FIREBIRD Orbital Simulation 2009 Documentation

Ian Lyon

Contents

1 Introduction

1.1 Project Background

- FIREBIRD, consisting two identical 1.5u cubesat satellites in low earth orbit (LEO), will monitor the patterns of relativistic electron bursts.
- To do this effectively, the satellites must cover a wide range with respect to one another.
  - This would be accomplished through the use of a spring between each satellite.
  - The creation of an initial relative velocity of 1-3 cm/s would cause the satellites to separate as they proceeded with their orbit.
- The satellite lifespan is very limited.
  - The effective distance of FIREBIRD is around 200-400 kilometers or less.
  - Once FIREBIRD passes this distance limit, the experiment and data collection is over.

1.2 Project Idea

- A slight change in mass changes the ballistic coefficient of an object, and makes it more susceptible to atmospheric drag forces.
- If the ballistic coefficient of the leading orbiting satellite can be decreased by increasing its mass, that satellite will slow down due to drag.

1.3 Benefits

- By increasing the drag of the leading satellite and slowing it down, the relative motion of the satellites can be slowed and reversed.
  - The satellites will still initially move apart at a speed of 1-3 cm/s due to the spring, but the heavier satellite will immediately start to slow down over a long period of time due to drag.

  - They will eventually reach a maximum relative distance, losing that initial relative speed and instead orbiting at the same speed.
  - Since drag forces would still be affecting the heavier satellite more, the relative orbital speed will continue to slow and cause the satellites to move closer together.
  - The satellites will eventually pass one another, moving apart once again.
- This could more than triple the current proposed lifespan of FIREBIRD of 2 - 5 months.
- A feasible mass change that lands within the FIREBIRD mass budget may bring about these desired effects; too much mass added is undesirable, since it would be expensive.

1.4 Proposed Solution

An accurate orbital simulator that mimics the change in drag given a change in mass could predict how much extra mass is required for the desired increase in lifespan.

- The accuracy of this program is of the utmost importance, given the precision needed to pinpoint the correct mass required
- Since time and technology constraints must remain in place for the duration of a ten-week internship at K7MSU, a less accurate substitute for the program can be implemented and used for the analysis.

2 Initial Assumptions

The relative orbital distances must be evaluated through time as accurately as possible. A program should numerically evaluate individual orbital trajectories.

- The program should be written in MATLAB.
- The program should utilize geodetic Newton's Laws of gravity.
  - The earth shall be assumed to be an ellipse, using the J2 Earth gravitational harmonics term (J2 perturbation).
  - The numerical evaluation of physics equations should use Earth-Centric-Inertial Cartesian coordinates.
- The program should be as accurate as possible over the simulation time (evaluated over more than 20 million steps for four-twelve months).
  - Step sizes could be as small as a second, or a tenth of a second for smaller time periods. The step size should be no larger than 3-5 seconds before numerical error becomes quite important.
- The atmospheric drag associated with LEO must be accurately accounted for.
  - The atmospheric density must be represented at required locations above the Earth at every step, provided by an accurate atmospheric model provided by NASA.
    * The location of a satellite in orbit must be geodetically known at every numerical step; coordinate transforms between Earth-Centric-Inertial to Earth-Centric-Earth-Fixed Cartesian coordinates to geodetic coordinates must be performed with every step.

  - The object surface area through time must be accurately represented.
    * The cross-sectional surface area changes through time, dependent on the orientation due to the strength of the magnetic field present at every step (passive attitude control from internal bar magnet). A magnetospheric model from NASA can provide this.
  - The ballistic and drag coefficients of each satellite must be determined.
    * Changes significantly with altitude, cross-sectional surface area, and surface texture. Programs may be present that can approximate drag coefficients non-empirically.

3 Discoveries and Assumption Revision

There is no computer at MSU powerful enough to feasibly fulfill requirements. For the greatest accuracy possible, the program could take months to run if integrated with the NASA magnetospheric and atmospheric models. This project currently requires supercomputers running for several days or weeks to achieve the most accurate results.

Also, there were some irregularities associated with the implementation of Geodetic Orbital Propagation, using the J2 gravitational harmonics term; this portion of the model was decided to be set aside for the purposes of the immediate simulation at hand, assuming atmospheric drag was the overwhelming extra force in play.

In the meantime, while code for the partial implementation of some of these accurate aspects is still present, commented, a few "temporary" simplifying assumptions can be made.

- The earth shall be assumed to be spherical when determining gravity.
- The atmospheric drag associated with LEO must be accurately accounted for.
  - The atmospheric density can be represented by a similar equation that gives quite a bit of error, but approximates the atmospheric density levels given altitude.
    * The altitude can be obtained through conversion from Earth-Centric-Inertial Cartesian coordinates to spherical coordinates, where atmospheric density is a simple function of altitude, assuming atmospheric density is not dependent on latitude, longitude, or solar activity.
  - The object surface area through time can be statically set, since orientation is not known because the magnetic model will not be implemented.
  - The ballistic and drag coefficients of each satellite must be determined.
    * Coefficients of roughly similar satellites can be substituted for this simulation.

4 Processes

What began as an extremely complicated program was inevitably scaled back some, due to technological limitations. The following sections summarize the essential parts of the program design that are integral to the simulation.

4.1 Numerical Computation

When numerically solving differential equations, one must redescribe the equations with code; it will use some initial inputs in an expression, which in turn processes the inputs in the same way a differential equation would. The difference between solving it numerically and finding the true analytical solution is that the numerical

processing is done over a discrete time step, instead of continuously. This time step is called the step size of the numerical process.

No numerical solution is perfect, and this is because the step size exists. For the purposes of this simulation, we will evaluate the differential equation modifies its inputs over a discrete step in time, instead of evaluating it continuously though time. This equation is known as a difference equation.

A difference equation works this way: if a satellite is defined as traveling at a certain speed, in a certain direction, from a certain point, the differential equations that describe its motion would be functions of acceleration, velocity and position. By inputting the initial velocities and positions in all three dimensions, one is able to determine the acceleration of the satellite at that brief moment in time. This means that one will be able to calculate where the satellite will be one time step later, and how fast its going at that next step; velocity gives us change in position over the time step (where will it be now, with respect to where it was before?), and acceleration gives us change in velocity over the time step (how fast will it be now, as opposed to its speed before?). Then, we can use those new values of position and velocity in the same equations, and calculate them again for the next time step. This process is repeated over and over again, until enough "simulation time" has passed for us to see a satisfactory solution curve.

The dangers of this method lie within the step size itself... what happens between each time step? The trick lies in the fact that the velocity and acceleration of the satellite are assumed to be constant throughout the duration of the step, and then changes discretely when the next step begins. This doesn't happen in real life; a satellite's velocity and acceleration are changing constantly and continuously as it moves through space. Acceleration changes in magnitude depending on how far away the satellite is from the Earth, and the velocity is always changing because acceleration is always present. If we assume that they are constant through a step, the satellite may not travel as far as it should if the real velocity actually increases continuously, or it might travel too far if the velocity is supposed to decrease continuously.

This is why numerical solutions are never perfect, and never can be perfect; they are only approximations. There will always be some numerical error associated with these solutions. However, when the step size of the simulation is reduced and more steps are evaluated for the same "simulation time," the error is reduced as well. If it were possible to reduce the step size to 0, the resulting numerical solution would be identical to the true solution of the differential equations, because it would be continuous. If decreasing the step size 0 means that the solution is equal to the true solution of the equations, this means that there would be no error. The goal of numerical computing has to do with obtaining a credible solution by reducing the step size until there is negligible and acceptable error.

The majority of differential equations do NOT have an analytical solution. They must be evaluated numerically, and there are many different algorithms for doing this very thing (some, like the midpoint method, are better than others). This is why numerical computing is so powerful; it is limited only by the computer's ability to perform calculations over as many time steps as possible with the smallest step size possible. Then, any differential equation with initial conditions can be solved.

4.2 For Loop

The key to numerical computation is repetition; an input is put into an equation, which gives an output. The output is then used as an input for the equation again, which in turn gives another output. This can be repeated as many times as needed.

The easiest way to perform many of these calculations many times very quickly is to create a `for` loop. The `for` loop is a very common function in many programming languages, and MATLAB is no exception.

A `for` loop is defined in MATLAB with the following:

```
for i = 1:steps
 % (anything goes here)
end
```

The variable `i` is an index, it essentially indicates what step in the loop is being identified. The `= 1:steps` says that the`i` index will be increased with each iteration of the loop by one, from `1` to whatever `steps` is. The index can make changes in a loop by referencing which step in the loop is active. Simulating the progression of time in a loop is simple:

```
t(1) = 5 ;                % The initial value of t

dt = 2 ;                  % The step size of t

for i = 1:4               % The loop will perform four iterations

   t(i+1) = t(i) + dt ;   % This equation says that "t at this step, plus the step size,
                          % equals t in the following step, or this step plus one"

end                       % Ends the loop
```

This is saying that the initial value of `t` is 5 seconds, and`dt`, or change in `t` (the step size) is 2seconds. 2 seconds is added to 5, which is 7; 2 seconds is added to 7, which is 9, and so on until the calculation is completed four times and we have five numbers:

```
t(1) = 5 ;
dt = 2 ;
for i = 1:4
   t(i+1) = t(i) + dt ;
end
>> t

t =

     5     7     9    11    13
```

Notice that the value of `t` is now not just a single number, it is a vector of five numbers. Indexing `t(1)` would still return 5, but indexing `t(3)` now returns 9, because it is the number in the third column in the vector.

4.3 Gravity

The only gravitational equations used are from derived from Newton's law of universal gravitation. To these equations, the Earth is defined as a massive perfect sphere or point, located at the center of Earth-Centric-Inertial Cartesian coordinates.

4.3.1 Derivation

Newton's law of universal gravitation for an Earth-satellite setting is defined as:

$$F_G = G\frac{Mm_{sat}}{r^2}$$

- F_G is the gravitational force between the Earth and the satellite.
- G is the gravitational constant.
- M is the mass of the Earth.
- m_{sat} is the mass of the satellite.
- r is the distance between the Earth and the satellite.

We need to determine how the satellite will move about the Earth; knowing its position at each step in time would do the trick. In order to know the position at each step, we need to know the velocity at each step, so we can see how much and in which direction the position changed. Also, since we know that we are dealing with gravity, we know that there are no constant speeds; acceleration MUST be involved as degreed by the laws of Newtonian classical physics. Therefore, we need to know how much and in what direction the velocity has changed at each step. Stated otherwise, we need an equation that defines the acceleration (in this case, the gravitational acceleration) at every step.

Since $F_G = m_{sat}a_{sat}$, the above form of the gravity equation can be rewritten as a vector equation involving the gravitational acceleration on the satellite:

$$a_{sat} = -\frac{GM}{r^2}\,\hat{r}$$

- a_{sat} is the gravitational acceleration on the satellite.
- $\hat{r}$ is the unit vector from the Earth to the satellite.

In this way, we can create a second-order differential equation that can be solved numerically.

4.3.2 Numerical Solving the Gravity Equations

```
% Gc = Gravitational Constant, Me = Mass of Earth, dt = Step Size,...
% r = Radius from Earth Center, ag = Satellite Gravitational Acceleration}
```

After defining our gravity in three dimensions and splitting our acceleration equations into six first-order differential equations, we have three equations of velocity and three of position. These equations are ready to be solved numerically. The numerical process can be defined within MATLAB using the Midpoint method set up inside a for loop:

```
for i=1:steps
t(i+1)=t(i)+dt;

ag = Gc*Me/(r^2);

xmid=x(i)+dt/2*vx;
ymid=y(i)+dt/2*vy;
zmid=z(i)+dt/2*vz;

rmid = sqrt(xmid^2+ymid^2+zmid^2);

vxmid=vx-dt/2*x(i)/r*ag;
vymid=vy-dt/2*y(i)/r*ag;
vzmid=vz-dt/2*z(i)/r*ag;

agmid = Gc*Me/(rmid^2);

x(i+1)=x(i)+dt*vxmid;
y(i+1)=y(i)+dt*vymid;
z(i+1)=z(i)+dt*vzmid;

r = sqrt(x(i+1)^2+y(i+1)^2+z(i+1)^2);

vx=vx-dt*xmid/rmid*agmid;
vy=vy-dt*ymid/rmid*agmid;
vz=vz-dt*zmid/rmid*agmid;

end
```

4.4 Drag

The most important part of the FIREBIRD analysis centered around how the presence of atmospheric drag would affect the two satellites. In this case, by increasing the mass of one satellite by some amount, it would be less susceptible to atmospheric drag forces, causing it to lose some momentum and prolong the effective lifespan of FIREBIRD.

4.4.1 The Drag Equation

The following equation gives the atmospheric drag from a high-velocity situation in terms of force; notice the drag force is proportional to the square of the velocity:

$$F_D = -\tfrac{1}{2}\rho v^2 C_d A \hat{v}$$

- F_D is the drag force.
- rho is the atmospheric density.
- v is the velocity of the object through the atmosphere.
- C_D is the dimensionless drag coefficient.
- A is the object cross-sectional area.
- $\hat{v}$is a unit vector in the direction of object velocity.

4.4.2 The Drag Coefficient

The drag coefficient is a number that is dependent upon a wide variety of factors, such as speed, shape, surface area, surface texture, and atmospheric density. It is usually and empirically found value from experiments in wind-tunnels. As of 2009, SSEL has not performed any empirical testing of the drag forces involved with the flight of a cubesat satellite. However, based upon research from other cubesat projects that have cited SMAD, I assume a coefficient that is roughly 4 will suffice for this simulation.

An accurate drag coefficient should be found for the greatest accuracy during essential simulations; an incorrect or incomplete description of the drag coefficients can make apparent required masses and separating velocities to be incorrect. Before important structural decisions are made, the appropriate drag coefficients must be found.

4.4.3 The Ballistic Equation

The ballistics of an object determines how susceptible it is to drag. The smaller the ballistic coefficient, the greater the drag forces on an object in motion through the atmosphere.

The ballistic coefficient can be related to the drag coefficient through mass and cross-sectional area:

$$BC = \frac{m_{sat}}{C_D \times A}$$

- BC is the Ballistic coefficient.
- M is the object mass.

We can quickly implement the above equation and find the ballistic coefficient through our existing m_{sat} and A:

$$BC = \frac{m_{sat}}{C_D \times A} = \frac{1.5}{4 \times 0.015} = 25$$

The ballistic coefficient is 25 kilograms per cubic meter, which is restated by other sources who cited SMAD. This furthers the strength of my assumption that a drag coefficient of 4 should be used for the drag equation, for the purposes of this simulation.

4.4.4 Creating Drag

In order to find out how drag will affect the position of the satellites at each step, we once again need to characterize the drag in terms of acceleration, just like we did with Newton's gravity.

The drag equation $F_D = -\frac{1}{2}\rho v^2 C_d A \hat{v}$ and $F_D = m_{sat} a_{sat}$ can be rewritten as:

$$a_{sat} = \frac{-\frac{1}{2}\rho v^2 C_d A \hat{v}}{m_{sat}}$$

This can be incorporated with the original equation involving acceleration due to gravity, because now we have a new source acceleration to simulate.

```
% ad = Drag Acceleration, rho = Atmospheric Density, v = Satellite Velocity Magnitude,...
% cd = Drag Coefficient, m2 = Satellite Mass, csa = Cross-Sectional Area,...
% height = Altitude above Earth-like Sphere, EER = Effective Spherical Earth Radius


for i = 1:steps
    t(i+1)=t(i)+dt;

    ag = Gc*Me/(r^2);

    ad = (0.5*rho*v^2*cd*csa)/(m2);

    xmid=x(i)+dt/2*vx;
    ymid=y(i)+dt/2*vy;
    zmid=z(i)+dt/2*vz;

    rmid = sqrt(xmid^2+ymid^2+zmid^2);

    vxmid=vx-dt/2*(x(i)/r*(ag)+vx/v*(ad));
    vymid=vy-dt/2*(y(i)/r*(ag)+vy/v*(ad));
    vzmid=vz-dt/2*(z(i)/r*(ag)+vz/v*(ad));

    vmid = sqrt(vxmid^2+vymid^2+vzmid^2);

    agmid = Gc*Me/(rmid^2);

    admid = (0.5*rho*vmid^2*cd*csa)/(m2);

    x(i+1)=x(i)+dt*vxmid;
    y(i+1)=y(i)+dt*vymid;
    z(i+1)=z(i)+dt*vzmid;

    r = sqrt(x(i+1)^2+y(i+1)^2+z(i+1)^2);

    height = r - EER ;

    rho = dragON*(10^(-1.545663328e-28*height^5 + 4.745120117e-22*...
        height^4 - 5.562944567e-16*height^3 + 3.139917515e-10*...
        height^2 - 9.287675901e-05*height - 1.658663453e+00)) ;

    vx=vx-dt*(xmid/rmid*(agmid)+vxmid/vmid*(admid));
    vy=vy-dt*(ymid/rmid*(agmid)+vymid/vmid*(admid));
    vz=vz-dt*(zmid/rmid*(agmid)+vzmid/vmid*(admid));

    if height < 0 ;
        break
    end
end
```

5 Future Processes

5.1 Epoch

Assigning a reference date/frame such as J2000, would probably help improve accuracy and ease the interactions between the program and the NASA models.

5.2 Advanced Gravitational Equations - Spherical Harmonics, Relativity

5.2.1 Better Gravitational Model

To more accurately define how the Earth's gravity affects other objects, its important to realize that the Earth is not a perfect sphere. Implementing a few terms of the Earth's spherical harmonics can help accurately define the gravity due to the lumpy placement of mass. The first harmonic term, J2, is commonly used. This defines the earth as an ellipse, not a sphere, making the gravitational equations geodetic, not spherical or point-based. The gravitational harmonics cause satellite orbits to lose their perfect elliptical paths, as defined by old Newtonian and Keplerian gravity. Instead, precession appears in the orbits over time.

Some orbital simulators use more spherical harmonic terms, up to J4, J6, or even J70. Each term in the equation greatly increases the computational time required, but adds much more complexity and accuracy to the equations that define our Earth's gravity.

Other options for increasing gravitational accuracy exist. One could add the gravity of the moon and the Sun, and modify the equations to accommodate relativity. Another approach would involve the use of the NASA Earth Gravitational Model 2008 (EGM2008), which provides an extremely accurate model of the Earth's gravity. The model utilizes over 2000 spherical harmonic terms, and would be almost impossible to process on personal computers due to the sheer processing time required.

5.2.2 Spherical Harmonics, J2 Term

For Newton's equations utilizing the J2 spherical harmonic term, non-drag differential equations set up for the midpoint method could be easily implemented within a simple program. However, some errors were present during my program simulations with J2 propagation, and it is unclear whether they were due to typos from my sources or from numerical error. At any rate, I scrapped the idea of incorporating J2 harmonics in my program, settling for the assumption that atmospheric drag would be the main source of satellite perturbation.

The J2 term is equal to 1.08263e-3.

Within a `for` loop, I had Newton's equations utilizing J2 propagation and no drag take this form:

```
% EqR = Equatorial Radius / Semi-major Axis, J2 = J2 Spherical Harmonic Constant


    t(i+1)=t(i)+dt;

    agx = Gc*Me/(r^2)*(1+(3/2)*J2*(EqR/r)^2*(1-5*(z(i)/r)^2));
    agy = Gc*Me/(r^2)*(1+(3/2)*J2*(EqR/r)^2*(1-5*(z(i)/r)^2));
    agz = Gc*Me/(r^2)*(1+(3/2)*J2*(EqR/r)^2*(1-5*(z(i)/r)^2));

    xmid=x(i)+dt/2*vx;
    ymid=y(i)+dt/2*vy;
    zmid=z(i)+dt/2*vz;

    rmid = sqrt(xmid^2+ymid^2+zmid^2);

    vxmid=vx-dt/2*x(i)/r*agx;
    vymid=vy-dt/2*y(i)/r*agy;
    vzmid=vz-dt/2*z(i)/r*agz;

    agmidx = Gc*Me/(rmid^2)*(1+(3/2)*J2*(EqR/rmid)^2*(1-5*(zmid/rmid)^2));
    agmidy = Gc*Me/(rmid^2)*(1+(3/2)*J2*(EqR/rmid)^2*(1-5*(zmid/rmid)^2));
    agmidz = Gc*Me/(rmid^2)*(1+(3/2)*J2*(EqR/rmid)^2*(1-5*(zmid/rmid)^2));
```

(Code continued on next page.)

```
x(i+1)=x(i)+dt*vxmid;
y(i+1)=y(i)+dt*vymid;
z(i+1)=z(i)+dt*vzmid;

r = sqrt(x(i+1)^2+y(i+1)^2+z(i+1)^2);

vx=vx-dt*xmid/rmid*agmidx;
vy=vy-dt*ymid/rmid*agmidy;
vz=vz-dt*zmid/rmid*agmidz;
```

This code seemed to work well for some conditions (at an inclination of 63.4 degrees, the satellite did *not* precess, identical to the behaviors of similar Molniya orbits) but for other conditions, large numerical errors began to appear. With a slight modification (probably stemming from a source typo) I was able to remove the numerical error, but the satellite began precessing wildly at an inclination of 63.4 degrees. Furthermore, it was precessing incorrectly for an orbit. This leads me to believe that either I had an incorrect equation for gravitational acceleration with J2 perturbation, or the step size of the simulation would need to be drastically reduced. It was for this reason that I did away with J2 spherical harmonics in this simulation.

5.3 NASA Atmospheric Modeling

5.3.1 NRLMSISE-00

Like gravity, the atmosphere of the Earth can always be modeled more accurately. However, atmospheric density is notoriously difficult to model, being dependent upon things like the Solar cycle, temperature, and humidity, not to mention the location on the Earth's surface. The NRLMSISE-00 Atmospheric Model is a complex empirical atmospheric model from NASA that models these and many more things about the atmosphere, from ground to space for a total range of 1000 kilometers from the surface.

5.3.2 Preparations

In order to use NRLMSISE-00, one must understand about converting between coordinate systems and frames of reference. The gravitational and drag physics equations above all take place in a particular reference frame and coordinate system: Earth-Centric-Inertial Cartesian coordinates. Since this is within an inertial reference frame, Newtonian laws of physics are valid because the axises are *not* spinning with the Earth as the Earth rotates around its axis. The axises are centered on the earth, but they stay continually pointed at a very far-away object, such as a distant star.

Since the Earth rotates, and the atmospheric model is dependent on the time of day sun cycles, one would need to convert to a reference frame that spins *with* the Earth. That is, one could potentially use Cartesian coordinates which x-axis would always be pointed out of the Prime Meridian. This is called Earth-Centric-Earth-Fixed Cartesian coordinates. Deriving an equations that effectively "rotates" the Earth around its equator and the ECEF coordinate system along with it is fairly trivial, by simply applying a transformation that oscillates the magnitudes of "x" and "y" as the ECEF coordinate system rotates through time, as defined by "one day = sidereal time":

```
xecef = ((xeci^2+yeci^2)^0.5)*cos((2*pi*time/sdt)-atan2(yeci,xeci)) ;
yecef = -((xeci^2+yeci^2)^0.5)*sin((2*pi*time/sdt)-atan2(yeci,xeci)) ;
zecef = zeci ;
```

Once the conversion from ECI to ECEF has been made, it is time to convert from ECEF Cartesian to Geodetic coordinates, which by nature is in a ECEF frame of reference. Since geodetic coordinates assume that the Earth is elliptical, slightly complex transforms must be made during conversion. There are functions within

MATLAB that do this automatically, but their speed can be improved when needed by going directly into the m file and hard-coding the necessary conversion routines within the orbit propagation program itself. Optimizing the conversion code could mean the difference between a few minutes of processing and a few hours. Such an example of part of an optimized version I modified is demonstrated here: (`fltn` = Earth flattening)

```
Longitude = atan2(yECEF,xECEF);
rho = hypot(xECEF,yECEF);
beta = atan2(zECEF,(1-fltn)*rho);
Latitude = atan2(zECEF+PoR*E2c2*sin(beta).^3,rho-EqR*Ecc2*cos(beta).^3);
betaNew = atan2((1-fltn)*sin(Latitude),cos(Latitude));

cnt = 0;
while any(beta(:) ≠ betaNew(:)) && cnt < 5
    beta = betaNew;
    Latitude = atan2(zECEF+PoR*E2c2*sin(beta).^3,rho-EqR*Ecc2*cos(beta).^3);
    betaNew = atan2((1-fltn)*sin(Latitude),cos(Latitude));
    cnt = cnt + 1;
end

sinphi = sin(Latitude);
N = EqR./sqrt(1-Ecc2*sinphi.^2);
Altitude = rho.*cos(Latitude)+(zECEF+Ecc2*N.*sinphi).*sinphi-N;
```

There are numerical instabilities associated with the conversion, therefore there is a loop within a loop in this example (attempting to converge on a stable latitude). Removal of this loop results in much faster processing time, but very significant errors at some extreme locations (at or near x, y, or z axises in Cartesian Coordinates).

When the longitude, latitude, and altitude are found, they then can be inputted into NRLMSISE-00; this, along with the inputted time, will return very accurate atmospheric densities for the drag equations.

5.4 Geomagnetic Modeling and Benefits

FIREBIRD has passive attitude control. This means that it contains a bar magnet inside which automatically aligns itself to the Earth's magnetic field. Instead of a satellite tumbling though space during flight, it maintains a very particular orientation depending upon its location in the Earth's magnetic field.

5.4.1 Cross-sectional Area

One of the main components of the drag equation has to do with cross-sectional area. For the purposes of the simulation, I decided to have both cubesats display a 0.015 square meter cross-sectional area during flight. That means that the atmosphere it was flying through would displace an area of 0.015 square meters as it progressed.

Since we know that FIREBIRD has passive attitude control, this is known to be a simplification: the cross-sectional area of a satellite will change drastically as it orbits the Earth, which in turn influences the acceleration due to drag. For instance, a 1.5u satellite can rotate with respect to the direction of the atmosphere while it travels around the Earth. Let's say you have powerful wind blowing directly at your back, and you are viewing the 1.5u cubesat head on with the long side pointed at you. You would see a 10 cm by 10 cm square. As the cubesat rotates, the square would turn into a longer rectangle; the sides of the cubesat previously perpendicular and parallel to you and the wind would now at approximately 35-55 degree angles.

This larger rectangle you would see would be the largest "cross-sectional area" possible, and would present the most area for wind / atmosphere to come into contact to; this means more drag. It is akin to feeling more drag when you slice through the air with the flat of a long blade than when you effortlessly slice normally with the sharp thin part.

Two equations, depending upon the orientation of the bar magnet within FIREBIRD, can describe the surface area of the cube given an orientation angle:

```
% wid = width of cubesat (short dimension), len = length of cubesat (long dimension),...
% rot = angle of rotation


% Axis of rotation through long dimension
   csa = -0.5*wid*len*(cos((2*pi*rot))*(2^(.5)-1)-2^(0.5)-1) ;
% Axis of rotation through short dimension, two local minimum areas
   csa = -.25*wid*(cos(2*pi*rot)*(2*((wid^2+len^2)^0.5)-wid-len)+2*cos(pi*rot)*(wid-len)-2*...
        ((wid^2+len^2)^0.5)-wid-len) ;
```

It is possible that both of these equations can be combined to describe the surface area through rotations in all three dimensions. This would probably be needed, since the angle of rotation and the angle with respect to the direction of motion would probably be quite different most of the time.

5.4.2 Incorporating Magnetism into Changing Area

The IGRF Geomagnetic Model is similar to the atmospheric model and the gravitational model. Depending upon the configuration, it would take inputs from either ECEF Cartesian coordinates, or Geodetic coordinates. The outputs of the model would be accurate magnetic force vectors (magnitude and direction) at many points in space around the Earth. The direction of these vectors would determine how the magnet within the cubesat would be oriented at any point in space, and the angle of orientation with respect the direction of motion through the atmosphere would give the true cross-sectional area for the drag equation.

5.4.3 Drag Coefficient and Area

The drag coefficient can also be linked with the changing cross-sectional area, and thus indirectly with magnetism; the coefficient changes with the attitude and cross sectional area of an object. These equations can utilize maximum and minimum values (if they can be determined) for the coefficient of drag given the corresponding orientation:

```
% Starts at min and goes to max with half a period, and returns to min with a full period
   cd = -((cdmax-cdmin)*cos(2*rot*pi)-cdmax-cdmin)/2 ;
% Starts at min1 and goes to max with half a period, then goes to min2 with a full period,
% then up to max with 1.5 periods, then back to min1 with 2 periods
   cd = -((2*sqrt(cdmin1^2+cdmin2^2)-cdmin1-cdmin2)*cos(2*rot*pi)-2*...
        (cdmin1-cdmin2)*cos(rot*pi)-2*sqrt(cdmin1^2+cdmin2^2)-cdmin1-cdmin2)/4;
```

5.5 Non-Empirical Drag Coefficient Determination

Drag coefficients are guesses at an unknown value of which has huge uncertainties at high altitudes. There are ways to test for it empirically with objects in wind tunnels, but the uncertainty can still be very high.

Nevertheless, there exist some programs, in use and development, which can mathematically analyze an object's drag coefficient in a variety environments by analyzing simulations of atmospheric making contact with an object and studying the energy losses associated with it. The results are surprisingly accurate for normal conditions, and uncertainties can be made known through the normal program processes.

The ANSYS interface with SolidWorks can also solve for drag coefficients.