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: India Tory

Student Number: 1006944121

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`

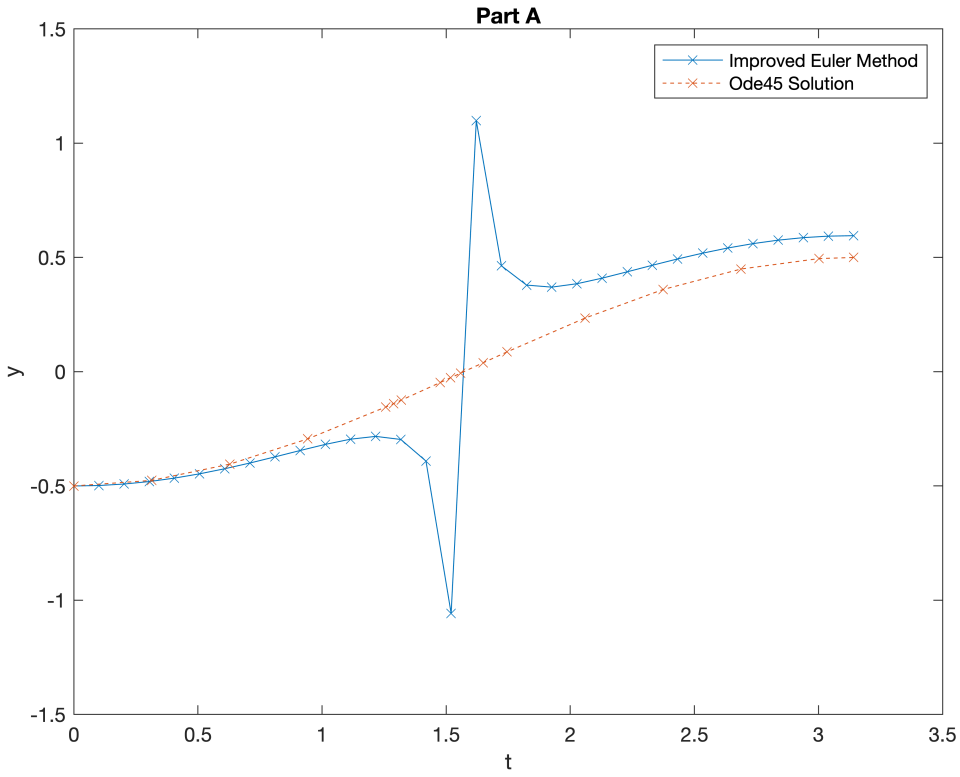
(d) y' = y^3 - t^2, y(0) = 1 from t=0 to t=1

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.

```
% constant step size
h = 0.1
```

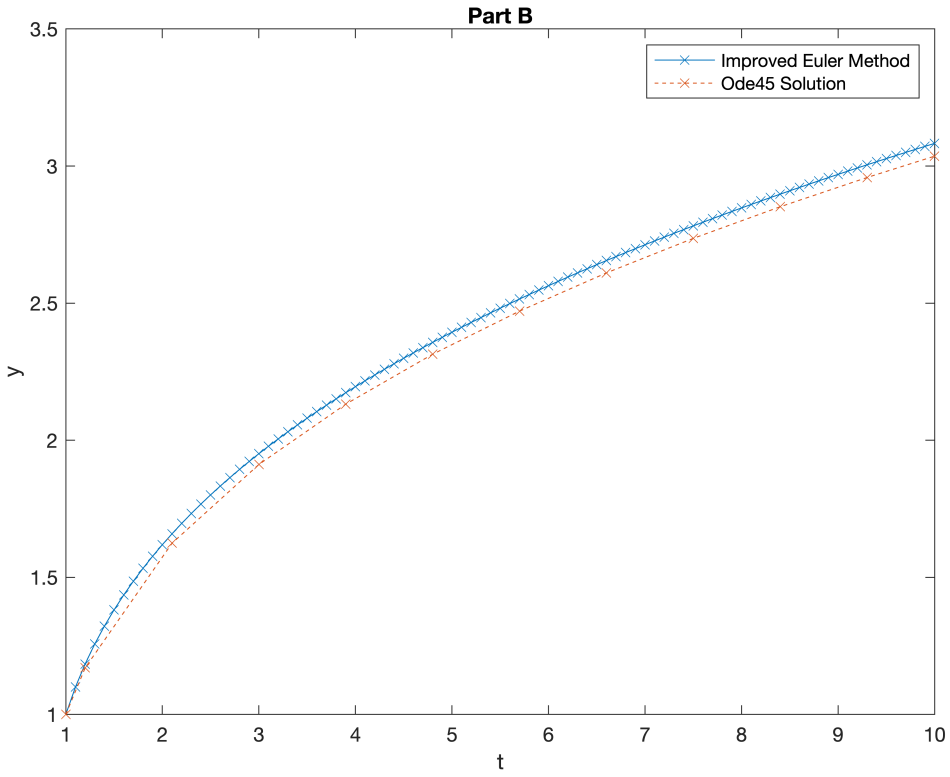
```
h = 0.1000
```

```

% A

% The two solutions are nearly identical, both starting around y = -0.5 when t = 0.
% They seem to continue along the same line, but the improved Euler method solution
% becomes very different at pi/2. The differential equation, written below this,
% includes tan(t) which has an asymptote when t = pi/2. The IEM is therefore pulled
% away from the actual solution at this point, while the actual solution remains on
% the same path. From about t = 2, the two return to the same slope but with seperatio
% between them in the y direction.

% CONDITIONS
% function: y' = y tan t + sin t, y(0) = -1/2
fa = @(t, y) y * tan(t) + sin(t);
ya = -1/2;
% range: t = 0 to pi
t0a = 0;
tNa = pi;

% Improved Eulers Method
[t, y] = impEuler(fa, t0a, tNa, ya, h);
plot(t, y, 'x-');
hold on;

% ODE Solver
solA = ode45(fa, [t0a, tNa], ya);
plot(solA.x, solA.y, 'x--');


title('Part A');
xlabel('t');
ylabel('y');
legend('Improved Euler Method', 'Ode45 Solution');
```



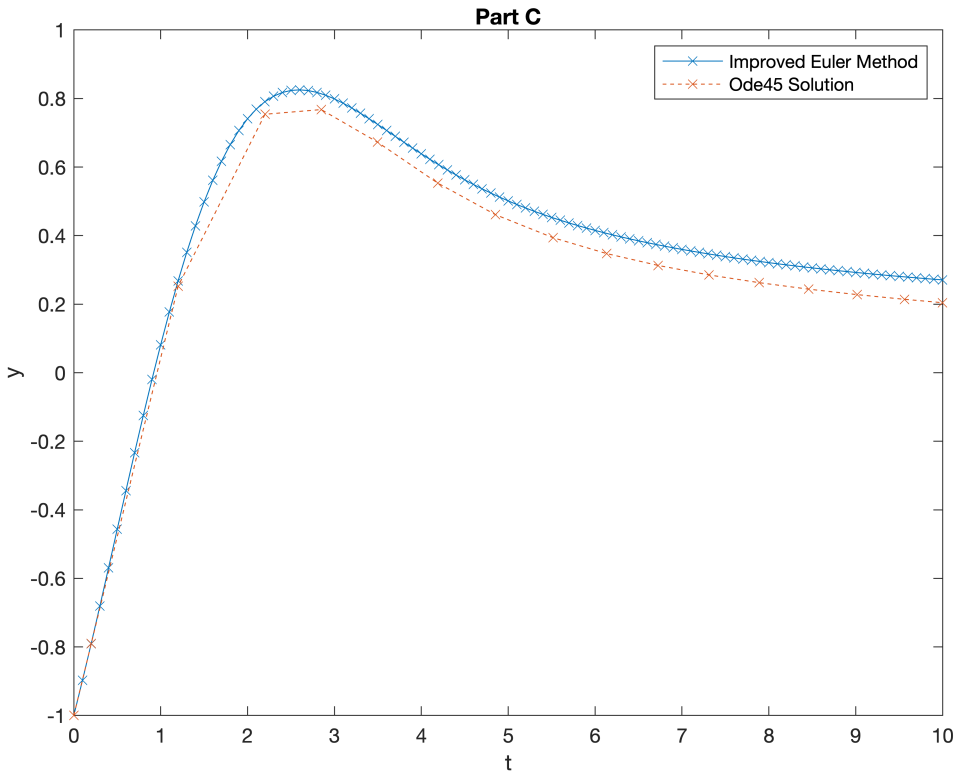

```
clf % clear for next problem

% B

% Both solutions shown on the "Part B" plot look very similar. They begin
% at the same point and seem to have very similar slopes. There appears to
% be a slight separation between the two plots: the IEM plot is slightly higher
% with the ode45 solution close below it.

% CONDITIONS
% function: y' = 1 / y^2 , y(1) = 1
fb = @(t, y) 1 / (y^2);
yb = 1;
% range: t = 1 to 10
tb0 = 1;
tbN = 10;

% Improved Eulers Method
[t, y] = impEuler(fb, tb0, tbN, yb, h);
plot(t, y, 'x-');

% ODE Solver
hold on;
solB = ode45(fb, [tb0, tbN], yb);
plot(solB.x, solB.y, 'x--');

title('Part B');
```

```
xlabel('t');
ylabel('y');
legend('Improved Euler Method', 'Ode45 Solution');
```




```
clf

% C

% From t = 0 to t = 1, the two plots appear to be identical. At t = 1, the
% ode45 solution appears to become a straight line while the IEM remains
% curved. From t = 3, the two appear to return to a very similar path.

% CONDITIONS
% function: y' =  1 - t y / 2, y(0) = -1
fC  = @(t, y) 1 - (t * y) / 2;
yC = -1;
% range: t= 0 to 10
tC0 = 0;
tCN = 10;

[t, y] = impEuler(fC, tC0, tCN, yC, h);
plot(t, y, 'x-');
hold on;

solC = ode45(fC, [tC0, tCN], yC);
plot(solC.x, solC.y, 'x--');
title('Part C');
xlabel('t');
```

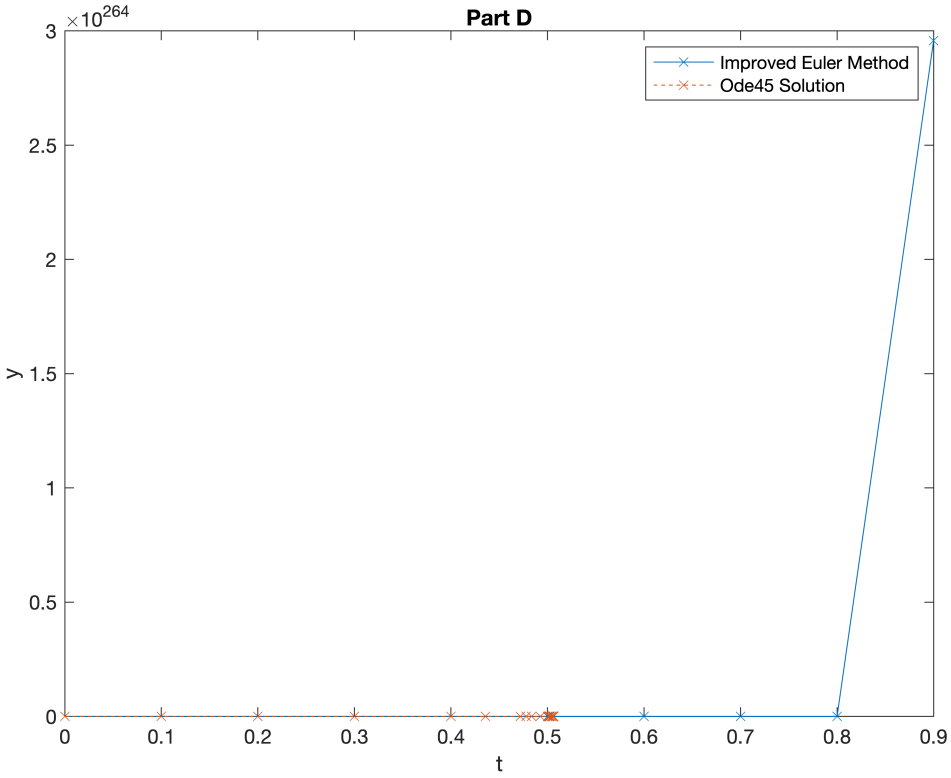
```
ylabel('y');
legend('Improved Euler Method', 'Ode45 Solution');
```




```
clf;

% D

% I recieved a warning on this plot. The ode45 function was unable to meet
% integration tolerances and could therefore not solve past t = 0.5. The
% IEM solution also appeared to go towards infinity around t = 0.8. Neither
% of these solutions seem feasible.

% CONDITIONS
% function: y' = y^3 - t^2, y(0) = 1
fD = @(t, y) y^3 - t^2;
yD = 1;

%range: t=0 to 1
tD0 = 0;
tDN = 1;

[t, y] = impEuler(fD, tD0, tDN, yD, h);
plot(t, y, 'x-');
hold on;

solD = ode45(fD, [tD0, tDN], yD);
```

```
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.
```

```
plot(solD.x, solD.y, 'x--');
title('Part D');
xlabel('t');
ylabel('y');
legend('Improved Euler Method', 'Ode45 Solution');
```




```
clf;
```

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`.

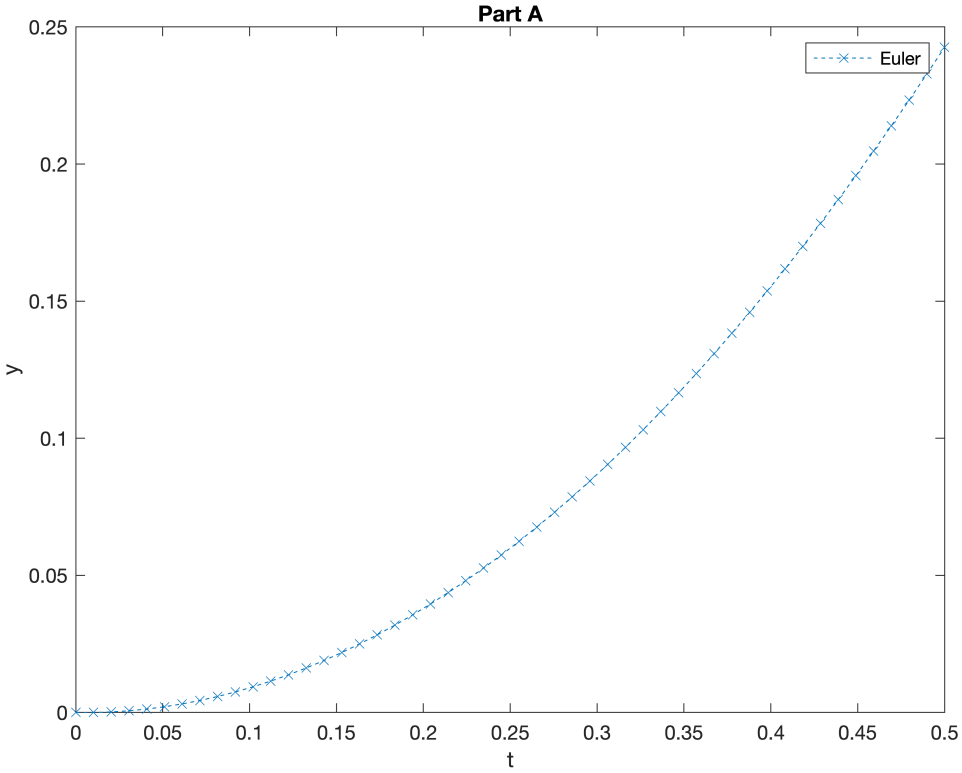
(b) Calculate the solution of the IVP and evaluate it at `t=0.5`.

(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
% y' = 2 t sqrt( 1 - y^2 ) , y(0) = 0, from t=0 to t=0.5
f3 = @(t,y) 2 * t * sqrt(1-y^2);
y0 = 0;
t0 = 0;
t1 = 0.5;

t = linspace(t0,t1,50);      % chosing to have 50 points
y = euler(f3,y0,t);           % solving for y using iode

% plot
plot(t, y, 'x--');
title('Part A');
xlabel('t');
ylabel('y');
legend('Euler');
```




```
% B
% y = sin(t^2) + sin(C)
% y(t) = sin(t^2)
t = 0.5;
yP = sin(t^2);  % use this equation
fprintf('Solution of IVP at %g:      y(%g) = %g\n', t, t, yP);
```

```
Solution of IVP at 0.5:      y(0.5) = 0.247404
```

```
% C
% from globalerror.pdf:
% variables to include:

% En <= ((1 + M) * Δt) / 2  * ( e^(M * Δt * n) - 1)
% En --> error of nth step
% n --> step number
% M --> upper boundary for f (see below)
% --> there exists an M >0 sothat|f| ≤ M,|∂tf| ≤ M, and|∂yf| ≤ M
% dt --> step length (Δt)




% D
% Defining variables from above
M = 2;
dt = 0.01;

% Error computation
t = t0:dt:t1;

y = euler(f3, y0, t);   % using iode code
fEx = @(t) sin(t.^2);   % exact solution

actualV = fEx(0.5);
eulerV = y(length(y));
actualEr = abs(actualV - eulerV);

% Computing error estimate
n = length(t) - 1;      % start at (- 1) because going (+1)
fEr = @(M, dt, n) ( (1+M) * dt * (exp (M*dt*n) - 1)) / 2;

compuEr = fEr(M, dt, n);

fprintf('dt = %g\n', dt);
```

```
dt = 0.01
```

```
fprintf('Computed Error: %g\n', compuEr);
```

```
Computed Error: 0.0257742
```

```
fprintf('Actual Error: %g\n', actualEr);
```

```
Actual Error: 0.0047319
```

```
% E
dt2 = 0.001;

% Error computation
t2 = t0:dt2:t1;
```

```
y2 = euler(f3, y0, t2);   % using iode code
fEx2 = @(t) sin(t.^2);   % exact solution

actualV2 = fEx2(0.5);
eulerV2 = y2(length(y2));
actualEr2 = abs(actualV2 - eulerV2);

% Computing error estimate
n2 = length(t2) - 1;      % start at (- 1) because going (+1)
fEr2 = @(M, dt2, n) ( (1+M) * dt2 * (exp (M*dt2*n2) - 1)) / 2;

compuEr2 = fEr(M, dt2, n2);

fprintf('dt2 = %g\n', dt2);
```

```
dt2 = 0.001
```

```
fprintf('Computed Error: %g\n', compuEr2);
```

```
Computed Error: 0.00257742
```

```
fprintf('Actual Error: %g\n', actualEr2);
```

```
Actual Error: 0.000472302
```

```
% The inital computed error was 0.0047319, while the new error after
% the change was 0.0004723. This confirms that Euler's method is first
% order, because when we decreased the step size by a factor of ten the
% resultant change in the error was also a decrease by a factor of ten.
```

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.

```
% The formula for updating the step size is attempting to reduce the step
% size if the error (D) is too large. If this condition is not considered,
% the solution might not be plausible (within the given tolerance range).
% The 0.9, seen in part c, is multiplied into the equation to ensure that
% the next step functions and is not too large.
```

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`.

(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
% Function
f = @(t,y) 2 * t * sqrt(1 - y^2);
% Parameters from description
t0 = 0;
t1 = 0.75;
y0 = 0;
h = 0.025;

% euler method and adaptive
t = t0:h:t1;

y = euler(f, y0, t);
plot(t, y, 'x--');
hold on;        % need to put all three solutions together

[t, y] = adaptiveEuler(f, t0, t1, y0, h);
plot(t, y);
hold on;        % still have one more solution to include

% COMMENTED OUT DUE TO ERROR
% exact = @(t) sin(t.^2); % equation
% [t,y] = exact(t1)
% plot(t, y, '--');


title('Part 5');
xlabel('t');
```

```
ylabel('y');
legend('Euler', 'Adaptive Euler'); % DID NOT INCLUDE EXACT DUE TO ERROR
```

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.

```
% (a)
% I could not see the answer due to the error above, but I assume that the
% Adaptive Euler Method is closer to the exact solution. I think this
% because the step size is constantly adjusted to ensure that it works for
% the given tolerance. This is not the case for the Euler method which has
% one step size throughout the whole process.

% (b)
f = @(t,y) 2 * t * sqrt(1 - y^2);
t0 = 0;
t1 = 1.5;
y0 = 0;
h = 0.025;

% SAME AS ABOVE
% fEx = @(t) sin(t.^2);
% [t, y] = fEx(t1)
% plot(t, y);
% hold on;

t = t0:h:t1;
y = euler(f, y0, t);
plot(t, y, 'x--');
```

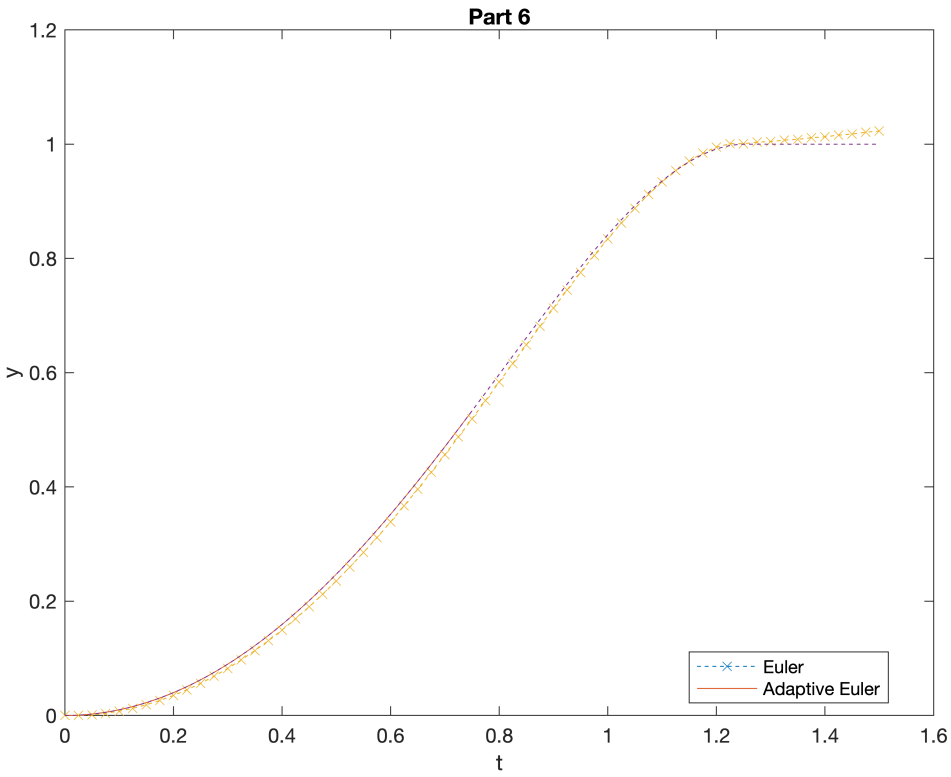
```
Warning: Imaginary parts of complex X and/or Y arguments ignored.
```

```
hold on;

[t, y] = adaptiveEuler(f, t0, t1, y0, h);
plot(t, y, '--');
```

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

```
title('Part 6');
xlabel('t');
```

```
ylabel('y');
legend('Euler', 'Adaptive Euler');
```




```
% (c)
% The exact solutions and approximations are the same until t = 1.
% At this point they begin to veer away from each other. This means that
% none of the methods are perfect for this solution.
```