v A \sqrt{2gh}$$

Downstream of a sharp-edged orifice, the jet contracts due to streamline curvature, forming a vena contracta (Figure 4.2). The vena contracta refers to the

narrowest cross-section of a fluid jet shortly downstream of an orifice or nozzle, where the streamlines converge, resulting in the smallest area, maximum velocity, and minimum pressure.

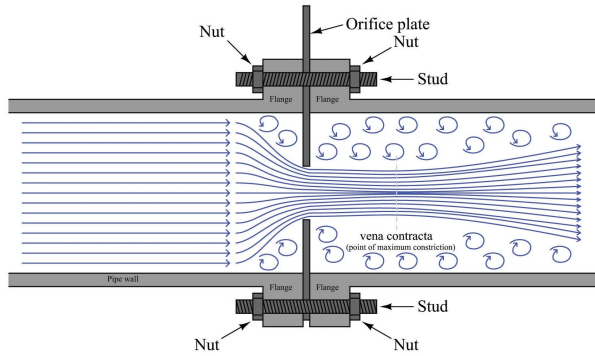


Figure 4.2: Vena contracta

The coefficient of contraction,  $C_A$ , is the ratio of the jet area at the vena contracta to the orifice area:

$$A_{\text{jet}} = C_A A$$

The actual volume flow rate therefore becomes:

$$\dot{v}_{\text{actual}} = C_A A \cdot C_v \sqrt{2gh}$$

The coefficient of discharge,  $C_d$ , is the ratio of the actual volume flow rate to the theoretical volume flow rate:

$$C_d = C_A \cdot C_v$$

Consequently,

$$\dot{v}_{\text{actual}} = C_d A \sqrt{2gh}$$

**Example 4.2.** Water escapes through a hole 20 mm in diameter in the side of a tank, with a water head of 3 m above the hole. Given a coefficient of velocity of 0.97 and a coefficient of reduction of area of 0.64, calculate (i) the velocity of the water jet as it exits the hole and (ii) the quantity of water escaping tonne per hour.

'''

#### Orifice Flow Calculator

Calculates the actual velocity and discharge rate of water escaping through an orifice under a given head. Uses coefficient of velocity ( $C_v$ ) and coefficient of area ( $C_a$ ) to account for real flow conditions.

```

"""

import math

# =====
# INPUT PARAMETERS
# =====

orifice_diameter_mm = 20 # Orifice diameter in millimeters
head_above_orifice_m = 3 # Height of water above orifice in meters
coefficient_of_velocity = 0.97 # Cv (accounts for velocity reduction)
coefficient_of_area = 0.64 # Ca (accounts for vena contracta)
gravitational_acceleration_m_per_s2 = 9.81 # Acceleration due to gravity
water_density_kg_per_m3 = 1000 # Density of water in kg/m³

# =====
# UNIT CONVERSIONS AND GEOMETRY
# =====

# Convert orifice diameter from millimeters to meters
orifice_diameter_m = orifice_diameter_mm / 1000

# Calculate orifice radius
orifice_radius_m = orifice_diameter_m / 2

# Calculate cross-sectional area of orifice
orifice_area_m2 = math.pi * orifice_radius_m**2

# =====
# VELOCITY CALCULATIONS
# =====

# Calculate theoretical velocity using Torricelli's theorem:  $v = \sqrt{2gh}$ 
theoretical_velocity_m_per_s = math.sqrt(
    2 * gravitational_acceleration_m_per_s2 * head_above_orifice_m
)

# Calculate actual velocity accounting for friction losses
actual_velocity_m_per_s = coefficient_of_velocity * theoretical_velocity_m_per_s

# =====
# FLOW RATE CALCULATIONS
# =====

# Calculate actual volumetric flow rate:  $Q = C_a \cdot A \cdot v$ 

```

```

# (Ca accounts for contraction of jet at vena contracta)
volumetric_flow_rate_m3_per_s = (
    coefficient_of_area * orifice_area_m2 * actual_velocity_m_per_s
)

# Calculate mass flow rate:  $\dot{m} = \rho \cdot Q$ 
mass_flow_rate_kg_per_s = water_density_kg_per_m3 * volumetric_flow_rate_m3_per_s

# Convert mass flow rate to per-hour basis
mass_flow_rate_kg_per_hour = mass_flow_rate_kg_per_s * 3600

# Convert mass flow rate to tonnes per hour
mass_flow_rate_tonnes_per_hour = mass_flow_rate_kg_per_hour / 1000

# =====
# OUTPUT RESULTS
# =====

print(f"(i) Actual velocity of the water jet: " f"{actual_velocity_m_per_s:.3f} m/s")
print("\n(ii) Quantity of water escaping per hour:")
print(f"    Volume flow rate: {volumetric_flow_rate_m3_per_s:.6f} m³/s")
print(f"    Mass flow rate: {mass_flow_rate_kg_per_s:.4f} kg/s")
print(f"    Mass per hour: {mass_flow_rate_kg_per_hour:.4f} kg/hour")
print(f"    Mass per hour: {mass_flow_rate_tonnes_per_hour:.4f} tonnes/hour")

```

## 4.4 Continuity Equation

Continuous flow exists in a flow system when the mass flow rate is constant throughout the system as shown in Figure 4.3. If in the diagram below, the mass flow rate at 1 is equal to that at 2, then continuity exists.

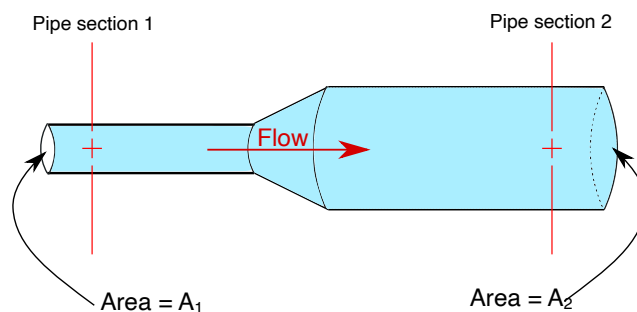


Figure 4.3: Continuity equation

Since  $\dot{m}_1 = \dot{m}_2$ , therefore

$$\rho_1 \cdot A_1 \cdot C_1 = \rho_2 \cdot A_2 \cdot C_2$$

If the fluid is incompressible (most liquids), then density will remain constant ( $\rho_1 = \rho_2$ ) and the above equations may be written as:

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

Or

$$v_1 = v_2$$

That is, the volume flow rate is constant for an incompressible fluid.

**Example 4.3.** A pipe decreases in diameter from 300 mm to 200 mm. Water flows from the larger to the smaller pipe at a constant rate of 18.4 kL/min. Calculate the mass flow rate and the velocities in the larger and smaller pipe.

```
"""
Pipe Flow Velocity Calculator

Calculates the mass flow rate and velocities in pipes of different
diameters carrying water at a constant volumetric flow rate. Uses
continuity equation to determine flow velocities.
"""

import math

# =====
# INPUT PARAMETERS
# =====

volumetric_flow_rate_kl_per_min = 18.4 # Flow rate in kiloliters/min
large_pipe_diameter_mm = 300 # Diameter of larger pipe in millimeters
small_pipe_diameter_mm = 200 # Diameter of smaller pipe in millimeters
water_density_kg_per_m3 = 1000 # Density of water in kg/m³

# =====
# UNIT CONVERSIONS
# =====

# Convert volumetric flow rate from kL/min to m³/s
# (1 kL = 1 m³, and 1 min = 60 s)
volumetric_flow_rate_m3_per_s = volumetric_flow_rate_kl_per_min / 60
```

```

# Convert pipe diameters from millimeters to meters
large_pipe_diameter_m = large_pipe_diameter_mm / 1000
small_pipe_diameter_m = small_pipe_diameter_mm / 1000

# Calculate pipe radii
large_pipe_radius_m = large_pipe_diameter_m / 2
small_pipe_radius_m = small_pipe_diameter_m / 2

# =====
# CROSS-SECTIONAL AREA CALCULATIONS
# =====

# Calculate cross-sectional area of larger pipe
large_pipe_area_m2 = math.pi * large_pipe_radius_m**2

# Calculate cross-sectional area of smaller pipe
small_pipe_area_m2 = math.pi * small_pipe_radius_m**2

# =====
# VELOCITY CALCULATIONS
# =====

# Calculate velocity in larger pipe using continuity equation:  $v = Q / A$ 
velocity_large_pipe_m_per_s = volumetric_flow_rate_m3_per_s / large_pipe_area_m2

# Calculate velocity in smaller pipe using continuity equation:  $v = Q / A$ 
velocity_small_pipe_m_per_s = volumetric_flow_rate_m3_per_s / small_pipe_area_m2

# =====
# MASS FLOW RATE CALCULATION
# =====

# Calculate mass flow rate using  $\dot{m} = \rho \cdot Q$ 
mass_flow_rate_kg_per_s = water_density_kg_per_m3 * volumetric_flow_rate_m3_per_s

# =====
# OUTPUT RESULTS
# =====

print(f"Mass flow rate: {mass_flow_rate_kg_per_s:.4f} kg/s")
print(
    f"Velocity in larger pipe ({large_pipe_diameter_mm} mm): "
    f"{velocity_large_pipe_m_per_s:.4f} m/s"
)
print(

```

```
f"Velocity in smaller pipe ({small_pipe_diameter_mm} mm): "
f"{velocity_small_pipe_m_per_s:.4f} m/s"
)
```

## 4.5 The Energy Equation for an Ideal Fluid

The steady flow equation is developed from the Law of Conservation of Energy. If there are no energy losses or energy additions in a flow system, then the total energy of a flowing fluid will remain constant.

A flowing fluid may lose energy as a result of fluid friction, heat energy transfer and fluid motors. For an ideal fluid, frictional losses are equal to zero. Energy may be added to a fluid via a pump or heat energy addition.

The total energy possessed by a flowing fluid consists of:

- internal energy
- potential energy
- kinetic energy and
- pressure energy (flow energy).

Generally in fluid mechanics the change in internal energy is considered to be negligible. Heat energy transfers are usually not considered in fluid mechanics and the frictional heat developed by a flowing fluid is relatively small. Therefore, the internal energy term is omitted from the energy balance.

Consider the flow system shown in Figure 4.4. For this system, assume:

- an ideal fluid
- that there are no energy losses or additions and
- steady flow conditions exist.

From the Law of Conservation of Energy:

The Total Energy at section 1 = The Total Energy at section 2

$$mZ_1g + \frac{mC_1^2}{2} + mP_1v_1 = mZ_2g + \frac{mC_2^2}{2} + mP_2v_2$$

Dividing through by  $mg$ :

$$\frac{mZ_1g}{mg} + \frac{mC_1^2}{2mg} + \frac{mP_1v_1}{mg} = \frac{mZ_2g}{mg} + \frac{mC_2^2}{2mg} + \frac{mP_2v_2}{mg}$$

Therefore

$$Z_1 + \frac{C_1^2}{2g} + \frac{P_1v_1}{g} = Z_2 + \frac{C_2^2}{2g} + \frac{P_2v_2}{g}$$

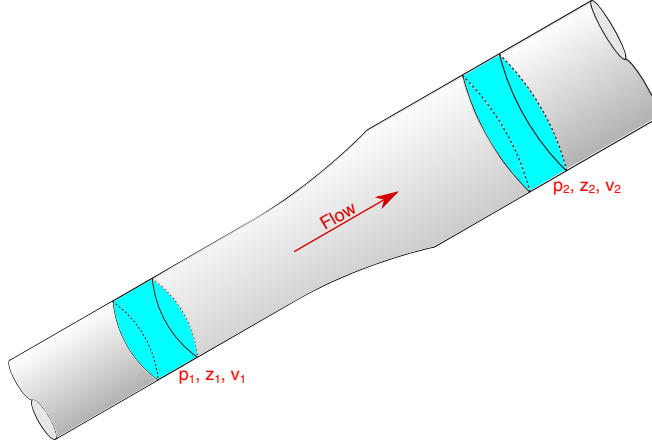


Figure 4.4: Energy equation

In fluid mechanics density ( $\rho$ ) is generally used in preference to specific volume, i.e.  $v = \frac{1}{\rho}$ . Therefore:

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

In fluid mechanics, specific weight represents the force exerted by gravity on a unit volume of a fluid. For this reason, units are expressed as force per unit volume (e.g.,  $N/m^3$ )

Specific weight is given  $\gamma = g\rho$

Where:

$\gamma$ : specific weight,  $N/m^3$

$g$ : gravitational acceleration,  $m/s^2$

$\rho$ : density,  $kg/m^3$

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

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.

Similarly, the total energy of a flowing fluid is known as the total head (H).

$$\text{Total Head} = \text{Elevation Head} + \text{Velocity Head} + \text{Pressure Head}$$

$$H = Z + \frac{C^2}{2g} + \frac{P}{\gamma}$$

The total head (H) will be a constant throughout a flow system if:

1. frictional losses (head loss) are equal to zero
2. work energy is not added by a pump (pump head) or removed by a motor.

**Example 4.4.** Consider a simple flow system consisting of a varying cross-section pipe. Water flows through this system at a rate of 2000 L/min. As the pipe increases in elevation from 30 m to 36 m it decreases in diameter from 10 cm to 3.0 cm. If the pressure is 6.5 MPa at the 30 m elevation, what is the pressure at 36 m?

```
"""
```

```
Energy Equation Pressure Calculator
```

```
Calculates the pressure at a higher elevation in a pipe using the energy
equation. Accounts for changes in velocity, elevation, and pressure between
two sections of a pipe carrying water.
```

```
"""
```

```
import math
```

```
# =====
# INPUT PARAMETERS
# =====
```

```
volumetric_flow_rate_l_per_min = 2000.0 # Flow rate in liters per minute
diameter_lower_section_m = 0.10 # Pipe diameter at lower section in meters
diameter_upper_section_m = 0.03 # Pipe diameter at upper section in meters
elevation_lower_section_m = 30.0 # Elevation at lower section in meters
elevation_upper_section_m = 36.0 # Elevation at upper section in meters
pressure_lower_section_pa = 6.5e6 # Pressure at lower section in Pascals
water_density_kg_per_m3 = 1000.0 # Density of water in kg/m³
gravitational_acceleration_m_per_s2 = 9.81 # Acceleration due to gravity
```

```
# =====
# UNIT CONVERSIONS
# =====
```

```
# Convert volumetric flow rate from L/min to m³/s
# (1 L = 0.001 m³, 1 min = 60 s)
```

```

volumetric_flow_rate_m3_per_s = volumetric_flow_rate_l_per_min / 60000.0

# =====
# CROSS-SECTIONAL AREA CALCULATIONS
# =====

# Calculate cross-sectional area at lower section
area_lower_section_m2 = math.pi * (diameter_lower_section_m / 2.0) ** 2

# Calculate cross-sectional area at upper section
area_upper_section_m2 = math.pi * (diameter_upper_section_m / 2.0) ** 2

# =====
# VELOCITY CALCULATIONS
# =====

# Calculate velocity at lower section using continuity equation:  $v = Q / A$ 
velocity_lower_section_m_per_s = volumetric_flow_rate_m3_per_s / area_lower_section_m2

# Calculate velocity at upper section using continuity equation:  $v = Q / A$ 
velocity_upper_section_m_per_s = volumetric_flow_rate_m3_per_s / area_upper_section_m2

# =====
# PRESSURE CALCULATION USING ENERGY EQUATION
# =====

# Apply the energy equation between lower and upper sections:
#  $P + \frac{1}{2} v^2 + gz = P + \frac{1}{2} v^2 + gz$ 
# Solving for P :
#  $P = P + \frac{1}{2} (v^2 - v^2) + g(z - z)$ 

pressure_upper_section_pa = (
    pressure_lower_section_pa
    + 0.5
    * water_density_kg_per_m3
    * (velocity_lower_section_m_per_s**2 - velocity_upper_section_m_per_s**2)
    + water_density_kg_per_m3
    * gravitational_acceleration_m_per_s2
    * (elevation_lower_section_m - elevation_upper_section_m)
)

# Convert pressure to MPa for output
pressure_upper_section_mpa = pressure_upper_section_pa / 1e6

# =====

```

```
# OUTPUT RESULTS
# =====

print(f"Flow rate: {volumetric_flow_rate_m3_per_s:.5f} m³/s")
print(f"Area 1: {area_lower_section_m2:.6f} m²")
print(f"Area 2: {area_upper_section_m2:.6f} m²")
print(f"Velocity 1: {velocity_lower_section_m_per_s:.4f} m/s")
print(f"Velocity 2: {velocity_upper_section_m_per_s:.4f} m/s")
print(
    f"Pressure at {elevation_upper_section_m} m elevation: "
    f"{pressure_upper_section_mpa:.4f} MPa"
)
```

## 4.6 Bernoulli's Equation

The Bernoulli's equation between two points in a fluid flow is given by:

$$P_1 + \frac{1}{2}\rho C_1^2 + \rho gh_1 = P_2 + \frac{1}{2}\rho C_2^2 + \rho gh_2$$

Where:

- $P_1$  and  $P_2$  are the pressures at points 1 and 2, respectively.
- $\rho$  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.

## 4.7 Venturi Meter

A venturi meter (Figure 4.5) measures liquid flow rates in pipelines. The device features a pipe section that narrows in the middle (called the throat) and widens at both ends, as shown below:

When the entrance and throat areas are known, and the pressure readings (or pressure difference) at these points are measured, Bernoulli's equation can be used to calculate the liquid velocity and flow rate. Typically, a venturi meter is installed horizontally in the pipeline, which simplifies calculations since there is no elevation difference between points ( $Z_1 = Z_2$ ), thus eliminating the height terms.

**Example 4.5.** A horizontal venturi meter has an inlet diameter of 450 mm and a throat diameter of 225 mm. The pressure difference between these two

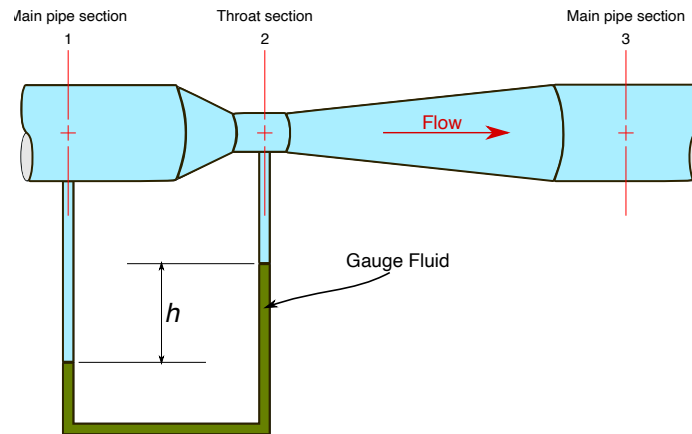


Figure 4.5: Venturi meter

points is equivalent to 381 mm of water. Given the density of fresh water is  $1000 \text{ kg/m}^3$ , calculate the mass flow rate through the meter.

```
"""
Venturi Meter Flow Calculator

Calculates the flow rate through a Venturi meter using Bernoulli's
equation. Determines throat velocity and volumetric/mass flow rates
from the pressure differential between inlet and throat sections.
"""

import math

# =====
# INPUT PARAMETERS
# =====

inlet_diameter_m = 0.450 # Inlet diameter in meters
throat_diameter_m = 0.225 # Throat diameter in meters
pressure_head_difference_m = 0.381 # Pressure head difference in meters
gravitational_acceleration_m_per_s2 = 9.81 # Acceleration due to gravity
water_density_kg_per_m3 = 1000 # Density of water in kg/m³

# =====
# CROSS-SECTIONAL AREA CALCULATIONS
# =====

# Calculate cross-sectional area at inlet
inlet_area_m2 = math.pi * (inlet_diameter_m / 2) ** 2
```

```

# Calculate cross-sectional area at throat
throat_area_m2 = math.pi * (throat_diameter_m / 2) ** 2

# =====
# VELOCITY CALCULATION USING BERNOULLI'S EQUATION
# =====

# Compute the squared area ratio (A_throat / A_inlet)2
area_ratio_squared = (throat_area_m2 / inlet_area_m2) ** 2

# Calculate throat velocity using Bernoulli's equation for a Venturi meter:
#  $c = \sqrt{2gh / (1 - (A/A)^2)}$ 
# where h is the pressure head difference
throat_velocity_m_per_s = math.sqrt(
    (2 * gravitational_acceleration_m_per_s2 * pressure_head_difference_m)
    / (1 - area_ratio_squared)
)

# =====
# FLOW RATE CALCULATIONS
# =====

# Calculate volumetric flow rate using  $Q = A \cdot v$  at throat section
volumetric_flow_rate_m3_per_s = throat_area_m2 * throat_velocity_m_per_s

# Calculate mass flow rate using  $\dot{m} = \rho \cdot Q$ 
mass_flow_rate_kg_per_s = water_density_kg_per_m3 * volumetric_flow_rate_m3_per_s

# =====
# OUTPUT RESULTS
# =====

print(f"Entrance area: {inlet_area_m2:.6f} m2")
print(f"Throat area: {throat_area_m2:.6f} m2")
print(f"Throat velocity: {throat_velocity_m_per_s:.4f} m/s")
print(f"Volumetric flow rate: {volumetric_flow_rate_m3_per_s:.4f} m3/s")
print(f"Mass flow rate: {mass_flow_rate_kg_per_s:.4f} kg/s")

```

**Example 4.6.** A 300 mm diameter pipe has a venturi meter with a throat diameter of 100 mm. A U-tube gauge filled with water and mercury measures a pressure head difference of 250 mm between the inlet and throat. The meter coefficient is 0.95. Using the density of water (1000 kg/m<sup>3</sup>), calculate the discharge rate through the pipe in m<sup>3</sup>/s.

```

"""
Venturi Meter with Mercury Manometer Calculator

Calculates the actual discharge through a Venturi meter using a mercury
manometer to measure pressure difference. Accounts for the coefficient
of discharge and converts mercury column height to equivalent water head.
"""

import math

# =====
# INPUT PARAMETERS
# =====

inlet_diameter_m = 0.3 # Inlet diameter in meters
throat_diameter_m = 0.1 # Throat diameter in meters
coefficient_of_discharge = 0.95 # Discharge coefficient (dimensionless)
mercury_column_height_m = 0.250 # Manometer reading in meters of mercury
mercury_specific_gravity = 13.6 # Specific gravity of mercury (water = 1)
gravitational_acceleration_m_per_s2 = 9.81 # Acceleration due to gravity

# =====
# CROSS-SECTIONAL AREA CALCULATIONS
# =====

# Calculate cross-sectional area at inlet
inlet_area_m2 = math.pi * (inlet_diameter_m**2) / 4

# Calculate cross-sectional area at throat
throat_area_m2 = math.pi * (throat_diameter_m**2) / 4

# =====
# EQUIVALENT WATER HEAD CALCULATION
# =====

# Convert mercury column height to equivalent water head
# The pressure difference measured by mercury must be converted to
# water equivalent by accounting for the density difference:
#  $h_{\text{water}} = h_{\text{mercury}} \times (\text{SG}_{\text{mercury}} - 1)$ 
equivalent_water_head_m = mercury_column_height_m * (mercury_specific_gravity - 1)

# =====
# DIAMETER RATIO CALCULATION
# =====

```

```

# Calculate diameter ratio (beta) for use in discharge equation
diameter_ratio = throat_diameter_m / inlet_diameter_m

# =====
# DISCHARGE CALCULATIONS
# =====

# Calculate theoretical discharge using Venturi meter equation:
# Q_theoretical = A × √[2gh / (1 - )]
# where = D/D is the diameter ratio
theoretical_discharge_m3_per_s = throat_area_m2 * math.sqrt(
    (2 * gravitational_acceleration_m_per_s2 * equivalent_water_head_m)
    / (1 - diameter_ratio**4)
)

# Calculate actual discharge accounting for losses:
# Q_actual = Cd × Q_theoretical
actual_discharge_m3_per_s = coefficient_of_discharge * theoretical_discharge_m3_per_s

# =====
# OUTPUT RESULTS
# =====

print(f"Cross-sectional area at inlet (A1): {inlet_area_m2:.4f} m²")
print(f"Cross-sectional area at throat (A2): {throat_area_m2:.4f} m²")
print(f"Equivalent water head (h_eq): {equivalent_water_head_m:.4f} m")
print(f"Theoretical discharge: {theoretical_discharge_m3_per_s:.4f} m³/s")
print(f"Actual discharge: {actual_discharge_m3_per_s:.4f} m³/s")

```

# References

- Bird, J. O. (2021). *Bird's engineering mathematics* (Ninth edition). Routledge.
- Bolton, W. (2021). *Engineering science* (Seventh edition). Routledge.
- Hannah, J., & Hillier, M. J. (1995). *Applied mechanics* (3rd edition). Longman Pub Group.
- 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.
- Sweigart, A. (2020). *Automate the boring stuff with python, 2nd edition: Practical programming for total beginners*. No Starch Press.
- Sweigart, A. (2021). *Beyond the basic stuff with python: Best practices for writing clean code*. No Starch Press.



# Author

Serhat Beyenir is an instructor at BCIT's Marine Campus, where he teaches Thermodynamics, Applied Mechanics, Propulsion Engines, and Propulsion Plant Simulator courses.

His interest in computing dates back to high school, when he taught himself BASIC to plot trigonometric functions on an Amstrad 64K microcomputer. He completed programming coursework at the university level but is largely self-taught in the use of Unix and Linux systems, later adopting macOS for personal projects in 2007. While teaching Power Engineering, he employed MATLAB for engineering computations and compiled his instructional materials into an open textbook, *A Brief Introduction to Engineering Computation with MATLAB*, published in 2011.

In February 2025, he attended a Python course at the West Vancouver Memorial Library and subsequently began using Python in Thermodynamics and Applied Mechanics lecture materials, gradually incorporating computational examples into his teaching.

Serhat holds a Class 1 Marine Engineer certification, a Master of Education in Distance Education, and a Bachelor of Science in Marine Engineering.



## Appendix A

# Applied Mechanics 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:

- $\omega_1$ : Initial angular velocity (rad/s)
- $\omega_2$ : Final angular velocity (rad/s)
- $\theta$ : Angular displacement (rad)
- $\alpha$ : Angular acceleration (rad/s<sup>2</sup>)
- $t$ : 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  $\theta$ ).

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

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

### A.5 Centre of Gravity

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

### A.6 Centroid

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

## 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

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

Where:

- Linear momentum is in  $\text{kg} \cdot \text{m/s}$ .
- $m$  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:

- Angular momentum is in  $\text{kgm}^2/\text{s}$
- $I$  is the moment of inertia in  $\text{kgm}^2$ .
- $\omega$  is the angular velocity in  $\text{rad/s}$ .

### A.10.3 Moment of inertia

$$I = mk^2$$

Where:

- $I$  is the moment of inertia in  $\text{kgm}^2$ .
- $m$  is the mass in  $\text{kg}$ .
- $k$  is the radius of gyration in  $\text{m}$ .

### A.10.4 Turning moment

$$\tau = I\alpha$$

Where:

- $\tau$  is the torque in  $\text{Nm}$ .
- $I$  is the moment of inertia in  $\text{kgm}^2$ .
- $\alpha$ : Angular acceleration in  $\text{rad/s}^2$ .

**A.10.5 Power by Torque**

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

Where:

$P$  is the power in watts (W),

$\tau$  is the torque in newton-meters (Nm), and

$\omega$  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  $kgm^2$ .
- $\omega$  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:

- $\sigma$  is the stress (Pa),
- $\tau$  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:

- $\varepsilon$  is the strain (unitless),
- $\Delta L$  is the change in length,
- $L_0$  is the original length.

**A.11.3 Hooke's Law**

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

Where:

- $\sigma$  is the stress (Pa).
- $\varepsilon$  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<sup>3</sup>/s

A: Cross-sectional area of the flow, m<sup>2</sup>

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

$\rho$ : density, kg/m<sup>3</sup>

$\dot{v}$ : volume flow, m<sup>3</sup>/s

**A.12.3 Specific Weight**

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

Where:

$\gamma$ : specific weight, N/m<sup>3</sup>

$g$ : gravitational acceleration, m/s<sup>2</sup>

$\rho$ : density, kg/m<sup>3</sup>

**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.

$$\textit{Total Head} = \textit{Elevation Head} + \textit{Velocity Head} + \textit{Pressure Head}$$

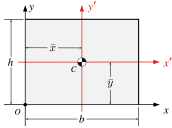
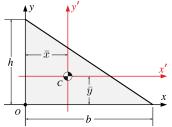
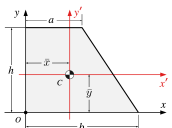
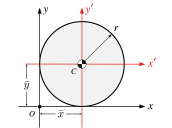
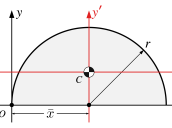
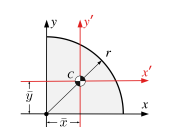
**A.12.6 Bernoulli's Equation**

$$P_1 + \frac{1}{2}\rho C_1^2 + \rho gh_1 = P_2 + \frac{1}{2}\rho C_2^2 + \rho gh_2$$

Where:

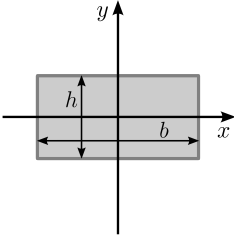
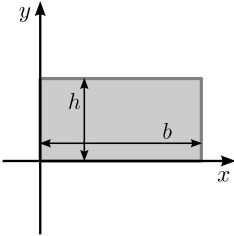
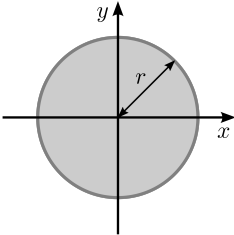
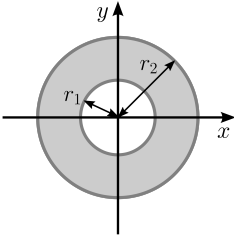
- $P_1$  and  $P_2$  are the pressures at points 1 and 2, respectively.
- $\rho$  is the density of the fluid.
- $C_1$  and  $C_2$  are the velocities of the fluid at points 1 and 2, respectively.
- $g$  is the acceleration due to gravity.
- $h_1$  and  $h_2$  are the heights of the fluid at points 1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

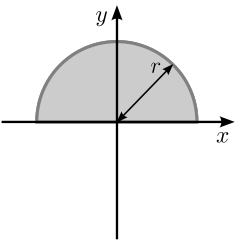
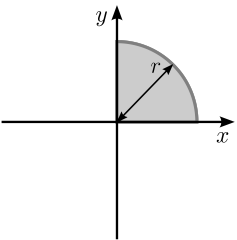
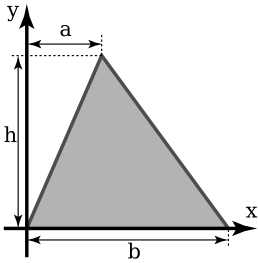
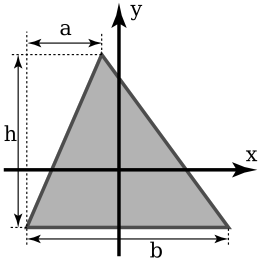
Table A.3: Centroids of Common Shapes

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

## 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

# 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 B.1: Greek letters.

Lower Case	Upper Case	Name
$\alpha$	A	alpha
$\beta$	B	beta
$\gamma$	$\Gamma$	gamma
$\delta$	$\Delta$	delta
$\epsilon$	E	epsilon
$\zeta$	Z	zeta
$\eta$	E	eta
$\theta$	$\Theta$	theta
$\iota$	I	iota
$\kappa$	K	kappa
$\lambda$	$\Lambda$	lambda
$\mu$	M	mu
$\nu$	N	nu
$\xi$	$\Xi$	xi
$o$	O	omicron
$\pi$	$\Pi$	pi
$\rho$	P	rho
$\sigma$	$\Sigma$	sigma
$\tau$	T	tau
$v$	$\Upsilon$	upsilon
$\phi$	$\Phi$	phi
$\chi$	X	chi

Lower Case	Upper Case	Name
$\psi$	$\Psi$	psi
$\omega$	$\Omega$	omega

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

Symbol	Name	Use	Course
$\Delta$	Delta	Change	Thermodynamics
$\Delta$	Delta	Displacement	Naval Architecture
$\nabla$	Nabla	Volume	Naval Architecture
$\Sigma$	Sigma	Sum	Thermodynamics, Naval Architecture, Applied Mechanics
$\sigma$	Sigma	Stress	Thermodynamics, Applied Mechanics
$\epsilon$	Epsilon	Modulus of elasticity	Thermodynamics, Applied Mechanics
$\eta$	Eta	Efficiency	Thermodynamics
$\mu$	Mu	Friction	Thermodynamics, Applied Mechanics
$\omega$	Omega	Angular velocity	Thermodynamics, Applied Mechanics
$\rho$	Rho	Density	Thermodynamics, Naval Architecture
$\tau$	Tau	Torque	Thermodynamics, Applied Mechanics

# Index

colon, 7  
conditional statements, 10  
control flow, 10  
Cosine rule, 19  
  
def keyword, 11  
  
for loop, 11  
  
indentation, 7  
  
lambda keyword, 12  
  
math module, 12  
  
print, 8  
  
Sine rule, 21  
Space diagram, 17  
Statics, 17  
  
variables, 8  
Vector diagram, 17  
  
while loop, 11