Homework 5 - Adaptive PECE Solver

Overview

This repository contains a simple implementation of a PECE solver using the 2nd-order Adams-Bashforth method as the predictor and the 2nd-order Adams-Moulton method as the corrector. The source code files include:

- `include/ode.h`: Defines the data structures representing the ODE (system). The 1st~3rd derivatives are provided to the constructors as C++ `std::function` objects. To reduce redundant code, static functions for constructing ODEs used in Part 2 are provided (`PredatorPrey`, `VanDerPol` and `MethodOfLines`).
- `include/solver.h`: Defines the data structure for the ODE solver. There are two overloads to the `Solver<dim>::Solve` function, which implement the non-adaptive and adaptive versions of the solver respectively.
- `include/plot.h`: Enables exporting solution data to be read by a uniform Python script and plotted using `matplotlib`.
- `src/main.cpp`: Solves the problems required by each part of the homework as different testcases.
- `script/plot.py`: Uses `matplotlib` to plot the data exported by the C++ program.

Third-party libraries are included as git submodules in `extern`. The `data` directory contains the solution and step size data exported as JSON files and `asset` directory contains the corresponding images of the plots. To build the repository, run `git submodule update --init --recursive` in the root directory to download the dependencies and use `cmake` to build the code.

Implementation Details

How to find starting values?

The PECE method used here requires f_{n-1} and f_n to predict y_{n+1}. To obtain f_{n-1}, we use:

- the midpoint method to estimate the point before the starting point to obtain y_{-1} and f_{-1} when adaptivity is not required
- the forward Euler method with step size estimated using the error tolerance and y_0'' when variable step size is required

The other starting values including y_0, t_0 and the tentative step size h_0 should be provided by the user as the `Solver<dim>::StartingValues` structure.

How to store past values?

During the iteration, the past values needed to be stored are:

- f_{n-1}, stored as `f_prev`
- h_{n-1}, stored as `h_prev`

For more details please see the implementation of `Solver<dim>::Solve`.

How to interpolate for off-step points?

Suppose the off-step point is located between `t_curr` and `t_next`, we use the following method for interpolation:

$$\hat{y}(t_n + h') = y_n + f_{n+1}h' + \frac{(f_{n+1} - f_n)h'^2}{2h_n}$$

How to estimate the error?

Since the PECE method should have the same LTE as the corrector method, we estimate the error at t_{n+1} as:

$$||\frac{1}{12}h_n^3 y'''(t_{n+1})||_2$$

If the error estimation is smaller than the error tolerance, the current step would be accepted and ready for interpolation.

How to choose the next step size or change the current step size?

We use the step size anticipated to make the error estimation equal to $0.9 \times \mathrm{tol}$ both when choosing the next step size and changing the current step size. In our case, this is

$$h_{\mathrm{next}} = h_{\mathrm{curr}}(\frac{0.9 \times \mathrm{tol}}{\hat{\mathrm{err}}})^{1/3}$$

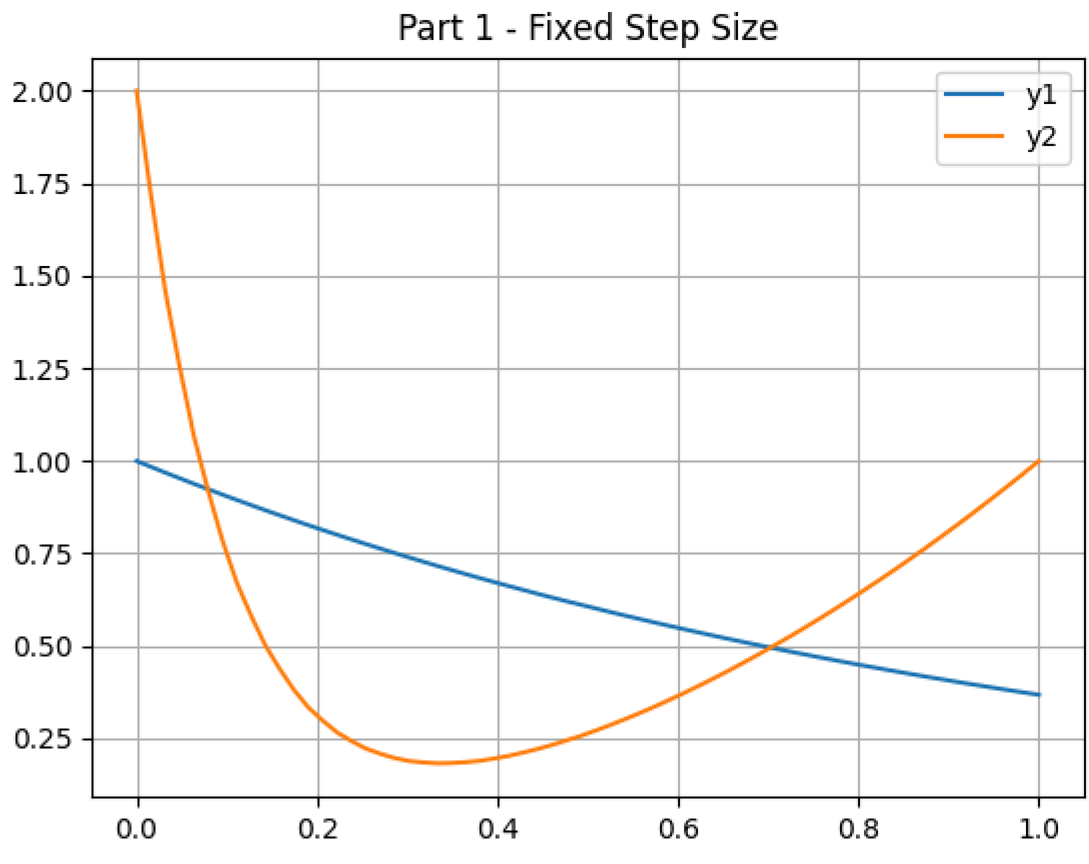
When predicting the next step size, we use the error estimation at current step point; when changing the current step size, we use the error estimation at the rejected step point.
When $||f'''(t_{\mathrm{curr}})||_2 = 0$, we keep the step size unchanged for the next step.

Usage

To run this code, first make sure you have cloned the submodules and then use `cmake` to build the project.
Run `pece.exe` in the root directory of this repository, and the data would be generated in the `data` directory. Then you can run `python script/plot.py` to view the plots. To save these plots as images, run `python script/plot.py --save`.

Results

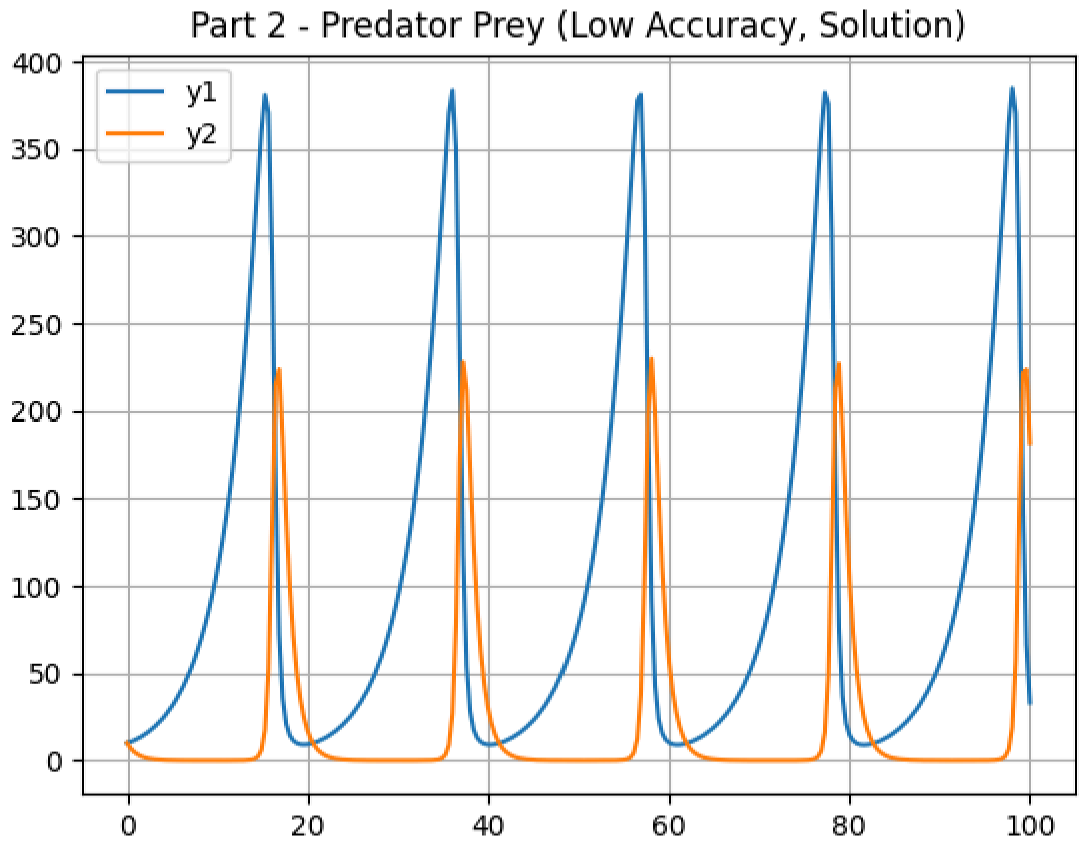
Part 1 - No Adaptivity

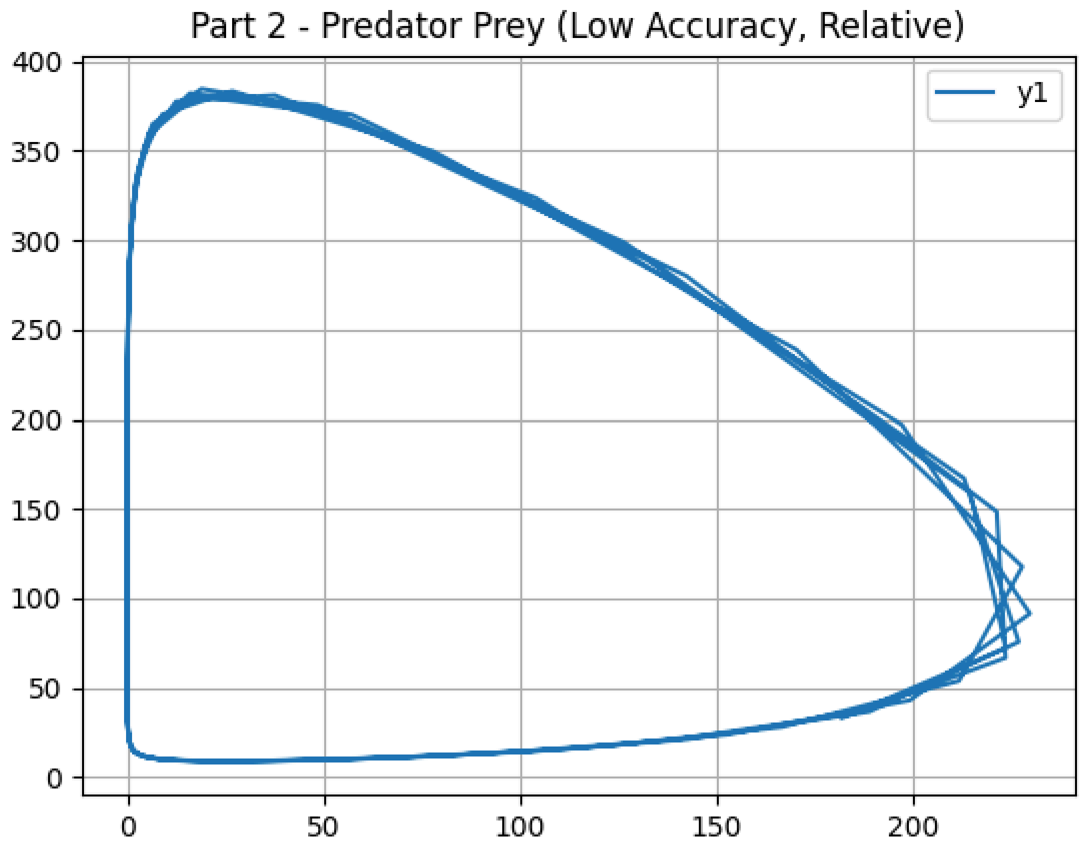


Part 2 - Adaptable Step Size

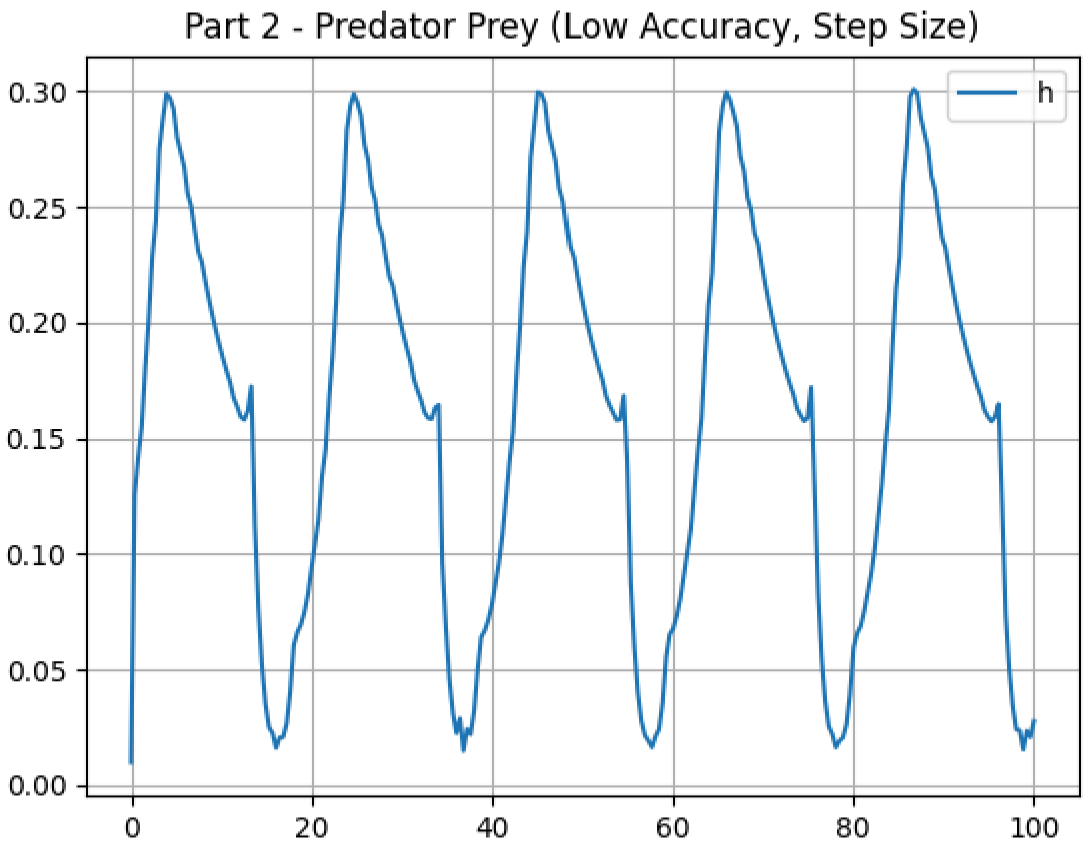
Predator Prey Problem

The solutions with error tolerance set to 1×10^{-3}:

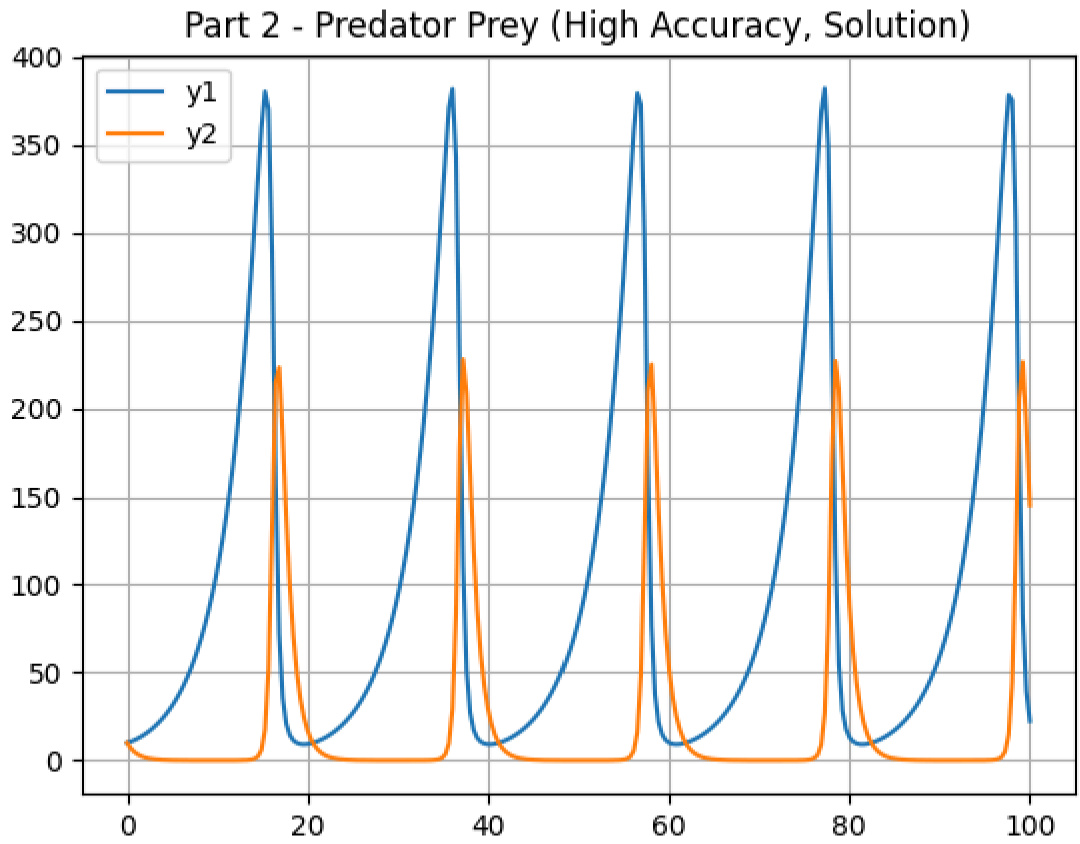

Part 2 - Predator Prey (Low Accuracy, Solution)
y1
y2
400
350
300
250
200
150
100
50
0
0
20
40
60
80
100

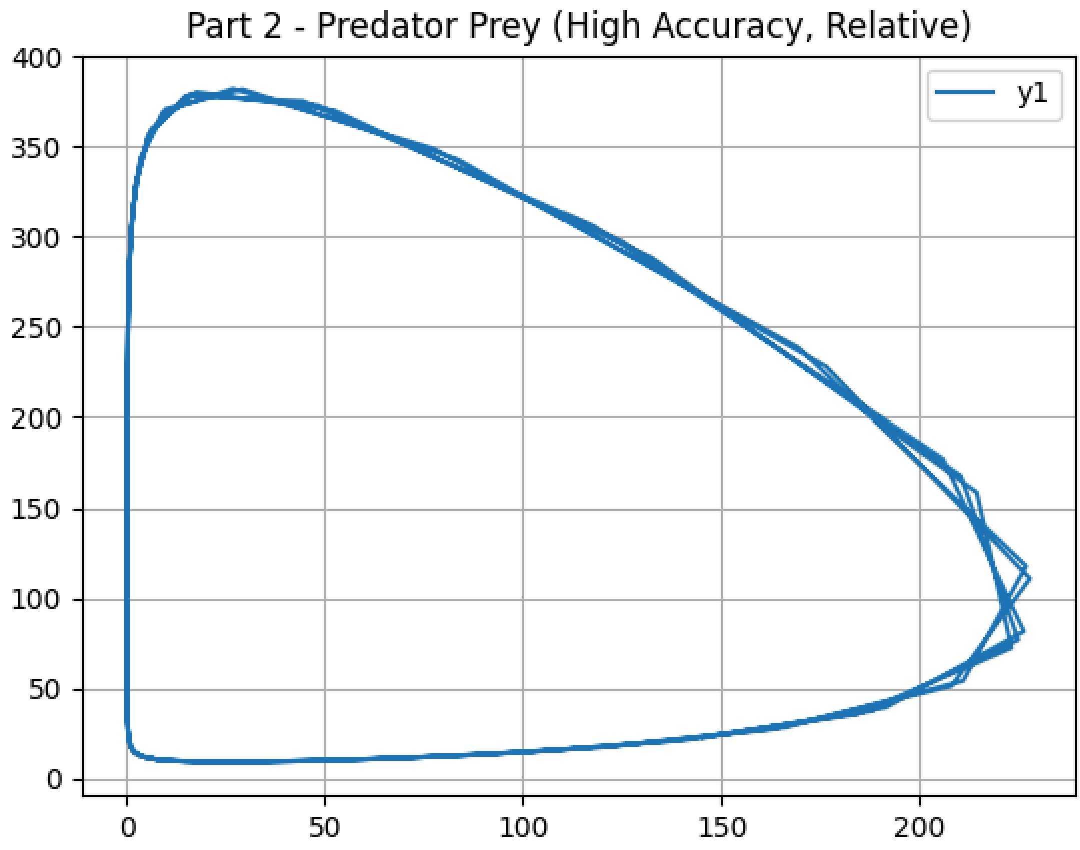

Part 2 - Predator Prey (Low Accuracy, Relative)
y1
400
350
300
250
200
150
100
50
0
0
50
100
150
200

The step sizes used around each interpolating points:

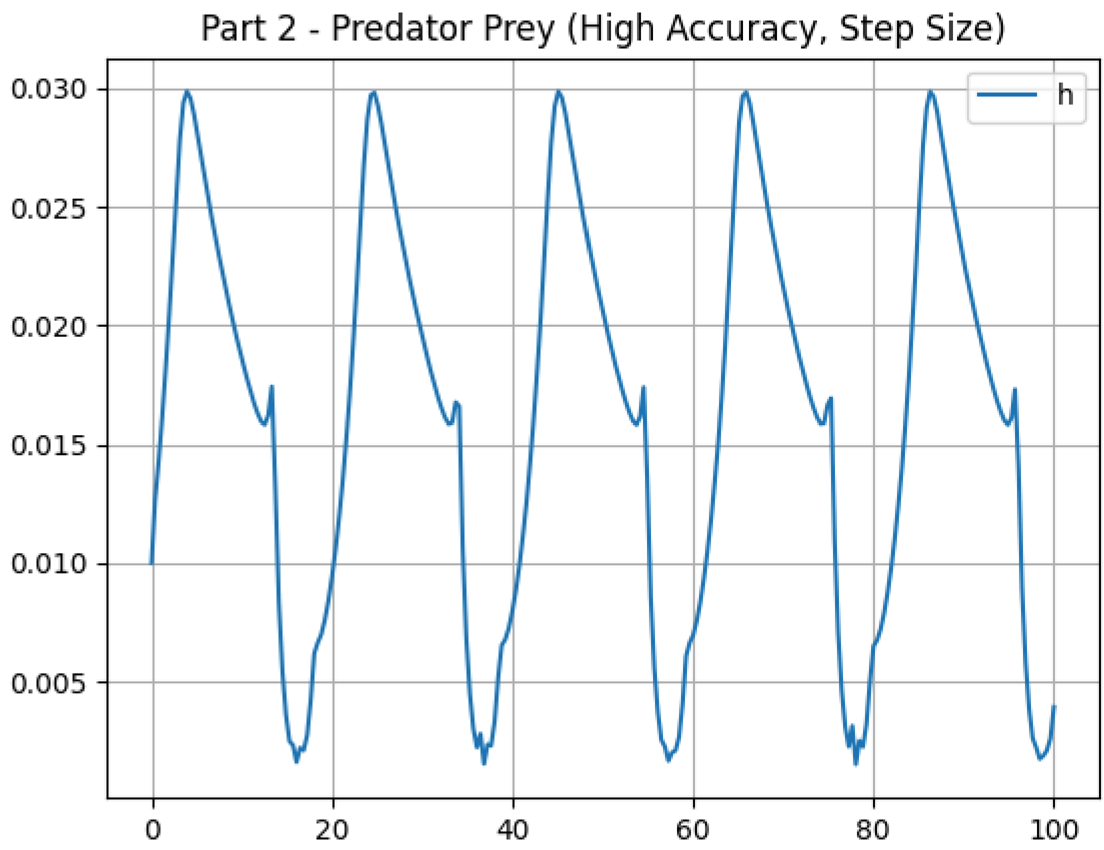


The solutions with error tolerance set to 1×10^{-6}:


Part 2 - Predator Prey (High Accuracy, Solution)
y1
y2
400
350
300
250
200
150
100
50
0
0
20
40
60
80
100

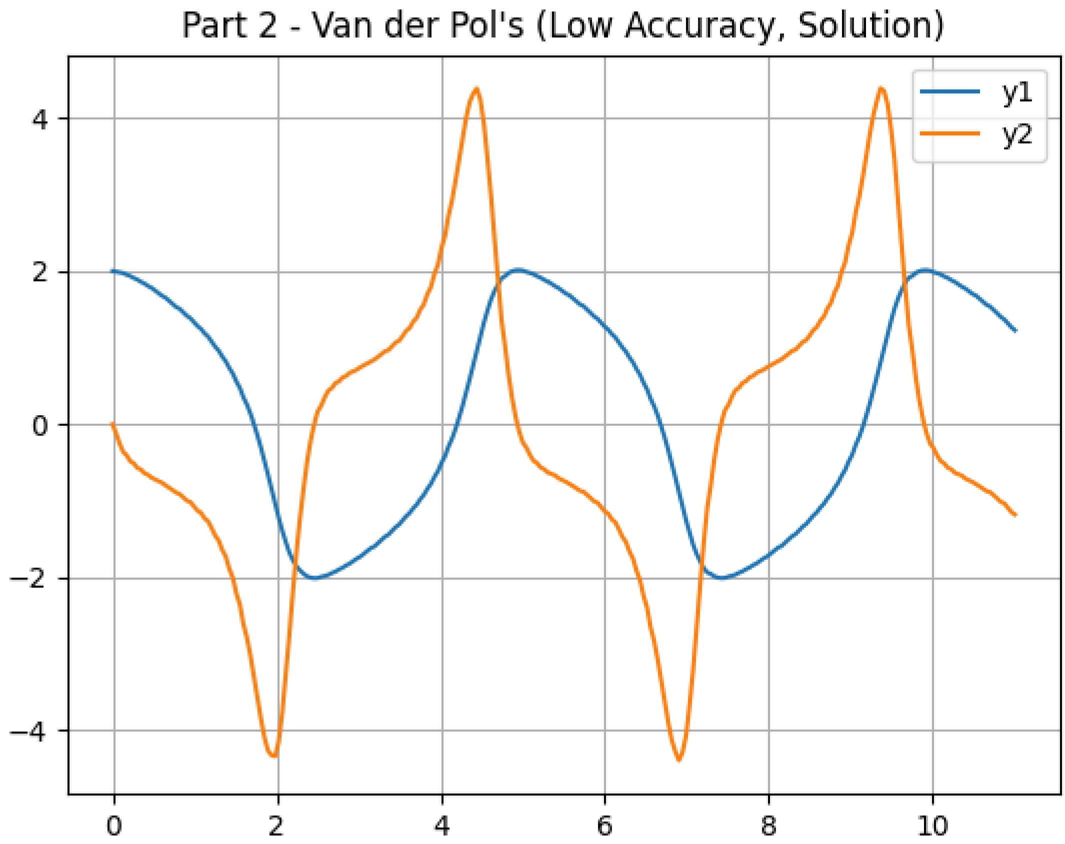

Part 2 - Predator Prey (High Accuracy, Relative)
y1
400
350
300
250
200
150
100
50
0
0
50
100
150
200

The step sizes used around each interpolating points:

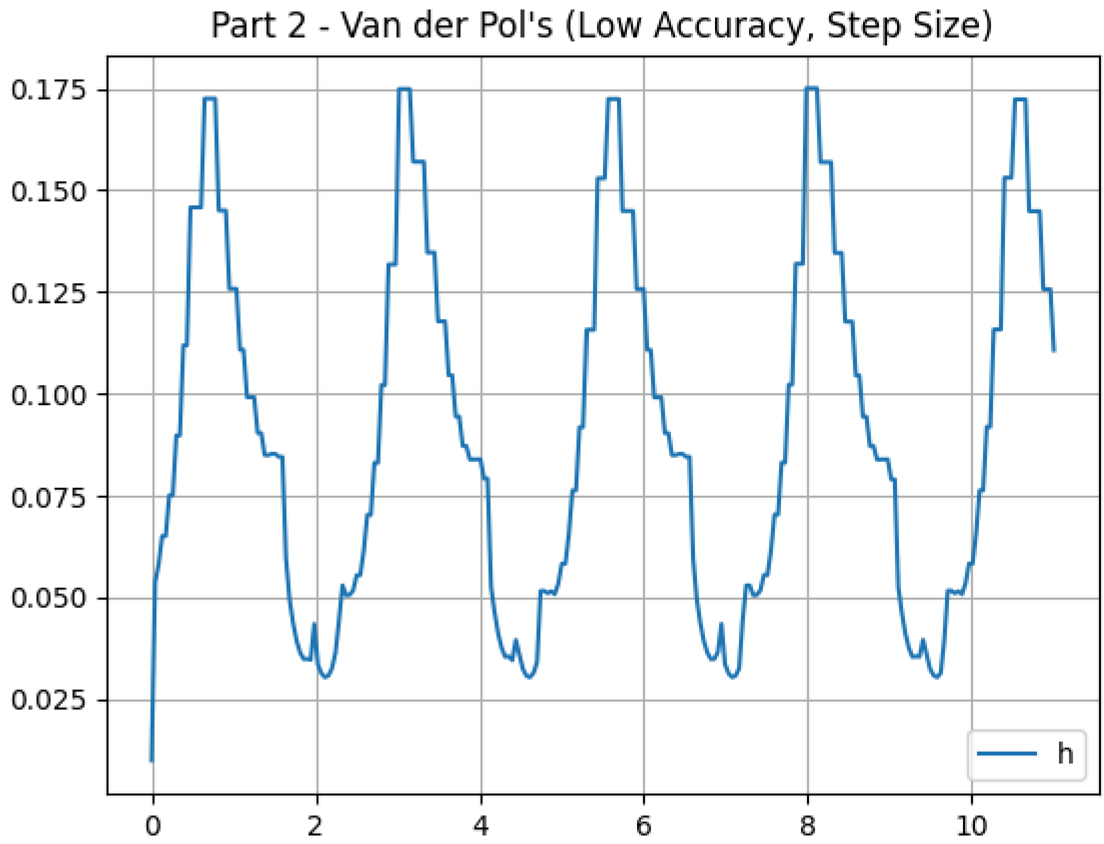


Van der Pol's Equation

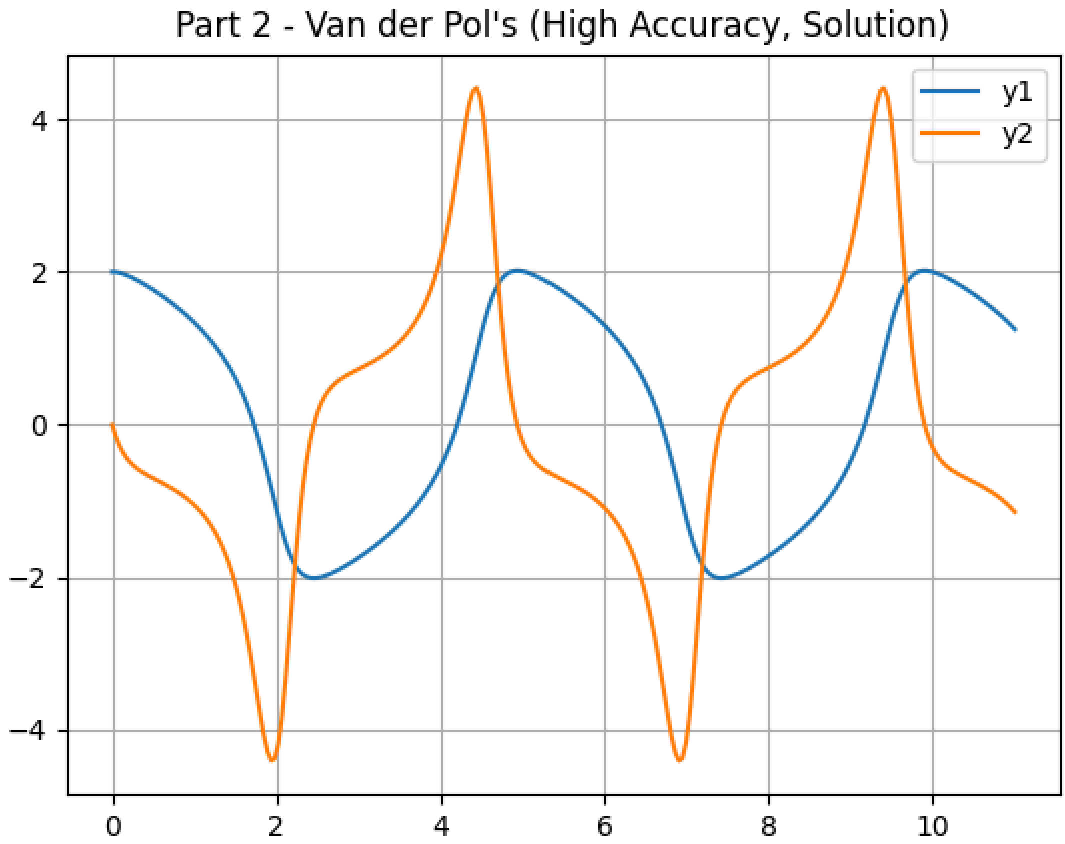
The solutions with error tolerance set to 1×10^{-3}:



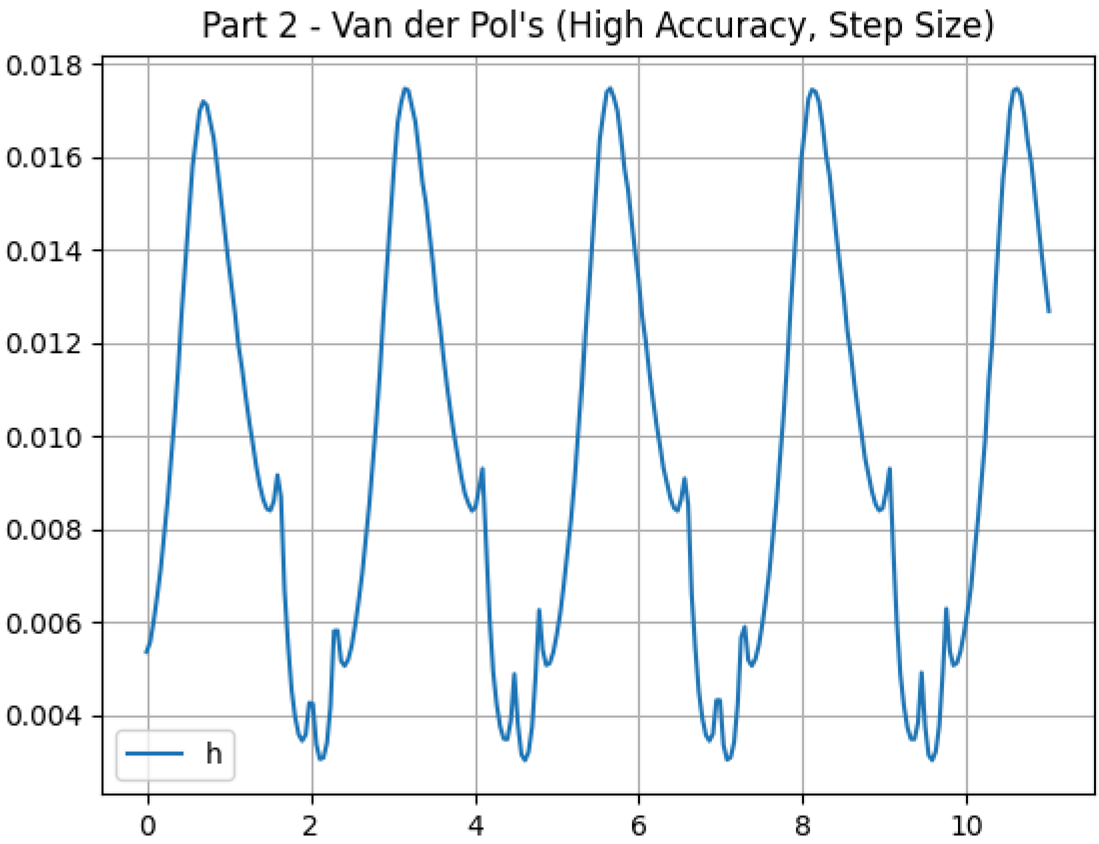
The step sizes used around each interpolating points:



The solutions with error tolerance set to 1×10^{-6}:



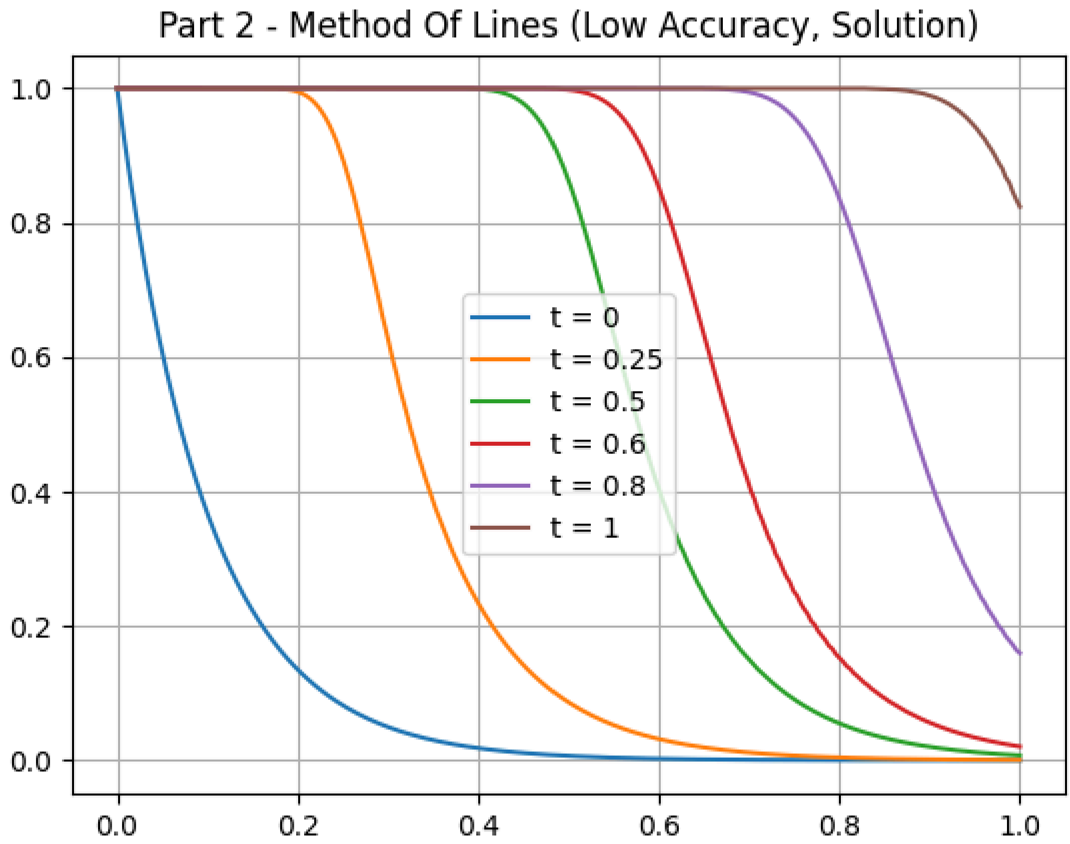
The step sizes used around each interpolating points:



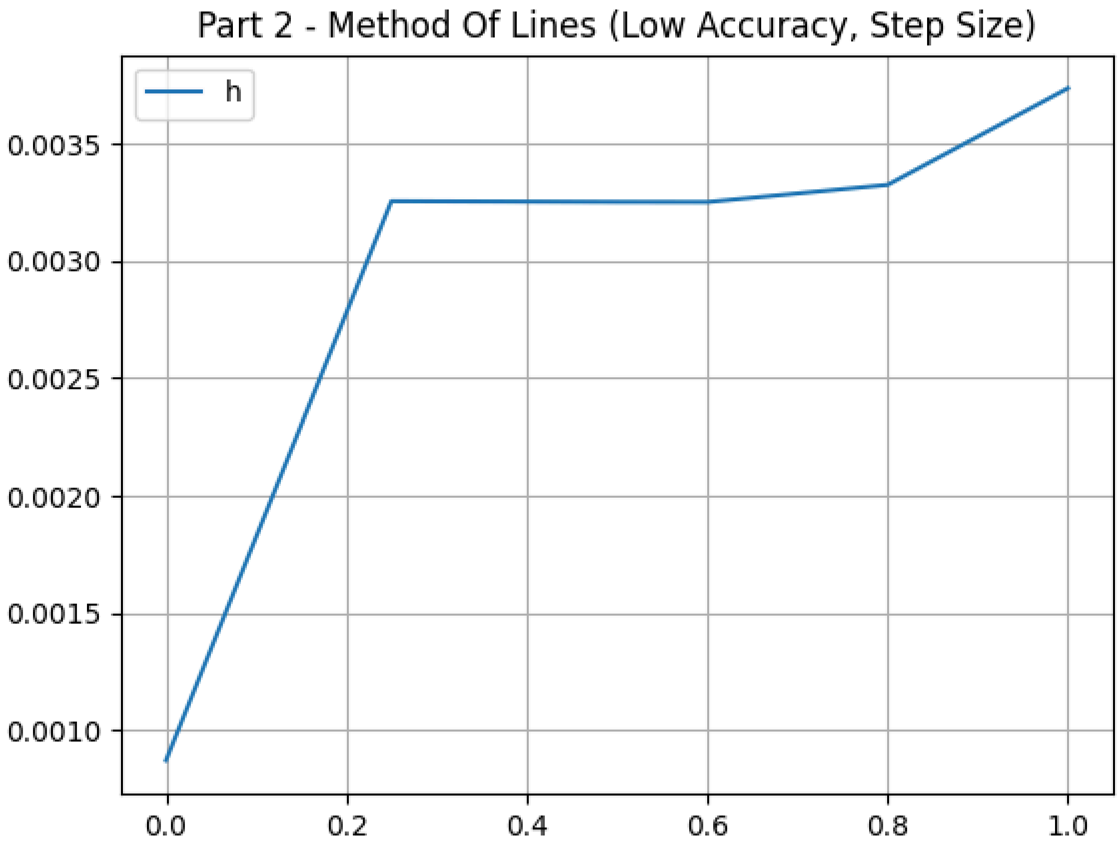
Method of Lines

The interval $x \in [0, 1]$ is discretized into 256 points.

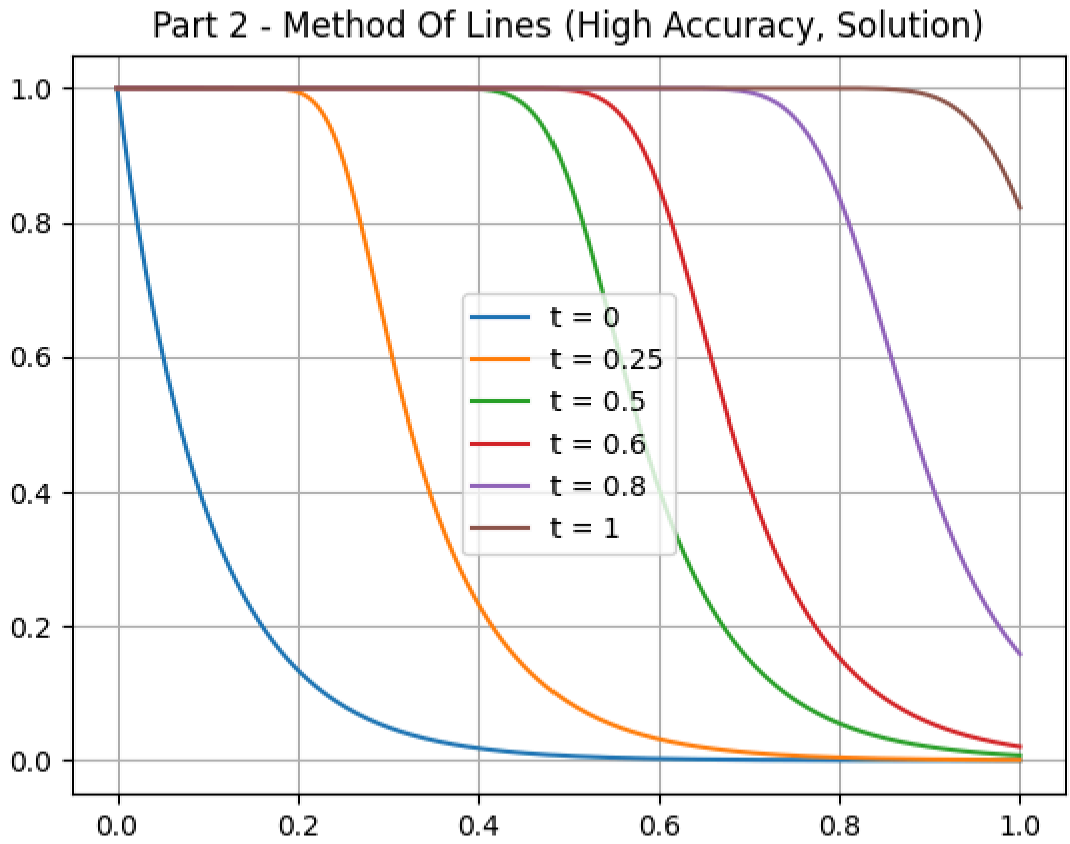
The solutions with error tolerance set to 1×10^{-3}:



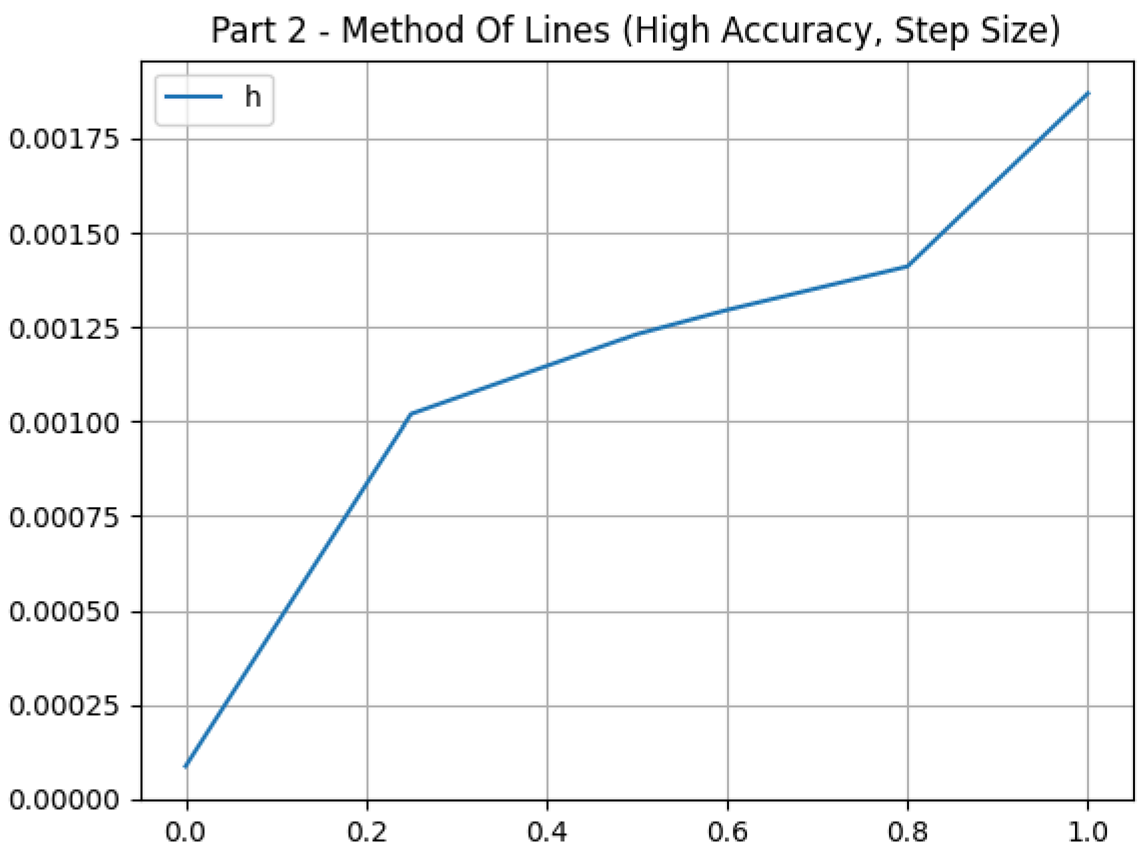
The step sizes used **around each interpolating points** of t:



The solutions with error tolerance set to 1×10^{-3}:



The step sizes used **around each interpolating points** of t:



Findings

In the three problem settings, even with different error tolerance, the changes of step size over time share similar patterns, which means that this is related to the properties of the problem (like the values of the solutions and the derivatives).