Math 585 Homework 4
Due Soon
By Marvyn Bailly

Problem 1 *T1: Prove the convergence of the backward Euler method, that is, the global error $E_n = y_n - y(t_n)$ is $\mathcal{O}(h)$ as $h \to 0$ for all $n = 0, \dots, N$, $h = (b-a)/N$. Here $y(t_n)$ is the exact solution to $y'(t) = f(t, y)$ at $t_n = a + nh$, and y_n is the numerical solution obtained by the backward Euler method: $y_{n+1} = y_n + f(t_{n+1}, y_{n+1})$, with $y_0 = y(a)$ taking the exact initial condition. (Hint: You may follow the convergence proof of the forward Euler method I did in class. The inequality $1 - x \le e^{-x}, x \ge 0$ might be useful.)*

Solution.
Consider the Backward Euler method that is defined by

$$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1}),$$

and we wish to show that the global error $E_n = y_n - y(t_n)$ is $\mathcal{O}(h)$ as $h \to 0$ for all $n = 0, \dots, N$, $h = (b-a)/N$, where $h = (b-a)/N$, $y(t_n)$ is the exact solution, and y_n is the numerical solution obtained by Backward Euler with the initial condition $y_0 = y(a)$. First we define an auxiliary function to be

$$\bar{y}_{n+1} = y(t_n) + hf(t_{n+1}, y(t_{n+1})),$$

and observe that

$$y(t_{n+1}) - \bar{y}_{n+1} = y(t_{n+1}) - y(t_n) + hf(t_{n+1}, y(t_{n+1})) = hT_n,$$

where T_n is the local truncation error of Backward Euler at the nth step which is given by

$$T_n = \frac{y(t_{n+1}) - y(t_n)}{h} - f(t_{n+1}, y_{n+1}) = \frac{-y''(\xi_n)}{2}h,$$

where $\xi_n \in [t_n, t_{n+1}]$. Assuming that $M = \max_{a \le t \le b} |y''|$ gives that maximum truncation error over all steps is

$$T = \max_{0 \le n \le N} |E_n| = \frac{M}{2}h.$$

Using these facts, observe that

$$\begin{aligned}
|E_{n+1}| &= |y_{n+1} - y(t_{n+1})| \\
&= |y_{n+1} - \bar{y}_{n+1} + \bar{y}_{n+1} - y(t_{n+1})| \\
&\le |y_{n+1} - \bar{y}_{n+1}| + |\bar{y}_{n+1} - y(t_{n+1})| \\
&= |y_n + hf(t_{n+1}, y_{n+1}) - y(t_n) - hf(t_{n+1}, y(t_{n+1}))| + |\bar{y}_{n+1} - y(t_{n+1})| \\
&= |y_n - y(t_n)| + h|f(t_{n+1}, y_{n+1}) - f(t_{n+1}, y(t_{n+1}))| + |\bar{y}_{n+1} - y(t_{n+1})|.
\end{aligned}$$

Recall that f is known to be Lipschitz and thus $|f(t_{n+1}, y_{n+1}) - f(t_{n+1}, y(t_{n+1}))| \leq L|y_{n+1} - y(t_{n+1})|$. Then we have that

$$\begin{aligned}
|E_{n+1}| &= |y_n - y(t_n)| + hL|y_{n+1} - y(t_{n+1})| + |\bar{y}_{n+1} - y(t_{n+1})| \\
&\leq |E_n| + hL|E_{n+1}| + hT \\
\implies |E_{n+1}| &\leq \left(\frac{1}{1-hL}\right)(|E_n| + hT),
\end{aligned}$$

where we restrict $hL < 1$. Repeating this process for $E_{n-1}, \ldots, E_0$ gives

$$\begin{aligned}
|E_{n+1}| &\leq \left(\frac{1}{1-hL}\right)(|E_n| + hT) \\
&\leq \left(\frac{1}{1-hL}\right)\left(\left(\frac{1}{1-hL}\right)(|E_{n-1}| + hT) + hT\right) \\
&= \left(\frac{1}{1-hL}\right)^2 |E_{n-1}| + \left(\frac{1}{1-hL}\right)^2 hT + \left(\frac{1}{1-hL}\right) hT \\
&= \left(\frac{1}{1-hL}\right)^2 (|E_{n-1}| + hT(1 + (1-hL))) \\
&\vdots \\
&= \left(\frac{1}{1-hL}\right)^{n+1} (|E_0| + hT(1 + (1-hL) + \cdots + (1-hL)^n)).
\end{aligned}$$

Noticing that we have a partial geometric sum of the form $\sum_{i=0}^{n}(1-hL)^i = \frac{1-(1-hL)^{n+1}}{hL}$ and re-indexing reduces the inequality to

$$\begin{aligned}
|E_n| &\leq \left(\frac{1}{1-hL}\right)^n \left(|E_0| + \frac{T}{L}(1-hL)^n\right) \\
&= \left(\frac{1}{1-hL}\right)^n \left(\frac{T}{L}(1-hL)^n\right) \\
&= \frac{T}{L}\left(\left(\frac{1}{1-hL}\right)^n - 1\right)
\end{aligned}$$

where $|E_0| = 0$ as we know $y_0 = y(a)$ is the exact initial condition. Now we wish to bound $(1-hL)^{-n}$. Note that since $n = 0, \ldots, N$ and $hL < 1$, we have

$$\begin{aligned}
|E_n| &\leq \frac{T}{L}\big((1-hL)^{-n} - 1\big) \\
&\leq \frac{T}{L}\Big((1-hL)^{-N} - 1\Big) \\
&= \frac{T}{L}\Big((1-hL)^{\frac{-(b-a)L}{hL}} - 1\Big)
\end{aligned}$$

$$= \frac{T}{L}\left(\left((1-hL)^{-\frac{1}{hL}}\right)^{(b-a)L} - 1\right)$$

All that is left to do is bound $(1-hL)^{-\frac{1}{hL}}$. Let's consider the function $g(x) = (1-x)^{-1/x}$. We know that $g(x)$ is strictly increasing on $(0,1)$ and blows up at $x = 1$. Since we restricted $x = hL < 1$, we have that $x < \frac{1}{1+\epsilon} \leq 1$ for some $\epsilon \geq 0$. Thus $g(x)$ is bounded on $(0,1)$ by $g(\frac{1}{1+\epsilon}) = (1 - \frac{1}{1+\epsilon})^{-\frac{1}{1+\epsilon}}$. Using this fact, we get that

$$\begin{aligned} |E_n| &\leq \frac{T}{L}\left(\left(1 - \frac{1}{1+\epsilon}\right)^{-\frac{(b-a)L}{1+\epsilon}} - 1\right) \\ &= \frac{Mh}{2L}\left(\left(1 - \frac{1}{1+\epsilon}\right)^{-\frac{(b-a)L}{1+\epsilon}} - 1\right) = \mathcal{O}(h). \end{aligned}$$

Therefore the convergence of the backward Euler method is $\mathcal{O}(h)$. $\square$

Problem 2 *Derive the two-step Adams-Moulton formula*

$$y_{n+1} = y_n + \frac{h}{12}(5f(t_{n+1}, y_{n+1}) + 8f(t_n, y_n) - f(t_{n-1}, y_{n-1})).$$

Solution.

We wish to derive the two-step Adams-Moulton formula. Recall that the Adam's family is given by

$$y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} f(t, y(t))dt = y(t_n) + \int_{t_n}^{t_{n+1}} p(t)dt,$$

where $f(t, y(t))$ is approximated by an interpolating polynomial $p(t)$. Since we wish to derive the two-step Adams-Moulton, which is an implicit multistep method, we use the points $t_{n+1}, t_n,$ and t_{n-1} corresponding to $f_{n+1}, f_n,$ and f_{n-1} respectively. Then we have the interpolating polynomial to be

$$\begin{aligned}
p_3(t) &= f_{n-1} + \frac{f_n - f_{n-1}}{h}(t - t_n) + \left(\frac{\frac{f_{n+1}-f_n}{h} - \frac{f_n - f_{n-1}}{h}}{2h}\right)(t - t_{n-1})(t - t_n)\\
&= f_{n-1} + \frac{f_n - f_{n-1}}{h}(t - t_n) + \left(\frac{f_{n+1} - 2f_n + f_{n-1}}{2h^2}\right)\big(t^2 - tt_n - tt_{n-1} + t_{n-1}t_n\big)\\
&= f_{n-1} + \frac{f_n - f_{n-1}}{h}(t - t_n) + A\big(t^2 - tt_n - tt_{n-1} + t_{n-1}t_n\big).
\end{aligned}$$

Then we have that

$$y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} p_3(t)dt,$$

and all that is left to do is compute the integral. Observe that

$$\begin{aligned}
\int_{t_n}^{t_{n+1}} p_3(t)dt &= \int_{t_n}^{t_{n+1}} f_{n-1} + \frac{f_n - f_{n-1}}{h}(t - t_n) + A\big(t^2 - tt_n - tt_{n-1} + t_{n-1}t_n\big)\mathrm{d}t\\
&= \frac{h}{2}(3f_n - f_{n-1}) + \int_{t_n}^{t_{n+1}} At^2 - Att_n - Att_{n-1} + At_{n-1}t_n\mathrm{d}t\\
&= \frac{h}{2}(3f_n - f_{n-1}) + \left[\frac{At^3}{3} - \frac{At^2}{2}t_n - \frac{At^2}{2}t_{n-1} + Att_nt_{n-1}\right]_{t_n}^{t_{n+1}}\\
&= \frac{h}{2}(3f_n - f_{n-1}) + \frac{A}{6}\left[2t_{n+1}^3 - 3t_{n-1}t_{n+1}^2 - et_nt_{n+1}^2 + 6t_{n-1}t_ny_{n+1} + t_n^3 - 3t_{n-1}t_n^2\right]\\
&= \frac{h}{2}(3f_n - f_{n-1}) + \frac{A}{6}\left[(-3t_{n-1} + t_n + 2t_{n+1})(t_n - t_{n+1})^2\right]\\
&= \frac{h}{2}(3f_n - f_{n-1}) + \frac{A}{6}\left[5h \cdot h^2\right]\\
&= \frac{h}{2}(3f_n - f_{n-1}) + \left(\frac{f_{n+1} - 2f_n + f_{n-1}}{12h^2}\right)(5h^3)\\
&= \frac{h}{2}(3f_n - f_{n-1}) + \frac{h}{12}(f_{n+1} - 2f_n + f_{n-1})
\end{aligned}$$

$$= \frac{h}{12}(5f(t_{n+1}, y_{n+1}) + 8f(t_n, y_n) - f(t_{n-1}, y_{n-1})).$$

Therefore we have found the two-step Adams-Moulton formula to be

$$\begin{aligned} y(t_{n+1}) &= y(t_n) + \int_{t_n}^{t_{n+1}} p_3(t)dt \\ &= y(t_n) + \frac{h}{12}(5f(t_{n+1}, y_{n+1}) + 8f(t_n, y_n) - f(t_{n-1}, y_{n-1})) \end{aligned}$$

□

Problem 3 *Show that the local truncation error of the Adams-Moulton method derived above is $\mathcal{O}(h^3)$.*

Solution.
Recall the two-step Adams-Moulton formula is given by

$$y_{n+1} = y_n + \frac{h}{12}(5f(t_{n+1}, y_{n+1}) + 8f(t_n, y_n) - f(t_{n-1}, y_{n-1})),$$

and has local truncation error of

$$T_n = \frac{1}{h}(y(t_{n+1}) - y(t_n)) - \frac{1}{12}(5y'(t_{n+1}) + 8y'(t_n) - y'(t_{n-1})).$$

Observe the following Taylor expansions

$$\begin{aligned} y(t_{n+1}) &= y(t_n) + hy'(t_n) + \frac{h^2}{2}y''(t_n) + \frac{h^3}{6}y''' + \mathcal{O}(h^4), \\ y(t_{n-1}) &= y(t_n) - hy'(t_n) + \frac{h^2}{2}y''(t_n) - \frac{h^3}{6}y''' + \mathcal{O}(h^4), \\ y'(t_{n+1})' &= y'(t_n) + hy''(t_n) + \frac{h^2}{2}y'''(t_n) + \mathcal{O}(h^3), \\ y'(t_{n-1})' &= y'(t_n) - hy''(t_n) + \frac{h^2}{2}y'''(t_n) + \mathcal{O}(h^3). \end{aligned}$$

Then plugging the expansions into the local truncation error yields

$$\begin{aligned} T_n &= \frac{1}{h}\left(\left(y(t_n) + hy'(t_n) + \frac{h^2}{2}y''(t_n) + \frac{h^3}{6}y''' + \mathcal{O}(h^4)\right) - y(t_n)\right) \\ &- \frac{1}{12}\left(5\left(y'(t_n) + hy''(t_n) + \frac{h^2}{2}y'''(t_n) + \mathcal{O}(h^3)\right) + 8y'(t_n)\right. \\ &\left.-\left(y'(t_n) - hy''(t_n) + \frac{h^2}{2}y'''(t_n) + \mathcal{O}(h^3)\right)\right) \\ &= \left(y'(t_n) + \frac{1}{2}hy''(t_n) + \frac{1}{6}h^2y'' + \mathcal{O}(h^3)\right) - \frac{1}{12}\big(12y'(t_n) + 6hy''(t_n) + 2h^2y''' + \mathcal{O}(h^3)\big) \\ &= \mathcal{O}(h^3). \end{aligned}$$

Thus we have shown that the local truncation error of the two-step Adams-Moulton formula is $\mathcal{O}(h^3)$.

□

Problem 4 *Show that the implicit midpoint method given below is A-stable.*

$$y_{n+1} = y_n + hf(t_{n+1/2}, y_{n+1/2}), \quad t_{n+1/2} = t_n + h/2, \quad y_{n+1/2} = \frac{y_n + y_{n+1}}{2}.$$

Solution.
We wish to show that the implicit midpoint method

$$y_{n+1} = y_n + hf(t_{n+1/2}, y_{n+1/2}), \quad t_{n+1/2} = t_n + h/2, \quad y_{n+1/2} = \frac{y_n + y_{n+1}}{2},$$

is A-stable. To do so, let's apply the method to the test problem

$$y' = \lambda y,$$

which gives

$$y_{n+1} = y_n + h\lambda y_{n+1/2} = y_n + h\lambda\left(\frac{y_n + y_{n+1}}{2}\right).$$

Now solving for y_{n+1} gives

$$\begin{aligned} y_{n+1} &= y_n + \frac{h}{2}\lambda y_n + \frac{h}{2}\lambda y_{n+1} \\ y_{n+1} - \frac{h}{2}\lambda y_{n+1} &= y_n + \frac{h}{2}\lambda y_n \\ \implies y_{n+1} &= \frac{y_n + \frac{h}{2}\lambda y_n}{1 - \frac{1}{2}h\lambda}. \end{aligned}$$

Now if we let $z = \lambda h = a + ib$ and look at the absolute stability of the method yields

$$\begin{aligned} \left|\frac{2+z}{2-z}\right| &\leq 1 \\ \left|\frac{2+a+ib}{2-a-ib}\right| &\leq 1 \\ |2+a+ib|^2 &\leq |2-a-ib|^2 \\ (2+a)^2 + b^2 &\leq (2-a)^2 + b^2 \\ (2+a)^2 &\leq (2-a)^2 \\ 4 + 4a + a^2 &\leq 4 - 4a + a^2 \\ 8a &\leq 0 \\ a &\leq 0, \\ \implies \mathrm{Re}(z) &\leq 0. \end{aligned}$$

Thus we have that the region of absolute stability is $\mathrm{Re}(z) \leq 0$ which means that the implicit midpoint method is A-stable.

$\square$

Problem 5 *C1: Solve the initial value problem*

$$y'(t) = -\frac{1}{t^2} - \frac{y}{t} - y^2, \quad 1 \le t \le 2, \quad y(1) = -1,$$

(a) *using the forward Euler method;*

(b) *using the improved Euler method;*

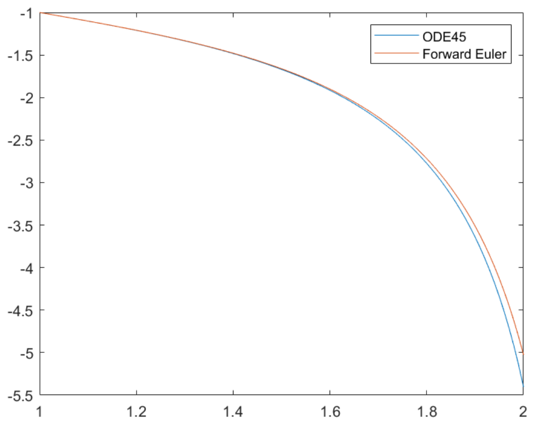
(c) *using the classical fourth-order Runge-Kutta method.*

Choose appropriate time steps h and demonstrate the convergence order for each method. You may use MATLAB `ode45` *to generate an "accurate" solution for reference.*

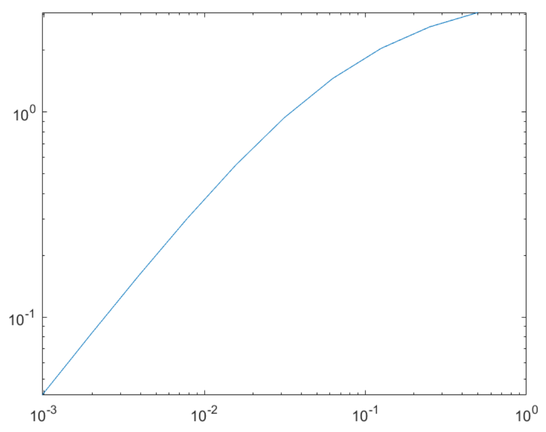
Solution.
Consider the initial value problem

$$y'(t) = -\frac{1}{t^2} - \frac{y}{t} - y^2, \quad 1 \le t \le 2, \quad y(1) = -1.$$

The MATLAB script in Listings 1 was used to run the entire problem.



(a) Forward Euler compared to exact solution.

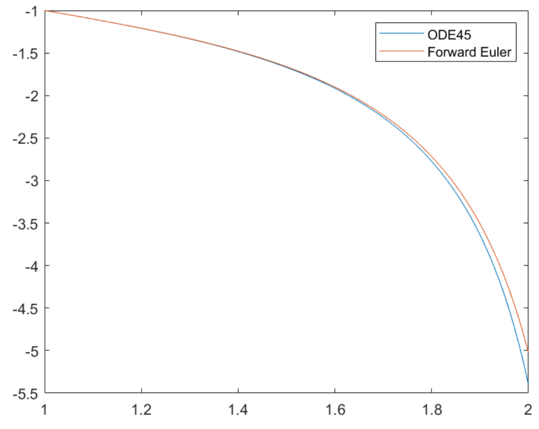


(b) Max error of Forward Euler Method.

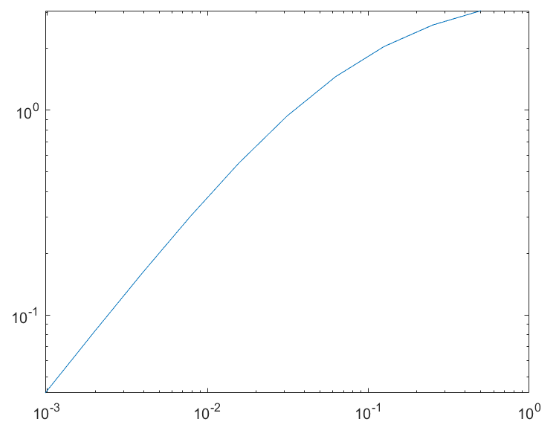
Figure 1: Plots used to analyze the convergence of the Forward Euler Method.

(a) First we wish to find the solution using the forward Euler method. I created the code in Listings 2 to preform the forward Euler method and picked $h = 0.01$ to be an appropriate h value. Using MATLAB's `ode45` function to find the exact solution of the initial value problem, we can plot the approximate solution from forward Euler compared to the exact solution. In figure 1a, we see that the Forward Euler does not give the best approximation. To study the convergence, we tested h values from $2^-1, \ldots, 2^{-10}$ and computed the error by using the infinity norm on the difference between the approximate solution and the exact solution. Plotting the max error of each h value on a loglog plot, as seen in figure 1b, we see that the error linearly reduces with h. Using MATLAB's `polyfit` function, we can find that slope of the line is around 1 which matches the expected convergence of Forward Euler method $\mathcal{O}(h)$.

(b) Next we wish to find the solution using the improved Euler method. I created the code in Listings 3 to preform the improved Euler method and picked $h = 0.01$ to be an appropriate h value. Using MATLAB's `ode45` function to find the exact solution of the initial value problem, we can plot the approximate solution from Improved Euler compared to the exact solution. In figure 2a, we see that the Improved Euler solution matches the exact solution closely. To study the convergence, we tested h values from $2^-1, \ldots, 2^{-10}$ and computed the error by using the infinity norm on the difference between the approximate solution and the exact solution. Plotting the max error of each h value on a loglog plot, as seen in figure 2b, we see that the error linearly reduces with h. Using MATLAB's `polyfit` function, we can find that slope of the line is around 2 which matches the expected convergence of Improved Euler method $\mathcal{O}(h^2)$.



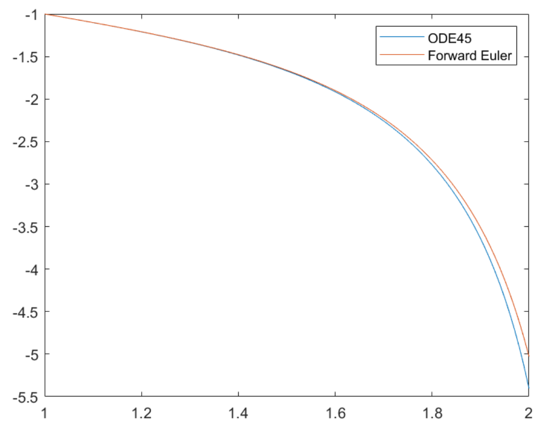
(a) Improved Euler compared to exact solution.



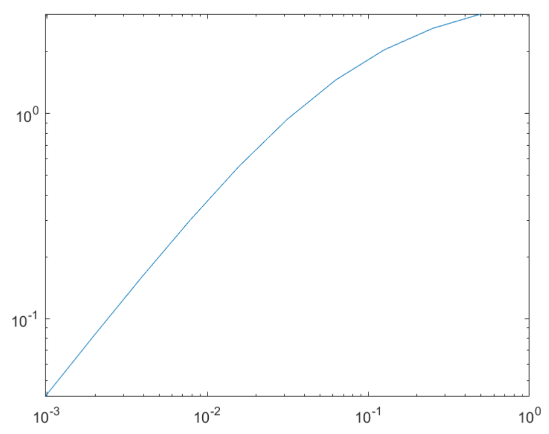
(b) Max error of Improved Euler Method.

Figure 2: Plots used to analyze the convergence of the Improved Euler Method.

(c) Next we wish to find the solution using the classical fourth-order Runge-Kutta method. I created the code in Listings 4 to preform the improved Euler method and picked $h = 0.01$ to be an appropriate h value. Using MATLAB's `ode45` function to find the exact solution of the initial value problem, we can plot the approximate solution from the fourth-order Runge-Kutta compared to the exact solution. In figure 3a, we see that the fourth-order Runge-Kutta matches the exact solution closely. To study the convergence, we tested h values from $2^-1, \ldots, 2^{-7}$ and computed the error by using the infinity norm on the difference between the approximate solution and the exact solution. Plotting the max error of each h value on a loglog plot, as seen in figure 3b, we see that the error linearly reduces with h. Using MATLAB's `polyfit` function, we can find that slope of the line is around 4 which matches the expected convergence of Forward Euler method $\mathcal{O}(h^4)$.



(a) Runge-Kutta compared to exact solution.



(b) Max error of Runge-Kutta Method.

Figure 3: Plots used to analyze the convergence of the Runge-Kutta Method.

Listing 1: **Main Script for C1**

```
clear All
format long

%setting up things
syms f(y,t);
f(y,t) = - 1/t^2 - y/t - y^2;
t0 = 1;
tf = 2;
y0 = -1;
h = 0.01;

%find exact solution
opts = odeset('RelTol',1e-12,'AbsTol',1e-12);
exact = ode45(@(t,y)- 1/t^2 - y/t - y^2,[t0,tf],y0,opts);

%% part A
```

```
[fet,fey] = forwardEuler(f,h,t0,tf,y0);
%plot approx with exact
figure(1)
plot(fet,deval(exact,fet));
hold on
plot(fet,fey);
legend("ODE45","Forward Euler");
hold off

%part B
[iet,iey] = improvedEuler(f,h,t0,tf,y0);
%plot approx with exact
figure(2)
plot(iet,deval(exact,fet));
hold on
plot(iet,iey);
legend("ODE45","Improved Euler");
hold off

%part C
[ret,rey] = runge(f,h,t0,tf,y0);
%plot approx with exact
figure(3)
plot(ret,deval(exact,fet));
hold on
plot(ret,rey);
legend("ODE45","Fourth Order Runge-Kutta");
hold off

%%
%compute convergence

%Forward Euler
n = 10;
fError = zeros(1,n);
hvals = 2.^-[1:n];
for i = 1:n
    h = hvals(i);
    [fet,fey] = forwardEuler(f,h,t0,tf,y0);
    fError(i) = norm(fey - deval(exact,fet),Inf);
end
%get slope
```

```
figure(4)
loglog(hvals,fError);
p = polyfit(log(hvals),log(fError),1);
p(1)

%%
%Improved Euler
n = 10;
iError = zeros(1,n);
hvals = 2.^-[1:n];
for i = 1:n
    h = hvals(i);
    [iet,iey] = improvedEuler(f,h,t0,tf,y0);
    iError(i) = norm(iey - deval(exact,iet),Inf);
end
%get slope
figure(5)
loglog(hvals,iError);
p = polyfit(log(hvals),log(iError),1);
p(1) %outputs


%%
%Runge
n = 7;
rError = zeros(1,n);
hvals = 2.^-[1:n];
for i = 1:n
    h = hvals(i);
    [ret,rey] = runge(f,h,t0,tf,y0);
    rError(i) = norm(rey - deval(exact,ret),Inf);
end

%get slope
figure(6)
loglog(hvals,rError);
p = polyfit(log(hvals),log(rError),1);
p(1) %outputs

format short
```

Listing 2: **Forward Euler Method**

```
function [times,values] = forwardEuler(f,h,t0,tf,y0)
```

```

    times = [t0:h:tf];
    values = zeros(1,length(times));

    values(1) = y0;



    for i=2:length(times)
        values(i) = values(i-1) + h*f(values(i-1),times(i-1));
    end

end
```

Listing 3: **Improved Euler Method**

```
function [times,values] = improvedEuler(f,h,t0,tf,y0)

    times = [t0:h:tf];
    values = zeros(1,length(times));

    values(1) = y0;



    for i=2:length(times)
        Y = values(i-1) + h*f(values(i-1),times(i-1));
        values(i) = values(i-1) + 0.5*h*(f(values(i-1),times(i-1))
            + f(Y,times(i)));
    end

end
```

Listing 4: **Classical Fourth-Order Runge-Kutta Method**

```
function [times,values] = runge(f,h,t0,tf,y0)

    times = [t0:h:tf];
    values = zeros(1,length(times));

    values(1) = y0;



    for i=2:length(times)
        th = (times(i-1) + times(i))/2;
        Y1 = values(i-1);
        Y2 = values(i-1) + 0.5*h*f(Y1,times(i-1));
        Y3 = values(i-1) + 0.5*h*f(Y2,th);
        Y4 = values(i-1) + h*f(Y3,th);
        values(i) = values(i-1) + 1/6 * h * (f(Y1,times(i-1)) + 2*
           f(Y2,th) + 2*f(Y3,th) + f(Y4,times(i)));
    end

end
```

□