Problem P27: **a.** In preparation for the problem **P29** below, write two solvers,

function [tt, zz] = feuler(f, eta, t0, tf, N)

function [tt, zz] = rk4(f, eta, t0, tf, N)

which imploement schemes (5.19) and (5.33), respectively, to solve the ODE IVP in (5.1) and (5.2). The first input to these solver is function $z = f(t, u)$. The other inputs are a vector of initial values $eta = u(t_0)$, the initial time $t0$, the final time tf, and the number of equal-length steps N. Each solver outputs the entire trajectory, so tt is a 1D array of length $N + 1$ starting with t_0 and ending with t_f. If $\eta \in \mathbb{R}^s$ then zz is a 2D array with s rows and $N + 1$ columns; each column i gives the solution $u(t)$ at the ith time tt.

Solution:
Consider the following code,

Code:

```
function [tt, zz] = feuler(f, eta, t0, tf, N)
% This function applies forward euler to obtain the solution
% of an ODE IVP. f(t, u), is an inline function which describes
% the left hand side. eta is the vector of initial values.
% t0 and tf are initial and final times and N is the number of
% steps.

k = (tf - t0)/N;
tt = t0:k:tf;
[m, n] = size(eta);
zz = zeros(m, N+1);
zz(:, 1) = eta;
for i = 1:N
        zz(:, i+1) = zz(:, i) + k.*(f((t0+(k*i)), zz(:, i)));
end

end
```

Code:

```
function [tt, zz] = rk4(f, eta, t0, tf, N)
% This function applies fourth order runge kutta
% to obtain the solution of an ODE IVP. f(t, u),
% is an inline function which describes
% the left hand side. eta is the vector of initial values.
% t0 and tf are initial and final times and N is the number of
% steps.

k = (tf - t0)/N;
tt = t0:k:tf;
[m, n] = size(eta);
zz = zeros(m, N+1);
zz(:, 1) = eta;
```

```
for i = 1:N
    Y1 = zz(:, i);
    Y2 = zz(:, i) + (k/2).*(f((t0+(k*i)), Y1));
    Y3 = zz(:, i) + (k/2).*(f((t0+(k*i) + k*.5), Y2));
    Y4 = zz(:, i) + k.*(f((t0+(k*i) + k*.5), Y3));
    zz(:, i+1) = zz(:, i) + (k/6).*(f((t0+(k*i)), Y1) + ...
        (2.*(f((t0+(k*i)+ k*.5), Y2))) + ...
        (2.*(f((t0+(k*i)+ k*.5), Y3))) + ...
        (f((t0+(k*(i+1))), Y4)) );

end

end
```

b. Solve the following simple problem exactly:

$$u''(t) + u(t) = 0 \qquad u(0) = 1, \qquad u'(0) = 0$$

Solution:
Clearly the solution is $u(t) = \cos(t)$, here is the solution worked out in detail. First we form the characteristic polynomial by substituting $u(t) = e^{rt}$,

$$\begin{aligned} u''(t) + u(t) &= 0, \\ r^2 e^{rt} + e^{rt} &= 0, \\ e^{rt}(r^2 + 1) &= 0. \end{aligned}$$

Since our characteristic polynomial has a roots $r = \pm i$ we form the general solution with,

$$u(t) = c_1 e^{it} + c_2 e^{-it}.$$

Applying Euler's formula we get a solution of the form,

$$u(t) = c_1(\cos(t) + i\sin(t)) + c_2(\cos(t) - i\sin(t)).$$

Computing the derivative we get,

$$u'(t) = c_1(-\sin(t) + i\cos(t)) + c_2(-\sin(t) - i\cos(t)).$$

Applying our initial values we get,

$$\begin{aligned} u(0) = 1 &= c_1(\cos(0) + i\sin(0)) + c_2(\cos(0) - i\sin(0)), \\ 1 &= c_1 + c_2, \end{aligned}$$

and,

$$\begin{aligned} u'(0) = 0 &= c_1(-\sin(0) + i\cos(0)) + c_2(-\sin(0) - i\cos(0)), \\ 0 &= c_1 i - c_2 i. \end{aligned}$$

Which gives us $c_1 = c_2 = 1/2$, so $u(t) = \cos(t)$ as expected.

c. The problem in **b**, for example on the interval $[0, 2]$, makes a good test case. Demonstrate that the final time numerical error of each solver in **a** converges at the expected rate as the timestep $k \to 0$.

Solution:
First we need to rewrite our second order problem as a system of first order equations of the form $u'(t) = f(t, u(t))$. Let $x(t) = u_1(t)$ and $x'(t) = u_2(t)$ and therefore we get the,

$$u_1'(t) = u_2(t)$$
$$u_2'(t) = -u_1(t)$$

So as a system with initial conditions η we get,

$$\begin{bmatrix} u_1'(t) \\ u_2'(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} u_1(t) \\ u_2(t) \end{bmatrix}$$

$$\eta = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Running this example using our two solvers we get the following convergence rates, $\approx O(k^1)$ for forward euler and $\approx O(k^4)$ for the fourth order Runge Kutta. This is as expected as each each step accumulates a truncation error of $O(k^2)$ for forward euler, after k steps the truncation error is $O(k)$. Similarly with RK4 which accumulates a truncation error of $O(k^5)$ at each step.

Figure 1: feuler on test case

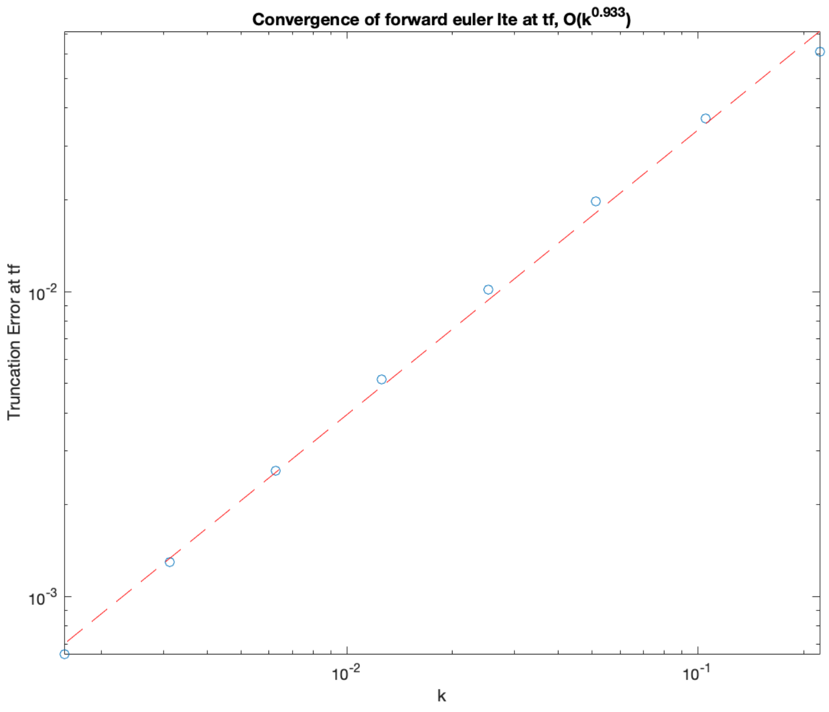
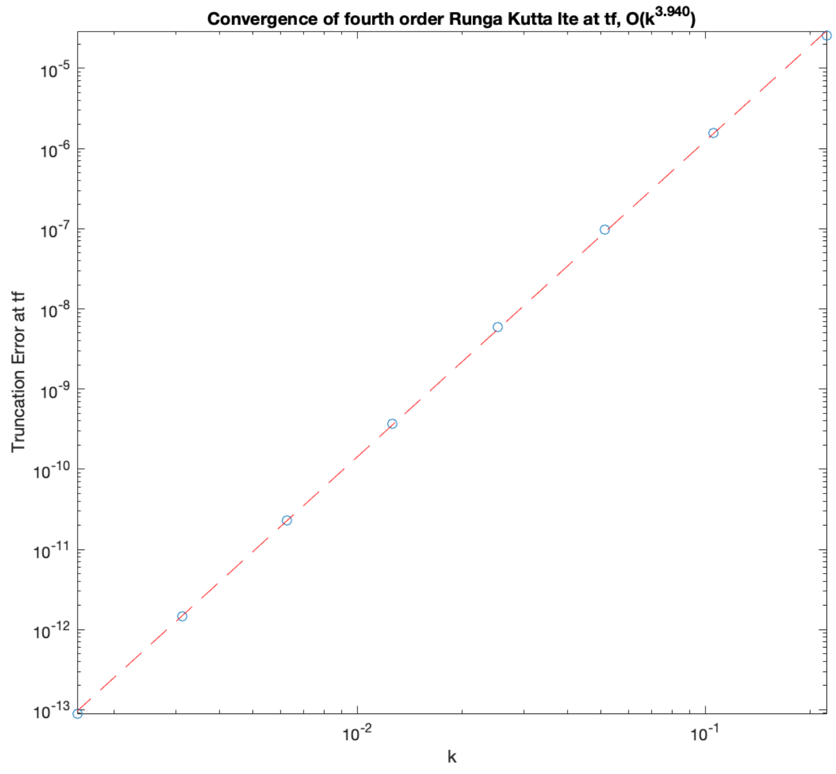


Figure 2: rk4 on test case



Code:

```
f = @(t, u) ([0 1; -1 0]*u);
eta = [1 0]';
t0 = 0;
tf = 2;

uexact = @(x) cos(x);
Nlist = [10 20 40 80 160 320 640 1280];
err = [];
klist = 2 ./ (Nlist - 1);



for k = 1:length(Nlist)
    [tt,zz] = rk4(f,eta,t0,tf,Nlist(k));
    err = [err abs(zz(1,Nlist(k) + 1) - uexact(tf))];
end

p = polyfit(log(klist),log(err),1);
fprintf('convergence at rate O(h^k) with k = %f\n',p(1))
loglog(klist,err,'o',klist,exp(p(2) + p(1)*log(klist)),'r--')
xlabel k,  ylabel('Truncation Error at tf')
text(0.01,0.0002,sprintf('O(h^{%.3f})',p(1)),'Color','r','FontSize',14)
```

```
title(sprintf('Convergence of fourth
    order Runga Kutta lte at tf, O(k^{%.3f})',p(1)))
axis tight
```

Problem P28: Compute the leading term in the local truncation error of the following methods. For parts (**a**) and (**b**), please follow the style of Example 5.9, wherein you learn the coefficient of the leading order term. For part (**c**) you can follow the style of Example 5.11 and only get the order, without knowing the leading order coefficient.

a The 2-step BDF method (5.25)

Solution:
Recall that the 2-step backwards differentiation method is given by,

$$\frac{3U^{n+1} - 4U^n + U^{n-1}}{2k} = f(U^{n+1}).$$

Computing the truncation error as the residual of the scheme applied to the exact solution, we get,

$$\tau^{n+1} = \frac{3u(t_{n+1}) - 4u(t_n) + u(t_{n-1})}{2k} - f(u(t_{n+1})).$$

Expanding via Taylor's Theorem about t_{n+1} and substituting the solution $f(u(t_{n+1})) = u'(t_{n+1})$ we get,

$$\begin{aligned}
\tau^{n+1} &= \frac{3u(t_{n+1}) - 4u(t_n) + u(t_{n-1})}{2k} - f(u(t_{n+1})),\\
&= \frac{1}{2k}\Big(3u(t_{n+1}) - 4\Big(u(t_{n+1}) - ku'(t_{n+1}) + \frac{1}{2}k^2u''(t_{n+1}) - \frac{1}{6}k^3u'''(t_{n+1}) + O(k^4)\Big)\\
&\quad + \Big(u(t_{n+1}) - 2ku'(t_{n+1}) + 2k^2u''(t_{n+1}) - \frac{8}{6}k^3u'''(t_{n+1}) + O(k^4)\Big)\Big) - u'(t_{n+1}),\\
&= \frac{1}{2k}\Big(2ku'(t_{n+1}) - \frac{4}{6}k^3u'''(t_{n+1}) + O(k^4)\Big) - u'(t_{n+1})\\
&= -\frac{1}{3}k^2u'''(t_{n+1}) + O(k^3)
\end{aligned}$$

b The trapezoidal method (5.22),

Solution:
Recall that the trapezoidal method is given by,

$$\frac{U^{n+1} - U^n}{k} = \frac{1}{2}(f(U^n) + f(U^{n+1})).$$

Computing the truncation error τ^* for the $\tau_n + \frac{1}{2}k$ term,

$$\tau^* = \frac{u(t_{n+1}) - u(t_n)}{k} - \frac{1}{2}(f(u(t_n)) + f(u(t_{n+1})))$$

Expanding via Taylor's Theorem about $t_* = t_n + \frac{1}{2}k$ and substituting the solution $f(u(t_{n+1})) = u'(t_{n+1})$ and $f(u(t_n)) = u'(t_n)$,

$$\begin{aligned}\tau^* = \frac{1}{k}\Bigg(&\left(u(t_*) + \frac{1}{2}ku'(t_*) + \frac{1}{8}k^2u''(t_*) + \frac{1}{48}k^3u'''(t_*) + O(k^4)\right)\\ &-\left(u(t_*) - \frac{1}{2}ku'(t_*) + \frac{1}{8}k^2u''(t_*) - \frac{1}{48}k^3u'''(t_*) + O(k^4)\right)\Bigg)\\ &-\frac{1}{2}\left(u'(t_n) + u'(t_{n+1})\right),\end{aligned}$$

$$\begin{aligned}\tau^* = \frac{1}{k}\Bigg(&\left(u(t_*) + \frac{1}{2}ku'(t_*) + \frac{1}{8}k^2u''(t_*) + \frac{1}{48}k^3u'''(t_*) + O(k^4)\right)\\ &-\left(u(t_*) - \frac{1}{2}ku'(t_*) + \frac{1}{8}k^2u''(t_*) - \frac{1}{48}k^3u'''(t_*) + O(k^4)\right)\Bigg)\\ -\frac{1}{2}\Bigg(&\left(u'(t_*) - \frac{1}{2}ku''(t_*) + \frac{1}{8}k^2u'''(t_*) - \frac{1}{48}k^3u''''(t_*) + O(k^4)\right)\\ &+\left(u'(t_*) + \frac{1}{2}ku''(t_*) + \frac{1}{8}k^2u'''(t_*) + \frac{1}{48}k^3u''''(t_*) + O(k^4)\right)\Bigg),\end{aligned}$$

$$\begin{aligned}\tau^* &= \frac{1}{k}\left(ku'(t_*) + \frac{1}{24}k^3u'''(t_*) + O(k^4)\right)\\ &\quad-\frac{1}{2}\left(2u'(t_*) + \frac{1}{4}k^2u'''(t_*) + O(k^4)\right),\\ &= -\frac{1}{12}k^2u'''(t_*) + O(k^4).\end{aligned}$$

c The explicit trapezoid method,

$$U^{n+1} = U^n + \frac{k}{2}\Big(f(U^n) + f\Big(U^n + kf(U^n)\Big)\Big).$$

Solution:
Computing the truncation error we get the following,

$$\tau^n = \frac{1}{k}\left(u(t_{n+1}) - u(t_n)\right) - \frac{1}{2}\Big(f(u(t_n)) + f\Big(u(t_n) + kf(u(t_n))\Big)\Big).$$

Note that by substitution of the exact solution,

$$f\Big(u(t_n) + kf(u(t_n))\Big) = f\Big(u(t_n) + ku'(t_n)\Big).$$

Expanding the function via Taylor's Theorem about $u(t_n)$, and substituting the exact solution and $f'(u)u' = u''$,

$$\begin{aligned} f\Big(u(t_n) + ku'(t_n)\Big) &= f(u(t_n)) + ku'(t_n)f'(u(t_n)) + \frac{1}{2}k^2(u'(t_n))^2 f''(u(t_n)) + \dots \\ &= u'(t_n) + ku''(t_n) + O(k^2). \end{aligned}$$

Substituting into our original truncation error calculation, and using Taylor's Theorem to expand $u(t_{n+1})$ about t_n we get,

$$\begin{aligned} \tau^n &= \frac{1}{k}\left(u(t_{n+1}) - u(t_n)\right) - \frac{1}{2}\Big(f(u(t_n)) + f\Big(u(t_n) + kf(u(t_n))\Big)\Big), \\ &= \frac{1}{k}\Big(u(t_n) + ku'(t_n) + \frac{1}{2}k^2u''(t_n) + \frac{1}{6}k^3u'''(t_n) + O(k^4) - u(t_n)\Big) \\ &- \frac{1}{2}(u'(t_n)) - \frac{1}{2}\Big(u'(t_n) + ku''(t_n) + O(k^2)\Big), \\ &= u'(t_n) + \frac{1}{2}ku''(t_n) + \frac{1}{6}k^2u'''(t_n) + O(k^3) - \frac{1}{2}(u'(t_n)) - \frac{1}{2}\Big(u'(t_n) + ku''(t_n) + O(k^2)\Big), \\ &= O(k^2). \end{aligned}$$

Problem P29: Consider the problem of two massive bodies, with masses m_1, and m_2. They are attracted by gravity only. They travel in a plane so their positions are given by vector-values functions $x_i(t) = (x_i(t), y_i(t))$ for $i = 1, 2$. Newton's second law and Newton's law of gravity combine to say:

$$m_1x_1'' = -Gm_1m_2\frac{x_1 - x_2}{|x_1 - x_2|^3}$$

$$m_2x_2'' = -Gm_1m_2\frac{x_2 - x_1}{|x_1 - x_2|^3}$$

We will consider the Earch and the Moon in isolation as our example. Thus the constants are,

$$m_1 = 5.972 \times 10^{24}kg$$

$$m_2 = 7.348 \times 10^{22}kg$$

$$G = 6.674 \times 10^{-11}m^3kg^{-1}s^{-2}$$

and we measure t in seconds and x_i, y_i in meters.

a By using notation $v_i = x_i'$, $w_i = y_i'$ for $i = 1, 2$, write the problem as a first oder ODE system of dimension $s = 8$, with solution column vector $u(t) \in \mathbb{R}^8$. Use the component ordering,

$$\begin{aligned} u(t) &= [x_1(t) \quad y_1(t) \quad x_2(t) \quad y_2(t) \quad v_1(t) \quad w_1(t) \quad v_2(t) \quad w_2(t)]^T \\ &= [u_1(t) \quad u_2(t) \quad u_3(t) \quad u_4(t) \quad u_5(t) \quad u_6(t) \quad u_7(t) \quad u_8(t)]^T \end{aligned}$$

That is, write the system in the form $u'(t) = f(t, u(t))$. Then implement a single function,

function z = fearthmoon(t, u)

which compute the right-hand-side function $f(t, u)$ of the ODE system.

Solution:
Rewriting the system in the form of $u'(t) = f(t, u(t))$ with the given substitution,

$$
\begin{aligned}
u'(t) &= [x_1'(t) \quad y_1'(t) \quad x_2'(t) \quad y_2'(t) \quad v_1'(t) \quad w_1'(t) \quad v_2'(t) \quad w_2'(t)]^T \\
&= [v_1(t) \quad w_1(t) \quad v_2(t) \quad w_2(t) \\
&\quad - Gm_2\frac{x_1(t) - x_2(t)}{|x_1 - x_2|^3} \quad - Gm_2\frac{y_1(t) - y_2(t)}{|x_1 - x_2|^3} \quad - Gm_1\frac{x_2(t) - x_1(t)}{|x_1 - x_2|^3} \quad - Gm_1\frac{y_2(t) - y_1(t)}{|x_1 - x_2|^3}]^T \\
&= [u_5(t) \quad u_6(t) \quad u_7(t) \quad u_8(t) \\
&\quad - Gm_2\frac{u_1(t) - u_3(t)}{|x_1 - x_2|^3} \quad - Gm_2\frac{u_2(t) - u_4(t)}{|x_1 - x_2|^3} \quad - Gm_1\frac{u_3(t) - u_1(t)}{|x_1 - x_2|^3} \quad - Gm_1\frac{u_4(t) - u_2(t)}{|x_1 - x_2|^3}]^T
\end{aligned}
$$

Finally let $|x_1 - x_2|^3 = \left(\sqrt{(u_1(t) - u_3(t))^2 + (u_2(t) - u_4(t))^2}\right)^3$.

b Solution:
The following is a code for the matlab function fearthmoon(t, u),

Code:

```
function z = fearthmoon(t, u)
m1 = 5.972e24;
m2 = 7.348e22;
G = 6.674e-11;
r3 = (sqrt((u(1) - u(3))^2 + (u(2) - u(4))^2))^3;

z = [u(5); u(6); u(7); u(8);
    -G*m2*((u(1) - u(3))/r3);
    -G*m2*((u(2) - u(4))/r3);
    -G*m1*((u(3) - u(1))/r3);
    -G*m1*((u(4) - u(2))/r3)];
end
```

The following are the figures produced by solving the system using rk4, feuler, and ode45, as well as the driver code for generating the figures,

Figure 3: feuler solution for $N = 40$

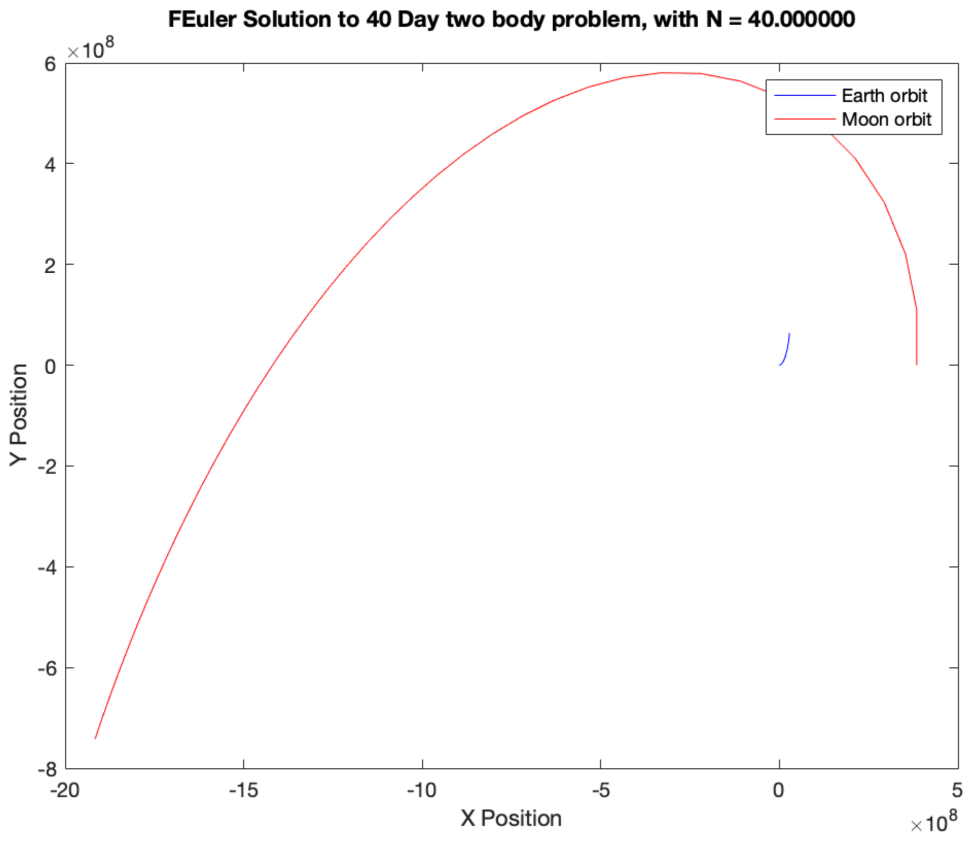


Figure 4: feuler solution for $N = 960$

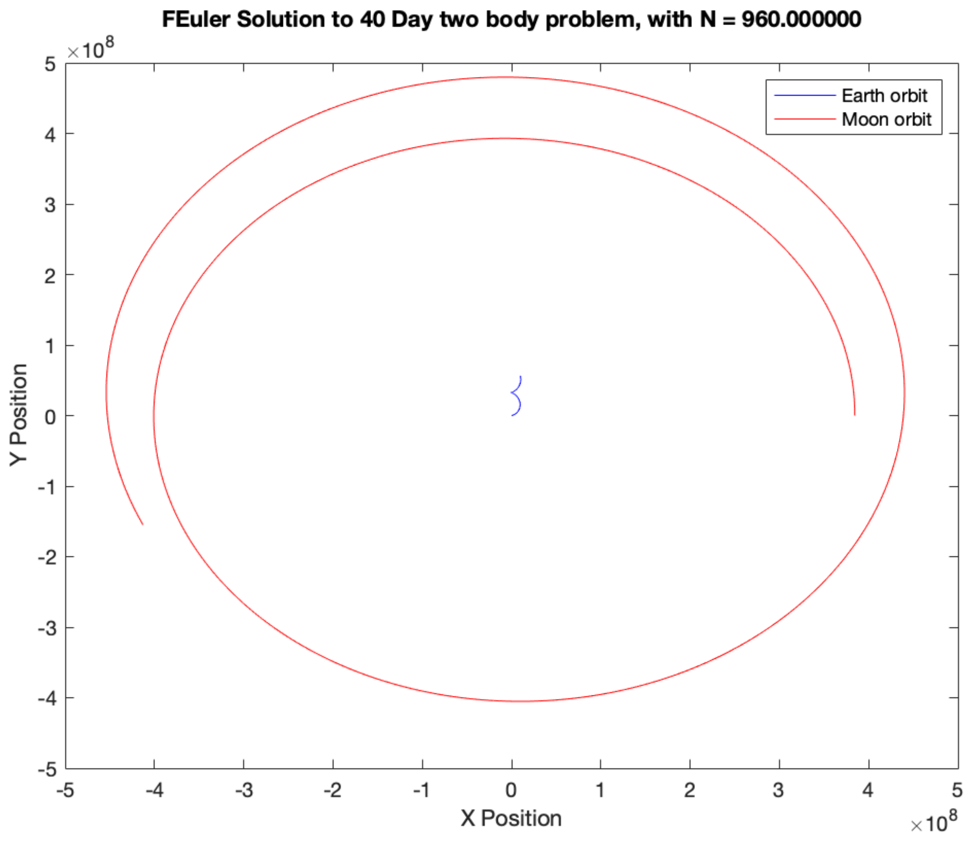


Figure 5: rk4 solution for $N = 40$

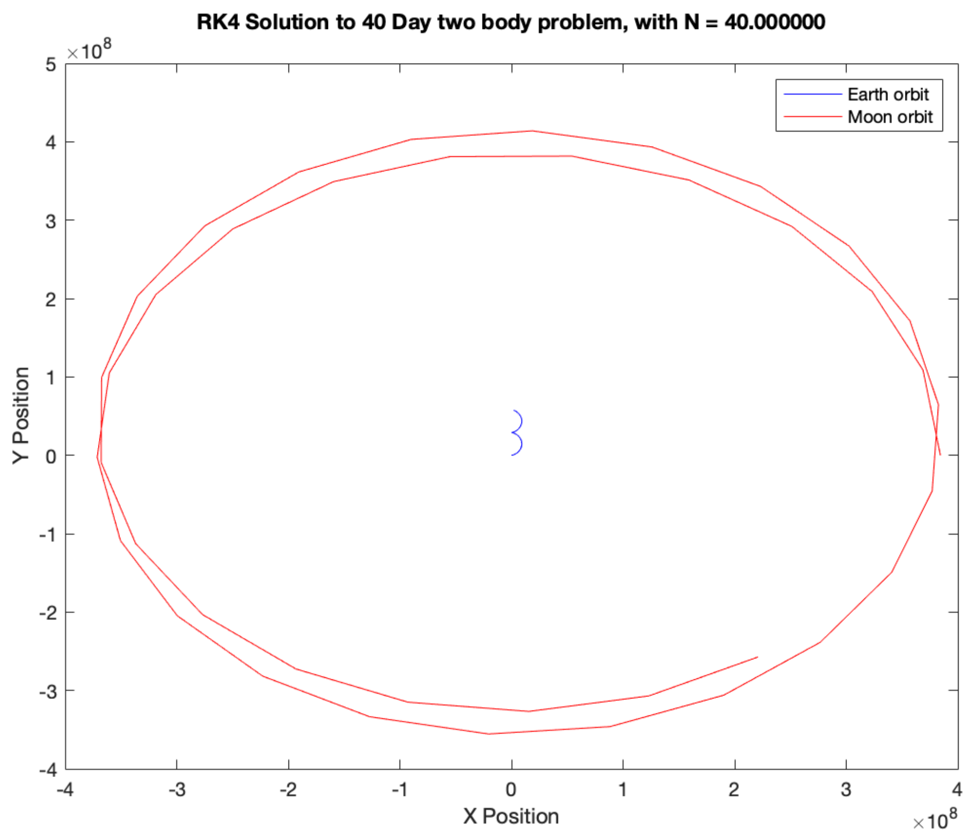


Figure 6: rk4 solution for $N = 960$

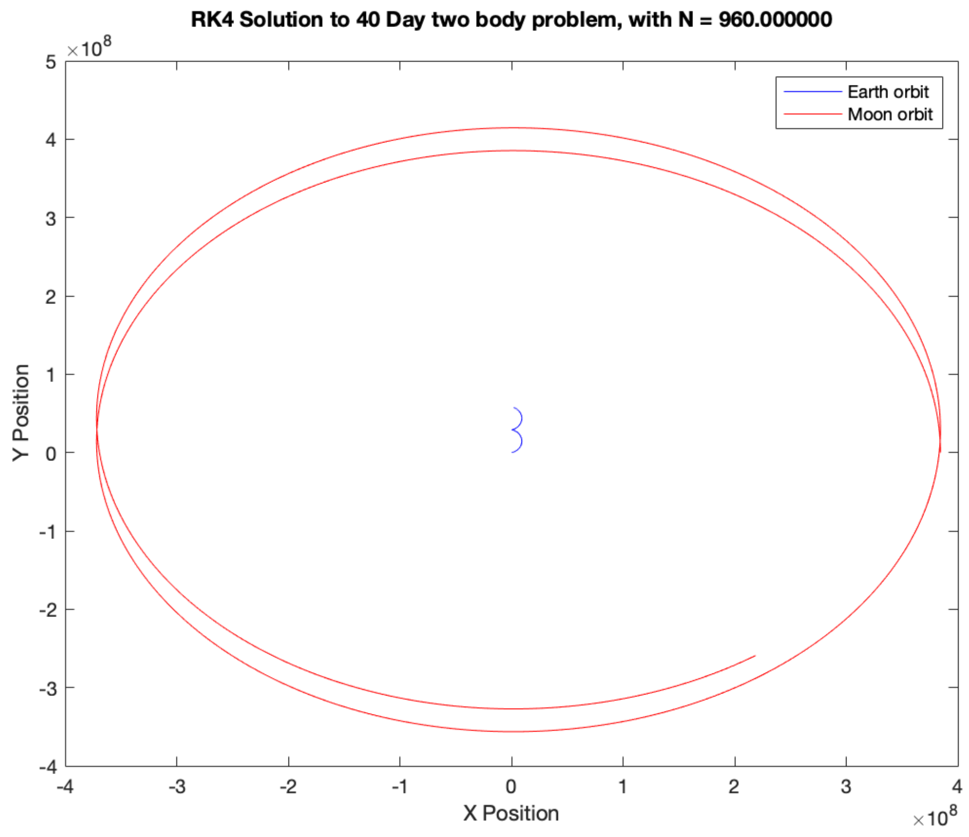
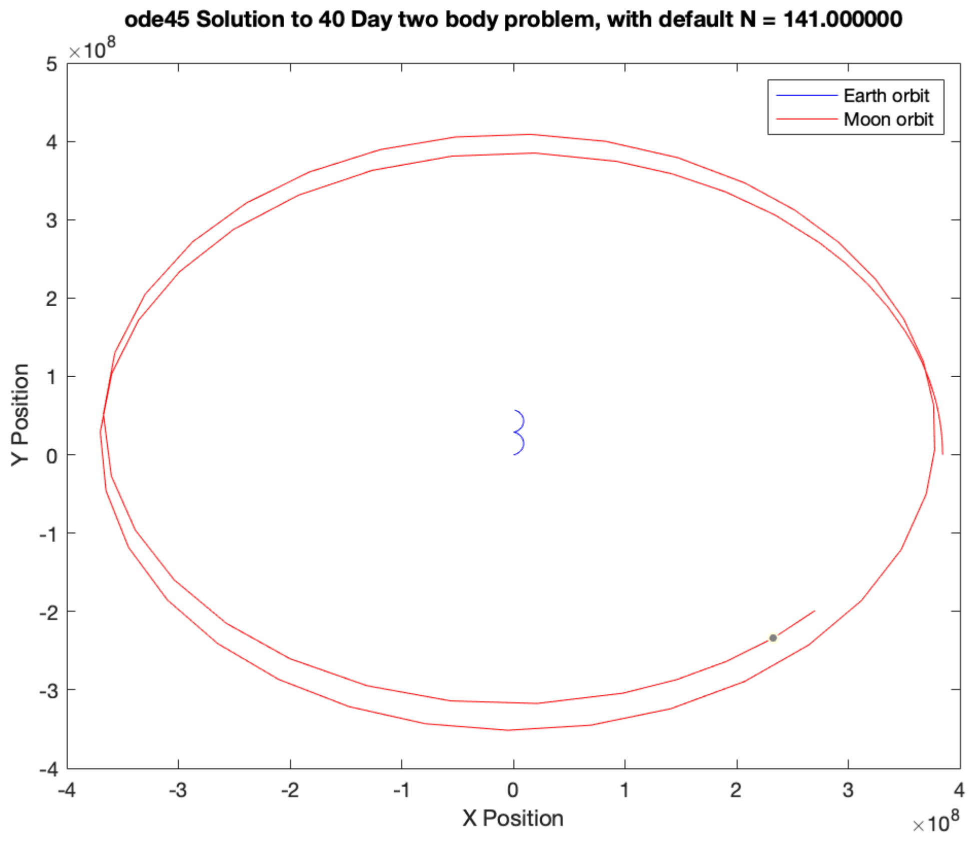


Figure 7: ode45 solution for default step sizes



Code:

```
eta = [0 0 3.844e8 0 0 0 0 1.0223e3]';
t0 = 0;
tf = 4.32e6;

Nlist = [40 960];
for k = 1:length(Nlist)
    [tt,zz] = feuler(@fearthmoon,eta,t0,tf,Nlist(k));
    plot(zz(1,:),zz(2,:), 'b', zz(3,:),zz(4,:), 'r' )
    title(sprintf('FEuler Solution to 40 Day ...
    two body problem, with N = %f\n',Nlist(k)))
    legend('Earth orbit', 'Moon orbit')
    xlabel ('X Position'),  ylabel ('Y Position')
end

[t, u] = ode45(@fearthmoon, [0 4.32e6], eta);
u = u';
plot(u(1,:),u(2,:), 'b', u(3,:),u(4,:), 'r' )
    title(sprintf('ode45 Solution to 40 Day ...
     two body problem, with default N = %f\n',length(t)))
    legend('Earth orbit', 'Moon orbit')
    xlabel ('X Position '), ylabel ('Y Position ')
```

c How long is a lunar month, if we used your computations in part **b**

Solution:
Consider the following matlab code, which computes the normalized difference vector between the moon position and earth position to get a relative moon orbit. Then we choose the first position and innerproduct that with the rest of the relative moon orbit, taking the maximum index to see which time step achieves the same position. It computes to approx 21 days which I certain is incorrect, and I'm not sure if it's a bug in the code or the calculation.

Code:

```
eta = [0 0 3.844e8 0 0 0 0 1.0223e3]';
t0 = 0;
tf = 4.32e6;
[tt,zz] = rk4(@fearthmoon,eta,t0,tf,960);
moonposition = [zz(3,:)' zz(4,:)'];
earthposition = [zz(1,:)' zz(2,:)'];
lunarorbit = (moonposition - earthposition)...
    ./vecnorm(moonposition - earthposition, 2, 2);
x0 = lunarorbit(1,:);
innerproducts = (x0*lunarorbit')';
[M, I] = max(innerproducts(2:end));
lunarmonthindays = (I + 1)/24;
lunarmonthindays =

  21.750000000000000
```