Warsaw University of Technology
Faculty of Electronics and Information Technology
Course : ENUME-2020

#11 Assignment C: Solving Ordinary Differential Equations

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

Author: Keshav Dandeva [302333]
Advisor: Paweł Mazurek

2^{nd} June 2020, Warsaw

Contents

1. Theoretical Introduction

Let us begin by first understanding the aim of this assignment. This assignment examines and analyses the solving of ordinary differential equations with the aid of MATLAB software. The implementation and systematic testing of algorithms for solving them and assessing the errors obtained.

1.1 Ordinary Differential Equation (ODE)

An **Ordinary Differential Equation (ODE)** is a differential equation containing one or more functions of one independent variable and the derivatives of those functions. Differential equations are commonly used for mathematical modeling of dynamic systems. Systems of differential equations describing real life problems are usually non-linear and analytic solutions are in most cases not possible to obtain – the only way to obtain them is using numerical methods. Numerical methods for ordinary differential equations are methods used to find numerical approximations to the solutions of ordinary differential equations (ODEs).

Initial Value Problems (IVP) :

Find a vector of functions $\mathbf{y}(t) \equiv [y_1(t).....y_M(t)]^T$ satisfying the following system of ODEs:

$$\frac{d\mathbf{y}(t)}{dt} = \mathbf{f}(t, \mathbf{y}(t)) \text{ for } t \in [0, T]$$

given the initial conditions, i.e. the vector $\mathbf{y}(0) \equiv [y_1(t).....y_M(t)]^T$

Solving IVP Using Numerical Methods :

A numerical method for solving an IV problem is defined by a recursive operator:

$$\mathbf{y}_n \equiv S(\mathbf{y}_{n-1}, \mathbf{y}_{n-2}, ...) \text{ for } n = 0, ..., N$$

generating a sequence $\mathbf{y}_n$ being an estimate of the sequence $\mathbf{y}(t_n)$, where:

$$0 = t_0 \leq t_1 \leq ... \leq t_n \leq ... \leq t_N = T$$

Such a method may be designed by replacing the derivative in:

$$\frac{d\mathbf{y}(t)}{dt} = \mathbf{f}(t, \mathbf{y}(t)) \text{ for } t \in [0, T]$$

with a formula of numerical differentiation e.g. by replacing:

$$\frac{dy(t)}{dt}|_{t=t_{n-1}} \text{ with } \frac{y(t_n) - y(t_{n-1})}{t_n - t_{n-1}} \text{ in } \frac{dy(t)}{dt}|_{t=t_{n-1}} = f(t_{n-1}, y(t_{n-1})) \text{ for } n = 0, 1...$$

the so-called forward Euler's method is obtained, viz.:

$$y_n = y_{n-1} + h_{n-1} f(t_{n-1}, y_{n-1}) \text{ for } n = 0, 1, ... \text{ and } y_0 = y(0)$$

where y_n is an estimate of $y(t_n)$ and $h_n = t_n - t_{n-1}$.

Stability of Methods for Solving IVP :

For any square matrix $\mathbf{A}$, having only single (possibly complex) eigenvalues $\lambda_1, \lambda_2, ... \in C$:

$$\mathbf{A} = \mathbf{V} \cdot \mathbf{\Lambda} \cdot \mathbf{V^{-1}}, \text{where } \mathbf{\Lambda} = diag\{\lambda_1, \lambda_2, ...\} \text{ and } \mathbf{V} \text{ is the matrix of eigenvectors}$$

This decomposition enables the transformation of a system of linear ODEs:

$$\mathbf{y}'(t) = \mathbf{A} \cdot \mathbf{y(t)} + \mathbf{B} \cdot \mathbf{u(t)}$$

into a set of independent scalar ODEs, viz.:

$$\begin{aligned} \mathbf{y}'(\mathbf{t}) &= \mathbf{V} \cdot \mathbf{\Lambda} \cdot \mathbf{V^{-1}} \cdot \mathbf{y(t)} + \mathbf{B} \cdot \mathbf{u(t)} \\ \mathbf{V^{-1}} \cdot \mathbf{y}'(\mathbf{t}) &= \mathbf{\Lambda} \cdot \mathbf{V^{-1}} \cdot \mathbf{y(t)} + \mathbf{V^{-1}} \cdot \mathbf{B} \cdot \mathbf{u(t)} \\ \mathbf{z}'(\mathbf{t}) &= \mathbf{\Lambda} \cdot \mathbf{z(t)} + \mathbf{V^{-1}} \cdot \mathbf{B} \cdot \mathbf{u(t)} \end{aligned}$$

where: $\mathbf{z(t)} \equiv \mathbf{V^{-1}} \cdot \mathbf{y(t)}$.

Thus, the results of stability analysis obtained for a scalar ODE, called "test equation":

$$y'(t) = \lambda \cdot y(t) \text{ with } \lambda \in C, \text{ for } y(0) = 1 \text{ and } t \in [0, T],$$

may be generalized on linear systems of ODEs.

1.2 Methods for solving ODEs

Numerical methods for solving first-order IVPs often fall into one of two large categories: 1. Single-step Methods and 2. Multi-step Methods. A further division can be realized by dividing methods into those that are explicit and those that are implicit. Explicit and implicit methods are approaches used in numerical analysis for obtaining numerical approximations to the solutions of time-dependent ordinary differential equations.

Explicit Method calculate the state of a system at a later time from the state of the system at the current time, whereas **Implicit Methods** find a solution by solving an equation involving both the current state of the system and the later one.

Mathematically, if $\mathbf{Y(t)}$ is the current system state and $\mathbf{Y(t + \Delta t)}$ is the state at the later time (Δt is a small time step), then, for an Explicit Method:

$$Y(t + \Delta t) = F(Y(t))$$

while for an Implicit Method one solves an equation:

$$G\Big(Y(t), Y(t + \Delta t)\Big) = 0$$

to find $\mathbf{Y(t + \Delta t)}$.

1.2.1 Single-Step Methods

In this method, we start from an initial point and then take a short step forward in time to find the next solution point. The process continues with subsequent steps to map out the solution. Single-step methods (such as Euler's method) refer to only one previous point and its derivative to determine the current value.

Methods such as Runge–Kutta take some intermediate steps (for example, a half-step) to obtain a higher order method, but then discard all previous information before taking a second step.

Order and stability of explicit single-step methods
The maximal possible order:

$$p(K) = \begin{cases} K & \text{for} \quad K = 1, 2, 3, 4 \\ K-1 & \text{for} \quad K = 5, 6, 7 \\ K-2 & \text{for} \quad K \geq 8 \end{cases}$$

Some example of single-step methods are:

1. General Runge-Kutta Method
   $y_n = y_{n-1} + h \cdot \sum_{k=1}^{K} w_k f_k$
   where:
   $f_k = f\left(t_{n-1} + c_k h, y_{n-1} + h \cdot \sum_{k=1}^{K} a_{k,K} f_K\right)$ for $k = 1, 2, ..., K$
2. Forward Euler Method $(K = 1, p = 1)$
   $y_n = y_{n-1} + h \cdot f(t_{n-1}, y_{n-1})$
3. Heun Method $(K = 2, p = 2)$
   $y_n = y_{n-1} + \frac{1}{2} h[f(t_{n-1}, y_{n-1}) + f(t_{n-1} + h, y_{n-1} + hf(t_{n-1}, y_{n-1}))]$

1.2.2 Multi-Step Methods

Multistep methods attempt to gain efficiency by keeping and using the information from previous steps rather than discarding it. Consequently, multistep methods refer to several previous points and derivative values. In the case of linear multistep methods, a linear combination of the previous points and derivative values is used.

A K -step linear method is defined by the following formula:
$y_n = \sum_{k=1}^{K_\alpha} \alpha_k y_{n-k} + h \sum_{k=k}^{K_\beta} \beta_k \cdot f(t_{n-k}, y_{n-k})$ with $y_0 = y(0), ..., y_{K-1} = y((K-1)h)$
where $k = 0$ or $k = 1$, $K_\alpha \leq 1, K_\beta \leq k, \max\{K_\alpha, K_\beta\} = K$

Some example of multi-step methods are:

1. Shichman Method or Implicit Gear Method $(K = p = 2)$
   $y_n = \frac{4}{3} y_{n-1} - \frac{1}{3} y_{n-2} + \frac{2}{3} h y'_n$
2. Explicit Adams Method (Adams-Bashforth methods)
   $y_n = y_{n-1} + h \sum_{k=1}^{K} \beta_k f(t_{n-k}, y_{n-k})$
3. Implicit Adams methods (Adams-Moulton methods)
   $y_n = y_{n-1} + h \sum_{k=0}^{K} \beta_k f(t_{n-k}, y_{n-k})$

2. Task 1

2.1 Question

We are provided with the system differential-algebraic equations of the form:

$$\left.\begin{array}{c}\frac{d\mathbf{v}(t)}{dt} = \mathbf{A}\cdot\mathbf{v}(\mathbf{t}) + \mathbf{b}\cdot x(t) \\ y(t) = \mathbf{c^T}\cdot\mathbf{v}(\mathbf{t})\end{array}\right\} \text{where: } \mathbf{A} = \begin{bmatrix} 0 & 1 & -1 \\ 0 & 1 & -2 \\ 1200 & -282 & -62 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} \text{ and } \mathbf{c} = \begin{bmatrix} 820 \\ -296.5 \\ -8 \end{bmatrix}$$

The task is to program the functions implementing the equations in MATLAB.

2.2 Implementation

First, we define all the variables used in the program. Starting with the definition of matrices **A, b** and **c**. Then, the transpose of the matrix c is done with the help of built-in MATLAB function *transpose(c)* and stored in a new matrix named ctrans.

I created a *for* loop for setting the value of the integration step h so that it can be changed dynamically and observations can be made for its different values. The variable t is also defined from 0 to 5 with the step size of h.

At last, the initial values of vector $\mathbf{v(t)}$ are set to 0 by creating a vector v0 with initial values 0.

The MATLAB Code for the Task is attached in Appendix for reference.

3. Task 2

3.1 Question

The task is to solve the system of equations provided in Task:1 with zero initial conditions for:

$$x(t) = \begin{cases} 0 & \text{for} \quad t \leq 0 \\ 1 & \text{for} \quad t > 0 \end{cases} \text{ for } t \in (0, 5)$$

and for:

$$x(t) = \begin{cases} 0 & \text{for} \quad t \leq 0 \\ \exp(-t) & \text{for} \quad t > 0 \end{cases} \text{ for } t \in (0, 5)$$

using **ODE45** procedure of MATLAB and setting the parameters 'AbsTol' and 'RelTol' to the values guaranteeing the maximum accuracy of the solution.

3.2 Implementation

In this task, we have to solve the given differential equation with the help of the MATLAB built-in function *ode45* for two conditions. We start by defining the conditions $x(t)$ for $t \in (0, 5)$. I have defined the two conditions as **Case-1:** $x(t)$ **and Case-2:** $x2(t)$ and solved the differential equation for each case separately.

To solve the equation using *ode45* method, first I created the function dvdt=@(t,v) A*v+b*x(t). Then passed this function and t and v0 as arguments to the function *ode45* which provided the result that I stored in a new vector **vo**.

The parameters **AbsTol** and **RelTol** are set using the *odeset* function. For the relative error tolerance, I set the value 10^{-6} because if the tolerance is too low, the large number of steps will increase the accumulated rounding errors, while for a too high tolerance the local errors dominate the accuracy of the result and for the absolute error tolerance, I set the value to 10^{-9} so that when solution value is around 0, the program would quit optimizing.

And finally, I calculate the function $y(t)$ by transposing the vector vo obtained from the *ode45* method and multiplying it with the transpose of **c** as given in the equation.

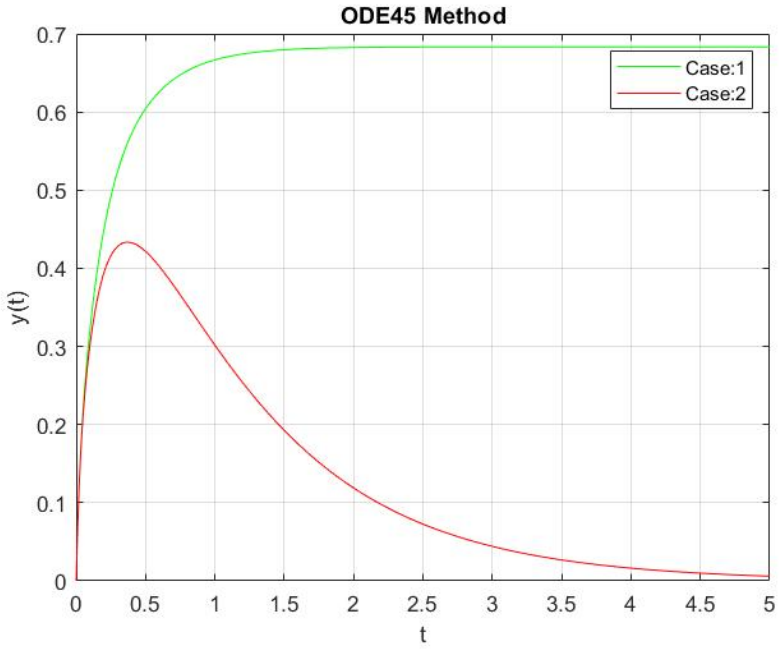
Hereinafter, the solution obtained from this method ($y(t)$) will be considered as the exact solution (most accurate) of the ODE.

The MATLAB Code for the Task is attached in Appendix for reference.

3.3 Graphical Representation

The graphs of the solutions obtained from solving the differential equation using *ode45* method are presented below. The graph is plotted using the *plot* function of MATLAB.

Figure 3.1: y(t) vs t



4. Task 3

4.1 Question

In this task, we have to solve the given differential equation for both functions $x(t)$ using two different methods:

- the explicit Adams-Bashforth method
- the implicit Gear method

for various values of the integration step $h \in [h_{min}, h_{max}]$.

We have to illustrate and explain the phenomenon of numerical instability for excessive values of h.

And at last, we have to compute the estimate $\hat{y}(t)$ of $y(t)$ and determine the following indicators of its uncertainty and plot their graphs:

$\delta_2(h) = \frac{\|\hat{y}(t;h) - \dot{y}(t)\|_2}{\|\dot{y}(t)\|_2}$ and $\delta_\infty(h) = \frac{\|\hat{y}(t;h) - \dot{y}(t)\|_\infty}{\|\dot{y}(t)\|_\infty}$

4.2 Implementation

4.2.1 Explicit Adams-Bashforth Method

The formula for solving differential equation with this method is:

$$v_n = v_{n-1} + h\sum_{k=1}^{K} \beta_k f(t_{n-k}, v_{n-k}) \text{ with } K = 2$$

Now, to solve the ODE using this method, first we need to find the initial values of $v(t)$. As the given $K = 2$, we need to find the values when $K = 1$.

First, we initialise the first value as 0. I create a new vector to store the result of this method named *vabe* and copy in it the value of the initialised vector $v0$.

Then we solve for $K = 1$, and store the value as the second value of the vector *vabe*. The formula used for calculating is same as given and the values for β_k are taken from the table provided in the assignment.

Now, we have the first 2 values required for the iterative algorithm to work. I used *for* loop to iterate the formula till the length of t and stored the results in the vector *vabe*.

Now, to find the solution of the equation, we use the vector *vabe* and multiply it with the transpose of vector **c** to get $y(t)$.

$$y(t) = c^T \cdot vabe(t)$$

Analysing Stability

Now, we will check the stability of the this method by taking different values of the integration step $h \in [h_{min}, h_{max}]$.

Figure 4.1: For h=0.01

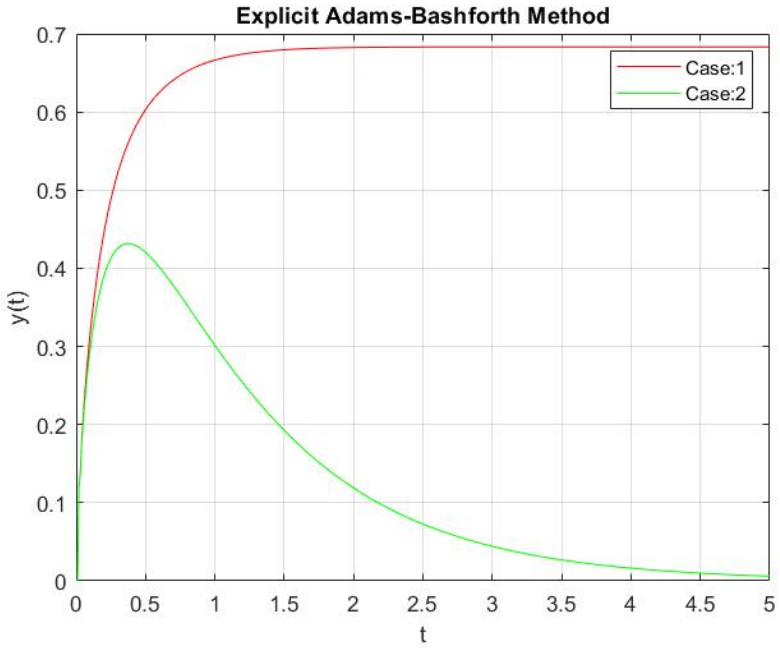


Figure 4.2: For h=0.005

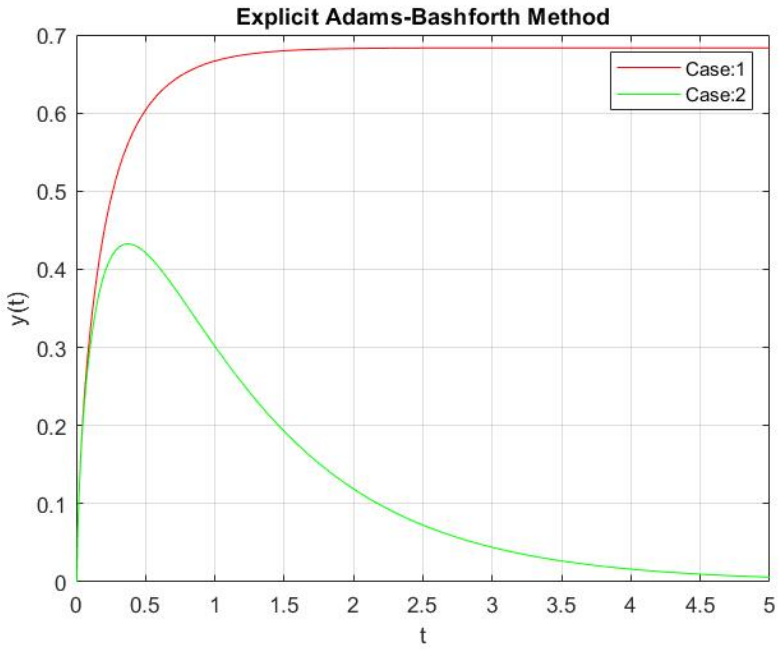


Figure 4.3: For h=0.015

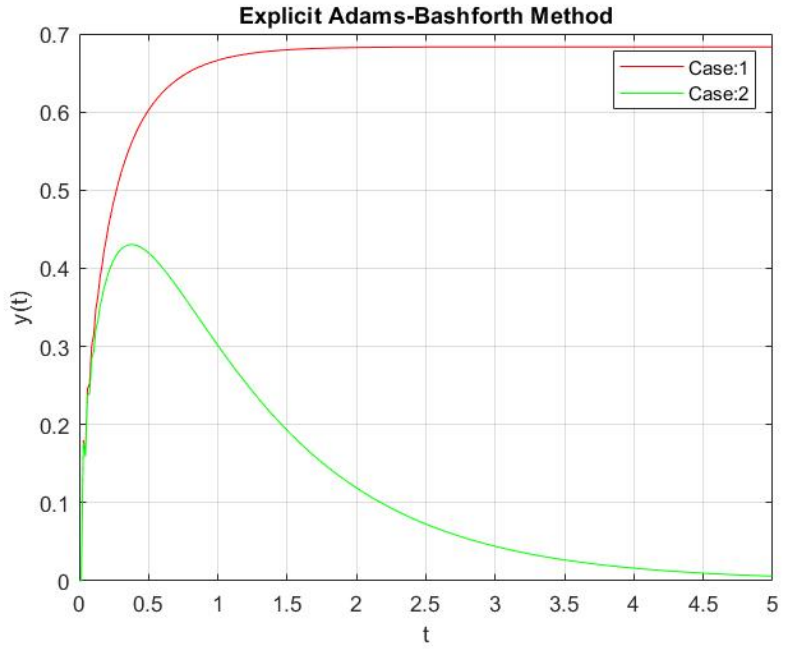
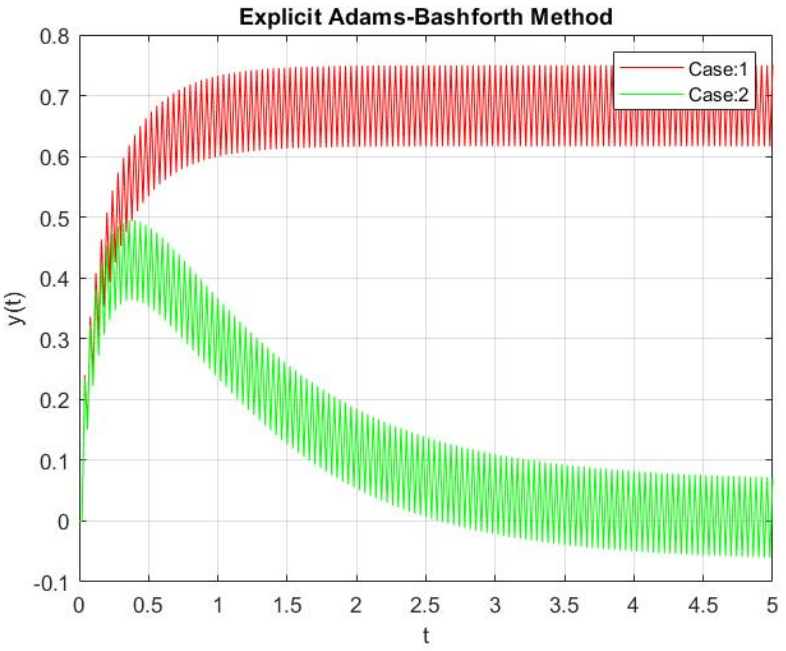


Figure 4.4: For h=0.02



We can see that step-size which is equal to 0.01 is the optimal step-size, because decreasing it to 0.005 does not influence the solution significantly but increasing it to 0.015 does and at 0.02 becomes unstable.

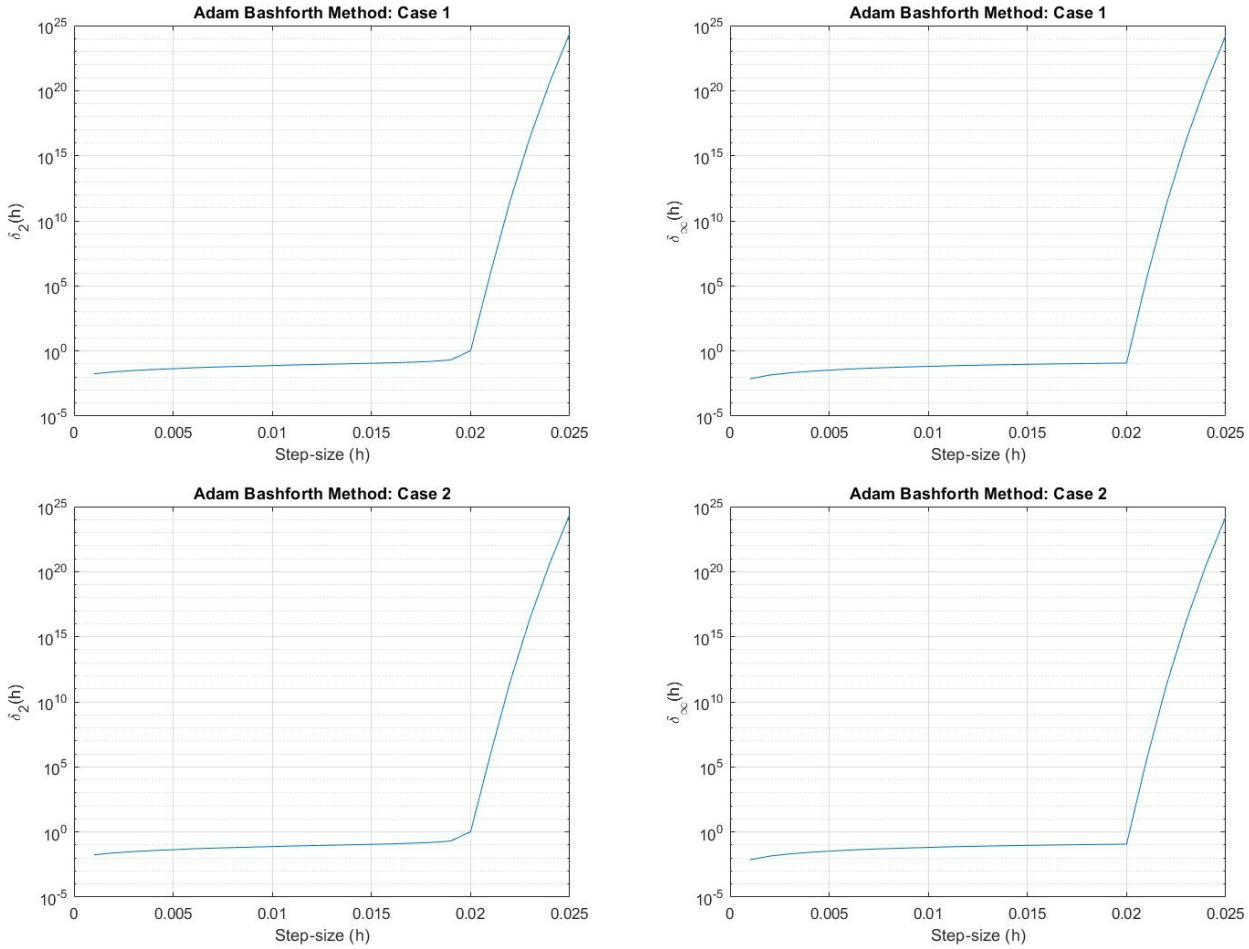
Also, we can see that the lower step-size the better the results, its because the difference between every two points is smaller but on the other hand the smaller the step size the longer it takes to find the solution because the number of arithmetic calculations increases, for example for $h = 0.001$ the function would have to go through 5000 iterations while using $h = 0.01$ we had 500 iterations and for $h = 0.02$ we get 250 iterations which seems to be small because we are not getting good results.

Uncertainty Indicators

The uncertainty indicators for this method are calculated using the formula:

$$\delta_2(h) = \frac{\|\hat{y}(t;h) - \dot{y}(t)\|_2}{\|\dot{y}(t)\|_2} \text{ and } \delta_\infty(h) = \frac{\|\hat{y}(t;h) - \dot{y}(t)\|_\infty}{\|\dot{y}(t)\|_\infty}$$

where $\dot{y}(t)$ is the exact solution obtained from the *ODE45* Method and $\hat{y}(t;h)$ is the solution obtained from this method (Explicit Adams-Bashforth Method)



We can see from the above graphs that in the **Explicit Adam Bashforth Method**, with increase in the size

of integration step h there is an increase in the uncertainty of the function.

The magnitude of uncertainty is relatively very less till the value of the integration step h reaches 0.02, after that the function starts becoming unstable and the magnitude of uncertainty increases exponentially with an increase in the value of the integration step h.

4.2.2 Implicit Gear Method

The formula for solving differential equation with this method is:

$$\mathbf{v}_n = \sum_{k=1}^{K} \alpha_k \mathbf{v}_{n-k} + h \cdot \beta_0 f(t_n, \mathbf{v}_n) \text{ with } K = 5$$

The first step in solving differential equations using implicit methods is to take the value of the vector $\mathbf{v}_n$ from RHS to LHS. Following are the calculations done to reach the formula that will be used for calculating the vector $\mathbf{v}_n$:

$$=> \mathbf{v}_n = \sum_{k=1}^{K} \alpha_k \mathbf{v}_{n-k} + h \cdot \beta_0 f(t_n, \mathbf{v}_n)$$

$$=> \mathbf{v}_n = \sum_{k=1}^{K} \alpha_k \mathbf{v}_{n-k} + h \cdot \beta_0 (\mathbf{A} \cdot \mathbf{v}_n + \mathbf{b} \cdot x(t_n))$$

$$=> \mathbf{v}_n = \sum_{k=1}^{K} \alpha_k \mathbf{v}_{n-k} + h \cdot \beta_0 \cdot \mathbf{A} \cdot \mathbf{v}_n + h \cdot \beta_0 \cdot \mathbf{b} \cdot x(t_n)$$

$$=> \mathbf{v}_n - h \cdot \beta_0 \cdot \mathbf{A} \cdot \mathbf{v}_n = \sum_{k=1}^{K} \alpha_k \mathbf{v}_{n-k} + h \cdot \beta_0 \cdot \mathbf{b} \cdot x(t_n)$$

$$=> \mathbf{v}_n (\mathbf{I} - h \cdot \beta_0 \cdot \mathbf{A}) = \sum_{k=1}^{K} \alpha_k \mathbf{v}_{n-k} + h \cdot \beta_0 \cdot \mathbf{b} \cdot x(t_n)$$

$$=> \mathbf{v}_n = (\mathbf{I} - h \cdot \beta_0 \cdot \mathbf{A})^{-1} \cdot \left[\sum_{k=1}^{K} \alpha_k \mathbf{v}_{n-k} + h \cdot \beta_0 \cdot \mathbf{b} \cdot x(t_n) \right]$$

Now, to solve the ODE using this formula obtained above, first we need to find the initial values of $v(t)$. As the given $K = 5$, we need to find the values when $K = 1, 2, 3, 4$.

First, we initialise the first value as 0. We create a new vector to store the result of this method named *vig* and copy in it the value of the initialised vector $v0$.

Then we solve for $K = 1, 2, 3, 4$, and store the value as the respective values of the vector *vig*. The formula used for calculating is same as given and the values for β_0 and α_k are taken from the table provided in the assignment.

Now, we have the first 5 values required for the iterative algorithm to work. I used *for* loop to iterate the formula till the length of t and stored the results in the vector *vig*.

Now, to find the solution of the equation, we use the vector *vig* and multiply it with the transpose of vector $\mathbf{c}$ to get $y(t)$.

$$y(t) = c^T \cdot vig(t)$$

Analysing Stability

Now, we will check the stability of the this method by taking different values of the integration step $h \in [h_{min}, h_{max}]$.

Figure 4.5: For h=0.01

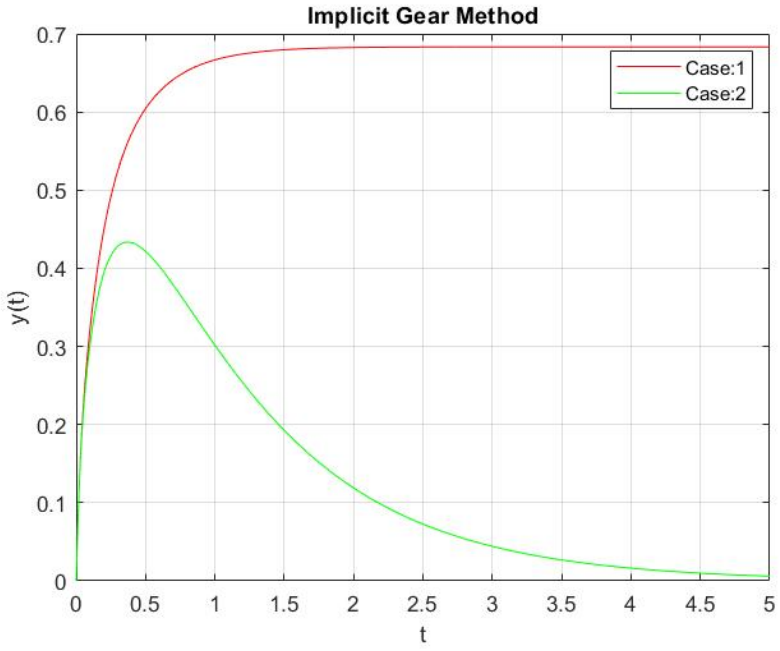


Figure 4.6: For h=0.04

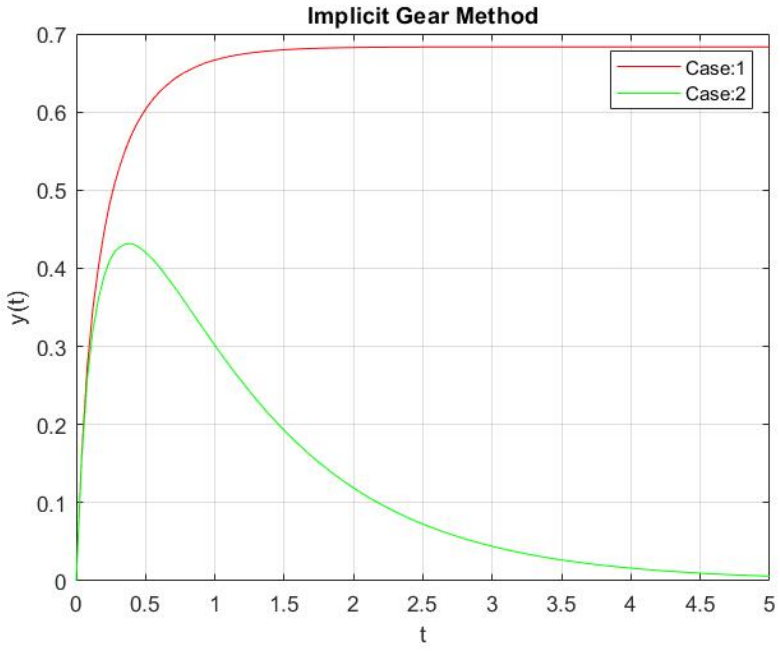


Figure 4.7: For h=0.05

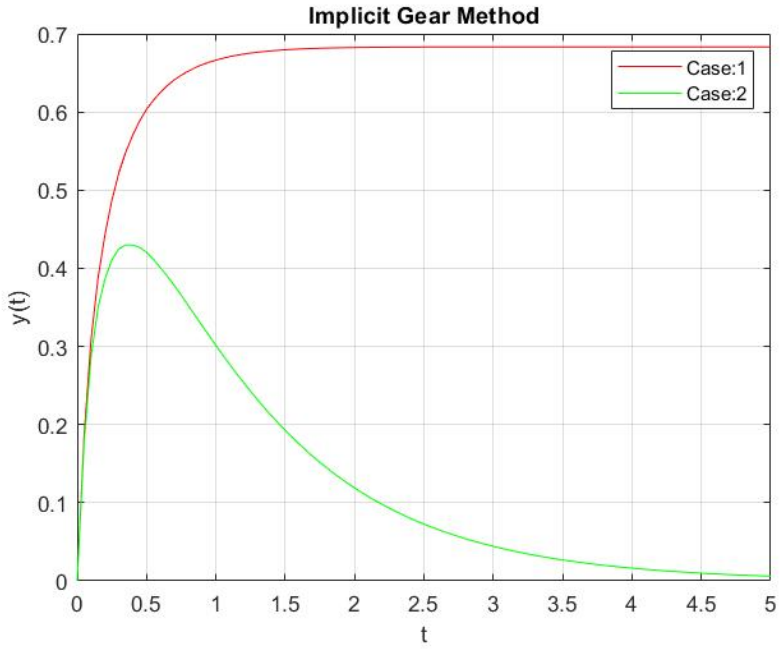
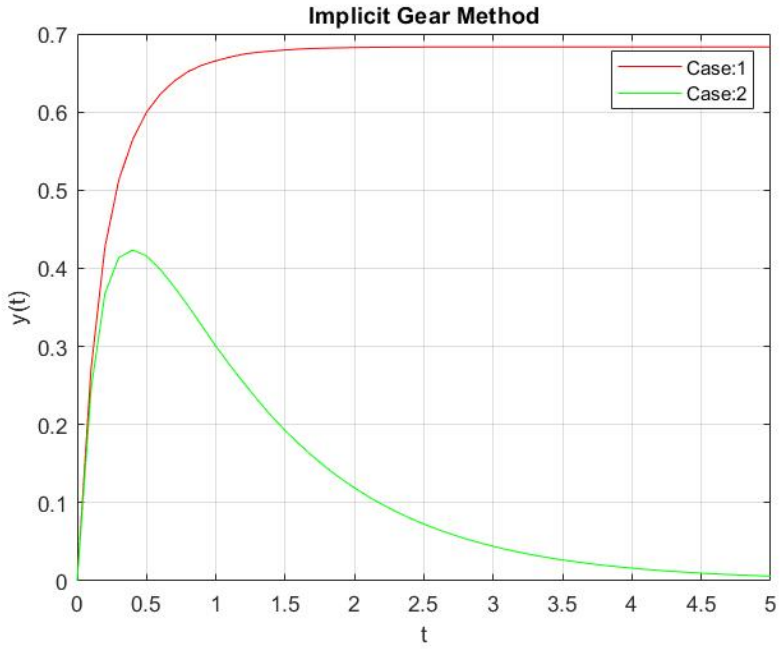


Figure 4.8: For h=0.1



We can see that step-size which is equal to 0.04 is the optimal step-size, because decreasing it to 0.01 does not influence the solution significantly but increasing it to 0.05 does and at 0.1 the fluctuations in the curve of the function are quite clearly visible.

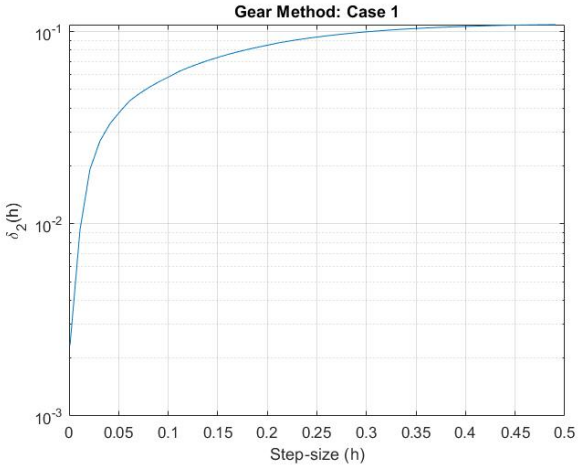
Also, we can see that the lower step-size the better the results, its because the difference between every two points is smaller but on the other hand the smaller the step size the longer it takes to find the solution because the number of arithmetic calculations increases, for example for $h = 0.01$ the function would have to go through 500 iterations while using $h = 0.04$ we had 126 iterations and for $h = 0.1$ we get 50 iterations which seems to be small because we are not getting good results.

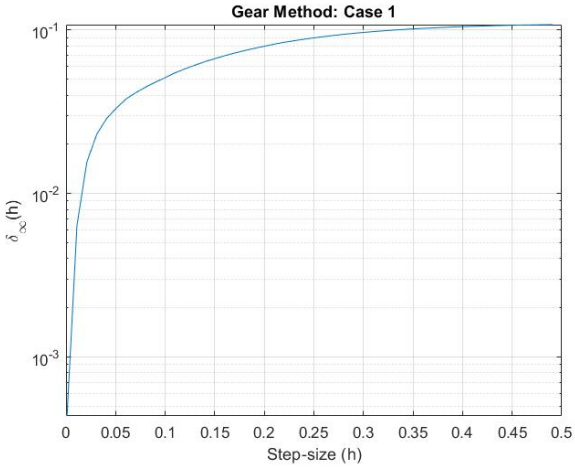
Uncertainty Indicators

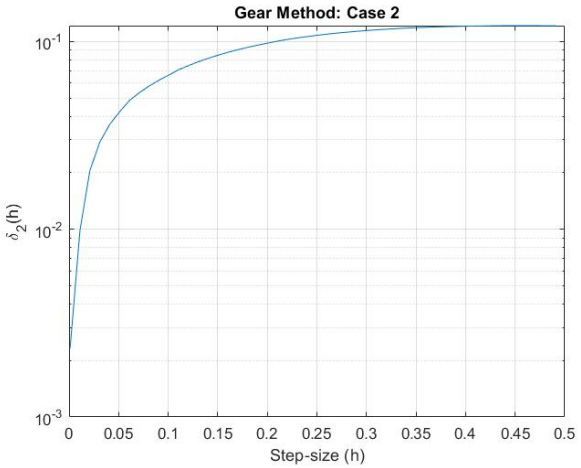
The uncertainty indicators for this method are calculated using the formula:

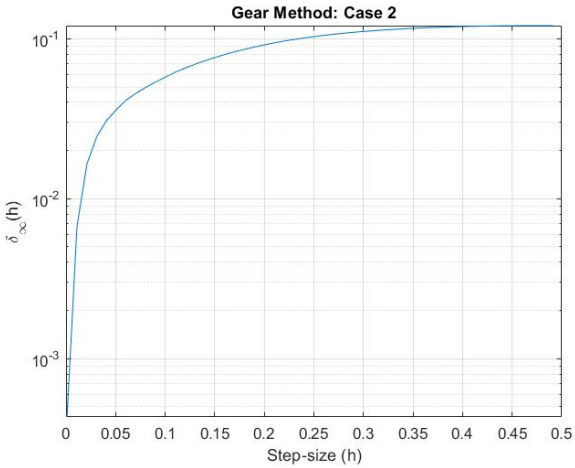
$$\delta_2(h) = \frac{\|\hat{y}(t;h) - \dot{y}(t)\|_2}{\|\dot{y}(t)\|_2} \text{ and } \delta_\infty(h) = \frac{\|\hat{y}(t;h) - \dot{y}(t)\|_\infty}{\|\dot{y}(t)\|_\infty}$$

where $\dot{y}(t)$ is the exact solution obtained from the *ODE45* Method and $\hat{y}(t;h)$ is the solution obtained from this method (Explicit Adams-Bashforth Method) for different values of h









We can see from the above graphs that in the **Implicit Gear Method**, with increase in the size of integration step h there is an increase in the uncertainty of the function up till a value after which the magnitude of uncertainty remains almost constant.

The magnitude of uncertainty is relatively very less till the value of the integration step h reaches 0.1, after that the function starts showing visible perturbation but still has the magnitude of uncertainty relatively less.

5. Conclusion

After performing all the above tasks, we can clearly observe that **ODE45 Method** is the best method to obtain solution of Ordinary Differential Equations as it provided the most accurate curve of the function and within less time. That is why, we used it as the exact solution for later tasks.

We observed from the tasks that, **Implicit Methods** require an extra computation (solving the Gear Method in Task 3), and they can be much harder to implement. Implicit methods are used because many problems arising in practice are stiff, for which the use of an explicit method requires impractically small time steps Δt to keep the error in the result bounded. For such problems, to achieve given accuracy, it takes much less computational time to use an implicit method with larger time steps, even taking into account that one needs to solve an equation of the explicit at each time step. We can observe this from the **Uncertainty Graphs** that the **Implicit Methods** had relatively less magnitude of error than **Explicit Methods** even for higher integration step values.

We also observed that **Explicit Methods** in general have less stability region and for excess values of h, they become numerically unstable. Also, the time taken by explicit methods to reach the correct solution is greater than the implicit method as we can see from the number of iterations required by each method to obtain the solution.

It is also quite evident from the values of uncertainty that **Implicit Gear Method** provides more accuracy to the solution than **Explicit Adams-Bashforth Method**.

6. Appendix

```
clear
close all
clc

A=[0 1 -1; 0 1 -2; 1200 -282 -62];
b=[0 ;0 ;-1];
c=[820 ;-296.5 ;-8];
ctrans= transpose(c);
x=@(t)1*(t>0); % case 1
x2=@(t)exp(-t)*(t>0); %case 2

v0=[0; 0; 0];

counter = 0;

for h=0.001:0.01:0.5   % For checking various values of integration step

    counter= counter +1;
    stepsize(counter)=h;

    t=0:h:5;
    %% ode45: Case 1
    dvdt=@(t,v) A*v+b*x(t);
    options = odeset('RelTol',1e-6,'AbsTol',1e-9);
    [to,vo] = ode45(dvdt,t,v0,options);

    votrans = transpose(vo);
    y = ctrans*votrans;


    %% ode45: Case 2

    dvdt2=@(t,v) A*v+b*x2(t);
    [to2,vo2] = ode45(dvdt2,t,v0,options);

    vo2trans = transpose(vo2);
    y2 = ctrans*vo2trans;

    %     figure(1)       % This is the graph of ODE Solution
    %     plot(t,y,'r-');
    %     grid on
    %     hold on
    %     plot(t,y2,'g-');
    %     title('ODE45 Method');
    %     xlabel('t');
    %     ylabel('y(t)');
    %     legend('Case:1','Case:2');

    %% Explicit Adams-Bashforth method: Case 1
```

```
50
51      vabe(:,1)= v0; % K = 0
52      vabe(:,2)= vabe(:,1)+h*(A*vabe(:,1)+b*x(t(1))); % K = 1
53
54      for i=3:length(t)
55          vabe(:,i)=vabe(:,i-1)+h*((3/2)*(A*vabe(:,i-1)+b*x(t(i-1)))+(-1/2)*(A*vabe(:,i-2)+b*x(t(i-2))));
56      end
57      y3 = ctrans*vabe;
58
59      clear vabe
60
61      abe1_Delta2(counter) = norm(y3-y,2);
62      abe1_DeltaInf(counter)= norm(y3-y,Inf);
63
64      %Task-3 Graphs
65  %     figure(4)
66  %     semilogy(stepsize,abe1_Delta2);
67  %     grid on
68  %     title('Adam Bashforth Method: Case 1');
69  %     xlabel('Step-size (h)');
70  %     ylabel('\Δ_2(h)');
71  %
72  %     figure(5)
73  %     semilogy(stepsize,abe1_DeltaInf);
74  %     grid on
75  %     title('Adam Bashforth Method: Case 1');
76  %     xlabel('Step-size (h)');
77  %     ylabel('\Δ_{\infty}(h)');
78
79
80      %% Explicit Adams-Bashforth method: Case 2
81
82      vabe2(:,1)= v0; % K = 0
83      vabe2(:,2)= vabe2(:,1)+h*(A*vabe2(:,1)+b*x2(t(1))); % K = 1
84
85      for i=3:length(t)
86          vabe2(:,i)=vabe2(:,i-1)+h*((3/2)*(A*vabe2(:,i-1)+b*x2(t(i-1)))+(-1/2)*(A*vabe2(:,i-2)+b*x2(t(i-2))));
87      end
88
89      y4 = ctrans*vabe2;
90      clear vabe2
91
92      abe2_Delta2(counter) = norm(y4-y2,2);
93      abe2_DeltaInf(counter) = norm(y4-y2,Inf);
94
95      %     figure(2)                 % This is the graph of Adam Bashforth Solution
96      %     plot(t,y3,'-r');
97      %     hold on
98      %     plot(t,y4,'-g');
99      %     grid on
100     %     title('Explicit Adams-Bashforth Method');
101     %     xlabel('t');
102     %     ylabel('y(t)');
103     %     legend('Case:1','Case:2');
104
105     %Task 3 Graph
106 %     figure(6)
107 %     semilogy(stepsize,abe2_Delta2);
108 %     grid on
109 %     title('Adam Bashforth Method: Case 2');
110 %     xlabel('Step-size (h)');
111 %     ylabel('\Δ_2(h)');
```

```
%
%     figure(7)
%     semilogy(stepsize,abe2_DeltaInf);
%     grid on
%     title('Adam Bashforth Method: Case 2');
%     xlabel('Step-size (h)');
%     ylabel('\Δ_{\infty}(h)');
    %% Implicit Gear Method: Case 1

    vig(:,1)= v0; %K=0
    vig(:,2)= (eye(3)-h*1*A)\(h*1*b*x(t(2)) + (1*vig(:,1))); %K=1
    vig(:,3)= (eye(3)-h*(2/3)*A)\(h*(2/3)*b*x(t(3)) + ((4/3)*vig(:,2)) + ((-1/3)*vig(:,1))); ...
        %K=2
    vig(:,4)= (eye(3)-h*(6/11)*A)\(h*(6/11)*b*x(t(4)) + ((18/11)*vig(:,3)) + ...
        ((-9/11)*vig(:,2)) + ((2/11)*vig(:,1))); %K=3
    vig(:,5)= (eye(3)-h*(12/25)*A)\(h*(12/25)*b*x(t(5)) + ((48/25)*vig(:,4)) + ...
        ((-36/25)*vig(:,3)) + ((16/25)*vig(:,2)) + ((-3/25)*vig(:,1))); %K=4

    beta0=60/137;
    alpha1=300/137;
    alpha2=-300/137;
    alpha3=200/137;
    alpha4=-75/137;
    alpha5=12/137;

    for i=6:length(t)
        vig(:,i)=(eye(3)-h*beta0*A)\(h*beta0*b*x(t(i)) + (alpha1*vig(:,i-1)) + ...
            (alpha2*vig(:,i-2)) + (alpha3*vig(:,i-3)) + (alpha4*vig(:,i-4)) + ...
            (alpha5*vig(:,i-5)));
    end

    y5 = ctrans*vig;
    clear vig
    ig1_Delta2(counter) = norm(y5-y,2);
    ig1_DeltaInf(counter) = norm(y5-y,Inf);

%     Task-3 Graphs
%     figure(8)
%     semilogy(stepsize,ig1_Delta2);
%     grid on
%     title('Gear Method: Case 1');
%     xlabel('Step-size (h)');
%     ylabel('\Δ_2(h)');
%
%     figure(9)
%     semilogy(stepsize,ig1_DeltaInf);
%     grid on
%     title('Gear Method: Case 1');
%     xlabel('Step-size (h)');
%     ylabel('\Δ_{\infty}(h)');
%

    %% Implicit Gear Method: Case 2

    vig2(:,1)= v0; %K=0
    vig2(:,2)= (eye(3)-h*1*A)\(h*1*b*x2(t(2)) + (1*vig2(:,1))); %K=1
    vig2(:,3)= (eye(3)-h*(2/3)*A)\(h*(2/3)*b*x2(t(3)) + ((4/3)*vig2(:,2)) + ...
        ((-1/3)*vig2(:,1))); %K=2
    vig2(:,4)= (eye(3)-h*(6/11)*A)\(h*(6/11)*b*x2(t(4)) + ((18/11)*vig2(:,3)) + ...
        ((-9/11)*vig2(:,2)) + ((2/11)*vig2(:,1))); %K=3
    vig2(:,5)= (eye(3)-h*(12/25)*A)\(h*(12/25)*b*x2(t(5)) + ((48/25)*vig2(:,4)) + ...
        ((-36/25)*vig2(:,3)) + ((16/25)*vig2(:,2)) + ((-3/25)*vig2(:,1))); %K=4
```

```
166
167     for i=6:length(t)
168         vig2(:,i)=(eye(3)-h*beta0*A)\(h*beta0*b*x2(t(i)) + (alpha1*vig2(:,i-1)) + ...
                (alpha2*vig2(:,i-2)) + (alpha3*vig2(:,i-3)) + (alpha4*vig2(:,i-4)) + ...
                (alpha5*vig2(:,i-5)));
169     end
170
171     y6 = ctrans*vig2;
172     clear vig2
173
174     ig2_Delta2(counter) = norm(y6-y2,2);
175     ig2_DeltaInf(counter) = norm(y6-y2,Inf);
176
177     %     figure(3)                  % This is the graph of Implicit Gear Method Solution
178     %     plot(t,y5,'r-');
179     %     hold on
180     %     plot(t,y6,'g-');
181     %     grid on
182     %     title('Implicit Gear Method');
183     %     xlabel('t');
184     %     ylabel('y(t)');
185     %     legend('Case:1','Case:2');
186
187     %     Task-3 Graphs
188 %     figure(10)
189 %     semilogy(stepsize,ig2_Delta2);
190 %     grid on
191 %     title('Gear Method: Case 2');
192 %     xlabel('Step-size (h)');
193 %     ylabel('\Δ_2(h)');
194 %
195 %     figure(11)
196 %     semilogy(stepsize,ig2_DeltaInf);
197 %     grid on
198 %     title('Gear Method: Case 2');
199 %     xlabel('Step-size (h)');
200 %     ylabel('\Δ_{\infty}(h)');
201
202
203 end
```

7. References

- ENUME Lecture Notes 2020 - Prof Roman Morawski
- ENUME 2020 Assignment C: Introduction Video - Prof Pawel Mazurek