

Seung Chung  
Steve Paschall  
Kathryn Weiss  
16.851 – Satellite Engineering  
Due: Wednesday, October 29, 2003

### Problem Set #4: Final Report

Subject: Orbit, Power and Communication Subsystems

Motivation: The power and communications subsystems aboard a spacecraft interact with one another as a function of the spacecraft's orbit to achieve a set of requirements. These requirements often involve pointing the satellite at a given location for a specified amount of time, and then transmitting the accumulated data to a particular ground location at a later time in the orbit. A mission also has a set of standard power and communication requirements that depend on the orbit:

1. Given the power requirement over the mission lifetime, the power subsystem (solar arrays and batteries) mass depends on the orbit.
2. The amount of data a spacecraft can transmit depends on the transponder bandwidth of the communication subsystem and the amount of time over which a specified set of ground stations are visible to the spacecraft. The mission may also require a specified amount of data transmission.

Furthermore, the power and communications subsystems impose additional mutual constraints on one another. The power required by the spacecraft depends on the communications subsystem design and the duration of its use. Using the trades between the Orbit, Power and Communication Subsystems as guidelines, ranges of orbit size and inclination, sizes of solar arrays and batteries, and communication subsystem power usage as well as antenna size can be found and optimized.

Problem Statement: What combination of orbit size and inclination, solar array and battery sizes, and communication subsystem power usage and antenna size yields an optimal solution given a specified ground station downlink site? The objective is not to model each subsystem to a high fidelity, but rather to better understand and model the mutual dependencies of the subsystems.

Approach: A program will be written using Matlab and STK. The program will use information input by the user to compare different combinations of orbits, solar array and battery size, and communication subsystem power needs and antenna size. The program will then output a set of optimal ranges for each of the subsystems. A more detailed description of our approach is found in the Solution section.

Solution: The inputs include the amount of data that needs to be transmitted and a predetermined fixed data rate. This data rate was mentioned in SMAD to be approximately 9.6kbps, which is based on current technological limitations. The user also inputs the latitude of the ground station with which the satellite will communicate as well as the total mission duration of the satellite.

Finally, the user inputs much power the satellite uses during daylight and eclipse and the mission lifetime.

Using the equations from page 546 of SMAD, the first module calculates the  $T$  (the amount of time the satellite must be in view of the ground station) needed to fulfill the data transfer requirement:

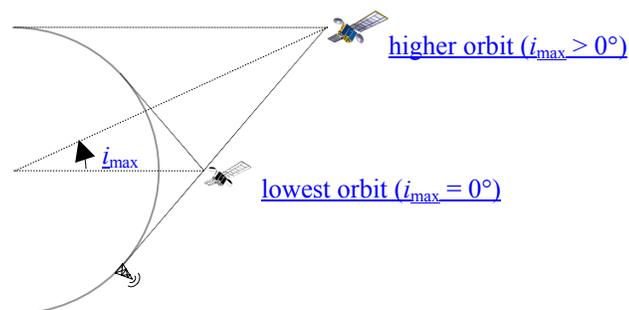
$$T = \frac{M \cdot D}{R} + T_{initiate},$$

where  $D$  (the total bits of data that must be transmitted) is defined by the mission requirement,  $R$  (the data transfer rate in bits per second), is known, and  $T_{initiate}$  (the time required to initiate communications) can be assumed to be two minutes according to SMAD.  $M$  (the margin needed to account for ground station down time) is approximated to be two or three in SMAD; in the implemented module, a margin of three is used.

Next, a set of orbits that provides  $T$  communication time is computed. Given the requirement that the satellite must communicate with the specified ground station on each flyby and the assumed restriction that only circular LEO's are desired, the orbit radius,  $r$ , and inclination,  $i$ , are the only constrained orbital parameters. Chapter 5 of SMAD discusses the computation of the ground station viewing time for a given LEO. This approach is adapted to instead compute set orbits that provide  $T$  communication time:

1. Compute minimum orbit radius that provides  $T$  communication time on each flyby.
2. For a set of feasible orbit radii, compute the maximum allowable orbit inclination that guarantees  $T$  communication time on all flybys.

For the first step, the minimum orbit radius that assures  $T$  communication time on each flyby is computed by recognizing the fact that this orbit must be equatorial. That is, if the orbit is inclined, the altitude of the satellite must be raised to assure that the ground station is still in view when the satellite is in the opposite side of the hemisphere with respect to the ground station latitude (note, with an inclined LEO, the ground track oscillates between northern and southern hemisphere). As the orbit radius is raised, however, the inclination can be increased and still assure communication, illustrated in Figure 1.



**Figure 1. Calculating Maximum Inclination**

Given the range of the feasible orbit radii, the second step of the algorithm computes the maximum inclination,  $i_{\max}$ , that guarantees the required communication time. The worst case is when the satellite is in the opposite hemisphere with respect to the ground station latitude, at the pole of its orbit. That is, the instantaneous longitude of the ascending node is  $90^\circ$  from the ground station's longitude. With this knowledge, spherical trigonometry along with the orbital period can be used to compute  $i_{\max}$  and the maximum distance from the satellite to the ground station during communication (see SMAD Chapter 5 for all necessary equations). Note that the angular rotation rate is assumed negligible relative to the orbital period of circular LEO.

Figure 2 and Figure 3, respectively, represents the maximum inclination and distance as a function of orbit radius and latitude for 1 MB of data transmitted at 9.6kps. Note that the maximum inclination angle increases as the ground station is moved toward the equator as expected.

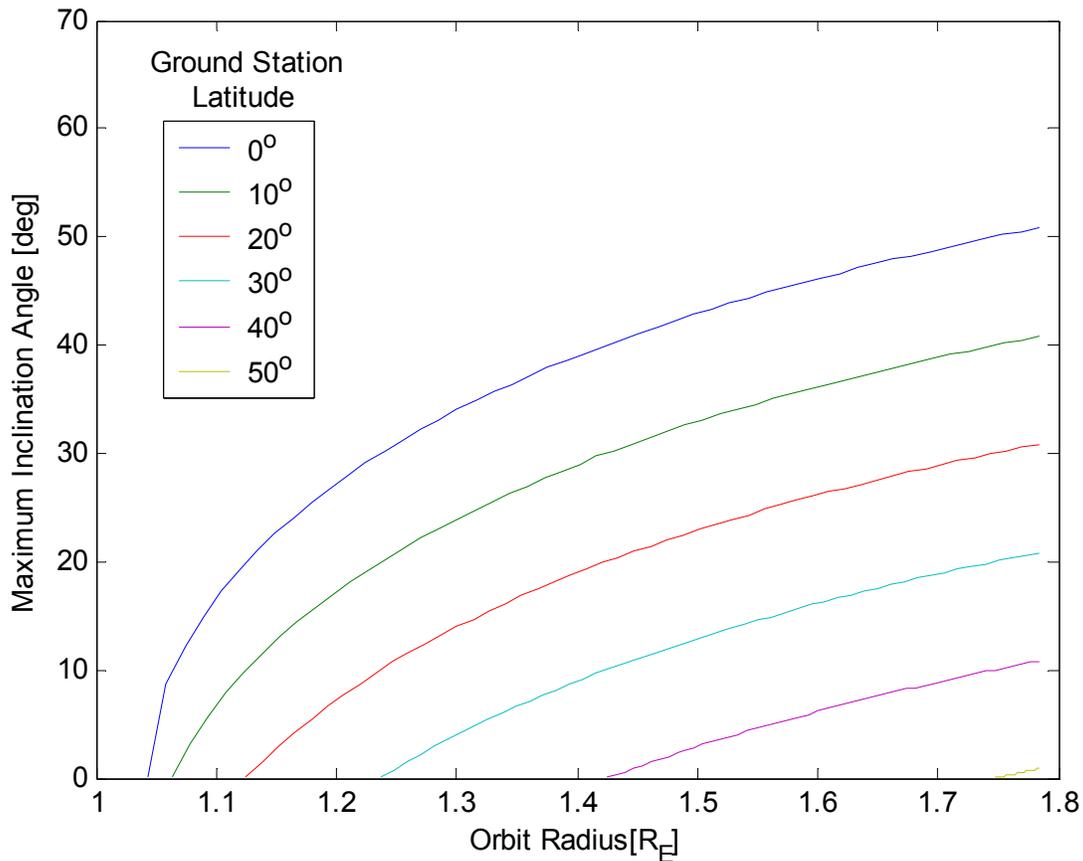
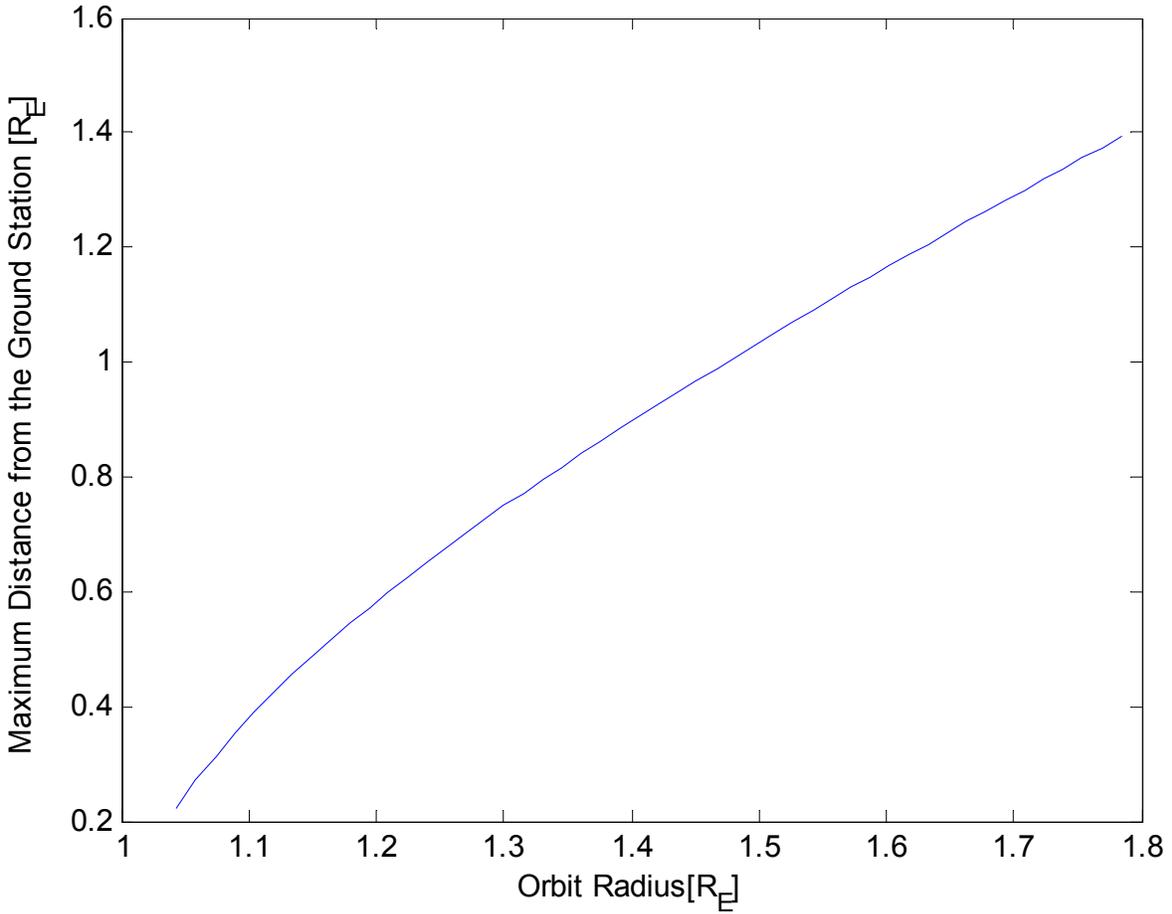


Figure 2. Orbit Radius vs. Inclination



**Figure 3. Orbit vs. Distance**

The second module takes as inputs the orbit that fulfills the input requirements from the previous module, the maximum distance to the ground station, the transmission frequency, and the diameter of the ground antenna. Using the following equations found in section 13.3 of SMAD, this module uses the inputs to determine how large the transmitter diameter needs to be to send the required amount of data. The module also determines how much power is needed by the communication subsystem to use the antenna.

$$L = \left( \frac{4\pi \cdot d_{ground} \cdot f}{c} \right)^2$$

$$d_{sat} = \sqrt{\frac{(G_{sat} \cdot c)^2}{E_{sat} \pi^2 \cdot f^2}}$$

$$G_{ground} = E_{ground} \left( \frac{\pi \cdot d_{ground} \cdot f}{c} \right)^2$$

$$P = \frac{SNR \cdot N_o \cdot R \cdot L \cdot L_a}{G_{sat} \cdot G_{ground}}$$

where L is the signal loss in free space based on the distance to the ground, frequency and speed of light; dsat is the diameter of the spacecraft's antenna based on the Gain (G), speed of light (c), efficiency of the satellite (E) and frequency (f); P is the power needed by the communication subsystem based on the signal-to-noise ratio (SNR), the noise density (No), data rate (R), free space signal loss, approximate atmospheric attenuation (La) and the gains of the satellite and ground antennas.

The final module's inputs include the power needed during daylight and eclipse as well as the mission duration and the calculated orbit's radius and inclination. The module uses STK to determine the eclipse times of the satellite. Based on the eclipse times and the power required by the satellite and using the following equations from Chapter 11.4 of SMAD, the module sizes solar arrays and batteries to fulfill these requirements.

Solar Array Sizing Equations:

$$A_{sa} = P_{sa} / P_{EOL} \quad \text{where}$$

$$P_{sa} = \frac{\left(\frac{P_e T_e}{X_e}\right) + \left(\frac{P_d T_d}{X_d}\right)}{T_d} \quad \text{and} \quad \begin{aligned} P_{EOL} &= P_{BOL} L_d \\ P_{BOL} &= P_o I_d \cos \theta \\ L_d &= (1 - \text{degradation} / \text{yr})^{\text{satellitelife}} \end{aligned}$$

Battery Sizing Equations:

$$C_r = \frac{P_e T_e}{(DOD) N n}$$

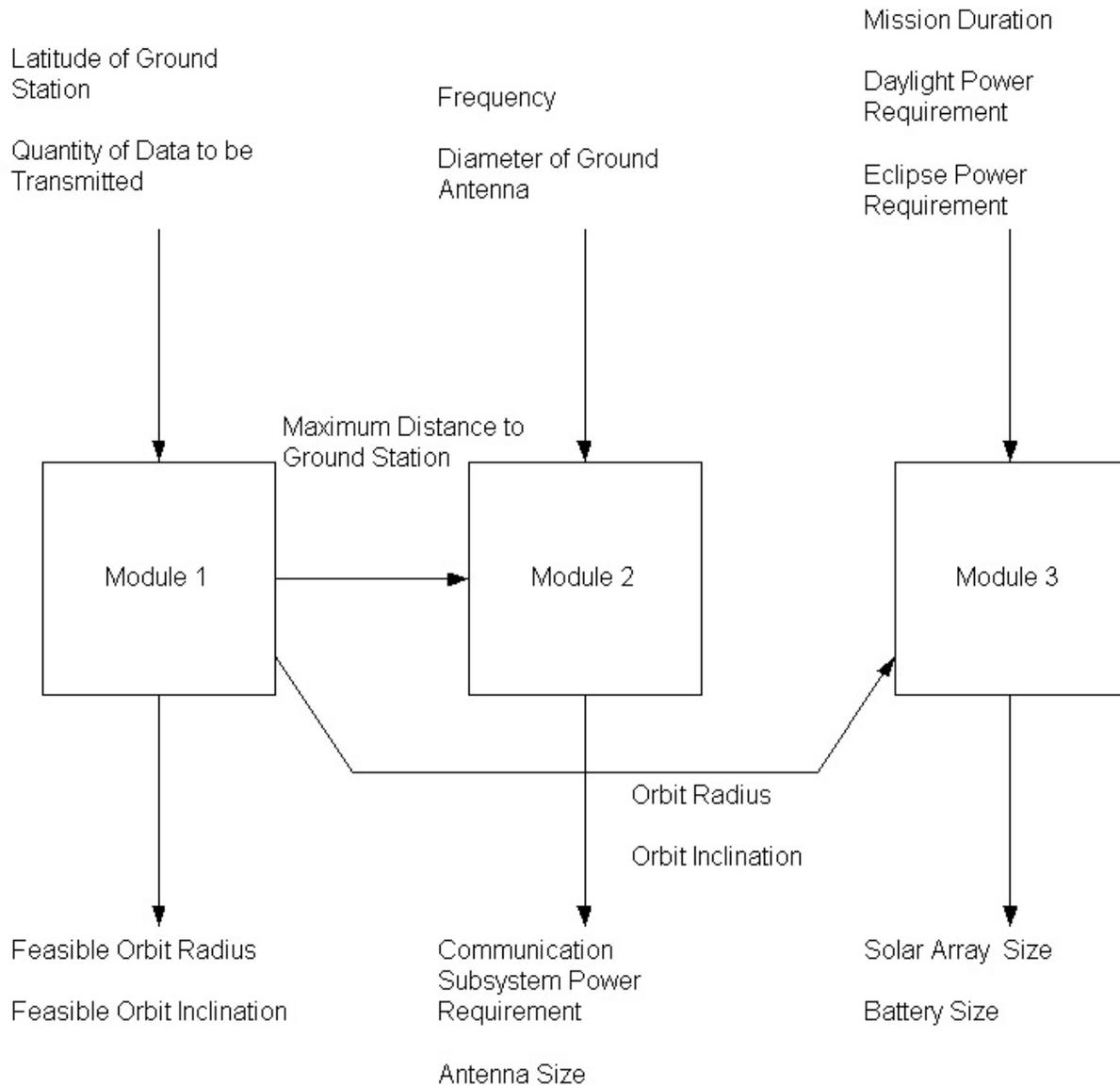
$$M = \frac{C_r}{E_D}$$

Table 1 lists the variable names, their definitions and whether or not they are fixed within the code. The module will output the solar array and battery mass.

Variable	Definition	Source	Units
Asa	Area of the solar array	Calculated	m*m
Peol	Power needed at end of life	Input	Watts
Pbol	Power needed at beginning of life	Input	Watts
Ld	Life degradation	Calculated	Unitless
	Degradation per year	Worst Case: Ga-arsenide = 0.0275 Silicon = 0.0375 Multijunction = 0.005	Unitless
Satellitelifelife	Mission Duration	Input	Years
Po	Power output	Ga-arsenide = 252.895 Silicon = 202.316 Multijunction = 300.74	Watts / m*m
Id	Inherent Degredation	Worst Case Id = 0.77	Unitless
$\theta$	Sun incidence angle	Worst Case $\theta = 23.5$	Degrees
Psa	Total power the solar array must provide	Calculated	Watts
Pe and Pd	Power requirement during eclipse and daylight	Calculated	Watts
Te and Td	Time in eclipse and daylight	Calculated by STK	Seconds
Xe and Xd	Efficiencies during eclipse and daylight	Direct Energy Transfer: Xe = 0.65 Xd = 0.85 Peak Power Tracking: Xe = 0.60 Xd = 0.80	Unitless
Cr	Battery capacity	Calculated	Watt hours
DOD	Depth of discharge	Worst Case at LEO: NiH2 = 40% NiCd = 10%	Unitless
N	Number of batteries	N = 1	Unitless
n	Battery to load transmission efficiency	n = 0.9	Unitless
M	Mass of battery	Calculated	Kg
Ed	Specific Energy Density	NiH2 = 35 NiCd = 45	Watt hours / kilogram

**Table 1. Power Subsystem Sizing Equation Variables**

Figure 4 illustrates the flow of the program including inputs and outputs to each model using a blackbox diagram.



**Figure 4. Blackbox Diagram**

Assumptions:

1. The orbits are circular.
2. The orbits are all Low-Earth Orbits (below 11,000 km).
3. The orbits do not go over the poles.
4. The amount of data that needs to be transmitted is sent on every fly-by.

Sample Test Runs and Conclusions:

Three sample test cases were run using the following input data. The three cases accounted for three different operating frequencies.

- Data Quantity: 8e6 bits
- Ground Station Latitude: 12 deg
- Ground Station Antenna Size: 3 m

- Daylight Power Needed: 110 W
- Eclipse Power Needed: 110 W
- Mission Duration: 5 years
- Frequency: S-Band, C-Band and Ku-Band

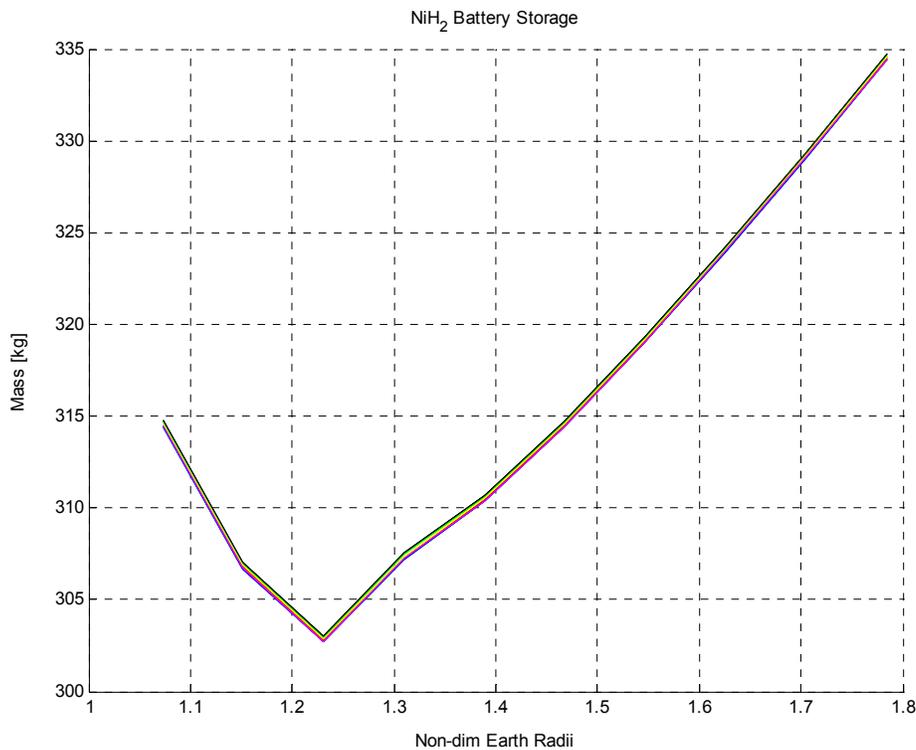
Outputs:

Satellite Antenna Size:

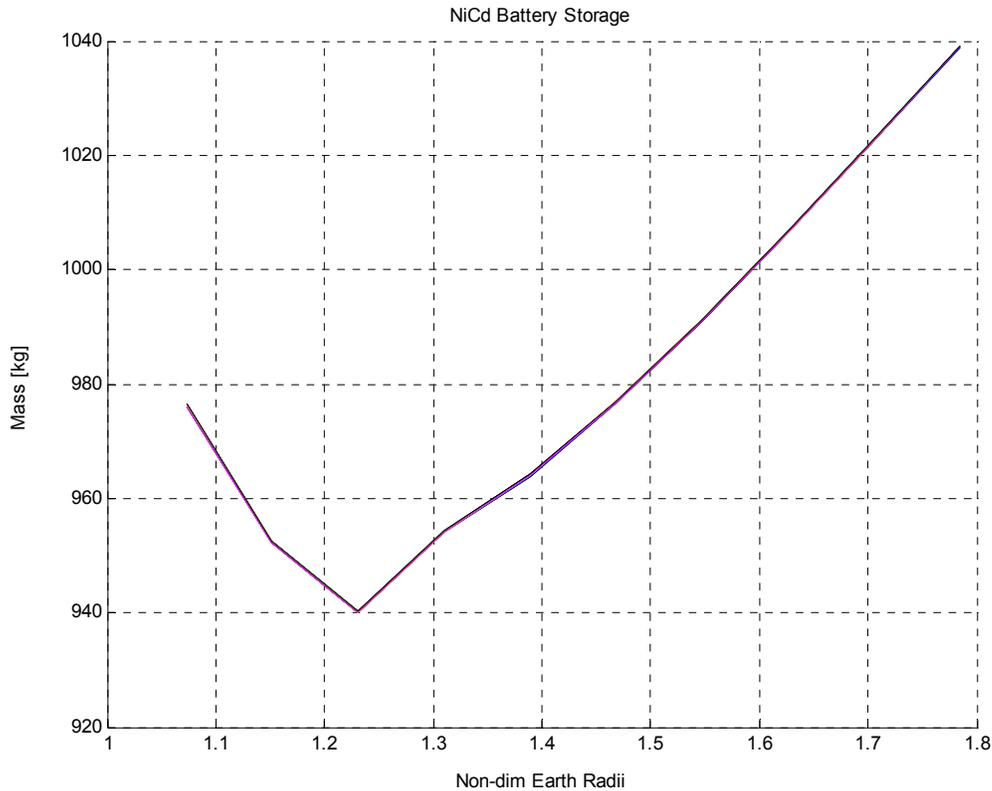
- S-Band Frequency (2.2e9 Hz): 0.475 m
- C-Band Frequency (4e9 Hz): 0.261 m
- Ku-Band Frequency (12.0e9 Hz): 0.087 m

Figures 5 and 6 illustrate the mass of the power subsystem (solar arrays and NiH<sub>2</sub> and NiCd batteries respectively) with respect to the orbit radius. The orbit radius has been normalized with respect to the Earth's radius.

The different solar-array materials and power configurations had a negligible effect on the power subsystem mass. The communication subsystem bandwidth also had no effect on the amount of power required by the communication subsystem. This was found to be a function of distance from the ground station, which grows as the orbit radius increases. The optimal solution to the problem lies at the minimum mass point on the curves.



**Figure 5. Mass of Power Subsystem with NiH<sub>2</sub> Batteries vs. Non-dimensional Orbit Radius**



**Figure 5. Mass of Power Subsystem with NiCD Batteries vs. Non-dimensional Orbit Radius**

Code:

```
function [radius, inclination, comm_power, diameter, solar_array_size, battery_size, power_mass] = Scenario(data_amt, latitude, frequency,
diameter_grnd_antenna, daylight_power_needed, eclipse_power_needed, mission_lifetime)
    data_rate = 96000; % bps from SMAD
    epsilon_min = 5 * pi/180; % radians

    time = compute_communication_time(data_amt, data_rate);
    [radius, inclination, max_distance] = compute_feasible_circular_LEO(time, epsilon_min, latitude);
    tic

    for i=1:length(radius)
        [comm_power(i), diameter(i)] = comm_sys(data_rate, max_distance(i), frequency, diameter_grnd_antenna);
        periods = calculatePeriods(radius(i), inclination(i));
        solar_array_size(i,:) = calculatePSA(daylight_power_needed, eclipse_power_needed, mission_lifetime, periods);
        battery_size(i,:) = size_batteries(eclipse_power_needed, periods);

        for j=1:length(solar_array_size(i,:)) %solar array loop
            for k = 1:length(battery_size(i,:))
                power_mass(i,2*(j - 1) + k) = solar_array_size(i,j) + battery_size(i,k);
            end
        end
    end
    toc
end

function [r, i_max, D_max] = compute_feasible_circular_LEO(T_min, epsilon_min, lat_gs)
% [r, i_max, D_max] = compute_feasible_circular_LEO(T_min, epsilon_min, lat_gs)
% Input
% T_min    minimum communication time per fly-by [sec]
```

```

% epsilon_min    minimum satellite elevation from surface [rad]
% lat_gs        latitude of the ground station [rad]
% Output
% r            radius of the orbit [m]
% i_max        maximum inclination [rad]
% D_max        maximum distance to ground station [m]
mu_E = 398600.4418e9; % Earth gravitational constant [m^3/s^2]
R_E = 6378136.49;    % Earth equatorial radius [m]
omega_E = 7.292115e-5; % Earth angular velocity [rad/s]
r_max = R_E + 5000000; % Maximum radius for which this module is valid [m]
n = 10;           % Number of data points

% Compute the minimum radius for which the ground station is in view for at least T_min.
r_min = fsolve(@min_radius_function, 10000000000, optimset, epsilon_min, lat_gs, T_min);
% If the minimum radius is greater than the maximum radius, then this
% module is no longer valid for the problem.
if (r_min > r_max | abs(imag(r_min)) > 1)
    error('No solution can be found for the given problem using this module!');
else
    r_min = real(r_min);
end

% Generate a set of radius
if (n <= 1)
    r = r_min;
else
    r = r_min:(r_max-r_min)/ceil(n-1):r_max;
end

rho = asin(R_E./r); % Earth Angular Radius
eta_max = asin(sin(rho)*cos(epsilon_min)); % Maximum nadir angle [rad]
lambda_max = pi/2 - epsilon_min - eta_max; % Maximum Earth central angle [rad]
P = 2*pi*sqrt(r.^3/mu_E); % Period of the orbit
lambda_min = acos(cos(lambda_max)./cos(T_min*pi./P)); % Worst case minimum Earth central angle [rad]
lat_pole_max = lat_gs - lambda_min; % Minimum latitude of the instantaneous orbit pole [rad]
i_max = lat_pole_max; % Maximum orbit inclination
D_max = R_E*sin(lambda_max)./sin(eta_max); % Maximum distance to the ground station [m]

%figure(3)
%plot(r/R_E,sin(lambda_max),r/R_E,sin(eta_max),r/R_E,sin(lambda_max)/sin(eta_max))

function x = min_radius_function(r, epsilon_min, lat_gs, T)
mu_E = 398600.4418e9; % Earth gravitational constant [m^3/s^2]
R_E = 6378136.49;    % Earth equatorial radius [m]

rho = asin(R_E./r); % Earth Angular Radius
eta_max = asin(sin(rho)*cos(epsilon_min)); % Maximum nadir angle [rad]
lambda_max = pi/2 - epsilon_min - eta_max; % Maximum Earth central angle [rad]
P = 2*pi*sqrt(r.^3/mu_E); % Period of the orbit
lat_pole_min = 0; % Minimum latitude of the instantaneous orbit pole [rad]
lambda_min_max = lat_gs - lat_pole_min; % Worst case minimum Earth central angle [rad]
x = P/pi.*acos(cos(lambda_max)./cos(lambda_min_max)) - T; % The computed communication time should equal the required communication
time.

function T_max = compute_communication_time(D_max, R_min)
% T_max = compute_communication_time(D_max,R_min)
% This module assume communication initiation time of 2 min and adds in a
% margin of 3 to the quantity of data.
%
% Ref: Wertz and Larson. Space Mission Analysis and Design, 2nd ed.
%
% Input
% D_max    Maximum quantity of Data [bit]
% R_min    Minimum data transfer rate [bit/sec]
% Output
% T_max    Maximum required communication time [sec]
T_initiate = 2*60; % Communication initiation time [sec]
M = 3; % Margin to account for missed passes
T_max = (D_max*M/R_min + T_initiate); % From SMAD 3rd ed.

```

```

function [output_data] = calculatePSA(dpn, epn, lifetime, periods)
Xe_direct_energy_transfer = 0.65; % Efficiency during eclipse for direct energy transfer
Xd_direct_energy_transfer = 0.85; % Efficiency during daylight for direct energy transfer
Xe_peak_power_tracking = 0.6; % Efficiency during eclipse for peak power tracking
Xd_peak_power_tracking = 0.8; % Efficiency during daylight for peak power tracking
Id = 0.77; % Nominal value for inherent degradation
theta = 0.4101; % The solar array is at worstcase Sun angle between equatorial and ecliptic planes
material_degradation_GA = .0275; % Gallium Arsenide degrades at 2.75% per year (worst case)
material_degradation_multijunction = .005; % Multijunction Solar cells degrade at 0.5% per year (worst case)
material_degradation_Si = .0375; % Silicon degrades at 2.75% per year (worst case)

% [periods] = calculatePeriods(altitude, inclination);
[periods] = [periods] / 60; % Convert seconds to minutes

psa_det = ((epn * periods(1)) / Xe_direct_energy_transfer + (dpn * periods(2)) / Xd_direct_energy_transfer) / periods(2);
psa_ppt = ((epn * periods(1)) / Xe_peak_power_tracking + (dpn * periods(2)) / Xd_peak_power_tracking) / periods(2);

power_output_Si = 202.316; % 14.8% * 1,367 W/m^2 (incident solar radiation)
power_BOL_Si = powerBeginningLife(power_output_Si, Id, theta);
power_EOL_Si = powerEndLife(power_BOL_Si, material_degradation_Si, lifetime);

power_output_multijunction = 300.74; % 22% * 1,367 W/m^2 (incident solar radiation)
power_BOL_multijunction = powerBeginningLife(power_output_multijunction, Id, theta);
power_EOL_multijunction = powerEndLife(power_BOL_multijunction, material_degradation_multijunction, lifetime);

power_output_GA = 252.895; % 18.5% * 1,367 W/m^2 (incident solar radiation)
power_BOL_GA = powerBeginningLife(power_output_GA, Id, theta);
power_EOL_GA = powerEndLife(power_BOL_GA, material_degradation_GA, lifetime);

silicon_area_direct_energy_transfer = psa_det / power_EOL_Si;
multijunction_area_direct_energy_transfer = psa_det / power_EOL_multijunction;
gallium_arsenide_area_direct_energy_transfer = psa_det / power_EOL_GA;

silicon_area_peak_power_tracking = psa_ppt / power_EOL_Si;
multijunction_area_peak_power_tracking = psa_ppt / power_EOL_multijunction;
gallium_arsenide_area_peak_power_tracking = psa_ppt / power_EOL_GA;

output_data(1) = silicon_area_direct_energy_transfer * 0.55; % 0.55 denisty of silicon cells
output_data(2) = multijunction_area_direct_energy_transfer * 0.85; % 0.85 denisty of multijunction cells
output_data(3) = gallium_arsenide_area_direct_energy_transfer * 0.85; % 0.85 denisty of ga-arsenide cells;
output_data(4) = silicon_area_peak_power_tracking * 0.55; % 0.55 denisty of silicon cells;
output_data(5) = multijunction_area_peak_power_tracking * 0.85; % 0.85 denisty of multijunction cells;
output_data(6) = gallium_arsenide_area_peak_power_tracking * 0.85; % 0.85 denisty of ga-arsenide cells;

% determines the beginning of life power production
% theta - Sun incidence angle between the vector normal to the surface in degrees
% output - power at beginning of life (W/m^2)
function power_BOL = powerBeginningLife(power_output, inherent_degradation, theta)
power_BOL = power_output * inherent_degradation * cos(theta);

% determines the end of life power production
% output - power at end of life (W/m^2)
function power_EOL = powerEndLife(power_BOL, material_degradation, lifetime) %lifetime in years
power_EOL = power_BOL * ( (1 - material_degradation) ^ lifetime );

function [battery_mass] = size_batteries(epn, periods)
NiH2_dod = 0.40; % Worst case from SMAD
NiCd_dod = 0.10; % Worst case from SMAD
N = 1;
n = 0.9;
% [periods] = calculatePeriods(altitude, inclination);
[periods] = [periods] / 60; % Convert seconds to minutes
NiH2_capacity = (epn * periods(1)) / (NiH2_dod * N * n);
NiCd_capacity = (epn * periods(1)) / (NiCd_dod * N * n);
NiH2_mass = NiH2_capacity / 35;
NiCd_mass = NiCd_capacity / 45;
battery_mass(1) = NiH2_mass;
battery_mass(2) = NiCd_mass;

function [periods] = calculatePeriods(radius, inclination)

```

```

stkinit;
remMachine = stkDefaultHost;
conid = stkOpen(remMachine); % Open the Connect to STK

% first check to see if a scenario is open
% if there is, close it
scen_open = stkValidScen;
if scen_open == 1
    stkUnload('/*')
end

cmd = 'New / Scenario maneuver_scenario'; % set up scenario
stkExec(conid, cmd);
cmd = 'New / */Satellite sat1'; % put the satellite in the scenario
stkExec(conid, cmd);

% set the scenario epoch
epochDate = "'28 Sep 2003 00:00:00.00'";
startDate = epochDate;
stopDate = "'2 Oct 2003 00:00:00.00'";
cmd = ['SetEpoch * ' epochDate];
stkExec(conid, cmd);
stkSyncEpoch;

% set the time period for the scenario
stkSetTimePeriod(startDate, stopDate, 'GREGUTC');

% set the animation parameters
rtn = stkConnect(conid,'Animate','Scenario/maneuver_scenario','SetValues "28 Sep 2003 00:00:00.0" 60 0.1');
rtn = stkConnect(conid,'Animate','Scenario/maneuver_scenario','Reset');

% set up initial state
% STK expects fields in meters NOT kilometers
incl = abs(inclination*180/pi);
cmd = ['SetState */Satellite/sat1 Classical J2Perturbation ' startDate ' ' stopDate ' 60 J2000 ' epochDate ' ' num2str(radius) ' 0 '
num2str(incl,'%2.4f') ' 0 0 0'];
stkExec(conid, cmd);

% get eclipse duration from STK
[secData, secNames] = stkReport('*/Satellite/sat1', 'Eclipse Times');
if (length(secData{1})==0)
    eclipse_duration = 0;

        % set eclipse duration in seconds
        eclipse_duration_average = 0;

        % get sunlight duration from STK
        [secData, secNames] = stkReport('*/Satellite/sat1', 'Sun');
        sunlight_duration = stkFindData(secData{1}, 'Duration');

        % return eclipse and sunlight periods in seconds
        periods(1) = 0;
        periods(2) = sunlight_duration;
    else
        eclipse_duration = stkFindData(secData{1}, 'Total Duration');

        % set eclipse duration in seconds
        eclipse_duration = unique(eclipse_duration);
        x = length(eclipse_duration);
        eclipse_duration = eclipse_duration(2 : (x-1));
        eclipse_duration_average = mean(eclipse_duration);

        % get sunlight duration from STK
        [secData, secNames] = stkReport('*/Satellite/sat1', 'Sun');
        sunlight_duration = stkFindData(secData{1}, 'Duration');

        % set sunlight duration in seconds
        y = length(sunlight_duration);
        sunlight_duration = sunlight_duration(2 : (y-1));
    end

```

```

sunlight_duration_average = mean(sunlight_duration);

% return eclipse and sunlight periods in seconds
periods(1) = eclipse_duration_average;
periods(2) = sunlight_duration_average;
end

stkClose(conid) % close out the stk connection
stkClose % this closes any default connection

function [P, dia_sat] = comm_sys(R, d, freq, dia_gnd)
% Input:
% - R, data rate, [bits/s]
% - d, distance from satellite to ground antenna [m]
% - freq, communication frequency, [Hz]
% - dia_gnd, ground antenna diameter [m]
% Output:
% - P, satellite communication required signal power [W]
% - dia_sat, satellite antenna diameter [m]
% Assumes:
% - ground antenna at 300 K
% - satellite antenna gain = 60
% - communication frequency < 20 GHz
SNR = 10; % Conservative Signal-to-Noise ratio, SMAD p. 551.
eff_gnd = 0.5; % Ground antenna efficiency
eff_sat = 0.5; % Satellite antenna efficiency
G_sat = 60; % Satellite antenna gain
L_a = 100; % Approximate atmosphere attenuation (rain, clouds, etc.)

N0 = 1.38e-23 * 300; % Noise Density, (Boltzmann's constant [J/K]) * (Ground Antenna Temp [K])
L = (4*pi*d*freq/3e8)^2; % Free space signal loss
dia_sat = sqrt(G_sat*3e8^2/(eff_sat*pi^2*freq^2)); % receiver diameter, [m]
G_gnd = eff_gnd * (pi*dia_gnd*freq/3e8)^2; % Ground antenna gain
P = (SNR * N0 * R * L * L_a) / (G_sat * G_gnd); % Satellite communication required signal power, [W]

```