```
function [tout, yout] = ode45v4(ypfun, t0, tfinal, y0, tol, trace)
%ODE45 Solve differential equations, higher order method.
% ODE45 integrates a system of ordinary differential equations using
% 4th and 5th order Runge-Kutta formulas.
% [T,Y] = ODE45V4('yprime', T0, Tfinal, Y0) integrates the system of
% ordinary differential equations described by the M-file YPRIME.M,
% over the interval T0 to Tfinal, with initial conditions Y0.
% [T, Y] = ODE45V4(F, T0, Tfinal, Y0, TOL, 1) uses tolerance TOL
% and displays status while the integration proceeds.
%
% INPUT:
% ypfun  - String containing name of user-supplied problem
 description.
%         Call: yprime = fun(t,y) where F = 'fun'.
%         t      - Time (scalar).
%         y      - Solution column-vector.
%         yprime - Returned derivative column-vector; yprime(i) =
 dy(i)/dt.
% t0    - Initial value of t.
% tfinal- Final value of t.
% y0    - Initial value column-vector.
% tol   - The desired accuracy. (Default: tol = 1.e-6).
% trace - If nonzero, each step is printed. (Default: trace = 0).
%
% OUTPUT:
% tout  - Returned integration time points (column-vector).
% yout  - Returned solution, one solution column-vector per tout-
value.
%
% The result can be displayed by: plot(tout, yout).
%
% See also ODE23, ODEDEMO.

% C.B. Moler, 3-25-87, 8-26-91, 9-08-92.
% Copyright (c) 1984-94 by The MathWorks, Inc.

% The Fehlberg coefficients:
alpha = [1/4  3/8  12/13  1  1/2]';
beta  = [ [    1      0      0     0      0    0]/4
          [    3      9      0     0      0    0]/32
          [ 1932  -7200   7296     0      0    0]/2197
          [ 8341 -32832  29440  -845      0    0]/4104
          [-6080  41040 -28352  9295  -5643    0]/20520 ]';
gamma = [ [902880  0  3953664  3855735  -1371249  277020]/7618050
          [ -2090  0    22528    21970    -15048  -27360]/752400 ]';
pow = 1/5;
if nargin < 5, tol = 1.e-6; end
if nargin < 6, trace = 0; end

% Initialization
t = t0;
hmax = (tfinal - t)/16;
```

```
h = hmax/8;
y = y0(:);
f = zeros(length(y),6);
chunk = 128;
tout = zeros(chunk,1);
yout = zeros(chunk,length(y));
k = 1;
tout(k) = t;
yout(k,:) = y.';

if trace
   clc, t, h, y
end

% The main loop

while (t < tfinal) & (t + h > t)
   if t + h > tfinal, h = tfinal - t; end

   % Compute the slopes
   temp = feval(ypfun,t,y);
   f(:,1) = temp(:);
   for j = 1:5
      temp = feval(ypfun, t+alpha(j)*h, y+h*f*beta(:,j));
      f(:,j+1) = temp(:);
   end

   % Estimate the error and the acceptable error
   delta = norm(h*f*gamma(:,2),'inf');
   tau = tol*max(norm(y,'inf'),1.0);

   % Update the solution only if the error is acceptable
   if delta <= tau
      t = t + h;
      y = y + h*f*gamma(:,1);
      k = k+1;
      if k > length(tout)
         tout = [tout; zeros(chunk,1)];
         yout = [yout; zeros(chunk,length(y))];
      end
      tout(k) = t;
      yout(k,:) = y.';
   end
   if trace
      home, t, h, y
   end

   % Update the step size
   if delta ~= 0.0
      h = min(hmax, 0.8*h*(tau/delta)^pow);
   end
end

if (t < tfinal)
```

```
    disp('Singularity likely.')
    t
end

tout = tout(1:k);
yout = yout(1:k,:);

end

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

function ypfun = orbitODE(t,y)
% Implements the ODE from Homework 2 Problem 3 as a first order system
global steps;
steps = steps + 1;
mu = 0.012277471;
muhat = 1.0 - mu;
D1 = ((y(1) + mu)^2 + y(2)^2)^1.5;
D2 = ((y(1) - muhat)^2 + y(2)^2)^1.5;
ypfun = zeros(size(y));
ypfun(1) = y(3);
ypfun(2) = y(4);
ypfun(3) = y(1) + 2*y(4) - muhat*((y(1)+mu)/D1) - mu*((y(1)-muhat)/
D2);
ypfun(4) = y(2) - 2*y(3) - (muhat*(y(2)/D1)) - (mu*(y(2)/D2));
end

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

Published with MATLAB® R2016b