Warsaw, 4.VI.2019

Numerical Methods (ENUME)

# Assignment C: Solving ordinary differential equations

Final Report

**Name:** Jan Radzimiński

**Index number:** 293052

**Supervisor:** Andrzej Miękina Ph.D.

Warsaw, 4.VI.2019

# Table of contents:

## 0. General Remarks

This project was implemented using Matlab environment. The Matlab script containing solutions to all tasks is called ‘’AssC_JR.m”.

After presenting solution to a given task, script stops (by function “pause” ) and it waits for any key to be pressed to move on to the next task.

Also, after every task, script calls “clear”, “close all”, “hold off” and “clc” functions, so tasks don’t interfere with each other and each task is more readable.

At the end of the report can be found original page with given assignment and printed Matlab code used to solve given problems.

# 1. Task 1 – Lobatto IID method for solving ODE

## 1.1. Introduction

The **first task** was about creating a program for solving differential equation:

$$9y'' + 6y' + 10y = 0 \quad for\ t \in [0, 10], \qquad y(0) = 0, \quad y'(0) = 2$$

by means of implicit Lobatto IID order 2 method, defined by following Butcher's table:

$$\begin{array}{c|cc} 0 & \frac{1}{2} & \frac{1}{2} \\ 1 & -\frac{1}{2} & \frac{1}{2} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

Lobatto IID order 2 is a Runge Kutta implicit method for solving differential equations. It was introduced in 1981 by Nørsett and Wanner.

The task was to compere obtained solution with built-in Matlab method ''ode113'' with constant integration step h = 0.01 . For the best solution, ode113 options settings parameters RelTol and AbsTol were set to their minimal values.

## 1.2. Implementation of method

The first problem with providing solution was the second order of given equation – we are only able to find original function with only first derivative, the method does not work with second derivative. Therefore, there was a need of using given substitution:

$$y_2 = y_1'$$

$$9{y_2}' + 6y_2 + 10y_1 = 0$$

which, after transformation, can be shown as:

$$y_1' = y_2$$

$$y_2' = -\frac{6}{9}y_2 - \frac{10}{9}y_1$$

where $y_1$ is the function that we are looking for.

Therefore, with such form, the problem was to solve problem, by solving these two first order differential equations at the same time.

In my solution, I am treating $y_1$ and $y_2$ as matrix y consisting values of these two functions (so actually values of $y_1$ and $y'_1$). The coefficients of the above equations are stored in matrix A:

$$A = \begin{bmatrix} 0 & 1 \\ -10/9 & -6/9 \end{bmatrix}$$

The 2nd order Lobatto IIID method yields:

$$y_n = y_{n-1} + \frac{h}{2}(f_1 + f_2)$$

Where:

$$f_1 = A \cdot [y_{n-1} + \frac{h}{2}(f_1 + f_2)]$$

$$f_2 = A \cdot [y_{n-1} + \frac{h}{2}(-f_1 + f_2)]$$

Which may be also shown in given way:

$$\begin{bmatrix} I - \frac{A \cdot h}{2} & -\frac{A \cdot h}{2} \\ \frac{A \cdot h}{2} & I - \frac{A \cdot h}{2} \end{bmatrix} \cdot \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} A \cdot y \\ A \cdot y \end{bmatrix}$$

Where I is 2x2 identity matrix.

Since we are looking for $f_1$ and $f_2$, this system of equations may be solved by means of pseudo-inversion Matlab operator '\'. The result is 4x1 vector were first two values are $f_1$ coefficients for $y_1$ and $y_2$ and last two values are $f_2$ coefficients for these functions.

After these calculations we have vector f with $f_1$ and $f_2$ coefficients for both $y_1$ and $y_2$ for given iteration so we may calculate:

$$y_1(n) = y_1(n-1) + \frac{h}{2}(f(1) + f(3))$$

$$y_2(n) = y_2(n-1) + \frac{h}{2}(f(2) + f(4))$$

And after 10/h (which for h=0.01 is 1000) iterations $y_1$ is the solution to given ODE.

## 1.3. Graphs

### 1.3.1. Functions obtained with ode113 function:

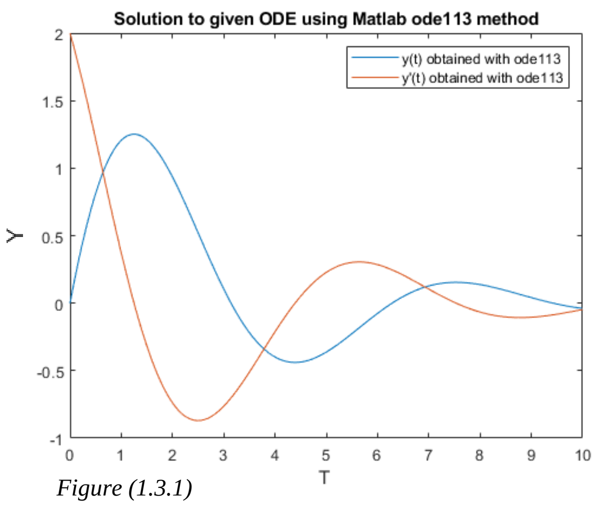

*Figure (1.3.1)*

### 1.3.2. Functions obtained with my Lobatto IIID function:

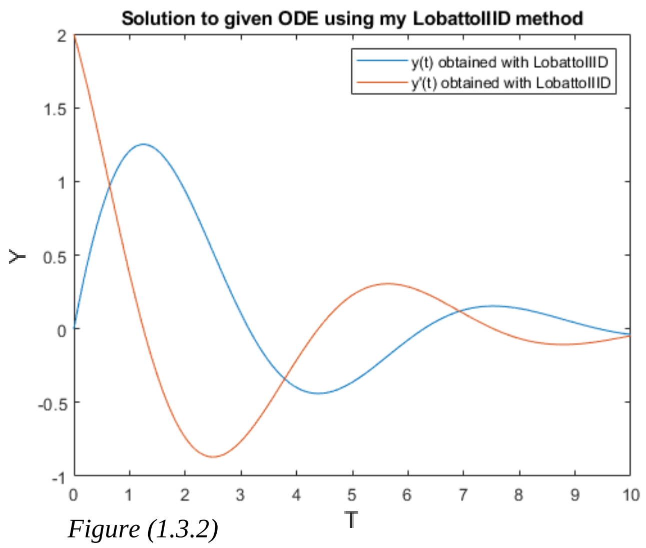

*Figure (1.3.2)*

### 1.3.3. Function y(t) obtained by both methods:

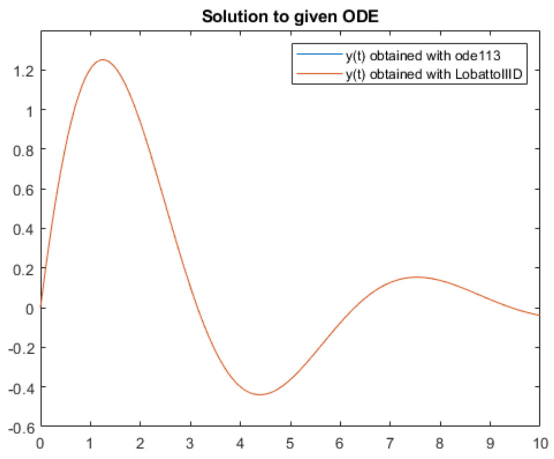

*Figure (1.3.3)*

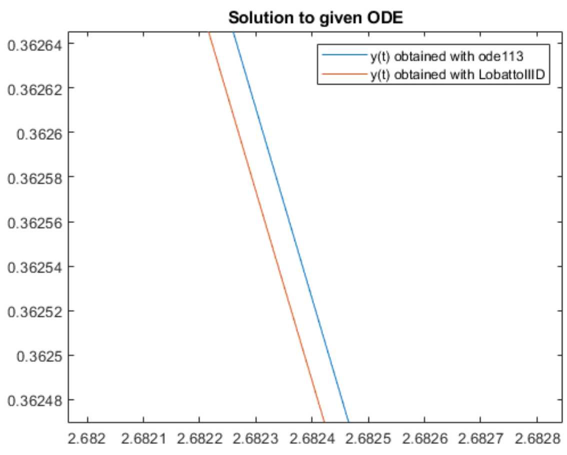

*Figure (1.3.4)*

### 1.3.4. Conclusions:

As can be seen on Figure 1.3.3 functions obtained with implemented by me Lobatto IIID method and built-in matlab ode113 are almost identical – only after zooming very closely to part of a curve (Figure 1.3.4) we can see a small difference between two obtained curves.

Therefore, for given h=0.01, Lobatto IIID method works very well, almost indistinguishable from ‘’ode113’’ Matlab method.

# 2. Task 2 & 3 – Dependence of the accuracy on the integration step h

## 2.1. Introduction

In the second task, the main goal was to carry out a systematic investigation of the dependence of the accuracy of the solution (obtained by means of Lobatto IIID method from Task 1) on the integration step h and comparison obtained results with accuracy of the solution obtained with explicit Euler method.

For calculating errors for this task, two indicators was used:

$$\delta_2(h) = \frac{||\hat{y}(t;h) - \dot{y}(t,h)||_2}{||\dot{y}(t,h)||_2} \qquad \rightarrow \textit{the root mean suare error}$$

$$\delta_\infty(h) = \frac{||\hat{y}(t;h) - \dot{y}(t,h)||_\infty}{||\dot{y}(t,h)||_\infty} \qquad \rightarrow \textit{the maximum error}$$

For better representation for small values of h, I have made vector h for which errors are calculated with usage of logspace function (instead of linspace). Also all graphs are made in logarithmic scale (by loglog function).

## 2.2. Graphs

### 2.2.1. Root Mean Square Errors:

#### 2.2.1.1. Without single points marked:

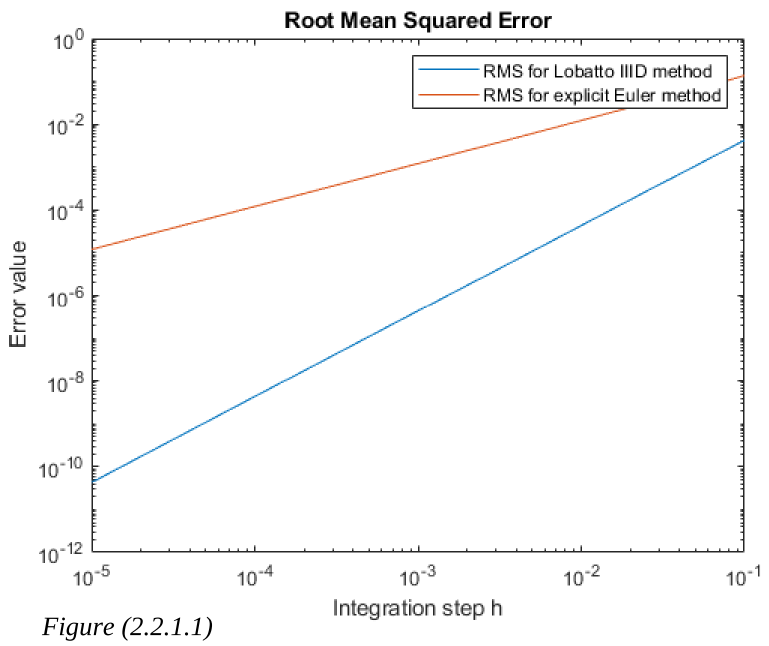

*Figure (2.2.1.1)*

2.2.1.2. With single points marked:

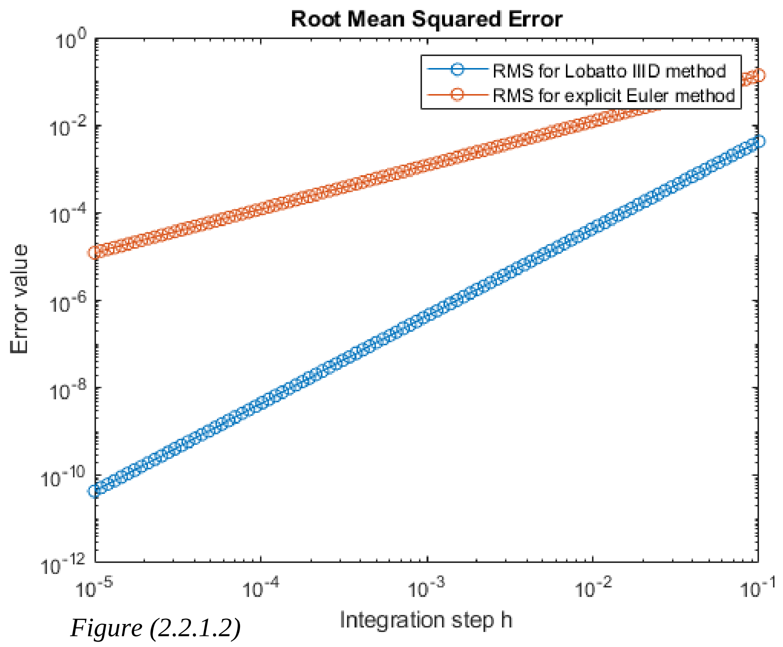

*Figure (2.2.1.2)*

### 2.2.2. Maximum Errors:

2.2.2.1. Without single points marked:

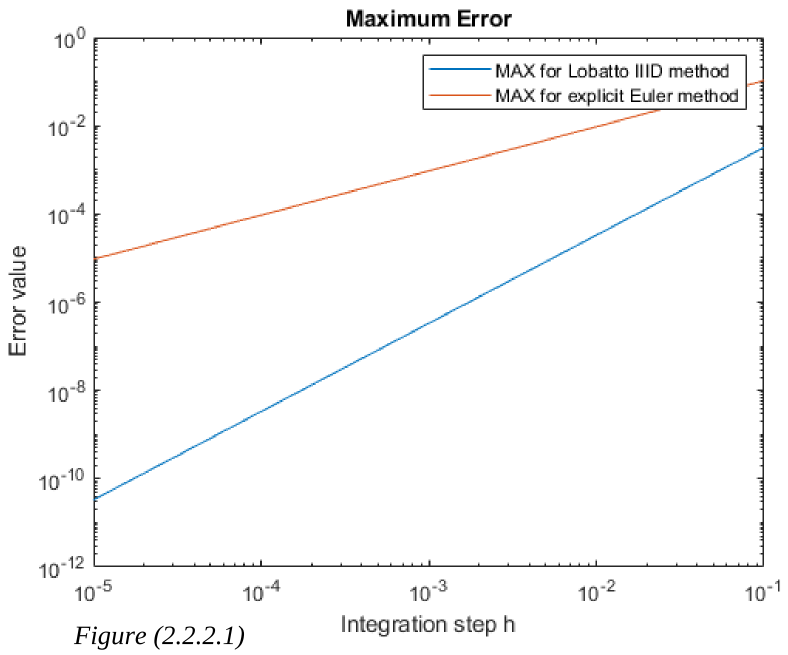

*Figure (2.2.2.1)*

2.2.2.2. With single points marked:

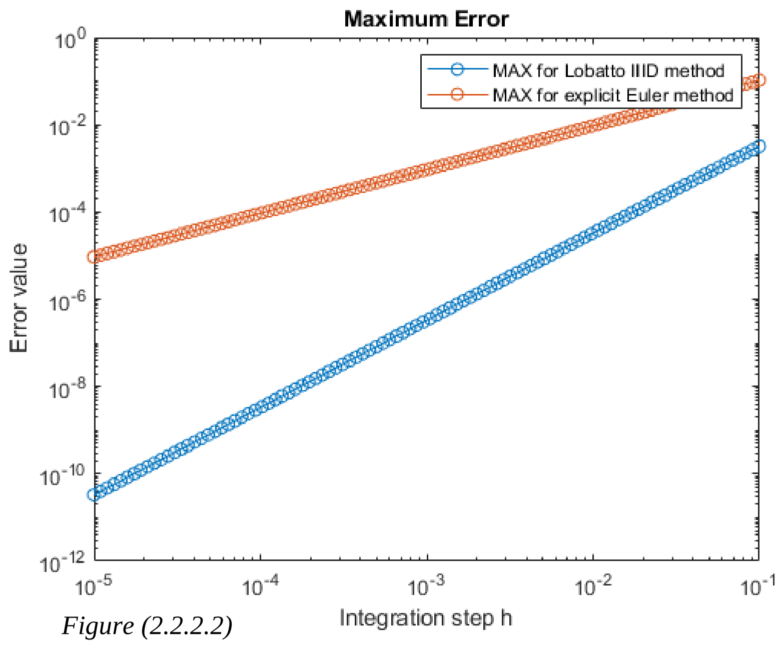

*Figure (2.2.2.2)*

## 2.3. Conclusions

For both methods, RMS error and Max error are increasing linear function – therefore, as expected, the smaller integration step h, the more accurate the solution is for both, Lobatto IIID and explicit Euler methods.

We can also see, that Lobatto IIID method has smaller values of both RMS and Max errors than explicit Euler for all values of integration steps h.

Moreover, Lobatto IIID method has bigger slope of functions describing both errors. Therefore, these functions for even smaller h will decrease more rapidly, so, with the same decrease of integration step for both methods, Lobatto IIID will have even more accurate solutions for smaller h's than Euler (for Euler method with same change of h errors will decrease, but much slower than in Lobatto case).

That follows the theory, because for ordinary differential equations, implicit methods generally work much better than explixit methods. Moreover, implemented in this assignment Lobatto IIID is a 2nd order method while Euler is 1st order method, so Lobatto should have better and better results  for smaller integration steps, while Euler not that much – and this is actually what we can see on graphs.

## 3. Script functions description

- `function [t, y] = Lobatto(tspan, y0, A, h)`
  - Function solving second order differential equation by means of implicit Lobatto IIID method. It takes vector of points – steps for which next values of function will be calculated, vector of two initial y0 values – values at $t_0$ of y(t) and y'(t), coefficients of system of equations after substitutions and step – h.
- `function [t, y] = Euler(tspan, y0, A, h)`
  - Function solving second order differential equation by means of explicit Euler method. It takes same arguments as previous one.
- `function [Y] = S2(t, y0, A, h, opt, r)`
  - Function that takes calculates Root Mean squared error of a solution with given h, calculated with Lobatto or Euler method. It takes same arguments as previous functions, optimized options for ode113 that is used as reference as 'ot' argument and r argument which if is 'L' it calculates error for Lobatto method or for Euler otherwise.
- `function [Y] = Sinf(t, y0, A, h, opt, r)`
  - Function that calculates Maximum error of solution with given h of Lobatto or Euler method. Arguments same as in previous one.
- `function dydt = odefcn(y, A)`
  - function used in ode113 method that allows it solving second order equations. Since it takes also matrix 'A' as argument it is universal even for different initial equations.

```
% JAN RADZIMINSKI (293052)
% ==== ENUME - ASSIGNMENT C PROJECT ====
hold off;
close all;
clear;
clc;

% ==== Task 1 ====
% Initial Data
y0 = [0 2];

% Note: Since I wanted to make my program more
general, both ode113 and my
% Lobatto and Euler functions take A as
argument, so they should work
% also for similar equations with different
coefficients - one should just
% change A matrix
A = [0, 1; -10/9, -6/9];

h=0.01;
tspan = 0:h:10;

% Matlab ode113 Solution
options = odeset('AbsTol', 1*10^(-16),
'RelTol', 2.22045*10^-14);
[tm,ym] = ode113(@(tm,ym) odefcn(ym, A), tspan,
y0, options);
figure(1)
plot(tm,ym(:,1),'-') %t,y(:,2),'.-')
hold on

% My Solution with Lobatto IID method
[X, Y] = Lobatto(tspan, y0, A, h);
figure(1)
plot(X, Y(1, :))
hold on
title('Solution to given ODE')
xlabel('T')
ylabel('Y')
legend('y(t) obtained with ode113', 'y(t)
obtained with LobattoIIID')


pause;

% ==== Task 2 & 3 ====
clc;
hold off;
close all;

% Not: With first line of hi activated program
works quite fast when with second
% line one should wait quite a while till
calculations are finished
hi = logspace(-1, -3, 30);
% hi = logspace(-1, -5, 500);
k=1;

for i = hi
    tspan = 0:i:10;
    RMS_L(k) = S2(tspan, y0, A, i, options,
'L');
    MAX_L(k) = Sinf(tspan, y0, A, i, options,
'L');
    RMS_E(k) = S2(tspan, y0, A, i, options,
'E');
    MAX_E(k) = Sinf(tspan, y0, A, i, options,
'E');
    k=k+1;
end
```

```
% Plotting RMS Errors
figure(1)
loglog(hi, RMS_L, '-');
hold on;
loglog(hi, RMS_E, '-');
hold on;
title('Root Mean Squared Error')
xlabel('Integration step h')
ylabel('Error value')
legend('RMS for Lobatto IIID method', 'RMS for
explicit Euler method')

% Plotting Maximum Errors
figure(2)
loglog(hi, MAX_L, '-');
hold on;
loglog(hi, MAX_E, '-');
hold on;
title('Maximum Error')
xlabel('Integration step h')
ylabel('Error value')
legend('MAX for Lobatto IIID method', 'MAX for
explicit Euler method');


% ==== FUNCTIONS USED: ====
% Lobatto IIID implicit Runge Kutta method:
function [t, y] = Lobatto(tspan, y0, A, h)
y(1, 1) =  y0(1);
y(2, 1) = y0(2);
I = eye(size(A));
M = [I-((1/2)*h*A), -(1/2)*h*A; (1/2)*h*A, I-
((1/2)*h*A)];
t=tspan;
for i=2:size(t, 2)
    R=[A*y(:, i-1); A*y(:, i-1)];
    f = (M\R);
    y(1, i) = y(1, i-1) + h/2*( f(1) + f(3) );
    y(2, i) = y(2, i-1) + h/2*( f(2) + f(4) );
end
end

% Euler explicit Runge Kutta method:
function [t, y] = Euler(tspan, y0, A, h)
y(1, 1) =  y0(1);
y(2, 1) = y0(2);
t=tspan;

for i=2:size(t, 2)
    y(:, i) = y(:, i-1) + h*A*y(:, i-1);
end
end

% RMS Error (opt are ode113 options and r is
which error (for Lobatto 'L' for Euler sth
else):
function [Y] = S2(t, y0, A, h, opt, r)

if r == 'L'
[t, y1] = Lobatto(t, y0, A, h);
else
[t, y1] = Euler(t, y0, A, h);
end

[~,ym] = ode113(@(tm,ym) odefcn(ym, A), t, y0,
opt);
ym=ym.';
Y=norm((y1(1, :) - ym(1, :)))/norm(ym(1, :));
end

% Max Error (opt are ode113 options and r is
which error (for Lobatto 'L' for Euler sth
else):
```

```
% Max Error (opt are ode113 options and r is
which error (for Lobatto 'L' for Euler sth
else):
function [Y] = Sinf(t, y0, A, h, opt, r)

if r == 'L'
[t, y1] = Lobatto(t, y0, A, h);
else
[t, y1] = Euler(t, y0, A, h);
end

[~,ym] = ode113(@(tm,ym) odefcn(ym, A), t,
y0, opt);
ym=ym.';
Y=norm(y1(1, :) - ym(1, :),
'inf')/norm(ym(1, :), 'inf');
end

% Function used in ode113 method
function dydt = odefcn(y, A)
dydt = zeros(2,1);
dydt(1) = A(1, 1)*y(1) + A(1, 2)*y(2);
dydt(2) = A(2, 1)*y(1)+ A(2, 2)*y(2);
end
```