# Determining Orbital Elements Using Methods In Computer Programming

Matthew Altis
*Georgia Institute of Technology, Atlanta, Georgia, 30332, United States of America*

## I. Abstract

Computing orbital elements, position, and velocity from a given range and range rate vector is a process that involves a considerable amount of computation. While the mathematics involved is not complicated, it quickly becomes very time-consuming to solve for the numerous different orbital elements. In the event where multiple orbits need to be calculated, a computer program is essential to calculate the elements in a timely manner. This report investigates the methodology and approach used when converting the mathematical equations involved with orbital elements calculation into a coherent, comprehensive computer code. The use of different programming concepts, such as function writing, vector/matrix operations, conditional statements, as well as iteration are discussed as to how they can be used to perform the necessary computations.

# II. Nomenclature

$\theta g$ = Greenwich Sidereal Time

D = *Days*

$\lambda_E$ = *Longitude*

$\theta =$ Local Sidereal Time

*L* = *Latitude*

$\vec{r} = Position\ Vector$

$\vec{V} = Velocity\ Vector$

e = Eccentricity

*p* = Semi-Latus Rectum

a = *Semi-Major Axis*

$\omega =$ Argument of Periapsis

$\Omega =$ Ascending Node

i = *inclination angle*

*v* = True Anomaly

E = Eccentric Anomaly

*M* = *Mean Anomaly*

# III. Introduction

Before investigating the specific computations that were programmed into the code, it is necessary to understand the problem at hand, as well as the initial conditions provided. This problem involves processing the range and range rate vectors of a satellite recorded by a radar station at a given latitude and longitude. The date and time of the first radar station measurement are provided. Additionally, a time of flight from the first measurement is given. The first task involves converting the recorded range and range rate vectors to position and velocity vectors within the Earth-centered, equatorial coordinate system. With these vectors computed, the next task is determining the orbital elements for the recorded object in orbit. Then, considering the provided time of flight, the final task is to determine the object's true anomaly and new position/velocity vectors after the elapsed time.

# IV. Approach

To solve the entirety of the problem in a timely and convenient fashion, a single Matlab function was written. The function is listed below and takes in, as input, the range, range rate, latitude, longitude, theta g0, date of given theta g0, date of radar measurement, and the time of flight.

```
function[Position1,Velocity1,i,omega,e,p,a,w,v,v2,Position2,Velocity2,IterationHist]=ISSFinder2(Range,Range_Rate,Latitude,Longitude,Theta_g0,Year,Month1,Day1,Month2,Day2,Hour2,Minute2,TOF)
```

To begin the computation, the first step was to get the number of time passed from the initial theta g0 date to the time of initial radar observation. The values for theta_g0 and day number were input to obtain the Greenwich sidereal time:

$$\theta g = \theta g0 + 1.00273779093 \cdot 360° \cdot D$$

The initial result for sidereal time is a very large angle, so it is necessary to reduce the angle by dividing it by 360, multiplying the whole number result of the division by 360, and then subtracting it from the initial sidereal time. In Matlab, the whole number was obtained by rounding down using:

New Number = floor(Old Number)

With the corrected Greenwich sidereal time and given longitude, the local sidereal time can be obtained using:

$$\theta = \theta g + \lambda_E$$

Using $\theta$ and the given Latitude, a transformation matrix can be produced:

$$[D]^{-1} = \begin{matrix} sinLcos\theta & -sin\theta & cosLcos\theta \\ sinLsin\theta & cos\theta & cosLsin\theta \\ -cosL & 0 & sinL \end{matrix}$$

Once the range is converted to the topocentric horizon frame, we can obtain position and range rate in the planet-centered system by multiplying the above matrix by our range and range-rate vectors.

It is important to note that the following formula was used to code in the matrix multiplication, which is used multiple times throughout the report:

```
X= D11.*rh(1) + D12.*rh(2) + D13.*rh(3);

Y= D21.*rh(1) + D22.*rh(2) + D23.*rh(3);

Z= D31.*rh(1) + D32.*rh(2) + D33.*rh(3);

Position1 = [X,Y,Z];
```

In the above code, D## is the respective element of the transformation matrix, while rh(#) is a component of the vector being multiplied (position or range rate vectors). This multiplication gives us our correct position vector, but for velocity, we must correct for the angular velocity of the earth:

$\vec{V} = \dot{\rho} + (\vec{w}\ x\ \vec{r})$ where $\vec{w}$ = [0 0 .053883] rad/TU

Another common operation used throughout the report is cross product, where the following code format was used for A X B:

A X B = (A2*B3-B2*A3)$\hat{X}$ – (A1*B3-B1*A3)$\hat{Y}$ + (A1*B2-B1*A2)$\hat{Z}$

Before the six fundamental orbital elements could be computed, the following three vectors were computed:

$$\vec{h} = \vec{r}\ x\ \vec{v}$$

$$\vec{n} = \hat{z}\ \text{x}\ \vec{h}$$

$$\vec{e} = \frac{1}{\mu}[\left(V^2 - \frac{\mu}{r}\right)\vec{r} - (\vec{r} \cdot \vec{v})\vec{v}]$$

For the dot product of two vectors A and B, the following code was used:

A · B = (A1*B1) + (A2*B2) +(A3*B3)

The six primary orbital elements were computed with the following formulas:

e = $||\vec{e}|| = \sqrt{{e_x}^2 + {e_y}^2 + {e_z}^2}$ (This formula was used to compute all vector magnitudes throughout the report)

$$p = \frac{h^2}{\mu}$$

$$a = \frac{p}{(1-e^2)}$$

$$\omega = cos^{-1}(\frac{\vec{n}\cdot\vec{e}}{ne})$$

$$\Omega = cos^{-1}\left(\frac{n_x}{n}\right)$$

$$i = cos^{-1}(\frac{h_z}{h})$$

$$v = cos^{-1}(\frac{\vec{e}\cdot\vec{r}}{er})$$

For the orbital elements $\Omega, \omega, v$, "if" statements were coded in to ensure that each of these angles were in the correct half plane. Adjustments were made to the angle by subtracting the incorrect value from 360 to obtain the correct angle. Below is the code format used for each angle:

```
if w < 180 & evec(3) > 0
    w = w;
elseif w > 180 & evec(3) > 0
    w = 360 - w;
elseif w < 180 & evec(3) < 0
    w = 360 - w;
elseif w > 180 & evec(3) < 0
    w = w;
end
```

With the corrected orbital elements, the time of flight must now be taken into consideration. Finding the true anomaly after the elapsed time involves iteratively solving Kepler's problem using a while loop. However, a couple fundamental values must be computed first:

$$Period = 2\pi\sqrt{\frac{a^3}{\mu}}$$

$$M = E - esinE = \sqrt{\frac{\mu}{a^3}}(TOF) - 2\pi k + M_0$$

The above values for *k* (number of periapsis passages) and $M_0$ were computed using the period and true anomaly, respectively. Solving Kepler's problem for the new eccentric anomaly E became a matter of implementing the following while loop:

```
E=.5;
dM=1;
num=0;
b=[];
c=[];
d=[];
f=[];
while abs(dM) > 1.e-7
    b=[b;E];
    dM = M - (E-(e.*sin(E)));
    c=[c;dM];
    dMdE = 1-(e.*cos(E));
    d=[d;dMdE];
    E = E + (dM./dMdE);
    f=[f;E];
    num=num+1;
end
iteration=[0:(num-1)];
iteration=transpose(iteration);
IterationHist=table(iteration,b,c,d,f);
IterationHist.Properties.VariableNames = {'i',
'Ei','dM','dMdE','EiNext'};
```

The above loop takes in initial values for E, dM, and will continue performing Kepler's problem until dM is below a given threshold value. When the loop finishes, it outputs the new value for the eccentric anomaly. Additionally, the table keeps track of the iteration number, E, dM, dM/dE, and EiNext. These values are then placed into a table, which allows one to see the values at each step in the iteration.

Using the new outputted value for E, the new true anomaly can be calculated using the following:

$$v = cos^{-1}(\frac{cosE-e}{1-ecosE})$$

Using this new true anomaly, as well as our earlier orbital elements (*p*, *r*, and *e)* to obtain our new position and velocity in the perifocal-eccentricity system:

$$\vec{r}_w = rcosv\hat{X}_w + rsinv\hat{Y}_w$$

$$\vec{V}_w = \sqrt{\frac{\mu}{p}}\left(-sinv\hat{X}_w + (e + cosv)\hat{Y}_w\right)$$

The final step in our computation is to multiply a rotation matrix [R] by both the position and velocity vectors obtained above. Using our orbital elements $\omega, \Omega, i$, the 3x3 rotation matrix can be computed and multiplied to obtain our final position and velocity vectors in the earth-centered, equatorial-equinox coordinate system.

# V. Results and Discussion

Inputting the range and range rate vectors given in the project description yielded the following position and velocity vectors for the initial time of radar observation:

$\vec{r} = [-.6925, -.4304, .6848]$ DU

$\vec{v} =$ [.7235, -.4828, .4275] DU/TU

When comparing the magnitudes of velocity and position with the given range and range rate, we can see that the values for the position vector expectedly increased, as the distance is taken from the center of the earth rather than from a radar station on the surface.

The following orbital elements were obtained from the code:

*e* = .00048

*p* = 1.0651 DU

*a* = 1.0651 DU

*w* = 121°

$\Omega = 169.5037°$

*i* = 51.263°

*v* = 294.5381°

Based on the obtained orbital elements, the most noticeable observation is the very small eccentricity value. This is expected, as the ISS has a nearly circular orbit around the earth. Additionally, due to this orbital shape, our semi-latus rectum and semi-major axis (*p* and *a)* are almost identical in value.

Performing the iteration to solve for Kepler's problem yielded a new true anomaly 45 minutes later:

$V_2 = 148.2242°$

The iteration steps Matlab used to solve Kepler's problem are shown below:

| i | Ei | dM | dMdE | EiNext |
|---|---|---|---|---|
| 0 | 0.5 | -1.5509e+06 | 0.99957 | -1.5515e+06 |
| 1 | -1.5515e+06 | 659.73 | 1.0004 | -1.5509e+06 |
| 2 | -1.5509e+06 | 0.27179 | 1.0005 | -1.5509e+06 |
| 3 | -1.5509e+06 | 6.512e-06 | 1.0004 | -1.5509e+06 |
| 4 | -1.5509e+06 | 0 | 1.0004 | -1.5509e+06 |

Because the elapsed time given in the problem statement is only a fraction of the orbital period of the ISS, the result can be interpreted as the satellite having traveled a little over 210 degrees over the 45 minutes, or roughly 60% of one full revolution around earth.

The final output of the program included the new position and velocity vectors of the ISS after the 45 minutes:

$\vec{r}_2 = [.1355, .6525, -.8305]$ DU

$\vec{v}_2 = [-.9508, .1846, -.0104]$ DU/TU

For the final position and velocity, we can expect the position of the satellite to have different signs for each position value. Additionally, the velocity vector is expected to point in different directions at different points in the orbit.

## VI. Conclusion

This project demonstrates how a multi-faceted problem with numerous steps can be implemented into a single Matlab function. There are plenty of advantages in using a program for this orbital determination process, namely the ability to easily change the input variables. Instead of working out each individual scenario by hand, a program can quickly give results for a large number of different inputs. However, it must be noted that the results obtained by the code may be slightly different than those obtained through hand-calculation, especially considering the fact that Matlab stores each computation results precisely rather than rounding. Additionally, choosing a different threshold value when solving Kepler's problem iteratively may yield a slightly different true anomaly value. Finally, the assumption of earth being spherical implemented in the code causes results from the program to

be very slightly different than their true values. Overall, the small source of error in the code is made up for by its efficiency in solving many different inputs in a short period of time.

# VII. Appendix

In order to run the Matlab code, first it is necessary to ensure that the Matlab file (.m) is downloaded and is present in the current accessed folder within Matlab. The next step is to enter the input variables into the command window, which is typically the interface that is on the lower half of the screen. For the particular problem statement in the project description, the following input variables may be copy-pasted into the command window:

Range = [-.118260, -.080977, .05515];

Range_Rate = [-.513194, .776045, .001608];

Latitude = 33.7718;

Longitude = 84.395;

Theta_g0 = 340.62;

Year = 2020;

Month1 = 9;

Day1 = 1;

Month2 = 9;

Day2 = 18;

Hour2 = 20;

Minute2 = 15;

TOF = 2700;

With all of the above values stored in the command window, the code can be ran by typing the following and pressing “enter”:

```
[Position1,Velocity1,i,omega,e,p,a,w,v,v2,Position2,Velocity2,IterationHist]=ISSFinder2(Range,Range_Rate,Latitude,Longitude,Theta_g0,Year,Month1,Day1,Month2,Day2,Hour2,Minute2,TOF)
```

The function is interactive and can be modified to test out a variety of different conditions. Any of the above variables can be modified as long as certain formatting and notations are kept constant:

- Range and Range_Rate must be expressed as vectors in XYZ format: [X Y Z]
- For Latitude, north is understood as positive. If testing a south latitude, change the value to negative
- For Longitude, West is understood as positive. If testing an east longitude, change the value to negative
- Month, days, hours, minutes, years must be expressed as whole number integers
- Month1 and Day1 correspond to the date and time that theta_g0 is measured
- Theta_g0 must be inputted in degrees
- Hours must be expressed in 24 hour (military) time. For example, 8 PM would be entered as 20
- TOF (time of flight) must be expressed in seconds