Integrator Lab: Solving First Order ODEs in MATLAB and Picard Approximation

**Set up an inline function representation of an ODE and solve it**

**Exercise 1**

Objective: Solve an initial value problem and plot both the numerical approximation and the corresponding exact solution.

Details: Solve the IVP

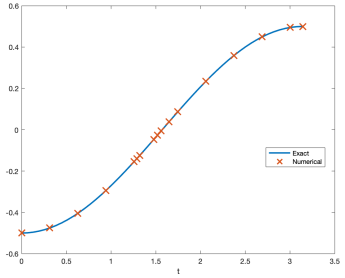
`y' = y tan t + sin t, y(0) = -1/2`

from `t = 0` to `t = pi`.

```
% Set up the right hand side of the ODE as an inline function
f = @(t,y) y*tan(t)+sin(t);

% The initial conditions
t0 = 0;
y0 = -1/2;

% The time we will integrate until
t1 = pi;

soln = ode45(f, [t0, t1], y0);
disp(soln); %to show pieces of data stucture computed from ode45
```

```
     solver: 'ode45'
    extdata: [1×1 struct]
          x: [0 0.3142 0.6283 0.9425 1.2566 1.2881 1.3195 1.4765 1.5177 1.5588 1.6491 1.7455 2.0597 2.3739 2.6880 3.0022 3.1416]
          y: [-0.5000 -0.4755 -0.4045 -0.2939 -0.1545 -0.1395 -0.1243 -0.0471 -0.0266 -0.0060 0.0391 0.0869 0.2348 0.3597 0.4494 0.4951 0.5000]
      stats: [1×1 struct]
      idata: [1×1 struct]
```

```
%Plotting the solution to visualize

% Construct the exact solution (solved by hand)
tt = linspace(0,pi,50);
yy = (-1/2).*cos(tt); %This is my exact solution

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

% Label to the axis and a legend
xlabel('t');
legend('Exact', 'Numerical','Location','Best');
```




**Computing an approximation at a specific point**

**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.

```
% Compute the solution at 10 grid points between 2 and 3:
tinterp = linspace(2, 3, 10)
```

```
tinterp = 1×10
```

```
    2.0000    2.1111    2.2222    2.3333    2.4444    2.5556    2.6667    2.7778    2.8889    3.0000
```

```
deval(soln, tinterp)
```

```
ans = 1×10
    0.2081    0.2572    0.3032    0.3454    0.3833    0.4166    0.4447    0.4673    0.4841    0.4950
```

**Errors, Step Sizes, and Tolerances**

**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.

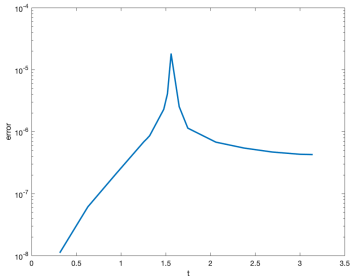
```
% Compute the exact solution
yexact = (-1/2).*cos(soln.x);

% Compute the pointwise error; note the use of MATLAB's vectorization
err = abs(yexact - soln.y);

disp(err);
```

```
   1.0e-04 *

  Columns 1 through 16

         0    0.0001    0.0006    0.0021    0.0070    0.0077    0.0087    0.0230    0.0408    0.1807    0.0254    0.0114    0.0068    0.0055    0.0047    0.0

  Column 17

    0.0043
```

```
fprintf('maximum error: %g \n', max(err));
```

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

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




```
%Maximum error value is 1.8068e-05
%Error is largest there because Matlab computes error as the
% difference between the different discretization parts when performing the integration.
% A largrer number of steps increases the error due to accumulation of rounding errors.
```

**Exercise 4**

Objective: Solve and visualize a nonlinear ode using ode45

Details: Solve the IVP

`y' = 1 / y^2 , y(1) = 1`

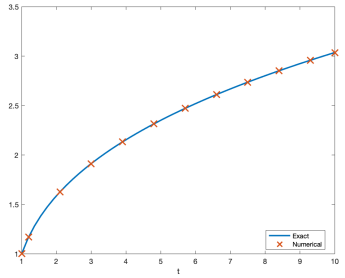
from `t=1` to `t=10` using `ode45`. Find the exact solution and compute the maximum pointwise error. Then plot the approximate solution and the exact solution on the same axes.

```
%inline function definiton
f = @(t,y) 1/(y^2);
```

```
% The initial conditions
t0 = 1;
y0 = 1;

% The time we will integrate until
t1 = 10;

soln = ode45(f, [t0, t1], y0);
disp(soln); %to show pieces of data stucture computed from ode45
```

```
     solver: 'ode45'
    extdata: [1×1 struct]
          x: [1 1.2010 2.1010 3.0010 3.9010 4.8010 5.7010 6.6010 7.5010 8.4010 9.3010 10]
          y: [1 1.1703 1.6248 1.9119 2.1319 2.3139 2.4711 2.6105 2.7364 2.8516 2.9583 3.0361]
      stats: [1×1 struct]
      idata: [1×1 struct]
```

```
%Plotting the solution to visualize

% exact solution (solved by hand)
tt = linspace(1,10,50);
yy = ((3.*tt)-2).^(1/3); %This is my exact solution

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

% Label to the axis and a legend
xlabel('t');
legend('Exact', 'Numerical','Location','Best');
```



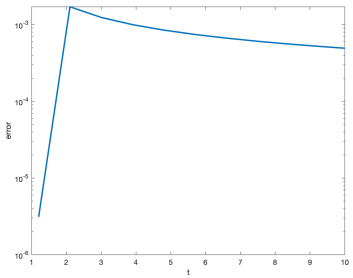

```
% Compute the exact solution
yexact =((3.*(soln.x))-2).^(1/3);

% Compute the pointwise error; note the use of MATLAB's vectorization
err = abs(yexact - soln.y);

disp(err);
```

```
         0    0.0000    0.0017    0.0012    0.0010    0.0008    0.0007    0.0007    0.0006    0.0006    0.0005    0.0005
```

```
fprintf('maximum error: %g \n', max(err));
```

```
maximum error: 0.0017118
```

```
semilogy(soln.x, err, '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`.

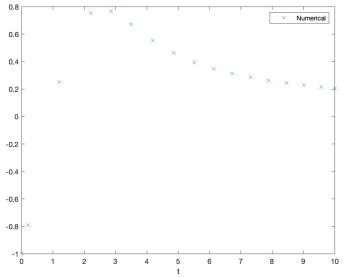
Your solution should show you defining the inline function, computing the solution in this interval, and plotting it.

Your axes should be appropriately labeled

```
%inline function definiton
f = @(t,y) 1-t.*y./2;

% The initial conditions
t0 = 0;
y0 = -1;

% The time we will integrate until
t1 = 10;

soln = ode45(f, [t0, t1], y0);
disp(soln); %to show pieces of data stucture computed from ode45
```

```
     solver: 'ode45'
    extdata: [1×1 struct]
          x: [0 0.2010 1.2010 2.2010 2.8484 3.4958 4.1908 4.8541 5.5174 6.1322 6.7254 7.3112 7.8898 8.4591 9.0172 9.5631 10]
          y: [-1 -0.7904 0.2527 0.7541 0.7677 0.6727 0.5530 0.4610 0.3939 0.3477 0.3130 0.2853 0.2627 0.2438 0.2278 0.2141 0.2043]
      stats: [1×1 struct]
      idata: [1×1 struct]
```

```
%Plotting the solution to visualize
plot(soln.x, soln.y, 'x');

% Label to the axis and a legend
xlabel('t');
legend('Numerical');
```




**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.

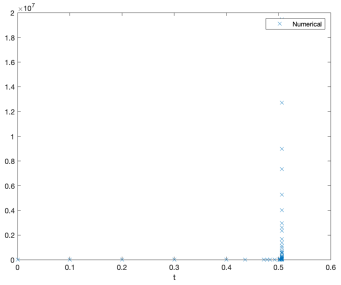
```
%inline function definiton
f = @(t,y) y.^3 -t.^2;

% The initial conditions
t0 = 0;
y0 = 1;

% The time we will integrate until
t1 = 1;

soln = ode45(f, [t0, t1], y0); %computing the solution in this interval
```

```
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.
```

```
disp(soln); %to show pieces of data stucture computed from ode45
```

```
      solver: 'ode45'
    extdata: [1×1 struct]
          x: [1×79 double]
          y: [1×79 double]
      stats: [1×1 struct]
      idata: [1×1 struct]
```

```
%Plotting the solution to visualize
plot(soln.x, soln.y, 'x');

% Label to the axis and a legend
xlabel('t');
legend('Numerical');
```




```
% The warning says: 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.

%MATLAB generated this warning message because the ode function cannot handle discontinuities
% that occur at that point in the interval. In this case, we would have to
% do a piecewise integration from t0 to the discontinuity and then from the
% discontinuity to t1
```

**Using symbolic variables to define functions**

**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.

```
% Start by defining the variables as symbolic
syms x y

% Define a function by simply writing its expression
f= sin(x)
```

$$f = \sin(x)$$

```
g = cos(x)
```

$$g = \cos(x)$$

```
h = sin(x)*cos(x)
```

$$h = \cos(x)\,\sin(x)$$

```
simplify(h) %simplify the function
```

ans =

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

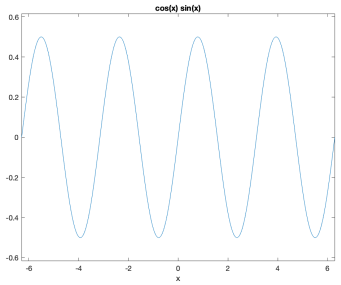
```
%compute value of this function for x=pi/4 and x=1
eval(subs(f,x,pi/4))
```

```
ans = 0.7071
```

```
eval(subs(f,x,1))
```

```
ans = 0.8415
```

```
%plot the function
ezplot(h)
```




**Obtaining Picard approximations**

**Exercise 8**

Objective: Solve your own Picard Approximation and compare it to the exact solution.

Details: Consider the IVP

$$\begin{cases} y' = 1+y^2 \\ y(0) = 1 \end{cases}$$

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.

```
clear all;
clf;
syms t s y;

% define the function f

f = 1+(y.^2); %symbolic function
```

```
% Initial approximation phi_0 = :

phi=[sym(1)];   % Keep a list with all the approximations

% Set up a loop to get successive approximations using Picard iterations

N=5;

for i = 1:N
    func=subs(f,y,phi(i));    % prepare function to integrate: y -> previous phi
    func=subs(func,t,s);      % variable of integration is s, so we need to change
                              % x -> s

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

    phi=cat(2,phi,[newphi]);       % update the list of approximations by adding new phi
end

% Show the last approximation

phi(N+1)
```

ans =

$$\frac{65536\,t^{31}}{109876902975}+\frac{32768\,t^{30}}{3544416225}+\frac{33636352\,t^{29}}{445414972275}+\frac{2189312\,t^{28}}{5119712325}+\frac{3771621376\,t^{27}}{2017062172125}+\frac{3496164352\,t^{26}}{522942044625}+\frac{1852466944\,t^{25}}{91423434375}+\frac{28632512\,t^{24}}{536350815}+\frac{234841821952\,t^{23}}{1884677169375}+\frac{14839805056\,t^{22}}{56729413125}+\frac{1241511104\,t^{21}}{2483105625}+\frac{400722032\,t^{20}}{456080625}+\frac{1157035136\,t^{19}}{808782975}+\frac{41}{1}$$

```
% Plot the approximation just found

picard=ezplot(phi(N+1),[0,5]);
set(picard,'Color','green');         % set the color of the graph to green

% In this case, the exact solution is
%
% y=[(t^3)/3]+t
%
% Compare the approximation and the exact solutions

hold on;
exact=ezplot([(t^3)/3]+t,[0,5]);

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