Solving Ordinary Differential Equations using the Euler Method

Marco Fronzi

1. Forward Difference Approximation

Let $y(t)$ be a function of a continuous variable t. The derivative of y with respect to t can be approximated using the **Forward Difference** formula:

$$\frac{dy}{dt} \approx \frac{y(t+\Delta t) - y(t)}{\Delta t} \tag{1}$$

where Δt is a small time step.

2. Euler Method for Solving ODEs

Consider a first-order ordinary differential equation (ODE):

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0 \tag{2}$$

Using the forward difference approximation from Eq. (1), the Euler update rule becomes:

$$y_{n+1} = y_n + \Delta t \cdot f(t_n, y_n) \tag{3}$$

This allows us to iteratively compute the solution at discrete time steps.

3. Python Implementation Recipe

Here is a basic Python implementation of the Euler method:

Listing 1: Euler's Method for First-Order ODE

```
# Euler's Method  y(t)
def euler_method(f, y0, t_end, dt):
    t_values = np.arange(0, t_end + dt, dt)  # Generate x values
    y_values = np.zeros(len(t_values))  # Store computed values
    y_values[0] = y0  # Initial condition

    # Apply Euler's method
    for i in range(0, len(t_values)-1):
        y_values[i+1] = y_values[i] + dt * f(y_values[i])  # Only 'y' is updated

    return t_values, y_values
```

4. Second-Order ODEs

Many physical problems involve second-order ODEs. When possible, to use Euler's method, we convert a second-order ODE into a system of first-order equations.

Consider the general form:

$$\frac{d^2y}{dt^2} = f(t, y, \frac{dy}{dt}) \tag{4}$$

We define:

$$v = \frac{dy}{dt} \tag{5}$$

so the system becomes:

$$\frac{dy}{dt} = v$$
$$\frac{dv}{dt} = f(t, y, v)$$

We can now apply the Euler method to both y and v:

$$y_{n+1} = y_n + \Delta t \cdot v_n$$
$$v_{n+1} = v_n + \Delta t \cdot f(t_n, y_n, v_n)$$

Python Example for a Spring (Simple Harmonic Oscillator)

Listing 2: Euler's Method for Second-Order ODE

```
# Euler's Method for second-order ODE (example for harmonic oscillator with f = -
    omega**2 y)

def euler_second_order(y0, v0, omega, t_end, dt):
    t_values = np.arange(0, t_end + dt, dt)
    y_values = np.zeros(len(t_values))
    v_values = np.zeros(len(t_values))
    y_values[0] = y0
    v_values[0] = v0

    for i in range(len(t_values) - 1):
        y_values[i+1] = y_values[i] + dt * v_values[i]
        v_values[i+1] = v_values[i] - dt * omega**2 * y_values[i]

    return t_values, y_values, v_values
```

5. Local and Global Errors

- **Local Truncation Error (LTE):** Error introduced in a single Euler step:

$$\text{LTE} \approx \mathcal{O}(\Delta t^2) \tag{6}$$

- **Global Truncation Error (GTE):** Accumulated error over multiple steps:

$$\text{GTE} \approx \mathcal{O}(\Delta t) \tag{7}$$

To understand why:

- Start from the Taylor expansion:

$$y(t + \Delta t) = y(t) + \Delta t \cdot y'(t) + \frac{\Delta t^2}{2} y''(t) + \mathcal{O}(\Delta t^3) \tag{8}$$

  Euler only uses the linear term, so the error per step is:

$$\text{LTE} = \frac{\Delta t^2}{2} y''(t) + \mathcal{O}(\Delta t^3) \tag{9}$$

  Hence, **local error is** $\mathcal{O}(\Delta t^2)$.

- Over $N = \frac{1}{\Delta t}$ steps, total error accumulates:

$$\text{GTE} = N \cdot \text{LTE} \approx \frac{1}{\Delta t} \cdot \Delta t^2 = \mathcal{O}(\Delta t) \tag{10}$$

6. Euler Method for Projectile Motion

Consider a projectile launched with initial velocity v_0 at an angle θ. The equations of motion are:

$$\frac{dv_x}{dt} = 0 \qquad \Rightarrow \quad v_x(t) = v_0 \cos\theta \tag{11}$$

$$\frac{dv_y}{dt} = -g \qquad \Rightarrow \quad v_y(t) = v_0 \sin\theta - gt \tag{12}$$

The Euler method is used to update position and velocity:

$$\begin{aligned} x_{n+1} &= x_n + \Delta t \cdot v_{x,n} \\ y_{n+1} &= y_n + \Delta t \cdot v_{y,n} \\ v_{x,n+1} &= v_{x,n} \\ v_{y,n+1} &= v_{y,n} - \Delta t \cdot g \end{aligned}$$

Python Code Snippet

Listing 3: Euler Method for Projectile Motion

```
import numpy as np

def euler_projectile_motion(v0, theta_deg, t_end, dt):
    theta = np.radians(theta_deg)
    vx0 = v0 * np.cos(theta)
    vy0 = v0 * np.sin(theta)

    t_values = np.arange(0, t_end + dt, dt)
    x_values = np.zeros(len(t_values))
    y_values = np.zeros(len(t_values))
    vx_values = np.zeros(len(t_values))
    vy_values = np.zeros(len(t_values))

    x_values[0] = 0.0
    y_values[0] = 0.0
```

```
    vx_values[0] = vx0
    vy_values[0] = vy0

    g = 9.81  # Acceleration due to gravity

    for i in range(len(t_values) - 1):
        x_values[i+1] = x_values[i] + dt * vx_values[i]
        y_values[i+1] = y_values[i] + dt * vy_values[i]
        vx_values[i+1] = vx_values[i]                 # Constant in ideal case
        vy_values[i+1] = vy_values[i] - dt * g        # Gravity acts on vertical
            only

        if y_values[i+1] < 0:
            # Optional: cut off remaining arrays
            x_values = x_values[:i+2]
            y_values = y_values[:i+2]
            vx_values = vx_values[:i+2]
            vy_values = vy_values[:i+2]
            t_values = t_values[:i+2]
            break

    return t_values, x_values, y_values, vx_values, vy_values
```

Errors in Projectile Motion

The Euler method introduces both positional and velocity errors, which accumulate over time. These errors become significant especially for long flight times or coarse time steps. Using smaller Δt improves accuracy but increases computational cost.

7. Summary of Errors in Euler's Method

- The method is sensitive to stiffness and instability in second-order systems.
- Better alternatives: Modified Euler, Runge-Kutta, implicit methods for stiff equations.