Text

Description automatically generated

Solving the linear ODE of this form:



Linear ODE has the form: 

=> 

Let and . Substituting into the equations, we have:

=> 

=> . Set to 0:



=> 

Substituting back to the equation:



=> . We know both u and v. We have y = uv  
=> 

Finally, from the condition:



The formula of the analytic solution is thus:

 (answer)

Text

Description automatically generated



=>   
We see that 

=> . In other words, we have: (proven)

Text

Description automatically generated

Rearrange the formula as follows:



LHS: 

RHS: 

Both sides equal -h => (proven)

Text

Description automatically generated

The general formula of the Euler’s Method:

Text, letter

Description automatically generated

Text, letter

Description automatically generated

Source: [*www.freecodecamp.org/news/eulers-method-explained-with-examples*](http://www.freecodecamp.org/news/eulers-method-explained-with-examples)

Firstly:



In this exercise, we have:

  
=> 

The initial value is  


=>  =>

Next steps are:

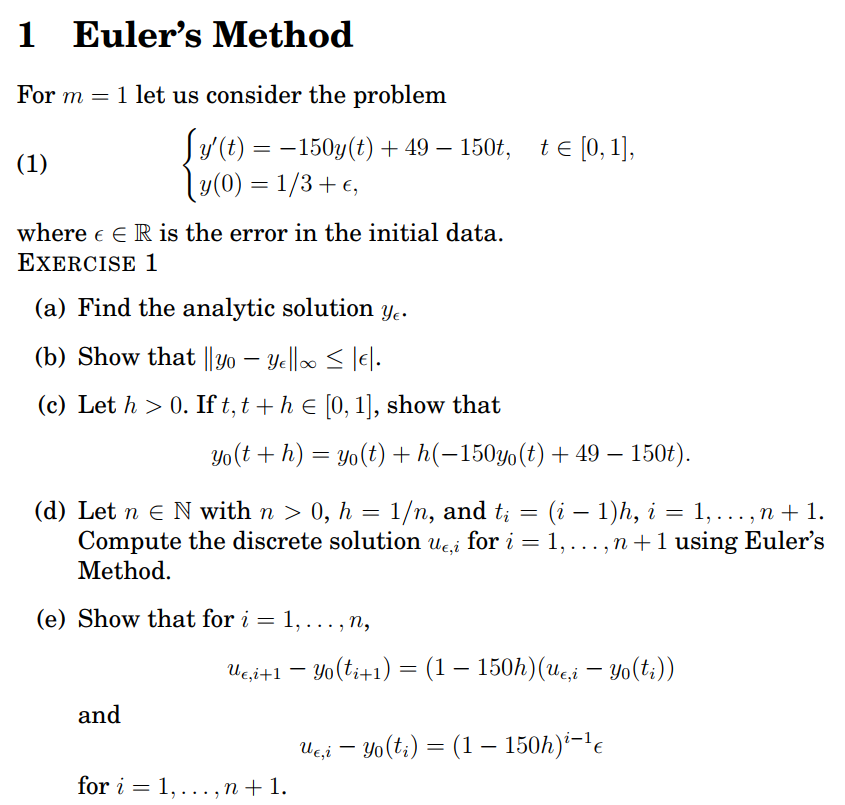




and so on

The final general discrete solution by the Euler’s method is:

(answer)



**1) Prove that **

From part (d), we can substitute into the equation:



We can also see that =>

Thus, we can again substitute and :



=> 

=> 

=>   
=>  =>   
We have: 

=> 

We have LHS = RHS => The equationis proven

**2) Prove that** 

We have already proven from part 1 that 

=>  (1), decreasing by one step backward

Now, let’s decrease by one step further

. Replace this into the above equation (1), we have:



=> 

=>   
This will recurse back to (i-1) number of steps. At the base case:

and 

=> (proven)

Text

Description automatically generated

We need to write t = 1 in terms of i and h



=> 

From part (e), we have: . The error can be calculated as follows:

=> (answer)

Text

Description automatically generated

From part (e), we can derive the formula as:

. At i = 1, we have , satisfying the conditions. Now we only need to check the the other end where i = n + 1

=> 

First we can find n where 

Analysis: if , violating the conditions

, satisfying the condition

=> The condition on n to obtain is (answer)

Graphical user interface, text, application

Description automatically generated

The forward Euler method is given by



On the other hand, the backward Euler method is given by



=> 

=> and is approximated by 

=>   
The general formula of the backward Euler method is therefore:

(answer)

Graphical user interface, text, application

Description automatically generated

We can prove similarly in Exercise 1 - (e) that:

(1)

We move backward one step:

(2). Replacing this into (1)

=> 

We can recurse this pattern up to the i = n. At the base case:

 =>  (proven)

Graphical user interface, text, application

Description automatically generated

Plugging in from (b), we have:

=>

This is the reverse case compared to Exercise 1 part (g). We only need to choose the range of n that is a difference between [75, infinity]  
=> Condition on n is and  (answer)

Text, letter

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Source: [*https://en.wikipedia.org/wiki/Heun%27s\_method*](https://en.wikipedia.org/wiki/Heun%27s_method)

The Matlab code for heun.m is

function [t, u] = heun(func, range,initialvalue, partitions)

t = linspace(range(1),range(2),partitions + 1);

h = (range(2) - range(1))/partitions;

u = zeros([1, partitions + 1]);

u(1) = initialvalue;

for i=2:partitions+1

temporary = u(i-1) + h\*func(t(i-1),u(i-1));

u(i) = u(i-1) + (h/2)\*(func(t(i-1),u(i-1)) + func(t(i),temporary));

end

The Matlab code for heuntest.m is

clc;

f1 = @(t,y) (-150)\*y + 49 - 150\*t;

[t,u] = heun(f1, [0,1], 1/3, 4);

disp("t is ")

disp(t)

disp("u is" )

disp(u)

Testing the function:

A picture containing letter

Description automatically generated

Text, letter

Description automatically generated

The Matlab code for heunplot.m is

n = 40;

epsilon=0.01;

f1 = @(t,y) (-150)\*y + 49 - 150\*t;

figure(1)

[t,u]=heun(f1, [0, 1], 1/3 + epsilon, n);

y = epsilon \* exp(-150.\*t) - t + 1/3;

plot(t, y,'r')

hold on

plot(t, u,'b')

legend('Exact values y','Approximated values u')

title("n = 40")

xlabel("time step")

error40 = max(abs(u - y));

disp("Maximum error when n = 40: " + error40)

figure(2)

n = 73;

[t,u]=heun(f1, [0, 1], 1/3 + epsilon, n);

y = epsilon \* exp(-150.\*t) - t + 1/3;

plot(t, y,'r')

hold on

plot(t, u,'b')

legend('Exact values y','Approximated values u')

title("n = 73")

xlabel("time step")

error73 = max(abs(u - y));

disp("Maximum error when n = 73: " + error73)

figure(3)

n = 75;

[t,u]=heun(f1, [0, 1], 1/3 + epsilon, n);

y = epsilon \* exp(-150.\*t) - t + 1/3;

plot(t, y,'r')

hold on

plot(t, u,'b')

legend('Exact values y','Approximated values u')

title("n = 75")

xlabel("time step")

error75 = max(abs(u - y));

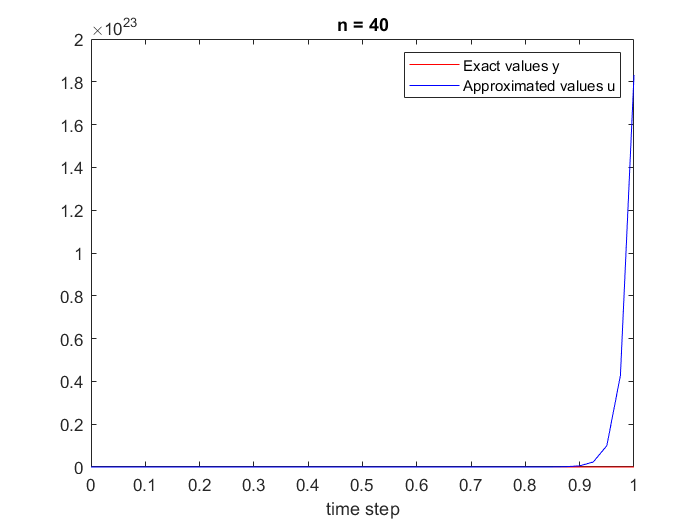
disp("Maximum error when n = 75: " + error75)

The Maximum errors for each case is:

Text

Description automatically generated with medium confidence

Plotting the exact and approximated values for each case:

  
Chart, line chart

Description automatically generated  
Chart, line chart

Description automatically generated

Text, letter

Description automatically generated

The Matlab code for heunerror.m is

arrn = 5000:10:5100;

epsilon=0.01;

f1 = @(t,y) (-150)\*y + 49 - 150\*t;

for i=1:length(arrn)

n=arrn(i);

[t,u]=heun(f1, [0,1], 1/3 + epsilon, n);

y = epsilon \* exp(-150.\*t) - t + 1/3;

arrerror(i)=norm(u-y,inf);

end

plot(log(arrn),log(arrerror),'.--')

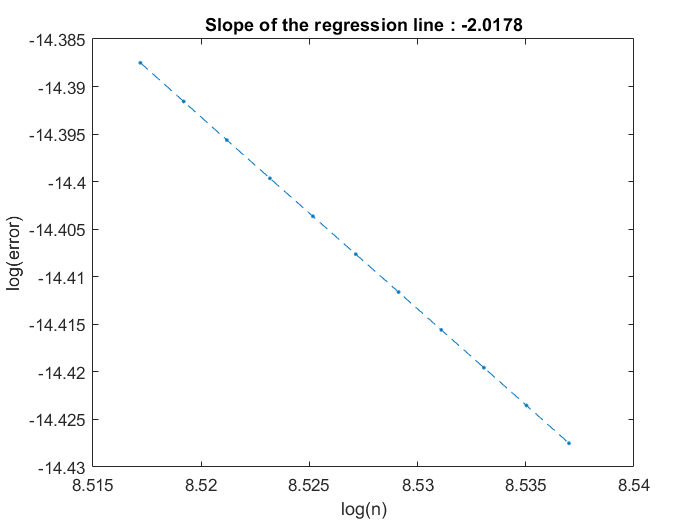
a=polyfit (log(arrn), log(arrerror) ,1);

title (['Slope of the regression line : ', num2str(a(1))])

xlabel('log(n)')

ylabel('log(error)')

Plotting the slope of the regression line:



The value of the errors are in the scale of 10e-7. The order of the convergence of the scheme seems to be 2, which is the absolute value of the slope of the regression line based on log(n) and log(error)

Text

Description automatically generated

The Matlab code for ode.m is

f1 = @(t,y) (-150)\*y + 49 - 150\*t;

figure(1)

epsilon=0.1;

[t,u] = ode23(f1,[0,1],1/3 + epsilon);

for i=1:length(t)-1

delta\_t(i) = t(i + 1) - t(i);

end

plot(delta\_t)

title("epsilon = 0.1")

xlabel("time step")

ylabel("delta t")

figure(2)

epsilon=0.001;

[t,u] = ode23(f1,[0,1],1/3 + epsilon);

for i=1:length(t)-1

delta\_t(i) = t(i + 1) - t(i);

end

plot(delta\_t)

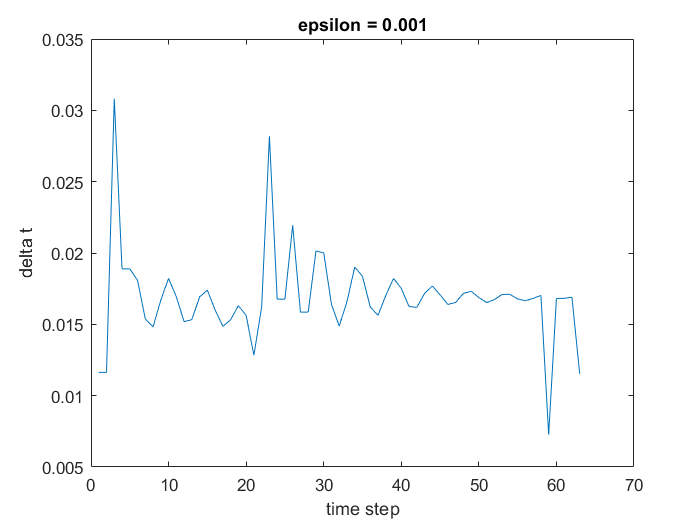
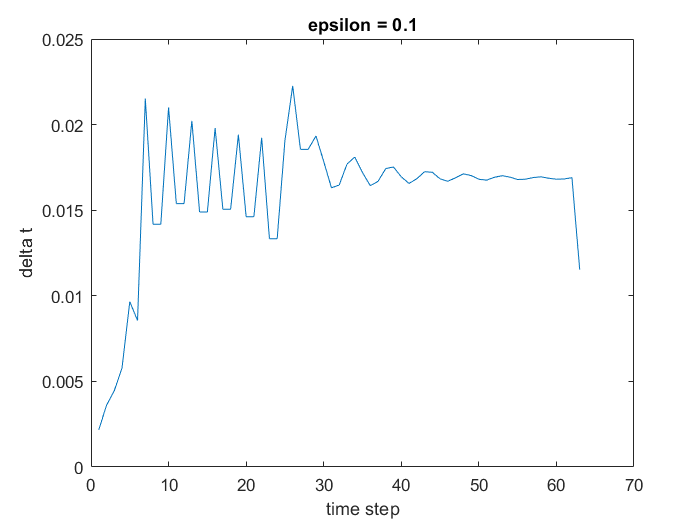
plot(delta\_t)

title("epsilon = 0.001")

xlabel("time step")

ylabel("delta t")

Plotting the delta\_t for both cases of different errors:



The difference in timestep seems to fluctuate in the first half and starts to stay stable in the latter half. Finally, the difference drops at the end.