MATH 4073

HW#3

William Keely

1.)

a.) $Y_{i+1} = Y_i + hf(t_i, y_i)$, $Y(a) = \propto$ and $h = \frac{b-a}{N}$, $t \in [a,b]$.

$\Rightarrow Y_{i+1} = Y_i + h(y^2 + \frac{1}{t^2})$, $h = \frac{2-1}{N} = \frac{1}{N}$ and $1 \leq t \leq 2$. With $Y_i(1) = -0.5$

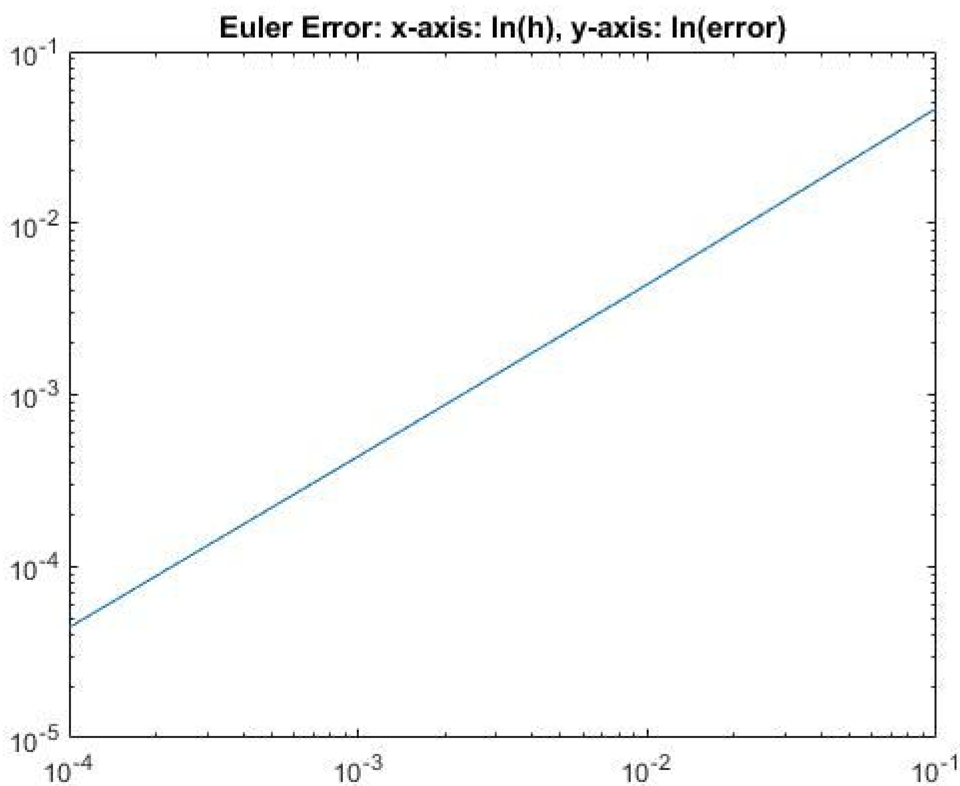
b.)

```
clear;
f = @(t,y)((y^2)+(1/(t^2)));
h = .1;
t0 = 1;
tn = 2;
y = -.5;
yexact = @(t)(1/(2*t))*(sqrt(3)*tan((sqrt(3)/2)*log(abs(t)))-1);

%iterative Euler Forward method.
for t = t0 : h : tn-h
   y = y + f(t,y)*h;
   t = t + h;
   fprintf('%f \t %f\t %f\n',t,y,yexact(t));
end

%Plots the valus of h, and the error in plot with ln(h) being the x-axis
%and ln(error) being the y-axis.
hvals = [.1, .01, .001, .0001];
errorvals = [0.046707, 0.004395, 0.000437, 0.000044];
loglog(hvals,errorvals)
title('Euler Error: x-axis: ln(h), y-axis: ln(error)')
```

c.)

h	**Apprx Y_n**	**Exact Y**	**Error**	**rate:(with ln(h) and ln(error))**
.1	0.093127	0.046420	0.046707	≈1
.01	0.050815	0.046420	0.004395	≈1
.001	0.046857	0.046420	0.000437	≈1
.0001	0.046464	0.046420	0.000044	≈1

d.) We can see that based on the rate column in our table that the slope of our line is ≈ 1. The when plotting the ln(h) and the ln(error), we see that it forms a line which is what we predicted it would be in class. We can note that for the smaller the h, the more accurate our method will be. Linear decay of error.



2.)

a.) $Y_{i+1} = Y_i + hf(t,y) + \frac{h^2}{2!} f^{(2)}(t,y)$

The first f(t,y) is already given: $(y^2 + \frac{1}{t^2})$. To find the "second derivative" we will use implicit differentiation:

$$\frac{d}{dt} f(t,y(t)) = 2y \frac{df}{dt} - \frac{2}{t^3} = 2y(y^2 + \frac{1}{t^2}) - \frac{2}{t^3} = 2y^3 - \frac{2y}{t^2} - \frac{2}{t^3}$$

$$\Rightarrow Y_{i+1} = Y_i + hf(t,y) + \frac{h^2}{2!} f^{(2)}(t,y) = Y_i + h(y^2 + \frac{1}{t^2}) + \frac{h^2}{2!}(2y^3 - \frac{2y}{t^2} - \frac{2}{t^3})$$

b.)

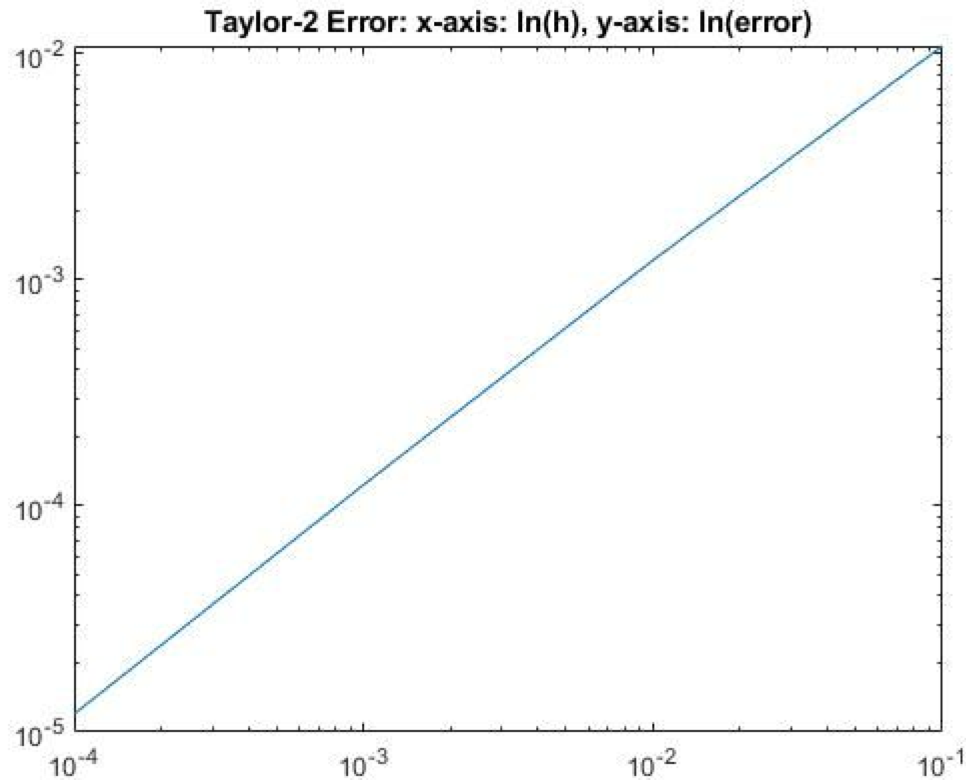
```
clear;
df = @(t,y)(2*(y.^3)-(2*y/t.^2)-(2/t));
f = @(t,y)((y.^2)+(1/(t.^2)));
h = .0001;
t0 = 1;
tn = 2;
y = -.5;
yexact = @(t)(1/(2*t))*(sqrt(3)*tan((sqrt(3)/2)*log(abs(t)))-1);

%iterative Euler Forward method.
for t = t0 : h : tn-h
   y = y + f(t,y)*h+df(t,y)*((h.^2)/2);
   t = t + h;
   fprintf('%f \t %f\t %f\n',t,y,yexact(t));

end

%Plots the valus of h, and the error in plot with ln(h) being the x-axis
%and ln(error) being the y-axis.
hvals = [.1, .01, .001, .0001];
errorvals = [0.010848, 0.001224, 0.000124, 0.000012];

loglog(hvals,errorvals)
title('Taylor-2 Error: x-axis: ln(h), y-axis: ln(error)')
```

c.)

h	**Apprx Y_n**	**Exact**	**Error**	**rate:(with ln(h) and ln(error))**
.1	0.035572	0.046420	0.010848	≈1
.01	0.045196	0.046420	0.001224	≈1
.001	0.046296	0.046420	0.000124	≈1
.0001	0.046408	0.046420	0.000012	≈1

d.)

We can see that based on the rate column in our table that the slope of our line is ≈ 1 which means we the error will decay linearly. The when plotting the ln(h) and the ln(error), we see that it forms a line which is what we predicted it would be in class. We can note that for the smaller the h, the more accurate our method will be. From our results we can see that the Taylor-2 method is much more accurate, even with just a step size of $\frac{1}{10}$.



3.)

a.)

To find our Taylor-3 method we just need to find $f^{(3)}(t,y(t))$.

$\Rightarrow f^{(3)} = 6y^2(2y^3 - 2y/t^2 - 2/t^3) - 4y/t^3 + 6/t^4$

b.)

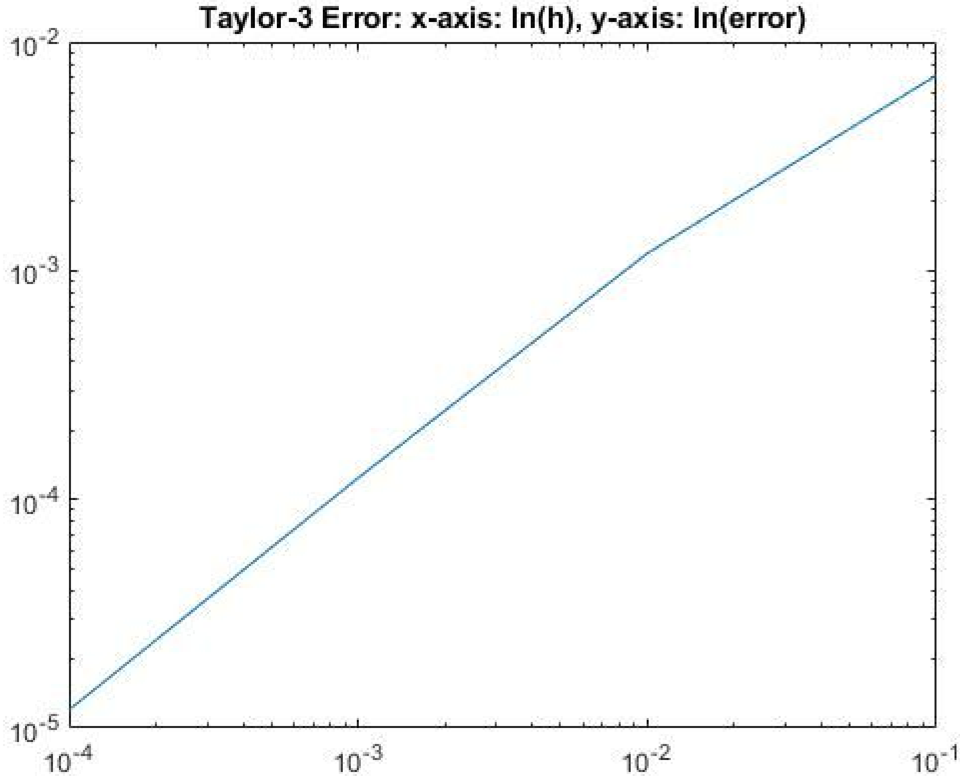
```
clear;
df = @(t,y)(2*(y.^3)-(2*y/t.^2)-(2/t));
f = @(t,y)((y.^2)+(1/(t.^2)));
h = .0001;
t0 = 1;
tn = 2;
y = -.5;
yexact = @(t)(1/(2*t))*(sqrt(3)*tan((sqrt(3)/2)*log(abs(t)))-1);
```

```
%iterative Euler Forward method.
for t = t0 : h : tn-h
   y = y + f(t,y)*h+df(t,y)*((h.^2)/2)+(6*(y.^2)*df(t,y)-(4*y/t.^3)+(6/t.^3))*((h.^3)/6);
   t = t + h;
   fprintf('%f \t %f\t %f\n',t,y,yexact(t));

end

%Plots the valus of h, and the error in plot with ln(h) being the x-axis
%and ln(error) being the y-axis.
hvals = [.1, .01, .001, .0001];
errorvals = [0.007153, 0.001190, 0.000124, 0.000012];

loglog(hvals,errorvals)
title('Taylor-3 Error: x-axis: ln(h), y-axis: ln(error)')
```

c.)

h	**Apprx Y_n**	**Exact**	**Error**	**rate:(with ln(h) and ln(error))**
.1	0.039267	0.046420	0.007153	≈.7
.01	0.045230	0.046420	0.001190	≈.7
.001	0.046296	0.046420	0.000124	≈1
.0001	0.046408	0.046420	0.000012	≈1

d.) We can see from the table that we have less error than the Taylor-2 method even for the larger step sizes. The similarity in the bottom two rows is probably due to round off by MATLAB. Without the round off error we still have a similar rate for our line (note: the slope is after ln() has been applied to h and the error.)



4.)

a.)

$K_1 = hf(t, y)$
$K_2 = hf(t+(h*.5), y+(K_1*.5))$
$K_3 = hf(t+(h*.5),y+(K_2*.5))$
$K_4 = hf(t+h, y+K_3)$

$\Rightarrow Y_{i+1} = Y_i + \frac{1}{6}(K_1+2K_2+2_3+K_4)$ where $f(t,y) = (y^2+\frac{1}{t^2})$.

b.)

```
clear;

f = @(t,y)((y.^2)+(1/(t.^2)));
h = .1;
t0 = 1;
```

```
tn = 2;
y = -.5;
yexact = @(t)(1/(2*t))*(sqrt(3)*tan((sqrt(3)/2)*log(abs(t)))-1);



%iterative Euler Forward method.
for t = t0 : h : tn-h
    %the k's
    k1 = h*f(t,y);
    k2 = h*f(t+(h*.5),y+(k1*.5));
    k3 = h*f(t+(h*.5),y+(k2*.5));
    k4 = h*f(t+h,y+k3);
    y = y + (1/6)*(k1 + (2*k2) + (2*k3) + k4);
    t = t + h;
    fprintf('%f \t %f\t %f\n',t,y,yexact(t));

end

%Plots the valus of h, and the error in plot with ln(h) being the x-axis
%and ln(error) being the y-axis.
hvals = [.1, .01, .001, .0001];
errorvals = [0.000002, 0.000002, 0.000002, 0.000002];

loglog(hvals,errorvals)
title('RK4 Error: x-axis: ln(h), y-axis: ln(error)')
```

c.)

h	**Apprx Y_n**	**Exact**	**Error**	**rate:(with <u>NO</u> ln(h) and ln(error))**
.1	0.046422	0.046420	0.000002	≈.000002
.01	0.046420	0.046420	0.000000	≈0.0

.001	0.046420	0.046420	0.000000	≈0.0
.0001	0.046420	0.046420	0.000000	≈0.0

d.) This method had some very interesting results. It was incredibly accurate when run in MATLAB with only minor error for h=.1, which is probably due to round off error. For all h greater than .1 the error was 0. Therefore we would probably need to look at more decimal places in order to see the change of effectiveness of using a smaller h. For the graph I set all values of the y-axis to ln(0.000002) so that we can see the behavior of the RK4 in MATLAB, even though based on our results in the table we should have the remaining 3 values be ln(0.000000), however ln(0) approaches -∞. This would not give us a nice graph. I tried running it with a step size of 0.5 and 0.05 and the 0.05 resulted in the exact answer. So we would need to try running it with a much larger value of h to see the change in error presented in the log plot.