Michael Cowan

Computational Physics

Problem Set 4

Chapter 3

3. In this problem, we write a program to simulate the orbit of an object around the sun. The program we write is modified from the downloaded program **orbit.m**. In all parts of the problem, the **orbit.m** program is modified to run for a set number of complete orbits instead of a set number of time steps. This was accomplished by noting that the angular position is calculated using an arctangent function defined on the interval [-π,π], which reverses its sign twice during one complete orbit. The main **for** loop was replaced by a **while** loop that executed until the object reached the desired number of complete orbits.

Part a: Additional lines are inserted to calculate the period, eccentricity, semimajor axis, and perihelion distance. The perihelion and aphelion lengths were found from the minimum and maximum values of the radial position. The period was calculated by using the **intrpf** function to interpolate the radial position at every second reversal.The semimajor axis was calculated by averaging the perihelion and aphelion distances, and the eccentricity was calculated using the formula . The eccentricity from this formula is then compared to the eccentricity found by the formula .

Code (my changes to **orbit.m** are in italics):

1. % orbit - Program to compute the orbit of a comet.
2. clear all; help orbit; % Clear memory and print header
3. %\* Set initial position and velocity of the comet.
4. r0 = input('Enter initial radial distance (AU): ');
5. v0 = input('Enter initial tangential velocity (AU/yr): ');
6. r = [r0 0]; v = [0 v0];
7. state = [ r(1) r(2) v(1) v(2) ]; % Used by R-K routines
8. %\* Set physical parameters (mass, G\*M)
9. GM = 4\*pi^2; % Grav. const. \* Mass of Sun (au^3/yr^2)
10. mass = 1; % Mass of comet
11. adaptErr = 1.e-3; % Error parameter used by adaptive Runge-Kutta
12. time = 0;
13. *orb=input('Enter the number of orbits to calculate over: ');*
14. tau = input('Enter time step (yr): ');
15. *rev=0;*
16. NumericalMethod = menu('Choose a numerical method:', ...
17. 'Euler','Euler-Cromer','Runge-Kutta','Adaptive R-K');
18. *iStep=0;*
19. *while (rev<(orb\*2))*
20. *iStep=iStep+1;*
21. %\* Record position and energy for plotting.
22. rplot(iStep) = norm(r); % Record position for polar plot
23. thplot(iStep) = atan2(r(2),r(1));
24. tplot(iStep) = time;
25. kinetic(iStep) = .5\*mass\*norm(v)^2; % Record energies
26. potential(iStep) = - GM\*mass/norm(r);
27. %\* Calculate new position and velocity using desired method.
28. if( NumericalMethod == 1 )
29. accel = -GM\*r/norm(r)^3;
30. r = r + tau\*v; % Euler step
31. v = v + tau\*accel;
32. time = time + tau;
33. elseif( NumericalMethod == 2 )
34. accel = -GM\*r/norm(r)^3;
35. v = v + tau\*accel;
36. r = r + tau\*v; % Euler-Cromer step
37. time = time + tau;
38. elseif( NumericalMethod == 3 )
39. state = rk4(state,time,tau,'gravrk',GM);
40. r = [state(1) state(2)]; % 4th order Runge-Kutta
41. v = [state(3) state(4)];
42. time = time + tau;
43. else
44. [state time tau] = rka(state,time,tau,adaptErr,'gravrk',GM);
45. r = [state(1) state(2)]; % Adaptive Runge-Kutta
46. v = [state(3) state(4)];
47. end
48. *%calculate the number of times the angle changes sign*
49. *if (time>tau && thplot(iStep)\*thplot(iStep-1)<0)*
50. *rev=rev+1;*
51. *end*
52. *%find the period, perihelion, aphelion, semimajor axis, and eccentrity for each orbit*
53. *if (mod(rev,2)==0 && rev>1)*
54. *aph(rev/2)=max(rplot);*
55. *perih(rev/2)=min(rplot);*
56. *semimaj(rev/2)=(aph(rev/2)+perih(rev/2))/2;*
57. *ecc(rev/2)=(aph(rev/2)-perih(rev/2))/(aph(rev/2)+perih(rev/2));*
58. *per(rev/2)=intrpf(0,[thplot(iStep),thplot(iStep-1),thplot(iStep-2)],[tplot(iStep),tplot(iStep-1),tplot(iStep-2)]);*
59. *end*
60. *end*
61. *fprintf('The average period is %g years\n',mean(per));*
62. *fprintf('The average aphelion distance is %g AU\n',mean(aph));*
63. *fprintf('The average perihelion distance is %g AU\n',mean(perih));*
64. *fprintf('The average semimajor axis length is %g AU\n',mean(semimaj));*
65. *fprintf('The average eccentricity is %g\n',mean(ecc));*
66. %\* Graph the trajectory of the comet.
67. figure(1); clf; % Clear figure 1 window and bring forward
68. polar(thplot,rplot,'+'); % Use polar plot for graphing orbit
69. xlabel('Distance (AU)'); grid;
70. pause(1) % Pause for 1 second before drawing next plot
71. %\* Graph the energy of the comet versus time.
72. figure(2); clf; % Clear figure 2 window and bring forward
73. totalE = kinetic + potential; % Total energy
74. plot(tplot,kinetic,'-.',tplot,potential,'--',tplot,totalE,'-')
75. *axis([0,mean(per),1.3\*min(potential),1.5\*max(kinetic)]);*
76. legend('Kinetic','Potential','Total');
77. xlabel('Time (yr)'); ylabel('Energy (M AU^2/yr^2)');
78. *%calculate the actual eccentricity*
79. *eps=sqrt(1+((2\*totalE(1)\*(mass\*r0\*v0)^2)/(GM^2\*mass^3)));*
80. *fprintf('The actual eccentricity is %g\n',eps);*

Results:

I simulated the Earth first, supposing a perfectly circular orbit of radius 1 AU:

Enter initial radial distance (AU): 1

Enter initial tangential velocity (AU/yr): 2\*pi

Enter the number of orbits to calculate over: 1

Enter time step (yr): .002

Choose a numerical method:

[ 1] Euler

[ 2] Euler-Cromer

[ 3] Runge-Kutta

[ 4] Adaptive R-K

pick a number, any number: 2

The average period is 1.00011 years

The average aphelion distance is 1.00636 AU

The average perihelion distance is 0.993796 AU

The average semimajor axis length is 1.00008 AU

The average eccentricity is 0.0062827

The actual eccentricity is 0

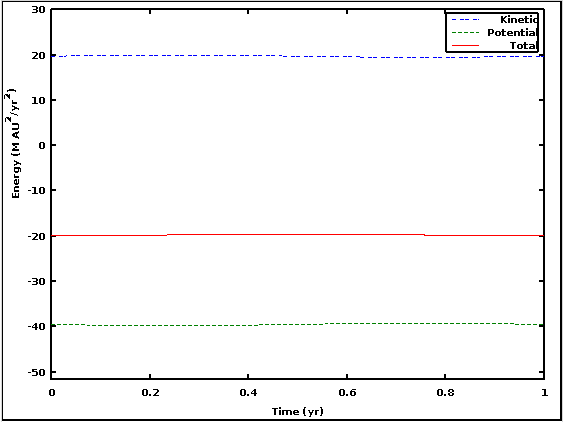
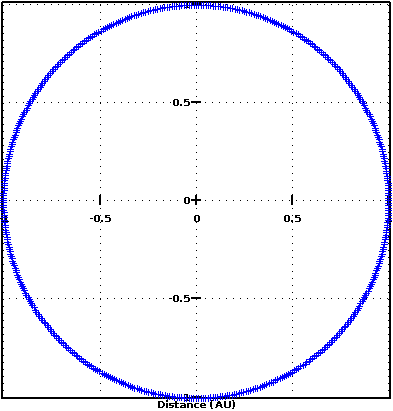


Figure 1: Simulating Earth’s orbit

I then put in some other numbers:

Enter initial radial distance (AU): 25

Enter initial tangential velocity (AU/yr): 1

Enter the number of orbits to calculate over: 1

Enter time step (yr): .05

Choose a numerical method:

[ 1] Euler

[ 2] Euler-Cromer

[ 3] Runge-Kutta

[ 4] Adaptive R-K

pick a number, any number: 2

The average period is 78.2323 years

The average aphelion distance is 25.0001 AU

The average perihelion distance is 11.5835 AU

The average semimajor axis length is 18.2918 AU

The average eccentricity is 0.366739

The actual eccentricity is 0.366743

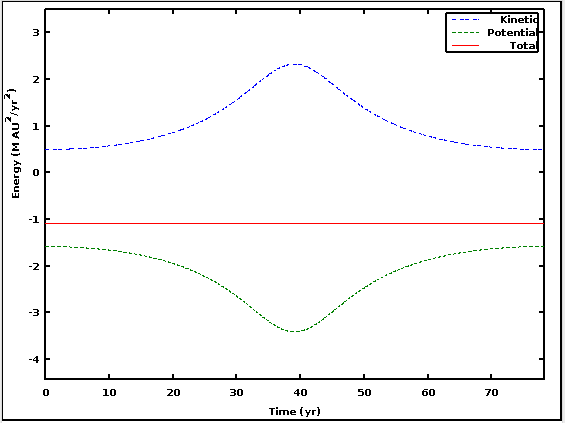
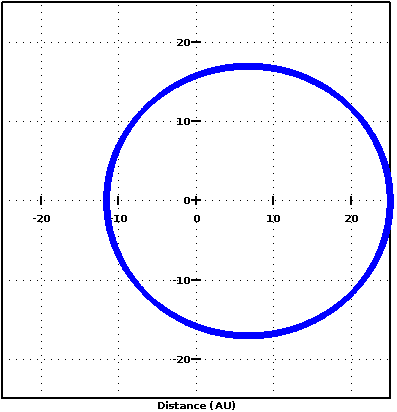


Figure 2: Another, more elliptical orbit

Part b and c: In part b of the problem, we are asked to verify Kepler’s Third Law. I removed the eccentricity calculations, since they were not needed. To create the graph to verify Kepler’s Third Law, I just wrote a second short program to graph the points collected.

In part c we are asked to verify the virial theorem: . It was easiest to just insert a few lines at the end of the main program to get the data for parts b and c together. All objects had data collected over 5 complete orbits and then averaged. The results for both parts b and c are listed in the table below.

Code (my changes to **orbit.m** are in italics):

1. % orbit - Program to compute the orbit of a comet.
2. clear all; help orbit; % Clear memory and print header
3. %\* Set initial position and velocity of the comet.
4. r0 = input('Enter initial radial distance (AU): ');
5. v0 = input('Enter initial tangential velocity (AU/yr): ');
6. r = [r0 0]; v = [0 v0];
7. state = [ r(1) r(2) v(1) v(2) ]; % Used by R-K routines
8. %\* Set physical parameters (mass, G\*M)
9. GM = 4\*pi^2; % Grav. const. \* Mass of Sun (au^3/yr^2)
10. mass = 1; % Mass of comet
11. adaptErr = 1.e-3; % Error parameter used by adaptive Runge-Kutta
12. time = 0;
13. *orb=input('Enter the number of orbits to calculate over: ');*
14. tau = input('Enter time step (yr): ');
15. *rev=0;*
16. NumericalMethod = menu('Choose a numerical method:', ...
17. 'Euler','Euler-Cromer','Runge-Kutta','Adaptive R-K');
18. *iStep=0;*
19. *while (rev<(orb\*2))*
20. *iStep=iStep+1;*
21. %\* Record position and energy for plotting.
22. rplot(iStep) = norm(r); % Record position for polar plot
23. thplot(iStep) = atan2(r(2),r(1));
24. tplot(iStep) = time;
25. kinetic(iStep) = .5\*mass\*norm(v)^2; % Record energies
26. potential(iStep) = - GM\*mass/norm(r);
27. %\* Calculate new position and velocity using desired method.
28. if( NumericalMethod == 1 )
29. accel = -GM\*r/norm(r)^3;
30. r = r + tau\*v; % Euler step
31. v = v + tau\*accel;
32. time = time + tau;
33. elseif( NumericalMethod == 2 )
34. accel = -GM\*r/norm(r)^3;
35. v = v + tau\*accel;
36. r = r + tau\*v; % Euler-Cromer step
37. time = time + tau;
38. elseif( NumericalMethod == 3 )
39. state = rk4(state,time,tau,'gravrk',GM);
40. r = [state(1) state(2)]; % 4th order Runge-Kutta
41. v = [state(3) state(4)];
42. time = time + tau;
43. else
44. [state time tau] = rka(state,time,tau,adaptErr,'gravrk',GM);
45. r = [state(1) state(2)]; % Adaptive Runge-Kutta
46. v = [state(3) state(4)];
47. end
48. *%calculate the number of times the angle changes sign*
49. *if (time>tau && thplot(iStep)\*thplot(iStep-1)<0)*
50. *rev=rev+1;*
51. *end*
52. *%find the period, perihelion, aphelion, semimajor axis, and eccentrity for each orbit*
53. *if (mod(rev,2)==0 && rev>1)*
54. *aph(rev/2)=max(rplot);*
55. *perih(rev/2)=min(rplot);*
56. *semimaj(rev/2)=(aph(rev/2)+perih(rev/2))/2;*
57. *per(rev/2)=intrpf(0,[thplot(iStep),thplot(iStep-1),thplot(iStep-2)],[tplot(iStep),tplot(iStep-1),tplot(iStep-2)]);*
58. end
59. end
60. *fprintf('The average period is %g years\n',mean(diff(per)));*
61. *fprintf('The average aphelion distance is %g AU\n',mean(aph));*
62. *fprintf('The average perihelion distance is %g AU\n',mean(perih));*
63. *fprintf('The average semimajor axis length is %g AU\n',mean(semimaj));*
64. %\* Graph the trajectory of the comet.
65. figure(1); clf; % Clear figure 1 window and bring forward
66. polar(thplot,rplot,'+'); % Use polar plot for graphing orbit
67. xlabel('Distance (AU)'); grid;
68. pause(1) % Pause for 1 second before drawing next plot
69. %\* Graph the energy of the comet versus time.
70. figure(2); clf; % Clear figure 2 window and bring forward
71. totalE = kinetic + potential; % Total energy
72. plot(tplot,kinetic,'-.',tplot,potential,'--',tplot,totalE,'-')
73. *axis([0,per(rev/2)]);*
74. legend('Kinetic','Potential','Total');
75. xlabel('Time (yr)'); ylabel('Energy (M AU^2/yr^2)');
76. *%confimation of virial theorem*
77. *disp('The average kinetic energy is:');*
78. *disp(mean(kinetic));*
79. *disp('The average potential energy divided by 2 is:');*
80. *disp(mean(potential)/2);*
81. a=[0.288833,1.00008,4.37217,18.2918,29.7024,56.4327];
82. T=[0.151265,0.998648,8.23633,72.734,144.514,375.452];
83. loglog(a.^3,T.^2,'+',[10^-2,10^6],[10^-2,10^6],'--');
84. xlabel('Semimajor axis length (AU) cubed');
85. ylabel('Period (yr) squared');
86. title('Period vs semimajor axis length');

Results:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Initial radial distance (AU) | 0.5 | 1 | 7 | 25 | 50 | 100 |
| Initial velocity (AU/yr) | 4.5 | 2π | 1.5 | 1.0 | 0.5 | 0.3 |
| Average semimajor axis (AU) | 0.288833 | 1.00008 | 4.37217 | 18.2918 | 29.7024 | 56.4327 |
| Average period (yr) | 0.151265 | 0.998648 | 8.23633 | 72.734 | 144.514 | 375.452 |
| Average Kinetic Energy | 65.169 | 19.738 | 4.5140 | 1.0790 | 0.66449 | 0.34975 |
| Average Potential Energy/2 | -66.729 | -19.738 | -4.5144 | -1.0791 | -0.66453 | -0.34976 |

Table 1: Values for five orbiting bodies

As the table above shows, the body orbiting at the smallest distance from the sun has the largest discrepancy from the expectation of the virial theorem, at about 2.4%. None of the others differ by more than 0.1%.

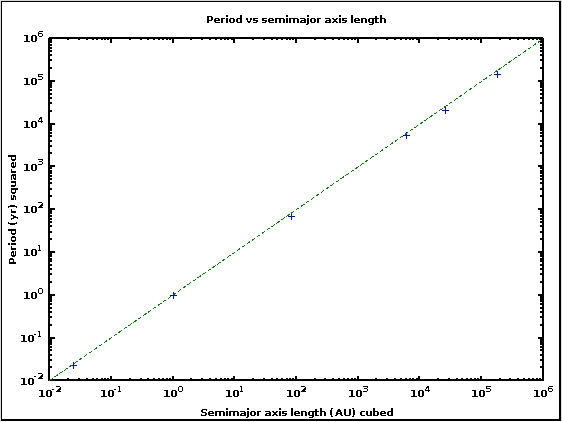


Figure 3: Confirmation of Kepler’s Third Law for six solar system bodies; the dashed line shows perfect adherence to the Law

Orbital Decay Program

In this problem we are examining the orbit of the International Space Station in Low Earth Orbit, first without and then with drag.

Part a. Convert the constant GM to units of Earth radii and days:

Part b. In this part we calculate how many orbits the ISS would make if there is no drag. Using the adaptive Runge-Kutta method from the **orbit.m** program, I again count the number of times that the angle changes sign (since it is confined to the interval [-π,π]) and divided by 2 to get the number of complete obits; I then interpolate to find the final angular position and add that fraction of the orbit to the total.

Code (original **orbit.m** in italics):

1. *% orbit - Program to compute the orbit of a comet.*
2. *clear all; help orbit; % Clear memory and print header*
3. %\* Set physical parameters (mass, G\*M)
4. GM = 1.15e4; % Grav. const. \* Mass of Earth (Earth radii^3/day^2)
5. mass = 3.7e5; % Mass of ISS
6. adaptErr = 1.e-6; % Error parameter used by adaptive Runge-Kutta
7. *time = 0;*
8. *%\* Set initial position and velocity of the comet.*
9. r0 = 1.0549;
10. v0 = sqrt(GM/r0);
11. *r = [r0 0]; v = [0 v0];*
12. *state = [ r(1) r(2) v(1) v(2) ]; % Used by R-K routines*
13. maxtime = input('Enter length of time to orbit (days): ');
14. tau = input('Enter time step (days): ');
15. rev=0;
16. iStep=0;
17. while time<maxtime
18. iStep=iStep+1;
19. %\* Record position for plotting.
20. *rplot(iStep) = norm(r); % Record position for polar plot*
21. *thplot(iStep) = atan2(r(2),r(1));*
22. *tplot(iStep) = time;*
23. *%\* Calculate new position and velocity using adaptive Runge-Kutta*
24. *[state time tau] = rka(state,time,tau,adaptErr,'gravrk',GM);*
25. *r = [state(1) state(2)]; % Adaptive Runge-Kutta*
26. *v = [state(3) state(4)];*
27. if (iStep>2 && thplot(iStep)\*thplot(iStep-1)<0)
28. rev=rev+1;
29. end
30. end
31. %interpolate to find the exact angle at the specified end time
32. thfinal=intrpf(maxtime,[tplot(iStep),tplot(iStep-1),tplot(iStep-2)],[thplot(iStep),thplot(iStep-1),thplot(iStep-2)]);
33. %Count the number of orbits by halving the number of reversals and adding in the fraction of the current orbit
34. if (thplot(iStep)>0)
35. orb=floor(rev/2)+thfinal/(2\*pi);
36. else
37. orb=floor(rev/2)+thfinal/(2\*pi)+1;
38. end
39. fprintf('The ISS makes %g orbits in %g days\n',orb,maxtime);
40. *%\* Graph the trajectory of the comet.*
41. *figure(1); clf; % Clear figure 1 window and bring forward*
42. *polar(thplot,rplot,'+'); % Use polar plot for graphing orbit*
43. *xlabel('Distance (Earth radii)'); grid;*

Result:

Enter length of time to orbit (days): 30

Enter time step (days): .01

The ISS makes 472.606 orbits in 30 days

This is very close to the number of orbits I calculated algebraically, 472.578, a difference of <0.006%.

Part c. In part c we are asked to use the formula to modify the **orbit.m** program. We then use this program to calculate the density of air necessary to cause the orbit of the ISS to decay by 2 km per month. In order to change the acceleration, it was necessary to modify the **gravrk.m** function, which actually does the work of computing the new position and velocity at each time step. Minor modifications also had to be made to **rka.m** and **rk4.m** to handle the additional parameters.

To get an order of magnitude estimate, I first ran the program with ρ increasing by a factor of 10 each iteration, and found that the correct value was on the order of 109 kg/Earth radius3. I then reran the program with ρ starting at 0 and increasing by 106 kg/Earth radius3 to obtain a better estimate. I then converted the value of ρ to kg/m3.

Code (original **orbit.m** in italics):

1. *% orbit - Program to compute the orbit of a comet.*
2. *clear all; help orbit; % Clear memory and print header*
3. %\* Set physical parameters (mass, G\*M)
4. GM = 1.15e4; % Grav. const. \* Mass of Earth (Earth radii^3/day^2)
5. mass = 3.7e5; % Mass of ISS
6. Cd=0.8;
7. A=500\*(6.371e6)^-2;
8. rho=0;
9. adaptErr = 1.e-6; % Error parameter used by adaptive Runge-Kutta
10. tau = input('Enter time step (days): ');
11. dif=2/(472.606\*6.371e3);
12. %\* Set initial position and velocity of the ISS
13. r0 = 1.0549;
14. v0 = sqrt(GM/r0);
15. rfinal=r0;
16. while (r0-rfinal<dif)
17. rho=rho+10^6;
18. *time = 0;*
19. *r = [r0 0]; v = [0 v0];*
20. *state = [ r(1) r(2) v(1) v(2) ]; % Used by R-K routines*
21. rev=0;
22. iStep=0;
23. while rev<2
24. iStep=iStep+1;
25. *%\* Record position for plotting.*
26. *rplot(iStep) = norm(r); % Record position for polar plot*
27. *thplot(iStep) = atan2(r(2),r(1));*
28. *tplot(iStep) = time;*
29. *%\* Calculate new position and velocity using adaptive Runge-Kutta*
30. [state time tau] = rka(state,time,tau,adaptErr,'gravrk\_mod',GM,Cd,A,rho,mass);
31. *r = [state(1) state(2)]; % Adaptive Runge-Kutta*
32. *v = [state(3) state(4)];*
33. if (iStep>2 && thplot(iStep)\*thplot(iStep-1)<0)
34. rev=rev+1;
35. end
36. end
37. %interpolate to find the exact radius after one orbit
38. rfinal=intrpf(0,[thplot(iStep),thplot(iStep-1),thplot(iStep-2)],[rplot(iStep),rplot(iStep-1),rplot(iStep-2)]);
39. end
40. rho=rho\*(6.371e6)^-3;
41. fprintf('The density of the atmosphere in LEO is %g kg/m^3\n',rho);

Results:

Enter time step (days): .01

The density of the atmosphere in LEO is 1.02863e-11 kg/m^3