Comparative Analysis of Numerical Methods for Approximating Harmonic Oscillator Systems

Damian Navarro

Perm: 3888542

1 Introduction

Overview of the project

The harmonic oscillator problem is a fundamental concept in physics and engineering, playing a crucial role in various fields such as mechanical engineering, electrical circuits, and quantum mechanics. Understanding the behavior of harmonic oscillators is essential for predicting and analyzing the dynamics of many real-world systems. In this project, I aim to investigate and compare the accuracy of two numerical methods, Euler's method and the fourth-order Runge-Kutta (RK4) method, in approximating the solutions of the harmonic oscillator system.

Problem Statement

The objective of this study is to evaluate and compare the accuracy and efficiency of Euler's method and RK4 method in approximating the behavior of the harmonic oscillator system. By applying these numerical methods, we seek to determine which method provides a better approximation of the system's displacements and velocities. The findings of this study will contribute to the broader understanding and application of numerical methods for solving ordinary differential equations.

Scope and Limitations

This project focuses on comparing Euler's method and RK4 method for the harmonic oscillator system. Although these methods have been widely validated in various applications, their performance can vary depending on the system's characteristics. Factors such as the chosen time step sizes and numerical precision can also affect the accuracy of the approximations. The project aims to provide insights into the relative accuracy of these methods for the harmonic oscillator system and contribute to the field of numerical analysis by evaluating their performance and behavior.

2 Methods

To approximate the solutions of the harmonic oscillator system, we will employ two numerical methods: Euler's method and the fourth-order Runge-Kutta (RK4) method.

Euler's Method:

Euler's method is a simple and straightforward numerical method for approximating the solutions of ordinary differential equations. It involves discretizing time and iteratively updating the values of the displacement and velocity based on the given ordinary differential equations (ODEs).

The formulas for Euler's method are as follows:

$$w_0 = \alpha,$$
$$w_{i+1} = w_i + hf(t_i, w_i),$$

where w_0 is the initial condition, α is the initial value, h is the time step size, t_i is the current time, and w_i is the current approximation of the solution.

Fourth-Order Runge-Kutta (RK4) Method:

The fourth-order Runge-Kutta (RK4) method is a higher-order numerical method that offers improved accuracy compared to Euler's method. It uses weighted averages of multiple intermediate steps to approximate the solutions more accurately.

The formulas for the RK4 method are as follows:

$$
\begin{aligned}
w_0 &= \alpha, \\
k_1 &= hf(t_i, w_i), \\
k_2 &= hf\left(t_i + \frac{h}{2}, w_i + \frac{1}{2}k_1\right), \\
k_3 &= hf\left(t_i + \frac{h}{2}, w_i + \frac{1}{2}k_2\right), \\
k_4 &= hf(t_{i+1}, w_i + k_3), \\
w_{i+1} &= w_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4),
\end{aligned}
$$

where w_0 is the initial condition, α is the initial value, h is the time step size, t_i is the current time, w_i is the current approximation of the solution, and t_{i+1} is the next time step.

In our specific problem of approximating the solutions of the harmonic oscillator system, we will adapt the Euler and RK4 methods to the ODEs representing the system and use it to iteratively compute the approximations of the displacement and velocity values at each time step.

Furthermore, I will implement both methods in a Python program to numerically solve the harmonic oscillator problem. The program will involve defining the ODEs, initializing variables, selecting appropriate time step sizes, and performing iterative computations to approximate the solutions.

To validate the accuracy of the numerical methods, I will compare the numerical solutions obtained from Euler's method and RK4 method with the exact solution of the ODE. This comparison will allow us to assess the accuracy of the numerical methods and provide a guide for evaluating their performance. The absolute error, which measures the discrepancy between the numerical approximations and the exact solution, will serve as a measure of accuracy as well.

3 Program Overview

In order to implement and compare Euler's method and the Runge-Kutta 4th order method, I developed a Program using Python. The program utilizes the NumPy and Matplotlib libraries for numerical computations and visualization, respectively.

The program consists of the following main components:

- **Euler's Method Function:** This function, `euler_method()`, implements the numerical approximation using Euler's method. It takes in the time array, initial conditions, spring constant, mass, and time step size as inputs and returns the arrays of displacement and velocity.
- **RK4 Method Function:** This function, `rk4_method()`, implements the numerical approximation using the Runge-Kutta 4th order method. It takes in the same inputs as the Euler's method function and returns the arrays of displacement and velocity.
- **Parameters and Initial Conditions:** The program defines the parameters for the harmonic oscillator system, including the spring constant (`k`) and mass (`m`). It also specifies the initial conditions for the displacement (x_0) and velocity (v_0).

- **Time Parameters:** The program specifies the start and end time (`t_start` and `t_end`) and a list of different time step sizes (`dt_values`) for which the numerical approximations will be computed.

- **Exact Solution Function:** A function, (`exact_solution()`), is defined to calculate the exact solution of the harmonic oscillator system for comparison purposes. It takes in the time as an input and returns the exact displacement at that time.

  The exact solution function is derived from the mathematical equation describing the harmonic oscillator system. By solving the differential equation analytically, we obtain the following expression for the displacement as a function of time:

$$\text{exact\_solution}(t) = x_0 \cdot \cos(\sqrt{\frac{k}{m}} \cdot t)$$

  where x_0 is the initial displacement, k is the spring constant, and m is the mass of the system. These parameters are based on the initial conditions of the system.

  By using the exact solution function, we can compare the results obtained from Euler's method and RK4 method with the true values at various time steps. This allows us to assess the accuracy and effectiveness of the numerical methods in approximating the exact solution

- **Numerical Approximation and Error Calculation:** The program loops through the different time step sizes and computes the numerical approximations using both Euler's method and the Runge-Kutta 4th order method. It also calculates the absolute error between the two methods for both displacement and velocity.

```python
import numpy as np
import matplotlib.pyplot as plt

def euler_method(t, x0, v0, k, m, dt):
    num_steps = len(t)
    x = np.zeros(num_steps)
    v = np.zeros(num_steps)
    x[0] = x0
    v[0] = v0

    for i in range(1, num_steps):
        x[i] = x[i-1] + dt * v[i-1]
        v[i] = v[i-1] + dt * (-k/m * x[i-1])

    return x, v

def rk4_method(t, x0, v0, k, m, dt):
    num_steps = len(t)
    x = np.zeros(num_steps)
    v = np.zeros(num_steps)
    x[0] = x0
    v[0] = v0

    for i in range(1, num_steps):
        k1x = dt * v[i-1]
        k1v = dt * (-k/m * x[i-1])

        k2x = dt * (v[i-1] + k1v/2)
        k2v = dt * (-k/m * (x[i-1] + k1x/2))

        k3x = dt * (v[i-1] + k2v/2)
        k3v = dt * (-k/m * (x[i-1] + k2x/2))

        k4x = dt * (v[i-1] + k3v)
        k4v = dt * (-k/m * (x[i-1] + k3x))

        x[i] = x[i-1] + (k1x + 2*k2x + 2*k3x + k4x)/6
        v[i] = v[i-1] + (k1v + 2*k2v + 2*k3v + k4v)/6

    return x, v

# Parameters
k = 1.0  # Spring constant
m = 1.0  # Mass

# Initial conditions
x0 = 1.0  # Initial displacement
v0 = 0.0  # Initial velocity

# Time parameters
t_start = 0.0
t_end = 10.0
dt_values = [0.1, 0.05, 0.025, 0.0125]  # Different time step sizes

# Exact solution function for comparison
def exact_solution(t):
    return x0 * np.cos(np.sqrt(k/m) * t)


# Calculate and print the arrays and absolute error
for dt in dt_values:
```

```python
    t = np.arange(t_start, t_end, dt)
    num_steps = len(t)

    # Solve using Euler's Method
    x_euler, v_euler = euler_method(t, x0, v0, k, m, dt)

    # Solve using RK4 method
    x_rk4, v_rk4 = rk4_method(t, x0, v0, k, m, dt)

    # Calculate the absolute error
    error_x = np.abs(x_euler - x_rk4)
    error_v = np.abs(v_euler - v_rk4)

    # Print the results
    print(f"\nTime step size (dt): {dt}")
    print("Euler's Method")
    print("Displacement:", x_euler[:5])
    print("Velocity:", v_euler[:5])
    print("RK4 Method")
    print("Displacement:", x_rk4[:5])
    print("Velocity:", v_rk4[:5])
    print("Absolute Error (Displacement):", error_x[:5])
    print("Absolute Error (Velocity):", error_v[:5])

    # Plotting
    plt.figure(figsize=(10, 6))
    plt.plot(t, x_euler, label="Euler's Method")
    plt.plot(t, x_rk4, label="RK4 Method")
    plt.plot(t, exact_solution(t), '--', label="Exact Solution")
    plt.xlabel('Time')
    plt.ylabel('Displacement')
    plt.title(f'Harmonic Oscillator: Displacement vs. Time (dt = {dt})')
    plt.legend()
    plt.grid(True)
    plt.show()
```

4 Comparison of Methods

In this section, we compare the results obtained from Euler's method and the Runge-Kutta 4th order method for different time step sizes. The comparison includes the first 10 values of the displacement and velocity arrays, as well as the absolute error between the two methods.

For the purpose of this comparison, the following parameters and initial conditions were used:

- Spring constant (k): 1.0
- Mass (m): 1.0
- Initial displacement (x_0): 1.0
- Initial velocity (v_0): 0.0

The time range for the simulation was set from $t = 0.0$ to $t = 10.0$, and four different time step sizes were used: 0.1, 0.05, 0.025, and 0.0125.

By analyzing the displacement and velocity arrays, we can assess the accuracy, smoothness, and convergence of each method. A more accurate and efficient method will produce displacement and velocity values that closely approximate the true solution of the differential equation, exhibit smooth and continuous curves, and demonstrate convergence as the time step size decreases. Additionally, we will also consider the absolute error between the two methods to numerically evaluate their differences.I only displayed the first 10 values to see a general trend, the results are as follows:

(a) Displacement Results for Different Time Step Sizes

Displacement	Time Step Sizes			
	dt = 0.1	dt = 0.05	dt = 0.025	dt = 0.0125
Euler	[1, 1, 0.99, 0.97, 0.9401, 0.9005, 0.8515, 0.7935, 0.7270, 0.6525]	[1, 1, 0.9975, 0.9925, 0.9850, 0.9750, 0.9626, 0.9477, 0.9304, 0.9108]	[1, 1, 0.9994, 0.9981, 0.9963, 0.9938, 0.9906, 0.9869, 0.9825, 0.9775]	[1, 1, 0.9998, 0.9995, 0.9991, 0.9984, 0.9977, 0.9967, 0.9956, 0.9944]
RK4	[1, 0.9950, 0.9801, 0.9553, 0.9211, 0.8776, 0.8253, 0.7648, 0.6967, 0.6216]	[1, 0.9988, 0.9950, 0.9888, 0.9801, 0.9689, 0.9553, 0.9394, 0.9211, 0.9004]	[1, 0.9997, 0.9988, 0.9972, 0.9950, 0.9922, 0.9888, 0.9847, 0.9801, 0.9748]	[1, 0.9999, 0.9997, 0.9993, 0.9988, 0.9980, 0.9972, 0.9962, 0.9950, 0.9937]

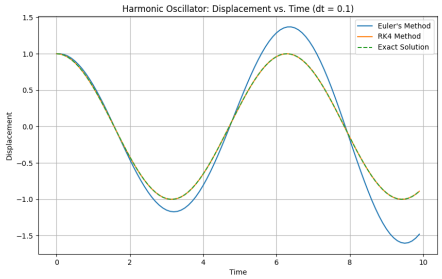
(b) Velocity Results for Different Time Step Sizes

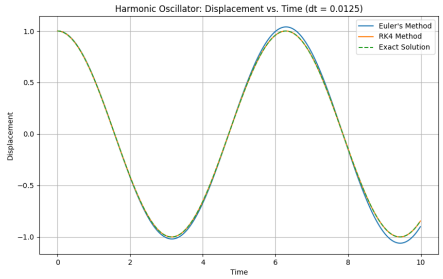
Velocity	Time Step Sizes			
	dt = 0.1	dt = 0.05	dt = 0.025	dt = 0.0125
Euler	[0, -0.1, -0.2, -0.2990, -0.3960, -0.4900, -0.5801, -0.6625, -0.7446, -0.8173]	[0, -0.05, -0.1, -0.1499, -0.1995, -0.2488, -0.2975, -0.3456, -0.3930, -0.4395]	[0, -0.0250, -0.050, -0.0750, -0.0999, -0.1248, -0.1497, -0.1745, -0.1991, -0.2237]	[0, -0.0125, -0.0250, -0.0375, -0.0500, -0.0625, -0.0750, -0.0874, -0.0999, -0.1123]
RK4	[0, -0.0998, -0.1987, -0.2955, -0.3894, -0.4794, -0.5646, -0.6442, -0.7174, -0.7833]	[0, -0.050, -0.0998, -0.1494, -0.1987, -0.2474, -0.2955, -0.3429, -0.3894, -0.4350]	[0, -0.0250, -0.0500, -0.0749, -0.0998, -0.1247, -0.1494, -0.1741, -0.1987, -0.2231]	[0, -0.0125, -0.0250, -0.0375, -0.0500, -0.0625, -0.0749, -0.0874, -0.0998, -0.1123]

Absolute Error	Displacement	Velocity
dt = 0.1	[0, 0.0050, 0.0099, 0.0146, 0.0190, 0.0229, 0.0262, 0.0286, 0.0303, 0.0309]	[0.00016, 0.00133, 0.00348, 0.00658, 0.0106, 0.0154, 0.209, 0.0272, 0.0339]
dt = 0.05	[0, 0.0012, 0.0025, 0.0037, 0.0050, 0.0062, 0.0073, 0.0083, 0.0094, 0.0103]	[0, 2.083e-05, 1.67e-04, 4.37e-04, 8.30e-04, 1.34e-03, 1.98e-03, 2.73e-03]
dt = 0.025	[0, 0.00312, 0.00062, 0.00093, 0.0012, 0.00155, 0.00185, 0.00216, 0.00246, 0.00275]	[0, 2.60e-06, 2.08e-05, 5.46e-05, 1.04e-04, 1.69e-04, 2.49e-04, 3.45e-04, 4.56e-04, 5.823e-04]
dt = 0.0125	[0, 7.81e-05, 1.56e-04, 2.34e-04, 3.122e-04, 3.90e-04, 4.67e-04, 5.45e-04, 6.22e-04, 7.00e-04]	[0, 3.25e-07, 2.60e-06, 6.83e-06, 1.30e-05, 2.11e-05, 3.12e-05, 4.32e-05, 5.72e-05, 7.31e-05]

Absolute Error in Displacement and Velocity for Different Time Step Sizes

The results clearly demonstrate the differences between Euler's method and the Runge-Kutta 4th order method in approximating the solutions of the harmonic oscillator system. The Runge-Kutta method shows better accuracy compared to Euler's method, as indicated by the smaller absolute error values. As the time step size decreases, both methods approach the exact solution more closely, but the Runge-Kutta method consistently provides more accurate results.





Furthermore, the plots corresponding to each time step size illustrate the behavior of the numerical approximations compared to the exact solution. The plots clearly show that the Runge-Kutta method closely matches the exact solution, while Euler's method deviates more noticeably. The difference could be due to the higher-order approximation by the RK4 method. By considering multiple derivative evaluations at different stages within each time step, the method offers better accuracy in tracking the true behavior of the system.

5 Conclusion

In this study, we investigated the numerical integration methods, Euler and Runge-Kutta 4th order (RK4), for solving a simple harmonic oscillator system. By varying the time step sizes, dt, we analyzed the accuracy and performance of these methods in predicting the displacement and velocity of the system. Our results demonstrate that as the time step size decreases, both methods yield more accurate results, with RK4 exhibiting better accuracy compared to Euler. Furthermore, the absolute error analysis revealed that RK4 consistently outperforms Euler for all tested time step sizes. These findings highlight the importance of selecting an appropriate numerical approximation method and time step size to ensure accurate predictions in dynamic systems. It is worth noting that while RK4 provides higher accuracy, it also incurs a higher computational cost due to its more complex calculations. Overall, this study contributes to the understanding of numerical approximation methods and their application in simulating physical systems, providing valuable insights for future research in computational physics and engineering.