ODE Lab: Creating your own ODE solver in MATLAB

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: Mustafa Khan

Student Number: 1006717037

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.

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.

```
g = @(t,y) y .* tan(t) + sin(t);
t0 = 0;
tN = pi;
y0 = -0.5
```

```
y0 = -0.5000
```

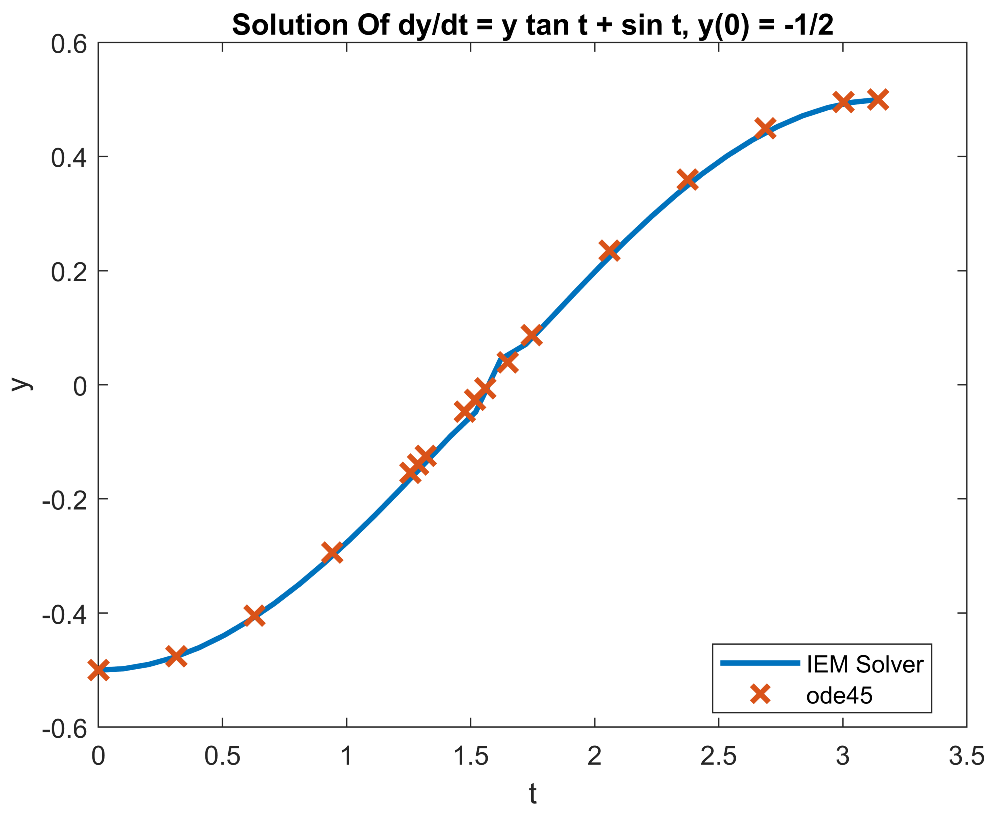
```
h = 0.1
```

```
h = 0.1000
```

```
soln = IEM(g, t0, tN, y0, h);
ode45_soln = ode45(g, [t0, tN], y0);

%Visualizing and comparing the solution

% By using the method of integrating factor and multiplying the ODE in
% stanard form by mu(x) = cos(x), you get the solution shown in yy.

tt_estimate = linspace(t0, tN, length(soln))
```

```
tt_estimate = 1×32
         0    0.1013    0.2027    0.3040    0.4054    0.5067    0.6081    0.7094 ⋯
```

```

% Plot both on the same figure, plotting the approximation with x's
plot(tt_estimate, soln, ode45_soln.x, ode45_soln.y, 'x', 'MarkerSize',10, 'LineWidth', 2);
% NOTE: the MarkerSize and LineWidth are larger than their defaults of 6
% and 1, respectively.  This makes the print out more readable.

% Add a label to the axis and a legend
xlabel('t');
ylabel('y');
title('Solution Of dy/dt = y tan t + sin t, y(0) = -1/2');
legend('IEM Solver', 'ode45', 'Location','Best');
```



```
%% OBSERVATIONS
% Using IEM Solver, if the stepsize is small at around pi/2, the tan(t)
% term in the ODE causes a bump/divergence than ode45.
```

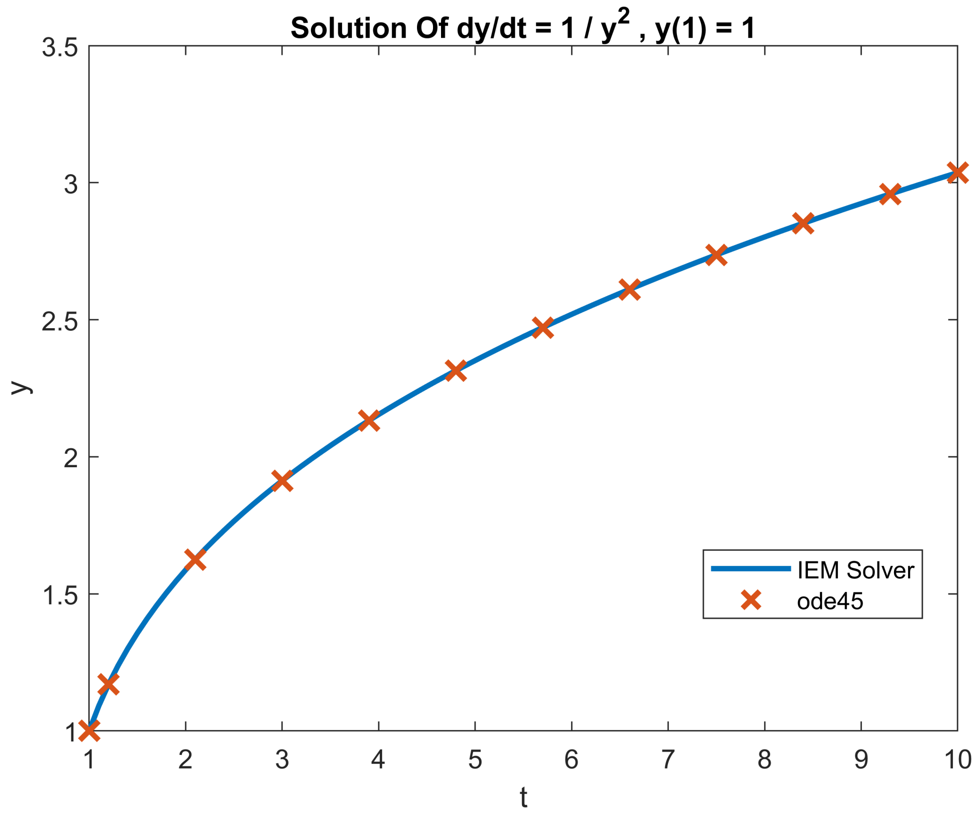
```
g = @(t,y) 1/(y.^2);

t0 = 1;
y0 = 1;
tN = 10;
h = 0.1
```

```
h = 0.1000
```

```
soln = IEM(g, t0, tN, y0, h);
ode45_soln = ode45(g, [t0, tN], y0);

%Visualizing and comparing the solution
tt_estimate = linspace(t0, tN, length(soln))
```

```
tt_estimate = 1×91
    1.0000    1.1000    1.2000    1.3000    1.4000    1.5000    1.6000    1.7000 ···
```

```
% Plot both on the same figure, plotting the approximation with x's
plot(tt_estimate, soln, ode45_soln.x, ode45_soln.y, 'x', 'MarkerSize',10, 'LineWidth', 2);
% NOTE: the MarkerSize and LineWidth are larger than their defaults of 6
% and 1, respectively.  This makes the print out more readable.
% Add a label to the axis and a legend
xlabel('t');
ylabel('y');
title('Solution Of dy/dt = 1 / y^2 , y(1) = 1');
legend('IEM Solver', 'ode45', 'Location','Best');
```



```
%% OBSERVATION
% IEM Solver matches well with ode45 even if the step size isn't
% particulairly small.
```

```
g = @(t,y) 1 - t.*y/2;

t0 = 0;
y0 = -1;
tN = 10;
h = 0.1
```

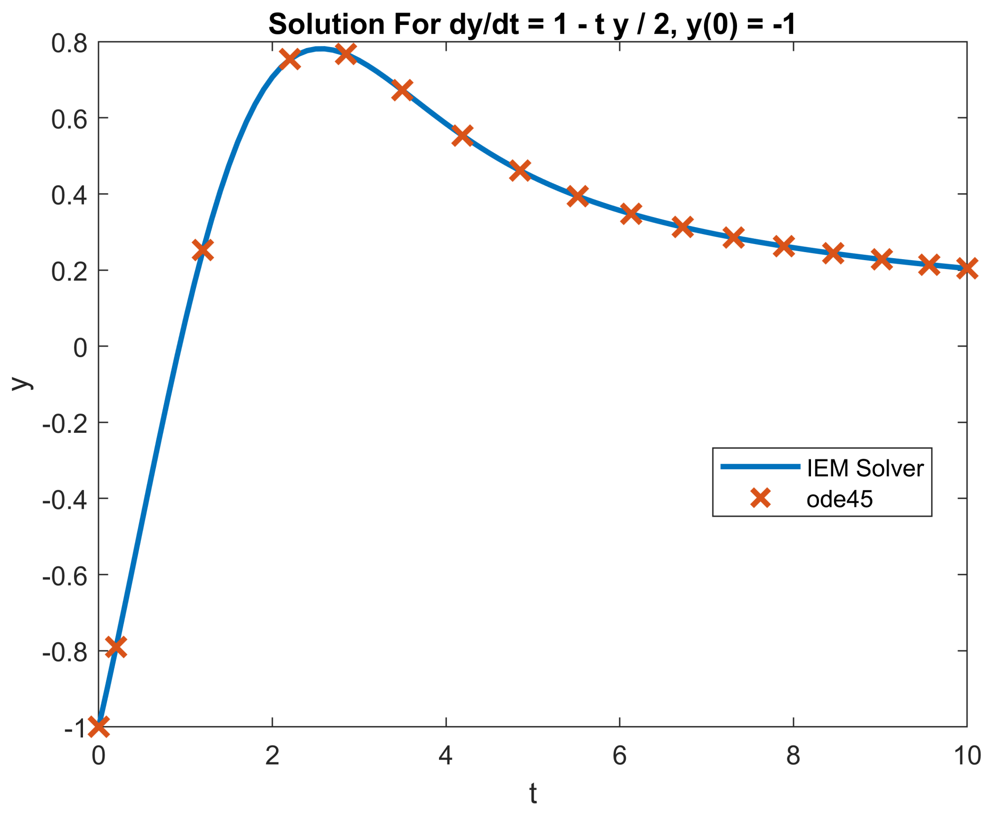
```
h = 0.1000
```

```
soln = IEM(g, t0, tN, y0, h);
ode45_soln = ode45(g, [t0, tN], y0);

%Visualizing and comparing the solution
tt_estimate = linspace(t0, tN, length(soln))
```

```
tt_estimate = 1×101
```

```
        0    0.1000    0.2000    0.3000    0.4000    0.5000    0.6000    0.7000 ⋯
```

```
% Plot both on the same figure, plotting the approximation with x's
plot(tt_estimate, soln, ode45_soln.x, ode45_soln.y, 'x', 'MarkerSize',10, 'LineWidth', 2);
% NOTE: the MarkerSize and LineWidth are larger than their defaults of 6
% and 1, respectively.  This makes the print out more readable.

% Add a label to the axis and a legend
xlabel('t');
ylabel('y');
title('Solution For dy/dt = 1 - t y / 2, y(0) = -1');
legend('IEM Solver', 'ode45', 'Location','Best');
```



```
%% OBSERVATION
% IEM Solver mathces well with ode45 with small step sizes.
```

```
g = @(t,y) y.^3 - t.^2;

t0 = 0;
y0 = 1;
tN = 1;
h = 0.0001
```

```
h = 1.0000e-04
```

```
soln = IEM(g, t0, tN, y0, h);
```

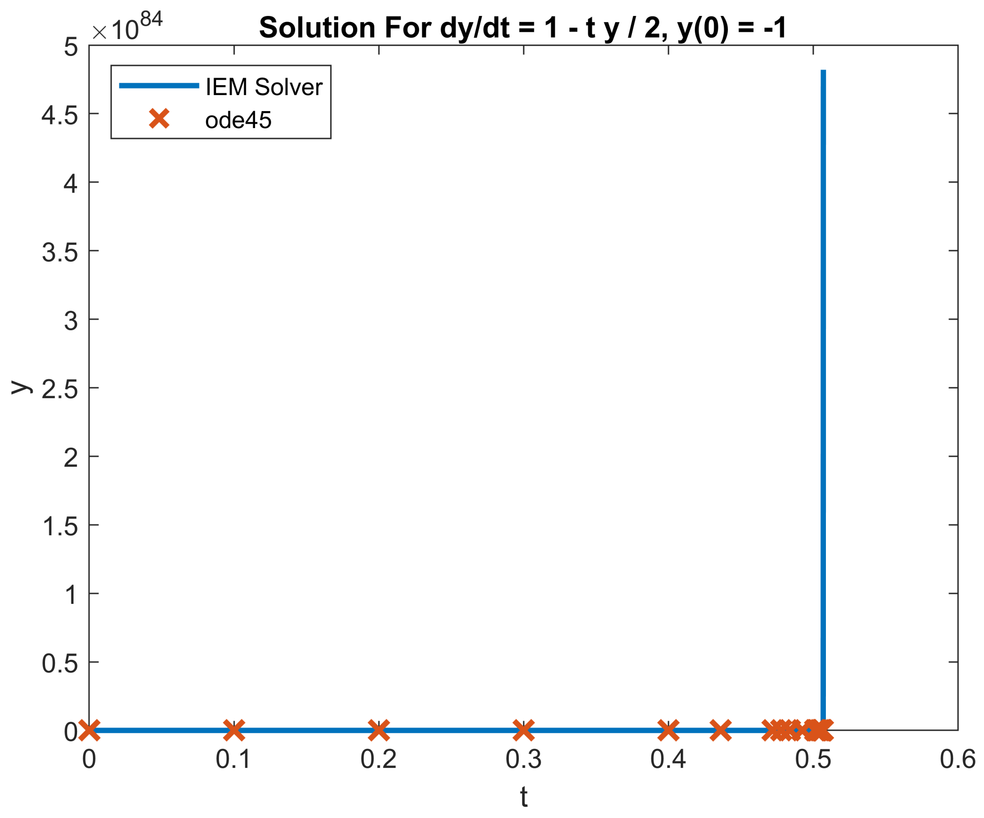
```
ode45_soln = ode45(g, [t0, tN], y0);
```

```
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.
```

```
%Visualizing and comparing the solution
tt_estimate = linspace(t0, tN, length(soln))
```

```
tt_estimate = 1×10001
         0    0.0001    0.0002    0.0003    0.0004    0.0005    0.0006    0.0007 ···
```

```
% Plot both on the same figure, plotting the approximation with x's
plot(tt_estimate, soln, ode45_soln.x, ode45_soln.y, 'x', 'MarkerSize',10, 'LineWidth', 2);

% Add a label to the axis and a legend
xlabel('t');
ylabel('y');
title('Solution For dy/dt = 1 - t y / 2, y(0) = -1');
legend('IEM Solver', 'ode45', 'Location','northwest');
```



```
%% OBSERVATION
% IEM Solver does not match with ode45 because the slope becomes very large
% at t=0.5, which would require very small stepsizes for ode45. Since this
% breaks the threshold of ode45 (which uses adaptive step sizes) it stops
% much earlier than IEM Solver which uses constant step sizes.
```

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`.

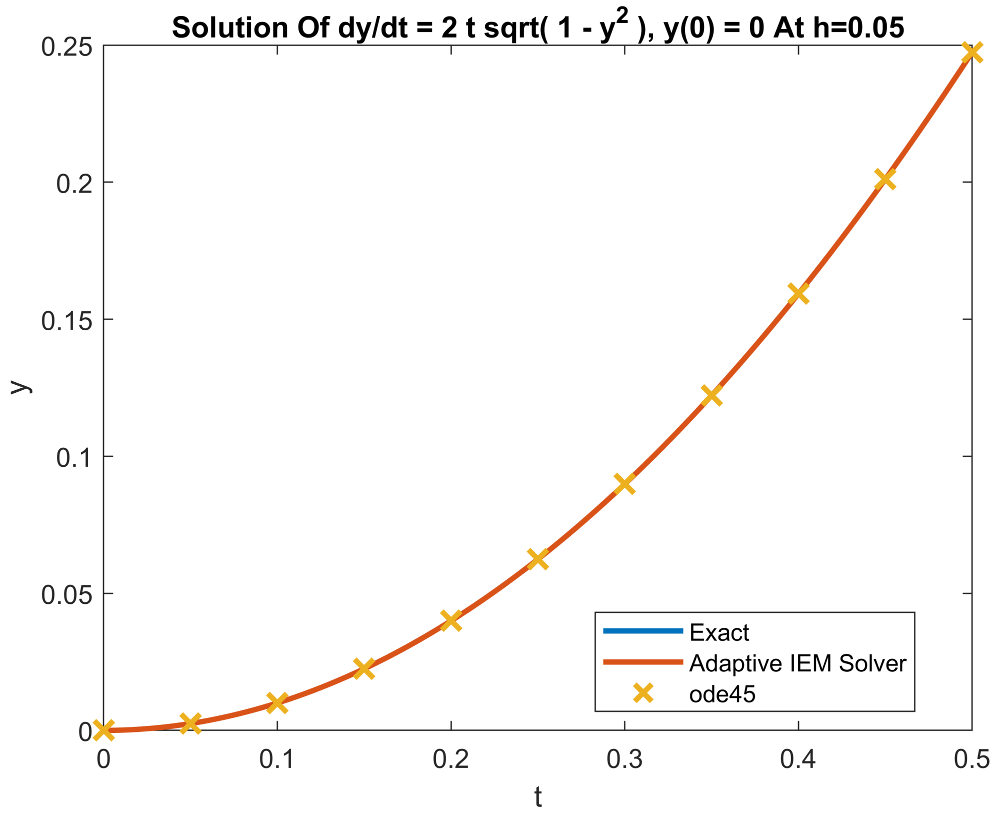
```
g = @(t,y) 2.*t*(1-y.^2).^(0.5)
```

```
g = function_handle with value:
    @(t,y)2.*t*(1-y.^2).^(0.5)
```

```

t0 = 0;
tN = 0.5;
y0 = 0;
h = 0.005;
soln = IEM(g, t0, tN, y0, h);
ode45_soln = ode45(g, [t0, tN], y0);

%Visualizing and comparing the solution

% By using the method of integrating factor and multiplying the ODE in
% stanard form by mu(x) = cos(x), you get the solution shown in yy.

tt_estimate = linspace(t0, tN, length(soln))
```

```
tt_estimate = 1×101
         0    0.0050    0.0100    0.0150    0.0200    0.0250    0.0300    0.0350 ⋯
```

```
tt = linspace(t0,tN,100);
yy = sin(tt.^2);

% Plot both on the same figure, plotting the approximation with x's
plot(tt, yy, tt_estimate, soln, ode45_soln.x, ode45_soln.y, 'x', 'MarkerSize',10, 'LineWidth',

% Add a label to the axis and a legend
xlabel('t');
ylabel('y');
title('Solution Of dy/dt = 2 t sqrt( 1 - y^2 ), y(0) = 0 At h=0.05');
legend('Exact', 'Adaptive IEM Solver', 'ode45', 'Location','Best');
```



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

```
% By solving using method of seperable variables we get y = sin(t^2 + c).
% The solution to the IVP at t=0.5 is
% Note: c = 0 when y(0).
% y(0.5) = sin(0.5^2) = 0.2474
```

(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.

```
%  ANSWER TO C
% |f| < 2*0.5*1 = 1
% M = 1
%
% ∂tf = 2*sqrt(1-y^2)
% |∂tf| < 2
% M = 2
%
% fmax = 2*0.5*sqrt(1-0)
% fmax = 1
%
% ymax = y0 + fmax*0.5
%       = 0 + 1*0.5
%      = 0.5
%
% ∂yf = 2t*-2y/(2sqrt(1-y^2) = -2ty/sqrt(1-y^2)
% |∂yf| < 2*0.5*0.5/sqrt(1-0.5^2) = 0.577
% M = 0.577
%
% M = 2

% Therefore En = (1+M)*Δt*(e^(M*Δt*n)-1)/2
%
%    where Δt is the time step, M is the upper bound of f, fy, fx (in this
%    case 2), and n is the current time step number. (ex the third step)
```

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

% En = (1+M)*Δt*(e^(M*Δt*n)-1)/2
%    = 3*

%error = sin(0.25) - s(end)
%  actual error = 0.0047

%
% (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.

% Making the time step smaller gives a better approxmation therefore the
% error decreases
```

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

```
f = @(t,y) 2.*t.*sqrt(1 - y^2);
stepsize = 0.0001;
t = 0:stepsize:0.5;

s = euler(f,0,t);

actual_error = sin(0.25) - s(end)
```

```
actual_error = 4.7221e-05
```

```
%       = 0.000047221
```

(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.

```
% En = (1+2) * 0.0001 * (exp(2 * 0.0001 * 0.5/0.0001) - 1)/2
%    = 0.00025774

f = @(t,y) 2.*t.*sqrt(1 - y^2);
stepsize = 0.25;
t = 0:stepsize:0.5;

s = euler(f,0,t);
```

```
actual_error = sin(0.25) - s(end)
```

```
actual_error = 0.1224
```

```
%       = 0.1224

% En = (1+2) * 0.25 * (exp(2 * 0.25 * 0.5/0.25) - 1)/2
%    = 0.6444


% Eulers method has a global error on the order of h. It can be seen that
% when h is large, so is the error, and when h is small so is the error.

% Making the time step smaller gives a better approxmation therefore the
% error decreasess
```

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.

Answer: D is the difference between improved euler with full step size vs. half a step size.

If a step size results in a D value that is under a tolerance, then improved euler method is used. on the other hand, if D is greater than the tolerance then the step size is changed to be smaller.

h is updated to h = 0.9*h*min(max(tol/abs(D), 0.3), 2);. Note that tol/abs(D) shows how far the tolerance abs(D) is in percentage terms (or as a ratio). In our case, this will always be a value less than 1.

The 0.9 is a safety factor to ensure success on the next try. The minimum and maximum are to prevent extreme changes from the previous stepsize. This should, in principle give an error of about 0.9*tol in the next try.

Note as well that max(tol/abs(D), 0.3) is so that we don't need to do any computationally expensive calculation when tol/abs(D) is a really small number. if that's the case, we select 0.3 and go with that. The min(0.3, 2) or min(tol/abs(D), 2) will never be 2 in our case but the logic for including a min() is the same, we don't want extreme changes from previous stepsizes.

Exercise 5

Objective: Compare Euler to your Adaptive Euler method.

Details: Consider the IVP from exercise 3.

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

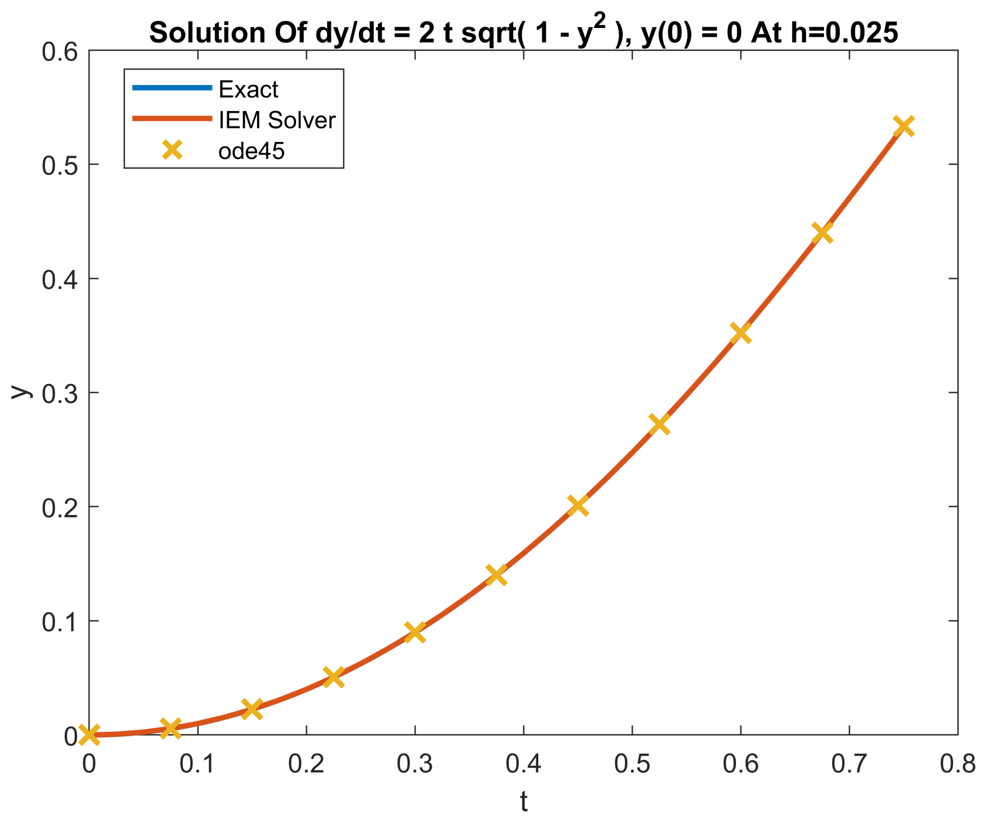
```
g = @(t,y) 2.*t*(1-y.^2).^(0.5)
```

```
g = function_handle with value:
    @(t,y)2.*t*(1-y.^2).^(0.5)
```

```

t0 = 0;
tN = 0.75;
y0 = 0;
h = 0.025;
soln = IEM(g, t0, tN, y0, h);
ode45_soln = ode45(g, [t0, tN], y0);

%Visualizing and comparing the solution

% By using the method of integrating factor and multiplying the ODE in
% stanard form by mu(x) = cos(x), you get the solution shown in yy.

tt_estimate = linspace(t0, tN, length(soln))
```

```
tt_estimate = 1×31
         0    0.0250    0.0500    0.0750    0.1000    0.1250    0.1500    0.1750 ···
```

```
tt = linspace(t0,tN,100);
yy = sin(tt.^2);

% Plot both on the same figure, plotting the approximation with x's
plot(tt, yy, tt_estimate, soln, ode45_soln.x, ode45_soln.y, 'x', 'MarkerSize',10, 'LineWidth',

% Add a label to the axis and a legend
xlabel('t');
ylabel('y');
title('Solution Of dy/dt = 2 t sqrt( 1 - y^2 ), y(0) = 0 At h=0.025');
legend('Exact', 'IEM Solver', 'ode45', 'Location','Best');
```



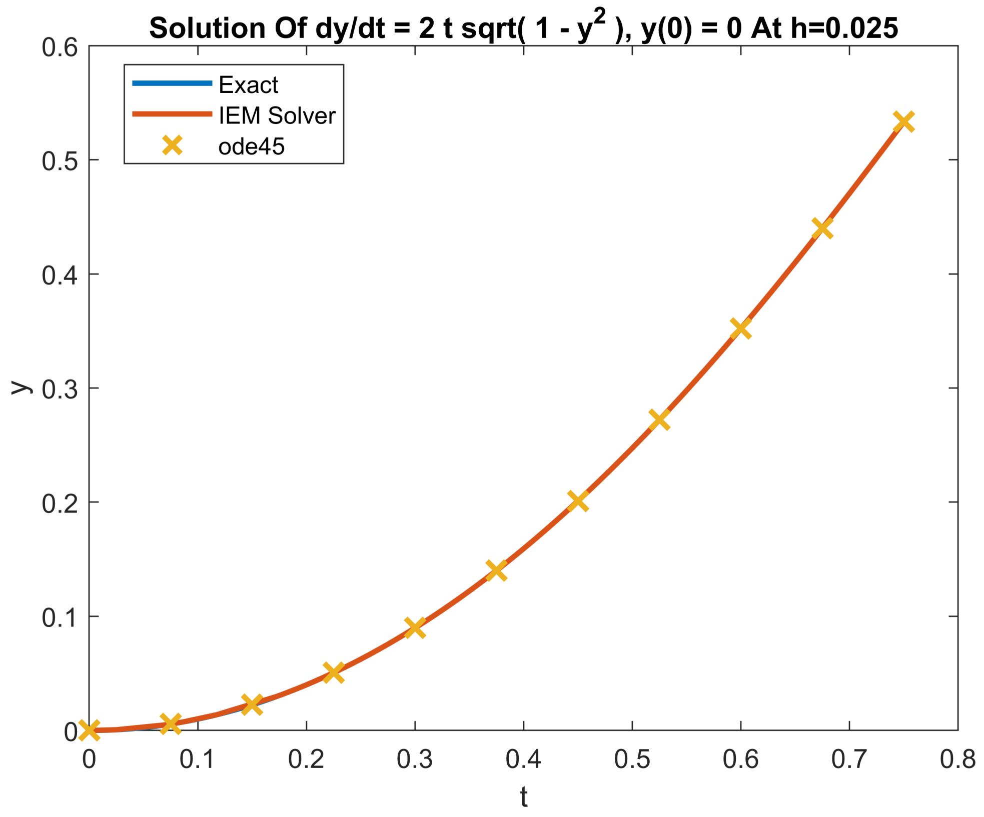
(b) Use your Adaptive Euler method to approximate the solution from `t=0` to `t=0.75` with initial `h=0.025`.

```
g = @(t,y) 2.*t*(1-y.^2).^(0.5)
```

```
g = function_handle with value:
    @(t,y)2.*t*(1-y.^2).^(0.5)
```

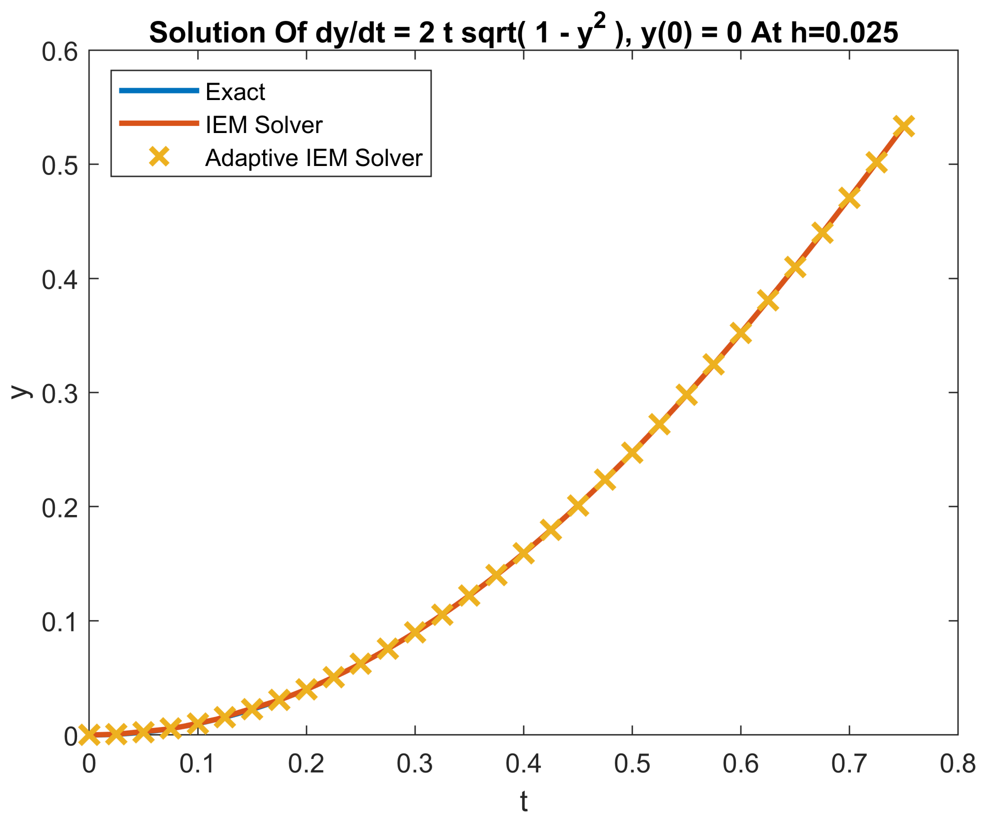
```

t0 = 0;
tN = 0.75;
y0 = 0;
h = 0.025;
[adaptive_soln, tt_adaptive] = adaptiveIEM(g, t0, tN, y0, h);
ode45_soln = ode45(g, [t0, tN], y0);

%Visualizing and comparing the solution

% By using the method of integrating factor and multiplying the ODE in
% stanard form by mu(x) = cos(x), you get the solution shown in yy.

tt_estimate = linspace(t0, tN, length(soln));
tt = linspace(t0,tN,100);
yy = sin(tt.^2);

% Plot both on the same figure, plotting the approximation with x's
plot(tt, yy, tt_adaptive, adaptive_soln, ode45_soln.x, ode45_soln.y, 'x', 'MarkerSize',10, 'Li

% Add a label to the axis and a legend
```

```
xlabel('t');
ylabel('y');
title('Solution Of dy/dt = 2 t sqrt( 1 - y^2 ), y(0) = 0 At h=0.025');
legend('Exact', 'IEM Solver', 'ode45', 'Location','Best');
```



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

```
% Plot both on the same figure, plotting the approximation with x's
plot(tt, yy, tt_adaptive, adaptive_soln, tt_estimate, soln, 'x', 'MarkerSize',10, 'LineWidth',

% Add a label to the axis and a legend
xlabel('t');
ylabel('y');
title('Solution Of dy/dt = 2 t sqrt( 1 - y^2 ), y(0) = 0 At h=0.025');
legend('Exact', 'IEM Solver', 'Adaptive IEM Solver', 'Location','northwest');
```



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.

```
% Unlike the Improved Euler Method, the Adaptive IEM is closer to the
% actual solution. While the IEM had a fixed step-size, in the Adaptive IEM
% a local error estimate can be used to decide how stepsize, h, should be
% modified to achieve a desired accuracy. As explained in the answer to 4.c.
% The 0.9 is a safety factor to ensure success on the next try.
% The minimum and maximum are to prevent extreme changes from the
% previous stepsize. These allow the numerical approximations to be
% much more accurate and within a certain error tolerance.
```

(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`.

```
g = @(t,y) 2.*t*(1-y.^2).^(0.5)
```

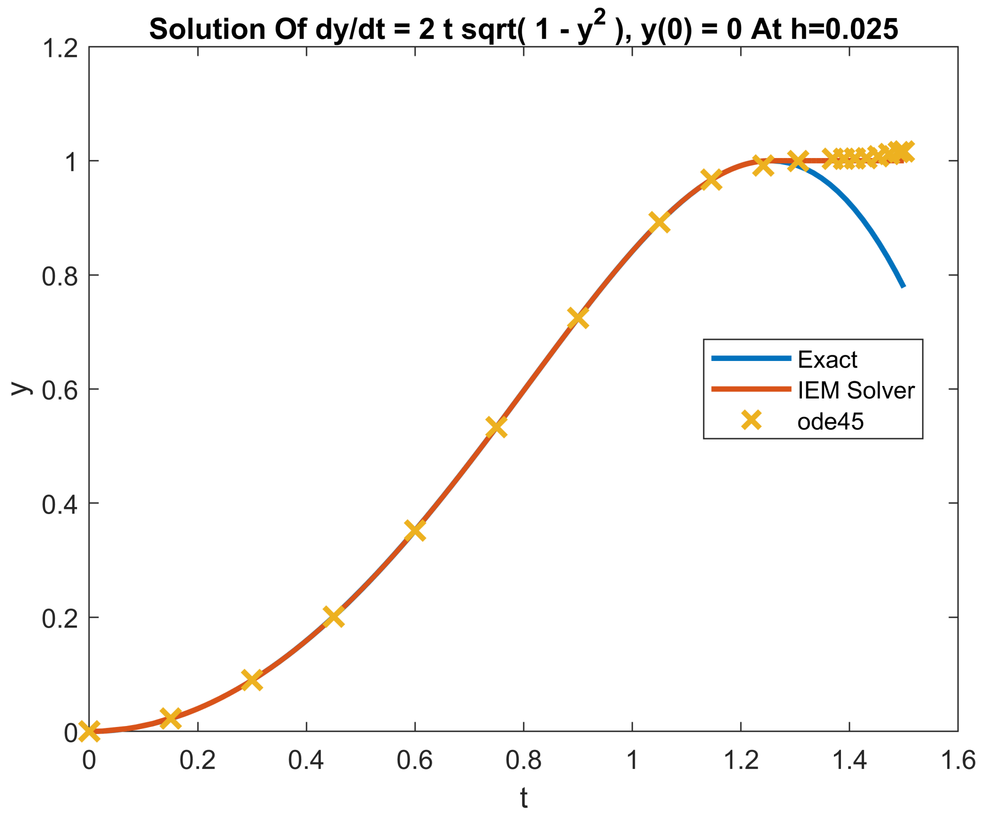
```
g = function_handle with value:
    @(t,y)2.*t*(1-y.^2).^(0.5)
```

```
t0 = 0;
```

```
tN = 1.5;
y0 = 0;
h = 0.025;
[adaptive_soln, tt_adaptive] = adaptiveIEM(g, t0, tN, y0, h);
ode45_soln = ode45(g, [t0, tN], y0);

%Visualizing and comparing the solution

% By using the method of integrating factor and multiplying the ODE in
% stanard form by mu(x) = cos(x), you get the solution shown in yy.

tt_estimate = linspace(t0, tN, length(soln));
tt = linspace(t0,tN,100);
yy = sin(tt.^2);

% Plot both on the same figure, plotting the approximation with x's
plot(tt, yy, tt_adaptive, adaptive_soln, ode45_soln.x, ode45_soln.y, 'x', 'MarkerSize',10, 'Lir
```

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

```

% Add a label to the axis and a legend
xlabel('t');
ylabel('y');
title('Solution Of dy/dt = 2 t sqrt( 1 - y^2 ), y(0) = 0 At h=0.025');
legend('Exact', 'IEM Solver', 'ode45', 'Location','Best');
```



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

```
% IEM and Adaptive IEM both use slopes to guide their approximations. Since
% the slope is zero at the point of divergence, it is very likely that the
% algorithm is oscillating around that point and not progressing forward.
% More specifically, at y = 1, the function i.e.
% the derivative = 0 so for the IEM,
% all the intermediate evaluations for the function
% become 0 so y doesnt change.
```