Dragos Pop (2618006)

Assignment 4 – ODEs

Concise method description

The function implemented in Matlab performs time integration of a generic system of ODEs using the fourth-order Runge-Kutta scheme. More exactly, at each time step, the four slopes k_i are computed and summed together as in the formula $y_{n+1} = y_n + \frac{1}{6}h(k_1 + 2k_2 + 2k_3 + k_4)$, where h is the size of the timestep, and y_n is the approximation from the previous iteration. Finally, the discrete-time and the solution estimates are each stored in a vector and outputted.

Function syntax

Given the required name of the Matlab file, the function can be called by *Assignment_4_Pop()*, specifying the right arguments. These are:

- f = a function handle with two parameters: time value t and y (which can be either a value or a vector for functions with more variables)
- t0 = the initial time
- h = the size of the timestep and, at the same time, the accuracy
- n = the number of steps
- y0 = an initial condition for the estimates (needs to match the number of variables y from the function handle)

Accordingly, calling *Assignment_4_Pop(@(t,y) 2*t, 0, 0.125, 10, 0)* returns a vector tHist = [0 0.1250 0.2500 0.3750 0.5000 0.6250 0.7500 0.8750 1.0000 1.1250 1.2500]^T, which stores the discrete times at which the solution is approximated, and a vector yHist = [0 0.0469 0.1250 0.2344 0.3750 0.5469 0.7500 0.9844 1.2500 1.5469 1.8750]^T, containing the approximation of the solution at the given time steps.

Restrictions

One disadvantage of the function is the extended computational effort required for calculating four slopes at each time step in order to compute the final result.

Program design

In the beginning, the initial condition vector y0 is being transposed if it is not already a vertical vector. Then, the initial conditions for t and y are set and stored in the output. Afterward, for each time step, the slopes are implemented as in the picture from the right. At the end of each iteration, the approximation is done after the formula $y_{n+1} = y_n + \frac{1}{6}h(k_1 + 2k_2 + 2k_3 + k_4)$, while to the current time t is added the timestep h. Lastly, the discrete times and the corresponding approximations are returned.

$$k_1 = f(t_n, y_n)$$
$$k_2 = f\left(t_n + \frac{h}{2}, y_n + h\frac{k_1}{2}\right)$$
$$k_3 = f\left(t_n + \frac{h}{2}, y_n + h\frac{k_2}{2}\right)$$
$$k_4 = f(t_n + h, y_n + hk_3)$$

Figure 1 – The formulas used for computing the slopes k

Figure interpretation

Figure 2 was generated by a secondary Matlab function, namely *PlotPhasePortrait(@(t, y) [y(2);-y(2)/2 - sin(y(1))], -2*pi, 2*pi, -pi, pi)*. The respective function takes as input a function handle with two variables and the limits of the domain intervals. Firstly, the initial time is set to 1, the size of the timestep to 0.125, and the number of steps to 100. Then, for a range of initial conditions ranging in the domain intervals, the method *Assignment_4_Pop()* is called and its results plotted, representing the function variables against each other.

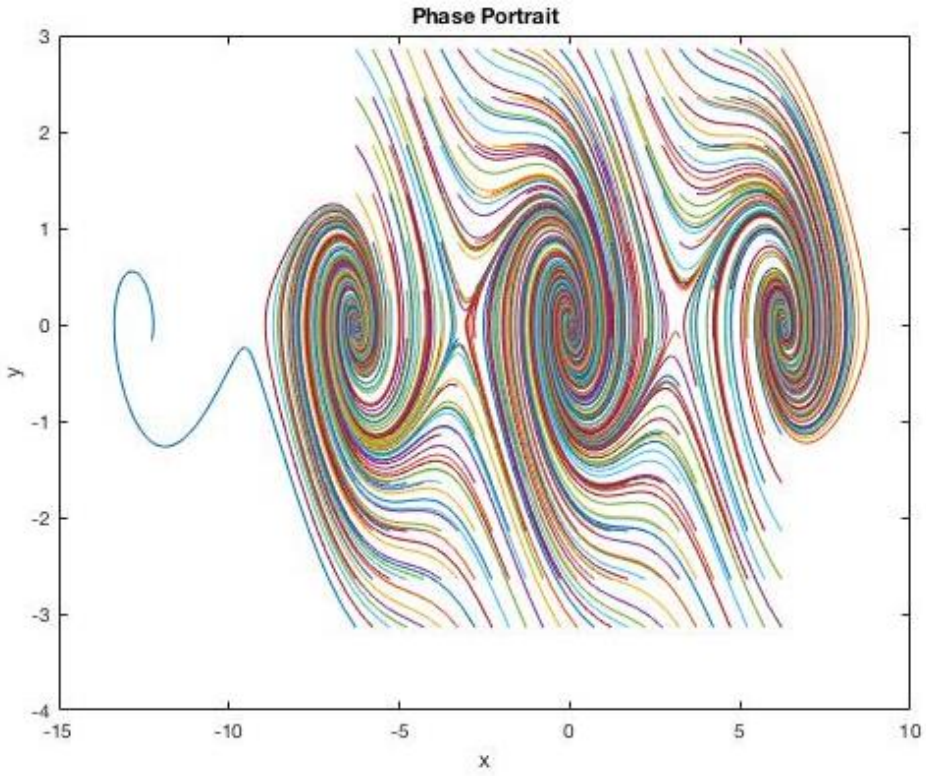

Figure 2 – Phase portrait of the ODEs system x' = y, y' = -y/2-sin using Runge Kutta 4 scheme for a range of initial conditions in the domain $(x, y) \in [-2\pi, 2\pi] \times [-\pi, \pi]$

Accordingly, in the above illustration, one can observe the phase portrait of the ODEs system x' = y, y' = -y/2 – sin y, using Runge Kutta 4 for a range of initial conditions in the domain $(x, y) \in [-2\pi, 2\pi] \times [-\pi, \pi]$. Moreover, it is visible from the picture that there are 4 equilibria, roughly at the positions (6.29, 0), (0.01, 0), (-6.26, 0) and (-12.5, 0), which can be classified as stable spirals. Besides the equilibria states, one can remark 3 unstable saddles at (3, 0), (-3.15, 0), (-9.1, 0).

Order of convergence compared to the Forward Euler method

To compare the order of convergence of the Runge Kutta 4 scheme with the one of Forward Euler method, the following system of ODEs was used: x' = - x, y' = - y, x(0) = 1, y(0) = 1. First of all, the two methods (*Assignment_4_Pop()* and *ForwardEuler()*) were performed on the given system, keeping the values of the parameters unchanged: f = @(t, y) [-y(1); -y(2)], t0 = 1, h = 0.125, n = 100, and y0 = [1; 1]. After the results were generated, the Matlab ODE solver, *ode45()*, was called to find the true values of the system, such that the total error can be calculated as the sum of all absolute values of the approximation minus the corresponding true value. Subsequently, the error sum for Runge Kutta 4 was discovered to be 0.0012, while the total error for Forward Euler was 0.5104, suggesting that the order of convergence of the first scheme is larger, converging faster to the true solution. Moreover, if the errors are plotted against the discrete times and a horizontal line y = h * p is drawn, where p is the order of convergence of the method, one can see that the absolute errors of Forward Euler are never larger than y = h (see Figure 3) and the ones of Runge Kutta 4 are not higher than $y = h^4$ (see Figure 4), meaning that the order of convergence is 1 and 4, respective. As a conclusion, it is fair to state that the increased accuracy of the Runge Kutta 4 scheme compensates for the additional computational effort required.

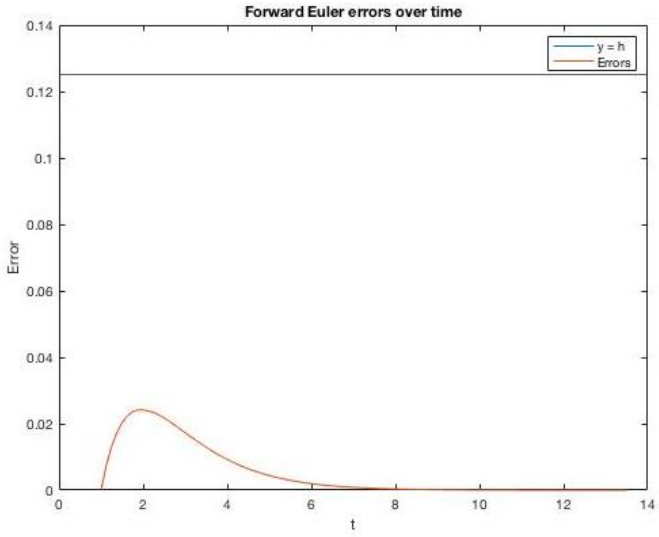

Figure 3 – The Forward Euler errors over time

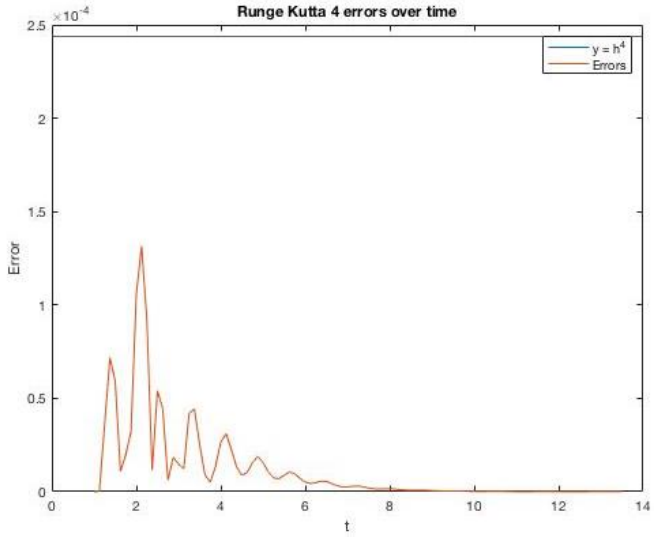

Figure 4 – The Runge Kutta 4 errors over time