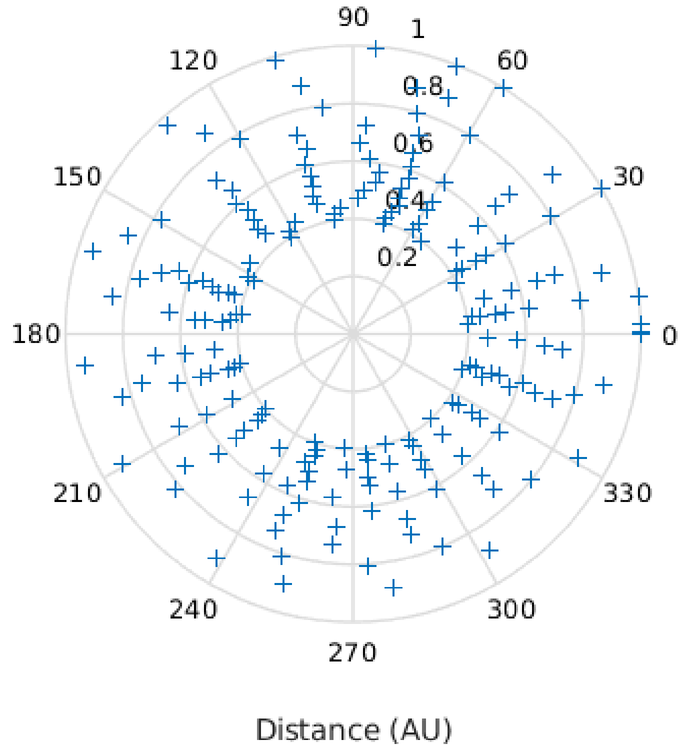
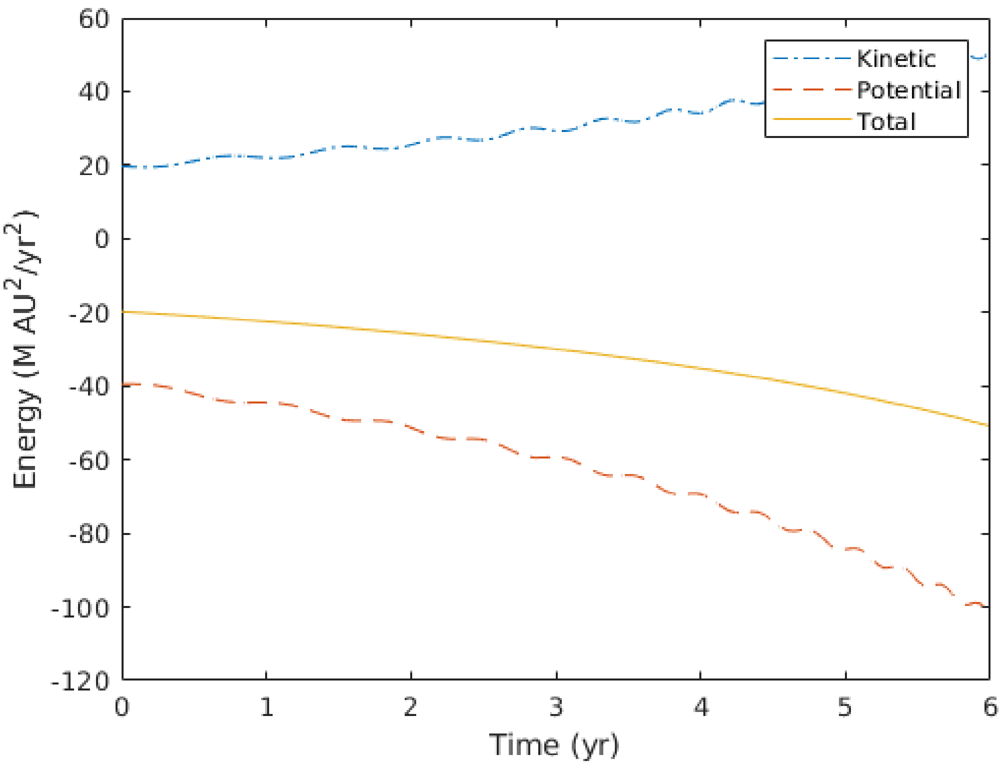
```
%Problem 2 - compute the orbit of a comet.

type gravrk.m
clear all; % Clear memory and print header

%* Set initial position and velocity of the comet.
r0 = 1; % AU
v0 = 2 * pi; % AU/yr
r = [r0 0];  v = [0 v0];
state = [ r(1) r(2) v(1) v(2) ];   % Used by R-K routines

%* Set physical parameters (mass, G*M)
GM = 4*pi^2;       % Grav. const. * Mass of Sun (au^3/yr^2)
mass = 1.;         % Mass of comet
C = (GM * mass) / (100 * (r0 * v0)^2); % constant for drag force
param = [GM, C];
adaptErr = 1.e-3; % Error parameter used by adaptive Runge-Kutta
time = 0;

%* Loop over desired number of steps using specified
%  numerical method.
nStep = 200;
tau = 0.001;
NumericalMethod = 4; % use adaptive RK4
for iStep=1:nStep

  %* Record position and energy for plotting.
  rplot(iStep) = norm(r);          % Record position for polar plot
  thplot(iStep) = atan2(r(2),r(1));
  tplot(iStep) = time;
  kinetic(iStep) = .5*mass*norm(v)^2;   % Record energies
  potential(iStep) = - GM*mass/norm(r);

  %* Calculate new position and velocity using desired method.
  if( NumericalMethod == 1 )
    accel = -GM*r/norm(r)^3;
    r = r + tau*v;             % Euler step
    v = v + tau*accel;
    time = time + tau;
  elseif( NumericalMethod == 2 )
    accel = -GM*r/norm(r)^3;
    v = v + tau*accel;
    r = r + tau*v;             % Euler-Cromer step
    time = time + tau;
  elseif( NumericalMethod == 3 )
    state = rk4(state,time,tau,'gravrk',GM);
    r = [state(1) state(2)];   % 4th order Runge-Kutta
    v = [state(3) state(4)];
    time = time + tau;
  else
    [state time tau] = rka(state,time,tau,adaptErr,'gravrk',param);
    r = [state(1) state(2)];   % Adaptive Runge-Kutta
```

```
    v = [state(3) state(4)];
  end

end

%* Graph the trajectory of the comet.
figure(1); clf;  % Clear figure 1 window and bring forward
polar(thplot,rplot,'+');  % Use polar plot for graphing orbit
xlabel('Distance (AU)');  grid;
pause(1)   % Pause for 1 second before drawing next plot

%* Graph the energy of the comet versus time.
figure(2); clf;   % Clear figure 2 window and bring forward
totalE = kinetic + potential;   % Total energy
plot(tplot,kinetic,'-.',tplot,potential,'--',tplot,totalE,'-')
legend('Kinetic','Potential','Total');
xlabel('Time (yr)'); ylabel('Energy (M AU^2/yr^2)');

% plot kinetic energy vs orbit number


function deriv = gravrk(s,t,param)
%  Returns right-hand side of Kepler ODE; used by Runge-Kutta routines
%  Inputs
%    s      State vector [r(1) r(2) v(1) v(2)]
%    t      Time (not used)
%    GM     Parameter G*M (gravitational const. * solar mass)
%  Output
%    deriv  Derivatives [dr(1)/dt dr(2)/dt dv(1)/dt dv(2)/dt]

% get parameters
GM = param(1);
C = param(2);

%* Compute acceleration
r = [s(1) s(2)];  % Unravel the vector s into position and velocity
v = [s(3) s(4)];
Fd = -C * norm(v) * v; % acceleration from Fd is just Fd because m=1
accel = -GM*r/norm(r)^3 + Fd;  % Gravitational acceleration

%* Return derivatives [dr(1)/dt dr(2)/dt dv(1)/dt dv(2)/dt]
deriv = [v(1) v(2) accel(1) accel(2)];
return;
```



90
60
30
0
330
300
270
240
210
180
150
120
1
0.8
0.6
0.4
0.2
Distance (AU)



Kinetic
Potential
Total
Energy (M AU^2/yr^2)
Time (yr)

Published with MATLAB® R2019a