Angles Only Orbit Determination Using Gauss' Method

Semester Project

Johnathan Corbin

Spaceflight Mechanics
Spring 2019
Rensselaer Polytechnic Institute
April 25, 2019

Contents

1 Nomenclature

Included below is a list of the common variables and terms used throughout the project.

Symbol	Description	Units
e_{sat}	eccentricity of the satellite	-
a_{sat}	semi-major axis of the satellite	km
Ω_{sat}	right ascension of the ascending node of the satellite	degrees
i_{sat}	inclination of the satellite	degrees
ω_{sat}	argument of perogee of the satellite	degrees
θ_{sat}	true anomaly of the satellite	degrees
E	eccentric anomaly	degrees
h	specific angular momentum	$\frac{km^2}{s}$
t_1	time of the first angle measurement	seconds
t_2	time of the second angle measurement	seconds
t_3	time of the third angle measurement	seconds
$\mathbf{R}_1$	position vector from Earth to satellite at t_1	km
$\mathbf{R}_2$	position vector from Earth to satellite at t_2	km
$\mathbf{R}_3$	position vector from Earth to satellite at t_3	km
ρ_1	slant range at t_1	km
ρ_2	slant range at t_2	km
ρ_3	slant range at t_3	km
$\hat{\rho_1}$	direction of ρ_1	-
$\hat{\rho_2}$	direction of ρ_2	-
$\hat{\rho_3}$	direction of ρ_3	-
${}^{N}C^{PF}$	direction cosine matrix of the body 313 matrix	-
${}^{PF}C^{POL}$	direction cosine matrix for polar to perifocal	-

2 Introduction

For my semester project, I am choosing to do a method of initial orbit determination of an unknown object. My goal is to use the Gauss method of preliminary orbit determination. The Gauss method works by taking three angle measurements from a known location to an unknown object whose orbit you wish to determine.

In my scenario, a satellite is orbiting the Earth with known orbital parameters. Additionally, an object with an unknown orbital parameters is also orbiting around Earth. I seek to use the location of the known satellite to determine the orbit of the unknown object. To do this, three angle measurements are taken from the known satellite in the direction of the unknown object. If the timescale between the angle measurements is sufficiently small, then it is possible to determine an estimate of the unknown object's orbit.

3 System Assumptions

- The system is a two body problem. Only the gravitational attraction of the Earth is considered to be acting on the satellites.
- The system takes place around Earth, but by changing μ, the program could be set to take place around any body.
- The Earth is a perfect sphere, so that the effects of precession can be ignored. The timescale is sufficiently small though that precession wouldn't have a large effect anyways.
- The topocentric horizon reference frame is used on the satellite to create the direction unit vectors to the unknown object from a given altitude and azimuth angle as seen from the satellite.
- The timescale between angle measurements must be fairly small so that only the first terms of the Lagrange coefficients are large enough to significantly contribute.
- The orbit of the satellite taking the measurements is known and that a position vector can be made to that satellite at each instance when an angle measurement is made.

4 Theory

To begin, we need to specify the orbital parameters of the known satellite, henceforth referred to as just *satellite*. I'm describing the satellite using the following parameters: eccentricity (e_{sat}), semi-major axis (a_{sat}), right ascension of the ascending node (Ω_{sat}), inclination (i_{sat}), argument of perogee (ω_{sat}), and the true anomaly (θ_{sat}). These six parameters fully describe the size, shape, and location of the satellite at an initial time period. Since it was assumed the Earth is a perfect sphere, all these values except the true anomaly will be constant throughout the derivations Figure 1 shows a diagram of the system. We seek to find the position vectors from the center of the Earth to the satellite at three separate times In the figure, we first need to find $\mathbf{R}_1$, $\mathbf{R}_2$, and $\mathbf{R}_3$, which denote the position vector from the center of the Earth to the satellite.

Now that our known satellite's position in space can be described, we need to specify three separate times that the angle measurements will be made at. The three times will be designated t_1, t_2, and t_3 in increasing order. t_1 will naturally occur at 0 seconds. The true anomaly initially given for the satellite is the true anomaly at time t_1. In order to find the satellite's true anomaly at times t_2 and t_3, Kepler's equations must be used and are as follows:

$$\sqrt{\frac{\mu}{a^3}}(t-t_o) = E - esin(E) \tag{1}$$

$$tan(\frac{\theta}{2}) = \sqrt{\frac{1+e}{1-e}}tan(\frac{E}{2}) \tag{2}$$

Equations (1) and (2) are the general form of Kepler's equations for elliptical orbits, which our known satellite is in if it is orbiting around Earth. Using equation (2), the eccentric anomaly can be found for the initial true anomaly. Similarly, since this takes place at $t_1 = 0$, equation (1) can then be used to find the time since

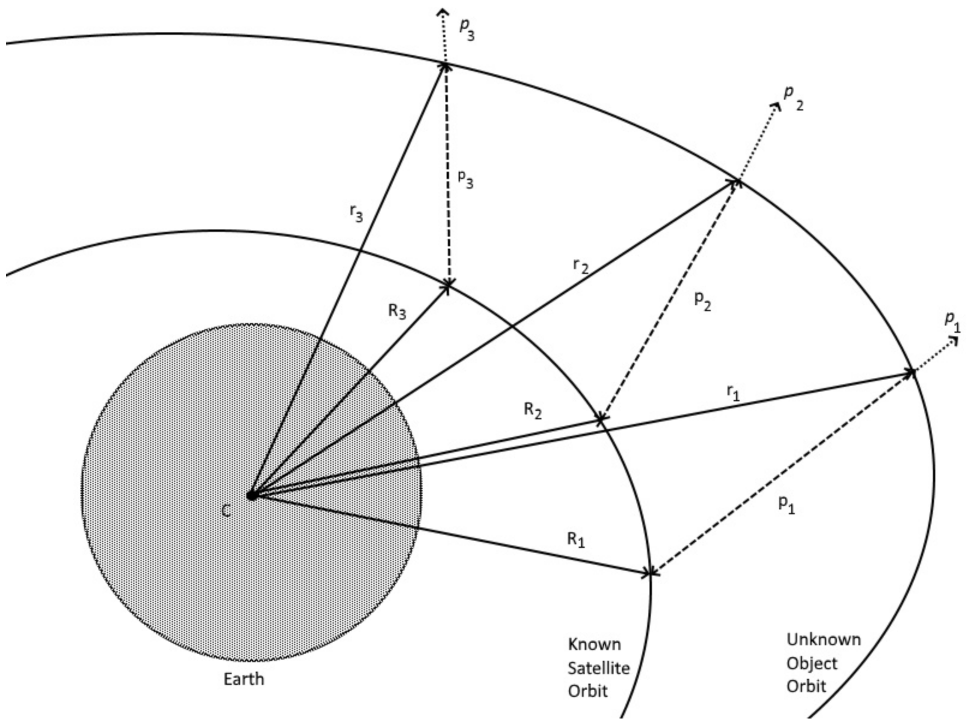


Figure 1: Diagram of the Various Orbits and Vectors

perogee. Once the initial time from perogee of the satellite is found, t_2 and t_3 can be used in equation (1) to back out the eccentric anomaly at the respective times. Equation (2) can then be used to find the true anomaly at t_2 and t_3 of the satellite. Using the Body 313 matrix, the orbital elements of the satellite at each true anomaly can be used to find the position vectors $\mathbf{R}_1$, $\mathbf{R}_2$, and $\mathbf{R}_3$. The Body 313 matrix is as follows:

$$\begin{bmatrix} -sin(\Omega)*cos(i)*sin(\omega)+cos(\Omega)*cos(\omega) & -sin(\Omega)*cos(i)*cos(\omega)-cos(\Omega)*sin(\omega) & sin(\Omega)*sin(i) \\ cos(\Omega)*cos(i)*sin(\omega)+sin(\Omega)*cos(\omega) & cos(\Omega)*cos(i)*cos(\omega)-sin(\Omega)*sin(\omega) & sin(i)*cos(\omega) \\ sin(i)*sin(\omega) & sin(i)*cos(\omega) & cos(i) \end{bmatrix}$$

Additionally, the direction cosine matrix from polar to perifocal coordinates is neccessary to find the position vectors. The direction cosine matrix is as follows:

$$\begin{bmatrix} cos(\theta) & sin(\theta) & 0 \\ -sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Using the previous two direction cosine matrices, the position vectors can be found as follows:

$$\mathbf{R} = \left[{}^{N}C^{PF}\right]\left[{}^{PF}C^{POL}\right]\begin{bmatrix} r_{pol} \\ \theta \\ 0 \end{bmatrix} \tag{3}$$

where r_{pol} is found at each true anomaly with the following equation:

$$r_{pol} = \frac{h^2}{\mu(1+ecos(\theta))} \tag{4}$$

Finally, the position vectors to the satellite at the times of the angle measurements are known. Next, it is assumed that the satellite is able to make a line of site observation of the unknown object. This is in order to generate angle measurements to the unknown object. In order to solve the system, six independent

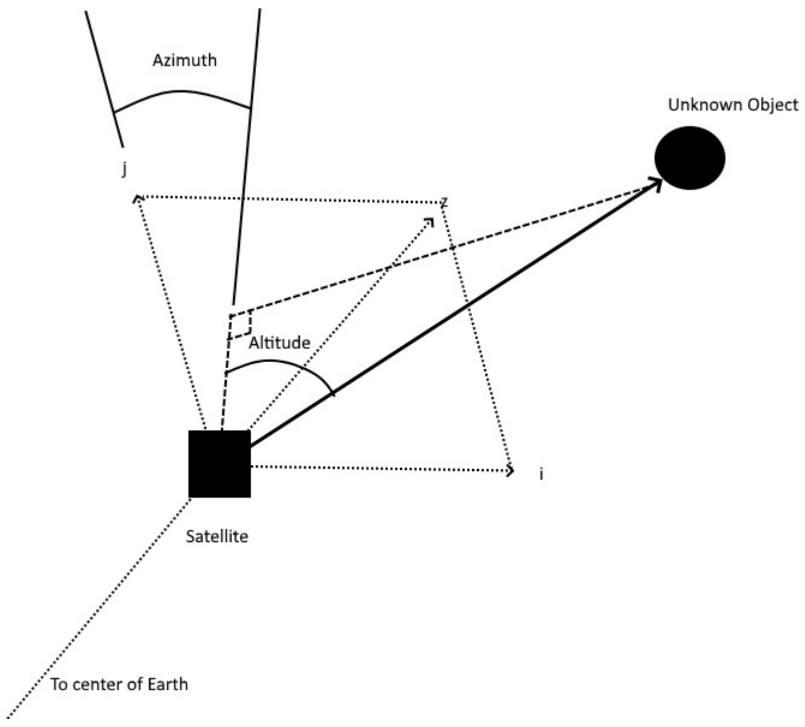


Figure 2: Diagram of the Azimuth and Altitude Angles

angles are needed. In this case, it is going to be three altitude angles and three azimuth angles. These angles are shown in figure 2.

With the angle measurements in hand, goal now is to find the slant ranges (ρ_1, ρ_2, and ρ_3) and their directions ($\hat{\rho_1}$, $\hat{\rho_2}$, and $\hat{\rho_3}$). Once these values are known, the position vectors from the center of the Earth to the unknown object can be calculated. Thus far, I have deviated significantly from the derivation for the Gauss method put forth by Curtis in the book *Orbital Mechanics for Engineering Students* (Curtis, 2013). In order to get to a point where the derivations used in the textbook are useful, it is necessary to calculate the directions of the slant ranges. The unit vector $\hat{\rho}$ in the line of sight direction for this system (topocentric horizon coordinate system) is given as the following:

$$\hat{\rho} = cos(altitude) * sin(azimuth)\hat{i} + cos(altitude) * cos(azimuth)\hat{j} + sin(altitude)\hat{k} \tag{5}$$

Using equation (5), the direction of the slant ranges ρ_1, ρ_2, and ρ_3 can be found for the instance where each measurement takes place as long as an azimuth and altitude angle are measured.

Up to this point, I have deviated from Curtis. In the textbook, Curtis derives these formulae for a ground station on Earth. I have added the extra complexity of changing the ground station to a satellite that moves in a much more complicated way with time. However, once the directions of the slant ranges have been found, the derivations put forth by Curtis once again apply and can be used (Curtis, 2013).

The further derivations of the method are long and tedious. In an effort to stay within the recommended page limit for this report, I am going to ask the reader to refer to pages 279 - 285 of *Orbital Mechanics for Engineering Students*, Third Edition (Curtis, 2013). These pages contain Curtis' derivations of Gauss method that I closely follow. In short, the derivations involve finding the Lagrange functions, the Lagrange coefficients, varying constants, and solving an eighth-order polynomial. Putting the full derivations here would certainly push me past the recommended 6 - 7 pages.

5 Results and Discussion

For the project, I coded everything into Matlab. The following is a general list of the steps that my code follows. To the code in more detail, please view the appendix of this paper.

1. Preliminary estimate of $\vec{\mathbf{r}}$ and $\vec{\mathbf{v}}$ at time t_2.
   (a) Declaration of constants.
   (b) Set values for measured angles to unknown object.
   (c) Set values for time when angles are measured.
   (d) From orbit parameters for the known satellite, compute position vector $\vec{\mathbf{R}}$ to the satellite at each point in time.
   (e) Using measured angles, compute unit vector from satellite to object at each time.
   (f) Compute various coefficients necessary in order to find the Lagrange coefficients.
   (g) Use the Lagrange coefficients to calculate the magnitude of the position vectors from the satellite to the unknown object.
   (h) Use the previously determined unit vectors and magnitudes to find position vectors to the unknown object.
   (i) Use Lagrange coefficients to find the velocity at time t_2.
2. Use method from Curtis to improve the initial $\vec{\mathbf{r}}$ and $\vec{\mathbf{v}}$ estimates.
   (a) Using old $\vec{\mathbf{r}}$ and $\vec{\mathbf{v}}$ values, recalculate the Lagrange coefficients.
   (b) Find new $\vec{\mathbf{r}}$ and $\vec{\mathbf{v}}$ values.
   (c) Repeat steps until the values no longer change.
3. Display computed values.
   (a) Using $\vec{\mathbf{r}}$ and $\vec{\mathbf{v}}$ of unknown object, compute the 6 classical orbit elements.
   (b) Plot Earth, the known orbit, and the unknown orbit.
   (c) Display orbit elements to the command line.

A lot of trial and error went into making sure my algorithm was producing accurate results. I used Example 5.11 and problem 5.14 from the textbook to validate and verify my code. Once I was able to produce the results that Curtis says are correct, I moved on to creating my own parameters for the system.

The main parameters I changed included the orbital elements of the satellite, the times the angle measurements take place at, and the angle measurements. Figure 3 is one of the best results I was able to produce with parameters I randomly choose. The figure is part of the output of my Matlab code. It is a plot of the Earth with the known satellite's orbit, the estimated orbit of the unknown object, a line to perogee for the unknown object, and lines to the positions of the satellite where the angle measurements take place at. In Matlab, the plot is 3-Dimensional and able to be rotated to get a better sense of the orbits being generated. The parameters are as follows:

$$\begin{bmatrix} e_{sat} \\ a_{sat} \\ \Omega_{sat} \\ i_{sat} \\ \omega_{sat} \\ \theta_{sat} \end{bmatrix} = \begin{bmatrix} 0 \\ 8000 \\ 0 \\ 0 \\ 0 \\ \frac{pi}{4} \end{bmatrix}$$

$$\begin{bmatrix} time \end{bmatrix} = \begin{bmatrix} 0 & 300 & 600 \end{bmatrix}$$

$$\begin{bmatrix} azimuth \end{bmatrix} = \begin{bmatrix} 10 & 89 & 90 \end{bmatrix}$$

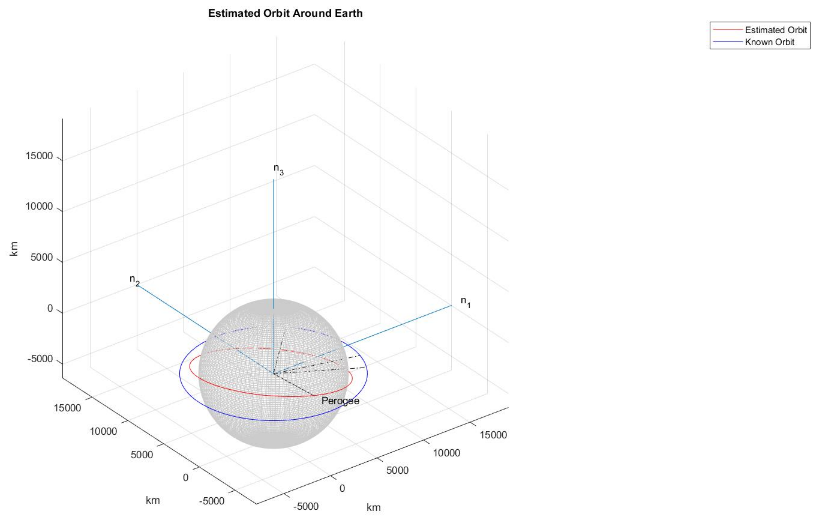


Figure 3: Results for the Stated Parameters (View 1)

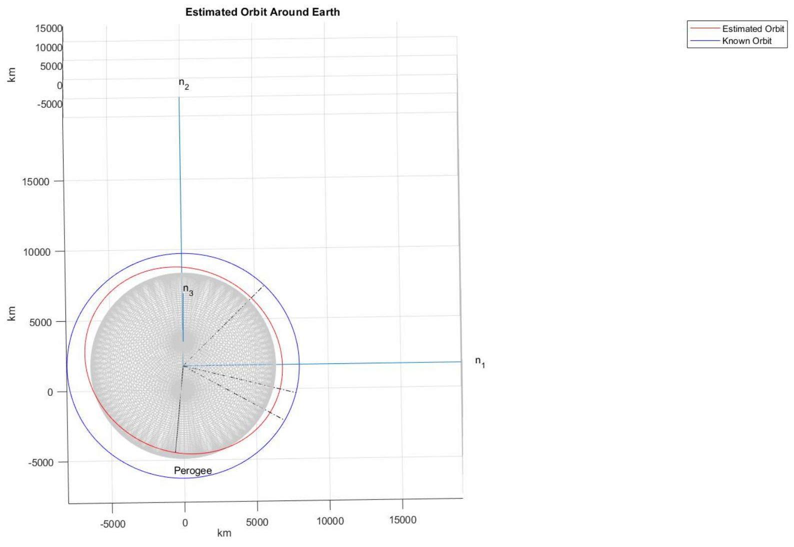


Figure 4: Results for the Stated Parameters (View 2)

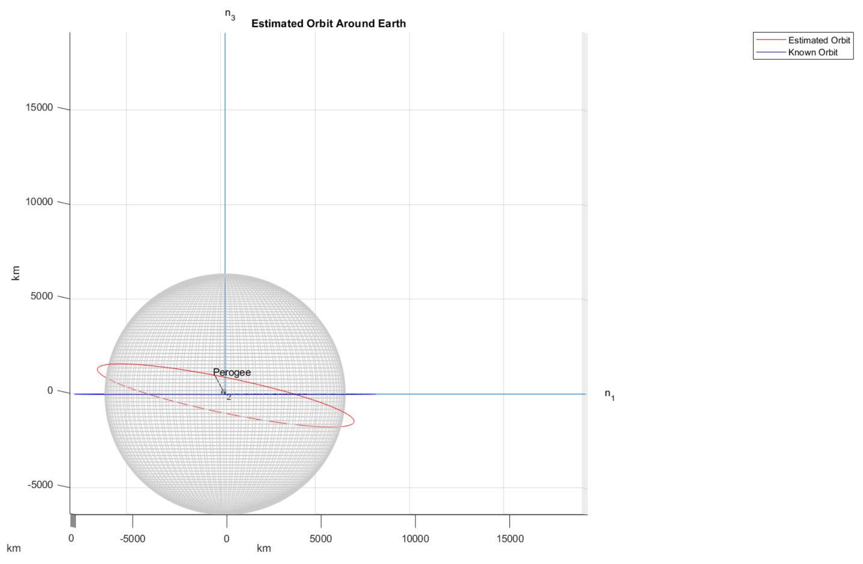
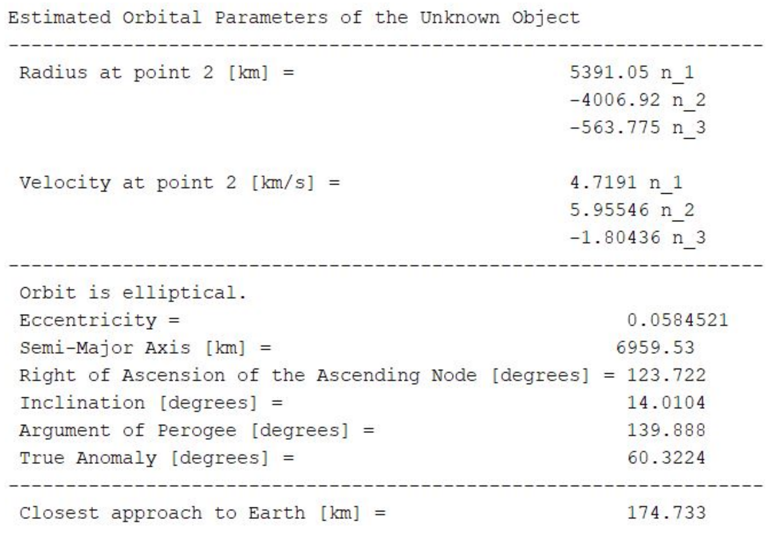


Figure 5: Results for the Stated Parameters (View 3)

$$[altitude] = \begin{bmatrix} 0 & 20 & 40 \end{bmatrix}$$

In addition to the graphic plot of the orbits, figure 6 shows an image of the command line output in Matlab for the same parameters.



```
Estimated Orbital Parameters of the Unknown Object
-----------------------------------------------------------------
 Radius at point 2 [km] =                        5391.05 n_1
                                                 -4006.92 n_2
                                                 -563.775 n_3

 Velocity at point 2 [km/s] =                    4.7191 n_1
                                                 5.95546 n_2
                                                 -1.80436 n_3
-----------------------------------------------------------------
 Orbit is elliptical.
 Eccentricity =                                       0.0584521
 Semi-Major Axis [km] =                              6959.53
 Right of Ascension of the Ascending Node [degrees] = 123.722
 Inclination [degrees] =                              14.0104
 Argument of Perogee [degrees] =                      139.888
 True Anomaly [degrees] =                             60.3224
-----------------------------------------------------------------
 Closest approach to Earth [km] =                     174.733
```

Figure 6: Command Line Results for Stated Parameters

I am very pleased with these results. The algorithm works well to estimate the orbits of unknown objects and the plots I was able to create really help visualize what's going on. These parameters took me a while to reach, usually the unknown object's orbit isn't so nearly well behaved. Small changes in angle parameters, the timescale, or even the satellite's orbit can have large effects on the estimated orbit.

During the derivations, Curtis makes several simplifying assumptions. Among those, it is assumed that the timescale is significantly small. This is to make any higher order terms of the Lagrange coefficients effectively become 0. As a result though, the algorithm only works when the timescale between measurements is very small. In the case picture above, the whole thing takes place within 10 minutes. Additionally, some terms were linearized, which limits the accuracy of the derivations. Figure 7 shows the output when all the parameters are the same, but the inclination of the satellite changes from 0 degrees to 45 degrees.

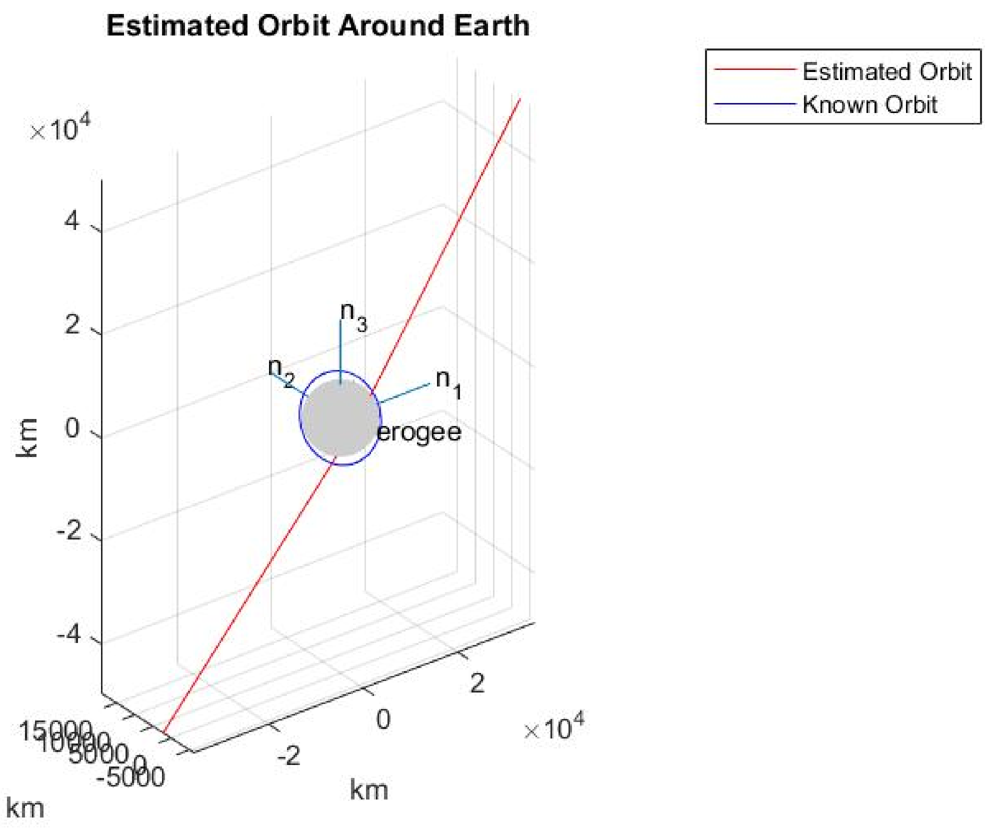


Figure 7: Results for Stated Parameters with Different Inclination

The small change in only one of the 15 total parameters yielded a wildly different result. In this case, the estimated orbit is hyperbolic in nature. Not only is it highly hyperbolic, it is also impossible. If one were to look closely, you would notice that the orbit is estimated as passing through the Earth. So the object would likely not be orbiting the Earth for very long. This results goes to show though that the algorithm is very finicky. Small changes to the initial parameters can have a large impact on the final estimates.

6 Conclusion and Recommendations

The algorithm I developed works very well, but in limited cases. Due to the simplifying assumptions pointed out earlier, the algorithm is really only accurate to a small degree and even then only when the timescale is very small. During my testing, I found that any timescale larger than several hours was too great and would ultimately lead to wildly inaccurate results.

A large portion of the trouble that I had in creating this project was with verifying my results. As I was changing parameters, it was hard to know for certain whether or not the data I found was good or bad. In the end, I was able to validate and verify my code against several examples and problems from the textbook. In order to emulate the approach from the textbook however, my satellite had to essentially orbit the Earth at the same speed and location as a ground station on Earth, something which is impossible, but would still produce correct results. Taking this approach allowed me to locate and fix various bugs in my code until I was happy that results were sufficiently accurate.

All told, this project utilized a lot of what was taught in class and then added onto it some. Fighting with the code and the physics behind this problem gave me a much better understanding of orbital mechanics and helped immensely with the simpler problems we came across in class. This project covers everything from the two-body problem to orbital elements and everything in-between.

If anyone were to use this code in the future, my major recommendation is to be aware of the risks. The results may seem incredibly accurate and well-polished, but a lot of simplifying assumptions were made. People unfamiliar with the code may try to use parameters or timescales well outside the capabilities of the code. Initially, I wanted to write this code to be able to detect if space debris would impact a satellite or not. After testing, I don't believe the code is accurate to a scale that would prove reliable for deciding whether or not a several meter wide satellite would be in danger of impact. Other, more reliable, methods exist and produce better results than those here, but the code here is excellent for getting a baseline understanding of the orbits.

7 Appendix

Included here is a copy of the main script used to run the algorithm developed for the project. In addition to the main script, the following functions are needed from Appendix D in the textbook:

1. *f_and_g*
2. *kepler_E*
3. *kepler_U*
4. *posroots*
5. *stumpC*
6. *stumpS*

I won't be including these functions in the appendix, but the final function needed is one I wrote myself called *satellite* that will be included in a section below the main script.

7.1 Matlab Code - Main Script

Spaceflight Mechanics - Semester Project - Spring 2019

Angles Only Orbit Determination of Unknown Object from a Known Satellite

Contents

Administrative Remarks

```
%Author: Johnathan Corbin
%February 9, 2019
%--------------------------------------------------------------------------
%Assumptions made about the system:
% - The two body problem applies.
% - This system takes place around the Earth (u = 398600).
% - The Earth is a perfect sphere so as to ignore effects of precession.
% - I use the topocentric horizon reference frame on the spacecraft to
%    create the direction unit vectors to the unknown object from a given
%    altitude and azimuth angle seen from the spacecraft.
% - The timescale between angle measurements must be fairly small so that
%    only the first terms of the Lagrange coefficients are large enough to
%    significantly contribute.
% - The orbit of the satellite taking the measurements is known and that a
%      position vector can be made to that satellite at each instance when
%      an angle measurement is made.
% - Values of the estimated orbit are output at the end of the script, any
%      value that is a rotation value are in degrees, not radians.
%--------------------------------------------------------------------------
% Derivations of the algorithm and some of the required functions were
% pulled from Orbital Mechanics for Engineering Students by Howard D.
% Curtis.
%--------------------------------------------------------------------------
%Required functions:
% f_and_g, kepler_E, kepler_U, orbit_elements, satellite, posroot, stumpC,
% stumpS
%--------------------------------------------------------------------------
```

Clearing figures and command line

```
clear, clc
close all
```

Declaring constants

```
global u

u = 398600; %gravitational parameter for earth
r_earth = 6378; %radius of earth
deg_rad = pi / 180; %converting degree to rad
```

Measured angle values

```
azimuth = [10, 89,  90] * deg_rad; %Azimuth angle of unknown object at times
%10 20 30 to generate hyperbolic
altitude = [0, 20, 40] * deg_rad; %Altitude angle of unkown object at times
```

Time vectors

```
t = [0, 5*60, 10*60]; %Array for time values, seconds
tau = [t(1) - t(2), 0, t(3) - t(2)]; %Calculating time intervals
T = tau(3) - tau(1);
```

Calculate position R at the three times of the known satellite from known satellite orbital parameters

```
e_sat = 0;
a_sat = 8000;
Omega_sat = 0;
i_sat = 0;
omega_sat = 0;
anomaly = pi / 4;

[sat_orbit, R] = satellite(e_sat, a_sat, Omega_sat, omega_sat, i_sat, anomaly, t);

%R = [5582.84, 0, 3073.9;...
 %   5581.5, 122.122, 3073.9;...
  %  5577.7, 244.186, 3073.9];
%Random position vectors to test algorithm
```

Determining unit vectors from satellite to unknown object

```
p = zeros(3); %Creating array for direction cosine vectors at the three times

%For loop to calculate the direction cosine vectors at each time
for i = 1:3
    p(i,1) = cos(altitude(i)) * sin(azimuth(i));
    p(i,2) = cos(altitude(i)) * cos(azimuth(i));
    p(i,3) = sin(altitude(i));
end

%p = [.846428, 0, .532504;...
 %   .74929, .463023, .47347;...
  %  .529447, .777163, .340152];
% Random direction cosine vectors to test algorithm
```

Cross products among the direction cosine vectors

```
p1 = cross(p(2,:),p(3,:));
p2 = cross(p(1,:),p(3,:));
p3 = cross(p(1,:),p(2,:));

D0 = dot(p(1,:),p1);
```

Computing the six scalar quantities

```
D = zeros(3);
for q = 1:3
   D(q,1) = dot(R(q,:), p1);
   D(q,2) = dot(R(q,:), p2);
   D(q,3) = dot(R(q,:), p3);
end
```

Calculating A and B

```
A = (-D(1,2)*tau(3)/T + D(2,2) + D(3,2) * tau(1) / T) / D0;
B = (D(1,2)*(tau(3)^2 - T^2)*tau(3)/T + D(3,2)*(T^2 - tau(1)^2) * tau(1)/T)/(6 * D0);
```

Calculating E

```
E = dot(R(2,:), p(2,:));
```

Calculating a, b, and c

```
a = -(A^2 + 2*A*E + (norm(R(2,:)))^2);
b = -2*u*B*(A + E);
c = -(u^2 * B^2);
```

Calculating the roots numerically

```
Roots = roots([1 0 a 0 0 b 0 0 c]);
x = posroot(Roots);
```

Calculating the Lagrange coefficients

```
f1 = 1 - (u*tau(1)^2/x^3)/2;
f3 = 1 - (u*tau(3)^2/x^3)/2;

g1 = tau(1) - (u*(tau(1)/x)^3)/6;
g3 = tau(3) - (u*(tau(3)/x)^3)/6;
```

Calculating position vector magnitudes from satellite to unknown object

```
P1 = 1/D0*((6*(D(3,1)*tau(1)/tau(3) + D(2,1)*T/tau(3))*x^3 + u*D(3,1)*(T^2 - tau(1)^2)*tau(1)/tau(3))/(
P2 = A + u * B / x^3;
P3 = 1/D0*((6*(D(1,3)*tau(3)/tau(1) - D(2,3)*T/tau(1))*x^3 + u*D(1,3)*(T^2 - tau(3)^2)*tau(3)/tau(1))/(
```

Calculating position vectors from earth center to unknown object

```
r1 = R(1,:) + P1*p(1,:);
r2 = R(2,:) + P2*p(2,:);
r3 = R(3,:) + P3*p(3,:);
```

Calculating velocity of unknown object

```
v2 = (1 / (f1*g3 - f3*g1))*(-f3*r1 + f1*r3);
```

Method to improve accuracy of results taken from Curtis

```
P1_old = P1;  P2_old = P2;  P3_old = P3;
diff1    = 1;     diff2    = 1;     diff3    = 1;
n    = 0;
nmax = 2000;
tol  = 1.e-10;

%...Iterative improvement loop from Curtis:
while ((diff1 > tol) && (diff2 > tol) && (diff3 > tol)) && (n < nmax)
    n = n+1;

%...Compute quantities required by universal kepler's equation:
    ro  = norm(r2);
    vo  = norm(v2);
    vro = dot(v2,r2)/ro;
    alpha   = 2/ro - vo^2/u;

%...Solve universal Kepler's equation at times tau1 and tau3 for
%   universal anomalies x1 and x3:
    x1 = kepler_U(tau(1), ro, vro, alpha);
    x3 = kepler_U(tau(3), ro, vro, alpha);

%...Calculate the Lagrange f and g coefficients at times tau1
%   and tau3:
    [ff1, gg1] = f_and_g(x1, tau(1), ro, alpha);
    [ff3, gg3] = f_and_g(x3, tau(3), ro, alpha);

%...Update the f and g functions at times tau1 and tau3 by
%   averaging old and new:
    f1    = (f1 + ff1)/2;
    f3    = (f3 + ff3)/2;
    g1    = (g1 + gg1)/2;
    g3    = (g3 + gg3)/2;

%...Equations 5.96 and 5.97:
    c1    =  g3/(f1*g3 - f3*g1);
    c3    = -g1/(f1*g3 - f3*g1);

%...Equations 5.109a, 5.110a and 5.111a:
    P1  = 1/D0*(       -D(1,1) + 1/c1*D(2,1) - c3/c1*D(3,1));
    P2  = 1/D0*(   -c1*D(1,2) +       D(2,2) -    c3*D(3,2));
    P3  = 1/D0*(-c1/c3*D(1,3) + 1/c3*D(2,3) -       D(3,3));

%...Equations 5.86:
    r1 = R(1,:) + P1*p(1,:);
    r2 = R(2,:) + P2*p(2,:);
    r3 = R(3,:) + P3*p(3,:);

%...Equation 5.118:
    v2    = (-f3*r1 + f1*r3)/(f1*g3 - f3*g1);

%...Calculate differences upon which to base convergence:
    diff1 = abs(P1 - P1_old);
    diff2 = abs(P2 - P2_old);
    diff3 = abs(P3 - P3_old);

%...Update the slant ranges:
    P1_old = P1;  P2_old = P2;  P3_old = P3;
end
%...End iterative improvement loop
```

Determing orbit elements of unknown object from r2 and v2

```
[mag_e, a, theta, Omega, inc, omega] = orbit_elements(r2, v2);
```

Plotting both orbits of known satellite and unknown object

```
    if (mag_e > 1)
        l_limit = -pi / 2;
        u_limit = pi / 2;
    else
        l_limit = 0;
        u_limit = 2 * pi;
    end

    %Converting estimated orbital parameters to radians
    W = Omega * deg_rad;
    w = omega * deg_rad;
    inclination = inc * deg_rad;

    %Computing direction cosine matrix from perifocal to N for unknown object
    C_N_PF_obj = [-sin(W)*cos(inclination)*sin(w)+cos(W)*cos(w), -sin(W)*cos(inclination)*cos(w)-cos(W)*
        cos(W)*cos(inclination)*sin(w)+sin(W)*cos(w), cos(W)*cos(inclination)*cos(w) - sin(W)*sin(w), s
        sin(inclination)*sin(w), sin(inclination)*cos(w), cos(inclination)];

    %Magnitude of momentum for unknown object
    h = norm(cross(r2, v2));

    %For loop to calculate X,Y,Z positions of the estimated orbit
    counter = 1;
    for angle = l_limit:.001:u_limit
        radius_polar = [h^2/(u * (1 + mag_e * cos(angle)));0;0];
        C_PF_polar_obj = [cos(angle), sin(angle), 0;...
        -sin(angle), cos(angle), 0;...
        0, 0, 1].';
       radius_pf = (C_PF_polar_obj * radius_polar);
       radius = (C_N_PF_obj * radius_pf).';
       for z = 1:3
           coordinate(z, counter) = radius(z);
       end
       counter = counter + 1;
    end
    per = 0;

    radius_polar = [h^2/(u * (1 + mag_e * cos(per)));0;0];
        C_PF_polar_obj = [cos(per), sin(per), 0;...
        -sin(per), cos(per), 0;...
        0, 0, 1].';
    radius_pf = (C_PF_polar_obj * radius_polar);
    radius = (C_N_PF_obj * radius_pf).';
    perogee = radius;

    %Plot the previously determined orbit
    plot3(coordinate(1,:), coordinate(2,:), coordinate(3,:), 'r')
    hold on

    %Plot the known orbit
    plot3(sat_orbit(1,:), sat_orbit(2,:), sat_orbit(3,:), 'b')

    %Plot a sphere to represent the Earth
    [xx, yy, zz] = sphere(100);
    lightGrey = 0.8*[1 1 1];
    surface(r_earth*xx, r_earth*yy, r_earth*zz, 'FaceColor', 'none', 'EdgeColor', lightGrey);
```

```
    %Draw and label the axes
    line([0 3*r_earth],   [0 0],   [0 0]); text(3*r_earth+1000,   0,   0, 'n_1')
    line(  [0 0], [0 3*r_earth],   [0 0]); text(  0, 3*r_earth+1000,   0, 'n_2')
    line(  [0 0],   [0 0], [0 3*r_earth]); text(  0,   0, 3*r_earth+1000, 'n_3')

    %Draw line to perogee
    line([0 perogee(1)], [0 perogee(2)], [0 perogee(3)], 'Color', 'k');
    text(perogee(1)*1.2, perogee(2)*1.2, perogee(3)*1.2, 'Perogee')

    %Draw line to each vector R of the known satellite at times
    line([0 R(1,1)], [0 R(1,2)], [0 R(1, 3)], 'Color', 'k', 'Linestyle', '-.')
    line([0 R(2,1)], [0 R(2,2)], [0 R(2, 3)], 'Color', 'k', 'Linestyle', '-.')
    line([0 R(3,1)], [0 R(3,2)], [0 R(3, 3)], 'Color', 'k', 'Linestyle', '-.')

    %Miscellanious graph adjustments
    grid on
    axis equal
    xlabel('km')
    ylabel('km')
    zlabel('km')
    view(3)
    title('Estimated Orbit Around Earth')
    legend('Estimated Orbit', 'Known Orbit')

    if (mag_e > -1) && (mag_e < 1)
        orbit = 'elliptical';
    else
        orbit = 'hyperbolic';
    end
```

Ouputting parameters to Command Line

```
fprintf('Estimated Orbital Parameters of the Unknown Object\n')
fprintf('-----------------------------------------------------------------')
fprintf('\n Radius at point 2 [km] =                          %g n_1', r2(1))
fprintf('\n                                                   %g n_2', r2(2))
fprintf('\n                                                   %g n_3\n', r2(3))
fprintf('\n Velocity at point 2 [km/s] =                      %g n_1', v2(1))
fprintf('\n                                                   %g n_2', v2(2))
fprintf('\n                                                   %g n_3\n', v2(3))
fprintf('-----------------------------------------------------------------')
fprintf('\n Orbit is %s.', orbit)
fprintf('\n Eccentricity =                                        %g', mag_e)
fprintf('\n Semi-Major Axis [km] =                               %g', a)
fprintf('\n Right of Ascension of the Ascending Node [degrees] = %g', Omega)
fprintf('\n Inclination [degrees] =                               %g', inc)
fprintf('\n Argument of Perogee [degrees] =                      %g', omega)
fprintf('\n True Anomaly [degrees] =                              %g', theta)
fprintf('\n')
fprintf('-----------------------------------------------------------------')
fprintf('\n Closest approach to Earth [km] =                      %g', norm(perogee) - r_earth);
fprintf('\n')
```

```
Estimated Orbital Parameters of the Unknown Object
-----------------------------------------------------------------
 Radius at point 2 [km] =                        5391.05 n_1
                                                 -4006.92 n_2
                                                 -563.775 n_3

 Velocity at point 2 [km/s] =                    4.7191 n_1
                                                 5.95546 n_2
                                                 -1.80436 n_3
-----------------------------------------------------------------
 Orbit is elliptical.
 Eccentricity =                                       0.0584521
 Semi-Major Axis [km] =                              6959.53
 Right of Ascension of the Ascending Node [degrees] = 123.722
 Inclination [degrees] =                              14.0104
 Argument of Perogee [degrees] =                      139.888
 True Anomaly [degrees] =                             60.3224
-----------------------------------------------------------------
 Closest approach to Earth [km] =                     174.733
```

7.2 Matlab Code - *satellite* Function

```
function [sat_orbit, position] = satellite(e, a, Omega, omega, i, theta, t)

    global u

    h = sqrt(u * a * (1-e^2));

    %Computing direction cosine matrix from perifocal to N for unknown object
    C_N_PF_obj = [-sin(Omega)*cos(i)*sin(omega)+cos(Omega)*cos(omega), -sin(Omega)*cos(i)*cos(omega)-cos
        cos(Omega)*cos(i)*sin(omega)+sin(Omega)*cos(omega), cos(Omega)*cos(i)*cos(omega) - sin(Omega)*si
        sin(i)*sin(omega), sin(i)*cos(omega), cos(i)];

    %Calculating initial eccentric anomaly from given theta initial
    E0 = 2 * atan2((tan(theta / 2) / sqrt((1 + e)/(1 - e))), 1);

    %Calculating time since pergogee t0
    t0 = (E0 - e*sin(E0)) / sqrt(u / a^3);

    %Calculating the mean anomalies at times
    M1 = sqrt(u / a^3) * (t(2) - t0);
    M2 = sqrt(u / a^3) * (t(3) - t0);

    %Calculating eccentric anomaly at times
    E1 = kepler_E(e, M1);
    E2 = kepler_E(e, M2);

    %Calculating actual anomaly values at times
    theta1 = 2 * atan2(sqrt((1 + e)/(1 - e)) * tan(E1 / 2), 1);
    theta2 = 2 * atan2(sqrt((1 + e)/(1 - e)) * tan(E2 / 2), 1);

    %For loop to calculate X,Y,Z positions of the estimated orbit
    counter = 1;
    coordinate = zeros(3, 629);
    for angle = 0:.01:(2*pi)
        radius_polar = [h^2/(u * (1 + e * cos(angle)));0;0];
        C_PF_polar_obj = [cos(angle), sin(angle), 0;...
        -sin(angle), cos(angle), 0;...
        0, 0, 1].';
       radius_pf = (C_PF_polar_obj * radius_polar);
       radius = (C_N_PF_obj * radius_pf).';
       for z = 1:3
           coordinate(z, counter) = radius(z);
       end
       counter = counter + 1;
    end

    %Determining position vector at the three theta values
    radius_polar = [h^2/(u * (1 + e * cos(theta)));0;0];
        C_PF_polar_obj = [cos(theta), sin(theta), 0;...
        -sin(theta), cos(theta), 0;...
        0, 0, 1].';
    radius_pf = (C_PF_polar_obj * radius_polar);
    radius = (C_N_PF_obj * radius_pf).';
    R(1, :) = radius;

    radius_polar = [h^2/(u * (1 + e * cos(theta1)));0;0];
        C_PF_polar_obj = [cos(theta1), sin(theta1), 0;...
        -sin(theta1), cos(theta1), 0;...
        0, 0, 1].';
    radius_pf = (C_PF_polar_obj * radius_polar);
    radius = (C_N_PF_obj * radius_pf).';
    R(2, :) = radius;

    radius_polar = [h^2/(u * (1 + e * cos(theta2)));0;0];
        C_PF_polar_obj = [cos(theta2), sin(theta2), 0;...
        -sin(theta2), cos(theta2), 0;...
        0, 0, 1].';
    radius_pf = (C_PF_polar_obj * radius_polar);
    radius = (C_N_PF_obj * radius_pf).';
    R(3, :) = radius;

    sat_orbit = coordinate;
    position = R;
end
```

References

Curtis, H. D. (2013). *Orbital mechanics for engineering students*. Butterworth-Heinemann.