

Applied Numerical Methods for Civil Engineering

CGN 3405 - 0002

Week 2: Mathematical Modeling & Engineering Problem Solving

Xinyu Chen

Assistant Professor

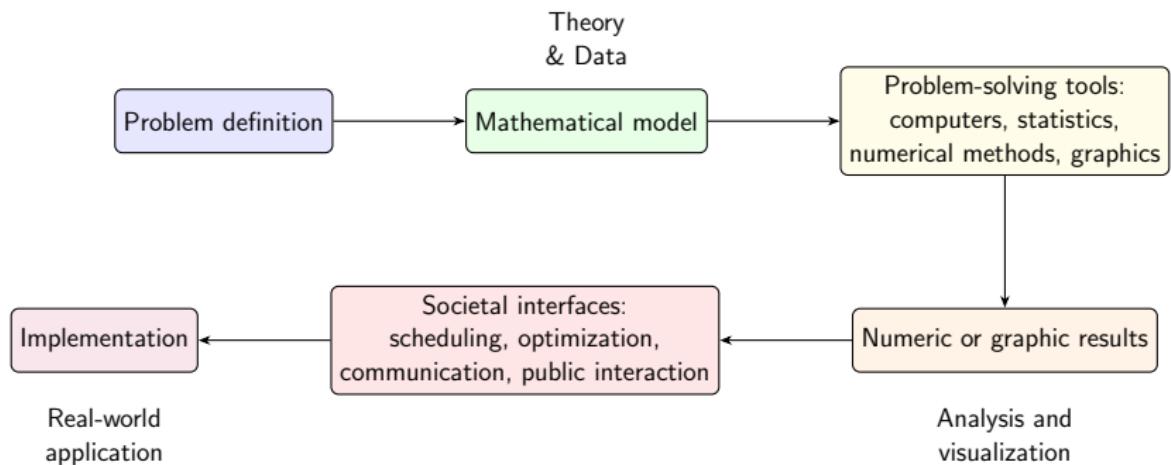
University of Central Florida

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**.

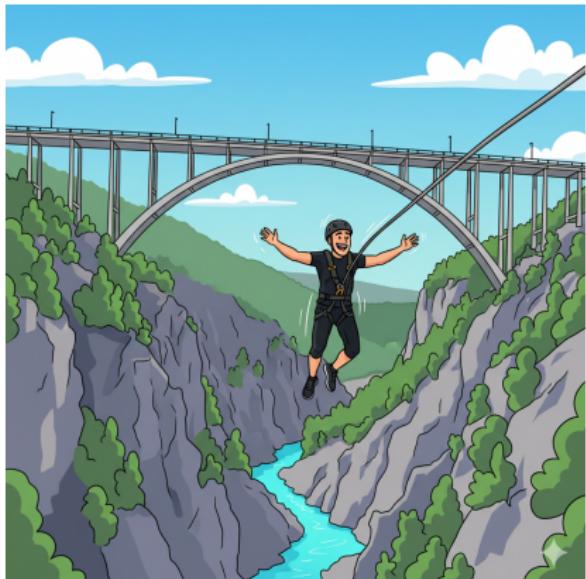
Engineering Problem Solving Process



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?
 - 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

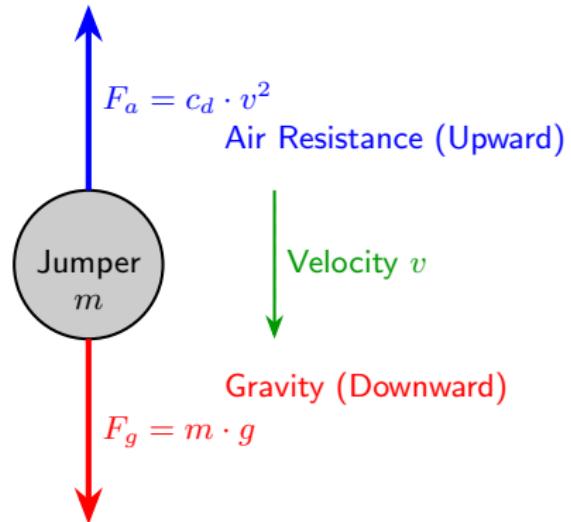
- m = mass (kg)
 - $g = 9.81 \text{ m/s}^2$, gravitational acceleration

- Air Resistance (Upward)

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

with

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



Newton's Second Law

Mathematical Model - Newton's Second Law

- From $F = m \cdot a$:

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

- 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 small discrete time steps Δt .

- Rewrite the formula of bungee jumper velocity:

$$\underbrace{v_{t+\Delta t}}_{\text{new}} = v_t + \Delta t \cdot \frac{dv_t}{dt}$$

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

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 $v = 0$
 - 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: $t = t + \Delta t$
 - Repeat!

A Real Case

Input. Mass $m = 50 \text{ kg}$, $g = 9.81 \text{ m/s}^2$, drag coefficient $c_d = 0.25 \text{ kg/m}$, and initial velocity $v_0 = 0$. (Given $\Delta t = 1 \text{ s}$)

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?

$$f(3) = f(2) + f(1) = 2 \quad f(4) = f(3) + f(2) = 3$$

$$f(5) = f(4) + f(3) = 5 \quad f(6) = f(5) + f(4) = 8$$

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

- 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 $v = 0$
 - 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: $t = t + \Delta t$
- Repeat!

```
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
10    return v
```

A Real Case

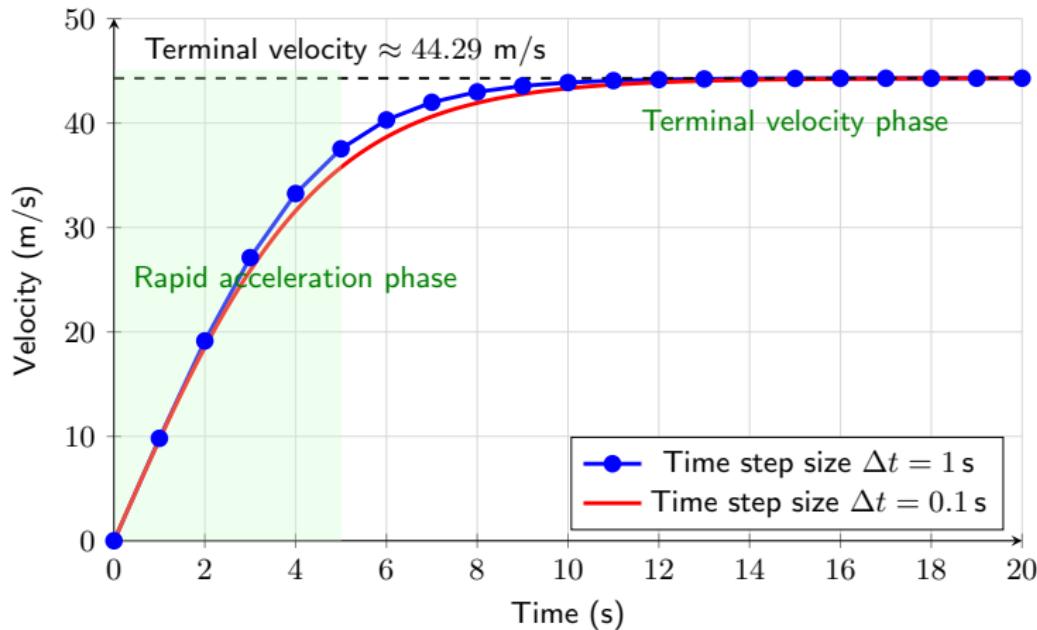
- Mass: $m = 50 \text{ kg}$
- Gravitational acceleration: $g = 9.81 \text{ m/s}^2$
- Drag coefficient: $c_d = 0.25$

```
1 import numpy as np
2
3 # Parameters
4 m = 50                      # Mass (kg)
5 g = 9.81                     # Gravity (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:

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

In this case:

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

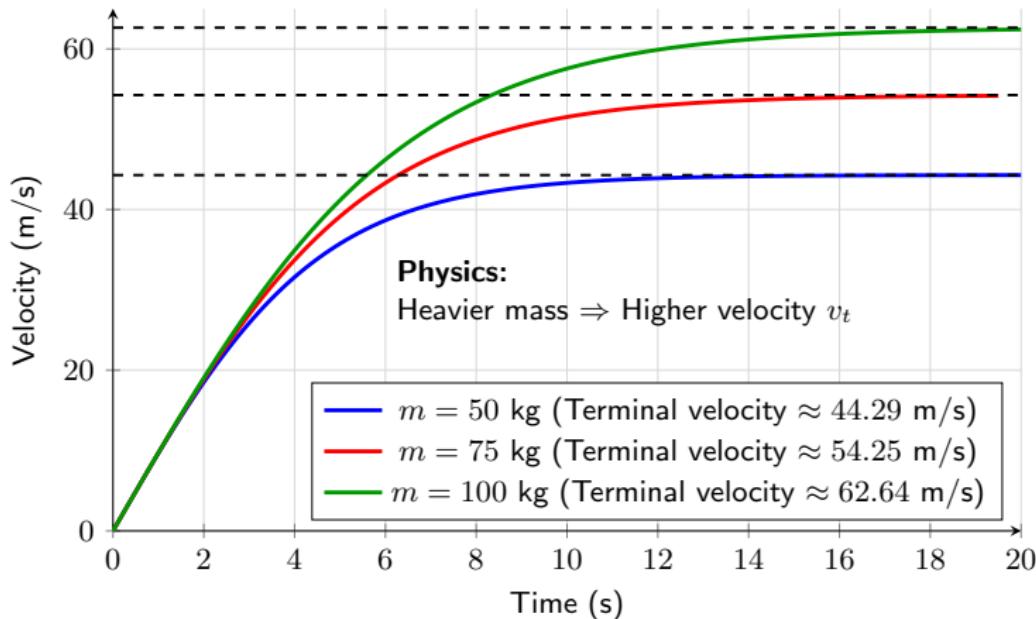
Numerical method insight.

- 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

Velocity vs. Time (Different Mass)

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

- Comparison among mass $m = 50 \text{ kg}$, 75 kg , 100 kg
- Input: $g = 9.81 \text{ m/s}^2$, and $c_d = 0.25 \text{ kg/m}$



Numerical vs. Analytical Solution

Going back to the ordinary differential equation:

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

which has solution:

$$v_t = \sqrt{\frac{mg}{c_d}} \tanh\left(\sqrt{\frac{gc_d}{m}} t\right)$$

```
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
7
8
9 delta_t = 1           # Time step size
10 t_end = 20            # Total time
11 time_steps = int(t_end / delta_t) + 1
12
13 # Computing the analytical solution
14 t = np.linspace(0, t_end, time_steps)
15 v_analytical = analytical_solution(m, g, cd, t)
```

Numerical Error Analysis

How to analyze errors?

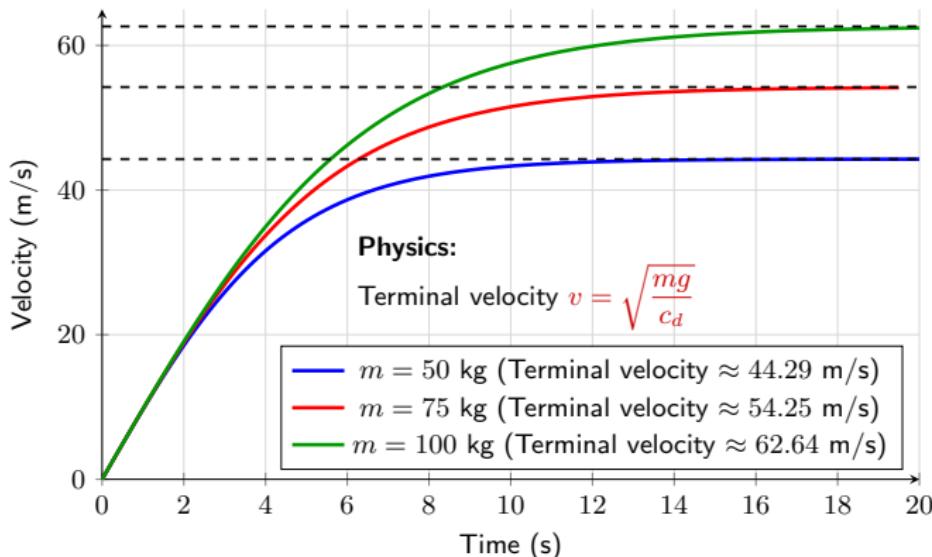
```
1 error = v - v_analytical
2 plt.plot(t, error, 'red')
3 plt.xlabel('Time (s)')
4 plt.ylabel('Error (m/s)')
5 plt.show()
```

- Why errors?
 - Euler method assumes constant acceleration over Δt .
 - 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. Computation time

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 = [50, 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 → Lower terminal velocity
- **Design implication:** Need different cords for different jumper weights!

Quizzes Now!

- **Today's participation:** 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 → 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