Euler's Method in Rust

Karan Kumar
July 5, 2025

Code Implementation

Rust Code Implementation for Euler's Method (n = 20)

```rust
fn f(t: f64, y: f64) -> f64 {
    t.cos() - y
}

fn analytical_solution(t: f64) -> f64 {
    0.5 * t.sin() + 0.5 * t.cos() + 0.5 * (-t).exp()
}

fn euler_method(n: usize) -> (Vec<f64>, Vec<f64>) {
    let t_initial = 0.0;
    let t_final = 5.0;
    let y_initial = 1.0;
    let h = (t_final - t_initial) / n as f64;

    let mut t = Vec::with_capacity(n + 1);
    for i in 0..=n {
        t.push(t_initial + i as f64 * h);
    }

    let mut y = vec![0.0; n + 1];
    y[0] = y_initial;

    for i in 0..n {
        y[i + 1] = y[i] + f(t[i], y[i]) * h;
    }

    (t, y)
}

fn main() {
    let n = 20;
    let (t, y_euler) = euler_method(n);

    for i in 0..=n {
        let y_exact = analytical_solution(t[i]);
        let error = y_euler[i] - y_exact;
        println!("{:.4}\t\t{:.6}\t\t{:.6}\t\t{:.6}", t[i], y_euler[i],
            y_exact, error);
    }
}
```

Listing 1: Euler's Method with 20 Steps, no CSV output

Rust Source Code

```rust
use std::error::Error;
use std::fs::File;
use csv::Writer;

// Function for the right-hand side of the ODE: y' = cos(t) - y
fn f(t: f64, y: f64) -> f64 {
    t.cos() - y
}

// Analytical solution: y(t) = 0.5*sin(t) + 0.5*cos(t) + 0.5*exp(-t)
fn analytical_solution(t: f64) -> f64 {
    0.5 * t.sin() + 0.5 * t.cos() + 0.5 * (-t).exp()
}

// Euler's method function for any given n
fn euler_method(n: usize) -> (Vec<f64>, Vec<f64>) {
    let t_initial = 0.0;
    let t_final = 5.0;
    let y_initial = 1.0;
    let h = (t_final - t_initial) / n as f64;

    // Time points
    let mut t = Vec::with_capacity(n + 1);
    for i in 0..=n {
        t.push(t_initial + i as f64 * h);
    }

    // Array to store y values
    let mut y = vec![0.0; n + 1];
    y[0] = y_initial;

    // Euler update
    for i in 0..n {
        y[i + 1] = y[i] + f(t[i], y[i]) * h;
    }

    (t, y)
}

fn main() -> Result<(), Box<dyn Error>> {
    let n = 1000;
    let (t, y_euler) = euler_method(n);

    // Compute analytical solution and error at each time point
    let mut y_exact = Vec::with_capacity(n + 1);
    let mut error = Vec::with_capacity(n + 1);

    for i in 0..=n {
        let ye = analytical_solution(t[i]);
        y_exact.push(ye);
        error.push(y_euler[i] - ye);
    }

    // Write to CSV in the current directory
    let mut wtr = Writer::from_writer(File::create("euler_output.csv")?);
    wtr.write_record(&["t", "y_euler", "y_exact", "error"])?;

```

```
    for i in 0..=n {
        wtr.write_record(&[
            t[i].to_string(),
            y_euler[i].to_string(),
            y_exact[i].to_string(),
            error[i].to_string(),
        ])?;
    }

    wtr.flush()?;
    println!("CSV file 'euler_output_for_n_1000.csv' written successfully.");
    Ok(())
}
```

Listing 2: Euler's Method Implementation in Rust

Output

Console Output

```
CSV file 'euler_output_for_n_1000.csv' written successfully.
```

Python Code for Plotting Absolute Error

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('euler_output_for_n_1000.csv')
df['error'] = df['error'].abs()

plt.figure(figsize=(10, 6))
plt.plot(df['t'], df['error'])
plt.xlabel('t')
plt.ylabel('Absolute Error')
plt.title('Absolute Error in Euler\'s Method (n=1000)')
plt.grid(True)
plt.show()
```

Listing 3: Python code for plotting absolute error

Plot of Absolute Error

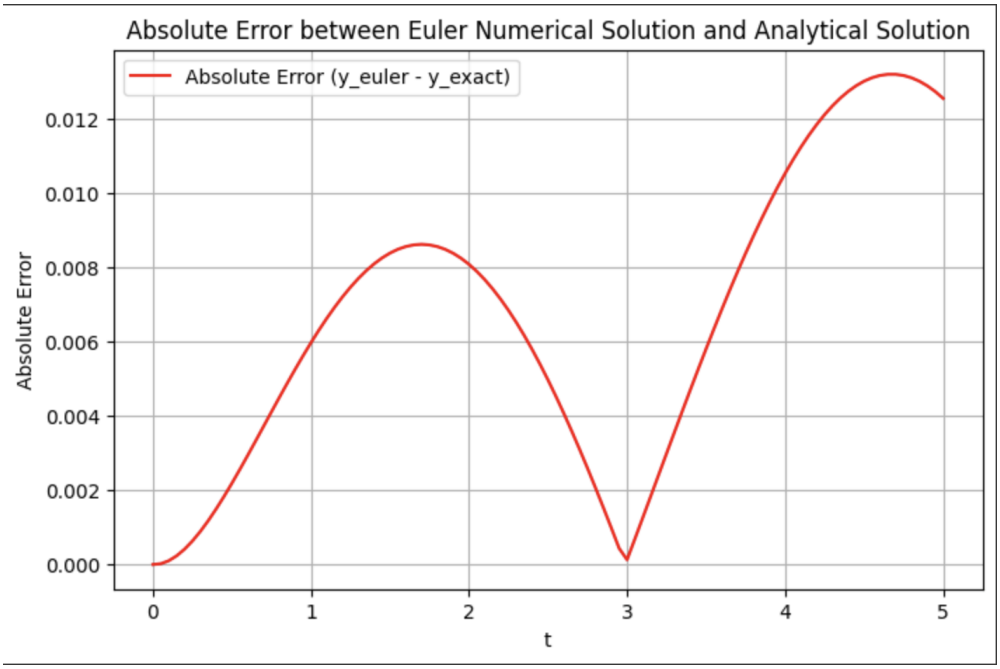

Figure 1: Absolute Error in Euler's Method for $n = 100$

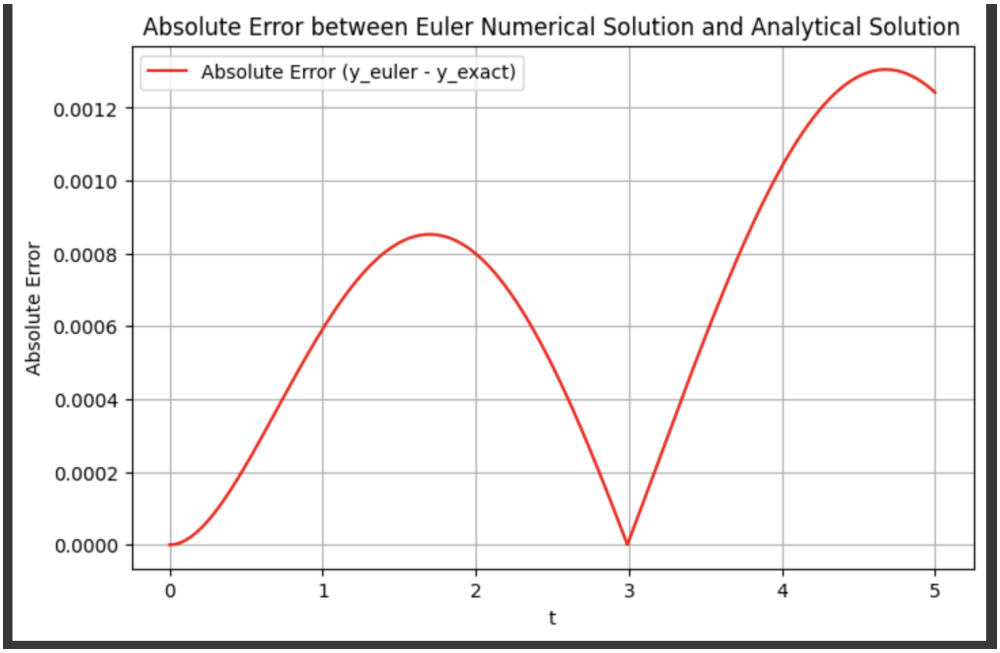

Figure 2: Absolute Error in Euler's Method for $n = 1000$