Homework 2.(b)

Modeling Complex Systems, Javier Lobato

Due date: Tuesday, February 27, 2018

C. Forward Euler implementation

euler_integrator.m function

```
function [tspan, y] = euler_integrator(fun, tspan, y0)
%EULER Implementation of the forward Euler integration method
% INPUTS:
%   fun: function that wants to be integrated
%   tspan: interval of integration, that goes from intial time to the final
%       time [t0 t1 t2 t3 ... tf] - it includes all timesteps
%   y0: initial conditions for the function, it must have the SAME length
%       as the desired output y
%
% OUTPUTS:
%   tspan: times in which the integration has been carried out (the same as
%       the input array
%   y: values of the function after the integration
%
% sample call (for a function with extra arguments):
%           [t, y] = euler_integrator(@(t,y) function(t,y,a), tspan, y0)

% Javier Lobato, created 02/20/2018

% First of all, the function will check that the tspan vector has all
% timesteps of the same size. To do that a subtraction of the i element
% with respect the i-1 element is carried out, comparing all the values of
% the vector with one of its elements (all must be the same)
timestep = tspan(2:length(tspan))-tspan(1:length(tspan)-1);

if all(round(timestep, 10) ~= round(timestep(1),10))
    disp('Not equally spaced tspan')
else
    h = timestep(1);
end

% ode45 returns a column vector, so to keep the structure when possible...
y = zeros([length(y0), length(tspan)]);

% Set the initial values to the solution vector
y(:, 1) = y0;

% Loop over the whole time vector - 1 (the first initial value has been
% already included in the solution)
for i = 1:length(tspan)-1
    % Definition of the Euler integration method
    y(:, i+1) = y(:, i) + h .* fun(tspan(i), y(:, i));
end
end
```

D. Heun's method implementation

heun_integrator.m function

```
function [tspan, y] = heun_integrator(fun, tspan, y0)
%HEUN_INTEGRATOR Implementation of Heun's integration method
% INPUTS:
%   fun: function that wants to be integrated
%   tspan: interval of integration, that goes from intial time to the final
%       time [t0 t1 t2 t3 ... tf] - it includes all timesteps
%   y0: initial conditions for the function, it must have the SAME length
%       as the desired output y
%
% OUTPUTS:
%   tspan: times in which the integration has been carried out (the same as
%       the input array
%   y: values of the function after the integration
%
% sample call (for a function with extra arguments):
%           [t, y] = heun_integrator(@(t,y) function(t,y,a), tspan, y0)

% Javier Lobato, created 02/20/2018

% First of all, the function will check that the tspan vector has all
% timesteps of the same size. To do that a subtraction of the i element
% with respect the i-1 element is carried out, comparing all the values of
% the vector with one of its elements (all must be the same)
timestep = tspan(2:length(tspan))-tspan(1:length(tspan)-1);

if all(round(timestep, 10) ~= round(timestep(1),10))
    disp('Not equally spaced tspan')
else
    h = timestep(1);
end

% ode45 returns a column vector, so to keep the structure when possible...
y = zeros([length(y0), length(tspan)]);

% Set the initial values to the solution vector
y(:, 1) = y0;

% Loop over the whole time vector - 1 (the first initial value has been
% already included in the solution)
for i = 1:length(tspan)-1
    % Definition of the Heun integration method
    K1 = h * fun(tspan(i), y(:, i));
    K2 = h * fun(tspan(i+1), y(:, i) + K1);
    y(:, i+1) = y(:, i) + (K1+K2)/(2);
end

end
```

E. Lotka Volterra System

`lvSystem.m` function

```
function [dxdt] = lvSystem(t, x, a)
%LVSYSTEM Lotka-Volterra System implementation
% INPUTS:
%   t: although not necessary for direct evaluation, it is necesary for the
%       different integration methods to work properly
%   x: 3-element vector in which the system will be evaluated
%   a: alpha value for the A matrix (already included in the code)
%
% OUTPUTS:
%   dxdt: output result of the system
%
% sample call:
%             dxdt = lvSystem(t, [x1 x2 x3], alpha)

% Javier Lobato, created 02/20/2018

% Preallocation of the output function in a column vector
dxdt = zeros([3,1]);

% A matrix definition with the alpha parameter from the input
A = [0.5 0.5 0.1; -0.5 -0.1 0.1; a 0.1 0.1];

% Looping over the three variable of the Lotka-Volterra system
for i = 1:3
    dxdt(i) = x(i)*(A(i,1)*(1-x(1)) + A(i,2)*(1-x(2)) + A(i,3)*(1-x(3)));
end

end
```

F. Driver

`hw2BDriver.m` function

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                              HOMEWORK #2.B                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Driver for the homework 2.B, this file requires the functions:
%       euler_integrator.m
%       heun_integrator.m
%       lvSystem.m
% to work properly. Running all will generate and save 12 figures for the
% different desired configurations

% Javier Lobato, created 02/20/18

% Let's clear the workspace variables and previous figures
clear all; close all; clc
```

```
%% SECTION G - CALCULATIONS
% Section values declaration: maximum time, initial values and the 3 alpha
% values and 3 timesteps in which the system will operate (ending with 9
% possible configurations)
maxtime = 200;
initVal = [0.3 0.2 0.1];
alpha = [0.75, 1.2, 1.5];
timesteps = [0.1, 0.5, 1.0];

% Preallocation of cells for both values and times for all the possible
% configurations of timestep size and alpha.
ode45_val = cell(length(alpha), length(timesteps));
euler_val = cell(length(alpha), length(timesteps));
heun_val = cell(length(alpha), length(timesteps));
ode45_time = cell(length(alpha), length(timesteps));
euler_time = cell(length(alpha), length(timesteps));
heun_time = cell(length(alpha), length(timesteps));

for i = 1:length(alpha) % Looping over the different alpha values
    for j = 1:length(timesteps) % Looping over the different timestep sizes
        % Each calling to a function will return the integration time as a
        % column vector and the integration values for each one of the
        % three components
        [ode45_time{i, j}, ode45_val{i, j}] = ode45(@(t,y) lvSystem(t, y, alpha(i)),
         ↪  0:timesteps(j):maxtime, initVal);
        [euler_time{i, j}, euler_val{i, j}] = euler_integrator(@(t,y) lvSystem(t, y, alpha(i)),
         ↪  0:timesteps(j):maxtime, initVal);
        [heun_time{i, j}, heun_val{i, j}] = heun_integrator(@(t,y) lvSystem(t, y, alpha(i)),
         ↪  0:timesteps(j):maxtime, initVal);
    end
end

%% SECTION G - PLOTTING
% Variable to take into account the number of created figures
figNo = 0;

for i = 1:length(alpha) % Looping over each alpha value
    for j = 1:length(timesteps) % Looping over each timestep size

        % If there is a NaN in any element of the Euler solution
        if any(isnan(euler_val{i,j}(1,:)))
            % Limits of the figure will be obtained from the maximum value
            % of either the ode45 or the Heun methods (multiplied by 1.25
            % to have a small 'margin')
            lowx = 1.25*min([ode45_val{i,j}(:,1)', heun_val{i,j}(1,:)]);
            highx = 1.25*max([ode45_val{i,j}(:,1)', heun_val{i,j}(1,:)]);
            lowy = 1.25*min([ode45_val{i,j}(:,2)', heun_val{i,j}(2,:)]);
            highy = 1.25*max([ode45_val{i,j}(:,2)', heun_val{i,j}(2,:)]);
            lowz = 1.25*min([ode45_val{i,j}(:,3)', heun_val{i,j}(3,:)]);
            highz = 1.25*max([ode45_val{i,j}(:,3)', heun_val{i,j}(3,:)]);
        end

        % Let's increase the figure number for each looping iteration
        figNo = figNo +1;
        % Create a new figure for each loop
        figure(figNo)

```

```
        % Left subplot: 3D view of the 3 species
        subplot(1,2,1);
        % Plot the three components (one per axis) of the three integration
        % methods previously calculated
        hold on
            plot3(euler_val{i,j}(1,:), euler_val{i,j}(2,:), euler_val{i,j}(3,:), 'k--','LineWidth',1)
            plot3(heun_val{i,j}(1,:), heun_val{i,j}(2,:), heun_val{i,j}(3,:), 'k-.', 'LineWidth',2)
            plot3(ode45_val{i,j}(:,1), ode45_val{i,j}(:,2), ode45_val{i,j}(:,3), 'k-','LineWidth',1)
        hold off
        % Force the same view for all the alpha and timstep size values
        view([120 10])
        % Replace the automatic axis value just if there is a NaN value in
        % the Euler solution
        if any(isnan(euler_val{i,j}(1,:)))
            xlim([lowx highx])
            ylim([lowy highy])
            zlim([lowz highz])
        end
        % Plot options - legend, title, axis labels...
        legend('Euler','Heun','ode45')
        title(['3D plot for $$\alpha$$ = ', num2str(alpha(i)), ', $$h = $$', num2str(timesteps(j))],'interpreter','latex','FontSize',18)
        xlabel('Specie 1')
        ylabel('Specie 2')
        zlabel('Specie 3')

        % Right subplot: 2D view of the first species evolution in time
        subplot(1,2,2);
        % Plot the first species versus time for the three integration
        % methods previously calculated
        hold on
            plot(euler_time{i,j}(:), euler_val{i,j}(1,:), 'x-', 'Color', [0.3 0.3 0.3], 'MarkerSize', 5)
            plot(heun_time{i,j}(:), heun_val{i,j}(1,:), '.-', 'Color', [0.6 0.6 0.6], 'MarkerSize', 10)
            plot(ode45_time{i,j}(:), ode45_val{i,j}(:,1), 'k-', 'LineWidth',1.1)
        hold off
        % Replace the automatic axis value just if there is a NaN value in
        % the Euler solution
        if any(isnan(euler_val{i,j}(1,:)))
            ylim([lowx highx])
        end
        % Plot options - legend, title, axis labels...
        title(['First species for $$\alpha$$ = ', num2str(alpha(i)), ', $$h = $$', num2str(timesteps(j))],'interpreter','latex','FontSize',18)
        legend('Euler','Heun','ode45')
        xlabel('Time')
        ylabel('First species')

        % Make the plot full screen and save it with .png format
        set(gcf, 'Position', get(0, 'Screensize'));
        print(['fig',num2str(figNo)],'-dpng','-r150')
    end
end
```

```matlab

%% SECTION H - CALCULATIONS
% Values declaration: maximum time, initial values, alpha values and a
% vector with different timestep size that will be analyzed
maxtime = 50;
initVal = [0.3 0.2 0.1];
alpha = [0.75, 1.2, 1.5];
Etimesteps = logspace(-2,0,40);

% Preallocation of a cell array JUST for the value. Given that time does
% not need to be stored in this case, no cell-array for time have been
% created (E stands for error)
Eode45_val = cell(length(alpha), length(Etimesteps));
Eeuler_val = cell(length(alpha), length(Etimesteps));
Eheun_val = cell(length(alpha), length(Etimesteps));

% Giving that ode45 solution will be considered as the most accurate one,
% let's increase the relative tolerance to have a more precise solution
options = odeset('RelTol',1E-8);

for i = 1:length(alpha) % Looping over the 3 values of alpha
    for j = 1:length(Etimesteps) % Looping over the timestep size
        % As said, time is not necessary here, so the output of the
        % function 'tspan' is replace by ~
        [~, Eode45_val{i, j}] = ode45(@(t,y) lvSystem(t, y, alpha(i)), 0:Etimesteps(j):maxtime, initVal, options);
        [~, Eeuler_val{i, j}] = euler_integrator(@(t,y) lvSystem(t, y, alpha(i)), 0:Etimesteps(j):maxtime, initVal);
        [~, Eheun_val{i, j}] = heun_integrator(@(t,y) lvSystem(t, y, alpha(i)), 0:Etimesteps(j):maxtime, initVal);
    end
end

% Each matrix will have a maximum value of error for each alpha value and
% each timestep size. Although there are just three species, just the error
% of the first specie is taken into account
eulerError = zeros([length(alpha), length(Etimesteps)]);
heunError = zeros([length(alpha), length(Etimesteps)]);

for i = 1:length(alpha) % Looping over the alpha value
    for j = 1:length(Etimesteps) % Looping over the timestep size
        % Saves the maximum value of the absolute error between both
        % integrator methods and the ode45 method (selected as reference)
        eulerError(i,j) = max(abs(Eode45_val{i,j}(:,1)-Eeuler_val{i,j}(1,:)'));
        heunError(i,j) = max(abs(Eode45_val{i,j}(:,1)-Eheun_val{i,j}(1,:)'));
    end
end

%% SECTION H - PLOTTING
% Variable figNo is taken from the previous section to have continuity and
% do not replace figures

for i = 1:length(alpha) % There will be a plot for each alpha value
    % Let's increase the figure number for each loop iteration
    figNo = figNo + 1;


```

```
    % Interpolation of the first part of the curve with a line in the
    % loglog scale, in order to 'get' the order of accuracy
    coefE = polyfit(log(Etimesteps(1:20)), log(eulerError(i,1:20)), 1);
    coefH = polyfit(log(Etimesteps(1:20)), log(heunError(i,1:20)), 1);

    % Open a new figure
    figure(figNo)

    % Plot the error of the Euler and Heun's methods and the fitting
    % obtained for the first 20 points
    hold on
        loglog(Etimesteps, eulerError(i,:),'o-','Color',[0.6 0.6 0.6],'LineWidth',0.5,'MarkerFaceColor',[0.6 0.6 0.6])
        loglog(Etimesteps, heunError(i,:),'s-','Color',[0.6 0.6 0.6],'LineWidth',0.5,'MarkerSize',8,'MarkerFaceColor',[0.6 0.6 0.6])
        loglog(Etimesteps(1:20), exp(coefE(2))*Etimesteps(1:20).^coefE(1),'k--','LineWidth',1.3)
        loglog(Etimesteps(1:20), exp(coefH(2))*Etimesteps(1:20).^coefH(1),'k:','LineWidth',3)
    hold off
    % Plot options - legend, title, axis label
    title(['First species absolute error for $$\alpha$$ = ', num2str(alpha(i))],'interpreter','latex','FontSize',18)
    legend('Euler absolute error','Heun absolute error',['Euler fit: ',num2str(coefE(1))],['Heun fit: ',num2str(coefH(1))],'Location','northwest')
    set(gca,'xscale','log','yscale','log')
    xlabel('Timestep size (h)')
    ylabel('Absolute error')
    % Make the plot full screen and save it with .png format
    set(gcf, 'Position', get(0, 'Screensize'));
    print(['error',num2str(i)],'-dpng','-r150')
end
```

G. Set of simulations of the system for different parameters

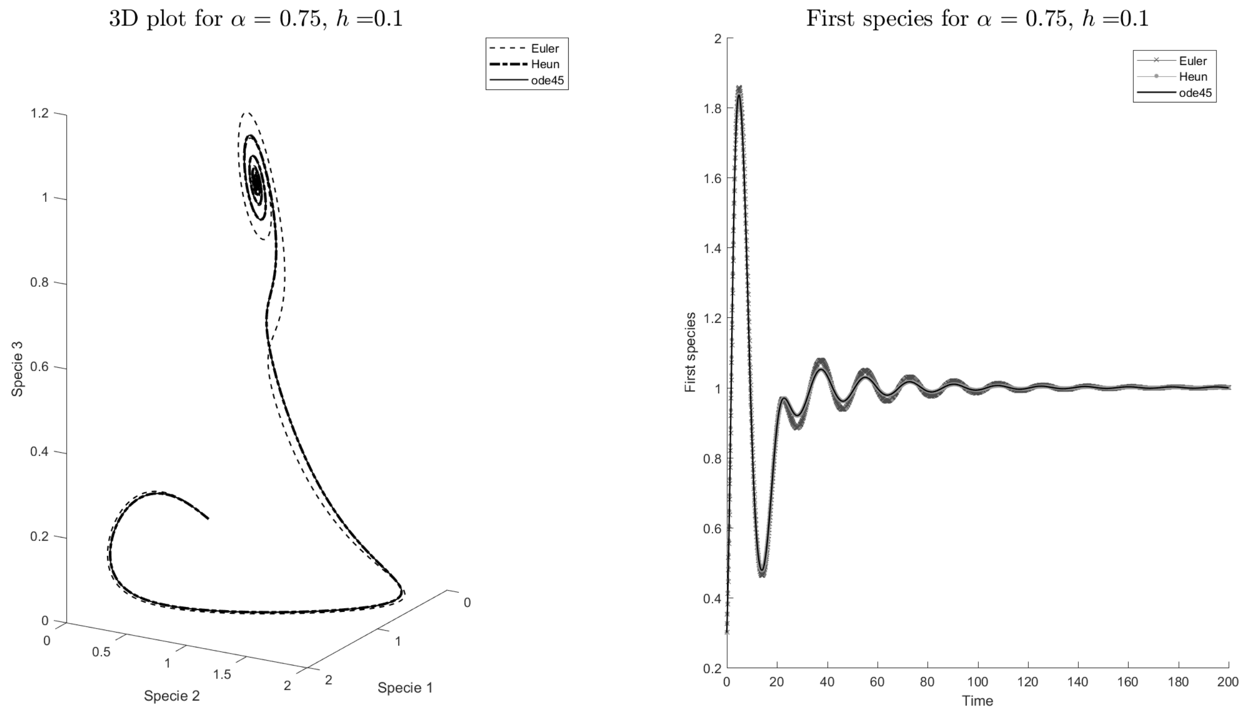


Figure 1: Values of $\alpha = 0.75$ and $h = 0.1$

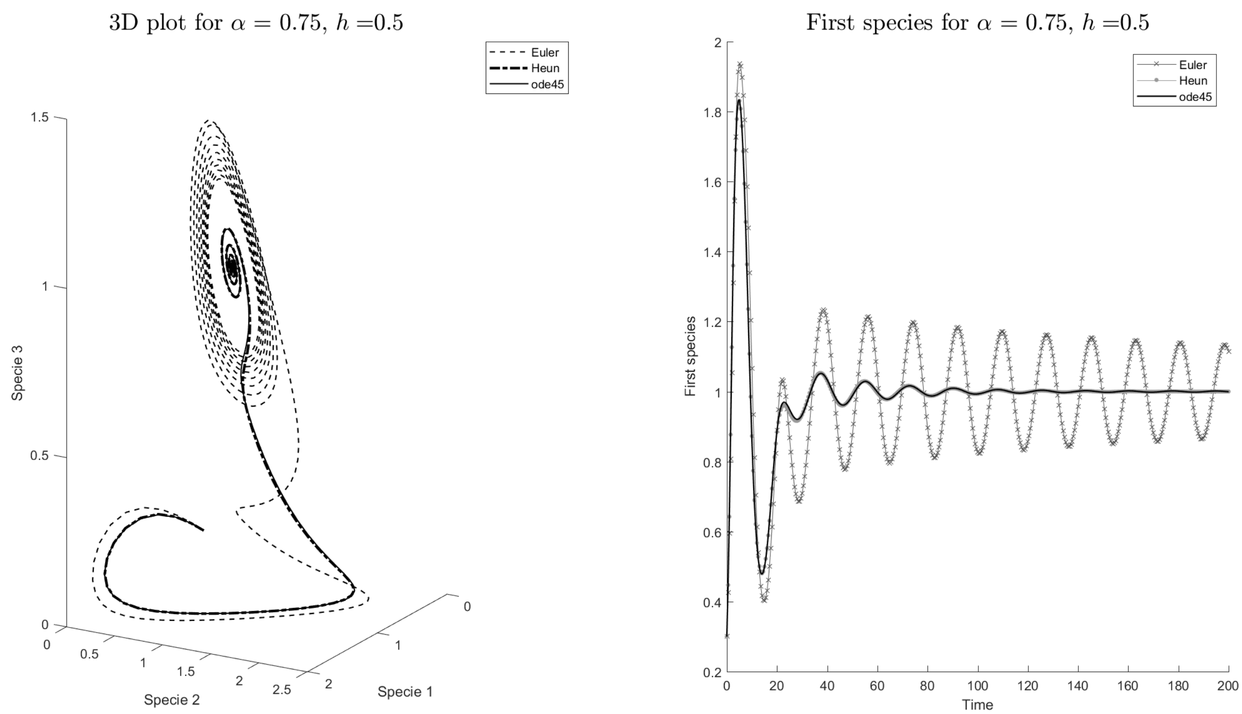


Figure 2: Values of $\alpha = 0.75$ and $h = 0.5$

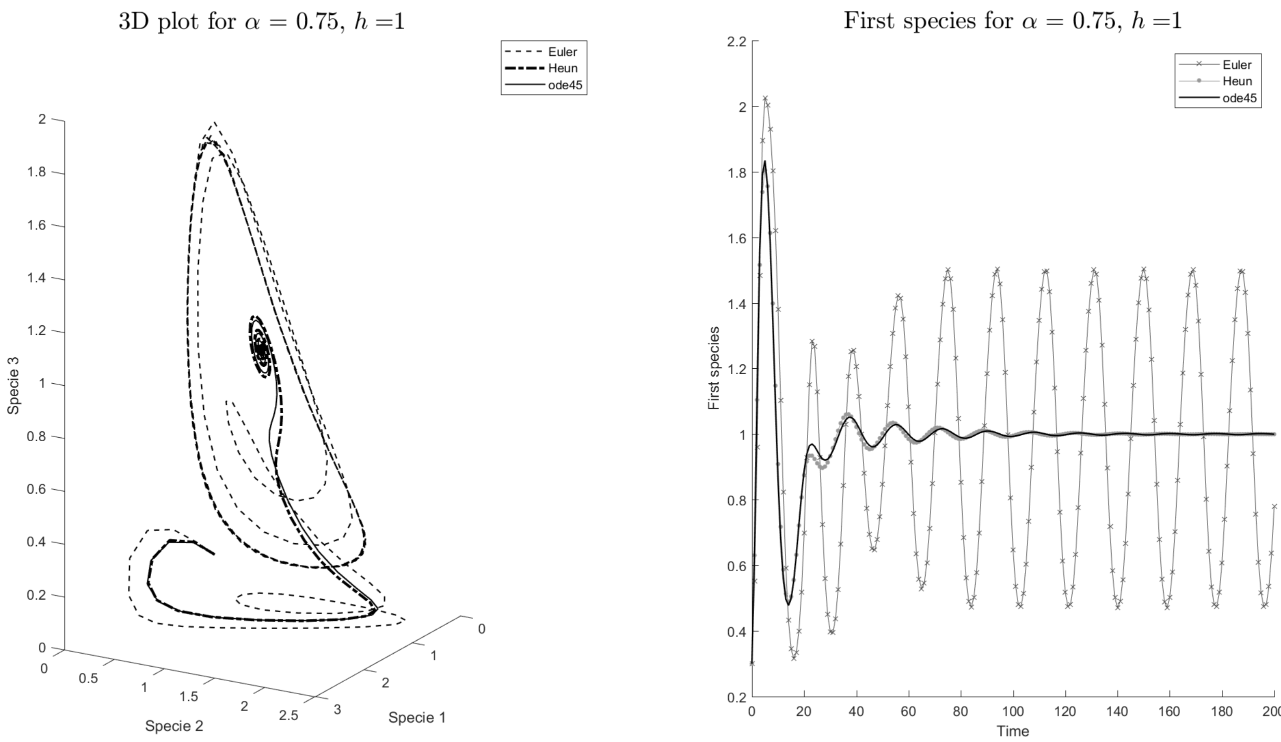


Figure 3: Values of $\alpha = 0.75$ and $h = 1$

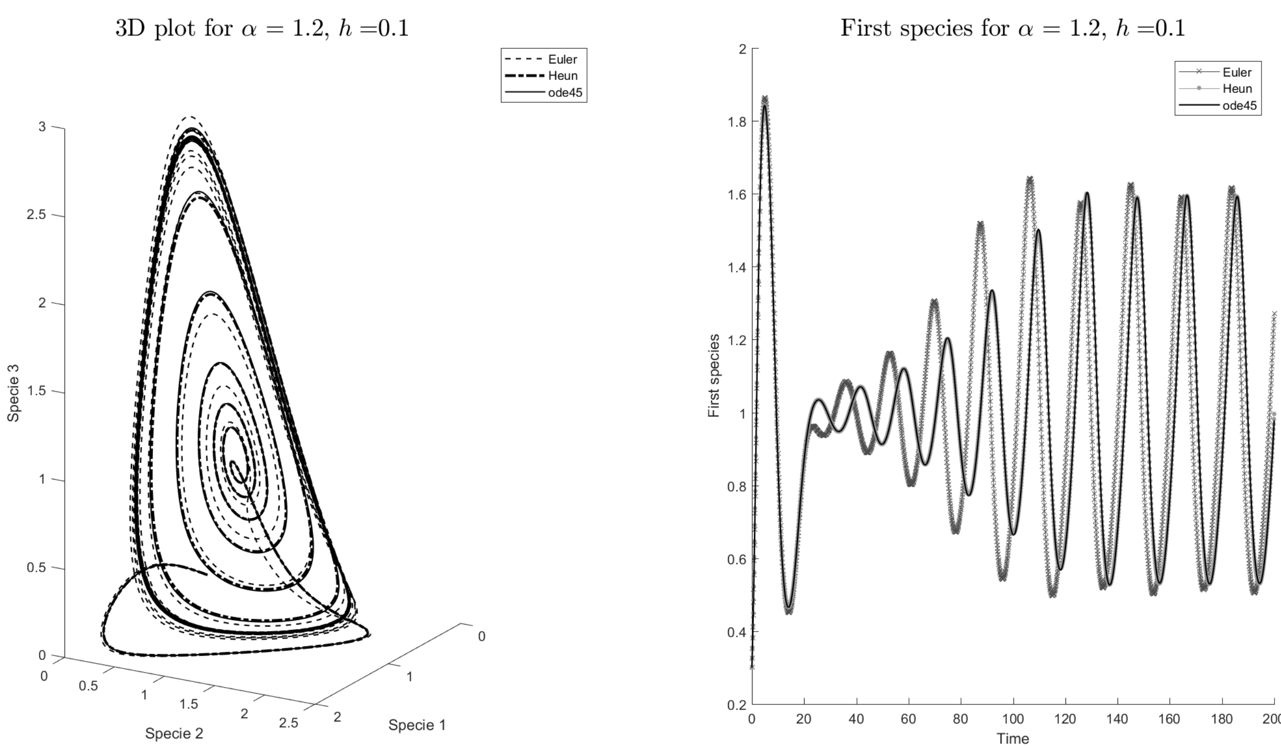


Figure 4: Values of $\alpha = 1.2$ and $h = 0.1$

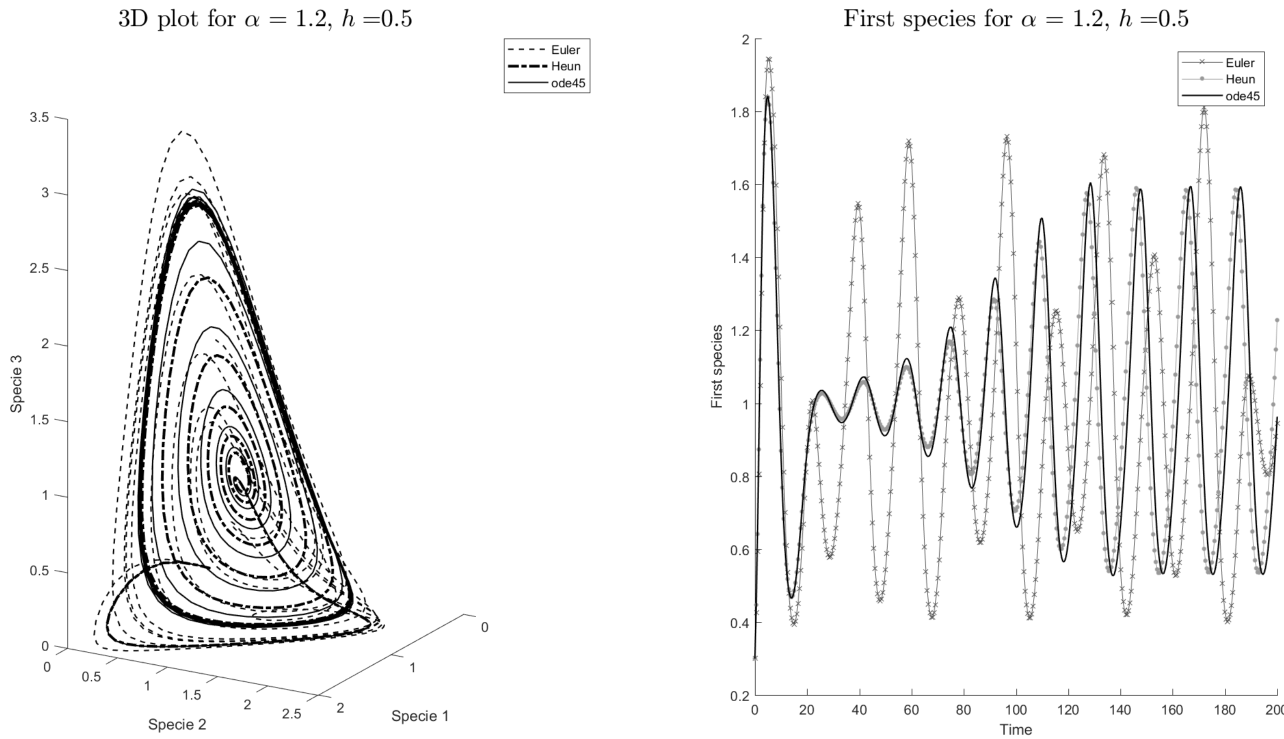


Figure 5: Values of $\alpha = 1.2$ and $h = 0.5$

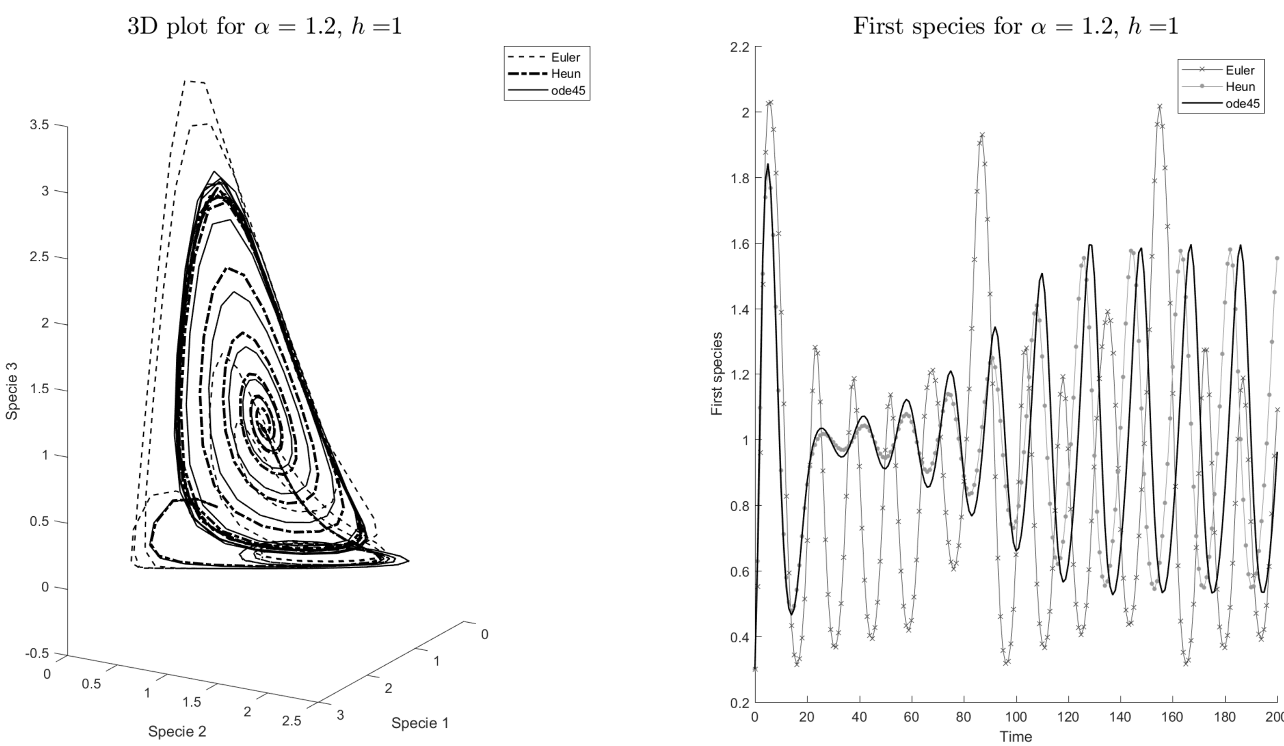


Figure 6: Values of $\alpha = 1.2$ and $h = 1$

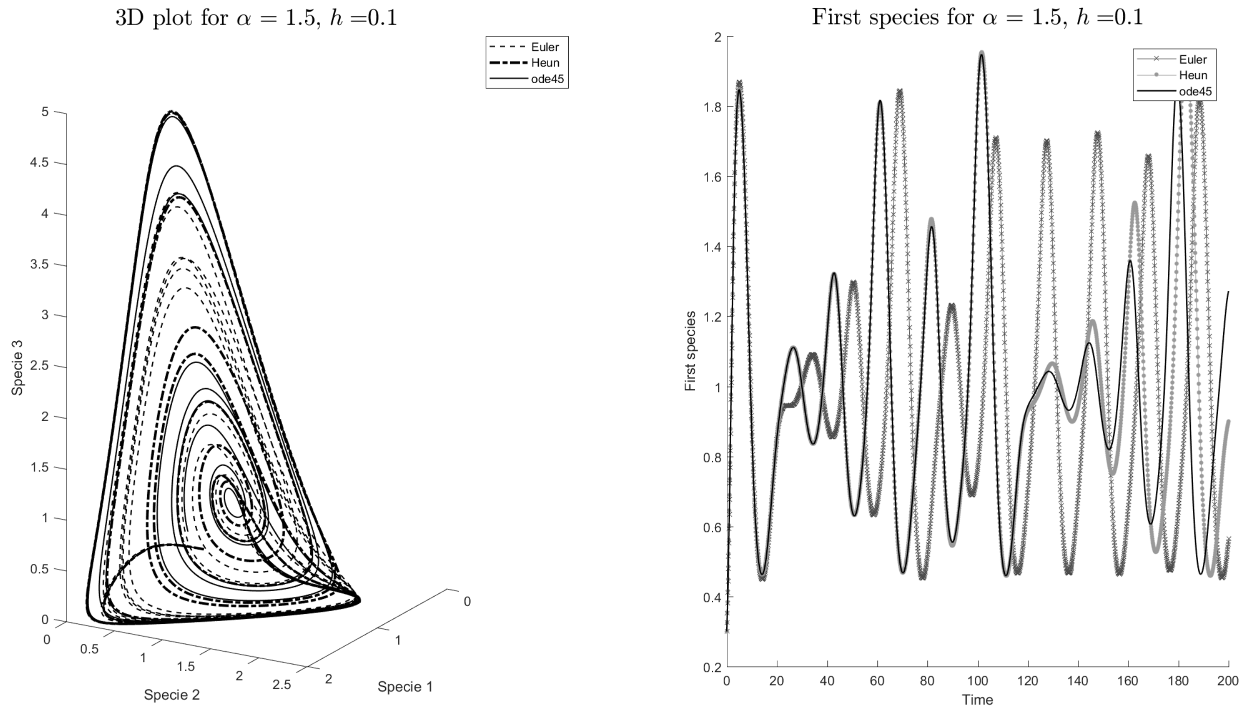


Figure 7: Values of $\alpha = 1.5$ and $h = 0.1$

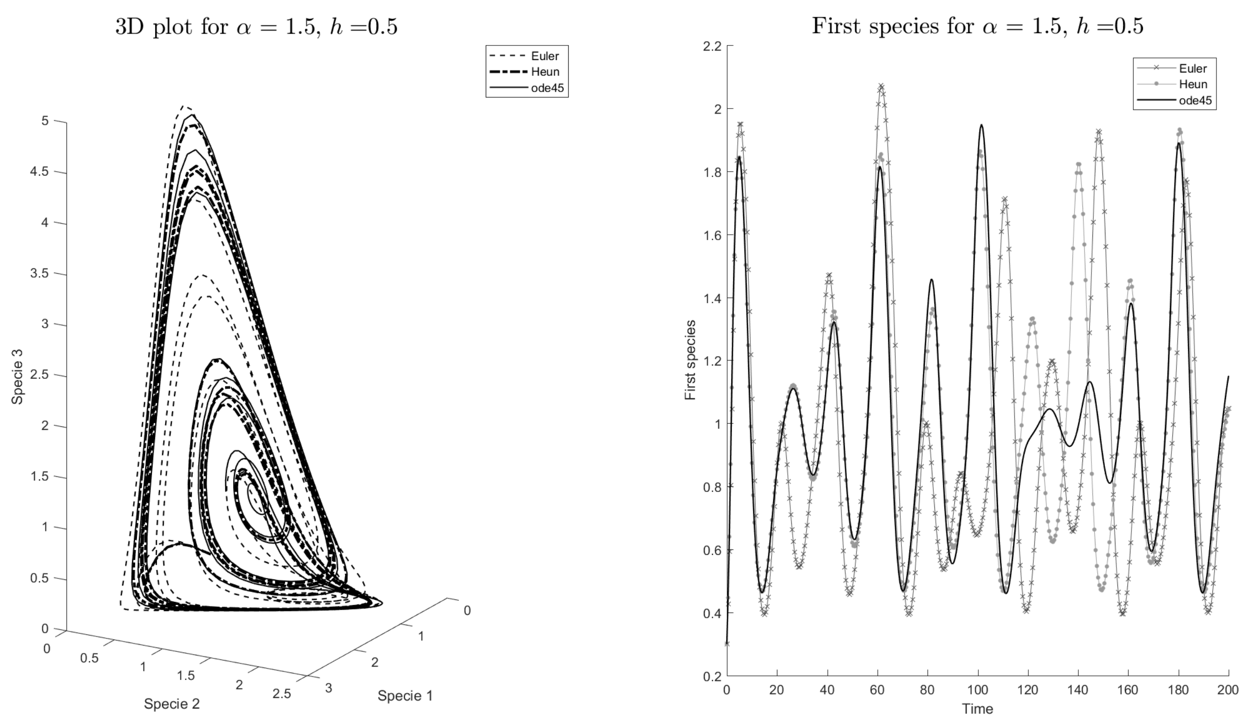
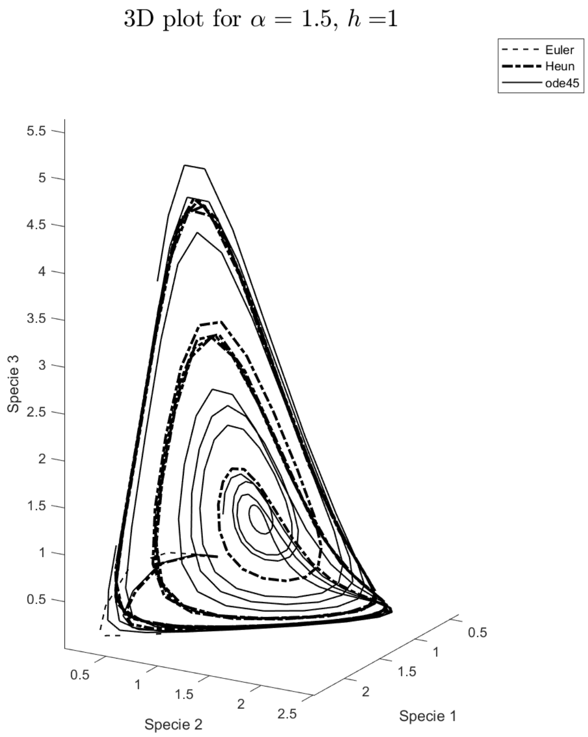


Figure 8: Values of $\alpha = 1.5$ and $h = 0.5$



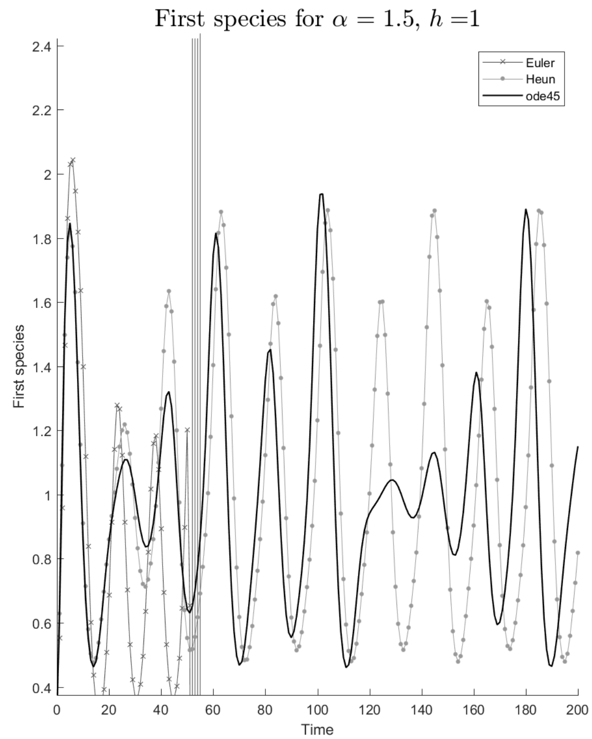


Figure 9: Values of $\alpha = 1.5$ and $h = 1$

H. Log-log scale errors

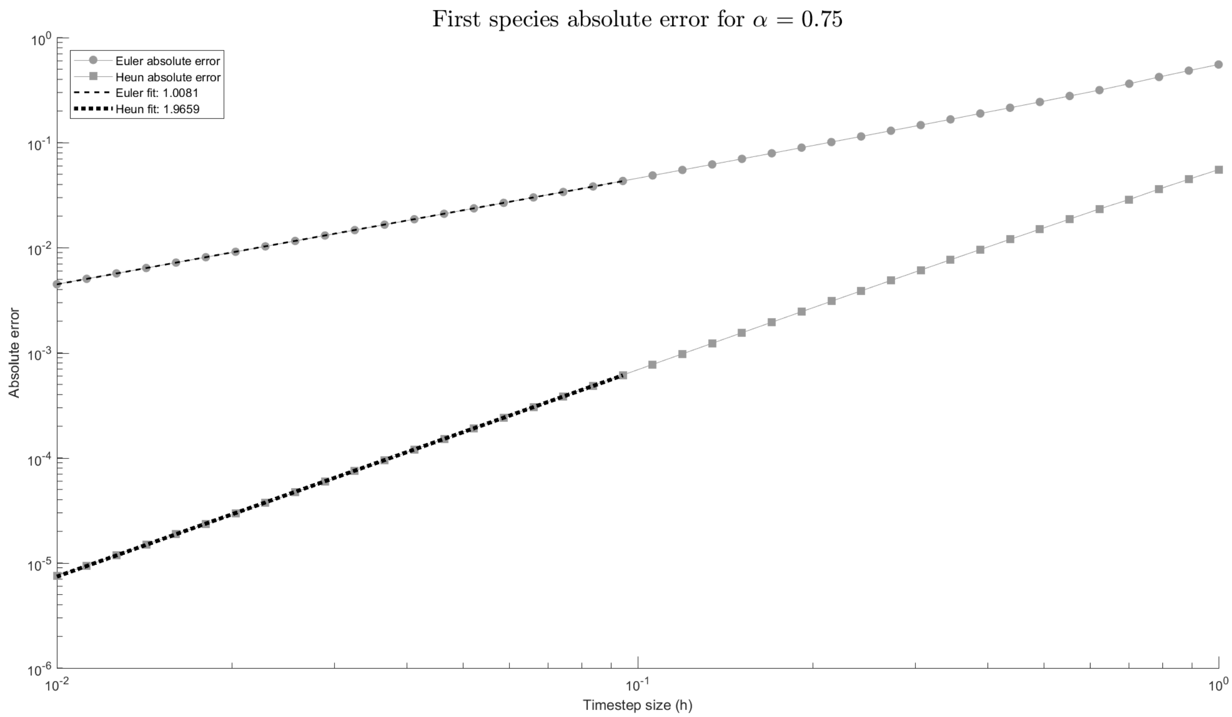


Figure 10: Error for different timestep size ($\alpha = 0.75$)

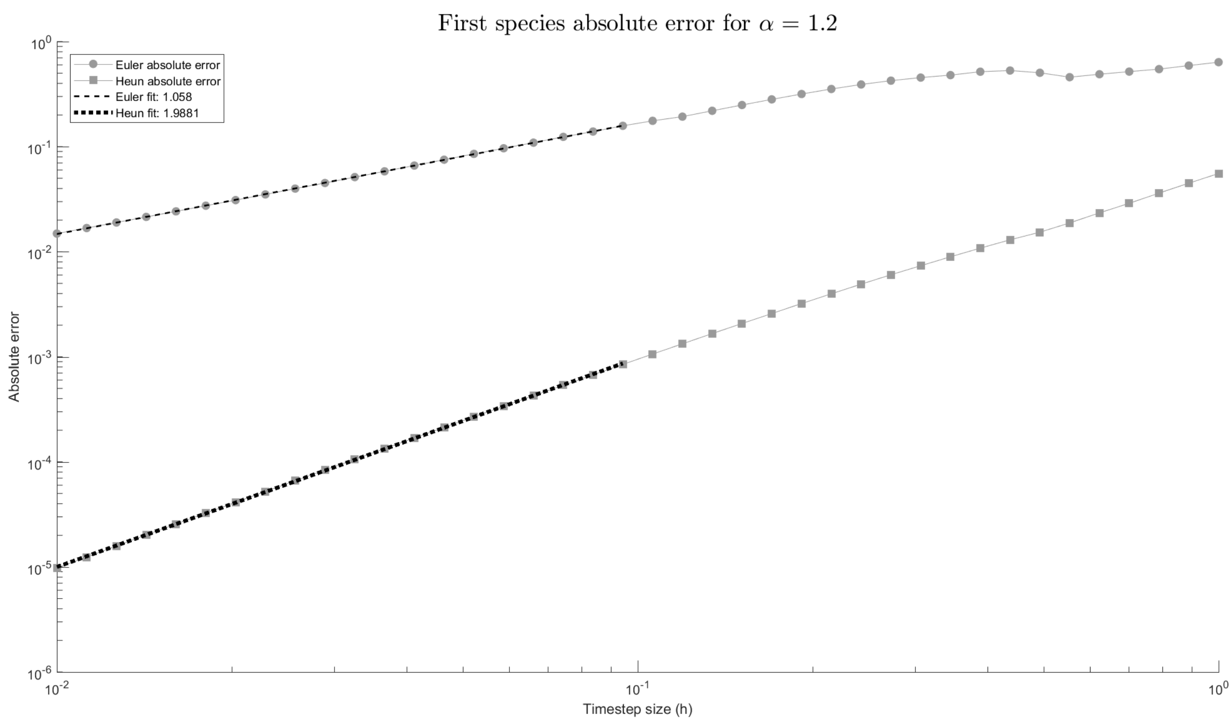


Figure 11: Error for different timestep size ($\alpha = 1.2$)

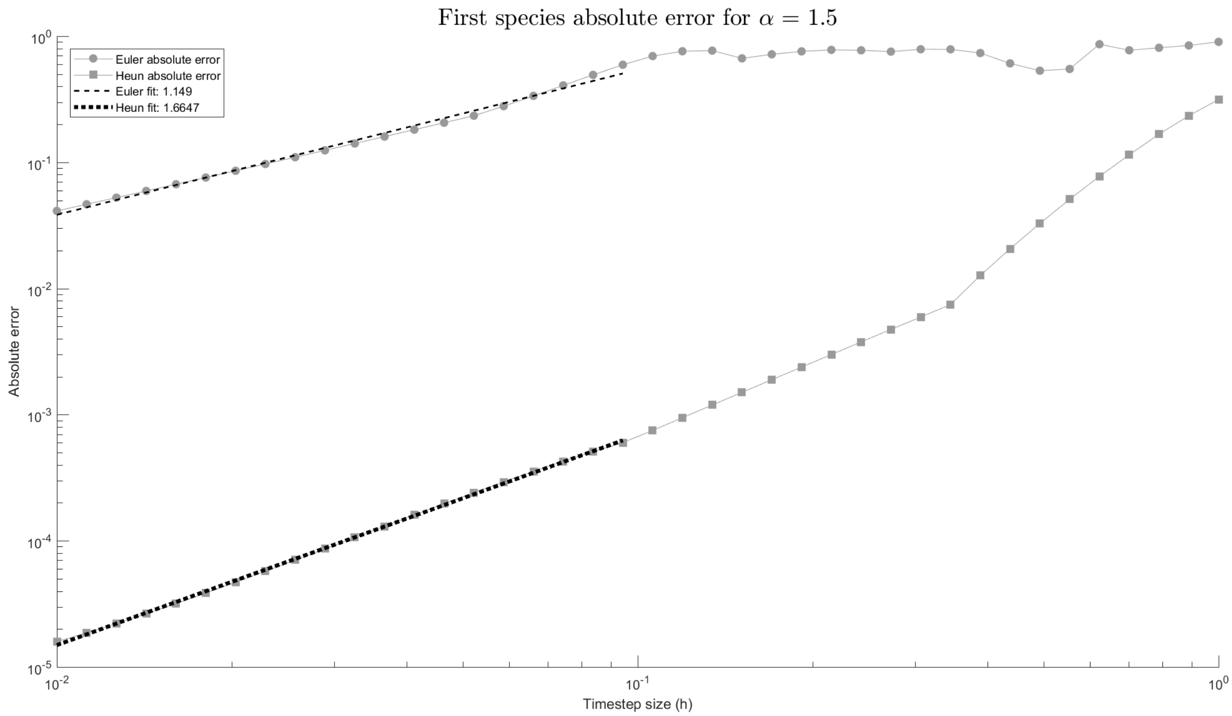


Figure 12: Error for different timestep size ($\alpha = 1.5$)

It can be seen that Euler blows up when the value of $\alpha = 1.5$ and $h = 1$. When α increases, both oscillations and amplitude increases. When h increases, the definition of the curve is worse, having bigger differences in the low order methods - compared to the Runge Kutta of 4th and 5th order. If both α and h increases, the system ends up with the failure of the Euler method. Another thing that must be noticed is that these methods have errors that are propagated with time. Thus, it is common to see the same shape with an offset depending on the method used. Euler is the lowest order method of the three and this can be seen in the figure of $\alpha = 0.75$ and $h = 0.5$, where both Heun and RK45 give more or less the same values while Euler solution differs a lot.

The log-log plot represents the error depending on the timestep size. This simulation has been done with a maximum simulation time of 50 seconds, so the blow up that Euler has between $\alpha = 1.5$ and $h = 1$ will not appear here. There is some random behavior in the errors for higher h and higher α. However, for lower h the error approach a line whose slope is ~ 1 for the Euler method and ~ 2 for the Heun method. This slope values represent the order of accuracy of each integration method.