ENUME project: assignemnent A #12

Samuele Ferraro

Warsaw University of Technology, Faculty of Electronics and Information Technology

8 June 2023

I declare that this piece of work, which is the basis for recognition of achieving learning out-comes in the Numerical Methods course, was completed on my own.

Contents

1 Mathematical symbols used

- x = general purpose independent variable.
- t = general purpose independent variable.
- y = general purpose dependent variable.
- e_n= Global error of a solving IVP method.
- h=step of integration for IVP solving methods
- $f(\cdot)$ = general purpose function as $y = f(x)$

2 Introduction: Ordinary differential equations

2.1 Definition: Initial Value Problem for an ODE

An ordinary differential equation (ODE) is a differential equation dependent on only one single independent variable

An Initial Value Problem (IVP), known also as Cauchy problem, consist in finding the functions $y_1(t)....y_m(t)$ satisfying the set of ODEs in Eq.(1):

$$\frac{dy_m(t)}{dt} = f_m(t, y_1(t), ..., y_m(t)) \text{ for } t \in [0, T] \tag{1}$$

With given initial values $y_m(0)$ for m= 1,...,M.

2.2 Design of numerical methods for solving IVPs

A numerical method for solving an IVP is defined as a recursive operator in Eq.(2):

$$y_n \equiv \mathcal{S}(y_{n-1}, y_{n-2}...) \text{ for } n = 0, ..., N \tag{2}$$

It generates a sequence $\{y_n\}$ that is an estimate of the sequence $\{y(t_n)\}$ where
$0 = t_0 \leq t_1 \leq t_2 \leq ... \leq t_n = T$

These method can be divided in:

- Single-step methods
- Multi-step methods

2.3 Convergence of methods for solving IVPs

A numerica method for solving an IVPs is convergent if for any set of ODEs, having a unique solution $y(t)$, the approximate solution $\{y_n\}$ that depends on the integration step h, converges to $\{y_(t_n)\}$ for $h \to 0$. The global error of the numerical solution is diminishing to zero:

$$e_n \equiv y_n - y(t_n) \longrightarrow 0 \text{ for } n = 0, 1.... \tag{3}$$

2.4 Examples of single-step methods for solving IVPs

A single-step method can be generalized using the Runge-Kutta formula in Eq.(4)

$$y_n = y_{n-1} + h \cdot \sum_{k=1}^{K} w_k f_k \tag{4}$$

where:

$$f_k = f\left(t_{n-1} + c_n h, y_{n-1} + h \cdot \sum_{p=1}^{K} a_{k,p} f_p\right) \text{ for } k = 1, 2, ..., K$$

$$c_k \in [0, 1] \text{ and } w_k \in [0, 1] \text{ for } k = 1, 2, ..., K$$

$$\sum_{k=1}^{K} w_k = 1$$

The values of a_k, w_k and $a_{k,p}$ are defined by the Butcher table (5):

$$\begin{array}{c|cccc} c_1 & a_{1,1} & a_{1,2} & \dots & a_{1,K} \\ c_2 & a_{2,1} & a_{2,2} & \dots & a_{2,K} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_K & a_{K,1} & a_{K,2} & \dots & a_{K,K} \\ \hline & w_1 & w_2 & \dots & w_K \end{array} \tag{5}$$

2.4.1 Euler explicit method

The Euler explicit method is defined in Eq.(6):

$$y_n = y_{n-1} + h \cdot f(t_{n-1}, y_{n-1}) \tag{6}$$

Eq.(6) can be derived using Eq.(4) with the Butcher table (7):

$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array} \tag{7}$$

2.4.2 Euler implicit method

The Euler implicit method is defined in Eq.(8):

$$y_n = y_{n-1} + h \cdot f(t_n, y_n) \tag{8}$$

Eq.(8) can be derived using Eq.(4) with Butcher table (9)

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array} \tag{9}$$

2.5 Examples of linear multi-step methods

The K-step linear method is defined by Eq.(10):

$$y_n = \sum_{k=1}^{K_\alpha} \alpha_k y_{n-k} + h \sum_{k=p}^{K_\beta} \beta_k \cdot f(t_{n-k}, y_{n-k}) \text{ with } y_0 = y(0), ..., y_{K-1} = y((K-1)h) \tag{10}$$

where $p = 0$ or $p = 1$, $K_\alpha \geq 1$, $K_\beta \geq p$ and $max\{K_\alpha, K_\beta\} = K$.
The method is called:

- *explicit* if $p = 1(\beta_0 = 0)$ because y_n depends only on $y_{n-1}, y_{n-2}, ..., y_{n-K}$ and $f(t_{n-1}, y_{n-1}, f(t_{n-2}, y_{n-2}, ..., f(t_{n-K}, y_{n-K})$
- *implicit* if $p = 1(\beta_0 \neq 0$ since y_n depends also on $f(t_n, y_n)$. Then it is necessary to solve a non linear algebraic equation Eq.(11):

$$\begin{aligned} & y_n - \beta_0 \cdot f(t_n, y_n) = F(y_{n-1}, y_{n-2}, ...) \\ & \text{where } F(y_{n-1}, y_{n-2}, ..) = \sum_{k=1}^{K_\alpha} \alpha_k y_{n-k} + h \sum_{k=1}^{K_\beta} \beta_k \cdot f(t_{n-k}, y_{n-k}) \end{aligned} \tag{11}$$

2.5.1 Adams-Bashforth method

It is an explicit method which formula is represented in Eq.(12):

$$y_n = y_{n-1} + h \sum_{k=1}^{K} \beta_k f(t_{n-k}, y_{n-k}) \tag{12}$$

Whose coefficient are defined in Table (13), where K is number of the steps:

K	β_1	β_2	β_3
1	$-\frac{1}{2}$	$---$	$---$
2	$\frac{3}{2}$	$-\frac{1}{2}$	$---$
3	...	...	...

(13)

Table (13) considers $K = 2$ as the maximum value, as required in Task #2 (§3).

2.5.2 Adams-Moulton method

It is an implicit method defined by the formula Eq.(14):

$$y_n = y_{n-1} + h \sum_{k=0}^{K} \beta_k f(t_{n-k}, y_{n-k}) \tag{14}$$

Whose coefficient are defined in Table (15), where K is the number of the steps:

K	β_0	β_1	β_2	β_3
0	1	− − −	− − −	− − −
1	$\frac{1}{2}$	$\frac{1}{2}$	− − −	− − −
2	$\frac{5}{12}$	$\frac{8}{12}$	$-\frac{1}{12}$	− − −
3	...	...	...	...

(15)

As said in §2.5.1, Table (15) considers $K = 2$ as the maximum value,

One can notice that the only difference between Eq.(12) and Eq.(14) is the starting point of k, that confirms the behaviour of the methods as explicit or implicit, as said in §2.5.

3 Assignment C

1. Use the MATLAB function ***ode45*** to solve the Lotka-Volterra equations:

$$\begin{cases} \dfrac{dx(t)}{dt} = p_1 x(t) - p_2 x(t) y(t) \\ \dfrac{dy(t)}{dt} = p_3 x(t) y(t) - p_4 y(t) \end{cases} \tag{16}$$

   for $t \in [0, 1]$, $p_1 = 13$, $p_2 = 0.14$, $p_3 = 0.06$, $p_4 = 13$, $x(0) = 520$, y(0)=30

2. Solve the set of equations Eq.(16) for , $t \in [0, 1]$ by means of

   - Euler explicit method
   - Euler implicit method
   - Adams-Moulton method or order 2
   - Adams-Bashforth method of order 2

   Use the step of integration $h = 0.005$.

3. Compute the relative aggregated errors Δ_x of the solutions obtained in Task #2:

$$\Delta_x = \frac{\sum_{n=1}^{N} (\hat{x}_n - \dot{x}_n)^2}{\sum_{n-1}^{N} \dot{x}_n^2} \tag{17}$$

   where

   - $\hat{x}_1, ..., \hat{x}_N$ are the estimates of x, obtained by means of one of the methods specified in Task #2;
   - $\dot{x}_1, ..., \dot{x}_N$ are the reference values of x, obtained by means of the function ***ode45*** with the parameters $RelTol = 10^{-8}$ and $AbsTol = 10^{-12}$.

4. Determine the dependence of the relative aggregated error Δ_x on step of integration $h \in [10^{-4}, 10^{-2}]$ for all four methods specified in Task #2.

3.1 Task #1: Solution by *ode45*

The Lokta-Volterra set of ODEs (Eq.(16)) has been implemented using the Matlab code in §4.1 and solved with ***ode45*** in §4.2. The plots of the solution by §4.2 is presented in Fig.1:

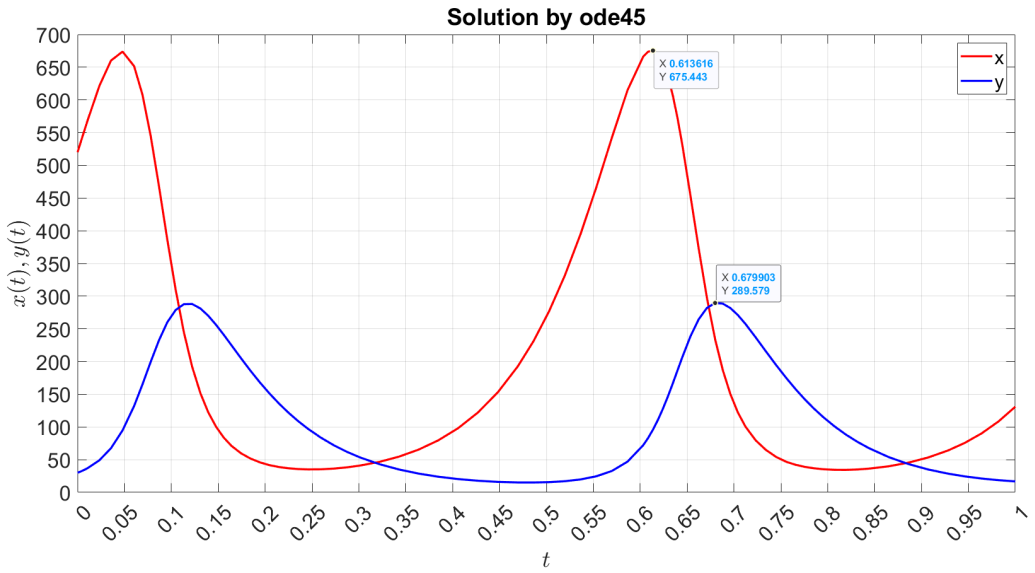

Figure 1: Plot of $x(t)$ and $y(t)$

In Fig.1 the peak values can be observed.

3.2 Task #2: implementation of different IVP solving algorithms

3.2.1 Euler explicit method

The method (§2.4.1) as been implemented using code in §4.3 and tested on Eq.(16) in code §4.4. The resulting plot from §4.4 is shown in Fig.2.

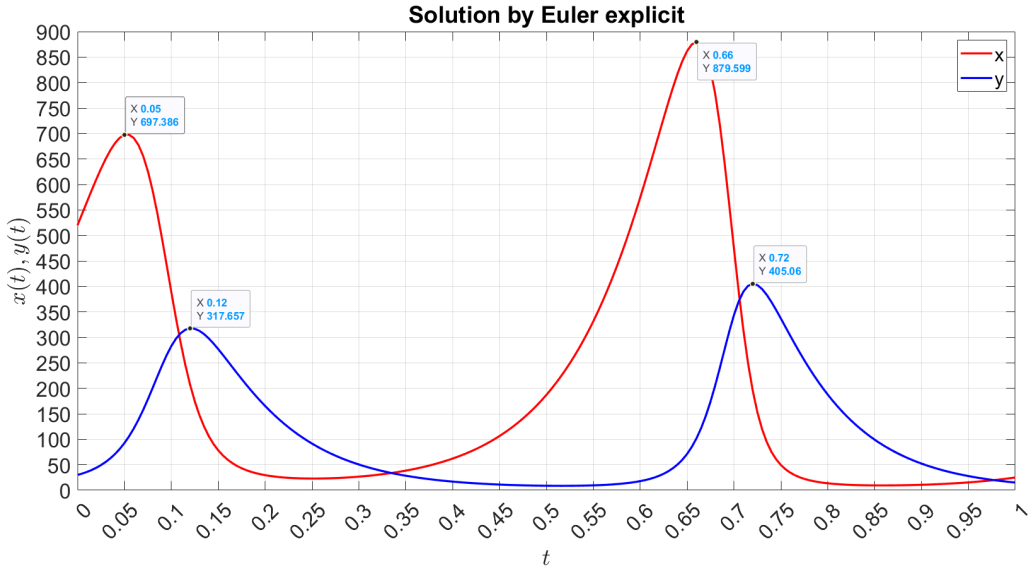

Figure 2: Plot of $x(t)$ and $y(t) using Euler explicit method$

One can observe that the plots are similar to Fig.1 one, but there are some differences:

- Peak values: Fig.1 $y(t)$ has a peak value of 675.443, while Fig.2 $y(t)$ one is 879.599.
- The shape: Fig.2 plot shape is dilated in confront of Fig.1 one, considering values of $t > 0.4$

3.2.2 Euler implicit method

The method §2.4.5 has been implemented on code §4.5 that implements the solution of the equations set Eq. (19):

$$\begin{cases} x_n = \dfrac{x_{n-1}}{1 - h * (p_1 - p_2 y_n)} \\ y_n = y_{n-1} + h \left\{ p_3 \left(\dfrac{x_{n-1}}{1 - h(p_1 - p_2 y_n)} \right) y_n - 13 y_n \right\} \end{cases} \tag{19}$$

The function have been applied to Eq.(16) through code §4.6 whose plot is presented in Fig.3

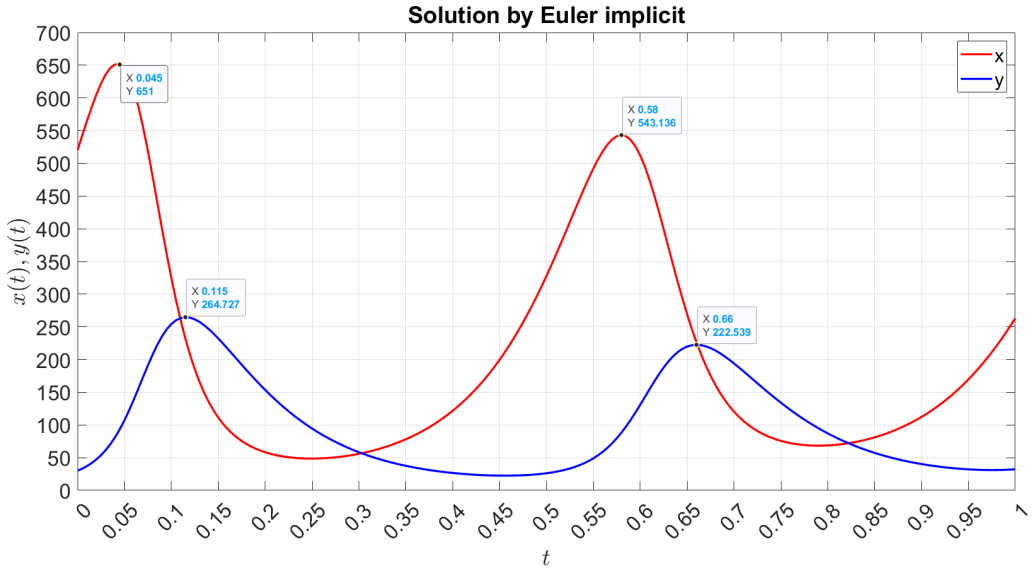

Figure 3: Plot of $x(t)$ and $y(t) using Euler explicit method$

One can notice that the solutions plotted in Fig.3 have an "opposite" behaviour in confront of the Euler explicit one (Fig.3): indeed, in this case the solution for $t > 0.4$ is compressed to smaller peaks values.

3.2.3 Adams-Bashforth method of order 2

The method (§2.5.1) has been implemented using script in §4.7 and tested on Eq.(16) in §4.8. The plot in Fig.4shows §4.8 results:

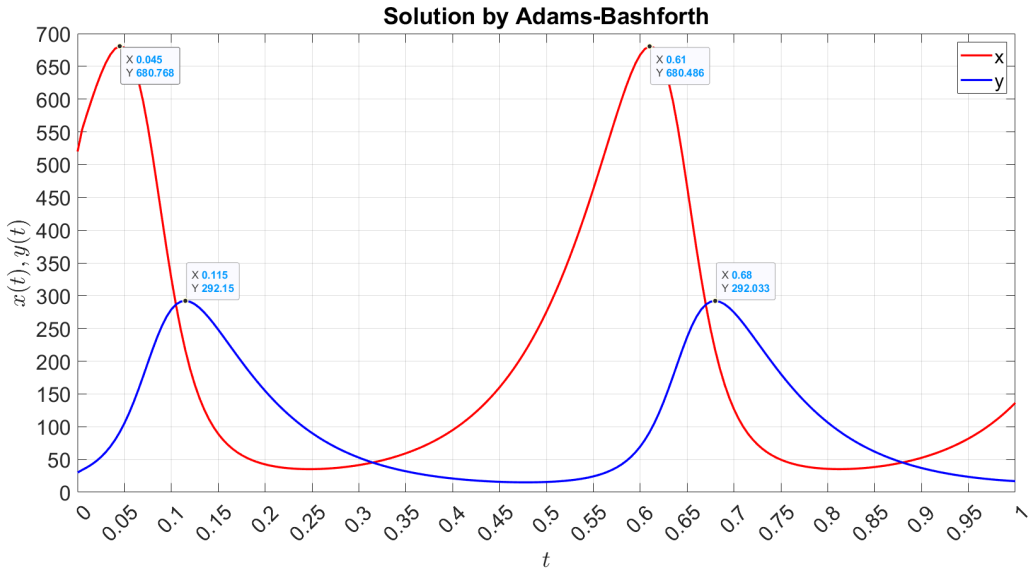

Figure 4: Plot of $x(t)$ and $y(t) using Adams - Bashforth method$

One can notice that the solution found using Adams-Bashforth method (Fig.4) is way more similar to Fig.1 than the Euler explicit one (Fig.2) and Euler implicit one(Fig.3) . The reason of this behaviour is the difference between single-step method (§2.4) and multi-step method(§2.5): using one previous step more in the calculations let Adams-Bashforth method reach a closer solution because it uses more data that Euler method. The choice of which one has to be used have to consider that Adams-Bashforth, even if it is more precise, uses more resources.

3.3 Task#3: calculation of aggregate error

The calculations of the aggregate error have been executed in code §4.9: this code calculates Δ_x error and Δ_y. In order to confront correctly, some basic interpolation methods have been used. The resulting data are shown in Fig.4:

	Δ_x	Δ_y
Euler Explicit	0,123429673	0,135422031
Euler Implicit	0,146903395	0,193077095
Adams-Bashforth	0,091823972	0,153466398

Figure 5: Aggregate errors for each method

One can notice that for $h = 0.005$ leads to an aggregated error between 9% and 19% but it does not explain a lot: there are no noticeable correlation between this results. At the moment, nothing consistent can be said about the precision of the methods.

3.4 Task #4: Relation between aggregate error and integration step

Task #4 has been implemented using code §4.10, whose plot is reported in Fig.5

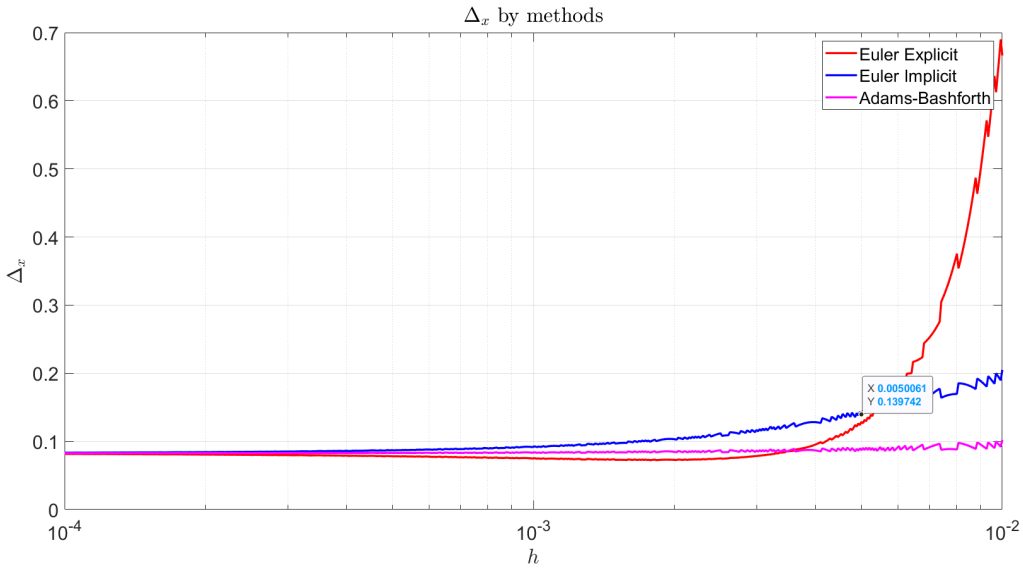

Figure 6: Plot of the aggregate error depending on the step of integration

In Fig.5 we can notice that if the step of integration is small enough, the aggregated error is almost the same for every method. But checking the highest values, it is possible to say something more:

- Implicit methods are more precise for big step of integration. This behaviour is expected: usually implicit methods allow to choose an higher h than the explicit ones, but with the trade off in spending resources in solving the non-linear equation.
- Adams-Bashforth method has good performances even if h gets higher. As said in §3.2.3, a multi step method uses more resources and data, but it guarantees a better error stability.

4 Appendix: MatLab codes used

4.1 lotka_volterra.m

```
function dydt = lotka_volterra(t,y,p1,p2,p3,p4);
    x=y(1);
    y=y(2);

    %Set implementation
    dxdt = p1*x-p2*x*y;
    dydt = p3*x*y-p4*y;

    %output
    dydt=[dxdt;dydt];
end
```

4.2 Task1.m

```
%parameters definition
tspan=[0,1];
p1=13;
p2=0.14;
p3=0.06;
p4= 13;
x0=520;
y0=30;
IC=[x0,y0];%Initial Conditions

%solving the set of ODE
[t, y] = ode45(@(t,y) lotka_volterra(t, y, p1, p2, p3, p4), tspan,IC);

%Taking the x and y function from the solution
x = y(:, 1);
y = y(:, 2);

plot(t, x, 'r', 'LineWidth', 2);  % Plot for x
hold on;
plot(t, y, 'b', 'LineWidth', 2);  % Plot for y
hold off;
grid on;
title('Solution by ode45');
xlabel("$t$", 'FontSize', 25, 'Interpreter', 'latex');
ylabel("$x(t), y(t)$", 'FontSize', 25, 'Interpreter', 'latex');
legend('x', 'y');
yticks ( 0 : 50 : 700  ) ;
xticks ( tspan(1) : 0.05: tspan(2)) ;
set(gca, 'FontSize', 20);
```

4.3 euler_exp.m

```
function [t,x,y] = euler_exp (odeSet, tspan, IC, h, p)

    %inizialization
    t=tspan(1):h:tspan(2);
    N=length(t);
    x=zeros(1, N);
    y=zeros(1, N);
```

```
    x(1)=IC(1);
    y(1)=IC(2);

    %implementig the algorithm
    for n= 2:N
        dydt = odeSet(t(n-1),[x(n-1), y(n-1)],p);
        x(n)=x(n-1)+h*dydt(1);
        y(n)=y(n-1)+h*dydt(2);
    end
end
```

4.4 euler_test.m

```
%parameters definition
tspan=[0,1];
p=[13, 0.14, 0.06, 13];
x0=520;
y0=30;
IC=[x0,y0];%Initial Conditions
h=0.005;


[t,x,y]=euler_exp(@(t,x,y)lotka_volterra(t,[x,y],p),tspan,IC,h,p);


plot(t, x, 'r', 'LineWidth', 2);  % Plot per x
hold on;
plot(t, y, 'b', 'LineWidth', 2);  % Plot per y
hold off;
grid on;
title('Solution by Euler explicit');
xlabel("$t$", 'FontSize', 25, 'Interpreter', 'latex');
ylabel("$x(t), y(t)$", 'FontSize', 25, 'Interpreter', 'latex');
legend('x', 'y');
yticks ( 0 : 50 : 900 ) ;
xticks ( tspan(1) : 0.05: tspan(2)) ;
set(gca, 'FontSize', 20);
```

4.5 euler_imp.m

```
function [t, x, y] = euler_imp(tspan, IC, h, p)
    % Initialization
    t = tspan(1):h:tspan(2);
    N = length(t);
    x = zeros(1, N);
    y = zeros(1, N);
    x(1) = IC(1);
    y(1) = IC(2);

    % Method implementation
    for n = 2:N
        % Current value definition
        y_curr=y(n-1);
        x_curr=x(n-1);

        %y_n equation that have to be solved
```

```
        f = @(y_next) y_curr + h*( p(3)*y_next*((x_curr)/(1-h*(p(1)-p(2)
            *y_next))) - p(1)*y_next)- y_next;

        %solutio of y_n
        y_next=fzero(f,y_curr);

        %calculation of x_next
        x_next= (x_curr)/(1-h*(p(1)-p(2)*y_next));


        % updating values
        x(n) = x_next;
        y(n) = y_next;
    end
end
```

4.6 Euler_imp$_t estm$

```
%parameters definition
tspan=[0,1];
p=[13, 0.14, 0.06, 13];
x0=520;
y0=30;
IC=[x0,y0];%Initial Conditions
h=0.005;

%calculation
[t,x,y]=euler_imp(tspan,IC,h,p);

%plotting
plot(t, x, 'r', 'LineWidth', 2);
hold on;
plot(t, y, 'b', 'LineWidth', 2);
hold off;
grid on;
title('Solution by Euler implicit');
xlabel("$t$", 'FontSize', 25, 'Interpreter', 'latex');
ylabel("$x(t), y(t)$", 'FontSize', 25, 'Interpreter', 'latex');
legend('x', 'y');
yticks ( 0 : 50 : 900 ) ;
xticks ( tspan(1) : 0.05: tspan(2)) ;
set(gca, 'FontSize', 20);
```

4.7 adams_bashforth.m

```
function [t, x, y] = adams_bashforth(odeSet, tspan, IC, h, p)
    % Initialization
    t = tspan(1):h:tspan(2);
    N = length(t);
    x = zeros(1, N);
    y = zeros(1, N);
    x(1) = IC(1);
    y(1) = IC(2);

    beta = [1.5, -0.5];

    for n = 2:N
```

```
        s = [0, 0];
        for k = 1:2
            if n-k>0
                ya=[x(n-k),y(n-k)];
                s = s + h*beta(k)*odeSet(t(n-k), ya, p);
            end
        end
        % Updating x and y values
        x(n) = x(n-1) + s(1);
        y(n) = y(n-1) + s(2);
    end
end
```

4.8 Adams_Bashforth_test.m

```
%parameters definition
tspan=[0,1];
p=[13, 0.14, 0.06, 13];
x0=520;
y0=30;
IC=[x0,y0];%Initial Conditions
h=0.005;

[t,x,y]=adams_bashforth(@(t,x,y)lotka_volterra(t,[x,y],p),tspan,IC,h,p);


plot(t, x, 'r', 'LineWidth', 2);  % Plot per x
hold on;
plot(t, y, 'b', 'LineWidth', 2);  % Plot per y
hold off;
grid on;
title('Solution by Adams-Bashforth');
xlabel("$t$", 'FontSize', 25, 'Interpreter', 'latex');
ylabel("$x(t), y(t)$", 'FontSize', 25, 'Interpreter', 'latex');
legend('x', 'y');
yticks ( 0 : 50:  800 ) ;
xticks ( tspan(1) : 0.05: tspan(2)) ;
set(gca, 'FontSize', 20)
```

4.9 task3.m

```
%initialization
tspan=[0,1];
p=[13,0.14,0.06,13];
x0=520;
y0=30;
IC=[x0,y0];%Initial Conditions
h=0.005;

%calculating the reference value.
options = odeset('RelTol', 1e-8, 'AbsTol', 1e-12);
[t_r, y_r] = ode45(@(t,y) lotka_volterra(t, y, p), tspan, IC, options);

%Solution by euler explicit method
[t_eulE, x_eulE, y_eulE]=euler_exp(@(t,x,y)lotka_volterra(t,[x,y],p),
   tspan,IC,h,p);
```

```
% Adapting the number of points: Every result have to been adapted to
    the reference

N = length(t_r); % Points by ode45
M = length(t_eulE); % points by Eulero explicit

% Adapting by interpolation
t_eulE_adapted = interp1(linspace(t_eulE(1), t_eulE(end), M), t_eulE,
    linspace(t_eulE(1), t_eulE(end), N));
x_eulE_adapted = interp1(t_eulE, x_eulE, t_eulE_adapted);
y_eulE_adapted = interp1(t_eulE, y_eulE, t_eulE_adapted);

N=length(t_eulE_adapted);

%calculation of the error by euler explicit method of x(t)
numerator=0;
denominator=0;
for i=1:N
    numerator = numerator + (x_eulE_adapted(i)-y_r(i,1))^2;
    denominator = denominator + (y_r(i,1)^2);
end
Delta_eulE_x=numerator/denominator;
disp(Delta_eulE_x);

%calculation of the error by euler explicit method of y(t)
numerator=0;
denominator=0;
for i=1:N
    numerator = numerator + (y_eulE_adapted(i)-y_r(i,2))^2;
    denominator = denominator + (y_r(i,2)^2);
end
Delta_eulE_y=numerator/denominator;
disp(Delta_eulE_y);
%-----------------------------------------------%
%Solution for euler implicit
[t_eulI,x_eulI,y_eulI]=euler_imp(tspan,IC,h,p);
% Adapting the number of points: Every result have to been adapted to
    the reference

N = length(t_r); % Points by ode45
M = length(t_eulI); % points by Eulero explicit

% Adapting by interpolation
t_eulI_adapted = interp1(linspace(t_eulI(1), t_eulI(end), M), t_eulI,
    linspace(t_eulI(1), t_eulI(end), N));
x_eulI_adapted = interp1(t_eulI, x_eulI, t_eulI_adapted);
y_eulI_adapted = interp1(t_eulI, y_eulI, t_eulI_adapted);

N=length(t_eulI_adapted);
%calculation of the error by euler explicit method of x(t)
numerator=0;
denominator=0;
for i=1:N
    numerator = numerator + (x_eulI_adapted(i)-y_r(i,1))^2;
    denominator = denominator + (y_r(i,1)^2);
end
Delta_eulI_x=numerator/denominator;
disp(Delta_eulI_x);
```

```
%calculation of the error by euler explicit method of y(t)
numerator=0;
denominator=0;
for i=1:N
    numerator = numerator + (y_eulI_adapted(i)-y_r(i,2))^2;
    denominator = denominator + (y_r(i,2)^2);
end
Delta_eulI_y=numerator/denominator;
disp(Delta_eulI_y);
%-----------------------------------------------%
%solution by Adams-Bashforth method
[t_AB, x_AB,y_AB]=adams_bashforth(@(t,x,y)lotka_volterra(t,[x,y],p),
   tspan,IC,h,p);
%Adapting of the points

N = length(t_r); % Points by ode45
M = length(t_AB); % points by Eulero explicit

% Adapting by interpolation
t_AB_adapted = interp1(linspace(t_AB(1), t_AB(end), M), t_AB, linspace(
   t_AB(1), t_AB(end), N));
x_AB_adapted = interp1(t_AB, x_AB, t_AB_adapted);
y_AB_adapted = interp1(t_AB, y_AB, t_AB_adapted);

N=length(t_AB_adapted);

%calculation of the error by Adams-Bashforth method of x(t)
numerator=0;
denominator=0;
for i=1:N
    numerator = numerator + (x_AB_adapted(i)-y_r(i,1))^2;
    denominator = denominator + (y_r(i,1)^2);
end
Delta_AB_x=numerator/denominator;
disp(Delta_AB_x)

%calculation of the error by Adams-Bashforth method of y(t)
numerator=0;
denominator=0;
for i=1:N
    numerator = numerator + (y_AB_adapted(i)-y_r(i,2))^2;
    denominator = denominator + (y_r(i,2)^2);
end
Delta_AB_y=numerator/denominator;
disp(Delta_AB_y);
```

4.10 Task4.m

```
tspan=[0,1];
p=[13,0.14,0.06,13];
x0=520;
y0=30;
IC=[x0,y0];%Initial Conditions
%creating the enough h points.
h=logspace(-4, -2, 600);

%calculating the reference value.
```

```
options = odeset('RelTol', 1e-8, 'AbsTol', 1e-12);
[t_r, y_r] = ode45(@(t,y) lotka_volterra(t, y, p), tspan, IC, options);

%Valueting Eulero Explicid method
Delta_eulE_x=zeros(1,length(h));
for a=1:length(h)

    [t_eulE, x_eulE, y_eulE]=euler_exp(@(t,x,y)lotka_volterra(t,[x,y],p)
       ,tspan,IC,h(a),p);

    %adapting points

    N = length(t_r); % Points by ode45
    M = length(t_eulE); % points by Eulero explicit

    % Adapting by interpolation
    t_eulE_adapted = interp1(linspace(t_eulE(1), t_eulE(end), M), t_eulE
       , linspace(t_eulE(1), t_eulE(end), N));
    x_eulE_adapted = interp1(t_eulE, x_eulE, t_eulE_adapted);
    y_eulE_adapted = interp1(t_eulE, y_eulE, t_eulE_adapted);

    numerator=0;
    denominator=0;
    for i=1:N
        numerator = numerator + (x_eulE_adapted(i)-y_r(i,1))^2;
        denominator = denominator + (y_r(i,1)^2);

    end
    Delta_eulE_x(a)=numerator/denominator;
end


%
   ------------------------------------------------------------------------%

%Solution for euler implicit

Delta_eulI_x=zeros(1,length(h));

for a=1:length(h)

    [t_eulI,x_eulI,y_eulI]=euler_imp(tspan,IC,h(a),p);
    % Adapting the number of points: Every result have to been adapted
       to the reference

    N = length(t_r); % Points by ode45
    M = length(t_eulI); % points by Eulero explicit

    % Adapting by interpolation
    t_eulI_adapted = interp1(linspace(t_eulI(1), t_eulI(end), M), t_eulI
       , linspace(t_eulI(1), t_eulI(end), N));
    x_eulI_adapted = interp1(t_eulI, x_eulI, t_eulI_adapted);
    y_eulI_adapted = interp1(t_eulI, y_eulI, t_eulI_adapted);

    N=length(t_eulI_adapted);
    %calculation of the error by euler explicit method of x(t)
    numerator=0;
    denominator=0;
```

```
    for i=1:N
        numerator = numerator + (x_eulI_adapted(i)-y_r(i,1))^2;
        denominator = denominator + (y_r(i,1)^2);
    end
    Delta_eulI_x(a)=numerator/denominator;
end

%
   ------------------------------------------------------------------------%


%
   ------------------------------------------------------------------------%


%Adams-bashforth error calculation

Delta_AB_x=zeros(1,length(h));

for a=1:length(h)
    %calculation of the solution by AB
    [t_AB, x_AB, y_AB]=adams_bashforth(@(t,x,y)lotka_volterra(t,[x,y],p)
       ,tspan,IC,h(a),p);

    %adapting points

    N = length(t_r); % Points by ode45
    M = length(t_AB); % points by Eulero explicit

    % Adapting by interpolation
    t_AB_adapted = interp1(linspace(t_AB(1), t_AB(end), M), t_AB,
       linspace(t_AB(1), t_AB(end), N));
    x_AB_adapted = interp1(t_AB, x_AB, t_AB_adapted);
    y_AB_adapted = interp1(t_AB, y_AB, t_AB_adapted);

    numerator=0;
    denominator=0;
    for i=1:N
        numerator = numerator + (x_AB_adapted(i)-y_r(i,1))^2;
        denominator = denominator + (y_r(i,1)^2);

    end
    Delta_AB_x(a)=numerator/denominator;
end


%-----------------plot-----------------------------------%



xline(0.005, 'Color', 'green');

semilogx(h, Delta_eulE_x, 'r', 'LineWidth', 2);
hold on;
semilogx(h, Delta_eulI_x, 'b', 'LineWidth', 2);
hold on;
semilogx(h, Delta_AB_x, 'm', 'LineWidth', 2);
hold off;
```

```
grid on;
title("$\Delta_x$ by methods", 'FontSize', 25, 'Interpreter', 'latex');
xlabel("$h$", 'FontSize', 25, 'Interpreter', 'latex');
ylabel("$\Delta_x$", 'FontSize', 25, 'Interpreter', 'latex');
legend('Euler Explicit', 'Euler Implicit', 'Adams-Bashforth');

set(gca, 'FontSize', 18);
```

5 References

Wikipedia: Ordinary Differential Equation (https://en.wikipedia.org/wiki/Ordinary_differential_equation)