matlab实践报告：太阳系模型仿真

name，2024/4/20

1 引言

本文探讨了在天体物理学中使用N体模拟进行计算分析的动态演变，重点关注重力相互作用。我们的模型建立在牛顿的万有引力定律之上，该定律规定每个点质量与其他点质量之间的吸引力与它们的质量乘积成正比，与它们之间的距离的平方成反比。我们简化了天体为点质量，忽略了旋转动力学、相对论效应和非引力力等因素，从而简化了模型，同时保留了模拟N体系统轨道动力学所需的核心要素。

该模拟采用数值积分来随时间演化每个天体的位置和速度，从而观察由引力相互作用引起的复杂动力学行为，例如轨道共振、混沌和多体系统的长期稳定性。在这个matlab计算机实践中，使用MATLAB实现了一个太阳系模型，用于演示利用傅里叶方法分析时间序列数据，并理解行星和恒星之间的相互作用。傅里叶方法经常用于分析周期性的时间序列数据（例如地震学）。

2 模型制定

该三维模型是用MATLAB实现的。MATLAB解决了行星的运动并将输出写入名为“数组”的数据列表中。它预测并存储了太阳系中所有行星和太阳的所有三个维度的位置和速度。该模型同时应用了牛顿的第二定律（$\vec{F} = m\vec{a}$）和万有引力定律：

$$\vec{F} = -G\frac{m_1 m_2}{r^3}\vec{r}$$

考虑一个质量 m_1 和质量 m_2 在空间中相距距离 $r_{1,2}$：

$$r_{1,2} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

注意，这只是两个点 (x_1, y_1, z_1) 和 (x_2, y_2, z_2) 之间的距离。牛顿的引力定律表明两个质量之间的力的大小 $|\vec{F_{1,2}}|$ 是：

$$|\vec{F_{1,2}}| = -G\frac{m_1 m_2}{r_{1,2}^2}$$

其中 $G = 6.67 \times 10^{-11}\ \mathrm{m^3\ kg^{-1}\ s^{-2}}$ 是引力常数。我们需要在三维空间中找到沿 x, y 和 z 轴作用的力，因此我们应用方程1得到：

$$F_{x,1,2} = -G\frac{m_1 m_2}{r_{1,2}^3} \times (x_2 - x_1)$$
$$F_{y,1,2} = -G\frac{m_1 m_2}{r_{1,2}^3} \times (y_2 - y_1)$$
$$F_{z,1,2} = -G\frac{m_1 m_2}{r_{1,2}^3} \times (z_2 - z_1)$$

3 模型基础

在本模型中，我们以牛顿的万有引力定律为基础，该定律规定每个点质量通过作用于连接两点的线上的力互相吸引。该力与它们的质量乘积成正比，与它们之间的距离的平方成反比：

我们根据牛顿万有引力定律得到了行星间的相互作用力。在两个质量之间的力是质量之积与它们之间距离的平方的倒数成正比。基于此，我们得到了以下净力公式：

$$F_{x,1} = -\sum_{i \neq 1} \frac{G m_1 m_i}{r_{1,i}^3} \times (x_i - x_1)$$

这个公式描述了质量 m_1 在 x 轴方向上的受力情况。类似地，我们可以得到在其他轴向的力，并将其与质量和加速度相关联，从而得到了描述每个质量运动的方程。

其中F是两个质量之间的引力的大小，G是引力常数，m_1和m_2是物体的质量，r是它们中心之间的距离。为了进行这项模拟，我们将天体近似为点质量，忽略了旋转动力学、相对论效应和非引力力等因素，这使得对它们轨道动力学的简化但深刻的探索成为可能。

这种方法虽然降低了复杂性，但保留了模拟N体系统轨迹和相互作用所需的核心要素，例如太阳系或星团。

4 数值模拟

该模拟采用数值积分来随时间演化每个天体的位置和速度。在给定初始条件（所有天体的位置、速度和质量）的情况下，模拟在离散的时间步骤上迭代，计算每个天体受到来自其他每个天体的净引力，并相应地更新它们的速度和位置。尽管这种方法计算密集，但它能够检查由引力相互作用引起的复杂动力学行为，例如轨道共振、混沌和多体系统的长期稳定性。

本质上，我们的模拟是一个工具，用于研究受到重力影响的N体系统的天体力学，从而提供关于这些系统的结构和时间演变的洞察力。通过抽象出实现的具体细节，我们关注于所涉及的物理学以及表征这些迷人动力学系统的新兴行为。

4.1 数值积分

我们采用Verlet积分方法，因其简单和在系统内能量守恒方面的有效性而选择。位置和速度通过以下迭代更新：

$$\vec{r}(t + \Delta t) = \vec{r}(t) + \vec{v}(t)\Delta t + \frac{1}{2}\vec{a}(t)\Delta t^2$$

4.2 算法,初始条件和参数设置

初始位置和速度基于天文数据设置，系统的质心速度进行了修正以防止漂移。

表 1: 太阳系模拟的初始条件

天体	质量	X(AU)	Y(AU)	Z(AU)	Vx	Vy	Vz
太阳	332946.000	0.000	0.000	0.000	0.000	0.000	0.000
水星	0.055	0.081	-0.448	-0.044	0.022	0.006	-0.001
金星	0.815	-0.711	-0.111	0.040	0.003	-0.020	-0.000
地球	1.000	-0.486	-0.888	0.000	0.015	-0.008	0.000
月球	0.012	-0.486	-0.886	0.000	0.014	-0.008	0.000
火星	0.107	-1.479	0.764	0.052	-0.006	-0.011	-0.000
木星	317.800	4.483	2.109	-0.109	-0.003	0.007	0.000
土星	95.160	8.521	-4.833	-0.255	0.002	0.005	-0.000
天王星	14.540	12.950	14.776	-0.113	-0.003	0.002	0.000
海王星	17.150	29.798	-2.495	-0.635	0.000	0.003	-0.000
冥王星	0.002	16.581	-30.527	-1.527	0.003	0.001	-0.001
哈雷彗星	0.000	-19.927	27.173	-9.964	0.000	0.000	0.000

变量	默认值	描述
which_bodies	ALL_BODIES	要在计算中考虑哪些行星（和太阳）
which_interactions	INTERACT_WITH_SUN_ONLY	指定相互作用的天体
which_star	MOVING_SUN	是否允许太阳摆动或固定
which_output	FIRST	将变量保存为不同的名称
run_time	1000	模拟的长度，以地球年为单位
plot_figures	false	在进行动画时绘制图像为png文件
plot_stride	10	绘制时跳过的时间级数

表 2: 模拟参数

5 收敛性

详细描述了用于确保模拟结果准确性和可靠性的收敛性测试。解释了确定收敛性的标准，并通过图表展示了研究结果，最好是通过图表或表格。讨论了这些测试对模拟数据可信度的影响。运行名为unit3.py的Python脚本，使用输入文件$solar_system.txt$，将结果输出到$output.txt$，使用20个时间步长，每个时间步长为15，10个时间步长，每个时间步长为30，10个时间步长，每个时间步长为40，10个时间步长，每个时间步长为50。显然，在地球-月球-太阳-水星系统中，当以20或更多的时间步长观察约一年时，地球的轨迹不再收敛。

Algorithm 1 主要模拟算法

procedure MAIN

初始化模拟参数并读取输入文件 $solor_system.txt$

对 n_{step} 个时间步进行主要时间积分循环

使用牛顿的普遍引力定律计算初始力和势能：

$$F = \frac{G \cdot m_1 \cdot m_2}{r^2}$$
$$U = -\frac{G \cdot m_1 \cdot m_2}{r}$$

其中 G 是引力常数，m_1 和 m_2 是粒子的质量，r 是它们之间的距离。

初始化用于存储结果的数组：位置、速度、能量

对 n_{step} 个时间步进行主要时间积分循环

for $i = 0$ to $n_{\text{step}} - 1$ **do**

使用Verlet积分算法更新粒子位置：

$$\mathbf{r}_{\text{new}} = \mathbf{r}_{\text{old}} + \mathbf{v}_{\text{old}} \cdot dt + \frac{1}{2}\mathbf{a}_{\text{old}} \cdot dt^2$$

计算粒子之间的新分离和力

使用Verlet积分算法更新粒子速度：

$$\mathbf{v}_{\text{new}} = \mathbf{v}_{\text{old}} + \frac{1}{2}(\mathbf{a}_{\text{old}} + \mathbf{a}_{\text{new}}) \cdot dt$$

计算动能：

$$KE = \frac{1}{2}mv^2$$

计算总能量：$E_{\text{total}} = KE + U$

将输出数据写入文件

end for

计算能量偏差和近日点

绘制轨道和能量

通过线性回归拟合周期和半长轴来验证开普勒第三定律

end procedure

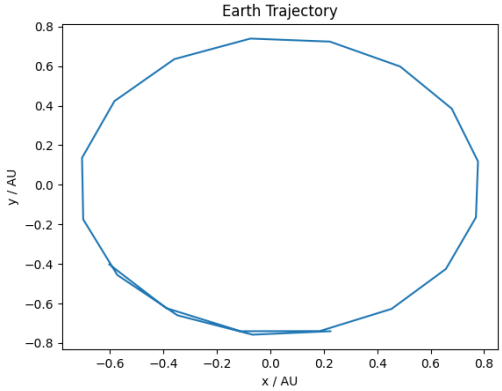

图 1: 20 * 15

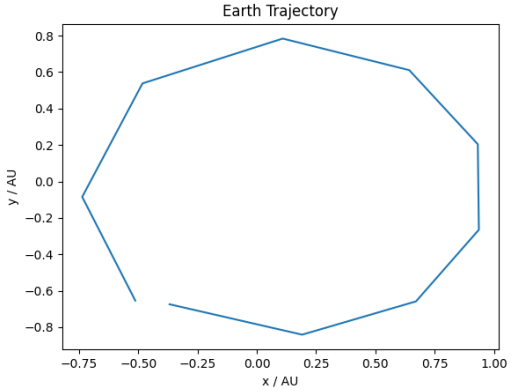

图 2: 10*30

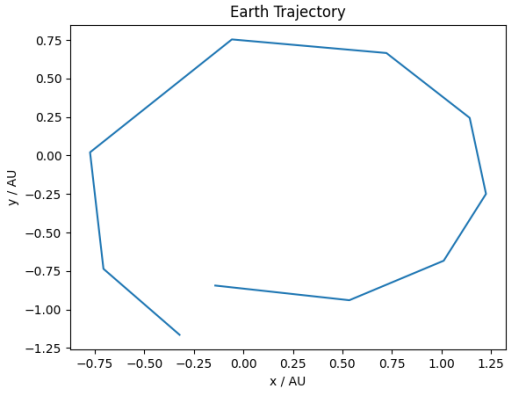

图 3: 10*40

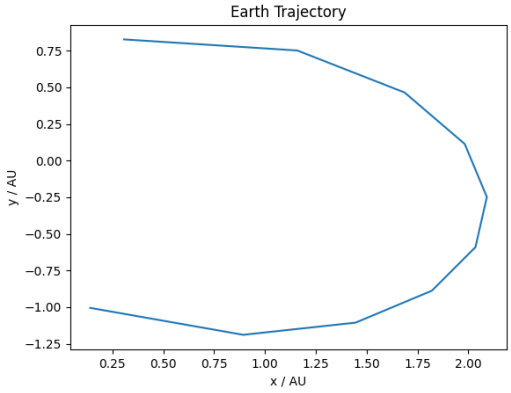

图 4: 10*50

显然，在地球-月球-太阳-水星系统中，当以20或更多的时间步长观察约一年时，地球的轨迹不再收敛。

6 结果

6.1 轨道

以下两个图表示在运行unit5.py 100,000天后太阳系中各个行星的轨迹。它们展示了整体视图和系统的近距离视图。

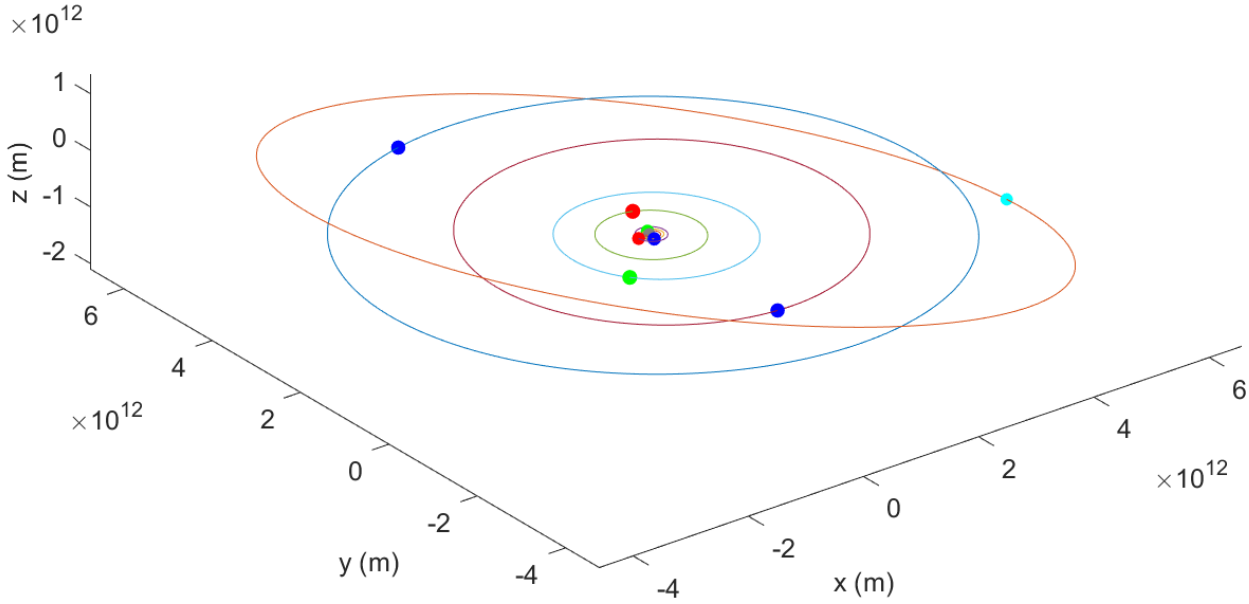

图 5: 太阳系轨迹图

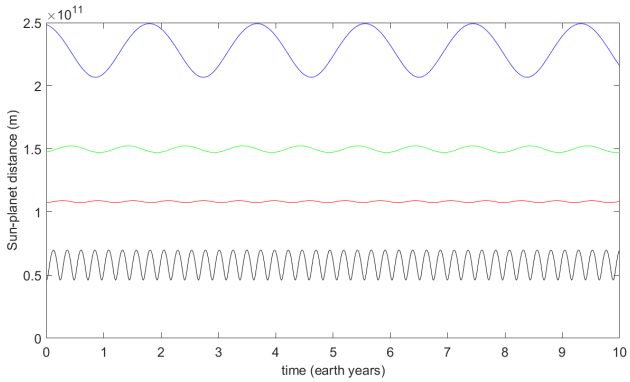

图 6: Inner solar sysytem 半径-时间变化图（10年）

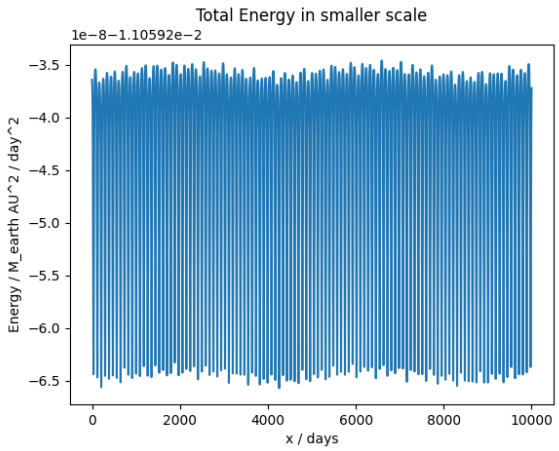

图 7: 地球的能量（10000天）

可以观察到模拟结果与实际情况相符。当太阳系完成365天时，地球恰好绕太阳完成一次轨道。

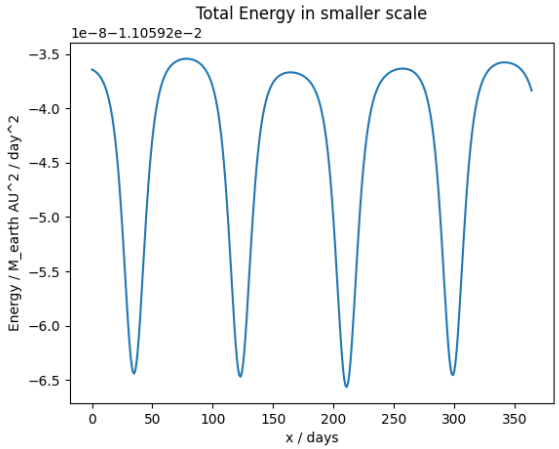

图 8: 地球的能量(一年）

从图中可以得出，在一年内，总能量大约有四个局部极小值，每个极小值约为-3.5，并且有四个局部极大值，每个极大值约为-6.5。

将时间尺度设置为10,000天后，可以观察到地球能量随时间的周期性变化，这是由其轨道运动引起的，叠加有更大频率调制。这种频率调制可能是由于地球在太阳系中的固有能量频率。

6.2 验证开普勒定律

我们将根据模拟结果验证开普勒第三定律，该定律表明 T^2 与 a^3 成正比，其中 T 是轨道周期，a 是半长轴（不是近日点!）。我们为每个行星生成了 T 与 a 的对数-对数图，发现它们大致遵循线性关系，表明与开普勒定律的良好一致性。然而，当将木星替换为质量增加20倍的超级木星时，开普勒定律逐渐失效。这是因为当木星的质量增加20倍时，周围的行星主要受到木星的影响，导致它们成为木星的卫星而不是遵循开普勒定律。

我们使用最小二乘法拟合了对数-对数图的数据点，并获得了斜率为0.71。根据开普勒定律，取对数后，$2\log(T)$ 应与 $3\log(a)$ 成正比，斜率为 $2/3 = 0.67$。因此，这种方法引入了一些误差，主要源于冥王星（倒数第二个数据点）。

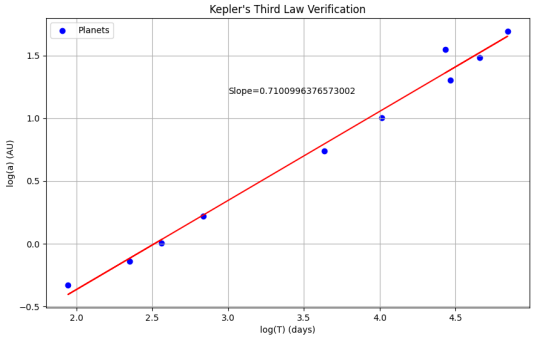

图 9: log(T)-log(a)

6.3 傅里叶变换

我们通过执行傅里叶变换来分析行星运动数据的周期性。傅里叶变换帮助我们识别了周期振荡的频率，并从中推断出行星的轨道周期。

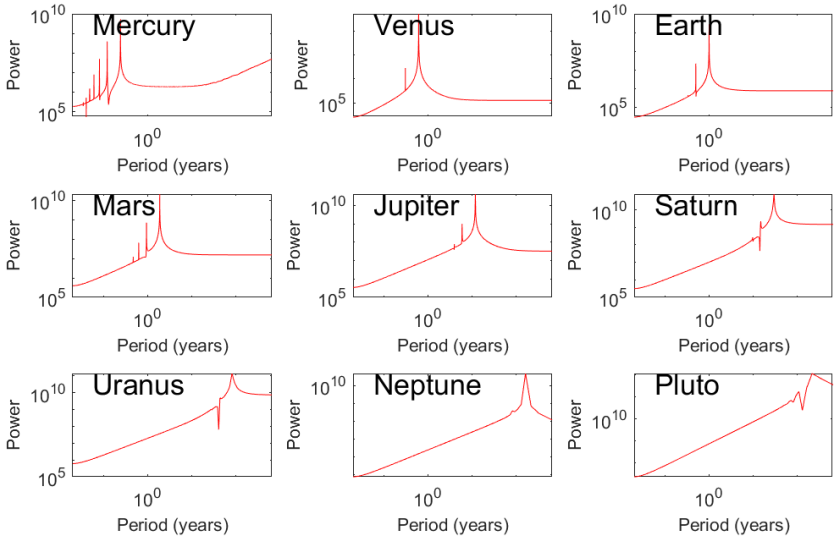

图 10: 傅里叶变换找到行星的本征频率

7 结论

总结了模拟的主要发现，强调它们的重要性以及它们在介绍中设定的目标达成的程度。反思了计算方法，突出了优点，并确定了任何限制或未来改进的领域。总的来说，我们在天体物理学中对N体模拟的计算分析提供了宝贵的见解，这些天体受到引力相互作用的控制。通过实施数值积分技术和对各种可观测量的检查，我们得出了一些关键的发现：

- 该模拟准确地重现了太阳系的预期轨道动力学，包括行星轨道、能量波动和轨道参数。
- 通过轨道周期与半长轴的对数-对数图验证开普勒第三定律，表明了良好的一致性，说明了我们的模拟在捕捉基本轨道关系方面的稳健性。
- 在用超级木星替换木星后观察到的开普勒第三定律的偏差突出了对大质量扰动体的轨道动力学的敏感性，强调了在多体系统中考虑引力相互作用的重要性。

我们的模拟方法虽然能够捕捉天体系统的总体动态，但存在一定的局限性和改进空间：

- 在我们的模型中进行的简化，如将天体视为点质量并忽略非引力力，限制了其适用于涉及旋转动力学、相对论效应和与外部天体的相互作用的更复杂情景。
- 与数值积分相关的计算成本可能会对模拟的时间和空间分辨率施加限制，特别是在轨道离心率较高或近距离相遇的情况下，可能会影响结果的准确性。
- 未来的改进可以包括完善模型以纳入额外的物理效应、优化计算算法以提高效率，并将模拟结果与天文观测或航天任务的观测数据进行验证。

总的来说，我们的计算分析为研究受引力相互作用控制的天体系统的复杂动态提供了有价值的工具，为理解基本的天体物理过程提供了见解，并为该领域的未来研究方向提供了信息。

参考文献

[1] Wikipedia contributors. “Solar System.” Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Solar_System.

[2] “Simulating Orbiting Planets in a Solar System using Python - Orbiting Planets Series 1,” The Python Coding Book, 29 Sep. 2021. https://thepythoncodingbook.com/2021/09/29/simulating-orbiting-planets-\in-a-solar-system-using-python-orbiting-planets-series-1/.

[3] “Visualize a Solar System with Python,” Python Geeks. https://pythongeeks.org/visualize-a-solar-system-with-python/.

[4] “Resource Library - Solar System,” National Geographic Society. https://education.nationalgeographic.org/resource/resource-library-solar-system/.

```
function [t,y,bodies,dat]=solve_solar_system(varargin)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function to solve for the motion of the planets in our solar system.    %
%                                                                          %
% Written by Dr Paul Connolly 2015 (copyright 2015).                       %
% Code provided as-is without any warranty.                                %
%                                                                          %
% Run by typing:                                                           %
% [t,y,bodies,dat]=solve_solar_system(1:10,trun);                           %
% where:                                                                   %
% 1:10 indicates to include interactions with all 10 bodies                %
% trun is the time in years you want the model to run for.                 %
%                                                                          %
% Or by typing:                                                            %
% [t,y,bodies,dat]=solve_solar_system(1,trun);                              %
% where:                                                                   %
% 1 indicates to include interactions with the sun only.                   %
% trun is the time in years you want the model to run for.                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% some variables to use (don't change these)++++++++++++++++++++++++++++++++
global n_bodies;
global G;
global m;
global Gm;
global interaction;
global interactions;
global time1;
global sign1;
global starti;
%--------------------------------------------------------------------------


% Stolen by star: n_bodies=11; mstar=5e30; zstar=1e9*1e3;
% Binary star system: n_bodies=11; mstar=m(1)./2; zstar=1e9.*1e3; uxstar=sqrt(F*zstar/(m));


% SET THE NUMBER OF BODIES +++++++++++++++++++++++++++++++++++++++++++++++++
% The order is: Sun, Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus,
% Neptune and Pluto, a total of 10 bodies
n_bodies=10; % This can be set to less than 10
if(nargin>0)
    n_bodies=varargin{1};
end
%--------------------------------------------------------------------------


% SET THE BODIES THAT A SINGLE BODY INTERACTS WITH (INCLUDING ITSELF!)+++++
% AND A FLAG FOR THE BODIES TO CONSIDER ++++++++++++++++++++++++++++++++++++
inds1=ones(n_bodies,1); %consider all bodies in the solar system
% inds1=[1 0 0 1 0 1 0 0 0 0 0]; % consider the sun, earth and jupiter
% inds1=[1 0 0 0 0 0 0 0 0 0 1]; % sun and other star (n_bodies=11)

% interactions should index the bodies being simulated
```

```
interactions=1:n_bodies; % this means all bodies interact with each other.
% interactions=1; % this means all bodies interact with body no 1, which is the sun.
if(nargin>1)
    interactions=varargin{2};
end
if(nargin>2)
    inds1=varargin{3};
end
%-------------------------------------------------------------------------


% SET THE TIME ARRAY TO GET SOLUTION ON IN YEARS ++++++++++++++++++++++++++
% Time to get solution on in years (default - set again based on input)++++
% tsol=[0:-0.01:-200];
tsol=[0:0.01:1000];
if(nargin>4)
    tsol=[0:0.01:varargin{5}];
end
%-------------------------------------------------------------------------

starti=1;
if nargin>3
    sun_flag=varargin{4};
    if sun_flag == 0
        starti=2;
    end
end


inds=find(inds1==1);


G=6.67e-11;
m=[1.989e30 5e30]; % the mass of the sun and other star (n_bodies=11)
zstar=1e12; % distance of other star away from sun (n_bodies=11)
%uxstar=sqrt(G.*m(1)./zstar);
uxstar=0.;


% INITIAL DATA FOR THE SOLAR SYSTEM (TAKEN FROM JPL EPHEMERIS) ++++++++++++
% the product of G and m for the bodies in the solar system
Gm=[G.*m(1)./1e9 22032.09 324858.63 398600.440 ...
    42828.3 126686511 37931207.8 ...
    5793966 6835107 872.4 G.*m(2)./1e9].*1e9;

% The positions (x,y,z) and the velocities (vx,vy,vz) of all the planets
x=[0 1.563021412664830E+07 -9.030189258080004E+07 -1.018974476358996E+08 ...
    -2.443763125844157E+08 -2.351654682753220 06E+08 -1.011712827283427E+09 ...
    2.934840841770302E+09 4.055112581124043E+09 9.514009594170194E+08 0].*1e3;
y=[0 4.327888220902108E+07 5.802615456116644E+07 1.065689158175689E+08 ...
    4.473211564076996E+07 7.421837640432589E+08 -1.077496255617324E+09 ...
    6.048399137411513E+08 -1.914578873112663E+09 -4.776029500570151E+09 0].*1e3;
z=[0 2.102123103174893E+06 6.006513603716755E+06 -3.381951053601424E+03 ...
    6.935657388967808E+06 2.179850895804323E+06 5.901251900068215E+07 ...
    -3.576451387567792E+07 -5.400973716179796E+07 2.358627841705075E+08 zstar./1e3].*1e3;
```

```
110
111 ux=[0 -5.557001175482630E+01 -1.907374632532257E+01 -2.201749257051057E+01 ...
112     -3.456935754608896E+00 -1.262559929908801E+01 6.507898648442419E+00 ...
113     -1.433852081777671E+00 2.275119229131818E+00 5.431808363374300E+00 uxstar./1e3].*1e3;
114 uy=[0 1.840863017229157E+01 -2.963461693326599E+01 -2.071074857788741E+01 ...
115     -2.176307370133160E+01 -3.332552395475581E+00 -6.640809674126991E+00 ...
116     6.347897341634990E+00 4.942356914027413E+00 -2.387056445508962E-02 0].*1e3;
117 uz=[0 6.602621285552567E+00 6.946391255404438E-01 1.575245213712245E-03 ...
118     -3.711433859326417E-01 2.962741332356101E-01 -1.434198106014633E-01 ...
119     4.228261484335974E-02 -1.548950389954096E-01 -1.551877289694926E+00 0].*1e3;
120 %--------------------------------------------------------------------------
121
122
123
124 % descriptor of the body being modelled++++++++++++++++++++++++++++++++++++
125 bodies={'Sun','Mercury','Venus','Earth','Mars','Jupiter','Saturn',...
126     'Uranus','Neptune','Pluto','Black hole'};
127
128
129 % Mean distance from the sun++++++++++++++++++++++++++++++++++++++++++++++
130 meandist=[7e8 5.79e10 1.082e11 1.496e11 2.279e11 7.783e11 1.426e12 ...
131     2.871e12 4.497e12 5.914e12 sqrt(x(end).^2+y(end).^2+z(end).^2)];
132
133 % This indexes the bodies by what we want to consider++++++++++++++++++++++
134 Gm=Gm(inds);
135 x=x(inds);
136 y=y(inds);
137 z=z(inds);
138 ux=ux(inds);
139 uy=uy(inds);
140 uz=uz(inds);
141 bodies=bodies(inds);
142 meandist=meandist(inds);
143
144 n_bodies=length(inds);
145 %--------------------------------------------------------------------------
146
147
148 % set x,y,z,vx,vy,vz for each consecutive planet and the tolerances for the
149 % solution
150 YINIT=[];
151 AbsTol=[];
152 for i=1:n_bodies
153     YINIT=[YINIT x(i) y(i) z(i) ...
154         ux(i) uy(i) uz(i)];
155     if(i==1)
156         AbsTol=[AbsTol [1./1e3
157                 1./1e3
158                 1./1e3
159                 1./1e6
160                 1./1e6
161                 1./1e6]'];
162     else
163         AbsTol=[AbsTol [sqrt(x(i).^2+y(i).^2+z(i).^2)./meandist(i)./1e6
164                 sqrt(x(i).^2+y(i).^2+z(i).^2)./meandist(i)./1e6
```

```
                sqrt(x(i).^2+y(i).^2+z(i).^2)./meandist(i)./1e6
                sqrt(ux(i).^2+uy(i).^2+uz(i).^2)./1e7
                sqrt(ux(i).^2+uy(i).^2+uz(i).^2)./1e7
                sqrt(ux(i).^2+uy(i).^2+uz(i).^2)./1e7]'];
    end
end
if(n_bodies==11) % for the black-hole, which has initial velocity zero (if simulated)
    AbsTol(end-5:end)=AbsTol(1:6)./100;
end
%-------------------------------------------------------------------------

% Pass options to the solver routine++++++++++++++++++++++++++++++++++++++
sign1=sign(tsol(end)-tsol(1));
options=odeset('RelTol',1e-6,'AbsTol',AbsTol,'OutputFcn',@myplotfun);
time1=0.;
% -------------------------------------------------------------------------


% Solve this problem using ODE113 ++++++++++++++++++++++++++++++++++++++++
[t,y]=ode113(@solar01,tsol.*365.25.*24.*3600,YINIT,options);
% -------------------------------------------------------------------------


% now parse y into something more human readable:++++++++++++++++++++++++++
for i=1:n_bodies
    eval(['dat.',bodies{i},'.xyz=y(:,[1:3]+(i-1).*6);']);
    eval(['dat.',bodies{i},'.uvw=y(:,[4:6]+(i-1).*6);']);
end
%-------------------------------------------------------------------------

% Function describing the time derivatives to be integrated+++++++++++++++++
function dy=solar01(t,y)

global n_bodies;
global G;
global m;
global Gm;
global interaction;
global interactions;
global starti;

dy=zeros(6.*(n_bodies),1);
% x, y, z, vx, vy, vz
for i=starti:n_bodies
    ind=interactions; % bodies we are considering interactions between
    i2=find(ind==i);  % a body can't interact with itself
    ind(i2)=[];

    % square of distance between this planet and the other objects
    R2=(y((i-1).*6+1)-y((ind-1).*6+1)).^2+...
        (y((i-1).*6+2)-y((ind-1).*6+2)).^2+...
        (y((i-1).*6+3)-y((ind-1).*6+3)).^2;
    % inverse square law between body i and the rest of them
    dy((i-1).*6+4)=dy((i-1).*6+4) ...
        -sum(Gm(ind)'./R2.*(y((i-1).*6+1)-y((ind-1).*6+1))./sqrt(R2));% dvx/dt
```

```
    dy((i-1).*6+5)=dy((i-1).*6+5) ...
        -sum(Gm(ind)'./R2.*(y((i-1).*6+2)-y((ind-1).*6+2))./sqrt(R2));% dvy/dt
    dy((i-1).*6+6)=dy((i-1).*6+6) ...
        -sum(Gm(ind)'./R2.*(y((i-1).*6+3)-y((ind-1).*6+3))./sqrt(R2));% dvz/dt

    % rate of change of position, because we are solving a second order ODE
    % - always the same
    dy((i-1).*6+1)=y((i-1).*6+4);
    dy((i-1).*6+2)=y((i-1).*6+5);
    dy((i-1).*6+3)=y((i-1).*6+6);
end
% --------------------------------------------------------------------


% Function to output time++++++++++++++++++++++++++++++++++++++++++++++++++
function status = myplotfun(t,y,flag)
global time1;
global sign1;
if(sign1==1) % check if the solution is going forward or backward-in-time
    if(t./86400./365.25>time1+1) % if last output was 1 year ago
        time1=time1+1;
        disp(['Time: ',num2str(t./86400./365.25),' yrs']);
    end
else
    if(t./86400./365.25<time1-1) % if last output was 1 year in future
        time1=time1-1;
        disp(['Time: ',num2str(t./86400./365.25),' yrs']);
    end
end
status=0.;
%--------------------------------------------------------------------
```

Listing 1: $solve_solar_system.m$

```
% Fourier analysis of the sun-planet distance for all the planets
figure
L=fix(0.6.*length(t));
NFFT=2^nextpow2(L);
Fs=1./abs((t(2)-t(1))./365.25./86400);
f = Fs/2*linspace(0,1,NFFT/2+1);

for i=2:length(bodies)
    subplot(3,3,i-1);

    Y=fft(sqrt(sum((y(:,[1:3]+6.*(i-1))-y(:,[1:3]+6.*(1-1))).^2,2))  ,NFFT)/L;

    plot(1./f,(2*abs(Y(1:NFFT/2+1))),'r')
    set(gca,'yscale','log','xscale','log')
    xlabel('Period (years)');ylabel('Power');
    text(0.1,0.9,bodies{i},'fontsize',15,'units','normalized');
end
```

Listing 2: $fourier_solar_system_one.m$

```
% Script to run the model for with and without interactions between planets
```

```
2
3  % SECTION 0: DO NOT CHANGE THESE ++++++++++++++++++++++++++++++++++++++++++
4  INTERACT_WITH_SUN_ONLY=1;
5  INTERACT_WITH_ALL_BODIES=2;
6  INTERACT_WITH_SUN_AND_JUPITER=3;
7
8  MOVING_SUN=1;
9  FIXED_SUN=0;
10
11 ALL_BODIES=1;
12 JUST_EARTH_SUN_AND_JUPITER=2;
13 ALL_BODIES_EXCEPT_JUPITER=3;
14
15 FIRST=1;
16 SECOND=2;
17 %--------------------------------------------------------------------------
18
19
20 % SECTION 1: USER INPUTS - CHANGE THESE++++++++++++++++++++++++++++++++++++
21 which_bodies=ALL_BODIES;
22 which_interactions=INTERACT_WITH_SUN_ONLY;
23 which_star=MOVING_SUN;
24 which_output=FIRST;
25 run_time=1000;
26 plot_figures=false;
27 plot_stride=10;
28 %--------------------------------------------------------------------------
29
30
31 % SECTION 2: PARSE USER INPUTS
32
33 % SELECT WHICH BODIES WE ARE CONSIDERING++++++++++++++++++++++++++++++++++++
34 switch which_bodies
35     case ALL_BODIES
36         bodies01=ones(1,10);
37     case JUST_EARTH_SUN_AND_JUPITER
38         bodies01=[1 0 0 1 0 1 0 0 0 0];
39     case ALL_BODIES_EXCEPT_JUPITER
40         bodies01=[1 1 1 1 1 0 1 1 1 1];
41     otherwise
42         disp('error which_bodies');
43         return;
44 end
45 %--------------------------------------------------------------------------
46
47
48 % SELECT WHICH BODIES INTERACT WITH WHICH++++++++++++++++++++++++++++++++++++
49 switch which_interactions
50     case INTERACT_WITH_SUN_ONLY
51         inters01=1;
52     case INTERACT_WITH_ALL_BODIES
53         inters01=1:10;
54     case INTERACT_WITH_SUN_AND_JUPITER
55         inters01=[1 6];
56     otherwise
```

```
        disp('error which_interactions');
        return;
end
%-------------------------------------------------------------------------



% SECTION 3: RUN THE MODEL
switch which_output
    case FIRST
    [t,y,bodies,dat]=solve_solar_system(10,inters01,bodies01,...
        which_star,run_time);
    case SECOND
    [t1,y1,bodies1,dat2]=solve_solar_system(10,inters01,bodies01,...
        which_star,run_time);
    otherwise
        disp('error which_output');
        return;
end
```

Listing 3: $run_solar_system.m$