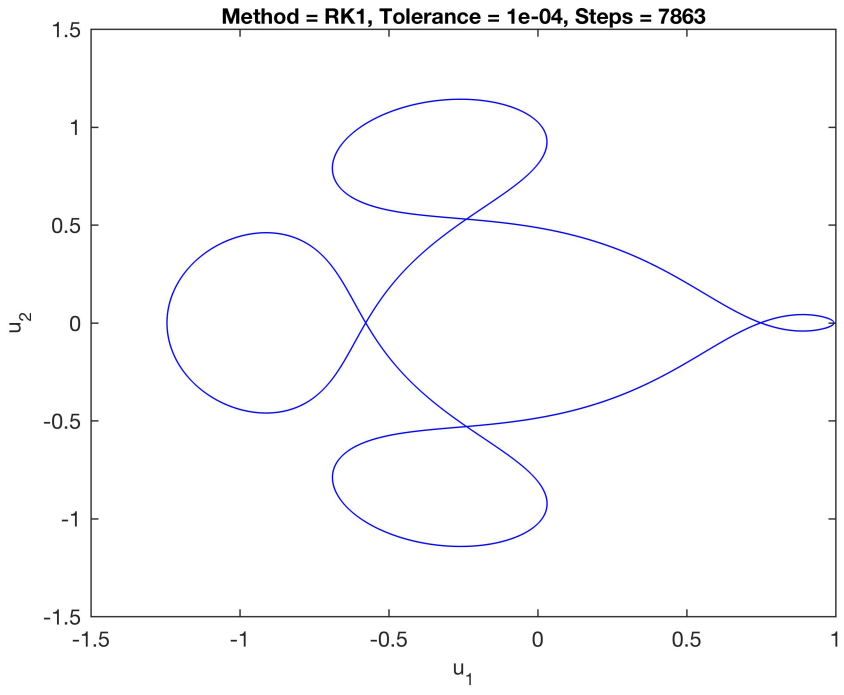
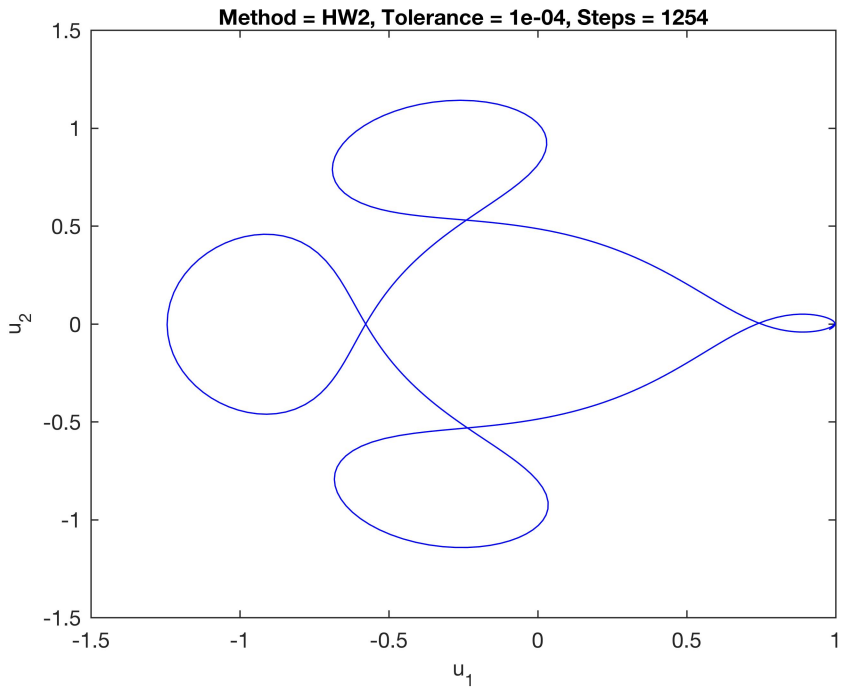
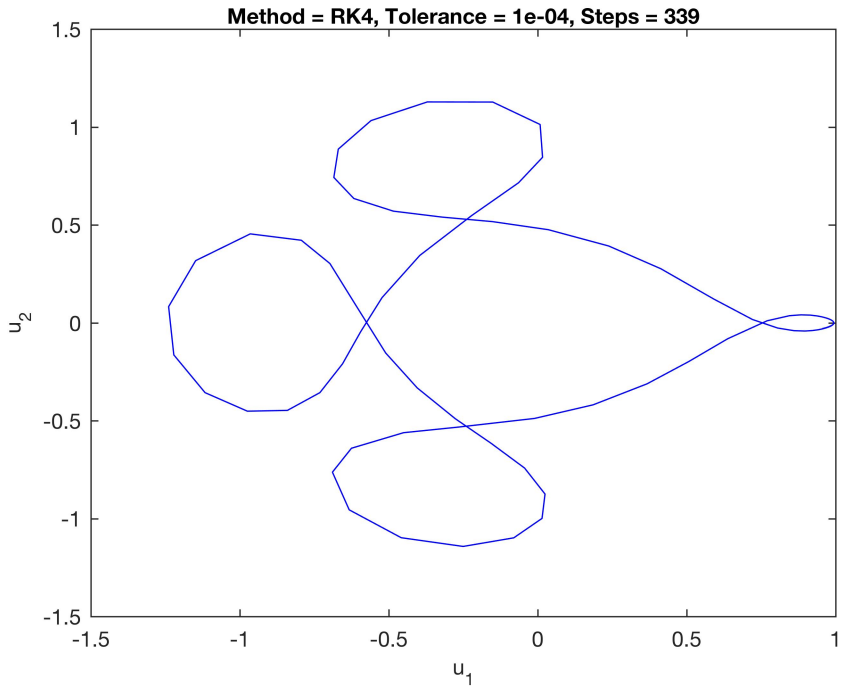
Andrew Alferman
MTH 552
Homework #4 (Corrected)
Due Wed, Feb 15, 2017 (Corrections due 12:00PM Sun, Feb 19, 2017)

1) The attached MATLAB code has been modified as directed. The code allows for either fixed stepsize or automatic stepsize to be selected, depending if the value assigned to "autostep" is true or false. The "method" variable represents the numerical method to be used; 'RK1' and 'ExplicitEuler' represent the Euler method, 'RK4' represents the RK method from problem 2 of Assignment 2, and 'HW2' represents the RK method from problem 2 of Assignment 2 (modified per email on 2/14/17). The "steps" global variable counts how often the function f is being evaluated.

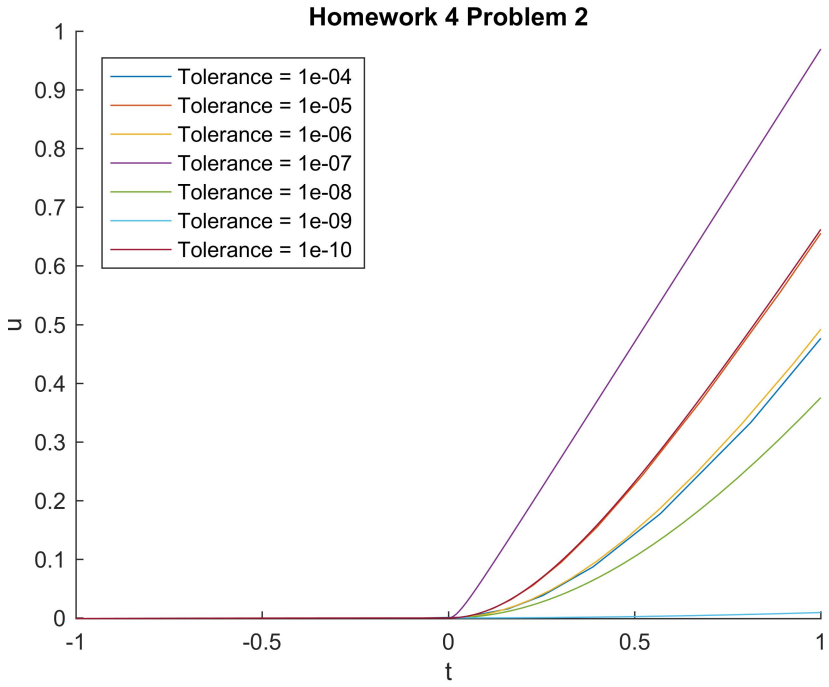
A tolerance of $1x10^{-4}$ was selected because it yielded plot of u_1 versus u_2 that was approximately correct visually for all three methods evaluated. A tolerance value of $2x10^{-4}$ with the HW2 method yielded a plot that was noticeably distorted near the completion of the first orbit, while the HW2 method and RK4 methods yielded acceptable plots at that tolerance value. The RK4 plot appeared slightly jagged in some portions of the orbit due to the small number of steps used. The code output the figures below. The number of steps can be found on each plot (7863, 1254, and 339 for RK1, HW2, and RK4, respectively). All three methods came up with comparable plots, however the RK4 method was faster than the other two methods. On the computer used to produce the figures, the RK1 compiled in approximately 0.544 seconds (decreased to 0.496 when using the "ExplicitEuler" function), the HW2 method compiled in 0.311 seconds, and the RK4 method compiled in 0.284 seconds. The RK4 method appeared to be fastest, though all computation times were very similar and the results may vary based on background program usage.



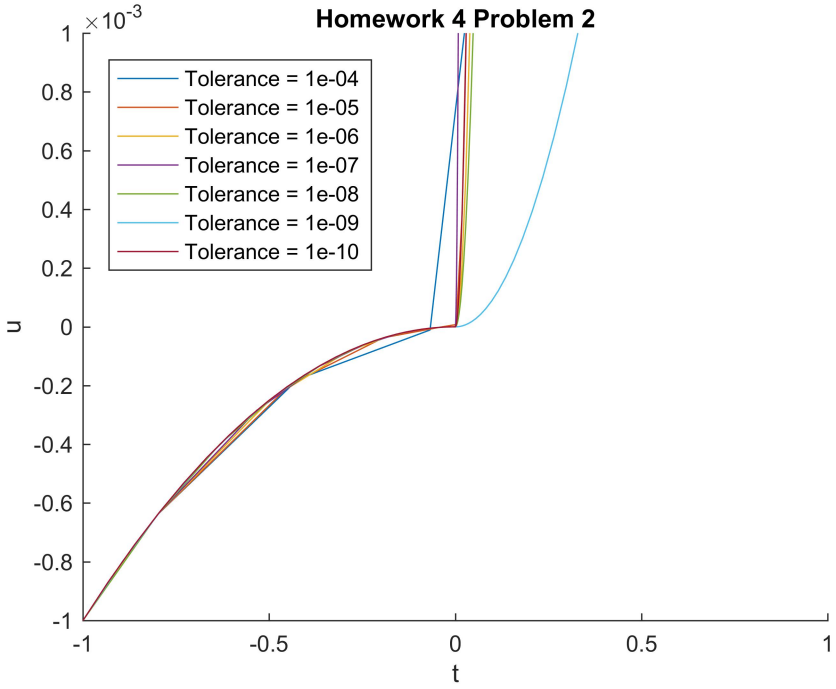

Method = HW2, Tolerance = 1e-04, Steps = 1254
1.5
1
0.5
0
-0.5
-1
-1.5
-1.5
-1
-0.5
0
0.5
1
u2
u1


Method = RK4, Tolerance = 1e-04, Steps = 339
1.5
1
0.5
0
-0.5
-1
-1.5
-1.5
-1
-0.5
0
0.5
1
u2
u1

2) The attached code has been modified was modified as directed. At first, it appears as if the solution to the IVP is completely flat for $-1 \leq t \leq 0$, while each of the curves for $0 < t \leq 1$ are different for each of the tolerances that were evaluated. This result is illustrated below:



Upon closer inspection, by limiting the y axis to $-0.001 \leq u \leq 0.001$ it can be seen that the IVP converges from $-1 \leq t < 0$, and the solution blows up when $0 \leq t$. The results do not converge when $0 \leq t$ because there is a discontinuity at $t = 0$ which "breaks" the numerical method (the existence and uniqueness theorem with a global Lipschitz condition isn't satisfied).


Homework 4 Problem 2
×10^-3
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
u
-1
-0.5
0
0.5
1
t
Tolerance = 1e-04
Tolerance = 1e-05
Tolerance = 1e-06
Tolerance = 1e-07
Tolerance = 1e-08
Tolerance = 1e-09
Tolerance = 1e-10

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This code is applicable to problem 1 of Homework 4
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all
close all

% Main routine (driver) for orbital ODE problem
% Specify name of user supplied function M-file with rhs of ode
odefun = 'orbitODE';
% Specify the method to be used
% Options are ExplicitEuler, RK1, RK4, HW2
method = 'RK1';
% Specify if you want automatic time steps, and if so, the tolerance
autostep = true;
TOL = 1.0*(10^-4);
global steps;
steps = 0;
% If automatic time steps are not to be used, specify the number of
 steps
NSTEP=5*(10^5);
% Specify the initial conditions
% Specify initial and final times
t0 = 0; tfinal = 17.1;
TSPAN = [t0,tfinal];
% Specify column vector of initial values
U0 = [0.994;0.0;0.0;-2.00158510637908252240537862224];
mu = 0.012277471;

% Build the Butcher array based on the method selected
if strcmp(method, 'RK4')
    A = [0 0 0 0; 0.5 0 0 0; 0 0.5 0 0; 0 0 1 0];
    b = [1/6;1/3;1/3;1/6];
    c = [0;0.5;0.5;1];
elseif (strcmp(method,'ExplicitEuler') || strcmp(method,'RK1'))
    A = 0;
    b = 1;
    c = 0;
elseif strcmp(method,'HW2')
    A = [0 0; 1 0];
    b = [0.5;0.5];
    c = [0;1];
end
% Series of if statements that will return the order of the RK method
 used
if round(sum(b),4) == 1.0000
    qorder = 1;
    if round(b.'*c,4) == 0.5000
        qorder = 2;
        if round(b.'*(c.^2),4) == round(1/3,4) && ...
           round(sum(b'.*sum(A'.*c)),4) == round(1/6,4)
            qorder = 3;
```

```
            if round(b.'*(c.^3),4) == 1/4 && ...
               round(sum(b'.*c'.*sum(A'.*c)),4) == 1/8 && ...
               round(sum(b'.*sum(A'.*c.^2)),4) == round(1/12,4) && ...
               round(sum(A.*b)*sum(A'.*c)',4) == round(1/24,4)
                qorder = 4;
            end
        end
    end
end

% Call the solver, based on if you want automatic time step or not
if autostep == false
    [t,U] = eulerw17d(odefun,TSPAN,U0,NSTEP,method,A,b,c,mu);
else
    [t,U] = RKw17sc(odefun,TSPAN,U0,TOL,A,b,c,qorder,mu);
end

% plot numerical solution;
figure;
hold off;
plot(U(1,:),U(2,:),'b');
xlabel('u_1')
ylabel('u_2')
titlestr = sprintf(['Method = ' method ...
    ', Tolerance = %2.0e, Steps = %i'],TOL,steps);
title(titlestr)
```

Published with MATLAB® R2016b

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This code is applicable to both problems 1 and 2 of Homework 4
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [T,U] = RKw17sc(odefun,TSPAN,U0,TOL,A,b,c,qorder,mu);
% RK method with stepsize control using Richardson extrapolation
% qorder = order of method used

dtmin = 1.e-5; % minimum allowed step size
sf = 0.8; %safety factor

q1 = qorder+1;
T0 = TSPAN(1); TFINAL = TSPAN(2);
dt = (TFINAL-T0)/10;
m = length(U0);
U = U0;
T(1)=T0;
kstep = 1;
while T(kstep) < TFINAL
    t = T(kstep);
    flag = 1;
    while flag == 1;
        %one big step with dt
        Ubig = RKexplicitstep(odefun,t,U(:,kstep),dt,A,b,c,mu);

        % two small steps with dt/2
        dt2 = dt/2;
        V = RKexplicitstep(odefun,t,U(:,kstep),dt2,A,b,c,mu);
        Usmall = RKexplicitstep(odefun,t+dt2,V,dt2,A,b,c,mu);

        %Richardson extrapolation
        fac = 2^qorder;
        Unew = (fac*Usmall-Ubig)/(fac-1);

        %Estimate of one-step error for Usmall method
        locerr =norm(Unew-Usmall,1);

         % conditions for accepting current dt
        if (locerr <= TOL ) | (dt < 1.001*dtmin)
            flag = 0;
            kstep = kstep+1;
            U(:,kstep) = Unew; %local extrapolation
            T(kstep) = t + dt;
        end
        dt = max(sf*((TOL/locerr)^(1/q1))*dt,dtmin); %new value for dt
        dt = min(dt,TFINAL-t);
    end
end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function U = eulerstep(odefun,t,U0,dt,mu)
% Takes one step of the foward Euler method
global steps;
U = U0 + dt*feval(odefun,t,U0,mu);
steps = steps + 1;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function yprime = orbitODE(t,y,mu)
% Implements the ODE from Homework 2 Problem 3 as a first order system
muhat = 1.0 - mu;
D1 = ((y(1) + mu)^2 + y(2)^2)^1.5;
D2 = ((y(1) - muhat)^2 + y(2)^2)^1.5;
yprime = zeros(size(y));
yprime(1) = y(3);
yprime(2) = y(4);
yprime(3) = y(1) + 2*y(4) - muhat*((y(1)+mu)/D1) - mu*((y(1)-muhat)/
D2);
yprime(4) = y(2) - 2*y(3) - (muhat*(y(2)/D1)) - (mu*(y(2)/D2));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function U = RKexplicitstep(odefun,t,U0,dt,A,b,c,mu)
% One step of a general explicit Runge Kutta method
% A is assumed to be strictly lower triangular
% b and c must be column vectors
%
global steps;
r = length(b);
m = length(U0);
K = zeros(m,r);%matrix whose j-th column is K_j
K(:,1) = feval(odefun,t,U0,mu);
for j = 2:r
    Y = U0 + dt*K(:,1:j-1)*(A(j,1:j-1).');
     % Y = U0 + dt*sum_{l=1}^{j-1} a_{jl}K_l
    K(:,j) = feval(odefun,t + c(j)*dt, Y,mu);
end
U = U0 + dt*K*b;
steps = steps + 1;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function yprime = hw4p2(t,u,mu)
% ODE specific to problem 2 of homework 4
if (round(t,12) == 0 & round(u,12) == 0)
    yprime = 0;
else
    yprime = (2*t*u)/(t^2 + u^2);
end
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This code is applicable to problem 2 of Homework 4
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all
close all

% Main routine (driver) for orbital ODE problem
% Specify name of user supplied function M-file with rhs of ode
odefun = 'hw4p2';
% Specify the method to be used
% Options are ExplicitEuler, RK1, RK4, HW2
method = 'RK4';
% Specify if you want automatic time steps
autostep = true;
steps = 0;
% If automatic time steps are not to be used, specify the number of
 steps
NSTEP=5*(10^5);
% Specify the initial conditions
% Specify initial and final times
t0 = -1; tfinal = 1;
TSPAN = [t0,tfinal];
% Specify column vector of initial values
U0 = -0.001;
% This is leftover from the orbital ODE problem.  Not worth fixing it
 now.
mu = 0.012277471;

% Build the Butcher array based on the method selected
if strcmp(method, 'RK4')
    A = [0 0 0 0; 0.5 0 0 0; 0 0.5 0 0; 0 0 1 0];
    b = [1/6;1/3;1/3;1/6];
    c = [0;0.5;0.5;1];
elseif (strcmp(method,'ExplicitEuler') || strcmp(method,'RK1'))
    A = 0;
    b = 1;
    c = 0;
elseif strcmp(method,'HW2')
    A = [0 0; 1 0];
    b = [0.5;0.5];
    c = [0;1];
end
% Series of if statements that will return the order of the RK method
 used
if round(sum(b),4) == 1.0000
    qorder = 1;
    if round(b.'*c,4) == 0.5000
        qorder = 2;
        if round(b.'*(c.^2),4) == round(1/3,4) && ...
           round(sum(b'.*sum(A'.*c)),4) == round(1/6,4)
            qorder = 3;
```

```
            if round(b.'*(c.^3),4) == 1/4 && ...
               round(sum(b'.*c'.*sum(A'.*c)),4) == 1/8 && ...
               round(sum(b'.*sum(A'.*c.^2)),4) == round(1/12,4) && ...
               round(sum(A.*b)*sum(A'.*c)',4) == round(1/24,4)
                qorder = 4;
            end
        end
    end
end

figure;
hold on;
% Call the solver, based on if you want automatic time step or not
if autostep == false
    [t,U] = eulerw17d(odefun,TSPAN,U0,NSTEP,method,A,b,c,mu);
    plot(t,U);
else
    for i = 4:10
        TOL = 1*10^(-i);
        [t,U,steps] =
 RKw17sc(odefun,TSPAN,U0,TOL,A,b,c,qorder,mu,steps);
        legendInfo{i} = [sprintf('Tolerance = %1.0e', TOL)];
        plot(t,U);
    end
end

% plot numerical solution;
xlabel('t')
ylabel('u')
xlim([-1,1])
ylim([-0.001,0.001])
title('Homework 4 Problem 2')
legend(legendInfo(4:10), 'Location', 'NorthWest')
```

Published with MATLAB® R2016b