Jack Dempsey

Mighty Big Problem

The setup of this problem is already known, so I will dismiss with an introductory section and move directly into describing the methods with which I approached it. Given that Mercury's orbit is a central force problem, and that my first task was to plot its orbit by itself, I'll begin with my derivation of the equations of motion and conversion of them to dimensionless form.

We begin with the following:

$$M_m \frac{d^2\overrightarrow{R_m}}{dt^2} = -GM_s M_m \frac{\overrightarrow{R_m}}{{R_m}^3}$$

Of course, in obtaining the denominator term on the right, we have taken the standard central force equation shown the direction using the unit vector of Mercury's position with respect to the force center (i.e., the sun).

Now we'll convert this to dimensionless form:

$$a * \frac{d^2\vec{\bar{r}}}{dt^2} = -GM_s \frac{\vec{r}}{a^3 * \bar{r}^3}$$

$$\frac{d^2\vec{r}}{dt^2} = \frac{-GM_s * \vec{r}}{a^3 * \bar{r}^3}$$

$$\tau = \sqrt{\frac{GM_s}{a^3}}$$

$$\frac{\tau^2 d^2\vec{r}}{d\bar{t}^2} = \frac{-GM_s * \vec{r}}{a^3 * \bar{r}^3}$$

$$\frac{d^2\vec{r}}{d\bar{t}^2} = \frac{-\vec{r}}{\bar{r}^3}$$

Now that we have this as our equation, it's easy enough to plug into MATLAB, breaking it up into Cartesian components. For initial conditions, we note that it will make things simpler if we start on an axis with purely tangential velocity, and therefore opt to start Mercury at its apogee. If we select our dimensional distance coefficient 'a' to be one astronomical unit, then we can simplify the conversion to dimensionless form by merely plotting in the actual distances. So then to get the initial velocity in dimensionless units, we exploit several known definitions of the energy of a given orbit.

$$E = \frac{-GM_sM_m}{\rho_p + \rho_a} = \frac{1}{2} * M_m * {v_y}^2 - \frac{GM_sM_m}{\rho_a}$$

where the perigee and apogee were determined by knowing that their sum must equal twice the semi-major axis (orbital radius), and that

$$\epsilon = \frac{\rho_a - \rho_p}{\rho_a + \rho_p}$$

We find that the initial y-velocity is given by

$$v_y = \sqrt{\frac{GM_s}{\rho_a} - \frac{GM_s}{\rho_p + \rho_a}}$$

Noting that our dimensionless velocity will be given by $a/\ \tau$ times our dimensional velocity, we find that we should start with Mercury moving at 1.2993 dimensionless velocity units, with a distance of .47034 (the apogee). For the record, the perigee is .30966.

Writing the code for this is straightforward enough, and is given in Appendix A. The plots of the orbit and of the angle at which the perihelion is found are given in figures 1 and 2. I'll also demonstrate the process of calling the function for a given time step, which will be done for example purposes here, but I will not include every iteration of this for all of the orbits plotted as part of this assignment. Note also that the first step taken here is to reduce the maximum allowable tolerance, which I did for each simulation I ran. We have not dealt with perihelions yet, and will return to this later.

The next challenge lies in adding in other planets. We note that each merely changes the effect on the second derivative of Mercury's position with respect to time by a term of

$$-GM_sM_m\frac{\vec{d}}{d^3}$$

where d is the distance between the planets. This is merely the result of the gravitational law applied to two massive bodies, and of course

$$d=\frac{\overrightarrow{R_m}-\overrightarrow{R_p}}{\left|\overrightarrow{R_m}-\overrightarrow{R_p}\right|^3}$$

This can then be used for all of the planets included, and put into dimensionless form in the same manner as above, though there will need to be a mass ratio of the mass of the planet to that of the sun. In our main simulations, I used Venus, Jupiter, and Earth, adding them in that order to our code. I wrote up functions for each combination of planets, though in fact the code is the same as long as the total number of planets is. Several examples are included in Appendix A, though all of the data discussed in this paper will have been generated using the four-planet model. Also, I should give a brief mention of the reasoning behind my choice of planets. Obviously, Jupiter has the largest effect given its mass, and I also opted to add Venus and Earth on account of their proximity to Mercury.

In our code, we have neglected the other planets effect on each other. This simplifies the calculations we're demanding of our machine, and is justified given the exceptionally slow rate of each planet's precession, excepting Mercury's. As a novelty, I have included a more complex function that describes a more complete simulation, in which the other planets pull on each other.

Before going on, it might be useful to stop and codify the initial conditions and mass ratios used for each planet. The latter are apparent in the code, but there's no harm in listing them all together. Each planet has been started at its apogee, and given only tangential velocity in the y-direction. All values are in dimensionless units. A final note about Venus' orbit: I did not bother to calculate its perigee and apogee, deeming its eccentricity small enough that I could treat it as circular.

Planet	Initial position (apogee)	Initial y-velocity	Mass Ratio
Mercury	.47034	1.2993	1.2e-7
Venus	.72	1.7851	2.45e-6
Jupiter	5.4496	.41796	9.5e-4
Earth	1.017	1.017	3.0e-6

Before discussing the plots we get from this, we'll first address the issue of tracking the perihelion. For this, we have a script entitled Perihelion, which parses through our array of orbital values and compares each value with the one before it, and has a Boolean marker to determine when we're seeking a perihelion for that iteration of our loop. If the difference is positive and we are still looking for that perihelion, we know that the previous value must have been the perihelion. This code is included in Appendix A after the planet functions. An example of calling this is given in our code for Mercury's orbit, at the start of Appendix A. This also shows the steps taken to generate our x and y values and our angles (in arcseconds), and to convert our time back to years.

My first attempt used a different script called 'div', which divided the total orbital array into chunks based on the time taken to complete a period. The data for this had several anomalous spikes and poorer values for our precession, and due to the need to iterate over the whole array multiple times, it took substantially more time to run. However, the numbers themselves are not terrible, and I believe that most of the errors could have been worked out with smaller timesteps and a little work cleaning up the transition between periods. It is included here for completeness, along with several plots made using its data.

In order to get the precession in arcseconds per century, then, we simply use the linear regression options MATLAB has as part of its standard package of plotting tools. This was confirmed by passing the times and angles of the perihelion to the polyfit() function, with the order of the fit set to 1.

For our planets-based simulation, I ran it with timesteps of .0001 for the equivalent of three and six centuries (note that one dimensionless unit of time is .15986 years, making one century around 625.547 dimensionless time units). Our most accurate simulation comes from running over one century with timesteps of .00001. The graphs of our orbits, perihelion position, and perihelion length are given in figures 3 through 9. Only one orbit has been provided because they look virtually identical, though the slightly thicker edge at the top and bottom fringe indicating the precession do get slightly more pronounced.

Moving on to our values, we see that we get steadily more accurate values for lower timespans and lower timesteps. Our six century gives us a precession of 545 arcseconds per century. While this is closer to the actual rate, the non-relativistic precession should actually be in the range of 520-530 arcseconds per century. We see for our one century simulation with timesteps reduced by a factor of 10, which gives us a precession of 528 arcseconds per century. As for the length of the perihelion, the range function gives us a total variation of 3.35e-05 AU. As we can see, the distance to the perihelion does not change much, but does change. It is not completely orderly, but it is roughly periodic. I believe that it roughly keeps time with the orbit of Jupiter, which has the greatest pull after the sun. The motion of Earth and Venus accounts for the remaining disorder. This disorderly, but roughly periodic, pattern can be seen in the position of the perihelion. It increases linearly with respect to time overall, but this pattern is made up of many smaller oscillatory motions caused by the other planets and their own orbits.

I mentioned before that we'd return to the motion of Mercury (figures 1 and 2) by itself, and this seems an appropriate place for that. We using our range function, we see that the distance hardly changes at all, as we would hope. However, there is actually a worrying amount of variability in the angle of the perihelion. However, over the whole period, we see that there is only a total precession of -.244 arcseconds per century. So while numerical error may be impacting individual values, its effect on the overall precession seems to be negligible.

Finally, before we move on to relativity, the graphs for the perihelion as determined by our 'div' method our given in figures 10 and 11, run using the data for our three century simulation. The spikes to which I alluded earlier are readily apparent here, and these distort the linear regression. However, we get a value of around 490 arcseconds/century, which is hardly terrible. Again, I believe that the loop being run is iterating close to correctly, but not quite. If it were to grab a value one over from the perihelion in changing cycles, this could account for a dramatic rise in the arctangent at that point. This analysis may seem unnecessary and obsessive, but I'm sure you understand the frustration that comes from an 'almost working' early attempt. I may continue to play with this some more over break.

Now then, the final part of the problem asks us to incorporate relativistic effects into our calculations. We note that the additional term breaks down immediately into dimensionless form when an a^2 term is pulled out from the top and the bottom, so we don't need to devote time to showing that. The code for this simulation is included in the final part of Appendix A.

Using three values of alpha (1.1e-.07, 2e-.07, and 5e-.07), I made the same measurements as before, and the graphs are shown in figures 12-18. As before, the orbits look very similar, though larger values of alpha due seem to produce thicker contours in places, as the precession occurs at a more rapid pace. Only one length variation plot has been shown, as there is so little change in the length of the perihelion that the graphs are exceptionally boring.

Most importantly, we've used the slopes of these graphs as the independent variable in another plot versus the corresponding values of alpha, and the curve for this fit gives us our linear function relating the precession rate to alpha (figure 19). The R^2 value for this fit ended up being around 1.00 (the code is shown), though this is less exceptional when you consider that we have only three data points. Regardless, according to the linear regression, for a value of alpha equal to 1.1e-08 AU^2, we should get a precession rate of right 42.2 arcseconds per century. This is even more accurate than our input of alpha by itself, which yielded only 41.9 arcseconds per century (figure 20). I believe this is due to the accumulation of numerical errors in solving the differential equation for such a tiny parameter. This is supported by the "fuzzy" nature of the graph tracking the perihelion's position over time for the actual value of alpha.

IHRTLUHC

Appendix A: Code

```
function [ derivs ] = MercOrb(t,x,b)
%Calculates the derivatives of Mercury's orbit, in the absence of other
```

```
%planets
%   Input:
% t = time
% x = (x-position, x-velocity, y-position, y-velocity)

temp = (x(1).^2+x(3).^2).^b;
derivs = [x(2);-x(1)./temp;x(4);-x(3)./temp];

end
```

```
EDU>> opt = odeset('AbsTol',1.0e-12,'RelTol',1.0e-9);
ic1 = [.47034;0;0;1.2993];
EDU>> [time,orb] = ode45(@MercOrb,[0.0:.0001:1251.09],ic1,opt1,1.5);
EDU>> x = orb(:,1);y= orb(:,3);
EDU>> plot(x,y)
EDU>> axis equal
EDU>> Perihelion
EDU>> T = T *.159860;
EDU>> angle = atand(periy./perix)*3600;
EDU>> plot(T,angle)
EDU>> plot(T,P)
EDU>> range(P)
ans =

   2.499948319467649e-07

EDU>> range(angle)

ans =

   1.313619105158305e+02
```

This is the main function used in our simulation. The following are just given as examples of the process of creating this.

```
function [ derivs ] = MercVJEOrb3(t,x)
%Calculates the derivatives of Mercury's orbit in the presence of Venus
% Jupiter
%   Input:
% t = time
% x = (x-position, x-velocity, y-position, y-velocity, x-position,
% x-velocity, y-position, y-velocity) - first for Mercury, then for
Venus,
% Jupiter, and Earth

temp1 = (x(1).^2+x(3).^2).^.5; % position vector for Mercury
temp1a = (x(1).^2+x(3).^2).^(1.5); % position vector raised to power
                                    % b = 1.5
```

```
temp2 = (x(5).^2+x(7).^2).^.5; % position vector for Venus
temp2a = (x(5).^2+x(7).^2).^(1.5); % position vector raised to power
                                   % b = 1.5

temp3 = (x(9).^2+x(11).^2).^.5; % position vector for Jupiter
temp3a = (x(9).^2+x(11).^2).^(1.5); % position vector raised to power
                                   % b = 1.5

temp4 = (x(13).^2+x(15).^2).^.5; % position vector for Earth
temp4a = (x(13).^2+x(15).^2).^(1.5); % position vector raised to power
                                   % b = 1.5

temp5a = (sqrt((x(1)-x(5)).^2+(x(3)-x(7)).^2)).^3; % absolute value of
                                    %the difference of
                               % positions of Mercury and Venus cubed
temp5b = (sqrt((x(1)-x(9)).^2+(x(3)-x(11)).^2)).^3; % absolute value of
                                    %the difference of
                               % positions of Mercury and Jupiter cubed
temp5c = (sqrt((x(1)-x(13)).^2+(x(3)-x(15)).^2)).^3; % absolute value
                                    %of the difference of
                                % positions of Mercury and Earth cubed

rat1 = 1.2e-7; % ratio of Mercury's mass to the sun's
rat2 = 2.45e-6; % ratio of Venus' mass to the sun's
rat3 = 9.5e-4; % ratio of Jupiter's mass to the sun's
rat4 = 3.0e-6; % ratio of Earth's mass to the sun's

derivs = [x(2);(-x(1)./temp1a)-(rat2.*((x(1)-x(5))/(temp5a)))-...
    ...
    (rat3.*((x(1)-x(9))/(temp5b)))-(rat4.*((x(1)-x(13))/(temp5c)));...
    ...
    x(4);(-x(3)./temp1a)-(rat2.*((x(3)-x(7))/(temp5a)))-...
    ...
    (rat3.*((x(3)-x(11))/(temp5b)))-(rat4.*((x(3)-x(15))/(temp5c)));...
    ...

    x(6); (-x(5)./temp2a);x(8); (-x(7)./temp2a);...

    x(10);-x(9)./temp3a;x(12);-x(11)./temp3a; ...

    x(14); (-x(13)./temp4a);x(16); (-x(15)./temp4a)];

end

function [ derivs ] = MercVOrb(t,x)
%Calculates the derivatives of Mercury's orbit, in the presence of
Venus,
%whose orbit is approximated as circular
%   Input:
% t = time
% x = (x-position, x-velocity, y-position, y-velocity, x-position,
% x-velocity, y-position, y-velocity) - first for Mercury, then for
Venus

temp1 = (x(1).^2+x(3).^2).^.5; % position vector for Mercury
temp1a = (x(1).^2+x(3).^2).^(1.5); % position vector raised to power
```

```
                                    % b = 1.5

temp2 = (x(5).^2+x(7).^2).^.5; % position vector for Venus
temp2a = (x(5).^2+x(7).^2).^(1.5); % position vector raised to power
                                    % b = 1.5

temp3 = (sqrt((x(1)-x(5)).^2+(x(3)-x(7)).^2)).^3; % absolute value of
the difference of
                               % positions of Mercury and Venus cubed

rat1 = 1.2e-7; % ratio of Mercury's mass to the sun's
rat2 = 2.45e-6; % ratio of Venus' mass to the sun's

derivs = [x(2);(-x(1)./temp1a)-(rat2.*((x(1)-x(5))/(temp3)));x(4); ...
    (-x(3)./temp1a)-(rat2.*((x(3)-x(7))/(temp3)));x(6); ...
    (-x(5)./temp2a)-(rat1.*((x(5)-x(1))/(temp3)));x(8); ...
    (-x(7)./temp2a)-(rat1.*((x(7)-x(3))/(temp3)))];

end

function [ derivs ] = MercVJOrb(t,x)
%Calculates the derivatives of Mercury's orbit in the presence of Venus
% Jupiter
%   Input:
% t = time
% x = (x-position, x-velocity, y-position, y-velocity, x-position,
% x-velocity, y-position, y-velocity) - first for Mercury, then for
Venus,
% and finally for Jupiter

temp1 = (x(1).^2+x(3).^2).^.5; % position vector for Mercury
temp1a = (x(1).^2+x(3).^2).^(1.5); % position vector raised to power
                                    % b = 1.5

temp2 = (x(5).^2+x(7).^2).^.5; % position vector for Venus
temp2a = (x(5).^2+x(7).^2).^(1.5); % position vector raised to power
                                    % b = 1.5

temp3 = (x(9).^2+x(11).^2).^.5; % position vector for Jupiter
temp3a = (x(9).^2+x(11).^2).^(1.5); % position vector raised to power
                                    % b = 1.5

temp4a = (sqrt((x(1)-x(5)).^2+(x(3)-x(7)).^2)).^3; % absolute value of
the difference of
                               % positions of Mercury and Venus cubed
temp5b = (sqrt((x(1)-x(9)).^2+(x(3)-x(11)).^2)).^3; % absolute value of
the difference of
                               % positions of Mercury and Jupiter cubed

rat1 = 1.2e-7; % ratio of Mercury's mass to the sun's
rat2 = 2.45e-6; % ratio of Venus' mass to the sun's
rat3 = 9.5e-4; % ratio of Jupiter's mass to the suns

derivs = [x(2);(-x(1)./temp1a)-(rat2.*((x(1)-x(5))/(temp4a)))-...
    (rat3.*((x(1)-x(9))/(temp4b)));x(4); ...
    (-x(3)./temp1a)-(rat2.*((x(3)-x(7))/(temp4a)))-...
    (rat3.*((x(3)-x(11))/(temp4b)));x(6); ...
    (-x(5)./temp2a)-(rat1.*((x(5)-x(1))/(temp4a)));x(8); ...
    (-x(7)./temp2a)-(rat1.*((x(7)-x(3))/(temp4a)));x(10);...
    -x(9)./temp3a;x(12);-x(11)./temp3a];

end

function [ derivs ] = MercVJEOrb(t,x)
%Calculates the derivatives of Mercury's orbit in the presence of Venus
% Jupiter
%   Input:
% t = time
% x = (x-position, x-velocity, y-position, y-velocity, x-position,
% x-velocity, y-position, y-velocity) - first for Mercury, then for
Venus,
% Jupiter, and Earth

temp1 = (x(1).^2+x(3).^2).^.5; % position vector for Mercury
temp1a = (x(1).^2+x(3).^2).^(1.5); % position vector raised to power
                                   % b = 1.5

temp2 = (x(5).^2+x(7).^2).^.5; % position vector for Venus
temp2a = (x(5).^2+x(7).^2).^(1.5); % position vector raised to power
                                   % b = 1.5

temp3 = (x(9).^2+x(11).^2).^.5; % position vector for Jupiter
temp3a = (x(9).^2+x(11).^2).^(1.5); % position vector raised to power
                                   % b = 1.5

temp4 = (x(13).^2+x(15).^2).^.5; % position vector for Earth
temp4a = (x(13).^2+x(15).^2).^(1.5); % position vector raised to power
                                   % b = 1.5

temp5a = (sqrt((x(1)-x(5)).^2+(x(3)-x(7)).^2)).^3; % absolute value of
                                    %the difference of
                               % positions of Mercury and Venus cubed
temp5b = (sqrt((x(1)-x(9)).^2+(x(3)-x(11)).^2)).^3; % absolute value of
                                    %the difference of
                               % positions of Mercury and Jupiter cubed
temp5c = (sqrt((x(1)-x(13)).^2+(x(3)-x(15)).^2)).^3; % absolute value
                                    %of the difference of
                                % positions of Mercury and Earth cubed
temp5d = (sqrt((x(5)-x(13)).^2+(x(7)-x(15)).^2)).^3% absolute value of
                                    %the difference of
                                % positions of Earth and Venus cubed

rat1 = 1.2e-7; % ratio of Mercury's mass to the sun's
rat2 = 2.45e-6; % ratio of Venus' mass to the sun's
rat3 = 9.5e-4; % ratio of Jupiter's mass to the sun's
rat4 = 3.0e-6; % ratio of Earth's mass to the sun's

derivs = [x(2);(-x(1)./temp1a)-(rat2.*((x(1)-x(5))/(temp5a)))-...
    (rat3.*((x(1)-x(9))/(temp5b)))-(rat4.*((x(1)-x(13))/(temp5c)));...
    x(4);(-x(3)./temp1a)-(rat2.*((x(3)-x(7))/(temp5a)))-...
    (rat3.*((x(3)-x(11))/(temp5b)))-(rat4.*((x(3)-x(15))/(temp5c)));...
    x(6); (-x(5)./temp2a)-(rat1.*((x(5)-x(1))/(temp5a)))-...
    -(rat4.*((x(5)-x(13))/(temp5d)));x(8); (-x(7)./temp2a)-...
    (rat1.*((x(7)-x(3))/(temp5a)))-(rat4.*((x(7)-x(15))/(temp5d)));...

    x(10);-x(9)./temp3a;x(12);-x(11)./temp3a;

    x(14); (-x(13)./temp4a)-(rat1.*((x(13)-x(1))/(temp5c)))-...
    -(rat2.*((x(13)-x(5))/(temp5d)));x(16); (-x(15)./temp4a)-...
    (rat1.*((x(15)-x(3))/(temp5c)))-(rat4.*((x(15)-x(7))/(temp5d)))];

end

function [ derivs ] = MercOrbR(t,x,a)
%Calculates the derivatives of Mercury's orbit, in the absence of other
%planets
%   Input:
% t = time
% x = (x-position, x-velocity, y-position, y-velocity)

temp = (x(1).^2+x(3).^2).^1.5;
derivs = [x(2);(-x(1)./temp).*(1+(a./(x(1).^2+x(3).^2)));...
    x(4);(-x(3)./temp).*(1+(a./(x(1).^2+x(3).^2)))];

end
```

Perihelion:

```
orbit = sqrt(x.^2+y.^2);
counter = 1;
B = true;
for i = 1:length(orbit)
    if i > 1 % This just gets us past the issue of possibly trying to
call
             % the orbit(0) element.
        if (orbit(i) > orbit(i-1))
            if B
                P(counter)=orbit(i-1);
                T(counter)=time(i-1);
                perix(counter) = x(i-1);
                periy(counter) = y(i-1);
                counter = counter + 1;
                B = false;
            end
        else
            B = true;
        end
    end
end
```

div:

```
orbit = sqrt(x.^2+y.^2);
period = 1.5/.0001;
periodi = 1; lower=1; periodn = 1;index = 1;
numperiods = floor(max(time)/1.5);
for i = 1:numperiods

    upper = periodn*period;
    for n = lower:upper
        temp(i,periodi)=orbit(n);
        periodi = periodi+1;
        if (orbit(n)==min(temp(i,:)))
            peri(i)=orbit(n);
            t(i)=time(n);
            px(i)=x(n);
            py(i)=y(n);
        end
    end
disp(peri(i))
periodi = 1;
lower = upper;
periodn = periodn + 1;
end
```

```
EDU>> p = polyfit(r,s,1);
EDU>> sfit = polyval(p,r);
EDU>> sresid = s - sfit;
EDU>> SSresid = sum(sresid.^2);
EDU>> SStotal = (length(s)-1) * var(s);
EDU>> rsq = 1 - SSresid/SStotal

rsq =

    1.0000
```

Appendix B: Figures

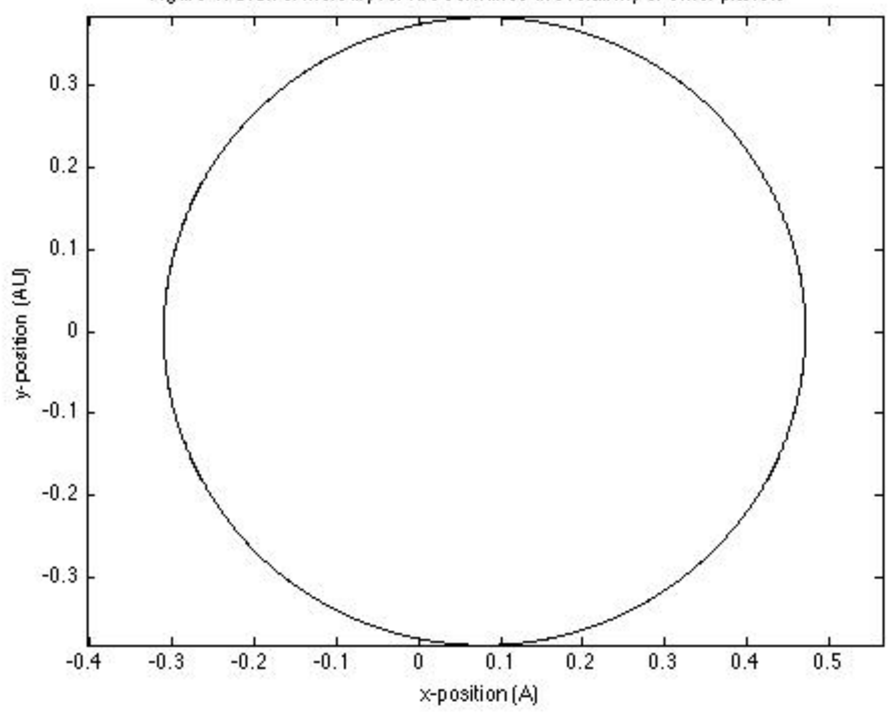


Figure 1: Orbit of Mercury for two centuries w/o relativity or other planets

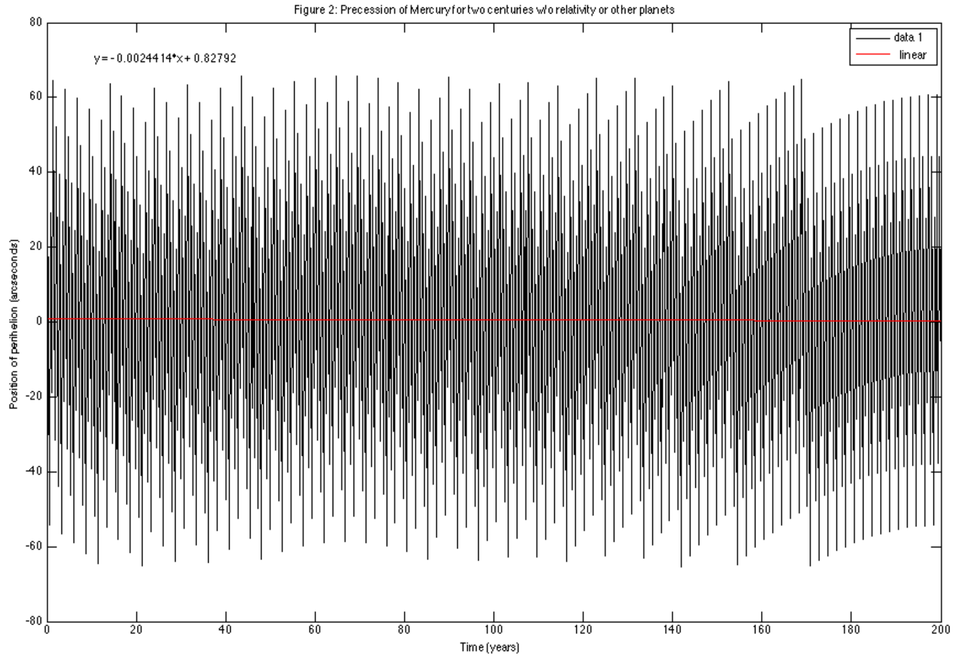


Figure 2: Precession of Mercury for two centuries w/o relativity or other planets

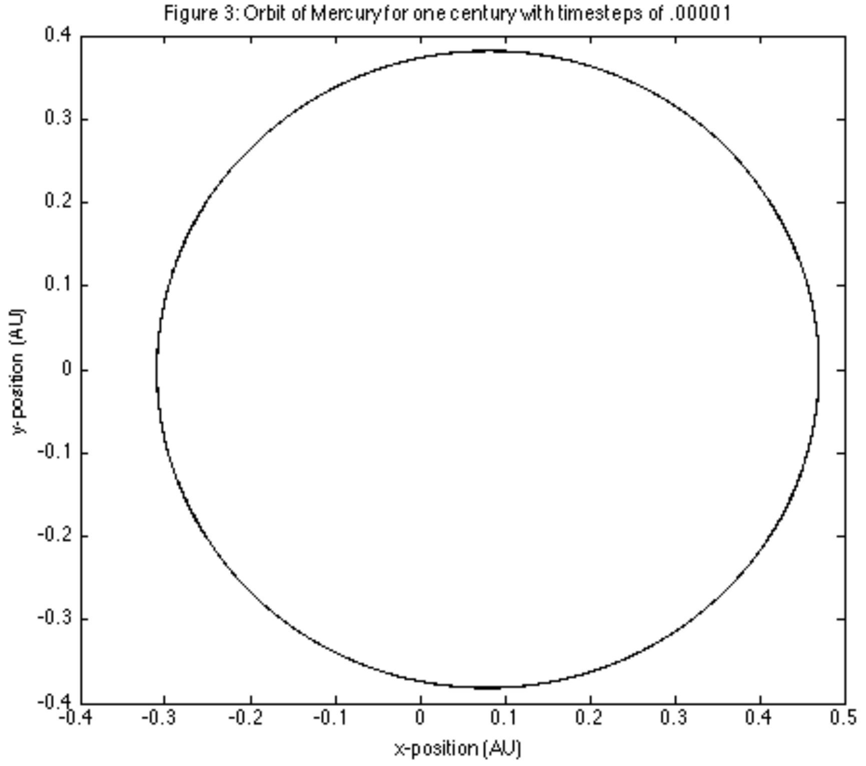


Figure 3: Orbit of Mercury for one century with timesteps of .00001

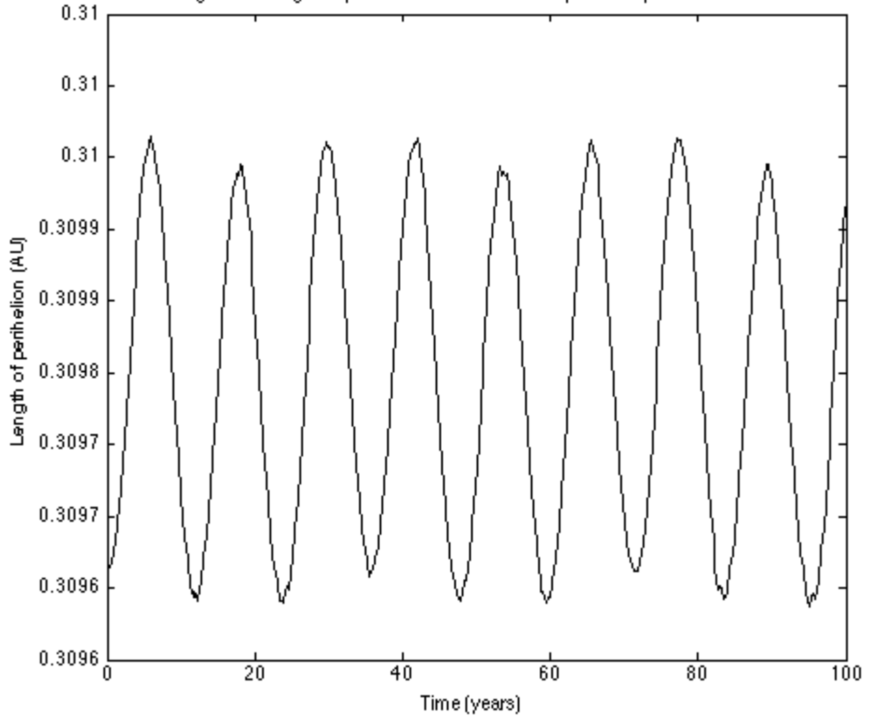


Figure 4: Length of perihelion over one century/timesteps of .00001

Figure 5: Precession of Mercury for one century; timesteps of .00001

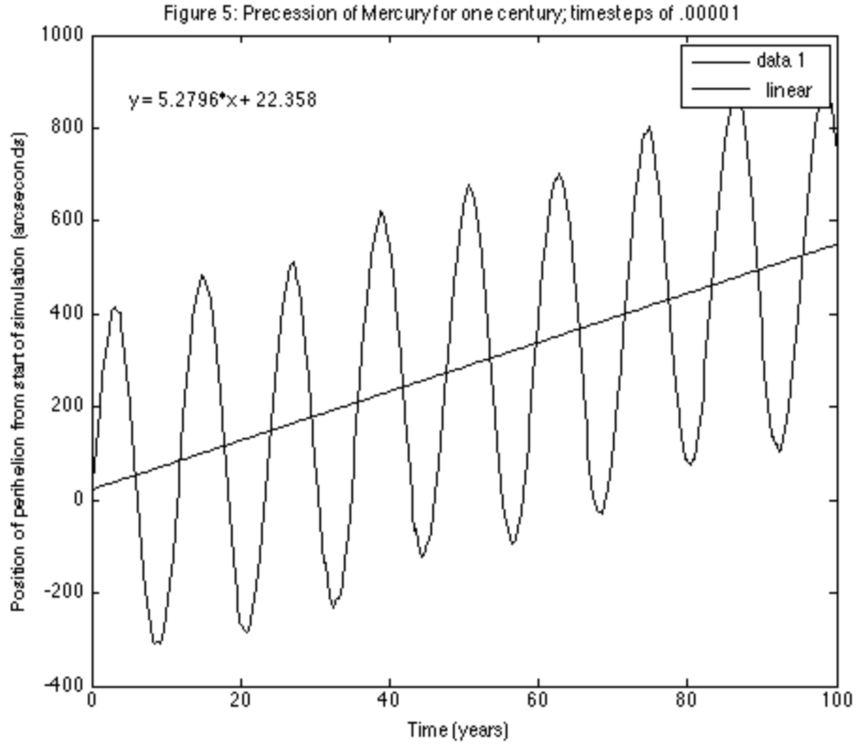


Figure 6: Length of perihelion over three centuries; timesteps of .0001

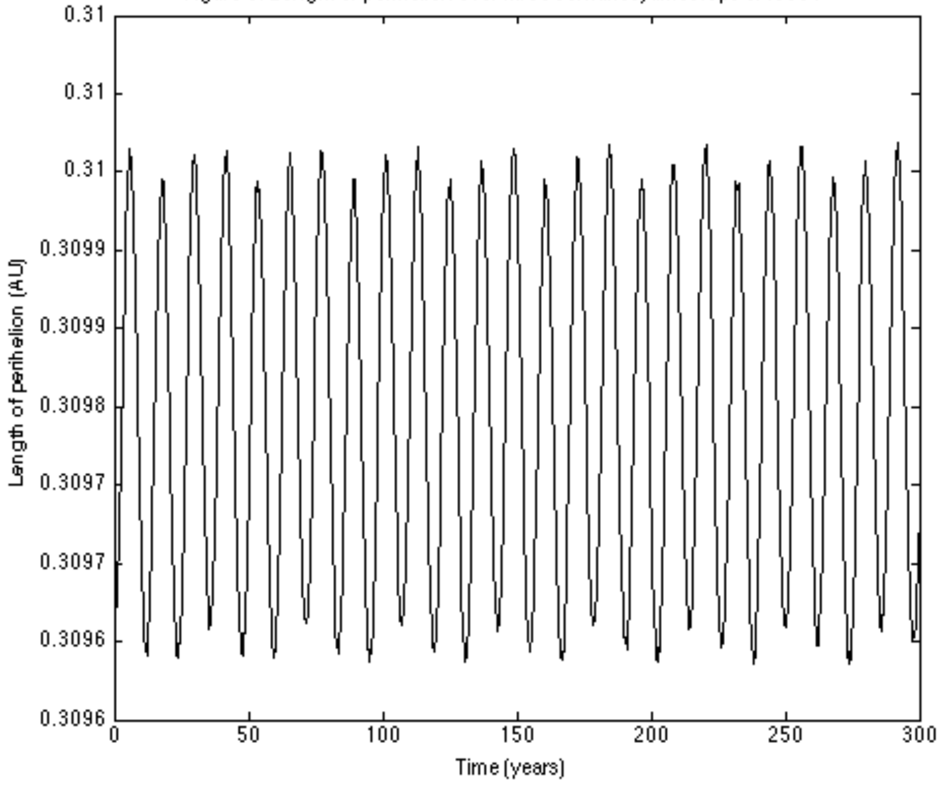


Figure 7: Location of perihelion over three centuries; timesteps = .0001

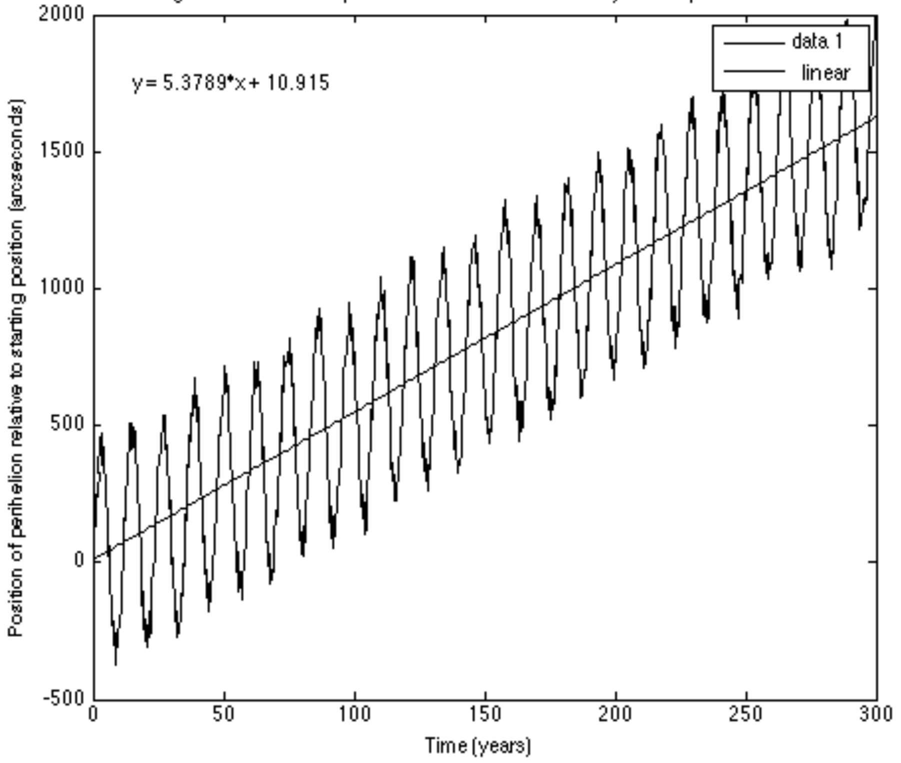


Figure 8: Length of perihelion over six centuries; timesteps of .0001

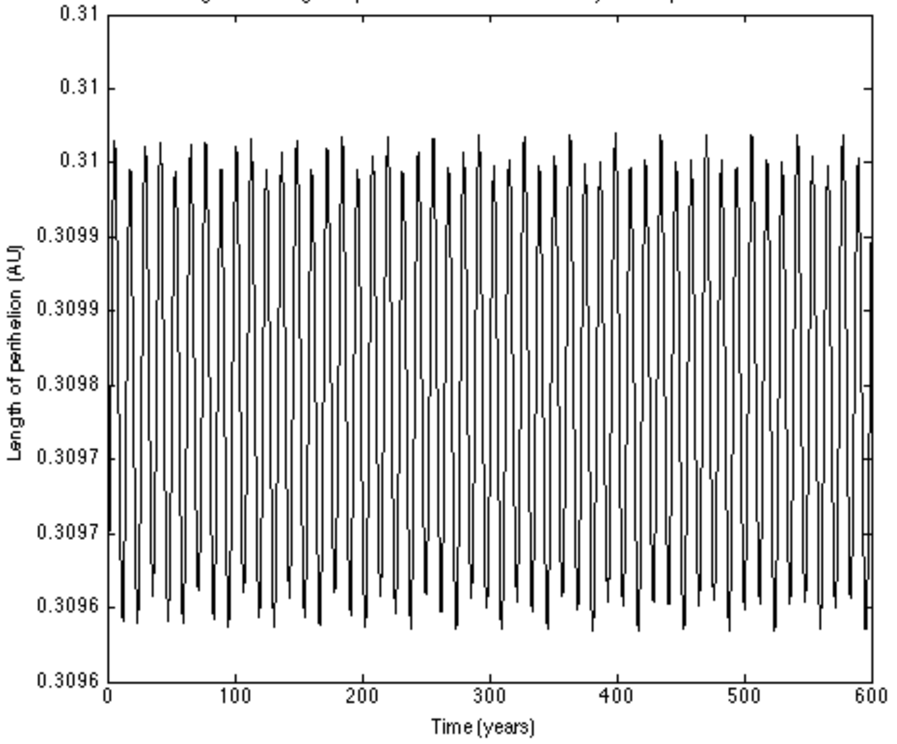


Figure 9: Position of perihelion over six centuries; timesteps of .0001

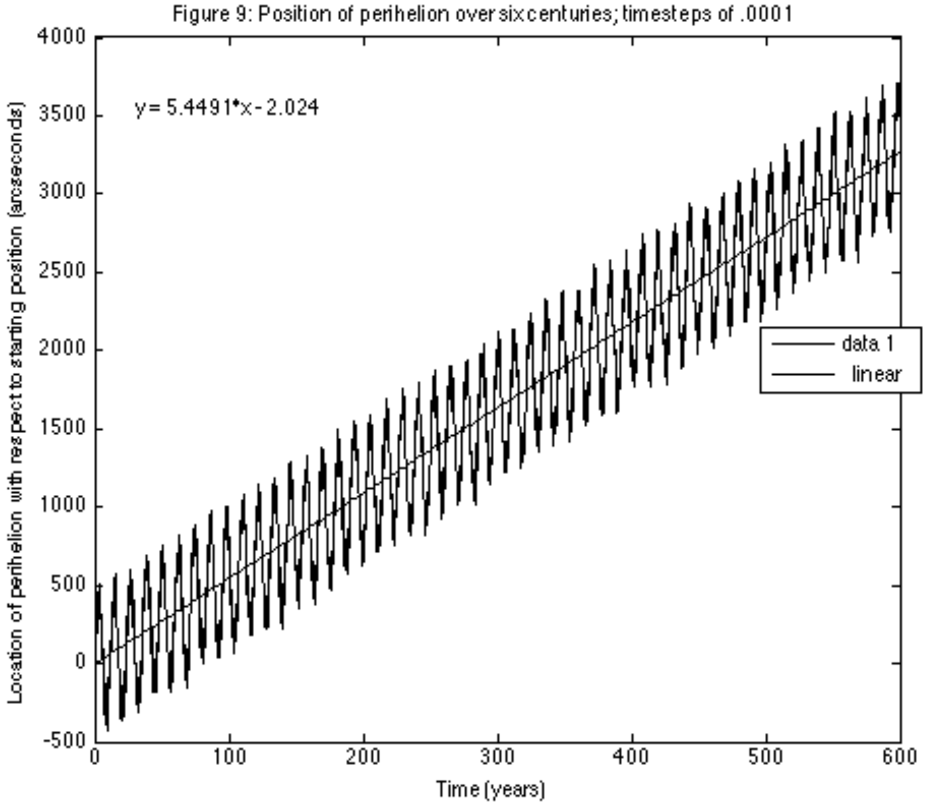


Figure 10: Perihelion length over three centuries using 'div'

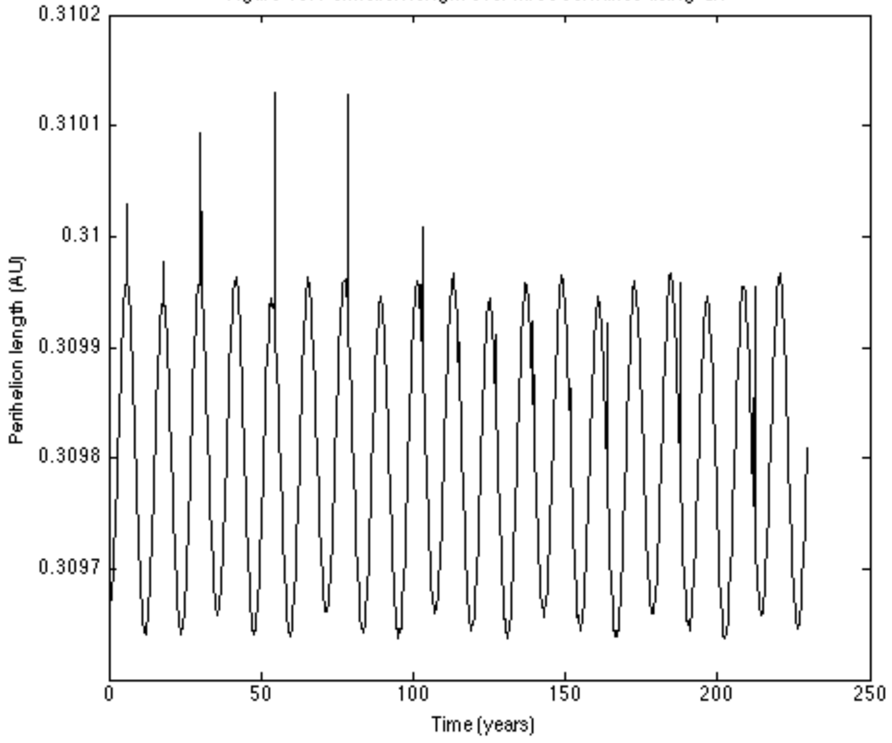


Figure 11: Precession of perihelion over three centuries using 'div'

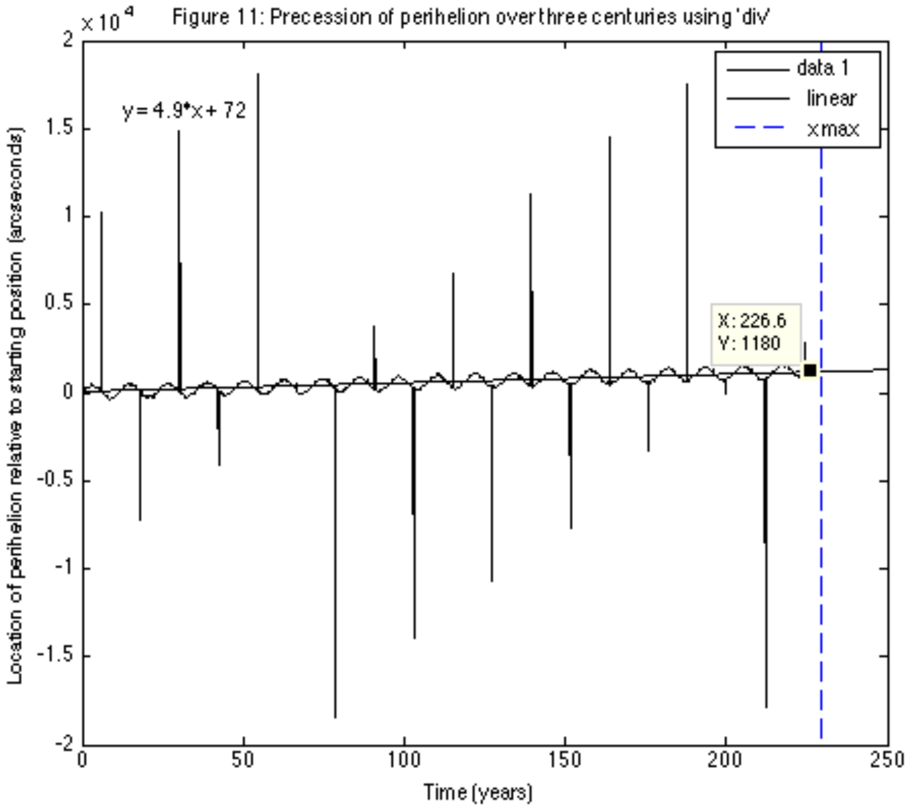


Figure 12: Relativistic orbit over one century; alpha = .00000011

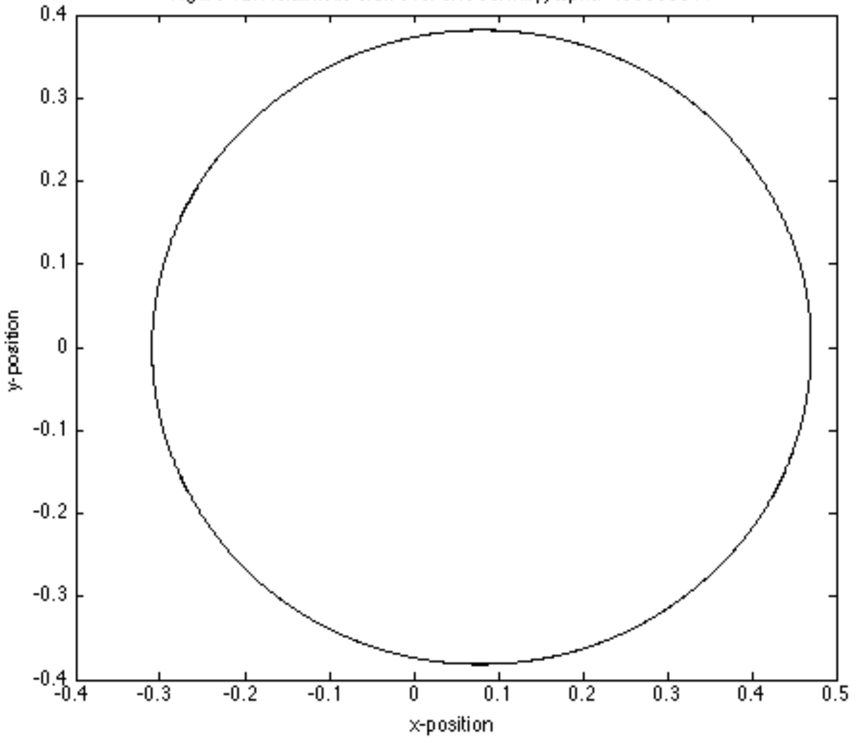


Figure 13: Relativistic orbit over one century for alpha = .0000002

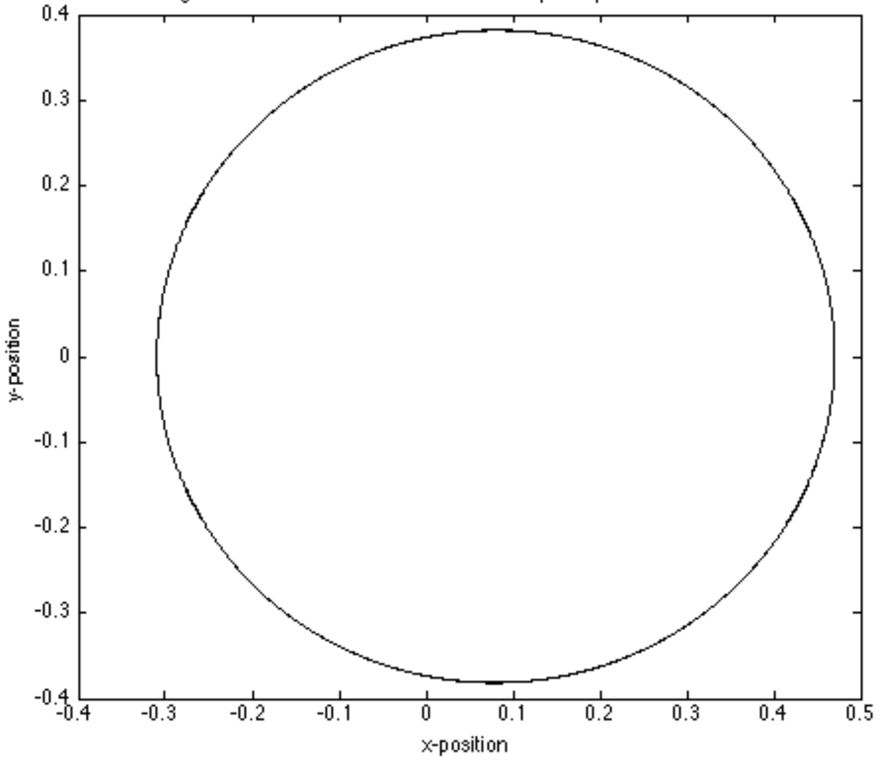


Figure 14: Relativistic orbit over one century for alpha = .0000005

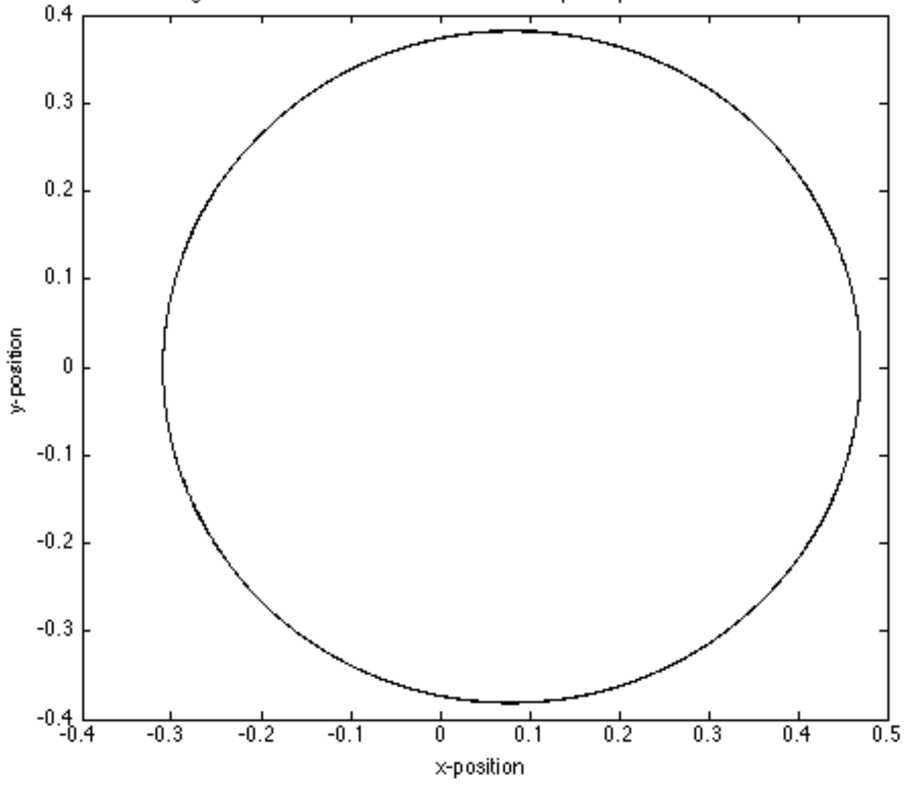


Figure 15: Length of perihelion over one century; alpha = .00000011

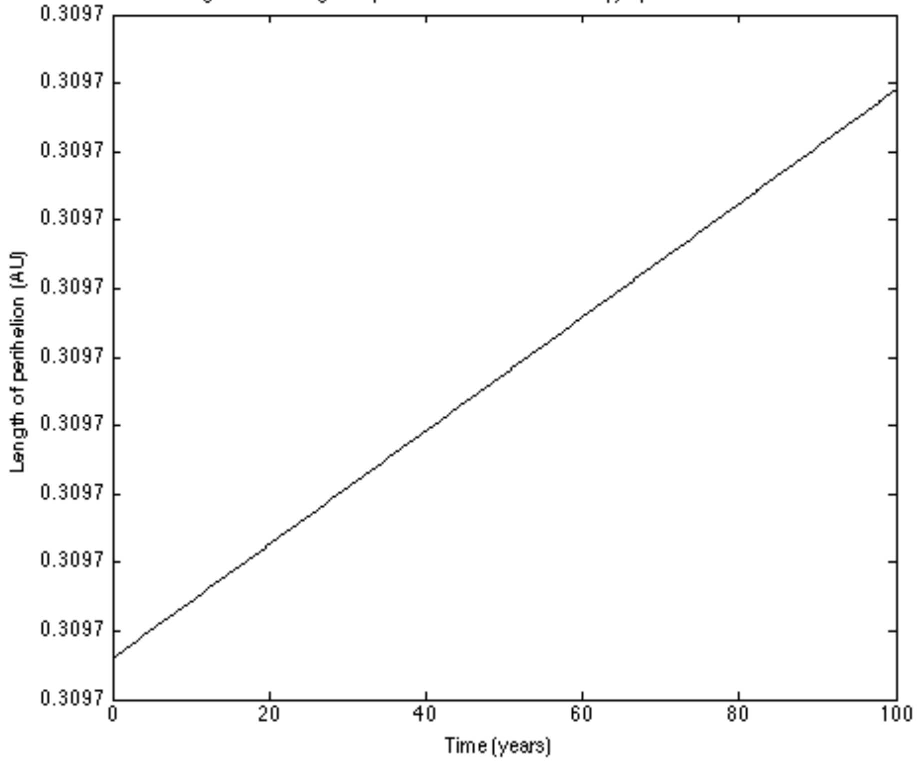


Figure 16: Position of perihelion over one century; alpha = .00000011

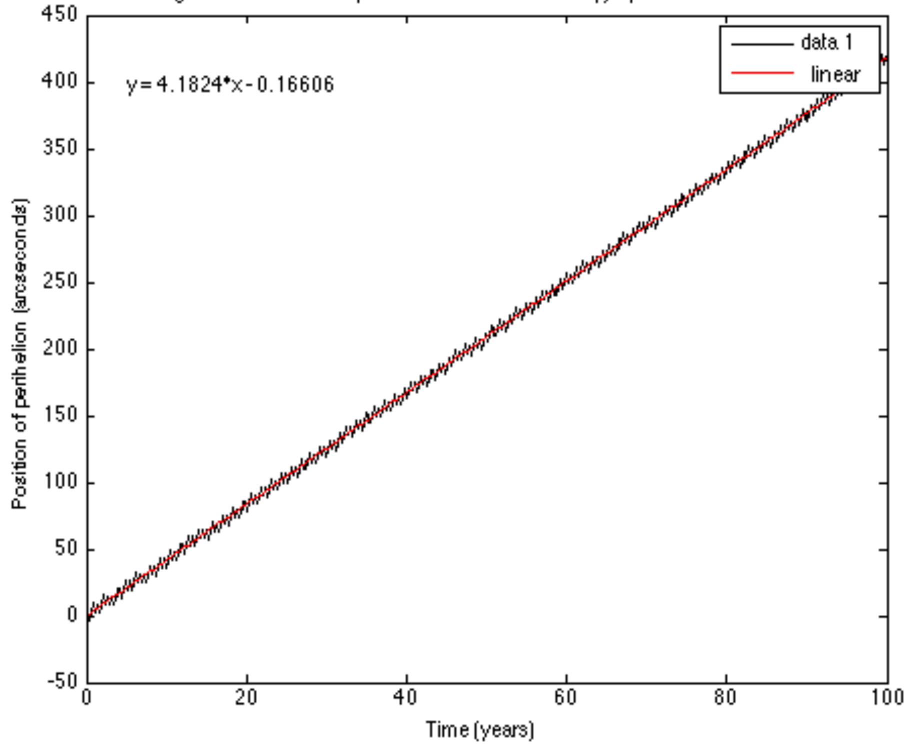


Figure 17: Position of perihelion over one century; alpha = .0000002

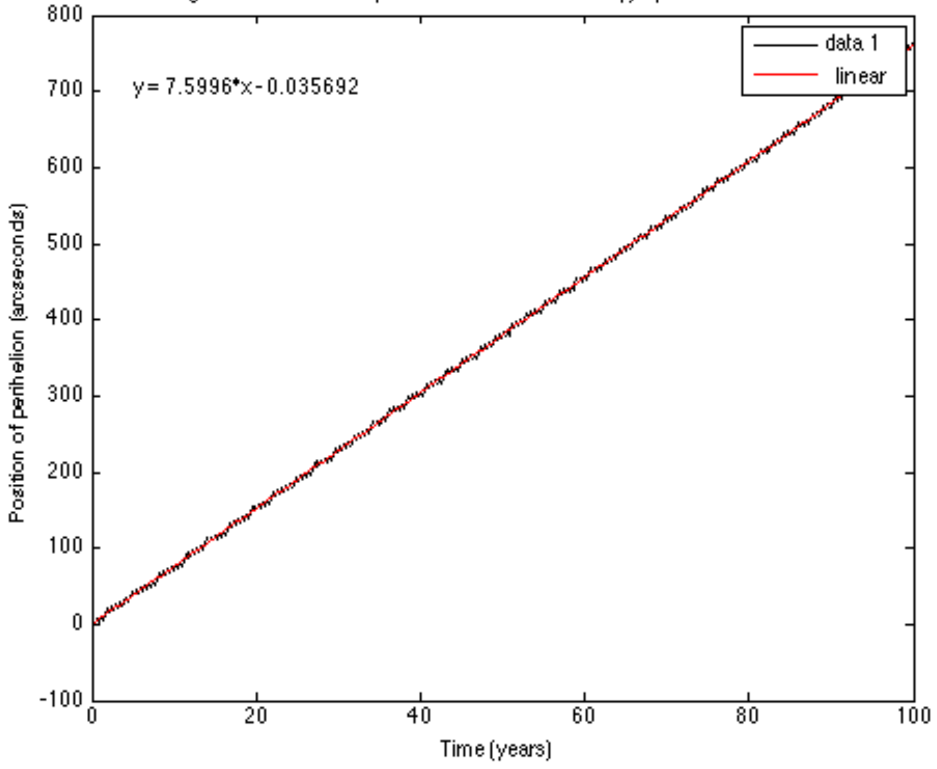


Figure 18: Position of perihelion over one century; alpha = .0000005

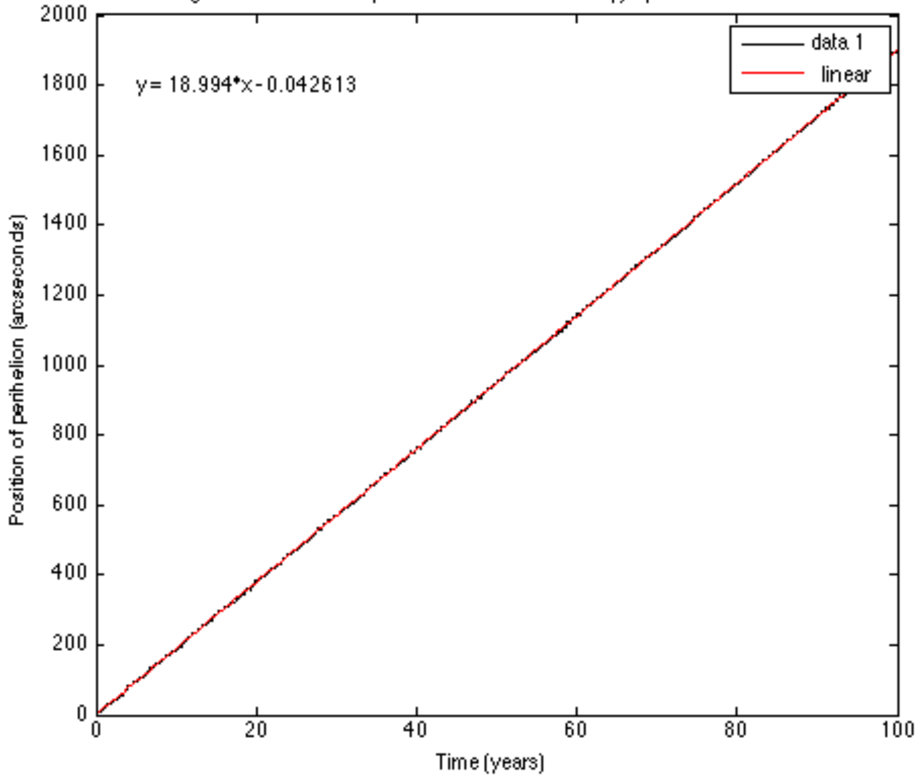


Figure 19: Relativity-based precession rate v. alpha

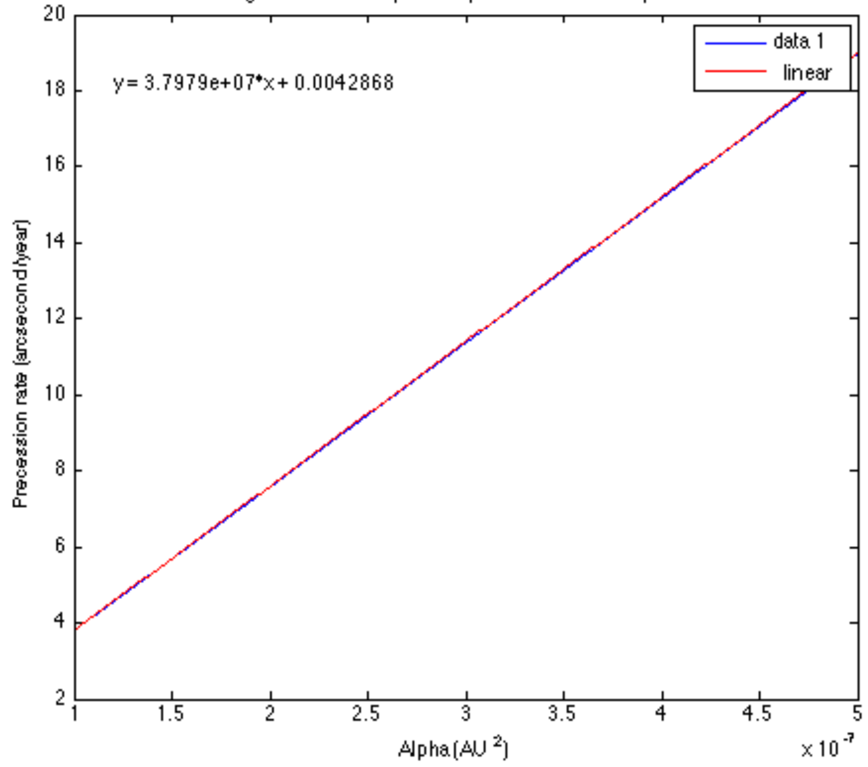
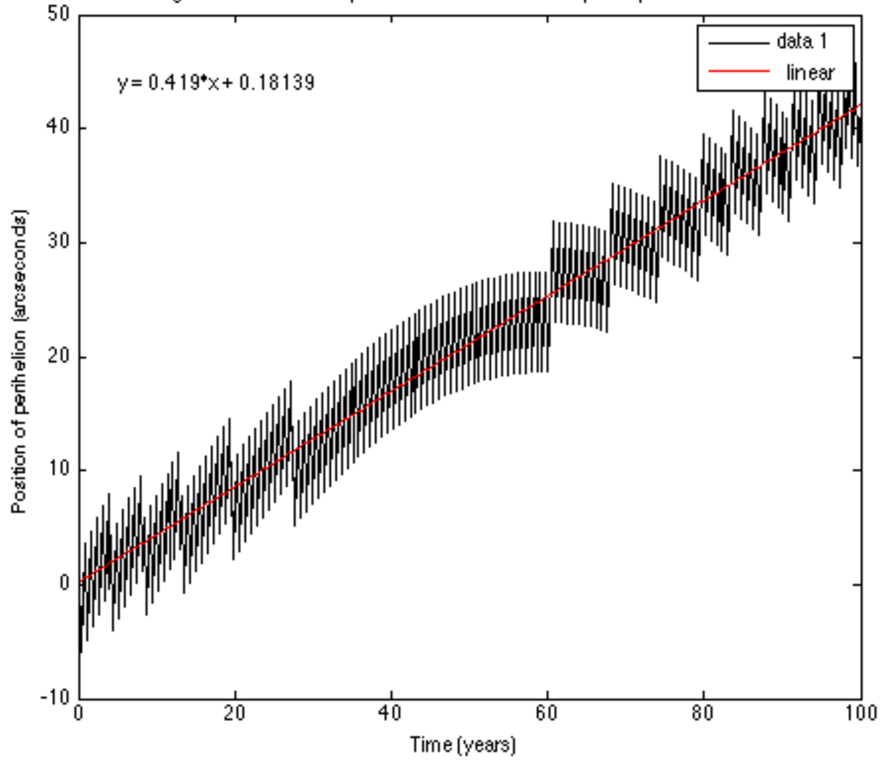


Figure 20: Position of perihelion over one century for alpha = 1.1e-8


data 1
linear
y = 0.419*x + 0.18139
Position of perihelion (arcseconds)
Time (years)