Applied Numerical Methods for Civil Engineering

CGN 3405 - 0002

Assignment 1: Euler's Method, Engineering Modeling, and Python Programming

Xinyu Chen

Assistant Professor

University of Central Florida

Question 1: Simple ODE

Euler's method for a simple ODE:

$$\frac{dy}{dx} = x + y, \quad y(0) = 1$$

(a) **Hand Calculation** (6 points)
Using Euler's method with a step size $\Delta x = 0.25$, compute an approximate value of $y(1)$.

- Clearly list all intermediate steps:
  - Values of x_i
  - Slopes $f(x_i, y_i)$
  - Approximations y_i

We use Euler's method: $y_{i+1} = y_i + f(x_i, y_i) \cdot \Delta x$

Step (i)	0	1	2	3	4
x_i	0	0.25	0.50	0.75	1.00
y_i	1	1.25	1.625	2.1563	2.8828
$f(x_i, y_i) = x_i + y_i$	1	1.50	2.125	2.9063	-
$y_{i+1} = y_i + f(x_i, y_i) \cdot \Delta x$	1.25	1.625	2.15625	2.8828	-

Numerical result: $y(1) \approx 2.8828$

Question 1: Simple ODE

(b) **Comparison and Interpretation** (9 points)
The exact solution of this ODE is:

$$y(x) = 2e^x - x - 1$$

- Compute the exact value of $y(1)$.
  Exact Value: $y(1) = 2e^1 - 1 - 1 = 2(2.71828) - 2 = 3.4366.$
- Compare it with your numerical result from part (a).
  Comparison: The numerical result (2.8828) is lower than the exact value (3.4366).
- Briefly comment on the numerical error and the role of step size.
  Euler's method uses a linear tangent; smaller step sizes reduce the truncation error by tracking the curve's change more frequently.
  [Key info should be included] A smaller step size Δx gives more accurate approximation.

Question 1: Simple ODE

(c) **Python Programming** (10 points)
Write a Python function that implements Euler's method for a general ODE:

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

- Use your function to approximate $y(1)$ for the given ODE.
- Verify that your Python result matches your hand calculation.
- Experiment with a smaller step size $(\Delta x = 0.1)$ and comment on the change in accuracy.

Python function:

```python
import numpy as np

def euler(x0, y0, x_end, delta_x):
    steps = int((x_end - x0) / delta_x)    # number of steps
    x = np.linspace(x0, x_end, steps + 1) # linear space
    y = np.zeros(steps + 1)
    y[0] = y0
    for i in range(steps):
        y[i+1] = y[i] + (x[i] + y[i]) * delta_x
    return x, y
```

Question 1: Simple ODE

Use Python function to approximate $y(1)$ and verify the result.

```
x0 = 0
y0 = 1
x_end = 1
delta_x = 0.25
x, y = euler(x0, y0, x_end, delta_x)
print(y[-1])
```

The output is

```
2.8828125
```

Experiment with a smaller step size $(\Delta x = 0.1)$.

```
delta_x = 0.1
x, y = euler(x0, y0, x_end, delta_x)
print(y[-1])
```

The output is

```
3.1874849202
```

Comment on $\Delta x = 0.1$: The approximation becomes closer to 3.4366 because the error per step is smaller, and the cumulative error from the exact solution is reduced.

Question 2: Bungee Jumping Velocity

A bungee jumper is modeled using Newton's second law with air resistance:

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

where:

- $m = 70$ kg
- $g = 9.81$ m/s^2
- $c_d = 0.25$ kg/m
- $v(0) = 0$

(a) **Mathematical Formulation** (6 points)
Explain briefly:

- What each term in the equation represents physically.
  m: mass; g: gravitational acceleration; c_d: drag coefficient.
- Why this problem is well-suited for a numerical method such as Euler's method.
  This is a non-linear ODE where velocity changes rapidly initially; Euler's method provides a discrete approximation of this dynamic process.

Question 2: Bungee Jumping Velocity

(b) **Numerical Solution with Euler's Method** (9 points)
Using a time step $\Delta t = 0.1$ s:

- Compute the velocity of the jumper from $t = 0$ to $t = 20$ seconds.
- Plot velocity vs. time.
- Identify:
  - The approximate terminal velocity
  - The time when the velocity first exceeds 45 m/s (if it does)

Python function:

```python
import numpy as np

def bungee_velocity(m, cd, delta_t, t_max):
    g = 9.81
    steps = int((t_max - 0) / delta_t)
    t = np.linspace(0, t_max, steps + 1)
    v = np.zeros(steps + 1) # v(0) = 0
    for i in range(steps):
        a = g - (cd/m) * (v[i]**2)
        v[i+1] = v[i] + a * delta_t
    return t, v
```

Question 2: Bungee Jumping Velocity

Compute the velocity of the jumper from $t = 0$ to $t = 20$ seconds.

```
m = 70
cd = 0.25
delta_t = 0.1
t_max = 20
t, v = bungee_velocity(m, cd, delta_t, t_max)
print(v)
```

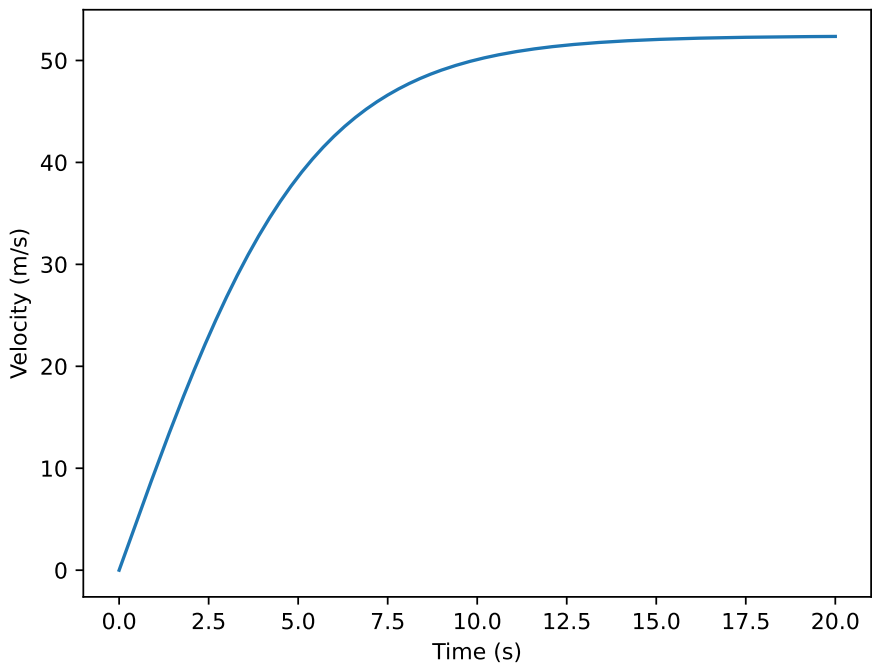
The output is

```
[ 0.   0.981  1.9616563   2.94128198  3.91919229  4.89470655
    5.86715007  6.83585598  7.80016707  8.75943757
    9.7130348   10.66034086 11.60075412 12.53369073
    13.45858594 14.3748954  15.28209625 16.17968822
    17.06719454 17.94416271 18.81016522 19.6648001
    20.5076914  21.33848947 ...
52.34910077 52.35137636 52.35356686 52.35567544 52.35770517]
```

Question 2: Bungee Jumping Velocity

Plot velocity vs. time

```
1 import matplotlib.pyplot as plt
2
3 plt.plot(t, v)
4 plt.xlabel('Time (s)')
5 plt.ylabel('Velocity (m/s)')
6 plt.savefig('velocity_mass70.pdf')
7 plt.show()
```



Question 2: Bungee Jumping Velocity

Identify the approximate terminal velocity

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

In this case:

$$v = \sqrt{\frac{mg}{c_d}} = \sqrt{\frac{70 \times 9.81}{0.25}} = 52.41\,\text{m/s}$$

```
1 v_term = np.sqrt((70 * 9.81) / 0.25)
2 print(v_term)
```

Question 2: Bungee Jumping Velocity

Identify the time when the velocity first exceeds 45 m/s

```
for i in range(t.shape[0]):
    if v[i] <= 45 and v[i + 1] > 45:
        print(t[i + 1])
```

```
6.9
```

The jumper exceeds $45\,\text{m/s}$ at approximately $t \approx 6.9$ seconds.

Question 2: Bungee Jumping Velocity

(c) **Python Programming and Safety Discussion** (10 points)

- Write a Python function that computes velocity using Euler's method.
- Modify the mass to $m = 80$ kg and repeat the simulation.
- Compare results and discuss:
  - How mass affects terminal velocity
  - Whether the jumper exceeds the safe velocity limit

Mass Comparison: Increasing mass to $80\,\text{kg}$ increases terminal velocity (to $\approx 56.03\,\text{m/s}$).

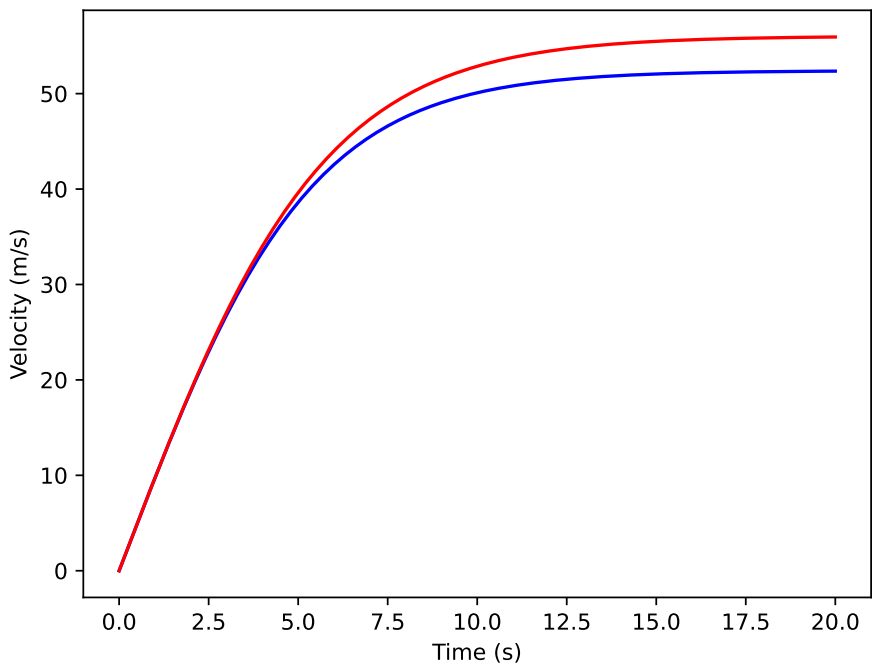
```
v_term = np.sqrt((80 * 9.81) / 0.25)
print(v_term)
```

Safety: Higher mass results in faster acceleration and a higher impact velocity, potentially exceeding safe limits more quickly.

Question 2: Bungee Jumping Velocity

```python
import matplotlib.pyplot as plt

t, v = bungee_velocity(70, 0.25, 0.1, 20)
plt.plot(t, v, 'blue')
t, v = bungee_velocity(80, 0.25, 0.1, 20)
plt.plot(t, v, 'red')
plt.xlabel('Time (s)')
plt.ylabel('Velocity (m/s)')
plt.show()
```



Question 3: Cantilever Beam Deflection

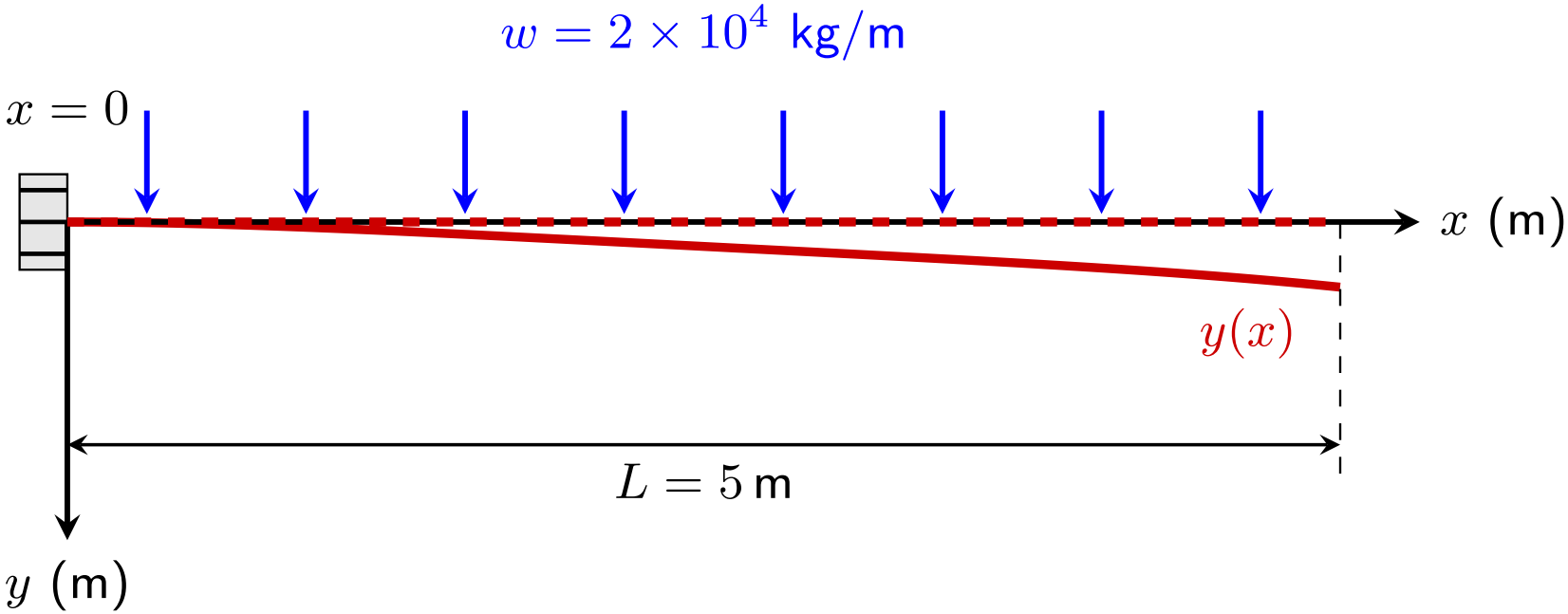
A cantilever beam of length $L = 5$ m is subjected to a uniform load. Given:

- Uniform load: $w = 2 \times 10^4$ kg/m
- Modulus of elasticity: $E = 2 \times 10^{11}$ Pa
- Moment of inertia: $I = 3.25 \times 10^{-4}$ m^4

The governing equation is:

$$\frac{dy}{dx} = c(4x^3 - 12Lx^2 + 12L^2x), \quad c = \frac{w}{24 \cdot E \cdot I}$$

with $y(0) = 0$.



Question 3: Cantilever Beam Deflection

(a) **Constant Evaluation** (6 points)

- Compute the constant c.
- Explain its physical meaning in the context of beam deflection.

$$c = \frac{w}{24 \cdot E \cdot I} = 1.282 \times 10^{-5}\,\mathrm{m}^{-3}$$

```python
import numpy as np

def const (w, E, I):
    return w / (24 * E * I)

w = 2 * 10 ** 4          # uniform load
E = 2 * 10 ** 11         # modulus
I = 3.25 * 10 ** ( -4) # moment of inertia
c = const (w, E, I)   # constant factor
print (c)
```

Meaning: This constant scales the deflection based on load and beam stiffness.

Question 3: Cantilever Beam Deflection

(b) **Euler's Method Solution** (9 points) Using Euler's method with $\Delta x = 0.125$ m:

- Compute the beam deflection $y(x)$ from $x = 0$ to $x = L$.
- Tabulate the numerical deflection values.

Python function:

```
def sloop(c, L, x):
    return c * (4*x**3 - 12*L*x**2 + 12*L**2*x)

L = 5                          # beam length
delta_x = 0.125                # step size
n = int(L / delta_x)           # number of steps
x = np.linspace(0, L, n + 1)
y = np.zeros(n + 1)
for i in range(n):
    y[i + 1] = y[i] + delta_x * sloop(c, L, x[i])
```

Question 3: Cantilever Beam Deflection

Table of the numerical deflection values

x_i	0	0.125	0.25	0.375	0.5	0.625	0.75	0.875
y_i	0	0	0.00006	0.00017	0.00034	0.00056	0.00082	0.00113
x_i	1	1.125	1.25	1.375	1.5	1.625	1.75	1.875
y_i	0.00148	0.00187	0.00230	0.00276	0.00326	0.00379	0.00434	0.00492
x_i	2	2.125	2.25	2.375	2.5	2.625	2.75	2.875
y_i	0.00553	0.00616	0.00681	0.00747	0.00816	0.00886	0.00958	0.01030
x_i	3	3.125	3.25	3.375	3.5	3.625	3.75	3.875
y_i	0.0110	0.0118	0.0126	0.0133	0.0141	0.0149	0.0157	0.0164
x_i	4	4.125	4.25	4.375	4.5	4.625	4.75	4.875
y_i	0.0172	0.0180	0.0188	0.0196	0.0204	0.0212	0.0220	0.0228
x_i	5							
y_i	0.0236							

Question 3: Cantilever Beam Deflection

(c) **Python Programming and Error Analysis** (10 points)

- Write Python code to compute the deflection numerically. [see Question 3b]
- Compute the analytical solution:

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

- Plot numerical vs. analytical solutions.
- Compute and comment on the maximum absolute error.

Python function:

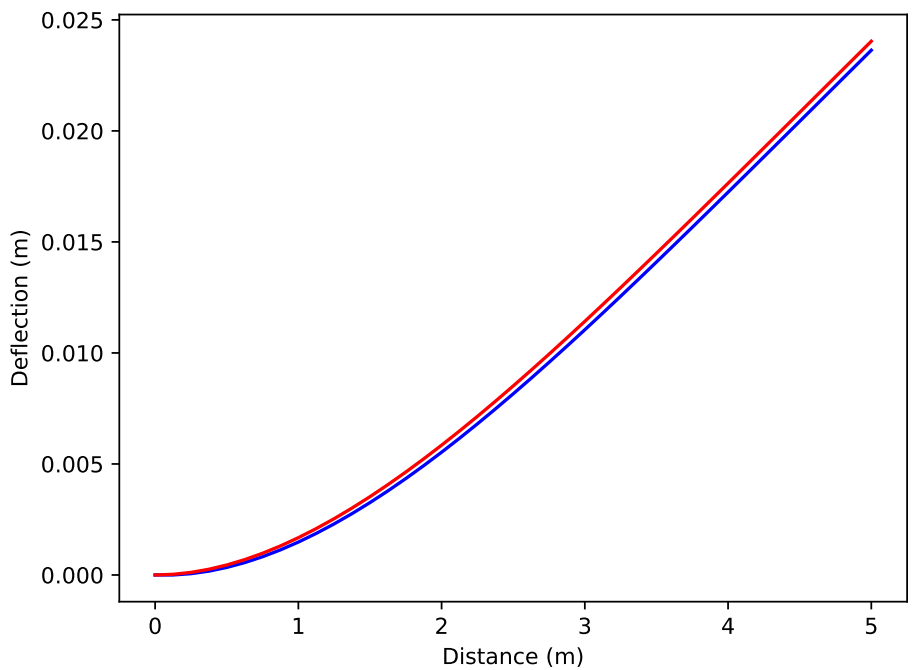
```
import numpy as np

c = const(w, E, I)
n = int(L / delta_x)
x = np.linspace(0, L, n + 1)
y_analytical = np.zeros(n + 1)
for i in range(1, n + 1):
    y_analytical[i] = c * (x[i]**4 - 4*L*x[i]**3 + 6*L**2*x[
        i]**2)
```

Question 3: Cantilever Beam Deflection

Plot numerical vs. analytical solutions

```
1 import matplotlib.pyplot as plt
2
3 plt.plot(x, y, 'blue')
4 plt.plot(x, y_analytical, 'red')
5 plt.xlabel('Distance (m)')
6 plt.ylabel('Deflection (m)')
7 plt.savefig('deflection.pdf')
8 plt.show()
```



Question 3: Cantilever Beam Deflection

Compute the maximum absolute error:

```
np.max(np.abs(y - y_analytical))
```

The output is

```
np.float64(0.0004056490384615396)
```

Error Comment: The maximum error is at the end of the beam ($x = L$). The numerical method slightly lags behind the exact solution because the slope is always calculated using the previous point's coordinates.