MATLAB$^{®}$ GUI Visualization of Classical Orbital Elements

A Senior Project

presented to

the Faculty of the Aerospace Engineering Department

California Polytechnic State University, San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Bachelor of Science

by

Nancy Teresa Cabrera

June, 2010

MATLAB® GUI Visualization of Classical Orbital Elements

Nancy Teresa Cabrera[1]
California Polytechnic State University, San Luis Obispo, CA, 93407

The classical orbital elements of an orbit are eccentricity, angular momentum, inclination, right ascension of ascending node, true anomaly, and argument of perigee. These six parameters define an orbit. Using MATLAB® to model a satellite orbiting Earth in three dimensions, a graphical user interface was created to allow a user to manipulate the orbital elements to desired quantities. In doing so, each parameter's impact on the orbit is visually displayed. This furthers the understanding of how the parameters are linked to the orbit. When the interface is first opened, the default circular orbit has a range of 20,000 kilometers, an angular momentum of about 133,929 km^2/s, an inclination of 45°, a true anomaly of 30°, and right ascension of ascending node and argument of perigee of 0°. This default orbit can easily by changed by sliders and input boxes on the interface. The user-friendly interface allows for anyone to better understand an orbit and its parameters.

Nomenclature

COEs	=	classical orbital elements
e	=	eccentricity
GUI	=	graphical user interface
h	=	angular momentum, km^2/s
i	=	inclination, °
$[Q]_{\bar{x}X}$	=	transformation matrix from perifocal frame to geocentric equatorial fame
r	=	range, km
$\vec{r}$	=	position vector, km
t	=	time, s
$\vec{v}$	=	velocity vecor, km/s
Ω	=	right ascension of ascending node, °
θ	=	true anomaly, °
μ	=	Gravitational parameter of Earth, 398600 km^3/s^2
ω	=	argument of perigee, °

I. Introduction

A MATLAB® graphical user interface (GUI) will be created to simulate a satellite orbiting the Earth. This 2-body system will be variable, in that the user will have the opportunity to change the classical orbital elements (COEs) of the satellite's orbit. Since the orbit will be displayed in a 3D plot, the user will see the effects of altering the COEs. This will allow for a deeper understanding of the COEs and their impact on an orbit.

The COEs being investigated are: inclination (i), argument of perigee (ω), eccentricity (e), right ascension of the ascending node (Ω), true anomaly (θ) and angular momentum (h). The GUI will have i, θ, ω, and Ω in units of degrees, and h will be in km^2/s; eccentricity is unit less. These values can be changed by the user. Depending on the eccentricity, the orbit will be circular or elliptical. The GUI will call functions that calculate the COEs; therefore

[1] Undergraduate Student, Aerospace Engineering Department, 1 Grand Avenue

changing the orbit. The functions will be built using equations mainly from Orbital Mechanics for Engineering Students[1] by Howard Curtis.

Mathematics is being implemented by the use of these equations. These COEs are the perfect example of science and mathematics complimenting each other. Science is the key to space research. The engineering aspect comes in by using MATLAB® to model it visually. The design of the MATLAB® GUI will allow the user to easily change the COEs and observe the changes to the orbit. This facilitates for interpretations to be made by allowing the user to analyze the importance of the COEs. This GUI can be used, by students studying orbits for the first time, because these fundamentals are essential for core understanding of orbits. With this visualization, students can truly understand COEs and it becomes much easier to be explained by the use of a model. The GUI will visually demonstrate what the COEs really are.

A. Objective

The objective of this senior project is to create a GUI in MATLAB®, which will display a satellite orbiting Earth in three dimensions. The GUI will have slider bars and input boxes which will allow the user to change the COEs. The GUI when first opened will have a default orbit with its classical orbital elements already selected. The COEs can easily be changed using the sliders and input boxes. Users will be able to grasp a deeper understanding of how the COEs influence an orbit.

II. Apparatus and Procedure

A. Apparatus

The program MATLAB® version 7.4.0.287 (R2007a) was used for the creation functions and GUI. MATLAB® is “The Language of Technical Computing”.

B. Procedure

Various functions needed to be complied, before the GUI was created. The first code created was a code that would yield the position and velocity vectors by providing h, i, Ω, e, ω, and θ. Once this code was made, a second function was made that would yield the next position and velocity vectors in the orbit, given an initial position and velocity vector. This function was made so it could be fed into ode45. Ode45 allowed for the position and velocity vectors to be given over a certain amount of time. To plot the orbit, a script was written. The script allowed for the orbit to be plotted in three dimensions using the previous function and ode45. For the script, an initial simple default orbit was chosen.

Once the groundwork for the GUI was complete, the GUI could now be constructed. GUIDE was used to facilitate the creation of the GUI. The title of “Visualization of Classical Orbital Elements” was placed at the top, centered. The 3D plot of the orbit was placed on the left hand side of the GUI, under the title. To change the value of the i and θ, slider bars were implemented. Four input boxes were implemented for h, e, Ω, and ω. These sliders and input boxes were all labeled on top with static text boxes, whose text cannot be altered. Finally, an “Update Plot” push button was placed to allow the orbit to be regenerated once the user has inputted new quantities for the COEs.

III. Analysis

The following section will discuss the equations used in the analysis. The first equation used was the orbital equation which combines the r, h, e, and μ into one equation.

$$r = \frac{h^2}{\mu}\frac{1}{1+ecos(\theta)} \tag{1}$$

Where r is the radial distance from the center of the Earth, and μ is the gravitational constant. This equation was used to calculate the corresponding angular momentum for the given r value. To find the position and velocity vectors in the perifocal frame, equations 2 and 3 were used. The perifocal frame, also known as the “natural frame”, was employed because its $\bar{x}\bar{y}$ plane is the plane of the orbit. This frame is centered on its focus, in this case Earth.

$$\vec{r} = \frac{h^2}{\mu}\frac{1}{1+ecos\theta}\begin{Bmatrix} cos\theta \\ sin\theta \\ 0 \end{Bmatrix} \tag{2}$$

$$\vec{v} = \frac{\mu}{h}\begin{Bmatrix} -sin\theta \\ e + cos\theta \\ 0 \end{Bmatrix} \tag{3}$$

These vectors were then individually multiplied by the transformation matrix, equation 4, to convert them into the geocentric equatorial plane. This was how the position and velocity vectors were calculated from the COEs.

$$[Q]_{\bar{x}X} = \begin{bmatrix} \cos\Omega\cos\omega - \sin\Omega\sin\omega\cos i & -\cos\Omega\sin\omega - \sin\Omega\cos i\cos\omega & \sin\Omega\sin i \\ \sin\Omega\cos\omega + \cos\Omega\cos i\sin\omega & -\sin\Omega\sin\omega + \cos\Omega\cos i\cos\omega & -\cos\Omega\sin i \\ \sin i\sin & \sin i\cos\omega & \cos i \end{bmatrix} \tag{4}$$

Next, the acceleration equation, shown below, was used to find the next position and velocity vectors after a certain amount of time.

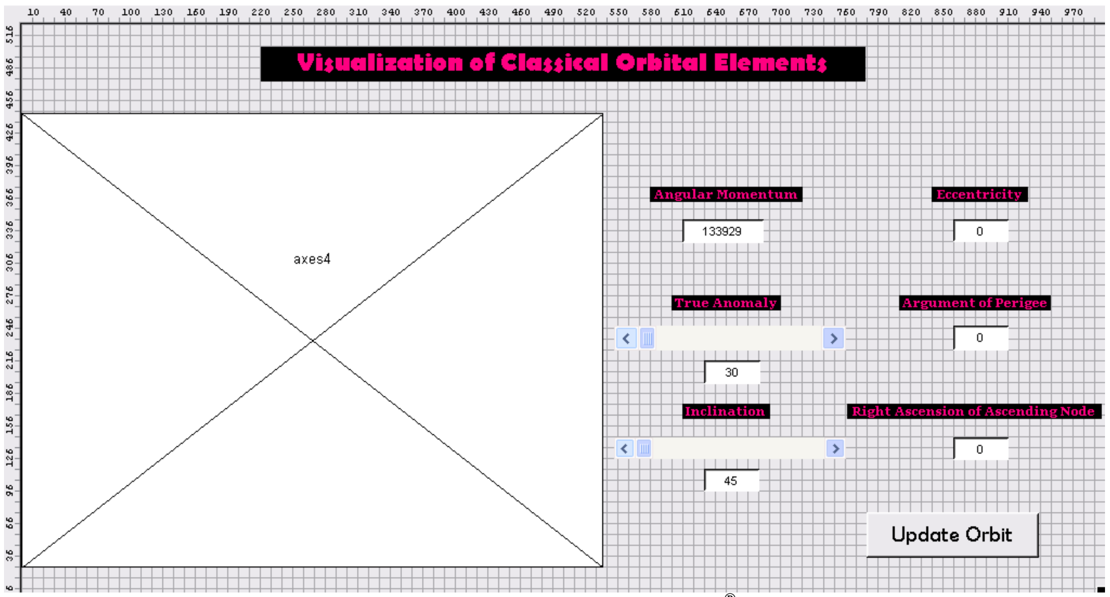
$$\ddot{\vec{r}} = \frac{-\mu}{r^3}\vec{r} \tag{5}$$

This was done using the ode45 function embedded in MATLAB®, with its time span set to the period of the orbit. Therefore the simulation will plot one orbit. With ode45 yielding the position and respective velocity vectors, the orbit was plot in 3D using the function plot3.

IV. Results and Discussion

It was very important before anything with the GUI was done, to make sure all the required functions and scripts were perfectly working. First the default orbit's COEs where chosen. It was meant to be as simple as possible, so the orbit had COEs of: h = 133,929 km^2/s, i = 45°, θ = 30°, Ω = 0°, ω = 0° and e = 0.

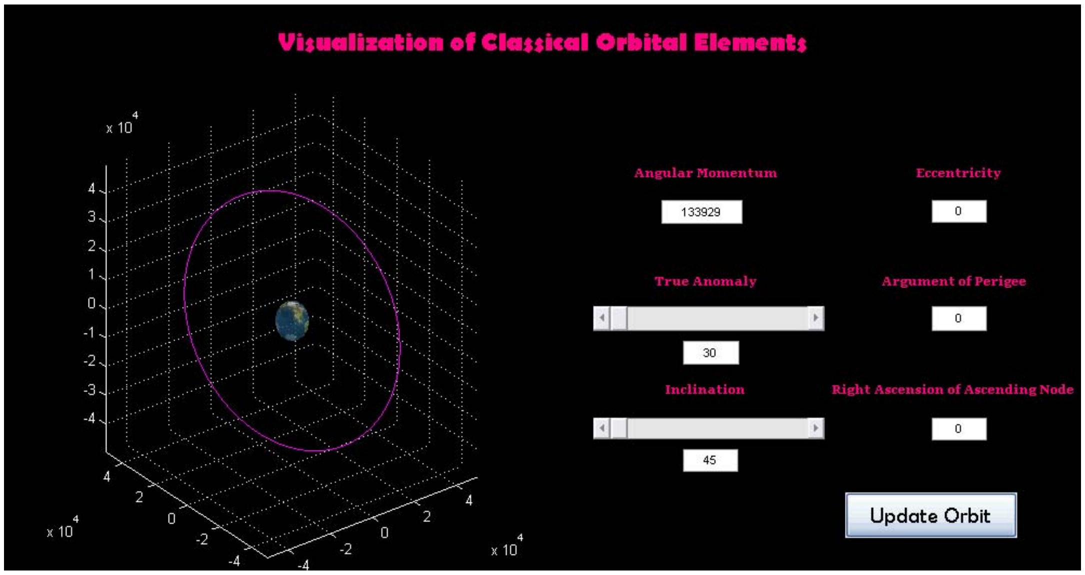
Once the sub-functions are created, the framework for the GUI was done; this is shown in Fig. 1.



Angular Momentum 133929
Eccentricity 0
True Anomaly 30
Argument of Perigee 0
Inclination 45
Right Ascension of Ascending Node 0

Figure 1. This is the configuration of the MATLAB® GUI in GUIDE.

The default simulation, which is made using the default COEs, has been incorporated into the GUI. In Fig. 2, the corresponding orbit is shown.

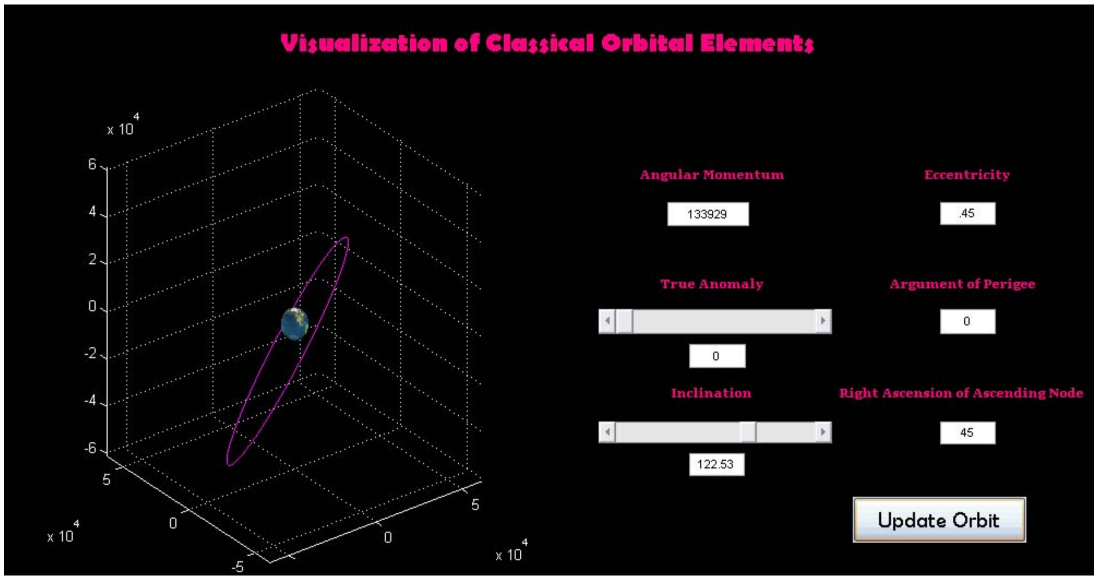


Angular Momentum: 133929
Eccentricity: 0
True Anomaly: 30
Argument of Perigee: 0
Inclination: 45
Right Ascension of Ascending Node: 0

Figure 2. This is the GUI display window with the default orbit.

The slider bars for θ and i have minimum and maximum values integrated into them. As a result, the user can change the value of the θ and i using either the text box or the slider.

The final step in the GUI construction was to have the push button "Update Orbit" functioning properly. The first step was to call the respective values from each text box of the COEs. After calling these values, the new orbit can be plotted using the same code which was used to plot the default orbit. A new orbit is shown in Fig. 3.



Angular Momentum: 133929
Eccentricity: .45
True Anomaly: 0
Argument of Perigee: 0
Inclination: 122.53
Right Ascension of Ascending Node: 45

Figure 3. This is a new orbit with a different e, i, θ and Ω displayed.

V. Conclusion

As a result of the functions and scripts, the GUI framework has been done. One of the functions made was to calculate the position and velocity vector from the h, i, Ω, e, ω, and θ. The second function provided the next position and velocity vector, by using the initial vectors. The time step between the initial and next vector was determined by ode45. This function was fed into ode45 and gives the orbit's vector for the duration of two days.

There were a couple problems. The tolerance for ode45 had to be fixed to 1 x 10^{-8}, to allow for a refined orbit. The time span was set to the period of one orbit to minimize the time it took to run ode45. There was a problem with the initial h of the default orbit as well. To fix this problem, the h was calculated using equation 1, and the problem was solved. After implementing the default code, the plot became 3D; it was 2D prior to this.

The push button "Update Orbit" then had to be activated. This required the values of the COEs to be called and then use them to plot the new orbit. The method to plot this new orbit was the same as for the default orbit. The user is now able to choose their COEs and have the relevant orbit illustrated.

Space is something we know a great deal about. Yet there is still so much more to be learned. This project will study one small, yet vital, aspect of space. From tracking objects and planets, COEs also allow for more efficient space travel. Once these COEs are fully understood, the possibilities are endless.

Reference

[1]Curtis, H. D., *Orbital Mechanics for Engineering Students*, Elsevier Ltd, Massachusetts, 2005, Chaps. 2 - 4.

[2]Attaway, S., *MATLAB: A Practical Introduction to Programming and Problem Solving*, Elsevier, Inc., Massachusetts, 2009, Chaps. 13.

[3]Marchand, P and Holland, O. T., *Graphics and GUIs with MATLAB*, 3rd edition, Chapman & Hall/CRC, Florida, 2003.

[4]Smith, S. T., *MATLAB Advanced GUI Development*, Dog Ear, Indiana, 2006, Chaps. 3.

Appendix

MATLAB® Code:

Getting position and velocity from COE's:

```
function [r,v] = coes_RV(h,inc,RA,e,omega,theta)

%From the orital elements, the position and velocity vectors are is found.
%The Inputs are inputed the following matter:
%     h is angular momentum in km^2/s
%     inc is inclination in radians
%     RA is right ascension of ascending node in radians
%     e is eccelntricity (unitless)
%     omega is argument of perigee in radians
%     theta is true anomaly in radians
%Equations (EQN) used come from Curtis



mu = 398600; %Gravitational parameter of Earth (km^3/s^2)

%Now following Algorithm 4.2 from Curtis, I use EQN. 4.37 to find r in
%perifocal frame, and EQN 4.38 to find v in perifocal plane

rperi = h^2/mu/(1+e*cos(theta))*[cos(theta); sin(theta); 0]; % (km)

vperi = mu/h.*[-sin(theta); e + cos(theta); 0]; % (km/s)

%Next use EQN 4.44 to find transformation matrix from perifocl to
%geocentric equatorial coordinates

Qperi=[cos(RA)*cos(omega)-sin(RA)*sin(omega)*cos(inc),    -cos(RA)*...
    sin(omega)-sin(RA)*cos(inc)*cos(omega),    sin(RA)*sin(inc);
    sin(RA)*cos(omega)+cos(RA)*cos(inc)*sin(omega),    -sin(RA)*...
    sin(omega)+cos(RA)*cos(inc)*cos(omega),    -cos(RA)*sin(inc);
    sin(inc)*sin(omega), sin(inc)*cos(omega), cos(inc)];

%Finally use, EQN 4.46 to find the r and V in the geocentric equatorial
```

```
%plane

r=Qperi*rperi; %in km

v=Qperi*vperi; %in km/s

%Convert r and v into row vectors

r = r'; %km

v = v'; %km/s
```

Function to feed into Ode45 to get new position and velocity:

```
function [accel] = orbit(t,y)

%y is the initial conditions, and time is in seconds

%Set constant values
mu = 398600; %Gravitational parameter of Earth (km^3/s^2)

%Pull out the initial conditions to components
rx=y(1); %km
ry=y(2); %km
rz=y(3); %km
vx=y(4); %km/s
vy=y(5); %km/s
vz=y(6); %km/s

%Normalize the position vector for futre use
R=norm([rx, ry, rz]);

%Find acceleration from the position vector
ax=-mu*rx/R^3; %km/s^2
ay=-mu*ry/R^3; %km/s^2
az=-mu*rz/R^3; %km/s^2

%Set up new conditions after t seconds
accel = [vx; vy; vz; ax; ay; az];
```

Function to plot Orbit:

```
%Senior Project

close all
clear all
clc

%First state initial orbital elements in order to find initial r and v in
%geocentric equatorial coordinates
```

```
%Set constants
Re = 6378; %Radius of Earth (km)
mu = 398600; %Gravitational Parameter of Earth (km^3/s^2)

%Angular Momentum
h0 = 133929.0857; %km^2/s

%Inclination
inc0 = 45*pi/180; %radians

%Right Ascension of Ascending Node
RA0 = 0; %radians

%Eccentricity
e0 = 0;

%Argument of Perigee
omega0 = 0*pi/180; %radians

%True Anomaly
theta0 = 30*pi/180; %radians

%Now find r in km and v in km/s
[r0,v0] = coes_RV(h0,inc0,RA0,e0,omega0,theta0);

%Find Range
range0 = norm(r0) - Re; %in km

%Find Period of orbit
T0 = 2*pi/sqrt(mu)*(norm(r0))^(1.5); %in seconds

%Now these are the initial conditions and time span
y0 = [r0 v0];
t0 = [0 172800]; %in seconds
% t0 = [0 T0]; %in seconds

%Use ode45 to get orbit
[t,y] = ode45('orbit', t0, y0);

% %Plot Orbit
% plot3(y(:,1), y(:,2), y(:,3))

% %Simulation
% for i = 1:length(t)
%     comet3(y(1:i,1), y(1:i,2), y(1:i,3))
%     drawnow
% end

%Simulation
for i = 1:length(t)
   plot3(y(1:i,1), y(1:i,2), y(1:i,3))
   drawnow
end
```

GUI function:

```
function varargout = COESorbit(varargin)
% COESORBIT M-file for COESorbit.fig
%      COESORBIT, by itself, creates a new COESORBIT or raises the existing
%      singleton*.
%
%      H = COESORBIT returns the handle to a new COESORBIT or the handle to
%      the existing singleton*.
%
%      COESORBIT('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in COESORBIT.M with the given input arguments.
%
%      COESORBIT('Property','Value',...) creates a new COESORBIT or raises
the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before COESorbit_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to COESorbit_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help COESorbit

% Last Modified by GUIDE v2.5 25-May-2010 20:46:39

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @COESorbit_OpeningFcn, ...
                   'gui_OutputFcn',  @COESorbit_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(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 COESorbit is made visible.
function COESorbit_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to COESorbit (see VARARGIN)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%Insert my DEFAULT in this section here
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%First state initial orbital elements in order to find initial r and v in
%geocentric equatorial coordinates

%Set handles.constants
handles.Re = 6378; %Radius of Earth (km)
handles.mu = 398600; %Gravitational Parameter of Earth (km^3/s^2)

%Angular Momentum
handles.h0 = 133929.0857; %km^2/s

%Inclination
handles.inc0 = 45*pi/180; %radians

%Right Ascension of Ascending Node
handles.RA0 = 0; %radians

%Eccentricity
handles.e0 = 0;

%Argument of Perigee
handles.omega0 = 0*pi/180; %radians

%True Anomaly
handles.theta0 = 30*pi/180; %radians

%Now find r in km and v in km/s
[handles.r0,handles.v0] =
coes_RV(handles.h0,handles.inc0,handles.RA0,handles.e0,handles.omega0,handles
.theta0);

%Find Range
handles.range0 = norm(handles.r0) - handles.Re; %in km

%Magnitude of radius
handles.r0norm=norm(handles.r0); % in km

%Find Period of orbit
handles.T0 = 2*pi/sqrt(handles.mu)*(norm(handles.r0))^(1.5); %in seconds

%Now these are the initial conditions and time span
handles.y0 = [handles.r0 handles.v0];
handles.t0 = [0 handles.T0]; %in seconds
% t0 = [0 T0]; %in seconds

%Use ode45 to get orbit
```

```
%Set tolerance
options = odeset('RelTol', 1e-6);

[handles.t,handles.y] = ode45('orbit', handles.t0, handles.y0, options);

% Plot Earth
[handles.Rex,handles.Rey,handles.Rez] = sphere();   %Sphere coordinates for
Earth
handles.earth = surf(handles.Re*handles.Rex, handles.Re*handles.Rey,
handles.Re*handles.Rez, 'edgeColor','none', 'faceColor', [.5 .9
.5],'faceAlpha', 0.55);

I = imread('earth_map1.jpg');
axis vis3d
set(handles.earth,'CData', I,'FaceColor','texturemap');
hold all
set(gcf,'color','black'); %[.85,.7,1]

set(gca,'Color','black','XColor','white', ...
    'YColor','white','ZColor','white')
% set(gca,'Color','w');

%Simulation
for i = 1:length(handles.t)
    orbit = plot3(handles.y(1:i,1), handles.y(1:i,2), handles.y(1:i,3),
'color', 'magenta');
    axis([-handles.r0norm-5000 handles.r0norm+5000 -handles.r0norm-5000
handles.r0norm+5000 -handles.r0norm-5000 handles.r0norm+5000])
    drawnow
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%END DEFAULT CODE%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Choose default command line output for COESorbit
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes COESorbit wait for user response (see UIRESUME)
% uiwait(handles.figure1);




% --- Outputs from this function are returned to the command line.
function varargout = COESorbit_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



% % --- Executes on button press in pushbutton1.
% function pushbutton1_Callback(hObject, eventdata, handles)
% % hObject    handle to pushbutton1 (see GCBO)
% % eventdata  reserved - to be defined in a future version of MATLAB
% % handles    structure with handles and user data (see GUIDATA)




function AngMomentum_Callback(hObject, eventdata, handles)
% hObject    handle to AngMomentum (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of AngMomentum as text
%        str2double(get(hObject,'String')) returns contents of AngMomentum as
a double


% --- Executes during object creation, after setting all properties.
function AngMomentum_CreateFcn(hObject, eventdata, handles)
% hObject    handle to AngMomentum (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end




function eccentricity_Callback(hObject, eventdata, handles)
% hObject    handle to eccentricity (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of eccentricity as text
%        str2double(get(hObject,'String')) returns contents of eccentricity
as a double


% --- Executes during object creation, after setting all properties.
function eccentricity_CreateFcn(hObject, eventdata, handles)
% hObject    handle to eccentricity (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function Arg_Perigee_Callback(hObject, eventdata, handles)
% hObject    handle to Arg_Perigee (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Arg_Perigee as text
%        str2double(get(hObject,'String')) returns contents of Arg_Perigee as
a double


% --- Executes during object creation, after setting all properties.
function Arg_Perigee_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Arg_Perigee (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function RAAN_Callback(hObject, eventdata, handles)
% hObject    handle to RAAN (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of RAAN as text
%        str2double(get(hObject,'String')) returns contents of RAAN as a
double


% --- Executes during object creation, after setting all properties.
function RAAN_CreateFcn(hObject, eventdata, handles)
% hObject    handle to RAAN (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes on slider movement.
function TrueAnomaly_Callback(hObject, eventdata, handles)
% hObject    handle to TrueAnomaly (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of
slider

TA1 = get(handles.TrueAnomaly, 'Value');

set(handles.TAedit, 'string', TA1);



% --- Executes during object creation, after setting all properties.
function TrueAnomaly_CreateFcn(hObject, eventdata, handles)
% hObject    handle to TrueAnomaly (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



% --- Executes on slider movement.
function Inclination_Callback(hObject, eventdata, handles)
% hObject    handle to Inclination (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of
slider

i1 = get(handles.Inclination, 'Value');

set(handles.INCedit, 'string', i1);

% --- Executes during object creation, after setting all properties.
function Inclination_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Inclination (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes during object creation, after setting all properties.
function text1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to text1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called





function TAedit_Callback(hObject, eventdata, handles)
% hObject    handle to TAedit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of TAedit as text
%        str2double(get(hObject,'String')) returns contents of TAedit as a
double

TA = str2num(get(handles.TAedit, 'string'));

set(handles.TrueAnomaly, 'value', TA);

% --- Executes during object creation, after setting all properties.
function TAedit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to TAedit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function INCedit_Callback(hObject, eventdata, handles)
% hObject    handle to INCedit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of INCedit as text
%        str2double(get(hObject,'String')) returns contents of INCedit as a
double
```

```
i = str2num(get(handles.INCedit, 'string'));

set(handles.Inclination, 'value', i);

% --- Executes during object creation, after setting all properties.
function INCedit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to INCedit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%First state initial orbital elements in order to find initial r and v in
%geocentric equatorial coordinates
hold off
% Plot Earth
[handles.Rex,handles.Rey,handles.Rez] = sphere();   %Sphere coordinates for
Earth
handles.earth = surf(handles.Re*handles.Rex, handles.Re*handles.Rey,
handles.Re*handles.Rez, 'edgeColor','none', 'faceColor', [.5 .9
.5],'faceAlpha', 0.55);

I = imread('earth_map1.jpg');
axis vis3d
set(handles.earth,'CData', I,'FaceColor','texturemap');
hold all
set(gcf,'color','black'); %[.85,.7,1]

set(gca,'Color','black','XColor','white', ...
    'YColor','white','ZColor','white')

%Set handles.constants
handles.Re = 6378; %Radius of Earth (km)
handles.mu = 398600; %Gravitational Parameter of Earth (km^3/s^2)

%Call COEs inputted by user

%Angular Momentum
handles.h1 = str2num(get(handles.AngMomentum, 'string')); %km^2/s

%Inclination
handles.inc1 = (get(handles.Inclination,'Value'))*pi/180; %radians
```

```
%Right Ascension of Ascending Node
handles.RA1 = (str2num(get(handles.RAAN, 'string')))*pi/180; %radians

%Eccentricity
handles.e1 = str2num(get(handles.eccentricity, 'string'));

%Argument of Perigee
handles.omega1 = (str2num(get(handles.Arg_Perigee, 'string')))*pi/180;
%radians

%True Anomaly TrueAnomaly
handles.theta1 = (get(handles.TrueAnomaly,'Value'))*pi/180; %radians

%Now find r in km and v in km/s
[handles.r1,handles.v1] =
coes_RV(handles.h1,handles.inc1,handles.RA1,handles.e1,handles.omega1,handles
.theta1);



%Find Range
handles.range1 = norm(handles.r1) - handles.Re; %in km

%Magnitude of radius
handles.r1norm=norm(handles.r1); % in km

%Find the semimajor axis
handles.rp=handles.h1^2/handles.mu/(1+handles.e1); %in km
handles.ra=handles.h1^2/handles.mu/(1-handles.e1); % in km
handles.a1=(handles.rp + handles.ra)/2; % in km

%Find Period of orbit
handles.T1 = 2*pi/sqrt(handles.mu)*(handles.a1)^(1.5); %in seconds

%Now these are the initial conditions and time span
handles.y1 = [handles.r1 handles.v1];
handles.t1 = [0 handles.T1]; %in seconds

%Use ode45 to get orbit

%Set tolerance
options = odeset('RelTol', 1e-6);

[handles.t2,handles.y2] = ode45('orbit', handles.t1, handles.y1, options);



%Simulation
for i = 1:length(handles.t2)
    plot3(handles.y2(1:i,1), handles.y2(1:i,2), handles.y2(1:i,3),'color',
'magenta')
    axis([-2*handles.a1-5000 handles.a1+5000 -handles.a1-5000 handles.a1+5000
-.5*handles.a1-5000 .5*handles.a1+5000])
    drawnow
end
```