Numerical Solution to the N-Body Problem Using MATLAB

Kyle H. Walker
The University of Colorado at Boulder, Boulder, CO, 80817

Differential equations representing the behavior of an n-body system were developed from the equation for gravitational force between two bodies. Substitution was used to reduce this system of differential equations. The equations were expanded from vector to scalar form, resulting in 6n equations. MATLAB code was developed to solve these equations, and the results were plot for the cases of n=2, n=3 and n=4. Additionally, initial conditions based on the Sun-Earth-Moon system were used to test the code.

Nomenclature

t	=	time
$\mathbf{F_n}$	=	Force on body n
$\mathbf{r_n}$	=	position of body n
$\dot{\mathbf{r}}_\mathbf{n}$	=	velocity of body n
$\ddot{\mathbf{r}}_\mathbf{n}$	=	acceleration of body n
G	=	gravitational constant
m_n	=	mass of body n

I. Introduction

The n-body problem refers to the interaction between an arbitrary number of bodies due to a forces between them that vary with the inverse square of the distances separating each body. Commonly these forces are either gravity or electromagnetic force. For cases involving three or more bodies, the n-body problem cannot practically be solved analytically, and numerical methods must be used.

II. Procedure

The gravitational force experienced by body "i" due to body "j" is

$$\mathbf{F_i} = G\frac{m_\mathrm{i} m_\mathrm{j}}{r_\mathrm{ij}}\left(\mathbf{r_j}-\mathbf{r_i}\right) \quad , \quad r_\mathrm{ij} = \left\|\mathbf{r_j}-\mathbf{r_i}\right\|$$

Generalized for "n" bodies, the force experienced by body "i" is

$$\mathbf{F_i} = G\,m_\mathrm{i}\sum_{j=1,\,j\neq i}^{j=n}\frac{m_\mathrm{j}}{r_\mathrm{ij}^3}\left(\mathbf{r_j}-\mathbf{r_i}\right)$$

Acceleration can be obtained by dividing out the mass of body "i".

$$\ddot{\mathbf{r}}_\mathbf{i} = G\sum_{j=1,\,j\neq i}^{j=n}\frac{m_\mathrm{j}}{r_\mathrm{ij}^3}\left(\mathbf{r_j}-\mathbf{r_i}\right)$$

The following substitution can be used to reduce the order of this set of differential equations.

$$\mathbf{y_1}, \mathbf{y_2}\ldots\mathbf{y_n} = \mathbf{r_1}, \mathbf{r_2}\ldots\mathbf{r_n}$$

$$\mathbf{y}_{n+1}, \mathbf{y}_{n+2} \ldots \mathbf{y}_{2n} = \mathbf{r}_1, \mathbf{r}_2 \ldots \mathbf{r}_n$$

This results in 2n first order differential vector equations. These equations can be expanded to 6n first order scalar differential equations that can be solved with a simple Runga Kutta numerical scheme.

$$\dot{\mathbf{y}}_i = \mathbf{y}_{n+i}$$

$$\dot{\mathbf{y}}_{n+i} = G \sum_{j=1,\, j \neq i}^{j=n} \frac{m_j}{y_{ij}^3} \left(\mathbf{y}_j - \mathbf{y}_i \right)$$

The full 18 scalar differential equations required for the n=3 case are developed in Appendix I. The MATLAB code used to solve these equations can be found in Appendix II.

The code was run for three sets of arbitrarily selected initial conditions and masses, one each for n=2, n=3 and n=4. In addition to this, a case was run with initial conditions and masses approximating the Sun-Earth-Moon system. The values chosen for each of the four cases are in Table 1 below.

		n=2	n=3	n=4	Sol
Time		2e4 seconds	6e4 seconds	1150 seconds	1 year
Body 1	$\mathbf{r_0}$ **(m)**	1e7, 1e7, 0	0,0,2e7	0,0,0	0,0,0
	$\mathbf{v_0}$ **(m/s)**	1e3, 1e3, 0	-2e3,0,0	5e5,5e5,5e5	0,0,0
	m (kg)	6e24	3e24	8e25	1.98892e30
Body 2	$\mathbf{r_0}$ **(m)**	(-1e7, -1e7, 0)	2e7,0,0	2e7,0,0	1.496e11,0,0
	$\mathbf{v_0}$ **(m/s)**	(-1e3, -1e3, 0)	0,1e3,0	0,0,0	0,2.977e4,0
	m (kg)	6e24	2e24	4e24	5.9742e24
Body 3	$\mathbf{r_0}$ **(m)**		1e7,2e7,0	0,2e7,0	1.500e11,0,0
	$\mathbf{v_0}$ **(m/s)**		1e3,0,1e3	0,0,0	0,3.079e4,0
	m (kg)		1e24	4e24	7.36e22
Body 4	$\mathbf{r_0}$ **(m)**			0,0,2e7	
	$\mathbf{v_0}$ **(m/s)**			0,0,0	
	m (kg)			4e24	

Table 1 – Initial conditions for the 4 cases

III. Results

Figure 1 shows the plots of the n=2, n=3 and n=4 cases. The bodies in n=2 traced out two helix shapes.

The interaction between the bodies in case n=3 is chaotic.

The initial conditions selected for n=4 were axisymmetric about the 1,1,1 vector, with three bodies moving towards the body originating from 0,0,0.

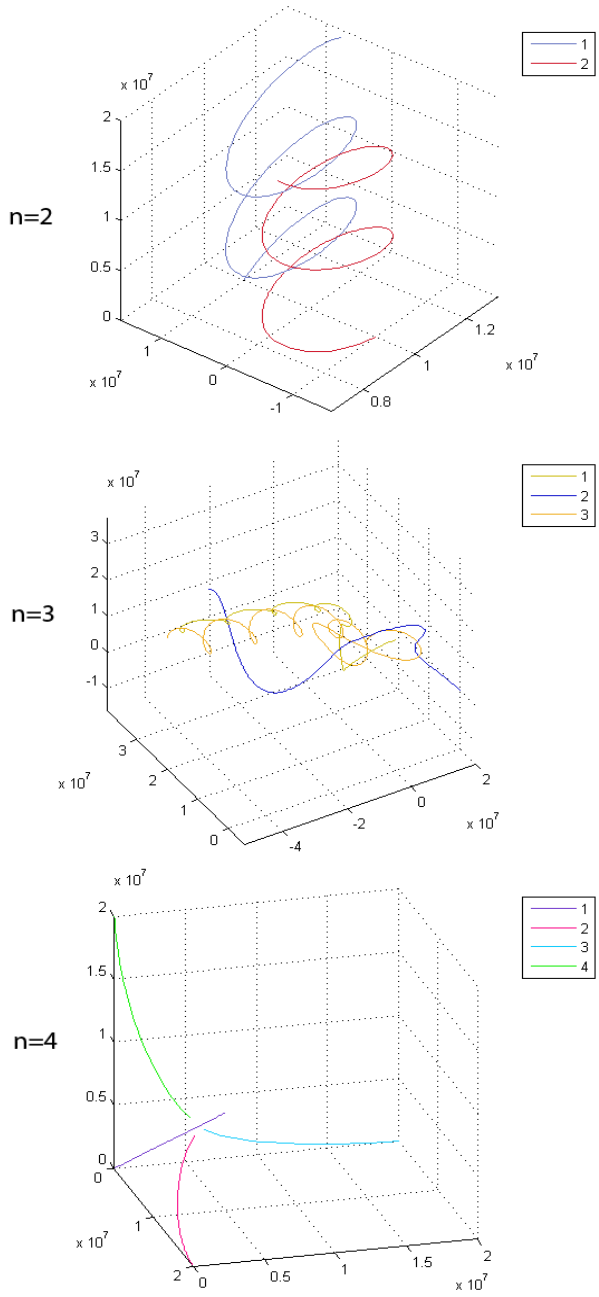

Figure 1: example plots for n=2, n=3 and n=3

The matlab function ode45 was unable to solve the system of equations that approximate a full year of the Sun-Earth-Moon system. The alternate differential equation solver, ode23, was able to handle the system.

Figure 2 shows a plot of the Sun-Earth-Moon system The motion of the Sun is too slight to be seen at this scale, and the Moon and Earth appear as a single line.

Figure 3 shows a one month simulation of the same system. The Moon orbits the earth one time.

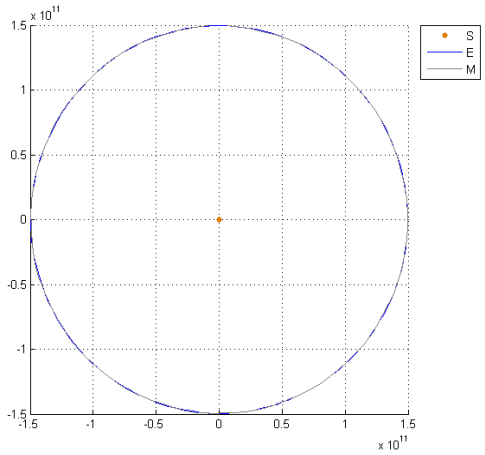

Figure 2: Sun-Earth-Moon

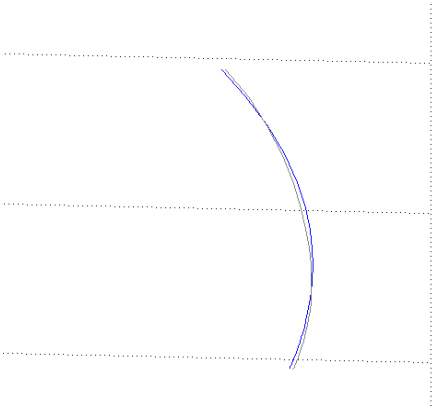

Figure 3: One month of the Sun-Earth-Moon system

Appendix I – Development of the 18 scalar ODE equations for solving a 3 body problem

The three vector differential equations for solving a 3 body problem are as follows.

$$\ddot{\mathbf{r}}_1 = G\left(\frac{m_2}{r_{12}^3}(\mathbf{r}_2-\mathbf{r}_1)+\frac{m_3}{r_{13}^3}(\mathbf{r}_3-\mathbf{r}_1)\right) \quad , \quad \ddot{\mathbf{r}}_2 = G\left(\frac{m_1}{r_{21}^3}(\mathbf{r}_1-\mathbf{r}_2)+\frac{m_3}{r_{23}^3}(\mathbf{r}_3-\mathbf{r}_2)\right)$$

$$\ddot{\mathbf{r}}_3 = G\left(\frac{m_1}{r_{31}^3}(\mathbf{r}_1-\mathbf{r}_3)+\frac{m_2}{r_{32}^3}(\mathbf{r}_2-\mathbf{r}_3)\right)$$

The following substitutions are made.

$$\mathbf{y}_1=\mathbf{r}_1, \quad \mathbf{y}_2=\mathbf{r}_2, \quad \mathbf{y}_3=\mathbf{r}_3, \quad \mathbf{y}_4=\dot{\mathbf{r}}_1, \quad \mathbf{y}_5=\dot{\mathbf{r}}_2, \quad \mathbf{y}_6=\dot{\mathbf{r}}_3$$

Resulting in

$$\dot{\mathbf{y}}_1=\mathbf{y}_4, \quad \dot{\mathbf{y}}_2=\mathbf{y}_5, \quad \dot{\mathbf{y}}_3=\mathbf{y}_6$$

$$\dot{\mathbf{y}}_4 = G\left(\frac{m_2}{y_{12}^3}(\mathbf{y}_2-\mathbf{y}_1)+\frac{m_3}{y_{13}^3}(\mathbf{y}_3-\mathbf{y}_1)\right) \quad , \quad \dot{\mathbf{y}}_5 = G\left(\frac{m_1}{y_{21}^3}(\mathbf{y}_1-\mathbf{y}_2)+\frac{m_3}{y_{23}^3}(\mathbf{y}_3-\mathbf{y}_2)\right)$$

$$\dot{\mathbf{y}}_6 = G\left(\frac{m_1}{y_{31}^3}(\mathbf{y}_1-\mathbf{y}_3)+\frac{m_2}{y_{32}^3}(\mathbf{y}_2-\mathbf{y}_3)\right)$$

These six vector equations can be expanded into the 18 scalar equations.

$$\begin{aligned} \dot{y}_{1,i}&=y_{4,i}, \quad \dot{y}_{1,j}=y_{4,j}, \quad \dot{y}_{1,k}=y_{4,k} \\ \dot{y}_{2,i}&=y_{5,i}, \quad \dot{y}_{2,j}=y_{5,j}, \quad \dot{y}_{2,k}=y_{5,k} \\ \dot{y}_{3,i}&=y_{6,i}, \quad \dot{y}_{3,j}=y_{6,j}, \quad \dot{y}_{3,k}=y_{6,k} \end{aligned}$$

$$\dot{y}_{4,i} = G\left(\frac{m_2}{y_{12}^3}(y_{2,i}-y_{1,i})+\frac{m_3}{y_{13}^3}(y_{3,i}-y_{1,i})\right)$$

$$\dot{y}_{4,j} = G\left(\frac{m_2}{y_{12}^3}(y_{2,j}-y_{1,j})+\frac{m_3}{y_{13}^3}(y_{3,j}-y_{1,j})\right)$$

$$\dot{y}_{4,k} = G\left(\frac{m_2}{y_{12}^3}(y_{2,k}-y_{1,k})+\frac{m_3}{y_{13}^3}(y_{3,k}-y_{1,k})\right)$$

$$\dot{y}_{5,i} = G\left(\frac{m_1}{y_{21}^3}(y_{1,i}-y_{2,i})+\frac{m_3}{y_{23}^3}(y_{3,i}-y_{2,i})\right)$$

$$\dot{y}_{5,j} = G\left(\frac{m_1}{y_{21}^3}(y_{1,j}-y_{2,j})+\frac{m_3}{y_{23}^3}(y_{3,j}-y_{2,j})\right)$$

$$\dot{y}_{5,k} = G\left(\frac{m_1}{y_{21}^3}(y_{1,k}-y_{2,k})+\frac{m_3}{y_{23}^3}(y_{3,k}-y_{2,k})\right)$$

$$\dot{y}_{6,i} = G\left(\frac{m_1}{y_{31}^3}(y_{1,i}-y_{3,i})+\frac{m_2}{y_{32}^3}(y_{2,i}-y_{3,i})\right)$$

$$\dot{y}_{6,j} = G\left(\frac{m_1}{y_{31}^3}\left(y_{1,j} - y_{3,j}\right) + \frac{m_2}{y_{32}^3}\left(y_{2,j} - y_{3,j}\right)\right)$$

$$\dot{y}_{6,k} = G\left(\frac{m_1}{y_{31}^3}\left(y_{1,k} - y_{3,k}\right) + \frac{m_2}{y_{32}^3}\left(y_{2,k} - y_{3,k}\right)\right)$$

Appendix II – Matlab Code

```
function nbody( bname, mass, r0, drdt0, t )
% Takes arrays of name, mass, initial pisition, initial velocity and time
% range.  Solves the n-body problem for these parameters and plots.

% get number of bodies
n = size(mass,1) ;

% concatenate initial conditions into single vector
y0 = [ r0(:,1); drdt0(:,1); r0(:,2); drdt0(:,2); r0(:,3); drdt0(:,3) ] ;

[t,y] = ode23(@nbody_ode,t,y0,[],mass) ;

% split y into position vectors to make it easier to plot
% r ( position, body number)
r_i = zeros(size(y,1),n) ;
r_j = r_i ;
r_k = r_i ;

for i=1:n
    r_i(:,i)  = y(:,i)      ;
    r_j(:,i)  = y(:,2*n+i)  ;
    r_k(:,i)  = y(:,4*n+i)  ;
end
%'Marker','.'
for i=1:n
    plot3( r_i(:,i), r_j(:,i), r_k(:,i),'Color',rand(1,3), ...
        'DisplayName',bname(i) )
    hold on
end
grid on
hold off
```

```
function ydot = nbody_ode(t,y,m)
% takes time, y vector containing positions and velocities, and masses
% returns ydot vector containing rates of change of the y vector in time

G = 6.673e-11 ;
n = size(y,1)/6 ;

% allocate arrays
ya_i = zeros(n,1) ;
yb_i = zeros(n,1) ;
ya_j = zeros(n,1) ;
yb_j = zeros(n,1) ;
ya_k = zeros(n,1) ;
yb_k = zeros(n,1) ;
yadot_i = zeros(n,1) ;
yadot_j = zeros(n,1) ;
yadot_k = zeros(n,1) ;
ybdot_i = zeros(n,1) ;
ybdot_j = zeros(n,1) ;
ybdot_k = zeros(n,1) ;

% split y into components to make it more readable
for i=1:n
    ya_i(i)  = y(i)      ;  % x component of r
    yb_i(i)  = y(n+i)    ;  % x component of dt/dt
    ya_j(i)  = y(2*n+i)  ;  % y component of r
    yb_j(i)  = y(3*n+i)  ;  % y component of dt/dt
    ya_k(i)  = y(4*n+i)  ;  % z component of r
    yb_k(i)  = y(5*n+i)  ;  % z component of dt/dt
end

% loop for each body, build ydot arrays
for i=1:n
    yadot_i(i) = yb_i(i) ;
```

```
    yadot_j(i) = yb_j(i) ;
    yadot_k(i) = yb_k(i) ;
    c = [ 0 0 0 ] ; % components of r''
    for j=1:n
        if ne(i,j) % exclude force from current body
            r_ij = [ ya_i(j) - ya_i(i), ...
                     ya_j(j) - ya_j(i), ...
                     ya_k(j) - ya_k(i) ] ;
            r_mag = ( r_ij(1)^2 + r_ij(2)^2 + r_ij(3)^2 )^.5 ;
            r_dir = (1/r_mag)*r_ij ;
            c = c + (G*m(j)/(r_mag^2))*r_dir ;
        end
    end
    ybdot_i(i) = c(1) ;
    ybdot_j(i) = c(2) ;
    ybdot_k(i) = c(3) ;
end

% concatenate ydot
ydot = [ yadot_i; ybdot_i; yadot_j; ybdot_j; yadot_k; ybdot_k ] ;
```

```
function twobodiestest

bname = [ '1'; '2' ] ;
mass = [ 6e24; 6e24 ] ;
r0 = [ 1e7 1e7 0; 1e7 -1e7 0 ] ;
drdt0 = [ 1000 1000 1000; -1000 -1000 1000 ] ;
time = [ 0 20000 ];

nbody( bname, mass, r0, drdt0, time )
```

```
function threebodiestest

bname = [ '1'; '2'; '3' ] ;
mass = [ 3e24; 2e24; 1e24 ] ;
r0 = [ 0 0 2e7 ; 2e7 0 0 ; 1e7 2e7 0 ] ;
drdt0 = [ -2000 0 0 ; 0 1000 0 ; 1000 0 1000  ] ;
time = [ 0 60000 ];

nbody( bname, mass, r0, drdt0, time )
```

```
function fourbodiestest

bname = [ '1'; '2'; '3'; '4' ] ;
mass = [ 8e25; 4e24; 4e24; 4e24 ] ;
r0 = [ 0 0 0 ; 2e7 0 0 ; 0 2e7 0 ; 0 0 2e7 ] ;
drdt0 = [ 5000 5000 5000 ; 0 0 0 ; 0 0 0 ; 0 0 0  ] ;
time = [ 0 1150 ];

nbody( bname, mass, r0, drdt0, time )
```

```
 function nbody_sol

bname = [ 'S'; 'E'; 'M' ] ;
mass = [ 1.98892e30; 5.9742e24; 7.36e22 ] ;
r0 = [ 0 0 0 ; 1.496e11 0 0 ; 1.496e11+4e8 0 0 ] ;
drdt0 = [ 0 0 0 ; 0 2.977e4 0 ; 0 2.977e4+1.02e3 0 ] ;
time = [ 0 1*2629872 ];

nbody( bname, mass, r0, drdt0, time )
```

```
function nbody_random
% generates random initial conditions for the n-body problem

% input parameters
n = 8 ;                 % number of bodies
m_range = [ 0 1e25 ] ;  % range of masses
g_range = [ -1e7 1e7 ] ;  % range of position components
v_range = [ -1e3 1e3 ] ;   % range of velocity components
t_range = [ 0 6000000 ] ; % time to integrate over

% generate initial conditions
bname = 1:1:n ;
mass = rand(n,1).*(m_range(2)-m_range(1)) + m_range(1) ;
r0 = rand(n,3).*(g_range(2)-g_range(1)) + g_range(1) ;
drdt0 = rand(n,3).*(v_range(2)-v_range(1)) + v_range(1) ;

nbody( bname, mass, r0, drdt0, t_range )
```