Report for Mathematical Modeling Lab Group Project

Group Members:

- Giorgi Ambokadze
- Luka Burduli

Date of the experiment: 14.09.2023

Constructor University Bremen

Abstract

This report presents the solutions and observations for the mathematical modeling lab project, which includes analyzing a damped harmonic oscillator and solving a boundary value problem (BVP). Various numerical methods was used to evaluate effectiveness of numerical methods, and analyze the results. The explicit Euler method, the implicit Euler method, and the solve_ivp function from SciPy were used to analyze a damped harmonic oscillator. The accuracy and stability of these methods for different damping coefficients (ζ) and time steps (Δt) was observed. For Boundary value problem shooting and finite difference methods were used and additionally solve_bvp function SciPy was implemented. The results are visualized through various plots, including velocity over coordinate, velocity over time, coordinate over time, and error over time steps.

Part 1: damped harmonic oscillator

Introduction

Numerical methods are essential for solving differential equations that cannot be solved analytically. This problem focuses on the damped harmonic oscillator, a second-order differential equation which describes the motion of a mass-spring-damper system. The damped harmonic oscillator is characterized by the equation:

$$\ddot{x} + 2\zeta\dot{x} + x = 0 \tag{1}$$

Where x is the displacement, $\dot{x}$ is the velocity, ζ is the damping coefficient, and $\ddot{x}$ is the acceleration. We compare three numerical methods: explicit Euler, implicit Euler, and solve_ivp from SciPy, to understand their performance in terms of accuracy and stability.

Theoretical Background

Given the second-order differential equation 1, We can introduce a new variable $y = \dot{x}$, so that the equation becomes system of two equations:

$$\begin{cases} \dot{x} = y \\ \dot{y} = -2\zeta y - x \end{cases} \tag{2}$$

The explicit Euler method is a simple first-order method that uses the forward difference approximation. We discretize the time interval into smaller time steps, denoted as Δt. Starting from the initial condition at time t=0, we aim to find the solution at successive time points. At each time step, we update the solution using the following formula:

$$x_{n+1} = x_n + \Delta t \cdot y_n \tag{3}$$

$$y_{n+1} = y_n - \Delta t \cdot (2\zeta y_n + x) \tag{4}$$

Here, x_n and y_n represent the position and velocity at time step n, respectively. ζ is the damping coefficient. The explicit Euler method is simple to implement but may not be very accurate, especially for stiff systems or large time steps. It's conditionally stable, meaning the step size needs to be small enough to prevent instability, particularly for stiff systems or large damping coefficients.

The implicit Euler method, on the other hand, uses the backward difference approximation. Similar to the explicit method, we discretize the time interval into smaller time steps, denoted as Δt.

At each time step, instead of using the derivative at the current time step to update the solution, the implicit Euler method uses the derivative at the next time step. The update formula becomes:

$$x_{n+1} = x_n + \Delta t \cdot y_{n+1} \tag{5}$$

$$y_{n+1} = y_n - \Delta t \cdot (2\zeta y_{n+1} + x_{n+1}) \tag{6}$$

The implicit Euler method is unconditionally stable, meaning it can handle larger time steps without encountering stability issues compared to the explicit method.

The solve_ivp function from SciPy offers efficient solution for initial value problems (IVPs) in ordinary differential equations (ODEs). It uses advanced numerical integration methods and provides high accuracy and efficiency across a wide range of problems. In our study, the solve_ivp function was used to compare the results obtained from other numerical approximations.

Results and Error analysis

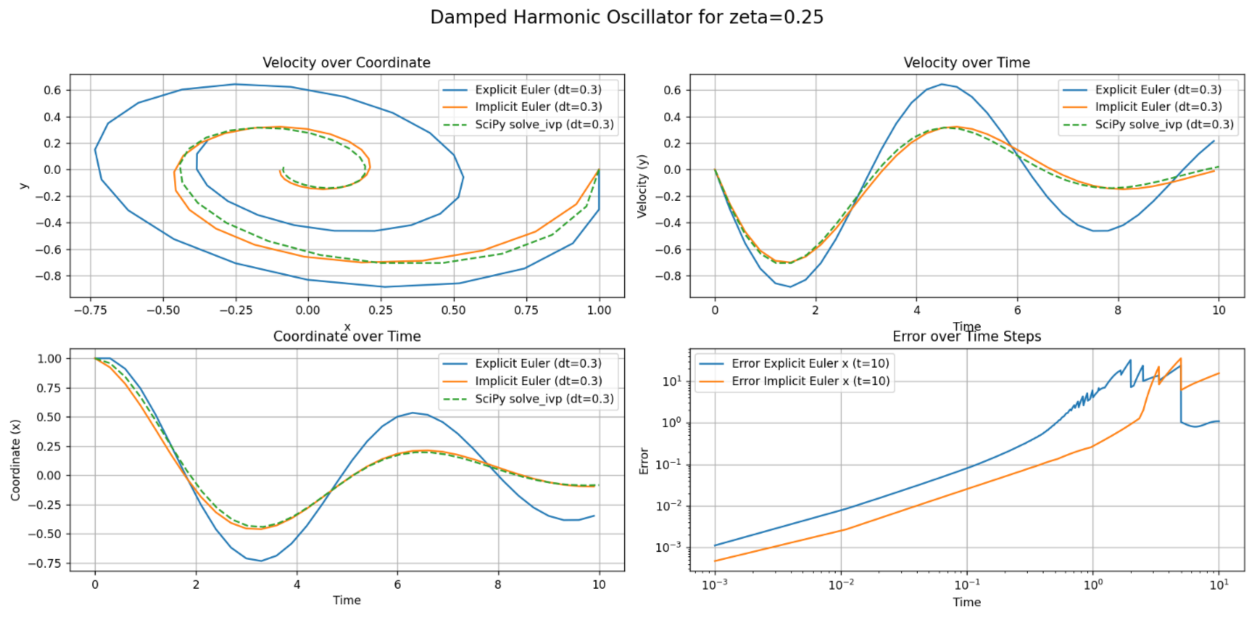

Figure 1: This graph represents damped harmonic oscillation solutions for $\zeta = 0.5$

For velocity dependance on coordinate (**figure 1 graph 1**), implicit Euler method and SciPy solver results align well for dt=0.3, when explicit method has higher error. Velocity and coordinate functions damp through time and again, implicit Euler method has higher accuracy. Errors in the explicit Euler method increase significantly with larger time steps. The implicit Euler method maintains low errors, showcasing its robustness. After dt=0.5 time step error becomes unstable for the explicit Euler method. Same occurs for the implicit method but dt=2.3.

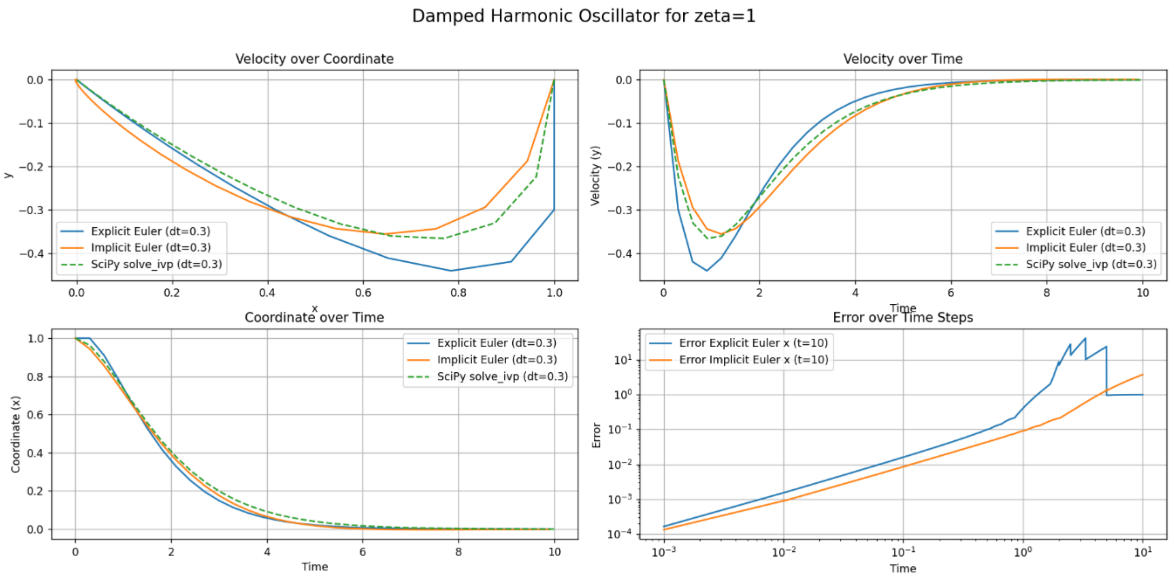

Figure 2: This graph represents damped harmonic oscillation solutions for $\zeta = 1$

Increased damping leads to rapid convergence towards zero velocity. The explicit Euler method fails to capture the damping accurately for larger time steps after dt=1.7. The implicit Euler method and solve_ivp provide consistent results, showing the system's damping effect. Same case is for figure 3. But this time at dt=0.3 time step error appears in the range of 10^3.

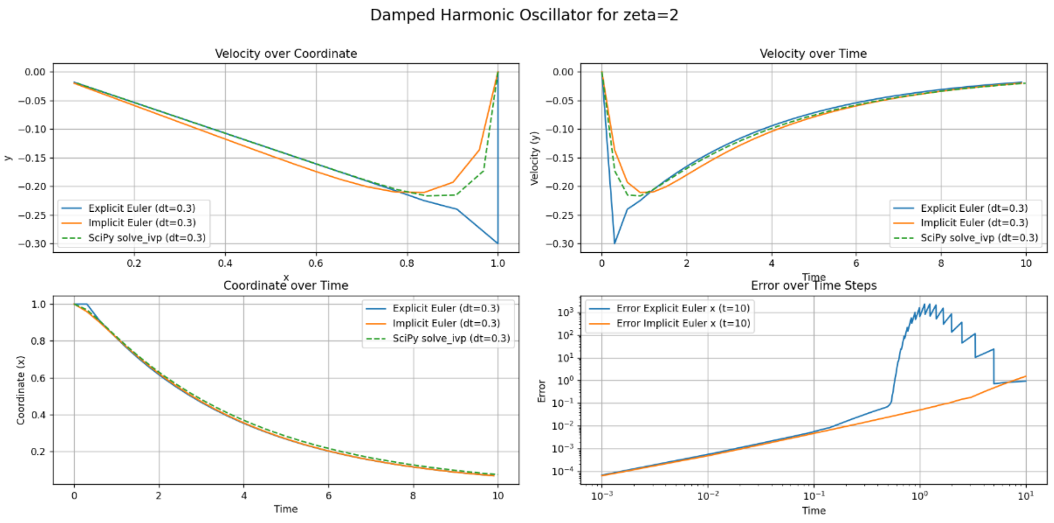

Figure 3: This graph represents damped harmonic oscillation solutions for $\zeta = 2$

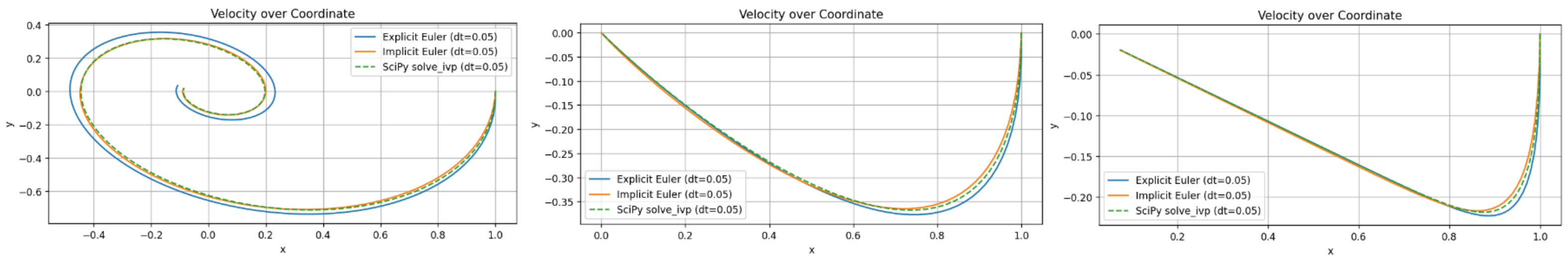

Figure 4: Velocity over Coordinate for different damping coefficients (zetas).

If we reduce time step size for implicit and explicit Euler methods, (this time it was decreased form dt=0.3 to dt=0.05) velocity over coordinate graphs are getting more accurate so the error is decreasing. Same result could be obtained by observing "Error over time steps" graphs of the following figures: figure 1 graph 4, figure 2 graph 4, figure 3 graph 4.

Part 2: damped harmonic oscillator

Introduction

Boundary value problems (BVPs) represent situations where the state of a system is defined by conditions at multiple points. These problems differ from initial value problems (IVPs), where conditions are specified only at a single point. We focus on a specific second-order linear differential equation with given boundary conditions, aiming to explore and apply three different numerical methods to solve it.

The equation under consideration is the same:

$$\ddot{x} + 2\zeta\dot{x} + x = 0 \tag{7}$$

With $\zeta = 0.25$ and boundary conditions:

$x(0) = 1, \qquad x(9) = 0, \qquad t\epsilon[0,9]$

To solve this BVP, we will employ the shooting method, the finite difference method, and the `scipy.integrate.solve_bvp` function in Python. Each method offers advantages and challenges, providing an understanding of numerical solutions to boundary value problems.

Theoretical Background

- Shooting Method

The shooting method is an intuitive approach that transforms the BVP into an initial value problem (IVP). The main idea is to guess the initial conditions, solve the resulting IVP, and iteratively adjust the guesses until the boundary conditions are satisfied.

To apply the shooting method to our problem, we start by guessing the initial slope $\dot{x}(0)$. Using this guess, we integrate the differential equation from $t = 0$ to $t = 9$. The solution at $t = 9$ is then compared to the boundary condition $x(9) = 0$. Based on the difference, we adjust the initial slope and repeat the process.

The shooting method is straightforward and works well for linear problems. However, it can become complicated for non-linear equations or systems with stiff behavior, which requires more complex root-finding and integration techniques.

- Finite Difference Method

The finite difference method converts the continuous domain into a grid and approximates the derivatives in the differential equation using finite difference formulas. This transforms the differential equation into a system of algebraic equations, which can be solved using linear algebra techniques.

To implement the finite difference method for our BVP, we divide the interval [0,9] into N equally spaced points. We approximate the second derivative $\ddot{x}$ and the first derivative $\dot{x}$ at these points using central difference formulas. Substituting these approximations into the differential equation gives us system of linear equations, which can be written in matrix form.

Solving this system provides the values of x at the grid points, approximating the continuous solution. The finite difference method is powerful for handling complex boundary conditions and non-linear problems. However, it requires careful attention to discretization errors and stability considerations, particularly for stiff equations.

- scipy.integrate.solve_bvp

The `scipy.integrate.solve_bvp` function in Python provides automated framework for solving BVPs. It uses collocation methods, where the solution is approximated by a set of basis functions, and the coefficients are determined to satisfy the differential equation and boundary conditions.

To use `scipy.integrate.solve_bvp`, we define the differential equation and boundary conditions as Python functions.

The `solve_bvp` function is robust and capable of handling a wide range of BVPs, including those with non-linear behavior and complex boundary conditions.

Results and Error analysis

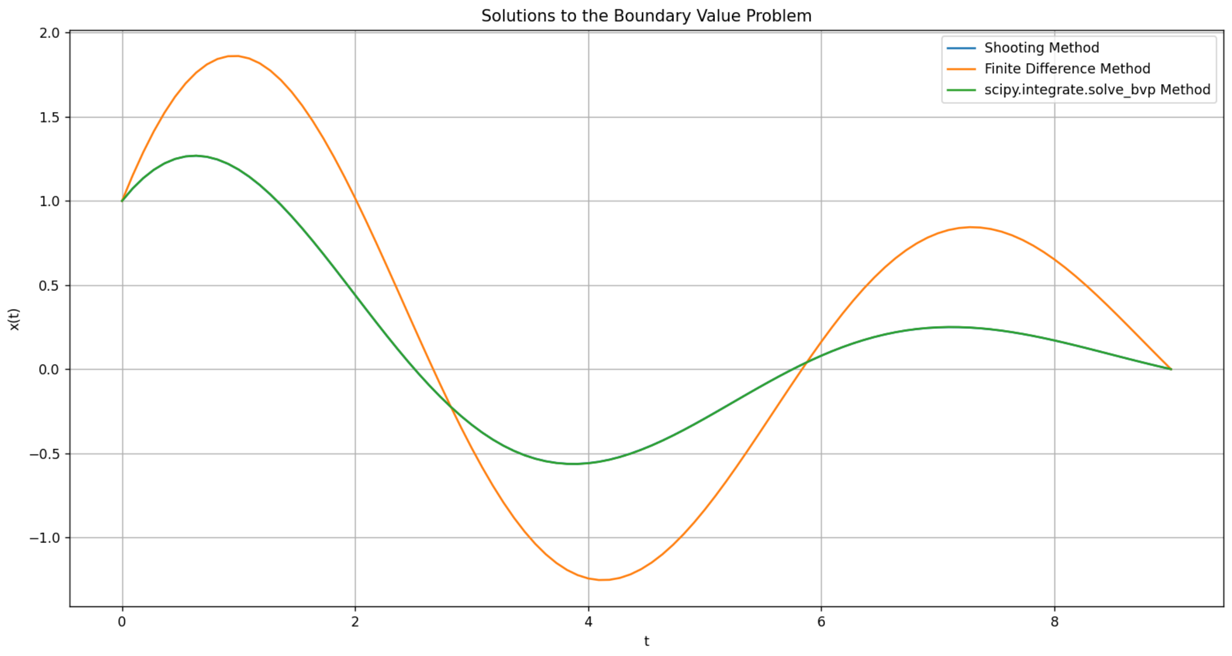

Figure 5: Solutions to the Boundary Value Problem

The numerical solutions to the boundary value problem $\ddot{x} + 2\zeta\dot{x} + x = 0$ with $\zeta = 0.25$ and boundary conditions $x(0) = 1$ $x(9) = 0$ were obtained using the Shooting Method, the Finite Difference Method, and the `scipy.integrate.solve_bvp` method. The results are presented in the figure 5.

The Shooting Method's solution is nearly identical to that of the scipy.integrate.solve_bvp method, indicating high accuracy. The root-finding process for the initial slope ensures that the boundary conditions are satisfied precisely, resulting in minimal error throughout the interval. The close agreement with the scipy.integrate.solve_bvp solution confirms the effectiveness of the Shooting Method for this problem.

The Finite Difference Method introduces discretization errors due to the finite grid size. These errors are more pronounced compared to the other methods, leading to deviations in the solution, especially in the mid-range values of t. The error can be attributed to the approximation of the derivatives and the linear system's solution, which may not capture the problem's dynamics as accurately. Refining the grid (i.e., increasing the number of points) could reduce these errors but at the cost of increased computational complexity.

If we increase N (number of number of grid points used to approximate the continuous solution) from 100 to 1000, the results might not change significantly due to the method already capturing the best possible solution at N=100.

The scipy.integrate.solve_bvp method demonstrates minimal error, producing a highly accurate solution. Its collocation approach ensure that the boundary conditions are met with high precision and that the solution curve is smooth and accurate. This method serves as a meter for the other numerical solutions, indicating the expected behavior of the solution to the BVP.

Discussion and conclusion

Damped Harmonic Oscillator

In analyzing the damped harmonic oscillator, the explicit Euler method, implicit Euler method, and `solve_ivp` function were employed to evaluate accuracy and stability. The explicit Euler method, though simple to implement, causes significant limitations, for example for larger time steps and higher damping coefficients. Errors increased when the time step $\Delta t \geq 0.5$, demonstrating its conditional stability and unsuitability for stiff systems.

On the other hand, the implicit Euler method showed more stability, handling larger time steps without instability. Its results closely aligned with the from the `solve_ivp` function. The `solve_ivp` function provided highly accurate and stable results across various damping coefficients and time steps, serving as a reliable benchmark for the Euler methods.

In conclusion, for the damped harmonic oscillator, the implicit Euler method and `solve_ivp` were found to be reliable and effective, particularly for stiff systems and larger damping coefficients. The explicit Euler method has errors for complex scenarios, so it is important to choose appropriate numerical methods based on the problem’s characteristics.

Boundary Value Problem (BVP)

For the boundary value problem, the shooting method, finite difference method, and `scipy.integrate.solve_bvp` function were used. The shooting method transformed the BVP into an initial value problem, provided accurate results which closely matched those from `solve_bvp`, ensuring precise boundary condition satisfaction with minimal errors.

The finite difference method introduced discretization errors due to the finite grid size, leading to deviations, especially in the mid-range values. While increasing grid points could reduce errors, it would also raise computational complexity. This method, though powerful for complex conditions, demands careful attention to discretization errors and stability.

The `solve_bvp` function demonstrated minimal errors and high accuracy, validating the other methods' results.

In conclusion, the `solve_bvp` function and shooting method were effective for solving the BVP, providing accurate and reliable solutions.

References

Sauer, T. (2011). Numerical Analysis. Pearson Education. *[1]*

https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_ivp.html *[2]*

https://matplotlib.org/stable/index.html *[3]*