ODE Lab: Creating your own ODE solver in MATLAB

Table of Contents

In this lab, you will write your own ODE solver for the Improved Euler method (also known as the Heun method), and compare its results to those of `ode45`.

You will also learn how to write a function in a separate m-file and execute it.

Opening the m-file lab3.m in the MATLAB editor, step through each part using cell mode to see the results. Compare the output with the PDF, which was generated from this m-file.

There are six (6) exercises in this lab that are to be handed in on the due date. Write your solutions in the template, including appropriate descriptions in each step. Save the .m files and submit them online on Quercus.

Student Information

Student Name: Fatema Rahman Farzin

Student Number: 1005423356

Creating new functions using m-files.

Create a new function in a separate m-file:

Specifics: Create a text file with the file name f.m with the following lines of code (text):

```
function y = f(a,b,c)
y = a+b+c;
```

Now MATLAB can call the new function f (which simply accepts 3 numbers and adds them together). To see how this works, type the following in the matlab command window: sum = f(1,2,3)

Exercise 1

Objective: Write your own ODE solver (using the Heun/Improved Euler Method).

Details: This m-file should be a function which accepts as variables (t0,tN,y0,h), where t0 and tN are the start and end points of the interval on which to solve the ODE, y0 is the initial condition of the ODE, and h is the stepsize. You may also want to pass the function into the ODE the way `ode45` does (check lab 2).

Note: you will need to use a loop to do this exercise. You will also need to recall the Heun/Improved Euler algorithm learned in lectures.

```
% separate heun.m file with the heun function attached
```

Exercise 2

Objective: Compare Heun with `ode45`.

Specifics: For the following initial-value problems (from lab 2, exercises 1, 4-6), approximate the solutions with your function from exercise 1 (Improved Euler Method). Plot the graphs of your Improved Euler Approximation with the `ode45` approximation.

(a) `y' = y tan t + sin t, y(0) = -1/2` from `t = 0` to `t = pi`

(b) `y' = 1 / y^2 , y(1) = 1` from `t=1` to `t=10`

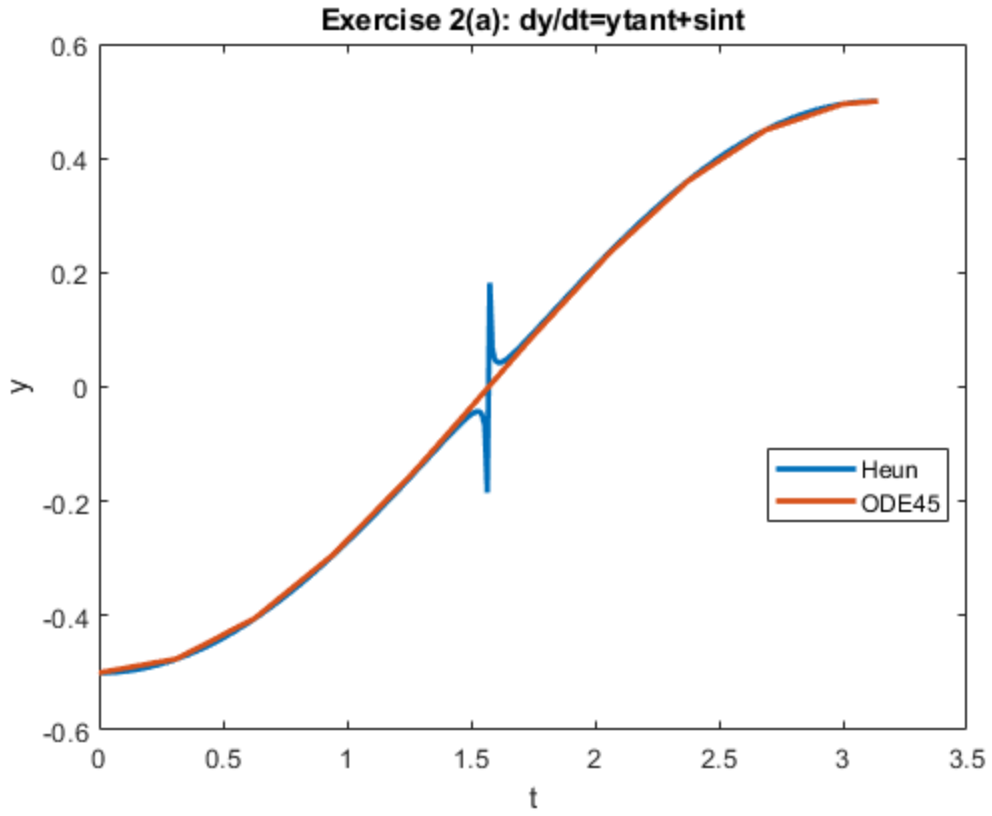
(c) `y' = 1 - t y / 2, y(0) = -1` from `t=0` to `t=10`

(d) `y' = y^3 - t^2, y(0) = 1` from `t=0` to `t=1`

Comment on any major differences, or the lack thereof. You do not need to reproduce all the code here. Simply make note of any differences for each of the four IVPs.

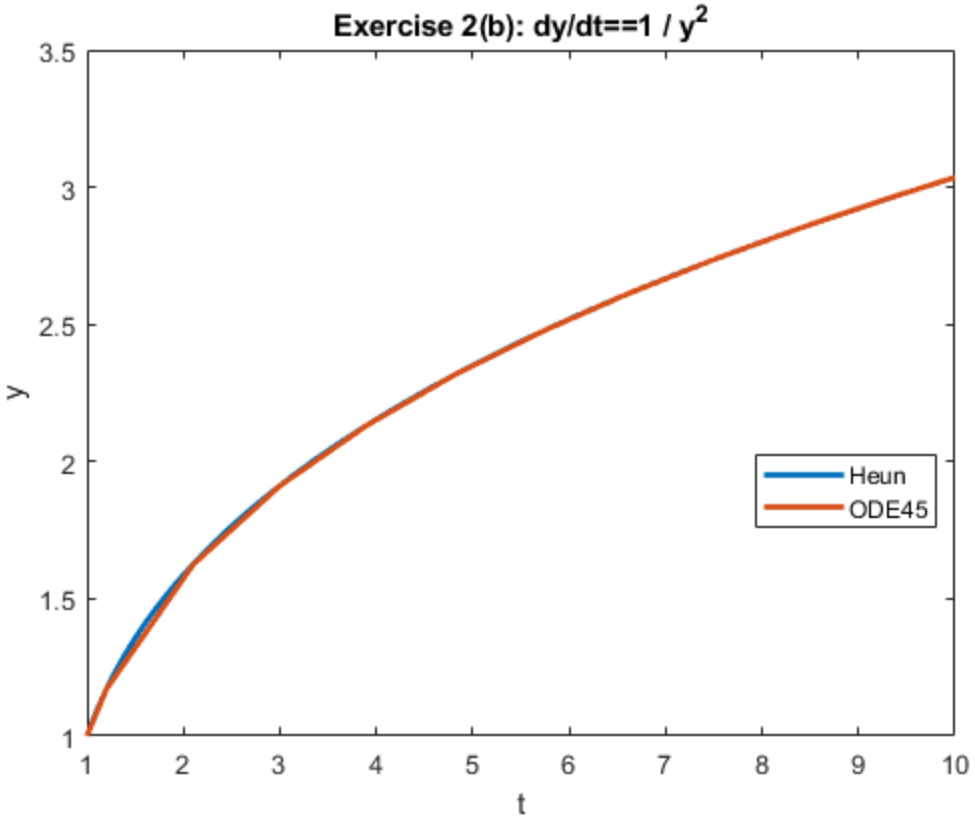
```
% (a) y' = y tan t + sin t, y(0) = -1/2 from t = 0 to t = pi
% Major differences between the ODE45 results and the Heun method result:
    % At around t=1.5, the Heun method curve gives a vertical asymptote,
 whereas the ode45 solution maintains a sinusoidal curve.         The value
 of t=1.5 is very close to pi/2, where tan(t) is undefined. Since the heun
 method is trying to compute y=(pi/2) using the slope at t=pi/2, the computed
 y values approach infinity, resulting in a vertical asymptote.
    % Aside from the vertical asymptote at t=pi/2, the Heun curve closely
 matches ode45 curve

clear all;
f = @(t,y) y*tan(t) + sin(t);
t0 = 0;
y0 = -0.5;
t1 = pi;
h=0.01;

% using ode45 to obtain solution
soln = ode45(f, [t0, t1], y0);

% using Heun method to obtain solution
[tsol, ysol] = heun(t0, t1, y0, h, f);

% plotting the heun solution
figure;
plot(tsol, ysol, 'LineWidth', 2);
hold on;
```

```
% plotting the ode45 solution
plot(soln.x, soln.y,'LineWidth', 2);
title('Exercise 2(a): dy/dt=ytant+sint');
xlabel('t');
ylabel('y');
legend('Heun','ODE45','Location','Best');
hold off;
```



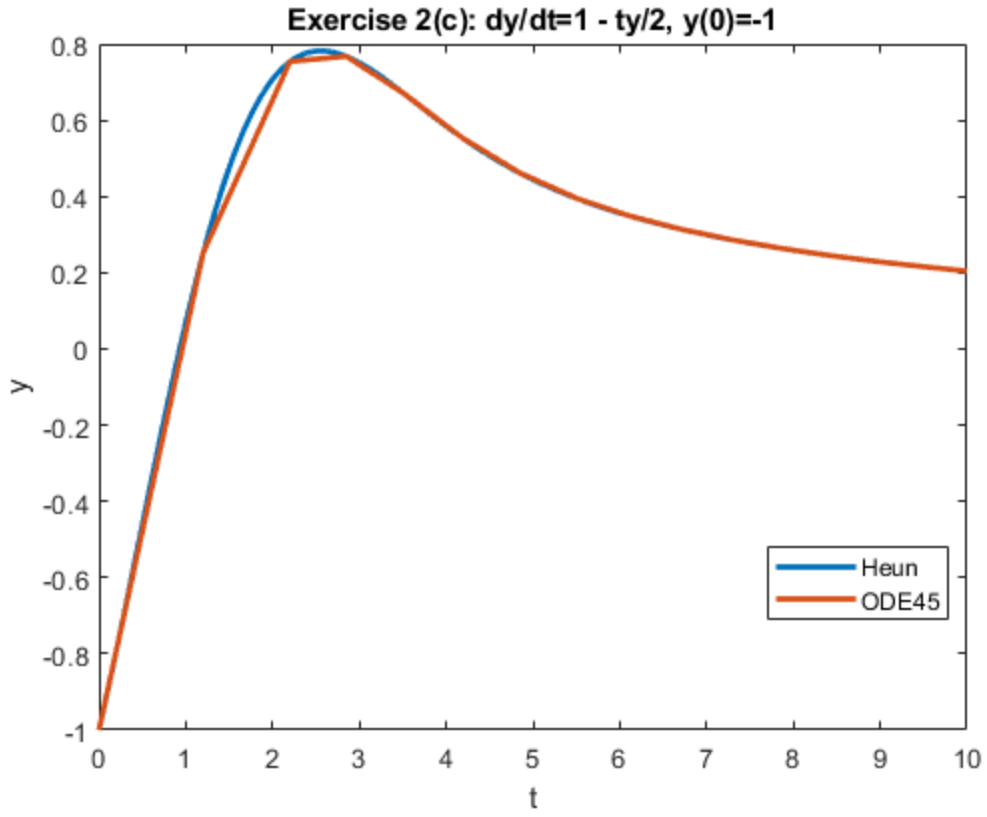

```
% (b) y' = 1 / y^2 , y(1) = 1 from t=1 to t=10
% Major differences between the ODE45 results and the Heun method result:
    % For this IVP, the ode45 and heun curves match each other closely
    % Difference can be noticed in the interval 1<t<3. This is likely due to
 the fact that the slope is changing more rapidly here compared to the rest of
 the curve.

    clear all;
f = @(t,y) 1/y^2;
t0 = 1;
y0 = 1;
t1 = 10;
h=0.01;

% using ode45 to obtain solution
soln = ode45(f, [t0, t1], y0);
```

```
% using Heun method to obtain solution
[tsol, ysol] = heun(t0, t1, y0, h, f);

% plotting the heun solution
figure;
plot(tsol, ysol, 'LineWidth', 2);
hold on;

% plotting the ode45 solution
plot(soln.x, soln.y,'LineWidth', 2);
title('Exercise 2(b): dy/dt==1 / y^2');
xlabel('t');
ylabel('y');
legend('Heun','ODE45','Location','Best');
hold off;
```




```
% (c) y' = 1 - t y / 2, y(0) = -1 from t=0 to t=10
% Major differences between the ODE45 results and the Heun method result:
    % For this IVP, the ode45 and heun curves match more closely towards the
 end of the interval, t>4.
    % Differences can be noticed in the interval 1<t<3. This is likely due to
 the fact that the slope is changing more rapidly here compared to the rest of
 the curve.

clear all;
f = @(t,y) 1 - t*y / 2;
```

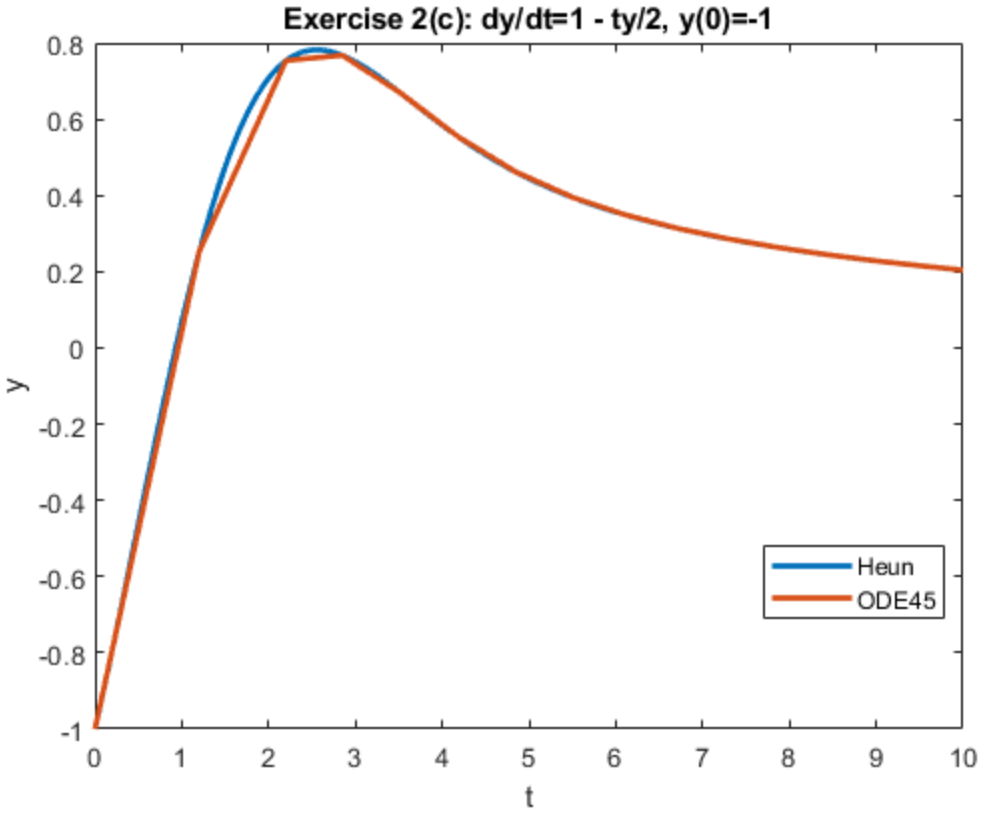
```
t0 = 0;
y0 = -1;
t1 = 10;
h=0.01;

% using ODE to obtain solution
soln = ode45(f, [t0, t1], y0);

% using Heun method to obtain solution
[tsol, ysol] = heun(t0, t1, y0, h, f);

% plotting the heun solution
figure;
plot(tsol, ysol, 'LineWidth', 2);
hold on;

% plotting the ode45 solution
plot(soln.x, soln.y,'LineWidth', 2);
title('Exercise 2(c): dy/dt=1 - ty/2, y(0)=-1');
xlabel('t');
ylabel('y');
legend('Heun','ODE45','Location','Best');
hold off;
```

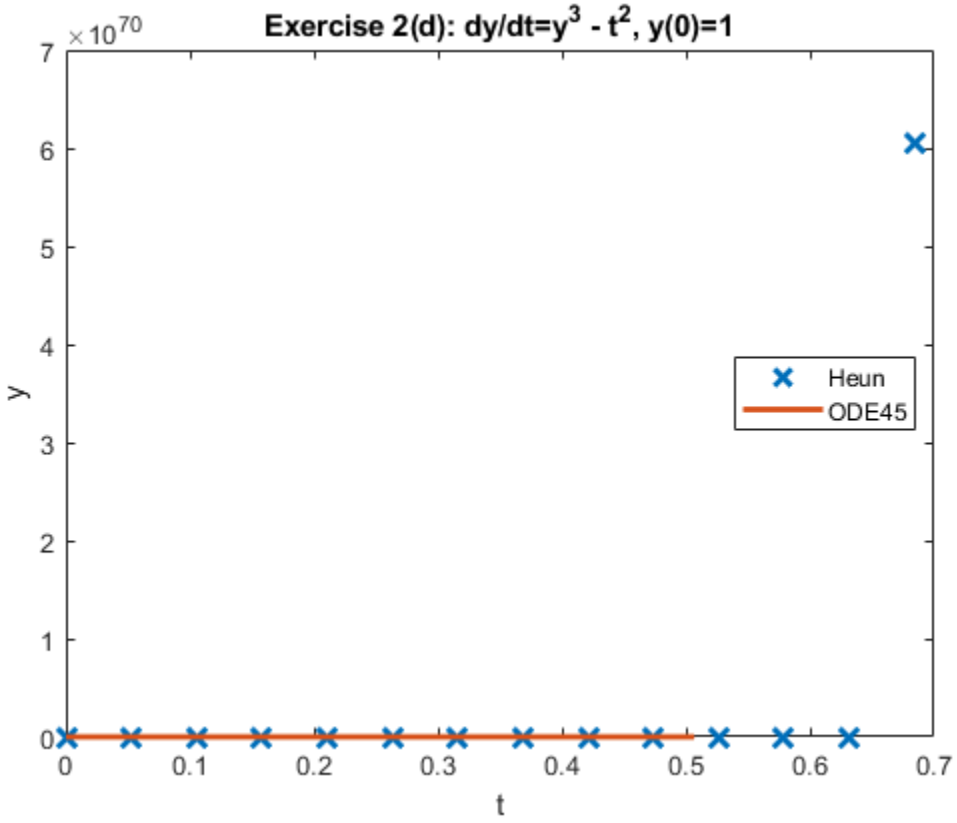



```
% (d) y' = y^3 - t^2, y(0) = 1 from t=0 to t=1
% Major differences between the ODE45 results and the Heun method result:
```

```
   % The heun and ode45 curves of this IVP do not match each other closely.
   % The ode45 solution when plotted separately shows rapidly increasing y
 values (vertical line) at around t=0.5, while still giving numerical y values
 for the rest of the interval 0.5<t<1. Compared to this, the heun curve jumps
 to a value of y=inf (especially when the step sizes are small).
   % Since the heun method computes y values based on the slope values, when
 the slope of the solution approaches an undefined value, the computed y
 values approach inf.

clear all;
f = @(t,y) y^3 - t^2;
t0 = 0;
y0 = 1;
t1 = 1;
h=0.05;

% using ODE to obtain solution
soln = ode45(f, [t0, t1], y0);

% using Heun method to obtain solution
[tsol, ysol] = heun(t0, t1, y0, h, f);

% plotting the heun solution
figure;
plot(tsol, ysol, 'x', 'MarkerSize', 10, 'LineWidth', 2);
hold on;

% plotting the ode45 solution
plot(soln.x, soln.y,'LineWidth', 2);
title('Exercise 2(d): dy/dt=y^3 - t^2, y(0)=1');
xlabel('t');
ylabel('y');
legend('Heun','ODE45','Location','Best');
hold off;
```

Warning: Failure at t=5.066046e-01. Unable to meet integration tolerances without reducing the step size below the smallest value allowed (1.776357e-15) at time t.


Exercise 2(c): dy/dt=1 - ty/2, y(0)=-1
y
t
Heun
ODE45


Exercise 2(d): dy/dt=y^3 - t^2, y(0)=1
×10^70
y
t
Heun
ODE45

Exercise 3

Objective: Use Euler's method and verify an estimate for the global error.

Details:

(a) Use Euler's method (you can use euler.m from iode) to solve the IVP

```
y' = 2 t sqrt( 1 - y^2 ) , y(0) = 0
```

from `t=0` to `t=0.5`.

(b) Calculate the solution of the IVP and evaluate it at `t=0.5`.

(c) Read the attached derivation of an estimate of the global error for Euler's method. Type out the resulting bound for En here in a comment. Define each variable.

(d) Compute the error estimate for `t=0.5` and compare with the actual error.

(e) Change the time step and compare the new error estimate with the actual error. Comment on how it confirms the order of Euler's method.

```
% (a) using IODE euler.m to solve the ODE

f = @(t,x) 2 .* t .* sqrt(1-x^2);               % the given ODE
x0 = 0;                                         % the initial condition value
h = 0.005;                                      % step size h
t0 = 0;                                         % left bound of interval
tN = 0.5;                                       % right bound of interval
N = (tN-t0)/h;                                  % number of intervals
tc = linspace(t0, tN, N);                       % array of independent
 variable

xc = euler(f, x0, tc);                          % using the euler.m function
fprintf("Approximate solution at t=%f, yn = %f\n", tc(N), xc(N));

% (b)
% Solution of the IVP: y=sin(t^2)
% at the given t value, y(t=0.5) = 0.25

% (c) Error bound, En = |yn - y(tn)|, where:
    % En is the bound of the error
    % yn is the approximated solution obtained using Euler's method
    % y(tn) is the true solution at time, tn
    % y(t) is the exact solution of the IVP obtained by solving the IVP
 manually

% Substituting in the following values:
    % exact solution y(t=0.5) = 0.25
    % the approximate solution (obtained using euler.m) yn = 0.245017

En = abs(xc(N)-0.25);
fprintf("Actual error: %f\n", En);
```

```
% (d) Error estimation
M = 2;
E_est = ((1+M)/2) * h^2 * M * N;
fprintf("Estimated error with step size h: %f\n", E_est);
% Variables used in calculation of estimated error, E_est:
    % M = upper bound of f(y, t) and its partial derivatives in [t0, tN]
    % h = step size
    % N = interval, in this case the last interval since t=0.5 is the endpoint

% doubling the step size:
h = h*2;                                % updating h
N = (tN-t0)/h;                          % since h was updated, N will change
 as well
E_est = ((1+M)/2) * h^2 * M * N;        % E_est for the new step size
fprintf("Estimated error with step size h*2: %f\n", E_est);

% As seen above, doubling the step size also doubles the error estimate.
% Similarly it can be shown that decreasing the step size by a factor of c,
 decreases the error estimate by a factor of c.
% This shows that the error estimate is linearly proportional to the step size
 for large n near the end of the interval,
% which confirms that Euler's method is first order.
```

```
Approximate solution at t=0.500000, yn = 0.245017
Actual error: 0.004983
Estimated error with step size h: 0.007500
Estimated error with step size h*2: 0.015000
```

Adaptive Step Size

As mentioned in lab 2, the step size in `ode45` is adapted to a specific error tolerance.

The idea of adaptive step size is to change the step size `h` to a smaller number whenever the derivative of the solution changes quickly. This is done by evaluating f(t,y) and checking how it changes from one iteration to the next.

Exercise 4

Objective: Create an Adaptive Euler method, with an adaptive step size `h`.

Details: Create an m-file which accepts the variables `(t0,tN,y0,h)`, as in exercise 1, where `h` is an initial step size. You may also want to pass the function into the ODE the way `ode45` does.

Create an implementation of Euler's method by modifying your solution to exercise 1. Change it to include the following:

(a) On each timestep, make two estimates of the value of the solution at the end of the timestep: `Y` from one Euler step of size `h` and `Z` from two successive Euler steps of size `h/2`. The difference in these two values is an estimate for the error.

(b) Let `tol=1e-8` and `D=Z-Y`. If `abs(D)<tol`, declare the step to be successful and set the new solution value to be `Z+D`. This value has local error `O(h^3)`. If `abs(D)>=tol`, reject this step and repeat it with a new step size, from (c).

(c) Update the step size as `h = 0.9*h*min(max(tol/abs(D),0.3),2)`.

Comment on what the formula for updating the step size is attempting to achieve.

```
% separate adaptiveEuler.m file with the adaptive euler function attached

% Answer (c)
% The formula for updating the step size is attempting to decrease or increase
 the step size based on the ratio of the estimated error and the error
 tolerance using the following principle:
% When estimated error << error tolerance, max(tol/abs(D),0.3) evaluates to
 a very large number, which forces the min function to evaluate to 2. This
 ensures that the step size, h never increases beyond h = 0.9*h*2
% On the other hand, when estimated error >> error tolerance, max(tol/
abs(D),0.3) evaluates to a very small number, which forces the min function
 to evaluate to said small number. This decreases the step size, h which
 ultimately leads to decreasing the error in the next iteration
```

Exercise 5

Objective: Compare Euler to your Adaptive Euler method.

Details: Consider the IVP from exercise 3.

`y' = 2 t sqrt( 1 - y^2 ) , y(0) = 0`

from `t=0` to `t=0.75`.

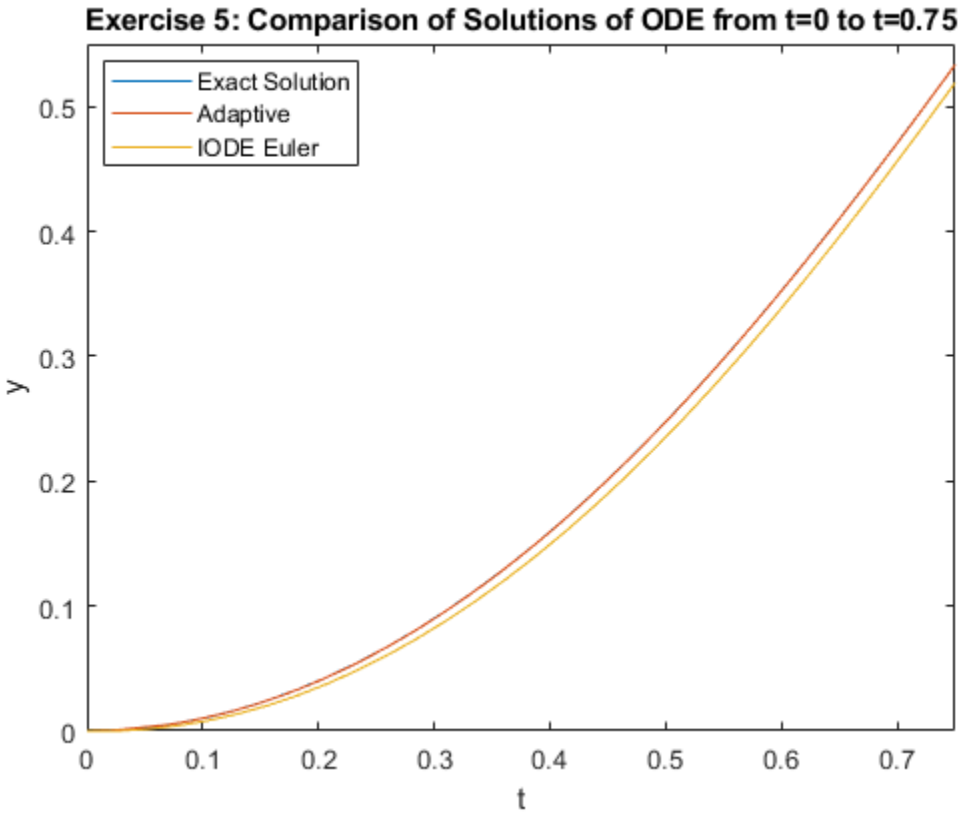
(a) Use Euler method to approximate the solution from `t=0` to `t=0.75` with `h=0.025`.

(b) Use your Adaptive Euler method to approximate the solution from `t=0` to `t=0.75` with initial `h=0.025`.

(c) Plot both approximations together with the exact solution.

```
% (a) using the IODE euler function to approximate solution
f = @(t,y) 2 .* t .* sqrt(1-y^2);               % the given ODE
y0 = 0;                                         % the initial condition value
t0 = 0;
tN = 0.75;
h = 0.025;                                      % h = step size
N = round((tN-t0)/h);                           % N = number of intervals

t_arr = linspace(t0, tN, N);                  % array of independent variable
y_euler = euler(f, y0, t_arr);

% (b) using the adaptive euler function to approximate solution
[t_adapt, y_adapt] = adaptiveEuler(t0, tN, y0, h, f);

% (c) plotting both solutions together with the exact solution:
sol = @(t) sin(t.^2);               % function for the exact solution
y_sol = sol(t_arr);

figure;
plot(t_arr, y_sol);
xlim([0 0.75]);
ylim([0 0.55]);
hold on;
plot(t_adapt, y_adapt);
```

```
hold on;
plot(t_arr, y_euler);
hold off;
legend( "Exact Solution", "Adaptive", "IODE Euler", 'Location', 'Best');
title('Exercise 5: Comparison of Solutions of ODE from t=0 to t=0.75');
xlabel('t');
ylabel('y');
```



Exercise 6

Objective: Problems with Numerical Methods.

Details: Consider the IVP from exercise 3 (and 5).

(a) From the two approximations calculated in exercise 5, which one is closer to the actual solution (done in 3.b)? Explain why.

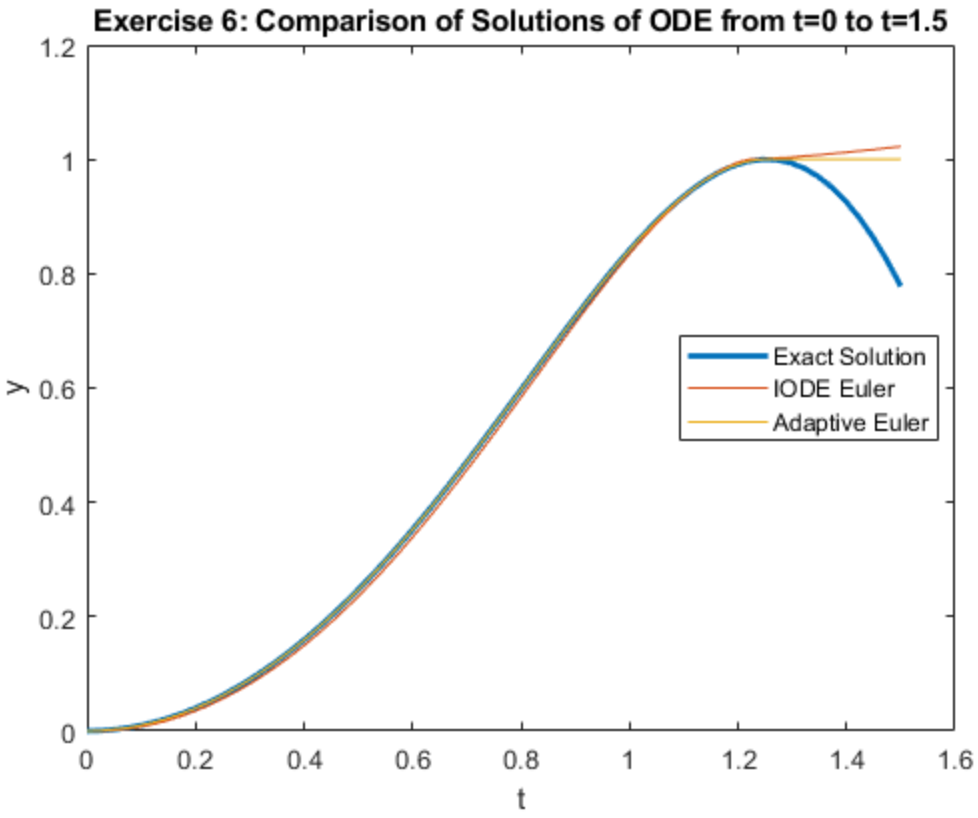
(b) Plot the exact solution (from exercise 3.b), the Euler's approximation (from exercise 3.a) and the adaptive Euler's approximation (from exercise 5) from `t=0` to `t=1.5`.

(c) Notice how the exact solution and the approximations become very different. Why is that? Write your answer as a comment.

```
% Answer (a)
% From the two approximations calculated in exercise 5, the adaptive euler
 function is closer to the actual solution.
```

```
% This is because for large n, near the end of the interval, the error for the
 euler method increases linearly with the step size.
% Whereas, for the adaptive euler method, the step size is adjusted on every
 iteration to make sure the error is within a certain tolerance,
% resulting in a smaller global error by the end of the interval.

f = @(t,y) 2 .* t .* sqrt(1-y^2);              % the given ODE
y0 = 0;                                        % the initial condition value
t0 = 0;
tN = 1.5;
h = 0.025;                                     % h = step size
N = round((tN-t0)/h);                          % N = number of intervals

% (b) plotting the exact solution:
t_sol = linspace(t0, tN, N);
sol = @(t) sin(t.^2);
y_sol = sol(t_sol);

figure;
plot(t_sol, y_sol, 'LineWidth', 2);
hold on;

% plotting the euler approximation:
t_euler = linspace(t0, tN, N);                 % array of independent
 variable
y_euler = euler(f, y0, t_euler);               % approximating solution using
 IODE Euler function

plot(t_euler, y_euler);
hold on;

% plotting the adaptive solution:
[t_adapt, y_adapt] = adaptiveEuler(t0, tN, y0, h, f);

plot(t_adapt, y_adapt);
hold off;
legend("Exact Solution", "IODE Euler", "Adaptive Euler", 'Location', 'Best');
title('Exercise 6: Comparison of Solutions of ODE from t=0 to t=1.5');
xlabel('t');
ylabel('y');

% Answer (c)
% The exact solution becomes very different in the plot from the point
 starting at y=1.
% In addition, MATLAB gives the warning - "Imaginary parts of complex X and/or
 Y arguments ignored".
% This is because for y>1, part of the ODE, sqrt(1-y^2) evaluates to a
 complex number and the function is trying to use this ODE to approximate the
 solution.
% On the other hand, the exact solution does not depend on the term sqrt(1-
y^2) and simply evaluates y = sin(t^2)
```

```
Warning: Imaginary parts of complex X and/or Y arguments ignored.
Warning: Imaginary parts of complex X and/or Y arguments ignored.
```



Published with MATLAB® R2023a