PROBLEM SET #4

TO: PROF. DAVID MILLER, PROF. JOHN KEESEE, AND MS. MARILYN GOOD

FROM: MATTHEW RICHARDS AND ANNA SILBOVITZ

SUBJECT: PROBLEM SET #4 (ORBITS, POWER, AND COMMUNICATION)

DATE: 10/28/2003

MOTIVATION

Ground communication is often an essential part of a satellite's function. At certain times, the satellite may need to transmit data to a certain location on the Earth. The data is sent at a specified rate, and the power needed by the transmitter is a function of this rate. The power system of the satellite must be able to provide power for the transmitter, as well as for other satellite functions. The orbit of the satellite impacts both of these systems.

PROBLEM STATEMENT

For a low-earth satellite, find the power system needed if the satellite must send data to the ground once a day. Given the ground site location, the altitude above the ground site that the satellite should pass, general power requirements (not including power for the transmission antenna), mission duration, and the data rate, find the orbit to ensure a reasonable pass over the ground site at least once a day, the solar array size and mass, types and mass of batteries, and the amount of data that can be transmitted during a typical downlink.

APPROACH

A tool using MATLAB and STK will evaluate the three sub-systems. The user will input a ground site location, the altitude the satellite should be at when at perigee, the data rate, power requirements, the mission duration, and the mission start date. The altitude of perigee will be constrained so that it will be no greater than 3000 km. To find the orbit, the tool will make the satellite be at perigee when it passes over the ground site. The orbit will be a repeating ground track, repeating once every sidereal day. The tool will determine the necessary semi-major axis, eccentricity, inclination, and period of the orbit to fit these constraints. STK will determine when the satellite has line-of-site access to the ground site, and the pass each day that is most directly overhead will be the pass used for data transfer. Since we're using a repeating ground track, we can get a pass that is nearly overhead each (sidereal) day. Using the average value for length of pass (averaging time of pass each day, over a reasonable number of days) and the data rate, the amount of data sent each transmission is determined. Using table 13-5 in *SMAD* with a fixed ground station antenna size, the tool will next determine how much power is required of the satellite transmitter. By using this in combination with the general power requirements input by the user, the tool will determine solar arrays (table 11-34 in *SMAD*) and batteries (table 11-40 in *SMAD*) needed. Eclipse times needed for these calculations can also be found using STK.

By varying the input parameters, an analysis can be done showing system trade-offs. The three trade-offs involve the orbital parameters, the amount of data that can be sent each transmission, and the size and mass of the power system. Changing altitude will affect the power system, as well as the amount of data that can be transmitted each day. Changing the data rate will affect data transmission as well. If the satellite must transmit a certain amount of data each day, the tool can be used to determine how that affects the orbit and the power system. Certain combinations of orbits and data rates may be desirable so that a needed amount of data is transmitted each day, but an unrealistic power system may result; this tool will identify these types of issues.

SOLUTION

A MATLAB program has been designed to solve the proposed problem. The tool flow is as follows:

1. The orbit is found so that the satellite passes through perigee over the ground site once each sidereal day.
2. STK is opened, and a scenario is set up with the ground site and satellite. STK returns data indicating how long each pass used for ground communication is, and how long the satellite is in eclipse each orbit.
3. The amount of data that can be transmitted to the ground each pass is determined, as well as the power needed for each transmission.
4. The solar arrays are designed and a battery is chosen

Each of these parts of the tool will be described in detail in this section.

Orbit Model

The orbit is determined using the ground site latitude and longitude, the altitude of the satellite at perigee, and the time the satellite is first at perigee (see MATLAB function *find_orbit* in Appendix B). The following assumptions and decisions are made:

1. The earth is circular with radius = 6378.137 km
2. Site elevation is not significant enough to use
3. A nearly circular orbit is desirable, so eccentricity will be minimized
4. Inclination will be set to 5 degrees above the value of the site latitude; for site latitude over 85 degrees, it will be set to 90

First the radius of perigee r_p is found by adding the radius of the earth to the altitude of perigee. Next, the initial eccentricity e_i is set to 0.01, for a nearly circular orbit. Using eccentricity and the radius of perigee, an initial value for the orbit's semi-major axis a_i and period p_i can be found (μ is the earth's gravitational constant in the correct units):

$$a_i = r_p/(1.0 - e_i) \qquad \text{[km]}$$
$$p_i = 2\pi * \text{sqrt}(a_i^3/\mu) \qquad \text{[minutes]}$$

However, the orbit must be a repeating ground track, which requires that the period p be equal to an integer number of sidereal days divided by an integer number of revolutions. The number of revolutions using the initial period is found:

$$n_{revs} = (1436.068167 \text{ minutes})/p_i$$

The number of revolutions is rounded down to an integer, and the final period, semi-major axis, and eccentricity is found. The eccentricity will always be larger than 0.01, but it will be small.

For the satellite to pass over the ground site, inclination i can be in the range:

$$site\_latitude <= i <= 180 \text{ deg} - site\_latitude$$

As mentioned, inclination is arbitrarily chosen to be *site_latitude* + 5 degrees.

The argument of perigee ω and longitude of ascending node Ω are found so that the satellite will be at perigee over the site at the desired starting time. Argument of perigee is found by the equation:

$$\omega = \mathrm{asin}(\sin(\mathit{site\_latitude})/\sin(i)) \qquad [\mathrm{deg}]$$

Longitude of ascending node is found by the following series of equations:

$$\mathit{temp1} = \mathrm{asin}(\cos(i)/\cos(\mathit{site\_latitude})) \qquad [\mathrm{deg}]$$
$$\mathit{temp2} = \mathrm{acos}(\cos(\mathit{temp1})/\sin(i)) \qquad [\mathrm{deg}]$$
$$\Omega = \mathit{gst} + \mathit{site\_longitude} - \mathit{temp2}; \qquad [\mathrm{deg}]$$

Where *gst* is Greenwich sidereal time, found from the starting time (see MATLAB function *find_gst* in Appendix B).

STK

The orbit found is entered into STK by connecting through MATLAB and setting up a new scenario (see MATLAB function *run_stk* in Appendix B). The ground site is also added to the STK scenario as a facility. STK can return reports detailing scenario behavior for a given time interval. STK is used in this tool to return a report indicating the length of time the satellite is in eclipse each orbit, and to find all of the time periods the satellite has line-of-sight access to the ground site. Ideally, the STK scenario should be run for the entire lifetime of the satellite, and averages taken over this time period. However, connecting to STK through MATLAB is slow, and STK is slow in doing calculations needed for the reports. Averaging over a 5 or even 1 year life time would make each individual run very slow, and repeated runs for trending would not have been possible. To solve this problem, averages are taken over a much smaller period. While it is acknowledged that this will not provide the most accurate answer, particularly with eclipse times since those may vary over the course of a year, it was decided that this method would be sufficient for the scope of this project.

The scenario is set to run for 12 days. For ground access times, the first and last days are eliminated, since the passes on each of those days will be partial (since the satellite is at perigee at the beginning of the scenario). Note that a sidereal day is nearly 4 minutes shorter than a calendar day. This will cause the pass over the ground site to occur approximately 4 minutes earlier each day.

STK reports the eclipse time for each orbit over the 12 days. The first and last eclipse times are thrown out since they may reflect partial eclipses. The remaining values are used and an average is taken.

Telecommunications Model

Next the power needed for each transmission can be found, as well as the amount of data that can be transmitted each pass (see MATLAB function *transmitter* in Appendix B). The following assumptions were made:

1. The ground station's antenna has a diameter of 5 meters
2. Each pass that should result in a transmission will be successful
3. The time needed to initialize a transmission is two minutes

The power needed was found from figure 13-5 in *SMAD*. Since the tool is only designed for low-earth orbits, the line indicating 'Earth coverage case' with a 5 meter ground antenna was used. From visual examination of this graph, the power needed is approximately equal to 10 times the data rate in Mbps:

$$P_{trans} = 10*(data\_rate \text{ [Mbps]}) \qquad \text{[W]}$$

For finding total data sent each pass, equation X.X from *SMAD* was used:

$$Data\_Sent = data\_rate*(F*T_{opass} - T_{init})/M \qquad \text{[bits]}$$

M represents how often transmissions are successful. Because of assumption (2) above, and because the amount of data sent on each successful transmission is what we want to calculate, *M* can be set to 1. T_{init} is the time needed to initialize each transmission, and has been set to 2 minutes, which is a typical value according to *SMAD*. T_{opass} is the time a pass directly over the ground site has line-of-sight access, and *F* alters this to account for passes that are not directly overhead. Since this tool is only using passes that are directly overhead, and STK calculates the line-of-sight time taking any slight variation into account, the pass time found by the STK module (T_{pass}) replaces this term. So the equation reduces to:

$$Data\_Sent = data\_rate*(T_{pass} - T_{init}) \qquad \text{[bits]}$$

Power Model

The power model consists of two MATLAB functions: one to select a power production architecture and one to select a power storage architecture. We chose to design a solar array-battery power subsystem for our earth-orbiting spacecraft. We did not consider radioisotope, nuclear reactor, or fuel cell for power sources as these are traditionally reserved for interplanetary missions.

The power production module outputs the mass of the solar array, the area the solar array, and the solar cell type (either Silicon, Thin Sheet Amorphous Si, Gallium Arsenide, Indium Phosphide, or Multijunction GaInP/GaAs) that will minimize the solar array area. (See MATLAB function *Power_Optimizing* in Appendix B.) The inputs to the module are mission life, power required (not including data transmission needs), power required for data transmission needs, time to downlink data, average time of eclipse, average time of sunlight, and inclination of the orbit.

Many assumptions were made in coding the power production module. Energy transfer efficiencies (X_e =.65 during eclipse and X_d =.85 during sunlight) were taken as constants from *SMAD*. The input power required for data transmission was assumed to include energy transfer efficiency losses (X_{dl} =1). Also, the non-transmitting power required was assumed to be constant in sunlight and eclipse (P_e=P_d). Calculating a dynamic value of inherent degradation—entailing analysis of the solar cell design and assembly process, temperature of array, and shadowing of cells— falls outside of the scope of our problem so a constant value of inherent degradation (.77) was taken from *SMAD*.

The first step in the power production module is to compute the power the solar array is required to collect during sunlight, P_{sa}. T_e, T_d, and T_{dl} are time in eclipse, time in sunlight, and time to downlink respectively.

$$P_{sa} = \frac{\left(\frac{P_e T_e}{X_e} + \frac{P_d T_d}{X_d} + \frac{P_{dl} T_{dl}}{X_{dl}}\right)}{T_d}$$

The second step is to compute beginning-of-life power, P_{bol}, per unit area. The solar illumination intensity, S_i, is reduced by solar cell efficiency, degradation, and angle-of-incidence factors. Each solar cell efficiency is fed into the equation using array X_{ss}. The orbit's inclination, *inc*, is taken as an input. The worst-case sun angle between equatorial and ecliptic planes is taken as 23.5 degrees.

$$P_{bol} = S_i \cdot X_{ss} \cdot I_d \cdot \cos\left(\frac{23.5\pi}{180} + inc\right)$$

The third step is to compute end-of-life power collection capability, P_{eol}, of the solar array. For the duration of the mission each solar cell experiences life degradation, D_{ss}, due to thermal cycling in and out of eclipses and other factors. Mission life, L_m, is an obvious input.

$$P_{eol} = P_{bol}(1-D_{ss})^{L_m}$$

The fourth and final step of the power production module is to determine the solar array area, A_{sa}, that meets the power collection requirements, P_{sa}. We must design for the end-of-life power collection capability, P_{eol}.

$$A_{sa} = \frac{P_{sa}}{P_{eol}}$$

A_{sa} and P_{eol} vary for each solar cell variety. The module selects a solar cell type that minimizes array area and proceeds to compute mass of the power subsystem using equation 10-13 in *SMAD*.

The power storage module outputs the optimal secondary battery type (choosing among Nickel-Cadium and three Nickel-Hydrogen varieties) as a function of mass. (See MATLAB function *Storage_Optimizing* in Appendix B.) Inputs to the function include the mission lifetime, number of times the satellite goes in and out of eclipse, power required (not including data transmission needs), power required for data transmission needs, time to downlink data, and average time of eclipse.

One assumption made in the module is that transmission efficiency between the battery and the load, n, is 90%. Also, in our module we are designing to meet a battery capacity requirement. Whether or not to have additional batteries to provide redundancy is a trade which may become the subject of future work.

The depth-of-discharge (DOD), or battery capacity removed during a discharge period such as eclipse, is the first calculation performed in the power storage module. The DOD is inversely related to the number of charge and discharge cycles undergone by a battery throughout its life. The number of cycles is determined by our spacecraft's orbital parameters (i.e. altitude). Figure 11-11 from *SMAD* is used in our function to determine DOD for our Nickel-Cadmium and Nickel Hydrogen battery types. To simply the calculation, cycle life is broken into seven groups for each secondary battery (less than 1000 cycles, between 1000 and 2000 cycles, etc.). The DOD for a given range of values is taken as the average of the maximum DOD and minimum DOD of that range.

The second step of the power storage module determines power required during eclipse, P_e. Designing for the batteries for the worst-case scenario, we assume the data transmission down to Earth occurs during eclipse.

$$P_e = \frac{\left(P_{av}T_e + P_{dl}T_{dl}\right)}{T_e}$$

The third step determines the battery capacity, C_r, required for the spacecraft. Depth-of-discharge, X_{dod}, is input to this calculation as an array representing the four battery types.

$$C_r = \frac{P_e T_e}{60 n X_{dod}}$$

The fourth and final step of the function selects among the battery types to minimize mass, M. Specific energy density, X_{ed}, is fed into this function as an array.

$$M = \frac{C_r}{X_{ed}}$$

USERS GUIDE

This tool has been tested to work with STK 4.3, and MATLAB should have already been configured to run with STK.

1. Place the following files in the MATLAB working directory:
   ground_comm.m
   find_gst.m
   find_orbit.m
   run_stk.m
   transmitter.m
   Power_Optimizing.m
   Storage_Optimizing.m
2. Open STK and MATLAB. Type 'stkinit' into the MATLAB prompt.
3. Run the program ground_comm by entering the following:

   [data_sent, orbit, M_sa, A_sa, M_bat] = ground_comm(site_lat, site_lon, pass_alt, dr, pow, start_date, life)

   Where the inputs are:
   site_lat: site latitude (degrees)
   site_lon: site longitude (degrees)
   pass_alt: altitude of perigee pass (km)
   dr: rate data at which can be transmitted (bps)
   pow: average power needs for the satellite, not including for data transmission (W)
   start_date: time at first perigee passage. This is an array in the format [year, month, day, hour, min], where month is a number 1 through 12
   life: mission duration (years)

   The outputs will be:
   data_sent: amount of data transmitted each pass (bits)
   orbit: orbital parameters, in the format [semi-major axis (km), eccentricity, inclination (deg), argument of perigee (deg), longitude of ascending node (deg)]
   M_sa: mass of the solar array (kg)

A_sa: area of the solar array (m^2)
M_bat: mass of the battery

Note that while the program runs, its progress and the outputs will be printed onto the MATLAB screen. Additionally, the solar cell type and battery type chosen will be printed to the screen.

4. Optionally, after the run the scenario will still be set up in STK. The orbit can be examined, and more reports can be created through the GUI if desired.

EVALUATION OF TOOL

Test Run

To test our tool, we chose to define a LEO spacecraft with a five year lifespan launched from the Eastern Range. The data rate for transmission is 10K bits per second.

Inputs	
Site Latitude (degrees)	28
Site Longitude (degrees)	280
Pass Altitude (km)	300
Data Rate (bps)	10,000
Non-Transmitting Power (W)	110
Start Date	26-Oct-03 12:00.00
Mission Length (years)	5

Outputs	
Average Data Sent (bits/transmission)	4.34E+06
Semi-Major Axis (km)	6932
Eccentricity	0.0367
Inclination (degrees)	33
Argument of Perigee (degrees)	59.54
Longitude of Ascending Node (degrees)	79.49
Solar Cell Type	Multijunction GaInP/GaAs
Mass of Solar Array (kg)	8.76
Area of Solar Array (m^2)	2.08
Secondary Battery Type	Nickel-Hydrogen (single pressure vessel)
Mass of Battery (kg)	2.96

The orbit is shown to be correct by examining the orbit in STK. Figure 1 shows the satellite and its orbit at the start time. The satellite is directly over the ground site, as desired. Figure 2 shows the satellite approximately one sidereal day later, and again the satellite is directly over the ground site. Further examination of the orbit by looking at the GUI and having STK generate reports verifies that the satellite is at perigee, and at the desired altitude at these times.

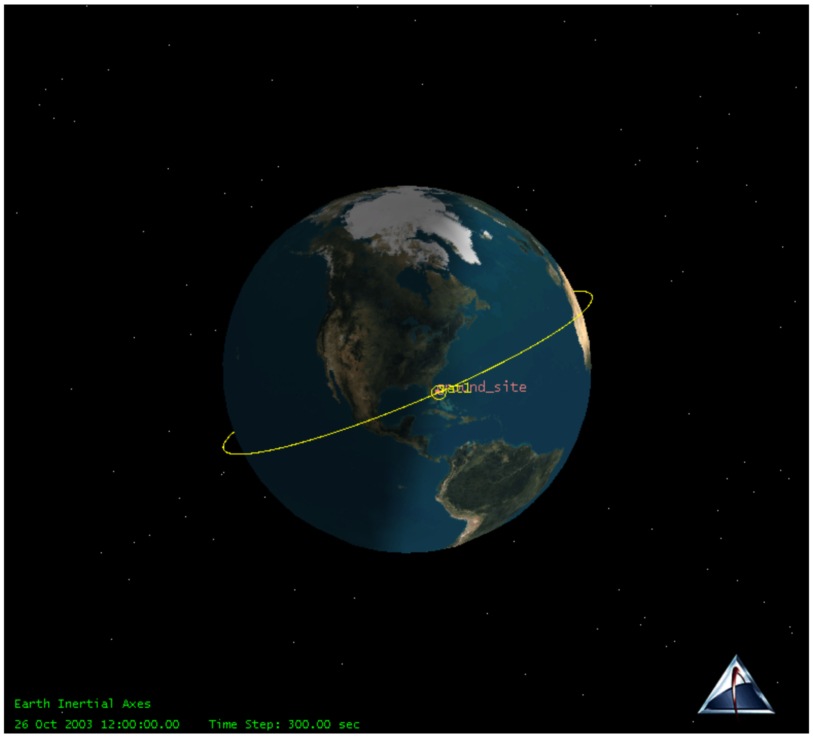

Figure 1. Satellite at start time (over ground site)

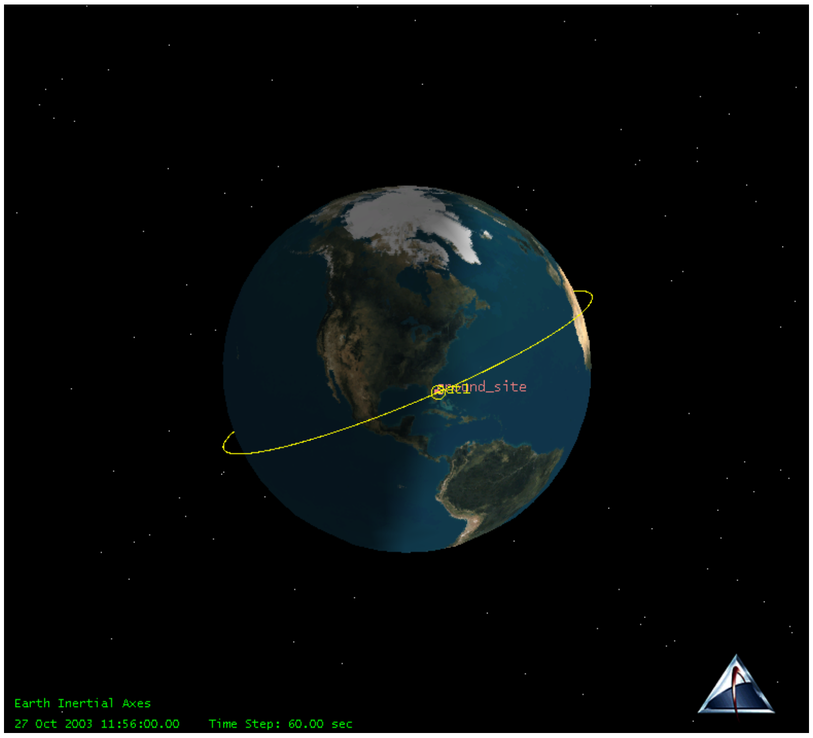

Figure 2. Satellite over ground site one sidereal day after start time

The power system is shown to be reasonable by comparison with the FireSat example in *SMAD*. The FireSat needs an average 110 W of power. The area of the solar array is estimated to be 2 m^2, the mass of the solar array is 9.6 kg. In our test run, we also input 110 W as average power. This value is then adjusted to account for power needs of the transmitter, when the data rate is 10 kbps. However, from table 13-5 in *SMAD* (that was used to find power of the transmitter, as described above), a data rate of 10 kbps only requires 0.1 W of power. So it does not significantly affect the power system, and the solar array outputs of our test run above are similar to those found in *SMAD* for FireSat. For the battery, only capacitance was calculated in the FireSat example, not mass. This value is 119 W-hr. But if we then choose the Nickel-Hydrogen (single-pressure vessel) that our tool chose in this situation, we can find the mass. The specific energy density of this battery is 57 W-hr/kg, so the mass of this battery for the FireSat would be 2.09 kg. So the value found in our test run for battery mass is also shown to be reasonable. Differences between our test run and the FireSat example occur because of the slight difference in power needs due to the transmitter, and also from a difference in orbit. Both orbits are LEO orbits, but they are not the same, and therefore may have slightly different eclipse times. But even with these differences, the comparison with FireSat shows that the tool is producing reasonable data.

Lastly, the average data sent each transmission is assumed to be valid because it is based on the orbit, and on reports from STK.

Trending Analysis

By performing repeated runs of the tool, trends can be observed. The tool demonstrates how the three sub-systems change with respect to each other. By holding most inputs constant and changing one, the relationship between parameters can be seen. To further give examples of the tool, several trends were graphed.

Figure 3 shows how the amount of data sent changes with perigee altitude. For higher altitude orbits, the satellite has line-of-sight access to the ground site for longer, so more data can be sent each transmission. This graph was created with a fixed data rate of 10 kbps. By varying the data rate in addition to the altitude, it can be shown how total data sent is dependent on both data rate and orbit.

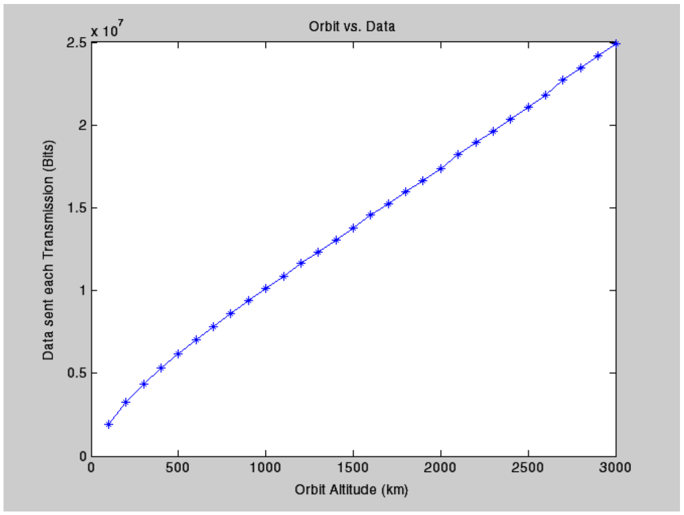

Figure 3. Perigee Altitude vs. Data Sent

Figure 4 shows how the power system changes with varying data rate, for a fixed orbit altitude. Both the mass of the power system and the size of the solar array increase with increasing data rate, to the point where the system would be unusable. To provide more information, this trend could be repeated for different orbit altitudes, to show how both altitude and data rate together affect the power system. The final trend shown, in Figure 5, demonstrates this. It plots how the power system mass and solar array size changes with changing altitude, for a fixed data rate of 10 kbps.

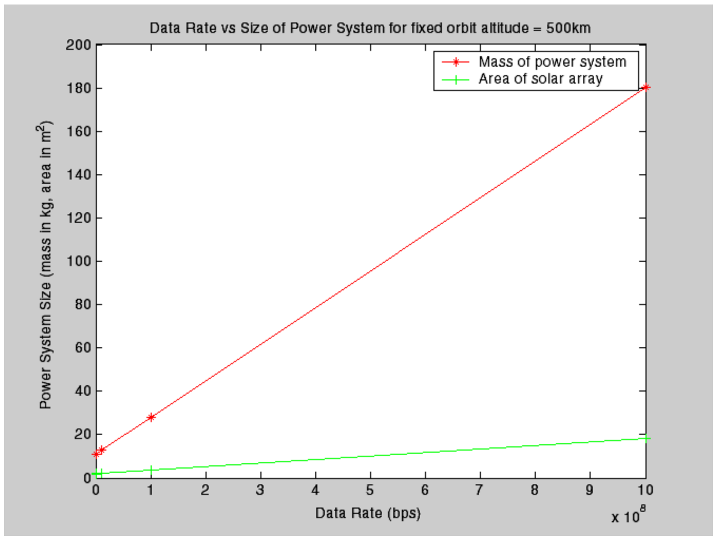

Figure 4. Data Rate vs. Power System

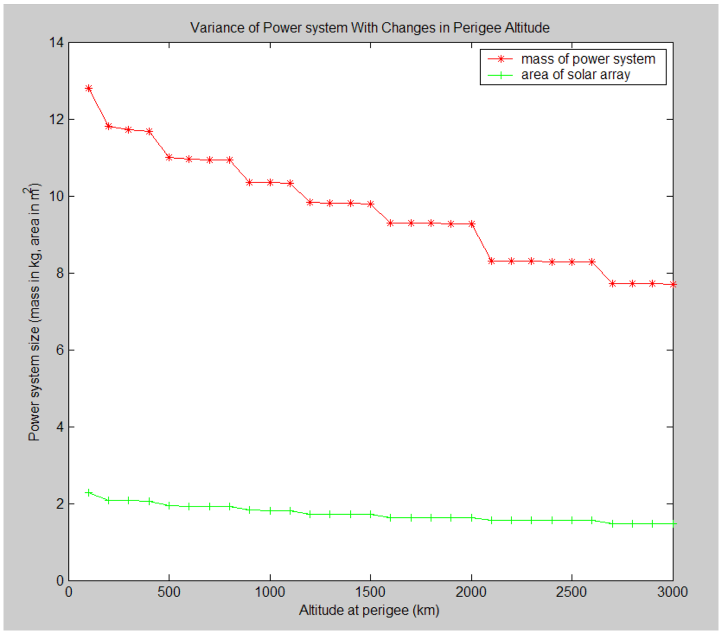

Figure 5. Perigee Altitude vs. Power System

APPENDIX A. SOURCES

Wertz and Larson, Space Mission Analysis and Design, Third Edition, El Segundo: Microcosm Press, 1999

Vallado, Fundamentals of Astrodynamics and Applications, New York: McGraw-Hill Co, 1997

APPENDIX B. CODE

Note: Code is long, but heavily commented (which makes it longer than it would be otherwise)

```
function [data_sent, orbit, M_sa, A_sa, M_bat] = ground_comm(site_lat, site_lon, pass_alt, dr, pow, start_date, life)

% top level function
% this function takes in user inputs, and calls functions to find orbit,
% amount of data sent and power for transmitter, and power system

% INPUTS
% site_lat: latitude of the ground site, in degrees
% site_lon: (east) longitude of the ground site, in degrees
% pass_alt: height above the ground the satellite should pass when passing
        % directly over the ground site, in kilometers
% dr: data rate of data transmitted from spacecraft to groundsite, in bps
% pow: power needs of the spacecraft, not including for data transmission,
        % in watts
% start_date: start time of mission, which is time satellite is first at
      % perigee over ground site. format is:
      % [year, month, day, hour, min]
% life: mission length, in years

% OUTPUTS
% data_sent: total amount of data sent during each pass (average), in bits
% orbit: orbital elements [semi-major axis (km), eccentricity, inclination
% (deg), argument of perigee (deg), longitude of ascending node (deg)
% M_sa: mass of solar array
% A_sa: area of solar array
% M_bat: mass of battery

if(pass_alt > 3000)
   disp('Altitude at perigee must be < 3000 km');
   return;
end

% first find gst of date
yr = start_date(1);
mo = start_date(2);
d = start_date(3);
h = start_date(4);
min = start_date(5);
gst = find_gst(yr, mo, d, h, min);

% then find orbit based on site, altitude requirement
 [period, orbit] = find_orbit(site_lat, site_lon, pass_alt, gst);
% set up orbit in STK, to find eclipse times and site passage times
[eclipse, pass_time] = run_stk(orbit, site_lat, site_lon, start_date);
% fnd power needs for transmission, amount of data sent each time
[data_sent, pow_dl] = transmitter(dr, pass_time);
% find solar array
[cell_type, M_sa, A_sa] = Power_Optimizing(life, pow, pow_dl, pass_time, eclipse, (period-eclipse), orbit(3)*pi/180);
% find battery
cyc = life*365*24*60/period;
[battery_name, M_bat] = Storage_Optimizing(life, cyc, pow, pow_dl, pass_time, eclipse);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function gst = find_gst(yr, mo, d, h, min)
```

```
% routine from Fundamentals of Astrodynamics and Applications, by Vallado
% first converts normal date format to julian date for the day
% then converts julian date and the time of day to greenwich sidereal time

% earth's mean angular rotation
w = 0.250684477337;

jd = 367*yr - floor((7*(yr + floor((mo+9)/12)))/4) + floor(275*mo/9) + d + 1721013.5;
tjd_ut = (jd - 2451545)/36525;
gst_day = 100.4606184 + 36000.77005361*tjd_ut + 0.00038793*tjd_ut^2 - (2.6e-8)*tjd_ut^3;
gst = gst_day + w*(h*60 + min);

while(gst > 360)
   gst = gst-360;
end
while(gst < 0)
   gst = gst + 360;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [period, orbit] = find_orbit(site_lat, site_lon, alt_p, gst)

% finds the orbital elements for an orbit that repeats once a sidereal day,
% and has it's radius of perigee over the ground site

% INPUTS
% site_lat: latitude of the ground site, in degrees
% site_lon: longitude of the ground site, in degrees
% alt_p: altitude of perigee of the orbit, in kilometers
% gst: start time as Greenwich sidereal time

% OUTPUTS
% orbit: orbital elements [semi-major axis (km), eccentricity, inclination
% (deg), argument of perigee (deg), longitude of ascending node (deg)
% period: orbit period in minutes

disp('Finding Orbit...')

% radius of the earth
r_earth = 6378.137;
% constant used for finding period: 2*pi/mu^.5
period_const = 0.00016587;
% length of sidereal day in minutes
sid_day = 1436.068167;
% radius of perigee
rp = r_earth + alt_p;
% start with e being nearly circular
% find period with this value, then adjust period so that it equals
% (1 sidereal day)/(integer number of revolutions)
% then, readjust e and a
e_initial = 0.01;
a_initial = rp/(1.0 - e_initial);
P_initial = period_const*a_initial^(3/2);
% we want number of revs to be an integer
n_revs = sid_day/P_initial;
% using floor to raise n_revs makes period go up slightly; so semi-major
% axis will go up slightly, radius of apogee will stay larger than radius
% of perigee, and eccentricity will stay positive
int_revs = floor(n_revs);
% period, semi-major axis, eccentricity
P = sid_day/int_revs;
a = (P/period_const)^(2/3);
e = 1.0 - rp/a;
% inclination
if(abs(site_lat) > 85)
   i = 90;
else
   i = abs(site_lat) + 5;
```

```
end
% argument of perigee to put perigee over the ground site
w = asin(sin(site_lat*pi/180)/sin(i*pi/180))*180/pi;
while(w > 360)
    w = w - 360;
end
while(w < 0)
    w = w + 360;
end
% longitude of ascending node
% first find target position's longitude meridian
b = asin(cos(i*pi/180)/cos(site_lat*pi/180));
lm = acos(cos(b)/sin(i*pi/180));
l_an = gst + site_lon - lm*180/pi;
while(l_an > 360)
    l_an = l_an - 360;
end
while(l_an < 0)
    l_an = l_an + 360;
end

orbit = [a, e, i, w, l_an];
period = P;

disp('semi-major axis (km): ');
disp(a);
disp('eccentricty: ');
disp(e);
disp('inclination (deg): ')
disp(i);
disp('argument of perigee (deg): ');
disp(w);
disp('longitude of ascending node (deg): ')
disp(l_an);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [eclipse, pass_time] = run_stk(orbit, site_lat, site_lon, date)

% opens stk, sets up the orbit and the ground site
% runs scenario
% finds average time craft is in eclipse each orbit
% finds average time of each pass over ground site (average of closest pass
% each day)

% INPUTS
% orbit: [semi-major axis (km), eccentricity, i (degrees), w (degrees), ascending node (degrees)]
% site_lat: site latitude in degrees
% site_lon: site longitude in degrees
% date: start date of orbit (time of first perigee)
        % [year, month, day, hour, minute]

% OUTPUTS
% eclipse: average time each orbit craft is in eclipse, in minutes
% pass_time: average time of the longest pass each day, in minutes

disp('Connecting to STK...');

year = date(1);
day = date(3);
hour = date(4);
min = date(5);

% initialize connection to STK, if this has not yet been done
stkinit;
% use the default connection data for connecting to STK
% port should be 5001, hostname 'localhost': this setup is what STK
% recommends and guides you to do when STK and Matlab are on the same
% machine
remMachine = stkDefaultHost;
```

```
%open the connection to STK
conid=stkOpen(remMachine);
% first check to see if a scenario is open
% if there is, close it
scen_open = stkValidScen;
if scen_open == 1
   stkUnload('/*')
end
% set up scenario
cmd = 'New / Scenario ground_comm';
stkExec(conid, cmd);
% put the satellite in the scenario
cmd = 'New / */Satellite sat1';
stkExec(conid, cmd);
% put the ground site in the scenario
cmd = 'New / */Facility ground_site';
stkExec(conid, cmd);
if(date(2) == 1)
   month = 'Jan';
elseif(date(2) == 2)
   month = 'Feb';
elseif(date(2) == 3)
   month = 'Mar';
elseif(date(2) == 4)
   month = 'Apr';
elseif(date(2) == 5)
   month = 'May';
elseif(date(2) == 6)
   month = 'June';
elseif(date(2) == 7)
   month = 'July';
elseif(date(2) == 8)
   month = 'Aug';
elseif(date(2) == 9)
   month = 'Sept';
elseif(date(2) == 10)
   month = 'Oct';
elseif(date(2) == 11)
   month = 'Nov';
elseif(date(2) == 12)
   month = 'Dec';
else
   disp('invalid date, month must be 1 through 12');
end
startDate = ['''', num2str(day), ' ', month, ' ', num2str(year), ' ', num2str(hour), ':', num2str(min), ':00.0'''];
stopDay = day + 12;
stopDate = ['''', num2str(stopDay), ' ', month ' ', num2str(year), ' ', num2str(hour), ':', num2str(min) ':00.0'''];
epochDate = startDate;
% set the scenario epoch
cmd = 'SetUnits / km sec GregUTC';
stkExec(conid, cmd);
cmd = ['SetEpoch * ' epochDate];
stkExec(conid, cmd);
stkSyncEpoch;
% set the time period for the scenario
stkSetTimePeriod(startDate, stopDate, 'GREGUTC');
% set the animation parameters
tmp = ['SetValues ', startDate, ' 60 0.1'];
rtn = stkConnect(conid,'Animate','Scenario/ground_comm',tmp);
rtn = stkConnect(conid,'Animate','Scenario/ground_comm','Reset');
% set orbit elements
cmd = ['SetState */Satellite/sat1 Classical TwoBody ' startDate, ' ', stopDate, ' 60 J2000 ', epochDate, ' ', num2str(orbit(1)), ' ', ...
      num2str(orbit(2)), ' ', num2str(orbit(3)), ' ', num2str(orbit(4)), ' ', num2str(orbit(5)), ' 0.0'];
stkExec(conid, cmd);
% set ground site position
cmd = ['SetPosition */Facility/ground_site Geocentric ', num2str(site_lat), ' ', num2str(site_lon), ' 0.0'];
stkExec(conid, cmd);
% get access data, satellite to ground site
intervals = stkAccess('*/Satellite/sat1', '*/Facility/ground_site');
```

```
[n,m] = size(intervals);
% loop through access data, eliminating first partial day and last partial day,
% find the pass overhead - at perigee, the repeating pass once a sidereal
% day - that we want to use

% length of sidereal day in minutes
sid_day = 1436.068167;
perigee_pass = zeros(1,10);
for(j=1:10)
   for(i=1:n)
       start_min = intervals(i).start/60;
       stop_min = intervals(i).stop/60;
       if(start_min < j*sid_day && stop_min > j*sid_day)
           perigee_pass(1,j) = stop_min - start_min;
       end
   end
end
% get average pass length
pass_time = mean(perigee_pass);
% find average eclipse time
[data, names] = stkReport('*/Satellite/sat1', 'Eclipse Times');
all_times = stkFindData(data{1}, 'Total Duration');
ecl_numbers = stkFindData(data{1}, 'Start Pass Number');
% for some reason ecl_numbers is not numeric, and won't convert right as an
% array, so convert one at a time
for(i=1:length(ecl_numbers))
    ecl_n(i) = str2num(ecl_numbers(i,:));
end
% the eclipse report lists several lines of data for each eclipse; the
% total duration values are repeated, but we don't want duplicates - find
% uniques; i are the indices we want
[b, i, j] = unique(ecl_n);
unique_times = all_times(i);
% remove first and last times, in case they are partial eclipse values
% average the rest
len = length(unique_times);
eclipse = mean(unique_times(2:len-1))/60;
stkClose(conid);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [data_sent, power] = transmitter(dr, pass_time)

% INPUTS
% dr: data rate of data transmitted from spacecraft to groundsite, in bps
% pass_time: average time of each spacecraft pass where downlink is made (minutes)

% OUTPUTS
% data_sent: average amount of data that can be sent each pass (bits)
% power: power needed for each tranmission (W)

% From figure 13-5 from SMAD
% power for different data rates
% for the ground station diameter of 5 meters, the plot indicates that
% power is equal to approximately 10x the data rate, in Mbps
power = 10*(dr/10e5);
% total data sent on each pass (on average)
data_sent = dr*(pass_time*60 - 2*60);
disp('Power needed for data transmission (W): ');
disp(power);
disp('Bits of data sent each transmission: ');
disp(data_sent);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [cell_type, M_sa, area] = Power_Optimizing(L_m, P_av, P_dl, T_dl, T_e, T_d, inc)
% This function selects the optimal solar cell type as a function of area

% INPUTS
% L_m: mission life (years)
% P_av: power requirement, not including data transmission needs (W)
% P_dl: power required for data tranmission needs (W)
% T_dl: time to downlink data (minutes)
% T_e: average time of eclipse (minutes)
% T_d: average time not in eclipse (minutes)
% inc: incliation of the orbit (rad)

% OUTPUTS
% cell_type: name of solar cell
% M_sa: mass of solar array (kg)
% area: area of solar array (m^2)

disp('Finding Solar Array...');

% Defining problem variables (source: SMAD)
X_e = .65;                    %energy transfer efficiency during eclipse, direct energy transfer
X_d = .85;                    %energy transfer efficiency during daylight, direct energy transfer
S_i = 1367;                   %solar illumination intensity, W/m^2
X_si = .148;                  %solar cell production efficiency, Silicon
X_th = .050;                  %solar cell production efficiency, Thin Sheet Amorphous Si
X_ga = .185;                  %solar cell production efficiency, Gallium Arsenide
X_ip = .180;                  %solar cell production efficiency, Indium Phosphide
X_mj = .220;                  %solar cell production efficiency, Multijunction GaInP/GaAs
I_d = .77;                    %inherent degradation
theta = 23.5 * (pi/180) + inc;      %worst-case sun angle, radians
% Compute power collected by solar array during daylight
P_sa = (P_av * T_e / X_e + P_av * T_d / X_d + P_dl * T_dl) / T_d;
% Vector of solar cell efficiencies
X = [X_si X_th X_ga X_ip X_mj];
% Compute power ouptut for various solar cell types (W/m^2)
P_o = S_i .* [X];
% Compute beginning-of-life power (W) per unit area for various solar cell types
P_bol = (cos(theta) * I_d) .* P_o;
% Compute actual lifetime degradation for various solar cell types
A_d = [.0375 .0375 .0275 .0375 .0375];
L_d = (1 - A_d) .^ L_m;
% Compute end-of-life power (W) per unit area for various solar cell types
P_eol = P_bol .* L_d;
% Compute solar array area (m^2) required to support input power requirement
A_sa = P_sa ./ P_eol;
% List best solar cell type (smallest cell area)
fprintf('\n');
disp('Best solar cell type, minimizing for area:');
Min = min(A_sa);
if Min == A_sa(1);
   cell_type = 'Silicon'
elseif Min == A_sa(2);
   cell_type = 'Thin Sheet Amorphous Si'
elseif Min == A_sa(3);
   cell_type = 'Gallium Arsenide'
elseif Min == A_sa(4);
   cell_type = 'Indium Phosphide'
elseif Min == A_sa(5);
   cell_type = 'Multijuction GaInP/GaAs'
end
% Compute mass (kg) of solar array
M_sa = .04 * P_sa;
% Area of chosen solar array
area = Min;
disp('Solar array mass (kg): ');
disp(M_sa);
disp('Solar array area (m^2): ');
disp(area);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [battery_name, mass] = Storage_Optmizing(L_m, cyc, P_av, P_dl, T_dl, T_e)
% This function selects the optimal battery as a function of mass

% INPUTS
% L_m: life of mission (years)
% cyc: cycle life (number of times satellite goes in and out of sun)
% P_av: average power requirement, not including for data transmission (W)
% P_dl: power for data transmission (W)
% T_dl: time of each data transmission (minutes)
% T_e: time of eclipse (minutes)

% OUTPUTS
% battery_name
% mass: mass of battery (kg)

disp('Finding Battery...');

% Defining problem variables (source: SMAD)
X_nc = 30;      %Nickel-Cadium specefic energy (W*hr/kg)
X_nhi = 43;     %Nickel-Hydrogen (individual pressure) specefic energy (W*hr/kg)
X_nhc = 56;     %Nickel-Hydrogen (common pressure) specefic energy (W*hr/kg)
X_nhs = 57;     %Nickel-Hydrogen (single pressure) specefic energy (W*hr/kg)
X_li = 110;     %Lithium-Ion specefic energy (W*hr/kg)
X_ss = 210;     %Sodium-Sulfur specefic energy (W*hr/kg)
n = 0.90;       %Transmission effeciency between the battery and the load
% Vector of specific energy densities
X = [X_nc X_nhi X_nhc X_nhs];
% Determine Depth-of-Discharge
if cyc < 1000
   DoD_nc = .65;
   DoD_nhi = .90;
   DoD_nhc = .90;
   DoD_nhs = .90;
elseif cyc >= 1000 & cyc < 2000
   DoD_nc = .55;
   DoD_nhi = .75;
   DoD_nhc = .75;
   DoD_nhs = .75;
elseif cyc >= 2000 & cyc < 4000
   DoD_nc = .45;
   DoD_nhi = .65;
   DoD_nhc = .65;
   DoD_nhs = .65;
elseif cyc >= 4000 & cyc < 10000
   DoD_nc = .35;
   DoD_nhi = .55;
   DoD_nhc = .55;
   DoD_nhs = .55;
elseif cyc >= 10000 & cyc < 20000
   DoD_nc = .30;
   DoD_nhi = .50;
   DoD_nhc = .50;
   DoD_nhs = .50;
elseif cyc >= 20000 & cyc < 40000
   DoD_nc = .20;
   DoD_nhi = .40;
   DoD_nhc = .40;
   DoD_nhs = .40;
elseif cyc > 40000
   DoD_nc = .15;
   DoD_nhi = .35;
   DoD_nhc = .35;
   DoD_nhs = .35;
end
% Determine Power Required During Eclipse
% Design for worst-case: assume downlink during eclipse
P_e = (P_av * T_e + P_dl * T_dl) / T_e;
% Determine Battery Capacity (W*hr)
DoD = [DoD_nc DoD_nhi DoD_nhc DoD_nhs];
C_r = (P_e * T_e/60) .* ((n * DoD).^-1);
% Determine Mass of Storage System
M = C_r ./ X;
% Display Lightest Power Storage System
fprintf('\n');
```

```
disp('Best secondary battery type (lightest):');
Min = min(M);
if Min == M(1);
    battery_name = 'Nickel-Cadium'
elseif Min == M(2);
    battery_name = 'Nickel-Hydrogen (individual pressure vessel)'
elseif Min == M(3);
    battery_name = 'Nickel-Hydrogen (common pressure vessel)'
elseif Min == M(4);
    battery_name = 'Nickel-Hydrogen (single pressure vessel)'
end
% Display Mass of Lightest Power Storage System
disp('Battery Mass (kg):');
Min = min(M);
disp(Min)
mass = Min;
```