Christopher Hynes
Kathryn Weiss
16.851 – Satellite Engineering
Due: Wednesday, October 1, 2003

Problem Set #2: Final Report

Subject: Orbit Maneuvering and Power

Motivation: Many of today's spacecraft use solar panels to power their various subsystems. There are a variety of factors that influence the size and type of solar panels to be used. Depending on the altitude and inclination of the spacecraft's orbit, the surface of the solar array may be eclipsed for some time. In addition, there are three types of solar cells (Gallium Arsenide, Multijunction and Silicon) that provide varying amounts of power. The surface area of a solar array must be adjusted so that the power requirements of the spacecraft's subsystems can be fulfilled given these constraints.

Problem Statement: What is the surface area of a solar panel needed to produce enough power to fulfill the power requirements of a given spacecraft? How does this surface area vary given the spacecraft's altitude and inclination? How does this surface area vary given the type of the solar cells, i.e. Gallium Arsenide, Multijunction or Silicon? How does this surface area vary given the type of power regulation, i.e. direct energy transfer or peak power tracking?

Approach: We will write a Matlab program to find solar array size needed to fulfill the power requirements for a spacecraft given a particular orbit. The program user will input the power needed during eclipse and daylight, the altitude and inclination of the spacecraft's orbit and the lifetime of the spacecraft. It is assumed that the orbit will be fixed throughout the lifetime of the spacecraft. Given the orbital parameters, the time the satellite is in eclipse and in daylight will be calculated using STK. The program will then determine the size of the solar array for three different solar cell types in the two power regulation systems. The results for the six combinations will be displayed in Matlab. The orbit and ground track will be displayed in STK.

Solution:

A. Requirements Specification

The SolarArraySize program shall allow the user to:

1. Choose their desired combination of power requirements and orbital elements,
2. Enter all parameters needed through an easy-to-use graphical user interface (GUI),
3. Display the results of the calculations numerically, and
4. Easily compare and contrast the results obtained from various solar cell materials.

1. Choose their desired combination of power requirements and orbital elements.

   To achieve this requirement, the user must input the following information:
   - Pe → Power needed during eclipse
   - Pd → Power needed during daylight
   - Orbit altitude
   - Orbit inclination
   - Mission duration

   The applicable formulas can be found in chapter 11.4 of SMAD. The eclipse and daylight periods were calculated using STK. These calculations are also documented in the comments of the code.

2. Enter all parameters needed through an easy-to-use graphical user interface (GUI).

   The input and output values of the SolarArraySize program are illustrated in the blackbox behavior diagram of Figure 1. More specifically, all calculations require power needed during eclipse and daylight (measured in Watts), orbit altitude (measured in kilometers), orbit inclination (measured in degrees) and mission duration (measured in years). The following outputs were calculated for both direct energy transfer and peak power tracking: total power needed (measured in watts) and surface area (measured in meters squared) for Gallium Arsenide, Multijunction and Silicon.

   The parameters are entered into Matlab through a GUI as shown in Figure 2. The output fields are dynamically updated to reflect the entries input by the users.

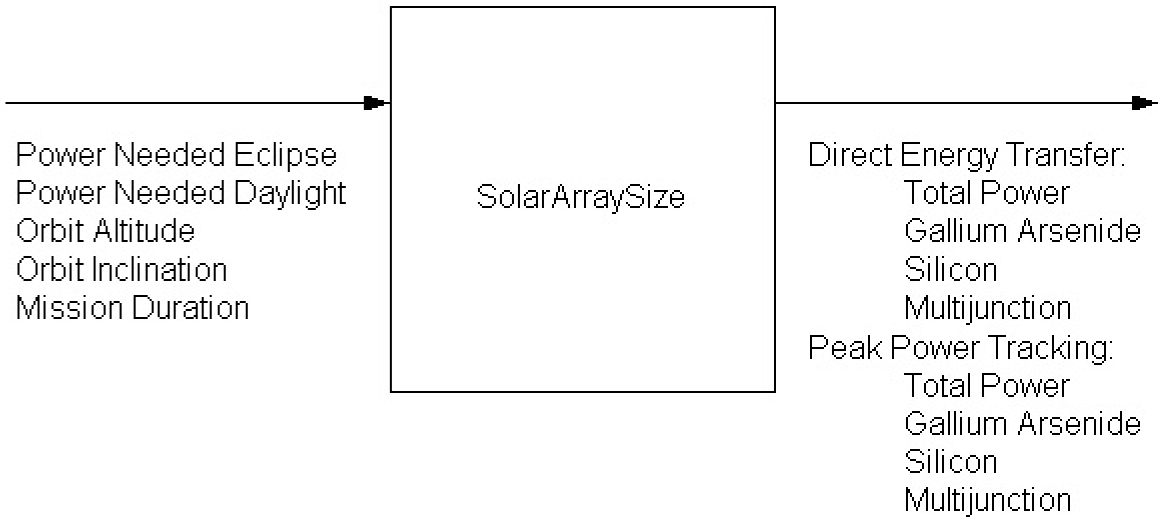
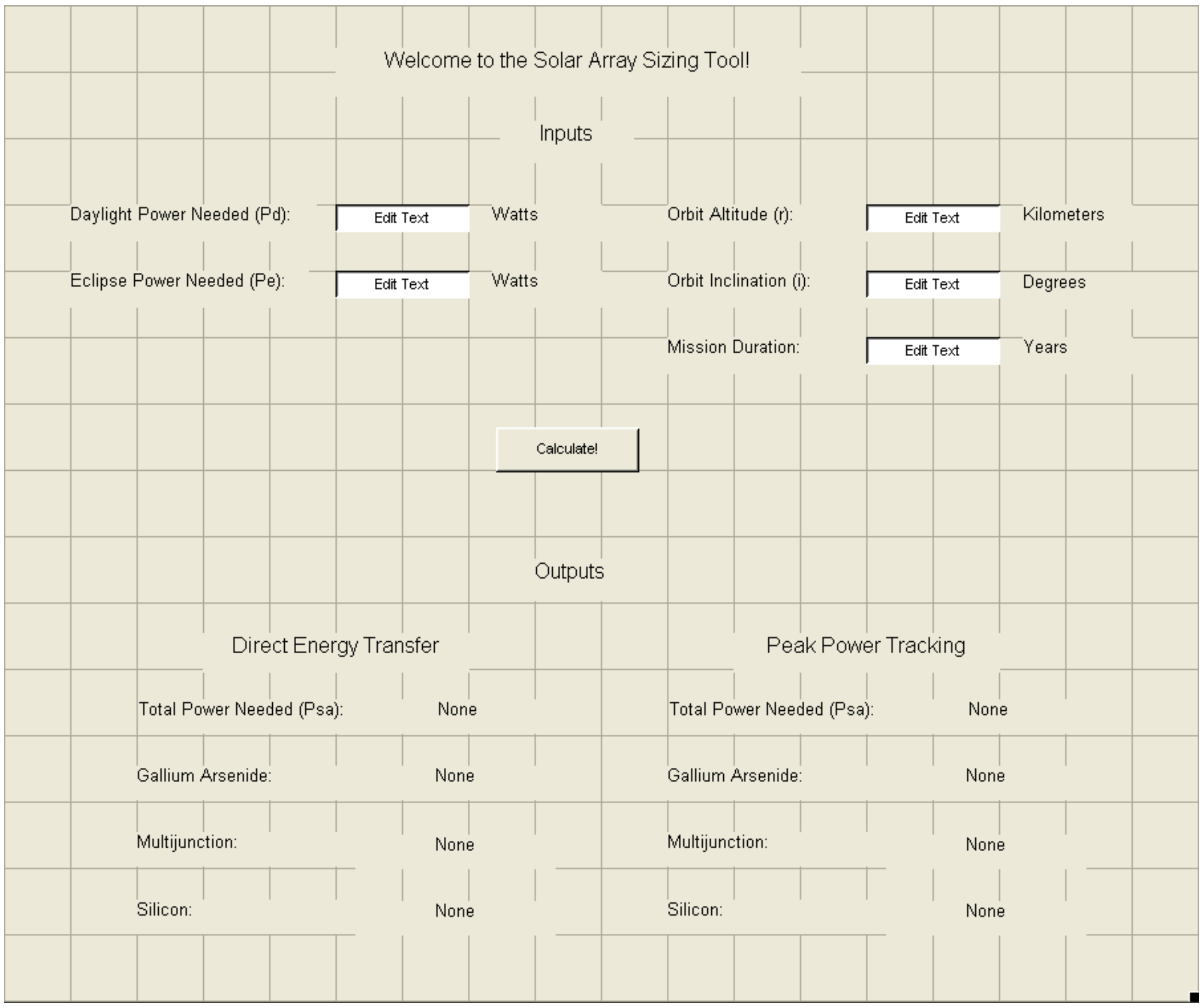


Figure 1. Blackbox Behavior Diagram


Welcome to the Solar Array Sizing Tool!

Inputs

Daylight Power Needed (Pd):	Edit Text	Watts	Orbit Altitude (r):	Edit Text	Kilometers
Eclipse Power Needed (Pe):	Edit Text	Watts	Orbit Inclination (i):	Edit Text	Degrees
			Mission Duration:	Edit Text	Years

Outputs

Direct Energy Transfer		Peak Power Tracking	
Total Power Needed (Psa):	None	Total Power Needed (Psa):	None
Gallium Arsenide:	None	Gallium Arsenide:	None
Multijunction:	None	Multijunction:	None
Silicon:	None	Silicon:	None

Figure 2. Graphical User Interface

3. Display the results of the calculations numerically.

   The numerical output is also displayed through the GUI after the SolarArraySize program calculates the total power needed and the sizes of the three solar cells for the two different power configurations. The orbit as well as the ground track is displayed in STK in a separate window. An example screen capture of the outputs to the user interface as well as the animation in STK are shown in Figure 3.

4. Easily compare and contrast the results obtained from various solar cell materials.

   The numerical outputs of the SolarArraySize can be compiled in tabular form (as done in Table 2) in order to perform trade-off analyses between the various materials.

B. Assumptions

a. Only circular orbits are handled by the SolarArraySize program.
b. Inherent degredation of the three materials is 0.77, the nominal value as stated on page 414 of SMAD.
c. The Sun incidence angle, theta, is set at a 23.5° angle, the worst-case Sun angle as stated on page 417 of SMAD.

C. Sample Test Runs

One example test run was executed using the data provided by SMAD for the FireSat satellite. Table 1 contains the information entered into the program through the GUI for test run.

Test	**Power Needed Eclipse (W)**	**Power Needed Daylight (W)**	**Orbit Altitude (km)**	**Orbit Inclination (deg)**	**Mission Duration (yrs)**
FireSat	110	110	700	0	5

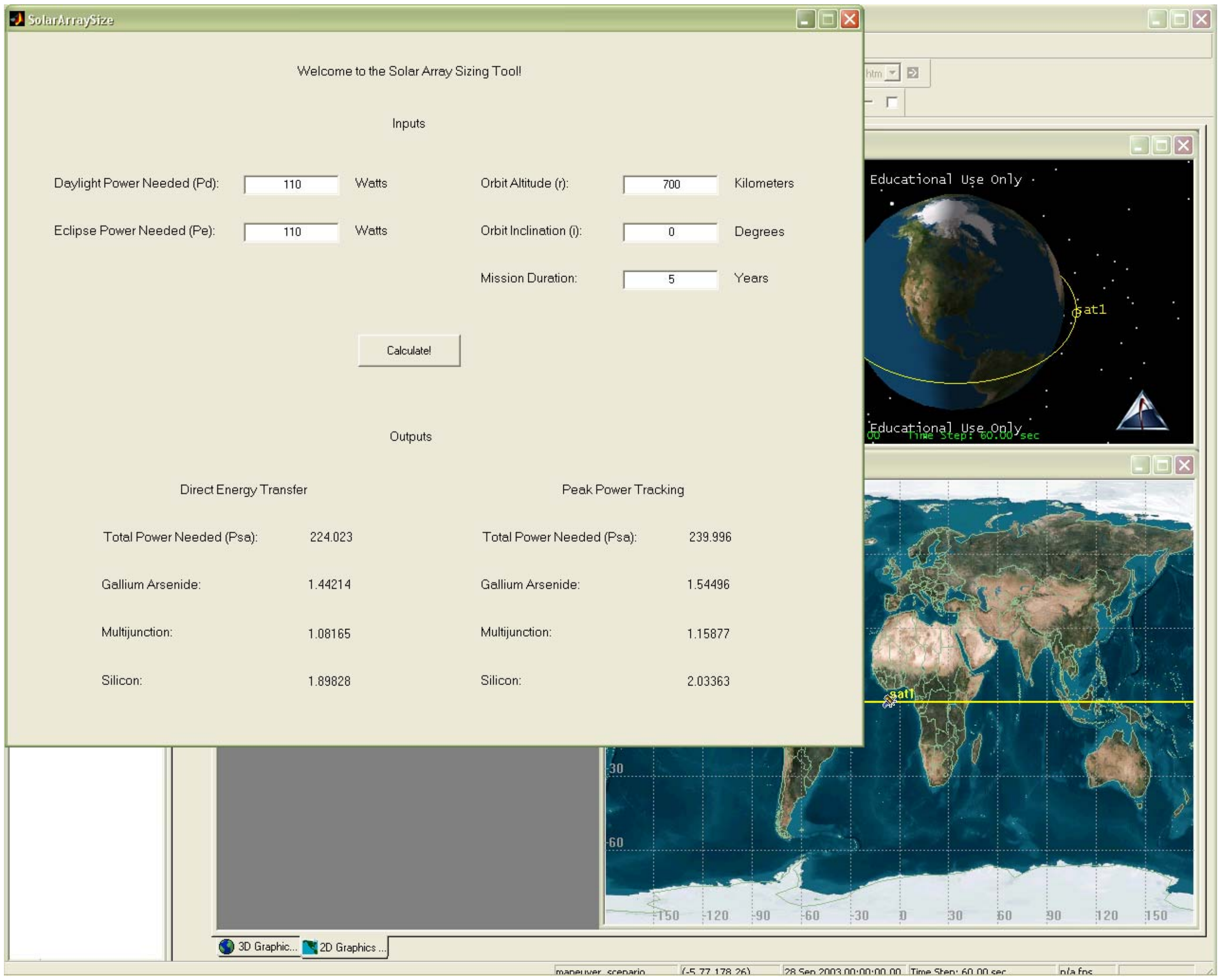
Table 1. Inputs for Example Test Run

The outputs can be compared with one another to determine which materials and/or power regulation configuration best suits the needs of the project. Figure 3 is a screen capture of the GUI and STK interface during the execution of the test run.

The results of the example test run are listed in Table 2. The results include the total power needed and the array sizes of the three types of material for both direct energy transfer and peak power tracking. As seen in Table 2, the Multijunction material provides the total power needed with the smallest amount of surface area. Next are the Gallium Arsenide and lastly the Silicon. The total power needed using direct energy transfer is nearly 15 Watts less than the peak power tracking configuration.

		Output Data
Direct Energy Transfer	**Total Power Needed**	224.023
	Gallium Arsenide	1.44214
	Multijunction	1.08165
	Silicon	1.89828
Peak Power Tracking	**Total Power Needed**	239.996
	Gallium Arsenide	1.54496
	Multijunction	1.15877
	Silicon	2.03363

Table 2. GUI Output from the Seven Test Runs



Welcome to the Solar Array Sizing Tool!

Inputs

Daylight Power Needed (Pd):	110	Watts	Orbit Altitude (r):	700	Kilometers
Eclipse Power Needed (Pe):	110	Watts	Orbit Inclination (i):	0	Degrees
			Mission Duration:	5	Years

Outputs

Direct Energy Transfer		Peak Power Tracking	
Total Power Needed (Psa):	224.023	Total Power Needed (Psa):	239.996
Gallium Arsenide:	1.44214	Gallium Arsenide:	1.54496
Multijunction:	1.08165	Multijunction:	1.15877
Silicon:	1.89828	Silicon:	2.03363

Figure 3. Screen Capture of SolarArraySize Program

D. User's Guide

a. Setup: To run the SolarArraySize program, first open both Matlab and STK. Note that STK Version 5 must be used to run the program. To initialize Matlab, the agiInit routine should be run to setup the appropriate paths, and the default port ('5001') should be accepted along with default hostname 'localhost'. Before running SolarArraySize, you must type 'stkinit' at the Matlab prompt to set up the connection between Matlab and STK software packages.

Before running the program, make sure that the following three files are all in the Matlab working directory:

SolarArraySize.m
SolarArraySize.fig
SolarArraySize.asv.

b. Running the Program: The SolarArraySize is run from the Matlab command prompt by entering the command *SolarArraySize.* This opens the GUI window, and you will have access to the solar array sizing program as shown in Figure 2.

c. Entering Parameters: Start by entering the requirements for your power system and your desired orbit. The required initial data for all solar array sizing includes: power needed during daylight in Watts, power needed during eclipse in Watts, the orbit altitude in kilometers, the orbit inclination angle in degrees and the mission duration in years. Enter the desired values for these parameters in the text boxes in place of the label "Edit Text".

d. Calculating Results: When all the necessary parameters have been entered, click the Calculate! button to calculate the total power needed and the solar array sizes of the three materials for the two power configurations. The program will perform the necessary calculations to output the orbit to STK for graphical display and the numerical data to the GUI. A short waiting time is required.

e. Displaying Results: The output parameters are displayed in the output window for the two power configurations as follows: Total Power Needed, Gallium Arsenide Size, Multijunction Size and Silicon Size. For sample output from both STK and the Solar Array Sizing Environment (Matlab window), see Figure 3.

E. Code

```
function varargout = SolarArraySize(varargin)
% Begin initialization code - DO NOT EDIT
   gui_Singleton = 1;
   gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @SolarArraySize_OpeningFcn, ...
                  'gui_OutputFcn',  @SolarArraySize_OutputFcn, ...
                  'gui_LayoutFcn',  [] , ...
                  'gui_Callback',   []);
   if nargin & isstr(varargin{1})
      gui_State.gui_Callback = str2func(varargin{1});
   end

   if nargout
      [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
   else
      gui_mainfcn(gui_State, varargin{:});
   end
% End initialization code - DO NOT EDIT

% Executes just before SolarArraySize is made visible.
function SolarArraySize_OpeningFcn(hObject, eventdata, handles, varargin)
   handles.output = hObject;
   guidata(hObject, handles);

   clear global daylight_power_needed;
```

```
    clear global eclipse_power_needed;
    clear global altitude;
    clear global inclination;
    clear global mission_lifetime;

    global daylight_power_needed;
    global eclipse_power_needed;
    global altitude;
    global inclination;
    global mission_lifetime;

% Outputs from this function are returned to the command line.
function varargout = SolarArraySize_OutputFcn(hObject, eventdata, handles)
    varargout{1} = handles.output;

% Executes on button press in calculate_button.
function calculate_button_Callback(hObject, eventdata, handles)
    global daylight_power_needed;
    global eclipse_power_needed;
    global altitude;
    global inclination;
    global mission_lifetime;

    daylight_power_needed = str2num(daylight_power_needed{1,1});
    eclipse_power_needed = str2num(eclipse_power_needed{1,1});
    altitude = str2num(altitude{1,1});
    inclination = str2num(inclination{1,1});
    mission_lifetime = str2num(mission_lifetime{1,1});

    [output_data] = calculatePSA(daylight_power_needed, eclipse_power_needed, altitude,
inclination, mission_lifetime);

    set(handles.total_power_needed_direct_energy_transfer_output, 'String', output_data(1));
    set(handles.silicon_direct_energy_transfer_output, 'String', output_data(2));
    set(handles.multijunction_direct_energy_transfer_output, 'String', output_data(3));
    set(handles.gallium_arsenide_direct_energy_transfer_output, 'String', output_data(4));
    set(handles.total_power_needed_peak_power_tracking_output, 'String', output_data(5));
    set(handles.silicon_peak_power_tracking_output, 'String', output_data(6));
    set(handles.multijunction_peak_power_tracking_output, 'String', output_data(7));
    set(handles.gallium_arsenide_peak_power_tracking_output, 'String', output_data(8));

% Executes during object creation, after setting all properties.
function daylight_power_needed_inputbox_CreateFcn(hObject, eventdata, handles)
    if ispc
        set(hObject,'BackgroundColor','white');
    else
```

```
        set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
    end

function daylight_power_needed_inputbox_Callback(hObject, eventdata, handles)
    global daylight_power_needed;
    daylight_power_needed = get(handles.daylight_power_needed_inputbox, 'String');

% Executes during object creation, after setting all properties.
function mission_duration_inputbox_CreateFcn(hObject, eventdata, handles)
    if ispc
        set(hObject,'BackgroundColor','white');
    else
        set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
    end

function mission_duration_inputbox_Callback(hObject, eventdata, handles)
    global mission_lifetime;
    mission_lifetime = get(handles.mission_duration_inputbox, 'string');

% Executes during object creation, after setting all properties.
function eclipse_power_needed_inputbox_CreateFcn(hObject, eventdata, handles)
    if ispc
        set(hObject,'BackgroundColor','white');
    else
        set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
    end

function eclipse_power_needed_inputbox_Callback(hObject, eventdata, handles)
    global eclipse_power_needed;
    eclipse_power_needed = get(handles.eclipse_power_needed_inputbox, 'String');

% Executes during object creation, after setting all properties.
function inclination_inputbox_CreateFcn(hObject, eventdata, handles)
    if ispc
        set(hObject,'BackgroundColor','white');
    else
        set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
    end

function inclination_inputbox_Callback(hObject, eventdata, handles)
    global inclination;
    inclination = get(handles.inclination_inputbox, 'String');

% Executes during object creation, after setting all properties.
function altitude_inputbox_CreateFcn(hObject, eventdata, handles)
    if ispc
```

```
        set(hObject,'BackgroundColor','white');
    else
        set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
    end

function altitude_inputbox_Callback(hObject, eventdata, handles)
    global altitude;
    altitude = get(handles.altitude_inputbox, 'String');

function [output_data] = calculatePSA(daylight_power_needed, eclipse_power_needed, altitude,
inclination, mission_lifetime)
    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 degredation
    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



    output_data(1) = ((eclipse_power_needed * periods(1)) / Xe_direct_energy_transfer +
(daylight_power_needed * periods(2)) / Xd_direct_energy_transfer) / periods(2);
    output_data(5) = ((eclipse_power_needed * periods(1)) / Xe_peak_power_tracking +
(daylight_power_needed * 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, mission_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, mission_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,
mission_lifetime);
```

```
    silicon_area_direct_energy_transfer = output_data(1) / power_EOL_Si;
    multijunction_area_direct_energy_transfer = output_data(1) / power_EOL_multijunction;
    gallium_arsenide_area_direct_energy_transfer = output_data(1) / power_EOL_GA;

    silicon_area_peak_power_tracking = output_data(5) / power_EOL_Si;
    multijunction_area_peak_power_tracking = output_data(5) / power_EOL_multijunction;
    gallium_arsenide_area_peak_power_tracking = output_data(5) / power_EOL_GA;

    output_data(2) = silicon_area_direct_energy_transfer;
    output_data(3) = multijunction_area_direct_energy_transfer;
    output_data(4) = gallium_arsenide_area_direct_energy_transfer;
    output_data(6) = silicon_area_peak_power_tracking;
    output_data(7) = multijunction_area_peak_power_tracking;
    output_data(8) = gallium_arsenide_area_peak_power_tracking;

% 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 [periods] = calculatePeriods(altitude, inclination)
    stkinit;

    remMachine = stkDefaultHost;

    % Open the Connect 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 maneuver_scenario';
    stkExec(conid, cmd);
    % put the satellite in the scenario
    cmd = 'New / */Satellite sat1';
    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
    cmd = ['SetState */Satellite/sat1 Classical J2Perturbation ' startDate ' ' stopDate ' 60 J2000 '
epochDate ' ' num2str(altitude*1000+6378000) ' 0 ' num2str(inclination) ' 0 0 0'];
    stkExec(conid, cmd);

    % get eclipse duration from STK
    [secData, secNames] = stkReport('*/Satellite/sat1', 'Eclipse Times')
    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 : (x-1));
    sunlight_duration_average = mean(sunlight_duration);
```

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

% close out the stk connection
stkClose(conid)

% this closes any default connection
stkClose
```