

# **Applied Numerical Methods for Civil Engineering**

CGN 3405 - 0002

## **Week 2: Mathematical Modeling & Engineering Problem Solving**

**Xinyu Chen**

Assistant Professor

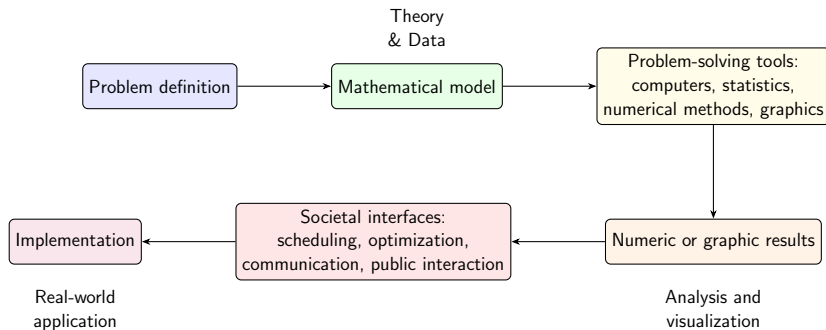
University of Central Florida

How to understand

## **Applied Numerical Methods for Civil Engineering?**

**Numerical methods** are techniques by which **mathematical problems** are formulated so that they can be solved with **arithmetic operations**.

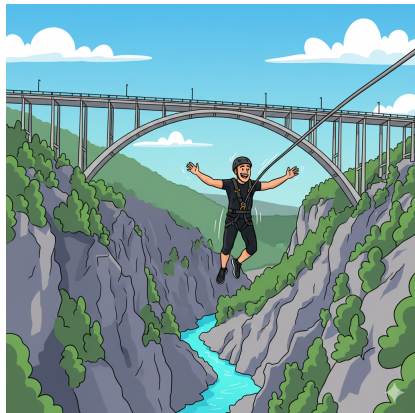
# Engineering Problem Solving Process



## Bungee Jumping

### Engineering Task.

- A bungee jumping company needs to **predict velocity vs. time** during free fall to design safe bungee cords.
- **Key Questions:**
  - What is the **maximum velocity** reached?  
(Safe limit: 45 m/s)
  - How long until maximum velocity?
  - What cord length is needed?



## Physical Forces $F_g$ and $F_a$

### Two Main Forces: Physical Forces Acting on Jumper

$$F = F_g - F_a = m \cdot g - c_d \cdot v^2$$

- Gravity (Downward)

$$F_g = m \cdot g$$

with

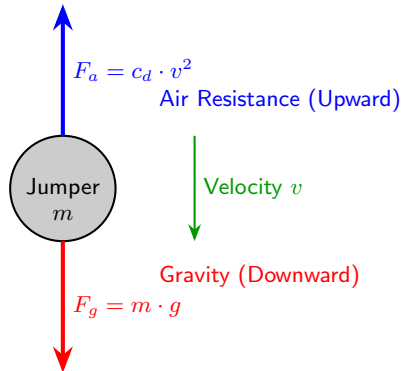
- $m$  = mass (kg)
- $g = 9.81 \text{ m/s}^2$ , gravitational acceleration

- Air Resistance (Upward)

$$F_a = c_d \cdot v^2$$

with

- $c_d$  = drag coefficient (kg/m)
- $v$  = velocity



## Newton's Second Law

### Mathematical Model - Newton's Second Law

- From  $F = m \cdot a$ :

$$F = m \frac{dv}{dt} = m \cdot g - c_d \cdot v^2$$

- Divide by  $m$ :

$$\frac{dv}{dt} = g - \frac{c_d}{m} v^2$$

Ordinary Differential Equation!!!

in terms of the differential rate of change in velocity.

- Initial condition:

$$v(0) = 0 \quad (\text{starts from rest})$$

- Problem definition:** Solve the velocity of the jumper in free fall as a function of time.
- Why Numerical Methods?**
  - Real engineering problems often **do not have simple analytical solutions!**

## Euler's Method (Numerical)

### Euler's Method - The Simplest Numerical Approach

- Essential idea:

Approximate continuous change with a small discrete time step size  $\Delta t$ .

- Rewrite the formula of bungee jumper velocity:

$$\begin{aligned}
 \underbrace{v_{t+\Delta t}}_{\text{new}} &= v_t + \Delta t \cdot \frac{dv_t}{dt} \\
 &= \underbrace{v_t}_{\text{old}} + \underbrace{\Delta t}_{\text{time step size}} \cdot \underbrace{\left(g - \frac{c_d}{m} v_t^2\right)}_{\text{acceleration}}
 \end{aligned}$$

from the ordinary differential equation:

$$\frac{dv}{dt} = g - \frac{c_d}{m} v^2$$

## Euler's Method (Numerical)

### Euler's Method - The Simplest Numerical Approach

- Formula of bungee jumper velocity:

$$\underbrace{v_{t+\Delta t}}_{\text{new}} = \underbrace{v_t}_{\text{old}} + \underbrace{\Delta t}_{\text{time step size}} \cdot \underbrace{\left(g - \frac{c_d}{m} v_t^2\right)}_{\text{acceleration}}$$

- Computing **bungee jumper velocity** (step-by-step):

- Start at  $t = 0$  and  $v = 0$
- Repeat across different time steps:
  - Compute **acceleration**:

$$a = g - \frac{c_d}{m} v_t^2$$

- Update **velocity**:

$$v_{t+\Delta t} = v_t + \Delta t \cdot a$$

- Increment time step:  $t = t + \Delta t$

## A Real Case

**Input.** Mass  $m = 50$  kg,  $g = 9.81$  m/s<sup>2</sup>, drag coefficient  $c_d = 0.25$  kg/m, and initial velocity  $v_0 = 0$ . (Given time step size  $\Delta t = 1$  s)

**Output.** Bungee jumper velocity  $v_t$ .

- At time  $t = 1$ :

$$a = g - \frac{c_d}{m} v_0^2 = 9.81 - 0.005 \times 0^2 = 9.81$$

$$v_1 = v_0 + \Delta t \cdot a = 0 + 9.81 = \mathbf{9.81}$$

- At time  $t = 2$ :

$$a = g - \frac{c_d}{m} v_1^2 = 9.81 - 0.005 \times 9.81^2 = 9.33$$

$$v_2 = v_1 + \Delta t \cdot a = 9.81 + 1 \times 9.33 = \mathbf{19.14}$$

- At time  $t = 3$

## A Real Case

**Input.** Mass  $m = 50$  kg,  $g = 9.81$  m/s<sup>2</sup>, drag coefficient  $c_d = 0.25$  kg/m, and initial velocity  $v_0 = 0$ . (Given time step size  $\Delta t = 1$  s)

**Output.** Bungee jumper velocity  $v_t$ .

- At time  $t = 1$ :

$$a = g - \frac{c_d}{m} v_0^2 = 9.81 - 0.005 \times 0^2 = 9.81$$

$$v_1 = v_0 + \Delta t \cdot a = 0 + 9.81 = \mathbf{9.81}$$

- At time  $t = 2$ :

$$a = g - \frac{c_d}{m} v_1^2 = 9.81 - 0.005 \times 9.81^2 = 9.33$$

$$v_2 = v_1 + \Delta t \cdot a = 9.81 + 1 \times 9.33 = \mathbf{19.14}$$

- At time  $t = 3$

$$a = g - \frac{c_d}{m} v_2^2 = 9.81 - 0.005 \times 19.14^2 = 7.98$$

$$v_3 = v_2 + \Delta t \cdot a = 19.14 + 1 \times 7.98 = \mathbf{27.12}$$

- ...

## The Basic Syntax of a for Loop in Python

### Description.

- A **for** loop in Python is a control flow statement used to iterate over items of any sequence (such as a list, tuple, string, set, or dictionary) in the order that they appear.
- It is primarily used when you need to execute a block of code a specific, predetermined number of times or for each item in a collection.

## The Basic Syntax of a for Loop in Python

### Fibonacci Sequence.

- Definition: Given  $f(1) = f(2) = 1$ , the Fibonacci sequence takes the form of

$$f(n) = f(n-1) + f(n-2), n > 2$$

- Write down  $f(3)$ ,  $f(4)$ ,  $f(5)$ ,  $f(6)$ ,  $f(7)$ ,  $\dots$  by yourself?

## The Basic Syntax of a for Loop in Python

### Fibonacci Sequence.

- Definition: Given  $f(1) = f(2) = 1$ , the Fibonacci sequence takes the form of

$$f(n) = f(n-1) + f(n-2), n > 2$$

- Write down  $f(3)$ ,  $f(4)$ ,  $f(5)$ ,  $f(6)$ ,  $f(7)$ ,  $\dots$  by yourself?

$$f(3) = f(2) + f(1) = 2$$

$$f(4) = f(3) + f(2) = 3$$

$$f(5) = f(4) + f(3) = 5$$

$$f(6) = f(5) + f(4) = 8$$

$$f(7) = f(6) + f(5) = 13$$

## The Basic Syntax of a for Loop in Python

### Fibonacci Sequence.

- Definition: Given  $f(1) = f(2) = 1$ , the Fibonacci sequence takes the form of

$$f(n) = f(n-1) + f(n-2), n > 2$$

- Python programming

```
1 import numpy as np
2
3 def fib(n):          # Input n>2
4     f = np.zeros(n)
5     f[0] = 1
6     f[1] = 1
7     for i in range(2, n):
8         f[i] = f[i - 1] + f[i - 2]
9     return f[n - 1]
```

## Python Programming for Euler's Method

- **Python programming example.** Computing **bungee jumper velocity**:

- Start at  $t = 0$  and  $v = 0$
- Repeat across different time steps:
  - Compute **acceleration**:

$$a = g - \frac{c_d}{m} v_t^2$$

- Update **velocity**:

$$v_{t+\Delta t} = v_t + \Delta t \cdot a$$

- Increment time step:  $t = t + \Delta t$

```

1 import numpy as np
2
3 def euler(m, g, cd, v0, delta_t, time_steps):
4     v = np.zeros(time_steps)           # Velocity
5     v[0] = v0                          # Initial velocity
6     for i in range(time_steps - 1):    # Repeat
7         a = g - cd / m * (v[i] ** 2)   # Acceleration
8         v[i + 1] = v[i] + delta_t * a  # Velocity
9     return v

```

## A Real Case

- Mass:  $m = 50$  kg
- Gravitational acceleration:  $g = 9.81$  m/s<sup>2</sup>
- Drag coefficient:  $c_d = 0.25$  kg/m

```

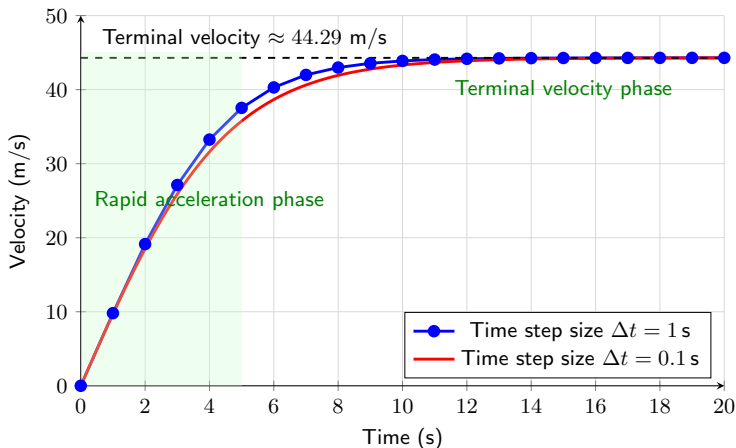
1 import numpy as np
2
3 # Parameters
4 m = 50                # Mass (kg)
5 g = 9.81              # Gravitational acceleration (m/s^2)
6 cd = 0.25             # Drag coefficient
7 v0 = 0               # Initial velocity
8
9 # Time setup
10 delta_t = 1          # Time step size
11 t_end = 20           # Total time
12 time_steps = int(t_end / delta_t) + 1
13
14 # Euler's method
15 t = np.linspace(0, t_end, time_steps)
16 v = euler(m, g, cd, v0, delta_t, time_steps)

```

## Velocity vs. Time

Bungee jumper **velocity vs. time** (w/ air resistance)

- Comparison between  $\Delta t = 1\text{ s}$  and  $\Delta t = 0.1\text{ s}$
- Input:  $m = 50\text{ kg}$ ,  $g = 9.81\text{ m/s}^2$ , and  $c_d = 0.25\text{ kg/m}$



## Velocity vs. Time

**Terminal velocity** (solving a simple quadratic equation):

$$\underbrace{a = g - \frac{c_d}{m}v^2 = 0}_{\text{acceleration} = 0} \Rightarrow v = \sqrt{\frac{mg}{c_d}}$$

In this case:

$$v = \sqrt{\frac{mg}{c_d}} = \sqrt{\frac{50 \times 9.81}{0.25}} = 44.29 \text{ m/s}$$

**Numerical method insight.**

- Demonstrates **importance of time step selection** in simulations
- **Fine time steps** give more accurate results
- **Coarse time steps** are faster to compute but less accurate

## Numerical vs. Analytical Solution

Going back to the ordinary differential equation:

$$\frac{dv}{dt} = g - \frac{c_d}{m}v^2$$

which has solution:

$$v_t = \sqrt{\frac{mg}{c_d}} \tanh\left(\sqrt{\frac{gc_d}{m}}t\right) \quad \text{tangent: } \tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

```

1 import numpy as np
2
3 def analytical_solution(m, g, cd, t):
4     v_term = np.sqrt(m * g / cd)
5     v_analytical = v_term * np.tanh(np.sqrt(g * cd / m) * t)
6     return v_analytical

```

```

1 delta_t = 1          # Time step size
2 t_end = 20           # Total time
3 time_steps = int(t_end / delta_t) + 1
4
5 # Computing the analytical solution
6 t = np.linspace(0, t_end, time_steps)
7 v_analytical = analytical_solution(m, g, cd, t)

```

# Numerical Error Analysis

## How to analyze errors?

```

1 error = v - v_analytical
2 plt.plot(t, error, 'red')
3 plt.xlabel('Time (s)')
4 plt.ylabel('Error (m/s)')
5 plt.show()

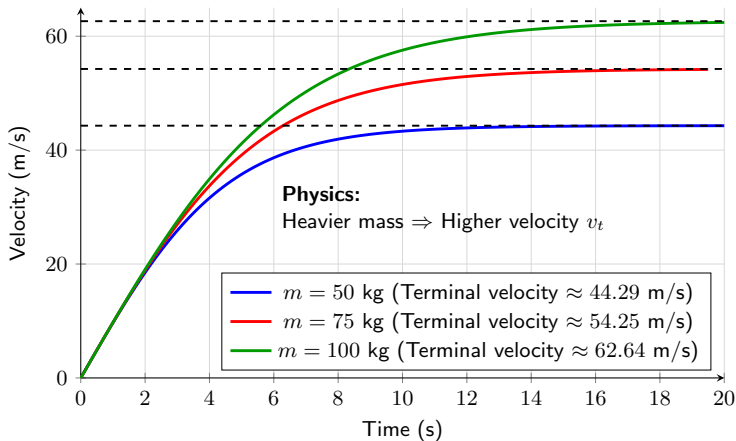
```

- Why errors?
  - Euler method assumes constant acceleration over  $\Delta t$ .
  - Smaller  $\Delta t \rightarrow$  Smaller error, but more computation.
- Time step comparison:
  - Time step size  $\Delta t = 1 \text{ s}$ : Error  $\approx 1.96 \text{ m/s}$
  - Time step size  $\Delta t = 0.1 \text{ s}$ : Error  $\approx 0.18 \text{ m/s}$
  - Time step size  $\Delta t = 0.01 \text{ s}$ : Error  $\approx 0.02 \text{ m/s}$
- Engineering trade-off: Accuracy vs. Computational cost

## Velocity vs. Time (Different Mass)

Bungee jumper **velocity vs. time** (w/ air resistance)

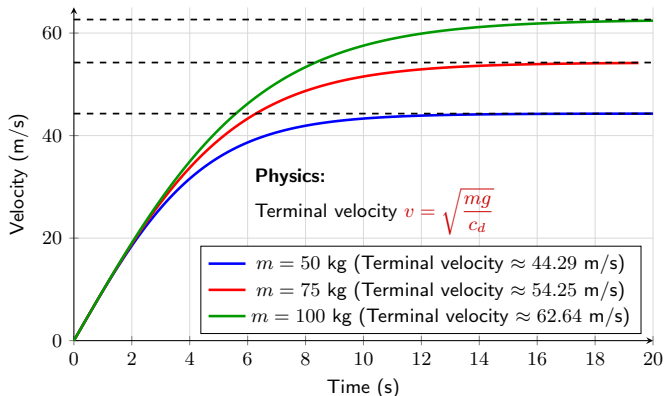
- Comparison among mass  $m = 50$  kg,  $75$  kg,  $100$  kg
- Input:  $g = 9.81$  m/s<sup>2</sup>, and  $c_d = 0.25$  kg/m



## Engineering Safety Analysis

**Safe limit:** Typically **45 m/s** (160 km/h) for bungee jumping

- Input:  $g = 9.81 \text{ m/s}^2$ , and  $c_d = 0.25 \text{ kg/m}$



- Terminal velocity exceeds safe limit? Increase drag coefficient (baggy clothing); Deploy parachute earlier; Use heavier cord for more drag.

## Parameter Sensitivity

How do mass and drag affect terminal velocity?

```
1 mass = [75, 100]
2 drag = [0.15, 0.25, 0.5]
3
4 for m in mass:
5     for cd in drag:
6         v_term = np.sqrt(m * g / cd)
7         print('Mass: {}'.format(m))
8         print('Drag coefficient: {}'.format(cd))
9         print('Terminal velocity: {}'.format(v_term))
10        print()
```

Results:

- Lighter jumpers → Lower terminal velocity
- Higher drag coefficient → Lower terminal velocity
- **Design implication:** Need different cords for different jumper weights!

## Quizzes Now!

- **Today's participation** (ungraded survey): Please check out

**"Class Participation Quiz 3"**

Time slot: **3:00PM – 3:30PM**

on Canvas.

- Online engagement (graded quizzes)

**"Quiz 3"** (14 questions)

Deadline: **11:59PM, January 21, 2026**

on Canvas.

## Quick Summary

### Wednesday's Class:

- Bungee jumping velocity vs. time
  - Newton's second law  $F = F_g - F_a = mg - c_d \cdot v^2 = m \cdot a$
  - Ordinary differential equation (the differential rate of change in velocity  $\rightarrow$  acceleration)

$$\frac{dv}{dt} = g - \underbrace{\frac{c_d}{m} v^2}_{\text{acceleration}}$$

- Euler's method for numerical computing

$$\underbrace{v_{t+\Delta t}}_{\text{new}} = \underbrace{v_t}_{\text{old}} + \underbrace{\Delta t}_{\text{time step size}} \cdot \underbrace{\left(g - \frac{c_d}{m} v_t^2\right)}_{\text{acceleration}}$$

- Numerical error analysis
- Sensitivity across different parameters
- Python programming
  - Fibonacci sequence
  - Numerical computing

## Quizzes Now!

- **Today's participation** (ungraded survey): Please check out

**"Class Participation Quiz 4"**

Time slot: **2:30PM – 3:00PM**

on Canvas.

- Online engagement (graded quizzes)

**"Quiz 4"** (15 questions)

Deadline: **11:59PM, January 23, 2026**

on Canvas.

## Euler's Method

**Euler's Method** is the **simplest numerical technique** for solving **Ordinary Differential Equations (ODEs)**.

- It approximates continuous change using small, discrete steps.
- When to use it?
  - When you know the rate of change  $\frac{dy}{dx}$
  - When you need a quick, approximate solution
  - When other methods are too complex

## Euler's Method

**Euler's Method** is the **simplest numerical technique** for solving **Ordinary Differential Equations (ODEs)**.

- It approximates continuous change using small, discrete steps.
- When to use it?
  - When you know the **rate of change**  $\frac{dy}{dx}$
  - When you need a **quick, approximate solution**
  - When other methods are too complex

**Bungee jumping velocity vs. time?**

- We know the rate of change in velocity:

$$\frac{dv}{dt} = g - \frac{c_d}{m}v^2$$

- We need an approximate solution:

$$\underbrace{v_{t+\Delta t}}_{\text{new velocity}} = \underbrace{v_t}_{\text{old velocity}} + \underbrace{\Delta t}_{\text{time step size}} \cdot \underbrace{\frac{dv}{dt}}_{\text{acceleration}}$$

## Mathematical Formulation

- **Example.** Given an ODE:

$$\frac{dy}{dx} = f(x, y)$$

with initial condition  $y(x_0) = y_0$

- **Euler's formula:**

$$\underbrace{y_{i+1}}_{\text{next value}} = \underbrace{y_i}_{\text{current value}} + \underbrace{\Delta x}_{\text{step size}} \cdot \underbrace{f(x_i, y_i)}_{\text{slope}}$$

$$x_{i+1} = x_i + \underbrace{\Delta x}_{\text{step size}}$$

- **Interpretation:**
  - $f(x_i, y_i)$  = slope at current point
  - $\Delta x$  = step size (small values!)
  - step size  $\times$  slope = predicted change in  $y$
  - Add to current  $y$  to get next  $y$

## Simple Example

- **Toy example:** Solve

$$\frac{dy}{dx} = x + y$$

with  $y(0) = 1$ , find  $y(1)$  using step size  $\Delta x = 0.5$ .

- ❶ Initialize  $x_0 = 0$  and  $y_0 = 1$
- ❷ First step ( $0 \rightarrow \Delta x$ )

$$f(x_0, y_0) = x_0 + y_0 = 1 \quad y_1 = y_0 + \Delta x \cdot f(x_0, y_0) = 1.5 \quad x_1 = x_0 + \Delta x = 0.5$$

## Simple Example

- **Toy example:** Solve

$$\frac{dy}{dx} = x + y$$

with  $y(0) = 1$ , find  $y(1)$  using step size  $\Delta x = 0.5$ .

- ❶ Initialize  $x_0 = 0$  and  $y_0 = 1$
- ❷ First step ( $0 \rightarrow \Delta x$ )

$$f(x_0, y_0) = x_0 + y_0 = 1 \quad y_1 = y_0 + \Delta x \cdot f(x_0, y_0) = 1.5 \quad x_1 = x_0 + \Delta x = 0.5$$

- ❸ Second step ( $\Delta x \rightarrow 2\Delta x$ )

$$f(x_1, y_1) = x_1 + y_1 = 2 \quad y_2 = y_1 + \Delta x \cdot f(x_1, y_1) = 2.5 \quad x_2 = x_1 + \Delta x = 1$$

So we have  $y(1) \approx y(x_2) = 2.5$ .

## Simple Example

- **Toy example:** Solve

$$\frac{dy}{dx} = x + y$$

with  $y(0) = 1$ , find  $y(1)$  using step size  $\Delta x = 0.5$ .

- **①** Initialize  $x_0 = 0$  and  $y_0 = 1$
- **②** First step ( $0 \rightarrow \Delta x$ )

$$f(x_0, y_0) = x_0 + y_0 = 1 \quad y_1 = y_0 + \Delta x \cdot f(x_0, y_0) = 1.5 \quad x_1 = x_0 + \Delta x = 0.5$$

- **③** Second step ( $\Delta x \rightarrow 2\Delta x$ )

$$f(x_1, y_1) = x_1 + y_1 = 2 \quad y_2 = y_1 + \Delta x \cdot f(x_1, y_1) = 2.5 \quad x_2 = x_1 + \Delta x = 1$$

So we have  $y(1) \approx y(x_2) = 2.5$ .

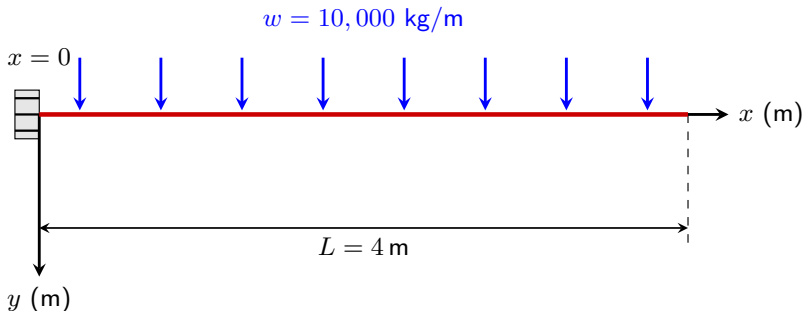
- Hint (Keep in mind!):

$$\underbrace{y_{i+1}}_{\text{next value}} = \underbrace{y_i}_{\text{current value}} + \underbrace{\Delta x}_{\text{step size}} \cdot \underbrace{f(x_i, y_i)}_{\text{sloop}} \quad x_{i+1} = x_i + \Delta x$$

## Cantilever Beam Deflection

### Engineering Task.

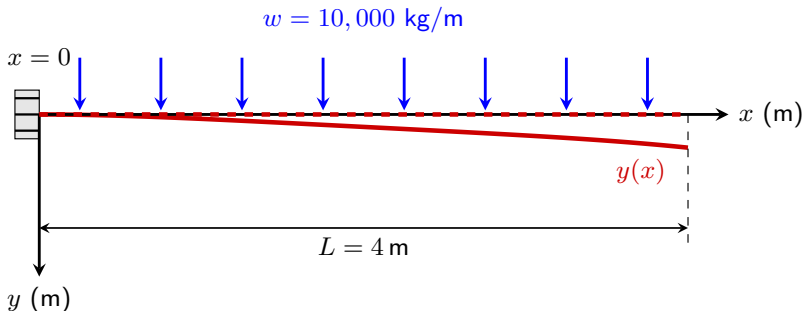
- Calculate the **deflection of a cantilever beam** under uniform load.
- Needed for: **Building codes, safety checks, material selection.**



## Cantilever Beam Deflection

### Engineering Task.

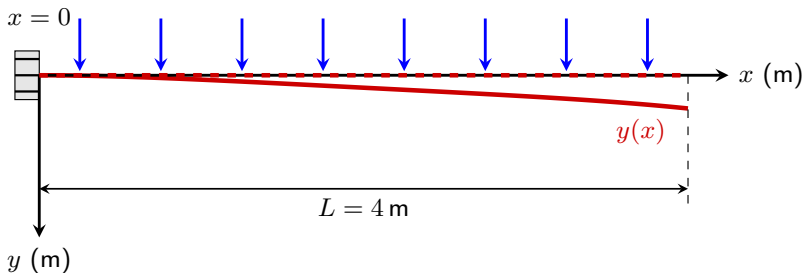
- Calculate the **deflection of a cantilever beam** under uniform load.
- Needed for: **Building codes, safety checks, material selection.**



## Cantilever Beam Deflection

- Use **Euler's method** to find deflection  $y(x)$  from  $x = 0$  to  $x = L$ .
- $y(x)$  is downward deflection at point  $x$  ( $x$  is distance from fixed end).
- **Given parameters:**
  - Uniform load:  $w = 1 \times 10^4 \text{ kg/m}$
  - Beam length:  $L = 4 \text{ m}$
  - Modulus:  $E = 2 \times 10^{11} \text{ Pa}$  (steel)
  - Moment of inertia:  $I = 3.25 \times 10^{-4} \text{ m}^4$

$$w = 10,000 \text{ kg/m}$$



## Cantilever Beam Deflection

- Use **Euler's method** to find deflection  $y(x)$  from  $x = 0$  to  $x = L$ .

$$\frac{dy}{dx} = \underbrace{\frac{w}{24 \cdot E \cdot I}}_{\text{constant}} (4x^3 - 12Lx^2 + 12L^2x)$$

- $x$  is distance from fixed end.
- $y(x)$  is downward deflection at point  $x$ .
- Given parameters:
  - Uniform load:  $w = 1 \times 10^4 \text{ kg/m}$
  - Beam length:  $L = 4 \text{ m}$
  - Modulus:  $E = 2 \times 10^{11} \text{ Pa}$  (steel)
  - Moment of inertia:  $I = 3.25 \times 10^{-4} \text{ m}^4$
- Compute the **constant factor**:

$$c = \frac{w}{24 \cdot E \cdot I} = \frac{10^4}{24 \times (2 \times 10^{11}) \times (3.25 \times 10^{-4})} = 6.41 \times 10^{-6}$$

## Euler's Method (Numerical)

- Idea:** Given the step size  $\Delta x = 0.25$  m, we start from  $y(0) = 0$  and **update the deflection** by

$$\underbrace{y_{i+1}}_{\text{next deflection}} = \underbrace{y_i}_{\text{current deflection}} + \underbrace{\Delta x}_{\text{step size}} \cdot \underbrace{\frac{dy}{dx}}_{\text{sloop}}$$

**update the position** by

$$\underbrace{x_{i+1}}_{\text{next position}} = \underbrace{x_i}_{\text{current position}} + \underbrace{\Delta x}_{\text{step size}}$$

where the sloop is given by

$$\frac{dy}{dx} = c(4x^3 - 12Lx^2 + 12L^2x)$$

- Number of steps (repeat **for** loop)

$$\frac{L}{\Delta x} = \frac{4}{0.125} = 32 \text{ steps}$$

- Compute the **constant factor**:

$$c = \frac{w}{24 \cdot E \cdot I} = \frac{10^4}{24 \times (2 \times 10^{11}) \times (3.25 \times 10^{-4})} = 6.41 \times 10^{-6}$$

## Python Programming

Compute the constant factor with Python programming.

- Given **parameters**: uniform load  $w = 1 \times 10^4$  kg/m, modulus  $E = 2 \times 10^{11}$  Pa, and moment of inertia  $I = 3.25 \times 10^{-4}$  m<sup>4</sup>.
- Compute the **constant factor**:

$$c = \frac{w}{24 \cdot E \cdot I} = \frac{10^4}{24 \times (2 \times 10^{11}) \times (3.25 \times 10^{-4})} = 6.41 \times 10^{-6}$$

```

1 import numpy as np
2
3 def const(w, E, I):
4     return w / (24 * E * I)
5
6 w = 10 ** 4           # uniform load
7 E = 2 * 10 ** 11      # modulus
8 I = 3.25 * 10 ** (-4) # moment of inertia
9 c = const(w, E, I)    # constant factor
10 print(c)

```

## Python Programming

- Update deflection and position:

$$\underbrace{y_{i+1}}_{\text{next}} = \underbrace{y_i}_{\text{current}} + \underbrace{\Delta x}_{\text{step size}} \cdot \underbrace{\frac{dy}{dx}}_{\text{sloop}} \quad \underbrace{x_{i+1}}_{\text{next}} = \underbrace{x_i}_{\text{current}} + \underbrace{\Delta x}_{\text{step size}}$$

with  $\frac{L}{\Delta x} = 16$  steps and sloop

$$\frac{dy}{dx} = c(4x^3 - 12Lx^2 + 12L^2x)$$

$$4*x**3 - 12*L*x**2 + 12*L**2*x$$

```

1 def sloop(c, L, x):
2     return c * (4*x**3 - 12*L*x**2 + 12*L**2*x)
3
4 L = 4                                # beam length
5 delta_x = 0.25                       # step size
6 n = int(L / delta_x)                 # number of steps
7 x = np.linspace(0, L, n + 1)
8 y = np.zeros(n + 1)
9 for i in range(n):
10    y[i + 1] = y[i] + delta_x * sloop(c, L, x[i])

```

## Numerical vs. Analytical Solution

Observing the first-order derivative of polynomials:

$$\frac{dy}{dx} = c(4x^3 - 12Lx^2 + 12L^2x)$$

- Write down the **analytical solution**?

$$y(x) = c(x^4 - 4Lx^3 + 6L^2x^2)$$

$$x^{**4} - 4*L*x^{**3} + 6*L^{**2}*x^{**2}$$

```

1 import numpy as np
2
3 w = 10 ** 4
4 E = 2 * 10 ** 11
5 I = 3.25 * 10 ** (-4)
6 c = const(w, E, I)
7 delta_x = 0.25
8 n = int(L / delta_x)
9 x = np.linspace(0, L, n + 1)
10 y = np.zeros(n + 1)
11 for i in range(1, n + 1):
12     y_analytical[i] = c * (x[i]**4 - 4*L*x[i]**3 + 6*L**2*x[
        i]**2)

```

## Numerical Error Analysis

**Deflection table** (numerical vs. analytical solution).

Distance $x$	Analytical solution	Numerical solution	Error
0	0	0	
0.25	0.00003688	0	
0.50	0.00014143	0.00007222	
0.75	0.00030491	0.00020763	
1.00	0.00051923	0.00039784	
1.25	0.00077687	0.00063502	
1.50	0.00107091	0.00091196	
1.75	0.00139506	0.00122206	
2.00	0.00174359	0.00155929	
2.25	0.00211140	0.00191827	
2.50	0.00249399	0.00229417	
2.75	0.00288744	0.00268279	
3.00	0.00328846	0.00308053	
3.25	0.00369434	0.00348438	
3.50	0.00410296	0.00389193	0.21 mm
3.75	0.00451285	0.00430138	0.21 mm
4.00	0.00492308	0.00471154	0.21 mm

Note: 1 meter = 1,000 millimeter (mm).

## Quick Summary

### Friday's Class:

- Apply Euler's method to engineering problem
- Compute numerical vs. analytical solutions
- Understand error accumulation
- Implement numerical methods with Python