Table of Contents

n-Body Propagator

```
clear all;
close all;
clc;
home;
```

Solar System Data:

```
% Table of gravitational parameters:
muSun       = 1.327124400189e11;
muMercury   = 2.20329e4;
muVenus     = 3.248599e5;
muEarth     = 3.9860044188e5;
muMoon      = 4.90486959e3;
muMars      = 4.2828372e4;
muCeres     = 6.26325e1;
muJupiter   = 1.266865349e8;
muSaturn    = 3.79311879e7;
muUranus    = 5.7939399e6;
muNeptune   = 6.8365299e6;
muPluto     = 8.719e2;
muEris      = 1.1089e3;

% J2000 data (PV):
% Position:
RSun_J2000      = [-1.067706805381631E+06 -4.182752718185146E+05
 3.086181725478008E+04];
RMercury_J2000  = [-2.052943316392625E+07 -6.733155053453046E+07
 -3.648992526181437E+06];
RVenus_J2000    = [-1.085242008576727E+08 -5.303290245135058E+06
 6.166496117013918E+06];
REarth_J2000    = [-2.756674048064499E+07 1.442790215211286E+08
 3.025066782881320E+04];
RMoon_J2000     = [-2.785834886487951E+07 1.440040417795086E+08
 6.652186445663124E+04];
RMars_J2000     = [2.069805421179929E+08 -2.425286841879086E+06
 -5.125451306189817E+06];
RCeres_J2000    = [-3.570100661713269E+08 1.185847406508015E+08
 6.929550952765751E+07];
RJupiter_J2000  = [5.974999178522581E+08 4.391864046755430E+08
 -1.519599985574219E+07];
```

```
RSaturn_J2000   = [9.573176521108806E+08 9.824380076870195E+08
 -5.518211788151336E+07];
RUranus_J2000   = [2.157907331373991E+09 -2.055043522898880E+09
 -3.559460196112704E+07];
RNeptune_J2000  = [2.513978490096488E+09 -3.739132780869018E+09
 1.906330132972622E+07];
RPluto_J2000    = [-1.478398655393869E+09 -4.182993264891145E+09
 8.752463989487143E+08];
REris_J2000     = [1.322247531122232E+10 4.602025019881923E+09
 -3.903660777776308E+09];
% Velocity:
VSun_J2000      = [9.312571926508239E-03 -1.282475570795343E-02
 -1.633507186347347E-04];
VMercury_J2000  = [3.700430442865286E+01 -1.117724068322721E+01
 -4.307791469481385E+00];
VVenus_J2000    = [1.391218600360602E+00 -3.515311993219235E+01
 -5.602056889533600E-01];
VEarth_J2000    = [-2.978494749858966E+01 -5.482119695038260E+00
 1.843295966752478E-05];
VMoon_J2000     = [-2.914141610973326E+01 -6.213103677848060E+00
 -1.148803176307656E-02];
VMars_J2000     = [1.171984953605987E+00 2.628323979722416E+01
 5.221336703898150E-01];
VCeres_J2000    = [-6.196623898412994E+00 -1.834193788479624E+01
 5.778897659018130E-01];
VJupiter_J2000  = [-7.900547720232828E+00 1.114339277066948E+01
 1.307023308633424E-01];
VSaturn_J2000   = [-7.421900386834246E+00 6.723930997204315E+00
 1.775749426204376E-01];
VUranus_J2000   = [4.646586369459156E+00 4.614774391558801E+00
 -4.308124107669631E-02];
VNeptune_J2000  = [4.474858465459663E+00 3.063881605796575E+00
 -1.659044011083001E-01];
VPluto_J2000    = [5.269124016493589E+00 -2.669250607326493E+00
 -1.250716402199096E+00];
VEris_J2000     = [-3.431877929177132E-01 1.676377476595594E+00
 1.504390854069972E+00];

J2000 = cell(13,1);
J2000{1}.body  = {'Sun', muSun, RSun_J2000, VSun_J2000};
J2000{2}.body  = {'Mercury', muMercury, RMercury_J2000,
 VMercury_J2000};
J2000{3}.body  = {'Venus', muVenus, RVenus_J2000, VVenus_J2000};
J2000{4}.body  = {'Earth', muEarth, REarth_J2000, VEarth_J2000};
J2000{5}.body  = {'Moon', muMoon, RMoon_J2000, VMoon_J2000};
J2000{6}.body  = {'Mars', muMars, RMars_J2000, VMars_J2000};
J2000{7}.body  = {'Ceres', muCeres, RCeres_J2000, VCeres_J2000};
J2000{8}.body  = {'Jupiter', muJupiter, RJupiter_J2000,
 VJupiter_J2000};
J2000{9}.body  = {'Saturn', muSaturn, RSaturn_J2000, VSaturn_J2000};
J2000{10}.body = {'Uranus', muUranus, RUranus_J2000, VUranus_J2000};
J2000{11}.body = {'Neptune', muNeptune, RNeptune_J2000,
 VNeptune_J2000};
J2000{12}.body = {'Pluto', muPluto, RPluto_J2000, VPluto_J2000};
```

```
J2000{13}.body = {'Eris', muEris, REris_J2000, VEris_J2000};
```

Gather input:

```
n = input('Enter number of bodies:\n');
runTime = input('\nEnter propagation time (in seconds, multiple of
 60):\n');
stepSize = input('\nEnter step size (in seconds):\n');
entries = runTime/stepSize;
bodyData = cell(n, 1);
trajectoryData = cell(n, 1);
for indices = 1:n
    trajectoryData{indices}.nthTrajectory = zeros((runTime/stepSize),
 3);
end
ephemerides = cell(n+1,1);
ephemerides{(n+1)}.barycenterOLD = [0 0 0]; % barycenter vector
 relative to initial barycenter
ephemerides{(n+1)}.barycenterNEW = [0 0 0];
ephemerides{(n+1)}.VBarycenter = [0 0 0];

for manyIndices = 1:n
    ephemerides{manyIndices}.R_ij = [0 0 0];
    ephemerides{manyIndices}.r_ij = [0 0 0];
end

h = stepSize;

for index = 1:n
    bodyData{index}.bodyName = input('\n\nEnter body name:\n', 's');
    bodyData{index}.muBody   = input('\nEnter gravitational parameter:
\n');
    bodyData{index}.RBody    = input('\nEnter initial barycentric
 position:\n');
    bodyData{index}.VBody    = input('\nEnter initial barycentric
 velocity:\n');
    bodyData{index}.output   = zeros(entries, 3);
    bodyData{index}.f_n        = 0;
end % end for


bigMu = 0;

for s = 1:n
    bigMu = bigMu + bodyData{s}.muBody;
end
```

```
Error using input
Cannot call INPUT from EVALC.

Error in nBody_Propagator (line 73)
n = input('Enter number of bodies:\n');
```

Processing:

```
for index = 1:stepSize:(runTime + 1)

    if (index ~= 1)
        % Get radial vectors, distances between attracting bodies:
        for t = 1:n
            bodyData{t}.f_n = 0;
        end

        for i = 1:n
            for j = 1:n
                if (bodyData{j}.bodyName ~= bodyData{i}.bodyName)
                    ephemerides{i}.R_ij = {bodyData{j}.RBody -
 bodyData{i}.RBody, bodyData{j}.bodyName};  % radial vector R_ij
                    ephemerides{i}.r_ij = {norm(ephemerides{i}.R_ij),
 bodyData{j}.bodyName}; % radial distance r_ij
                    bodyData{i}.f_n     = bodyData{i}.f_n
 - (bodyData{j}.muBody/(ephemerides{i}.r_ij(1,1)^3)) *
 ephemerides{i}.R_ij(1,1);
                end % end if
            end % end for
        end % end for

        for k = 1:n
            % Velocity Verlet
            bodyData{k}.VBody = bodyData{k}.VBody + (h/2) *
 bodyData{k}.f_n; % v_n+1/2
            bodyData{k}.RBody = bodyData{k}.RBody + h *
 bodyData{k}.VBody; % y_n+1
        end % end for

        % Recalculate barycentric properties:
        for l = 1:n
            ephemerides{(n+1)}.barycenterNEW = bodyData{l}.muBody *
 bodyData{l}.RBody;
        end % end for
        ephemerides{(n+1)}.barycenterNEW = (1 / bigMu) *
 ephemerides{(n+1)}.barycenterNEW - ephemerides{(n+1)}.barycenterOLD;
        ephemerides{(n+1)}.VBarycenter = (  ephemerides{(n
+1)}.barycenterNEW - ephemerides{(n+1)}.barycenterOLD ) / h;

        % Update radial vectors, distances:
        bodyData{:}.f_n = 0;
        for i = 1:n
            for j = 1:n
                if (bodyData{j}.bodyName ~= bodyData{i}.bodyName)
                    ephemerides{i}.R_ij = {bodyData{j}.RBody -
 bodyData{i}.RBody, bodyData{j}.bodyName};  % radial vector R_ij
                    ephemerides{i}.r_ij = {norm(ephemerides{i}.R_ij),
 bodyData{j}.bodyName}; % radial distance r_ij
```

```
                    bodyData{i}.f_n     = bodyData{i}.f_n
 - (bodyData{j}.muBody/(ephemerides{i}.r_ij(1,1)^3)) *
 ephemerides{i}.R_ij(1,1);
                end % end if
            end % end for
        end % end for


        for m = 1:n
            bodyData{m}.VBody = bodyData{m}.VBody + (h/2) *
 bodyData{m}.f_n;
        end % end for

        % Get new barycentric frame data
        if (index ~= 1)
            for o = 1:n
                bodyData{o}.RBody = bodyData{o}.RBody - ephemerides{(n
+1)}.barycenterNEW;
                bodyData{o}.VBody = bodyData{o}.VBody - ephemerides{(n
+1)}.VBarycenter;
            end  % end for
        end % end if

        ephemerides{(n+1)}.barycenterOLD = ephemerides{(n
+1)}.barycenterNEW;
    end % end if

    % Get trajectory points:
    if (index == 1)
        for p = 1:n
            trajectoryData{p}.nthTrajectory(index,1) =
 bodyData{p}.RBody(1,1);
            trajectoryData{p}.nthTrajectory(index,2) =
 bodyData{p}.RBody(1,2);
            trajectoryData{p}.nthTrajectory(index,3) =
 bodyData{p}.RBody(1,3);
        end % end for
    else
        for q = 1:n
            trajectoryData{q}.nthTrajectory((index - 1)/stepSize +
 1,1) = bodyData{q}.RBody(1,1);
            trajectoryData{q}.nthTrajectory((index - 1)/stepSize +
 1,2) = bodyData{q}.RBody(1,2);
            trajectoryData{q}.nthTrajectory((index - 1)/stepSize +
 1,3) = bodyData{q}.RBody(1,3);
        end % end for
    end % end if

end % End processing loop
```

Plotting:

```
figure('Name', 'N-Body Propagator Solution (By: TVW)');
```

```
for moreIndices = 1:n
    plot3(trajectoryData{moreIndices}.nthTrajectory(:,1), ...
        trajectoryData{moreIndices}.nthTrajectory(:,2), ...
        trajectoryData{moreIndices}.nthTrajectory(:,3));
    hold on;
end
grid on;
title({'N-Body Propagator Solution', 'per Kasdin & Paley (2011), App.
 C, p. 656 "velocity Verlet" algorithm'});
xlabel('x');
ylabel('y');
zlabel('z');
```

Published with MATLAB® R2020b