The Midpoint Method for Solving Ordinary Differential Equations

Marco Fronzi

1. Introduction

The **Midpoint Method** is a simple and effective second-order numerical scheme used to solve ordinary differential equations (ODEs) of the form:

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

It improves upon the basic Euler method by using an estimate of the derivative at the midpoint of the interval, rather than only at the beginning. This results in significantly improved accuracy.

2. Derivation of the Midpoint Method

Starting from the Taylor expansion of $y(t)$, we have:

$$y(t + h) = y(t) + hf(t, y) + \frac{h^2}{2} f_t(t, y) + \mathcal{O}(h^3)$$

Euler's method only keeps the first term:

$$y_{n+1} = y_n + hf(t_n, y_n)$$

The Midpoint Method instead computes a derivative estimate at the midpoint:

$$k_1 = f(t_n, y_n)$$

$$k_2 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1\right)$$

$$y_{n+1} = y_n + hk_2$$

So it uses the slope at the midpoint $(t_n + h/2, y_n + h/2 \cdot k_1)$ to update y.

3. Algorithm Summary

1. Evaluate the slope at the beginning of the interval: $k_1 = f(t_n, y_n)$
2. Estimate the midpoint value: $y_{\text{mid}} = y_n + \frac{h}{2}k_1$
3. Evaluate the slope at the midpoint: $k_2 = f(t_n + \frac{h}{2}, y_{\text{mid}})$
4. Take a full step using k_2: $y_{n+1} = y_n + hk_2$

4. Error Analysis

4.1 Local Truncation Error (LTE)

The **local truncation error** is the error made in a single step, assuming all previous steps are exact. For the midpoint method, the LTE is:

$$\text{LTE} = \mathcal{O}(h^3)$$

This comes from the neglected higher-order terms in the Taylor expansion.

4.2 Global Truncation Error (GTE)

The **global truncation error** is the accumulated error over all steps on an interval $[t_0, T]$. Since there are approximately $N = (T - t_0)/h$ steps, the global error becomes:

$$\text{GTE} = \mathcal{O}(h^2)$$

This makes the midpoint method a second-order accurate method, much more accurate than Eulers method, which has $\mathcal{O}(h)$ global error.

5. Python Pseudocode (Optional)

```
def midpoint_method(f, y0, t0, tf, h):
    t = t0
    y = y0
    ts = [t]
    ys = [y]

    while t < tf:
        k1 = f(t, y)
        k2 = f(t + h/2, y + h/2 * k1)
        y = y + h * k2
        t += h
        ts.append(t)
        ys.append(y)

    return ts, ys
```

Listing 1: Midpoint Method in Python