Eleonoor van Beers
4273427
23-11-2018
Hours spent: 16

Assignment 1- BASICS 2

The symbols, notation and units that will be used throughout this document are shown in Table 1, and the constants used in this document shown in Table **??**. All equations and constants stated in this document have been taken from Mission Geometry; Orbit and Constellation Design and Management by James R. Wertz [2].

Notation	**Definition**	**Units**
x	x-position	m
y	y-position	m
z	z-position	m
V_x	velocity in x-direction	m/s
V_y	velocity in y-direction	m/s
V_z	velocity in z-direction	m/s
a	semi-major axis	m
e	eccentricity	-
i	inclination	degrees
Ω	right ascension of the ascending node	degrees
ω	argument of perigee	degrees
θ	true anomaly	degrees
E	eccentric anomaly	degrees
M	mean anomaly	degrees
r	radius	m
H	angular momentum	m^2/s
bold	vector	-
$\hat{}$	unit vector	-
x	cross product	-
$\cdot$	dot product	-

Table 1: Symbols, notation and units used in this document

Notation	**Definition**	**Value**	**Units**
μ_e	gravitational parameter of Earth	398600.441	km^3/s^2

Table 2: Constants used in this document

In orbital mechanics, a classical 2-body problem is expressed using the six Keplerian orbital elements: a, e, i, Ω, ω and θ. The assumptions necessary for a two-body problem are as follows:

– the bodies are point masses

– there are no internal or external forces acting upon the system

The usage of the 6 Keplerian elements allows the position and velocity of the spacecraft or body to be known at any point in time. For Cartesian coordinates, this is no longer possible, as each component must be known at each point in time. However when the problem expands to n-body, the Kepler form no longer can be used. This is why it is necessary to convert between the two coordinate systems. The Keplierian elements can be visualised in Figure 1.

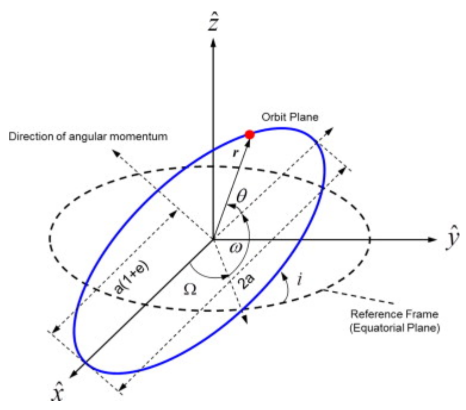

Figure 1: Visualisation of the Keplerian Elements, as depicted by Yuchul, Shin [3].

Question 1. Convert the following state-vector from Cartesian components to Kepler elements: x = 10157768.1264 m; y = -6475997.0091 m; z = 2421205.9518 m; xdot = 1099.2953996 m/s; ydot = 3455.1059240 m/s; zdot = 4355.0978095 m.

To convert the Cartesian to Keplerian elements, the following equations are used to calculate the velocity magnitude (1), radius magnitude (2) and angular momentum vector (3) from the Cartesian components given.

$$V = \|\mathbf{V}\| = \sqrt{V_x^2 + V_y^2 + V_z^2} \quad (1) \qquad r = \|\mathbf{r}\| = \sqrt{x^2 + y^2 + z^2} \quad (2) \qquad \mathbf{h} = \mathbf{r} \times \mathbf{V} \quad (3)$$

Using these values, the semi-major axis (4), eccentricity vector (5), eccentricity scalar (6), and inclination (7) can then be calculated using V, $\mathbf{V}$, r, $\mathbf{r}$, $\mathbf{h}$ and μ_e. The inclination is found by the equations is expressed in radians, and is therefore converted to degrees by multiplying by $180/\pi$.

$$a = \frac{1}{\frac{2}{r} - \frac{V^2}{2}} \quad (4) \qquad \mathbf{e} = \frac{\mathbf{V} \times \mathbf{H}}{\mu_e} - \frac{\mathbf{r}}{r} \quad (5) \qquad e = \|\mathbf{e}\| = \sqrt{e_x^2 + e_y^2 + e_x^2} \quad (6) \qquad i = a\cos(\frac{H_z}{\|\mathbf{H}\|}) \quad (7)$$

To determine Ω, and which quadrant ω, θ, E and M angles are in, a parameter N is introduced, shown by Equation 8, with its scalar calculated by Equation 9 (as the z-component of N is 0).

$$\mathbf{N} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \times \mathbf{H} \quad (8) \qquad N_{xy} = \sqrt{N_x^2 + N_y^2} \quad (9)$$

Ω can directly be calculated from $\mathbf{N}$ and N_{xy}, as shown in Equation 10. The atan2 function takes into account the signs of the inputs, therefore giving the angle in the correct quadrant. The equation results in an answer in radians, and is subsequently converted to degrees.

$$\Omega = a\tan2(\frac{N_y}{N_{xy}}, \frac{N_x}{N_{xy}}) \quad (10)$$

ω and θ can be calculated using Equations 11 and 13, and its sign (quadrant) is determined by a condition, respectively. These are shown next to the equations. The equations calculate the angles in radians, which are consequently converted to degrees. As both ω and θ are defined between 0° and 360°, if the angle calculated is negative, its absolute value should be subtracted from 360° to adhere to the notation. Once the θ is known, E and M (16) can be found using the equations (15) and (16).

$$\theta = a\cos(\hat{\mathbf{r}} \cdot \hat{\mathbf{e}}) \quad (11) \qquad \text{multiply by } -1 \text{ if } (\hat{\mathbf{e}} \times \mathbf{r}) \cdot \mathbf{H} > 0 \quad (12)$$

$$\omega = a\cos(\hat{\mathbf{e}} \cdot \hat{\mathbf{N}}) \quad (13) \qquad \text{multiply by } -1 \text{ if } (\hat{\mathbf{N}} \times \mathbf{e}) \cdot \mathbf{H} > 0 \quad (14)$$

$$E = 2 * \text{atan}(\tan(\frac{\theta/2}{\sqrt{\frac{1+e_s}{1-e_s}}}) \qquad (15)$$

$$M = E - e * \sin(E) \qquad (16)$$

Throughout the coding process, verification is performed by checking that the formulas used in the code are correct, the syntax is correct, and the units also consistent. To validate the code, a test is run using NASA data, provided in the Lecture Slides [1]. Both Cartesian and Keplerian elements are provided, and the calculated Cartesian components are compared. The given input Cartesian components, given Keplerian elements, and calculated Keplerian elements for the International Space Station on June 12, 2014, 12:00:00 hrs [NASA, 2014] is shown in Table ??. As can be seen, the calculated values are identical to the given data when rounded to 10 significant figures. Therefore the code is validated, and can be used for BASICS-2. The given input Cartesian components and calculated Keplerian elements for BASICS-2 can be seen in Table ??.

Cartesian Component [Given]	**Value**	**Units**
x	-2700816.14	m
y	-3314092.80	m
z	5266346.42	m
V_x	5168.606550	m/s
V_y	-5597.546618	m/s
V_z	-868.878445	m/s

Keplerian Element	**Value [Given]**	**Value [Calculated]**	**Units**
a	6787746.891	6787746.891	m
e	0.000731104	0.000731104128909	-
i	51.68714486	51.6871448604	$^\circ$
Ω	127.5486706	127.548670575	$^\circ$
ω	74.21987137	74.2198713719	$^\circ$
θ	24.10027677	24.1002767694	$^\circ$
E	24.08317766	24.0831776597	$^\circ$
M	24.06608426	24.0660842587	$^\circ$

Table 3: The given input Cartesian components, given Keplerian elements, and calculated Keplerian elements for the International Space Station on June 12, 2014, 12:00:00 hrs [NASA, 2014]. add ref

Cartesian Component [Given]	**Value**	**Units**
x	10157768.1264	m
y	-6475997.0091	m
z	2421205.9518	m
V_x	1099.2953996	m/s
V_y	3455.1059240	m/s
V_z	4355.0978095	m/s

Keplerian Element	**Value [Own]**	**Units**
a	12164958.9354	m
e	0.0138695262694	-
i	52.6776704353	$^\circ$
Ω	318.666326139	$^\circ$
ω	151.433761697	$^\circ$
θ	222.912676818	$^\circ$
E	223.4565352	$^\circ$
M	224.00310927	$^\circ$

Table 4: The given input Cartesian components and calculated Keplerian elements for BASICS-2

The results are similar (similar orders of magnitude) to the ISS measurements, therefore not immediately false.

Question 2. Convert the following state-vector from Kepler elements to Cartesian components: a = 12269687.5912 m; e = 0.004932091570; i = 109.823277603 deg; RAAN = 134.625563565 deg; argument of perigee = 106.380426142 deg; M = 301.149932402 deg.

First, as the equations require the angles to be in radians, they are converted to radians. Cartesian positions components can then further be expressed by Equation 17, where ξ and η are defined by by Equation 18.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} l_1 & l_2 \\ m_1 & m_2 \\ n_1 & n_2 \end{bmatrix} \begin{bmatrix} \xi \\ \eta \end{bmatrix} \quad (17) \qquad \begin{bmatrix} \xi \\ \eta \end{bmatrix} = \begin{bmatrix} r\mathrm{cos}\theta \\ r\mathrm{sin}\theta \end{bmatrix} \quad (18)$$

To solve for ξ and η, r and θ are needed. To calculate θ, E must be found from M, shown in Equation 19. E_i is taken to be M for the first iteration, and must converge to a value that differs from the previous iteration by less than 1E-10, as the components are expressed to 10 significant figures. From E, θ can then be computed, as per Equation 20. As θ is defined from 0° to 360° (0 to 2π), if the θ found is negative, its absolute value is subtracted from 2π. It is then converted to degrees.

$$E_{i+1} = E_i + \frac{M - E_i + e\mathrm{sin}E_i}{1 - e\mathrm{cos}E_i} \quad (19) \qquad \theta = 2\mathrm{atan}(\sqrt{\frac{1+e}{1-e}}\mathrm{tan}(E/2)) \quad (20)$$

To calculate r, Equation 21 is used, and thereby ξ and η can be found.

$$r = a(1 - e\mathrm{cos}E) \quad (21)$$

Parameters $l_{1,2}$, $m_{1,2}$ and $n_{1,2}$ are defined by Equations 22-27. Inputs Ω, ω and $\imath$ can then be substituted as they are given, and the Cartesian position components can be extracted from Equation 17.

$$l_1 = \mathrm{cos}\Omega\mathrm{cos}\omega - \mathrm{sin}\Omega\mathrm{sin}\omega\mathrm{cos}i \quad (22) \qquad l_1 = -\mathrm{cos}\Omega\mathrm{sin}\omega - \mathrm{sin}\Omega\mathrm{cos}\omega\mathrm{cos}i \quad (23)$$

$$m_1 = \mathrm{sin}\Omega\mathrm{cos}\omega + \mathrm{cos}\Omega\mathrm{sin}\omega\mathrm{cos}i \quad (24) \qquad m_2 = -\mathrm{sin}\Omega\mathrm{sin}\omega + \mathrm{cos}\Omega\mathrm{cos}\omega\mathrm{cos}i \quad (25)$$

$$n_1 = \mathrm{sin}\omega\mathrm{cos}i \quad (26) \qquad n_2 = \mathrm{cos}\omega\mathrm{sin}i \quad (27)$$

To find the Cartesian velocity components, first the H is calculated by Equation 28. Once found, H, μ_e, $l_{1,2}$ $m_{1,2}$, $n_{1,2}$, e and θ can be used to calculate the Cartesian velocity components using Equations 29-31.

$$H = \sqrt{\mu_e a(1-e^2)} \quad (28) \qquad \mathrm{V}_x = \frac{\mu_e}{H}(-l_1\mathrm{sin}\theta + l_2(e + \mathrm{cos}\theta) \quad (29)$$

$$\mathrm{V}_y = \frac{\mu_e}{H}(-m_1\mathrm{sin}\theta + m_2(e + \mathrm{cos}\theta) \quad (30) \qquad \mathrm{V}_z = \frac{\mu_e}{H}(-n_1\mathrm{sin}\theta + n_2(e + \mathrm{cos}\theta) \quad (31)$$

Again, throughout the coding process, verification is performed by checking that the formulas used in the code are correct,the syntax is correct, and the units also consistent. The same check is performed as in question 1, however this time reversed. Table **??** provides the input Keplerian elements, the given output Cartesian components and the calculated output Cartesian components. As can be seen, they are identical. Therefore the code is validated, and can be used for BASICS-2. The given input Cartesian components and calculated Keplerian elements for BASICS-2 can be seen in Table **??**.

Keplerian Element [Given]	**Value**	**Units**
a	6787746.891	m
e	0.000731104	-
i	51.68714486	$\circ$
Ω	127.5486706	$\circ$
ω	74.21987137	$\circ$
M	24.06608426	$\circ$

Cartesian Component	**Value [Given]**	**Value [Calculated]**	**Units**
x	-2700816.14	-2700816.14	m
y	-3314092.80	-3314092.80	m
z	5266346.42	5266346.42	m
V_x	5168.606550	5168.60655	m/s
V_y	-5597.546618	-5597.546618	m/s
V_z	-868.878445	868.878445002	m/s

Table 5: The given input Keplerian elements, given Cartesian components, and calculated Cartesian component for the International Space Station on June 12, 2014, 12:00:00 hrs [NASA, 2014]. [1]

.

Keplerian Element [Given]	**Value**	**Units**
a	12269687.5912	m
e	0.004932091570	-
i	109.823277603	$\circ$
Ω	134.625563565	$\circ$
ω	106.380426142	$\circ$
M	301.149932402	$\circ$

Cartesian Component	**Value [Calculated]**	**Units**
x	-3696459.03851207	m
y	8069268.49893917	m
z	8426536.55821229	m
V_x	3884.8809086	m/s
V_y	-2064.82916621	m/s
V_z	3646.34085825	m/s

Table 6: The given input Keplerian elements, calculated Cartesian component for BASICS-2.

The results are similar (similar orders of magnitude) to the ISS measurements, therefore not immediately false.

Bibliography

[1] Ron Noomen. Ae4-878.basics.v4-27$_{u}ntilsheet23.LectureSlidesMissionGeometryandOrbitDesign$, 2018.

[2] James R. Wertz. *Mission Geometry; Orbit and Constellation Design and Management.* Springer Netherlands, 2001.

[3] Shin et. al Yuchul. Radiation effect for a cubesat in slow transition from the earth to the moon. 2015.

A Appendix

A.1 Cartesian to Keplerian

```
import numpy as np

##################### KNOWN PARAMETERS ################################

mu_e = 398600.441  #[km^3/s^2]
au = 149597870.88  #[km]


##################### CARTESIAN TO KEPLERIAN ########################

#assign array for cartesian coordinates
keplerian = []

#input variables

x = 10157768.1264                                        #[m]
y = -6475997.0091                                       #[m]
z = 2421205.9518                                         #[m]
v_x = 1099.2953996                                      #[m/s]
v_y = 3455.1059240                                     #[m/s]
v_z = 4355.0978095                                     #[m/s]

#input variables in vector and scalar forms in m and m/s

r_v = np.array([x,                                     #vector form [m]
                y,
                z])
r_v = r_v/1000                                         #vector form [km]
r_s = (np.sqrt(x**2 + y**2 + z**2))/1000               #scalar form [km]
r_hat = r_v/r_s                                        #unit vector form [km]

V_v = np.array([v_x,                                   #vector form [m/s]
                v_y,
                v_z])
V_s = np.sqrt(v_x**2 + v_y**2 + v_z**2)                #scalar form [m/s]

#calculate angular momentum
h_v = np.cross(r_v,V_v/1000)                           #vector form [km^2/s]
h_s = np.sqrt(h_v[0]**2 + h_v[1]**2 + h_v[2]**2)       #scalar form [km^2/s]

#calculate a in km
a = (1/((2/r_s)-((V_s/1000)**2/mu_e)))*1000
keplerian.append(a)
print "a =", a

#calculate e
e_v = (np.cross(V_v/1000,h_v)/mu_e) - (r_v/r_s)       #vector form
e_s = np.sqrt(e_v[0]**2 + e_v[1]**2 + e_v[2]**2)       #scalar form
e_hat = e_v/e_s                                        #unit vector form
keplerian.append(e_s)
print "e =", e_s

#calculate i in radians
i = np.arccos(h_v[2]/h_s)
keplerian.append(i)
```

```
print "i =", i* (180/np.pi)

#calculate quardrant parameters to check the sign
nvector = np.array([0,
                    0,
                    1])
N_v = np.cross(nvector,h_v)                                #vector form
N_xy = np.sqrt(N_v[0]**2 + N_v[1]**2)                      #vector in xy plane
N_hat = N_v/N_xy                                           #unit vector form

#calculate RAAN in radians
omega = np.arctan2((N_v[1]/N_xy),(N_v[0]/N_xy))
if omega < 0:
    omega = (2*np.pi)+omega
keplerian.append(omega)
print "omega = ", omega*(180/np.pi)

#calculate argument of perigee in radians
if np.dot((np.cross(N_hat,e_hat)),h_v) > 0:
    w = np.arccos(np.dot(e_hat,N_hat))
    keplerian.append(w)
else:
    w = -np.arccos(np.dot(e_hat,N_hat))
    keplerian.append(w)
print "w = ", w*(180/np.pi)

#calculate true anomaly in radians
if np.dot((np.cross(e_v,r_v)),h_v) > 0:
    theta = np.arccos(np.dot(r_hat,e_hat))
    keplerian.append(theta)
else:
    theta = -np.arccos(np.dot(r_hat,e_hat))
    keplerian.append(theta)
if theta < 0:
    theta = (2*np.pi)+theta
print "theta = ", theta*(180/np.pi)

#calculate eccentric anomaly in radians
E = 2*np.arctan(np.tan(theta/2)/(np.sqrt((1+e_s)/(1-e_s))))
keplerian.append(E)
if E < 0:
    E = (2*np.pi)+E
print "E = ", E * (180/np.pi)

#calculate mean anomaly
M = E - e_s*np.sin(E)
if M < 0:
    M = (2*np.pi)+M
keplerian.append(M)
print "M = ", M * (180/np.pi)
```

A.2 Keplerian to Cartesian

```
import numpy as np

###################### KNOWN PARAMETERS ##################################

mu_e = 398600.441  #[km^3/s^2]
au = 149597870.88  #[km]
```

```
###################### KEPLERIAN TO CARTESIAN ##########################

#assign array for cartesian coordinates
cartesian = []

#input variables

a = 12269687.5912                               #[m], semi-major axis
e = 0.004932091570                              #[-], eccentricity
i = 109.823277603                             #[deg], inclination
omega = 134.625563565                           #[deg], RAAN
w = 106.380426142                              #[deg], argument of peri
#theta = 239.5437                             #[deg], true anomaly
#E = 239.5991                                 #[deg], eccentric anomaly
M = 301.149932402                               #[deg], mean anomaly

#transform degrees to radians
i = i*(np.pi/180)
omega = omega*(np.pi/180)
w = w*(np.pi/180)
#theta = theta*(np.pi/180)
#E = E*(np.pi/180)
M = M*(np.pi/180)

#calculate E from M through iteration
E=M
diff=1
while abs(diff) > 1e-10:
            diff=(M-E+e*np.sin(E))/(1-e*np.cos(E))
            E=E+diff

#calculate theta from E
theta = 2*np.arctan(np.sqrt((1+e)/(1-e))*np.tan(E/2))
if theta < 0:
    theta = (2*np.pi) + theta

#calculate r from input variables
r = a*(1-(e*np.cos(E)))                        #[m]


######### calculate the cartesian coordinates #######


#calculate the first matrix for position

l1 = np.cos(omega)*np.cos(w) - np.sin(omega)*np.sin(w)*np.cos(i)
l2 = -np.cos(omega)*np.sin(w) - np.sin(omega)*np.cos(w)*np.cos(i)
m1 = np.sin(omega)*np.cos(w) + np.cos(omega)*np.sin(w)*np.cos(i)
m2 = -np.sin(omega)*np.sin(w) + np.cos(omega)*np.cos(w)*np.cos(i)
n1 = np.sin(w)*np.sin(i)
n2 = np.cos(w)*np.sin(i)
mat1 = np.array([[l1, l2],
                 [m1, m2],
                 [n1, n2]])


#calculate the second matrix for position

curlye = r*np.cos(theta)  #curly e to perigee
longn = r*np.sin(theta)   #long n perpendicular to curly e
```

```
mat2 = np.array([[curlye],
                 [longn]])

#calculate the position coordinates in km
coordinates_c = np.dot(mat1,mat2)
x = coordinates_c[0]
y = coordinates_c[1]
z = coordinates_c[2]

print "x =", x
print "y =", y
print "z =",  z #m

cartesian.append(x)
cartesian.append(y)
cartesian.append(z)


#calculate the angular momentum
H = np.sqrt(mu_e*(a/1000)*(1-e**2))


#calculate the velocities in km/s
v_x = (mu_e/H)*(-l1*np.sin(theta) + l2*(e+np.cos(theta)))
v_y = (mu_e/H)*(-m1*np.sin(theta) + m2*(e+np.cos(theta)))
v_z = (mu_e/H)*(-n1*np.sin(theta) + n2*(e+np.cos(theta)))

cartesian.append(v_x*1000)
cartesian.append(v_y*1000)
cartesian.append(v_z*1000)

print "v_x =", v_x * 1000
print "v_y =", v_y*1000
print"v_z =", v_z*1000 #m/s
```