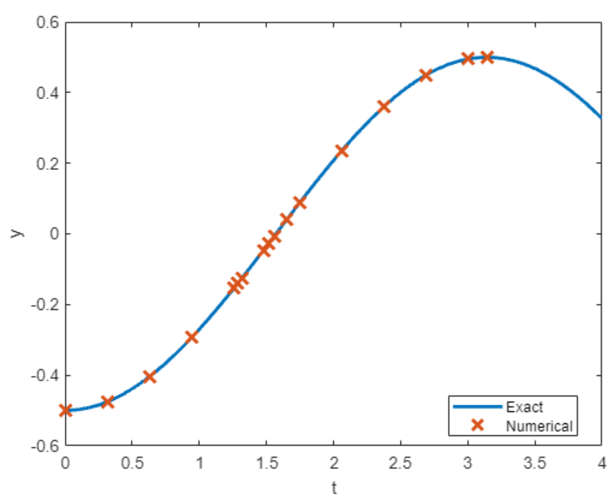
```
%% Integrator Lab: Solving First Order ODEs in MATLAB and Picard Approximation
%
% This lab will teach you to numerically solve first order ODEs using a
% built in MATLAB integrator, |ode45|.  |ode45| is a good, general purpose
% tool for integrating first order equations (and first order systems).  It
% is not always the right algorithm, but it is usually the right algorithm
% to try first. This lab will also teach you how to manipulate symbolic
% functions in MATLAB.
%
% You will learn how to use the |ode45| routine, how to interpolate between
% points, and how MATLAB handles data structures.  You will also learn how
% to use MATLAB for exact symbolic calculations and write your own Picard
% approximation code.
%
% Opening the m-file lab2.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 eight exercises in this lab that are to be handed in at the
% end of the lab.  Write your solutions in the template, including
% appropriate descriptions in each step. Save the .m file and submit it
% online using Quercus.
%
%% Student Information
%
% Student Name: Joaquin Arcilla
%
% Student Number: 1007930820
%
```

```
%% Exercise 1
%
% Objective: Solve an initial value problem and plot both the numerical
% approximation and the corresponding exact solution.
%
% Details: Solve the IVP
%
% |y' = y tan t + sin t, y(0) = -1/2|
%
% from |t = 0| to |t = pi|.
%
% Compute the exact solution (by hand), and plot both on the same figure
% for comparison, as above.
%
% Your submission should show the construction of the inline function, the
% use of ode45 to obtain the solution, a construction of the exact
% solution, and a plot showing both.  In the comments, include the exact
% solution.
```

```
%
% Label your axes and include a legend.

disp("Exercise 1: ")
```

```
Exercise 1:
```

```

% Numerical solution
%Function
func = @(t,y) y .* tan(t) + sin(t); %func is the derivative of y

% Initial conditions
t0 = 0;
y0 = -0.5;

% Range
t1 = pi;

% Solve:
sol1 = ode45(func, [t0, t1], y0);

% X and Y points to be graphed
num_x = sol1.x;
num_y = sol1.y;

% By hand solution
tt = linspace(0,4,50);
yy = ((sin(tt) .^ 2) ./ (2 .* cos(tt))) - (0.5 ./ cos(tt));

% Graphing
plot(tt, yy, num_x, num_y, 'x', 'MarkerSize',10, 'LineWidth', 2);
xlabel('t');
ylabel('y');
legend('Exact', 'Numerical','Location','Best');
```



```

```

```
%% Exercise 2
%
% Objective: Interpolate a solution at a number of grid points
%
% Details: For the solution you computed in exercise 1, use deval to
% compute the interpolated values at 10 grid points between 2 and 3.

disp("Exercise 2")
```

```
Exercise 2
```

```
xvals = linspace(2,3,10); %Create x values between 2 and 3
yvals = deval(sol1, xvals); %get the values
disp("Interpolated values are:")
```

```
Interpolated values are:
```

```
disp(yvals) %display
```

```
    0.2081    0.2572    0.3032    0.3454    0.3833    0.4166    0.4447    0.4673    0.4841    0.4950
```

```
%% Exercise 3
%
% Objective: Examine the error of a solution generated by |ode45|
%
% Details: For your solution to exercise 1, compute the pointwise error,
% identify the maximum value of the error, and visualize the error on a
% linear-log plot (use semilogy to plot the log of the error vs. t).
% Write in the comments where the error is largest, and give a brief
% (1-2 sentences) explanation of why it is largest there.  Make sure to
% label your axes.

disp("Exercise 3:")
```

```
Exercise 3:
```

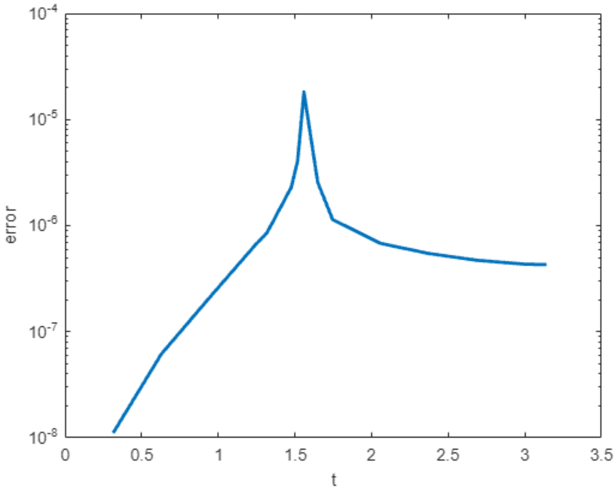
```
% Get analytical values:
yTrue = ((sin(sol1.x) .^ 2) ./ (2 .* cos(sol1.x))) - (0.5 ./ cos(sol1.x));

% Compute the pointwise error from exercise 1; note the use of MATLAB's vectorization
error = abs(yTrue - sol1.y);

% Display max error
[maxerror, index] = max(error);
fprintf('maximum error: %g \n', maxerror);
```

```
maximum error: 1.8068e-05
```

```
fprintf('happens at: %g \n', index);
```

```
happens at: 10
```

```
semilogy(sol1.x, error, 'LineWidth', 2);
xlabel('t');
ylabel('error');
```



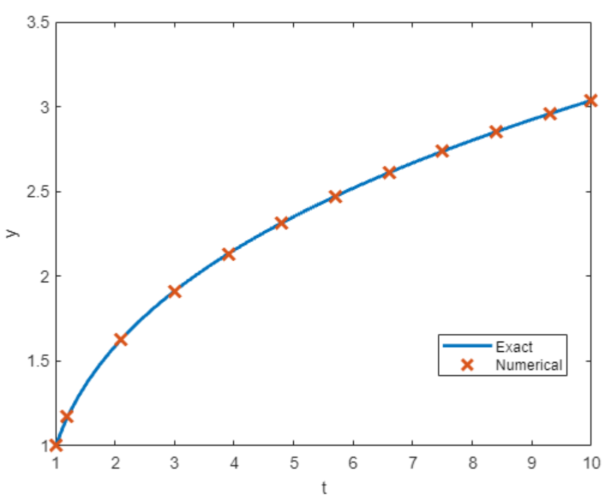

```
%Comments on graph
%Comparing this graph to the previous visualization of the ODE solution,
%the max error seems to have appeared where an inflection point happens in
%the y vs t graph. This inflection point may have been difficult to calculate
%As the points are suddenly not behaving as the previous points on the left
%were. Once the inflection point is passed however, the error lowers.
```

```
%% Exercise 4
%
% Objective: Solve and visualize a nonlinear ode using ode45
%
% Details: Solve the IVP
%
% |y' = 1 / y^2 , y(1) = 1|
%
% from |t=1| to |t=10| using |ode45|.  Find the exact solution and compute
% the maximum pointwise error.  Then plot the approximate solution and the
% exact solution on the same axes.
%
% Your solution should show the definition of the inline function,
% the computation of its solution in this interval, the computation of the
```

```
% exact solution at the computed grid points, the computation of the
% maximum error, and a plot of the exact and approximate solutions.
%Your axes should be appropriately labeled and include a legend.

disp("Exercise 4: ")
```

```
Exercise 4:
```

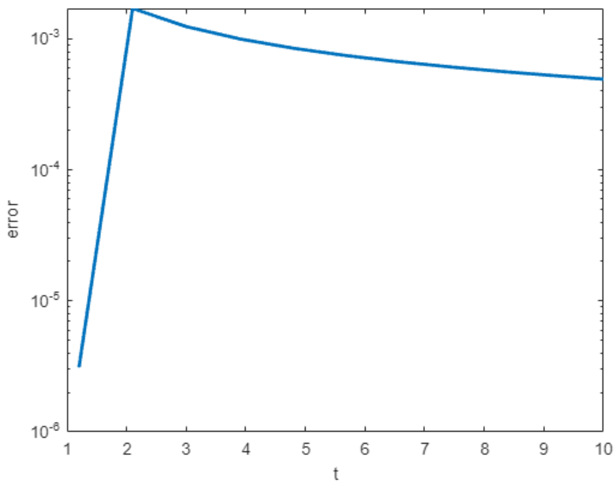
```

% Numerical solution
%Function
func4 = @(t,y) 1 ./ y.^2; %func is the derivative of y

% Initial conditions
t0 = 1;
y0 = 1;

% Range
t1 = 10;

% Solve:
sol4 = ode45(func4, [t0, t1], y0);

% By hand solution
tt = linspace(1,10,50);
realFunc4 = @(t) ((3 .* t) -2) .^ (1/3);

yy = realFunc4(tt);
errorY = realFunc4(sol4.x);

% Graphing
plot(tt, yy, sol4.x, sol4.y, 'x', 'MarkerSize',10, 'LineWidth', 2);
xlabel('t');
ylabel('y');
legend('Exact', 'Numerical','Location','Best');
```



```
% Error calculation
error = abs(errorY - sol4.y);
fprintf('maximum error: %g \n', max(error));
```

```
maximum error: 0.0017118
```

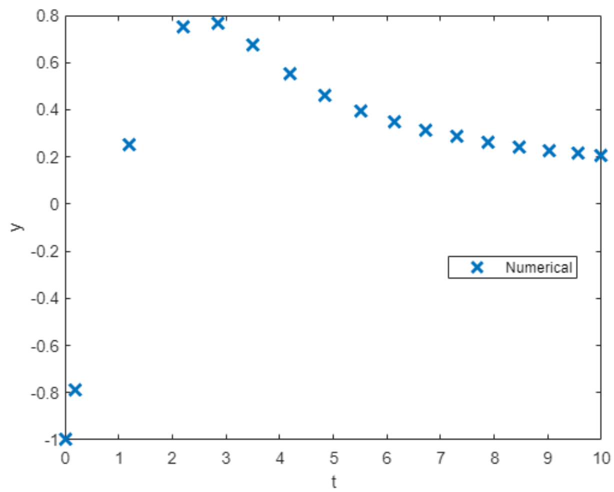
```
semilogy(sol4.x, error, 'LineWidth', 2);
xlabel('t');
ylabel('error');
```




```
%% Exercise 5
%
% Objective: Solve and visualize an ODE that cannot be solved by hand with
% |ode45|.
%
% Details: Solve the IVP
%
% |y' =  1 - t y / 2, y(0) = -1|
%
% from |t=0| to |t=10|.
%
% Your solution should show you defining the inline function, computing
% the solution in this interval, and plotting it.
%
% Your axes should be appropriately labeled

disp("Exercise 5:")
```

```
Exercise 5:
```

```
% Define function
func5 = @(t, y) 1 - (t .* y) ./ 2;

%Define initial conditions
t0 = 0;
y0 = -1;
t1 = 10;

%Solve:
sol5 = ode45(func5, [t0, t1], y0);

% Graphing
plot(sol5.x, sol5.y, 'x', 'MarkerSize',10, 'LineWidth', 2);
xlabel('t');
ylabel('y');
legend('Numerical','Location','Best');
```




```

```

```
%% Exercise 6 - When things go wrong
%
% Objective: Solve an ode and explain the warning message
%
% Details: Solve the IVP:
%
% |y' = y^3 - t^2, y(0) = 1|
%
% from |t=0| to |t=1|.
%
% Your solution should show you defining the inline function, and computing
% the solution in this interval.
%
```

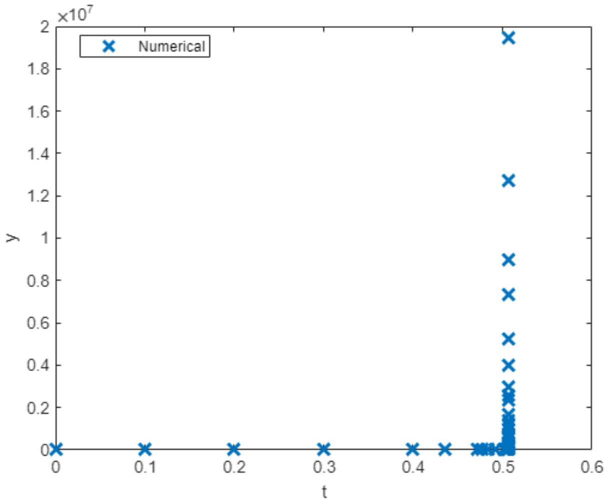
```
% If you try to plot the solution, you should find that the solution does
% not make it all the way to t = 1.
%
% In the comments explain why MATLAB generates the warning message that you
% may see, or fails to integrate all the way to t=1.  HINT: Try plotting
% the direction field for this with IODE.

disp("Exercise 6:")
```

```
Exercise 6:
```

```
% Define function
func6 = @(t, y) y .^3 - t .^2;

%Define initial conditions
t0 = 0;
y0 = 1;
t1 = 1;

%Solve:
sol6 = ode45(func6, [t0, t1], 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.
```

```
% Graphing
plot(sol6.x, sol6.y, 'x', 'MarkerSize',10, 'LineWidth', 2);
xlabel('t');
ylabel('y');
legend('Numerical','Location','Best');
```




```
%Based on the direction field in IODE, as t->infinity,
%The value of y climbs to infinity at an exponential rate.
%So the y value becomes too large for it to be graphed.
```

```
% I.e. y(1) = some insanely large number
```

```
%% Exercise 7
% Objective: Define a function using symbolic variables and manipulate it.
%
% Details: Define the function |f(x) = sin(x)cos(x)|
%
% Use MATLAB commands to obtain a simpler form of this function, compute
% value of this function for |x=pi/4| and |x=1|, and plot its graph.

disp('Exericse 7:')
```

```
Exericse 7:
```

```
% Declare symbolic variables
syms x

%Declare symbolic function
func7 = sin(x) * cos(x);

%Matlab commands
g = simplify(func7);
yVal1 = eval(subs(func7,x,(pi/4)));
yVal2 = eval(subs(func7,x,1));

%Display results
disp('Symbolic function is: ')
```

```
Symbolic function is:
```

```
func7
```

func7 = $\cos(x)\sin(x)$

```
disp('Simplified function is: ')
```

```
Simplified function is:
```

```
g
```

g =

$$\frac{\sin(2\,x)}{2}$$

```
disp('At x = π/4, y = ')
```

```
At x = π/4, y =
```

```
disp(yVal1)
```

```
    0.5000
```

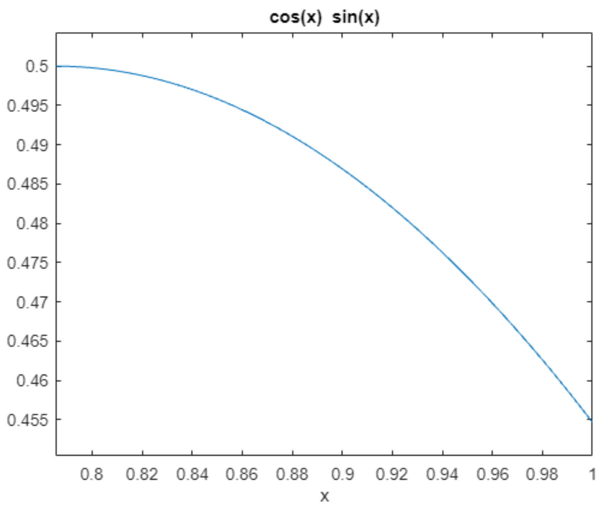
```
disp('At x = 1, y = ')
```

```
At x = 1, y =
```

```
disp(yVal2)
```

```
    0.4546
```

```
ezplot(func7, [pi/4, 1])
```

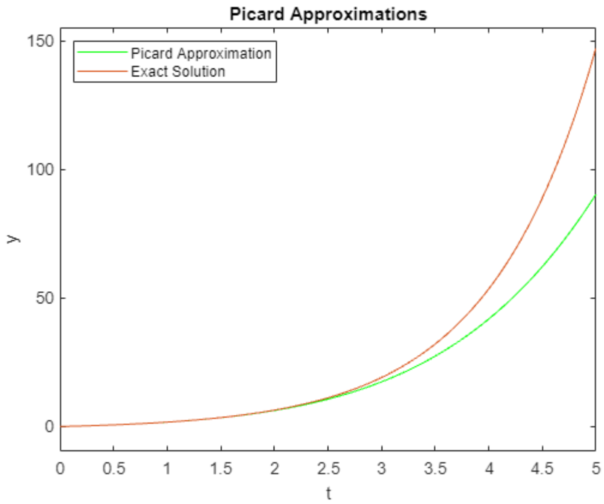



```
 
```

ans =

$$\frac{t\ (t^4 + 5\,t^3 + 20\,t^2 + 60\,t + 120)}{120}$$

```
hold off
```




```
%% Exercise 8
%
%  Objective: Solve your own Picard Approximation and compare it to the
%  exact solution.
%
%  Details: Consider the IVP
%      | y' = 1+y^2|
%      | y(0) = 1 |
%
% Find the Picard approximation phi_5.
% For better efficiency, do not keep all the previous approximations.

%
% Compute the exact solution (by hand), and plot both on the same figure
% for comparison, as above.
%
% Label your axes and include a legend.
%
% HINT. The initial condition has 1 instead of 0, so the Picard
% method needs to be adapted.

% Variables
syms t s y

%Function
func8 = 1 + y^2;

phis=[sym(1)]; %set initial value to 1

% Create loop for picard approximations

N=5;

for i = 1:N
```

```
    func8=subs(f,y,phis);   % Prepare integration with previous Phi
    func8=subs(func8,t,s);      % change t to s

    mynewphi = 1 + int(func8, s, 0 ,t);   % integrate to find next approximation

    phis=mynewphi;       % sets new phi
end

% Graph picard estimate
picard=ezplot(phis);
set(picard,'Color','blue');

% Real solution:

hold on;
exact=ezplot(tan(t+(pi/4)));
set(exact,'Color','red');

xlabel('t');
ylabel('y');
title('Picard Approximations');
legend('Picard Estimate', 'Exact Solution','Location','NorthEast');
```