```matlab
clear all; clc; close all;

ro = [7000 2900 -3800];
vo = [7.11 1.52 2.69];

[h,e,i,w,W,true_ana] = stateVec2OrbElem(ro,vo);

fprintf('True Anomaly: %0.3f  degrees \n', true_ana)
fprintf('Specific Angular Momentum: %0.0f km^2/s \n', norm(h))
fprintf('Eccentricity: %0.3f \n', norm(e))
fprintf('Inclination: %0.3f  degrees \n', i)
fprintf('Right ascension of the ascending Node: %0.3f degrees \n', W)
fprintf('Argument of perigee %0.3f degrees \n', w)

function [h,e,i,w,W,true_ano,N] = stateVec2OrbElem(r,v)
%This function will take two state vectors r and v and compute the six
%orbital elements
% r and v must be 3-D vectors
% Currently this is only for Geocentric orbits
mu = 3.9860044189e5;



h = cross(r,v); % specific angular momentum
i = acosd(h(3)/norm(h));  % inclination is hz divided by the magnitude of h
N = cross([0 0 1],h); % line of nodes is the unit vector k cross h

    if N(2) >= 0
        % Right ascesnsion of the ascending Node
        W = acosd(N(1)/norm(N));
    else
        W = 360-acosd(N(1)/norm(N));
    end
 % eccentricity vector
e = ((norm(v)^2 - mu/norm(r))*r - (dot(r,v)*v))/mu;

    if e(3) >= 0
        % argument of perigee
        w = acosd(dot(N,e)/(norm(N)*norm(e)));
    else
        w = 360-acosd(dot(N,e)/(norm(N)*norm(e)));
    end

    if (dot(r,v)/norm(r)) >= 0
        %true anomaly
        true_ano = acosd(dot(e,r)/(norm(e)*norm(r)));
    else
        true_ano = 360-acosd(dot(e,r)/(norm(e)*norm(r)));
    end


end
```

True Anomaly: 114.815 degrees

Specific Angular Momentum: 48846 km^2/s

Eccentricity: 0.700

Inclination: 101.788 degrees

Right ascension of the ascending Node: 16.496 degrees

Argument of perigee 217.930 degrees

>>

```
clear all; clc; close all;

h = 75000 ;
e = 0.3;
i = 95*pi/180;
W = 30*pi/180;
w = 50*pi/180;
true_ana = 40*pi/180;


[ro,vo] = OrbElem2StateVec(h,e,i,w,W,true_ana);

fprintf('Position Vector: [%0.0f %0.0f %0.0f] km \n', ro)
fprintf('Velocity Vector: [%0.3f %0.3f %0.3f] km/s \n', vo)



function [RX,VX,r,v,QXx] = OrbElem2StateVec(h,e,i,w,W,true_ano)
%This function will take orbital elements and compute two state vectors r and
v
% r and v must be 3-D vectors
mu = 3.9860044189e5;
r = (h^2/mu)/(1+e*cos(true_ano))*[cos(true_ano) sin(true_ano) 0]';
v = mu/h*[-sin(true_ano) (e+cos(true_ano)) 0]';
R3_W = [cos(W)  sin(W) 0;
      -sin(W) cos(W) 0;
      0       0      1;];
R1_i = [1 0 0;
        0 cos(i) sin(i);
        0 -sin(i) cos(i);];
R3_w = [ cos(w) sin(w) 0;
        -sin(w) cos(w) 0
        0 0 1;];
QXx = (R3_w) *(R1_i) *(R3_W);

RX = QXx.'*r;
VX = QXx.'*v;


end
```

Position Vector: [500 -866 11431] km

Velocity Vector: [-5.616 -3.345 1.021] km/s

>>

```
clear all; clc; close all;

rp = 500+6378;
mu = 3.9860044189e5;
e = 1.5;
i = 35*pi/180;
W = 150*pi/180;
w = 115*pi/180;
true_ana = 0*pi/180;
h = sqrt(rp*mu*(1+e*cos(true_ana)));

dt = 3600*2;
Me = mu^2/h^3*(e^2-1)^1.5*dt;
F_guess = 1.8;
func = @(F) e*sinh(F)-F-Me;
F = fzero(func,F_guess);
true_ana = 2*atan(sqrt((e+1)/(e-1))*tanh(F/2));

[ro,vo,r,v,QXx] = OrbElem2StateVec(h,e,i,w,W,true_ana);

fprintf('Position Vector: [%0.0f %0.0f %0.0f] km \n', ro)
fprintf('Velocity Vector: [%0.3f %0.3f %0.3f] km/s \n', vo)


function [RX,VX,r,v,QXx] = OrbElem2StateVec(h,e,i,w,W,true_ano)
%This function will take orbital elements and compute two state vectors r and
v
% r and v must be 3-D vectors
mu = 3.9860044189e5;
r = (h^2/mu)/(1+e*cos(true_ano))*[cos(true_ano) sin(true_ano) 0]';
v = mu/h*[-sin(true_ano) (e+cos(true_ano)) 0]';
R3_W = [cos(W)  sin(W) 0;
      -sin(W) cos(W) 0;
      0       0      1;];
R1_i = [1 0 0;
        0 cos(i) sin(i);
        0 -sin(i) cos(i);];
R3_w = [ cos(w) sin(w) 0;
        -sin(w) cos(w) 0
        0 0 1;];
QXx = (R3_w) *(R1_i) *(R3_W);

RX = QXx.'*r;
VX = QXx.'*v;


end
```

Position Vector: [46011 13466 -24274] km

Velocity Vector: [4.850 2.893 -3.452] km/s

```matlab
clear all; clc; close all;
mu = 3.9860044189e5;
R1 = [3286 5010 9787];
R2 = [-3259 260 13009];
R3 = [-11787 -7674 9628];

r1 = norm(R1);
r2 = norm(R2);
r3 = norm(R3);
N = r1*(cross(R2,R3)) + r2*(cross(R3,R1)) + r3*(cross(R1,R2));
D = cross(R2,R3) + cross(R3,R1) + cross(R1,R2);
S = R1*(r2-r3) + R2*(r3-r1) + R3*(r1-r2);

V2 = sqrt(mu/(dot(N,D)))*(cross(D,R2)/norm(R2)+S);

r = R2;
v = V2;
[h,e,i,w,W,true_ana] = stateVec2OrbElem(r,v);

v2 = sqrt(mu/(dot(N,D)))*(cross(D,R2)/norm(R2)+S);

rp = norm(h)^2/mu*1/(1+norm(e))-6378;
fprintf('True Anamoly: %0.3f  degrees \n', true_ana)
fprintf('Specific Angular Momentum: %0.0f km^2/s \n', norm(h))
fprintf('Eccentricity: %0.3f \n', norm(e))
fprintf('Inclination: %0.3f  degrees \n', i)
fprintf('Right ascesnsion of the ascending Node: %0.3f degrees \n', W)
fprintf('Argument of perigee %0.3f degrees \n', w)

if i <90
    fprintf('The orbit is Prograde: i<90 \n')
else
    fprintf('The orbit is Retrograde: i>90 \n')
end
if true_ana >180
    fprintf('The satellite is approaching perigee \n')
else
    fprintf('The satellite is going away perigee \n')
end
fprintf('Perigee Altitude %0.3f km \n', rp)
```

```
function [RX,VX,r,v,QXx] = OrbElem2StateVec(h,e,i,w,W,true_ano)
%This function will take orbital elements and compute two state vectors r and
v
% r and v must be 3-D vectors
mu = 3.9860044189e5;
r = (h^2/mu)/(1+e*cos(true_ano))*[cos(true_ano) sin(true_ano) 0]';
v = mu/h*[-sin(true_ano) (e+cos(true_ano)) 0]';
R3_W = [cos(W)  sin(W) 0;
      -sin(W) cos(W) 0;
      0       0      1;];
R1_i = [1 0 0;
        0 cos(i) sin(i);
        0 -sin(i) cos(i);];
R3_w = [ cos(w) sin(w) 0;
        -sin(w) cos(w) 0
        0 0 1;];
QXx = (R3_w) *(R1_i) *(R3_W);

RX = QXx.'*r;
VX = QXx.'*v;

end
```

True Anomaly: 80.000 degrees

Specific Angular Momentum: 75001 km^2/s

Eccentricity: 0.300

Inclination: 79.999 degrees

Right ascension of the ascending Node: 39.999 degrees

Argument of perigee 20.001 degrees

The orbit is Prograde: i<90

The satellite is going away perigee

Perigee Altitude 4477.849 km

```
clear all; clc; close all;
mu = 3.9860044189e5;

R1 = [5500 -2505 -3000];
R2 = [-3100 6910 -8850];
dt = 30*60;

r1 = norm(R1);
r2 = norm(R2);

dt_cond = cross(R1,R2);
if dt_cond(3) > 0
    dtheta = acos(dot(R1,R2)/(r1*r2));
else
    dtheta = (360 - acos(dot(R1,R2)/(r1*r2)))*pi/180;
end

A = sin(dtheta)*sqrt((r1*r2)/(1-cos(dtheta)));
zguess = 1.5;

z=fzero('lambert',zguess);

A = sin(dtheta)*sqrt((r1*r2)/(1-cos(dtheta)));
C_stu = (cosh(sqrt(-z)) - 1)/(-z);
S_stu = abs((sinh(sqrt(-z))-sqrt(-z))/(sqrt(-z)^3));
y = r1+r2+A*(z*S_stu-1)/sqrt(C_stu);

f = 1-y/r1;
g = A*sqrt(y/mu);
fdot = sqrt(mu/(r1*r2))*sqrt(y/C_stu)*(z*S_stu-1);
gdot = 1-y/r2;
V1 = 1/g*(R2-f*R1);
V2 = 1/g*(gdot*R2-R1);

[h,e,i,w,W,true_ano] = stateVec2OrbElem(R1,V1);

vr2 = mu/(norm(h))*norm(e)*sin(true_ano);
rp = norm(h)^2/mu*1/(1+norm(e))-6378;

fprintf('True Anomaly: %0.3f  degrees \n', true_ano)
fprintf('Specific Angular Momentum: %0.0f km^2/s \n', norm(h))
fprintf('Eccentricity: %0.3f \n', norm(e))
fprintf('Inclination: %0.3f  degrees \n', i)
fprintf('Right ascesnsion of the ascending Node: %0.3f degrees \n', W)
fprintf('Argument of perigee %0.3f degrees \n', w)
if i <90
    fprintf('The orbit is Prograde: i<90 \n')
else
    fprintf('The orbit is Retrograde: i>90 \n')
end
fprintf('Perigee Altitude %0.3f km \n', rp)
```

```
function func = lambert(z)
mu = 3.9860044189e5;
R1 = [5500 -2505 -3000];
R2 = [-3100 6910 -8850];
dt = 30*60;

r1 = norm(R1);
r2 = norm(R2);

dt_cond = cross(R1,R2);
if dt_cond(3) > 0
    dtheta = acos(dot(R1,R2)/(r1*r2));
else
    dtheta = (360 - acosd(dot(R1,R2)/(r1*r2)))*pi/180;
end

A = sin(dtheta)*sqrt((r1*r2)/(1-cos(dtheta)));
C_stu = (cosh(sqrt(-z)) - 1)/(-z);
S_stu = abs((sinh(sqrt(-z))-sqrt(-z))/(sqrt(-z)^3));
y = r1+r2+A*(z*S_stu-1)/sqrt(C_stu);
func = (y/C_stu)^1.5*S_stu+A*sqrt(y)-sqrt(mu)*dt;
end
```

True Anomaly: 3.193 degrees

Specific Angular Momentum: 65045 km^2/s

Eccentricity: 0.574

Inclination: 67.252 degrees

Right ascesnsion of the ascending Node: 143.500 degrees

Argument of perigee 205.632 degrees

The orbit is Prograde: i<90

Perigee Altitude 365.404 km

```
clear all; clc; close all;
mu = 3.9860044189e5;
Re = 6378;
r = [-2520 3875 -5560];
phi = -45*pi/180;
theta = 110*pi/180;
f =  0.003353;

r_site_xy = (Re/(sqrt(1-(1*f-f^2)*sin(phi)^2)))*cos(phi)*[cos(theta)
sin(theta)];
r_site_z = (Re*(1-f)^2/sqrt((1-(2*f-f^2)*sin(phi)^2)))*[sin(phi)];

r_site = [r_site_xy r_site_z];
rho_X = r-r_site;
R1 = [1 0 0;
    0 -sin(phi) cos(phi);
    0 cos(phi) sin(phi);];
R3 = [-sin(theta) cos(theta) 0;
    cos(theta) sin(theta) 0;
    0 0 1;];
QXx = R1*R3;
rho_x = QXx*rho_X';
rho = rho_x/norm(rho_x);
range = norm(rho_x);
a = asind(rho(3));

fprintf('Range to Satellite: %0.1f km \n',range)
fprintf('Elevation Angle: %0.1f degrees \n',a)
```

Range to Satellite: 1496.0 km

Elevation Angle: 30.1 degrees

```
clear all; clc; close all;
mu = 3.9860044189e5;
Re = 6378;
SatA.rp = 8525;
SatA.ra = 18000;
SatB.rp = 8525;
SatB.ra = 18000;
theta = 90*pi/180;

h1 = sqrt(2*mu)*sqrt(SatA.ra*SatA.rp/(SatA.ra+SatA.rp));
e1 = (SatA.ra - SatA.rp)/(SatA.rp + SatA.ra);
a = (SatA.rp + SatB.ra)/2;
v1 = h1/SatA.rp;
T = 2*pi/sqrt(mu)*a^(3/2);
E = 2*atan((tan(theta/2)*((1+e1)/(1-e1))^(-1/2)));
Me = E-e1*sin(E);
dt = Me*T/(2*pi);
T2 = T-dt;
a2 = (sqrt(mu)*T2/(2*pi))^(2/3);
r_new = 2*a2-SatA.rp;
h_new = sqrt(2*mu)*sqrt(SatA.rp*r_new/(SatA.rp+r_new));
v_new = h_new/SatA.rp;
dV = 2*abs(v_new - v1);


fprintf('Total delta V: %0.3f km/s \n',dV)
```

Total delta V: 0.400 km/s

```
clear all; clc; close all;
mu = 3.9860044189e5;
Re = 6378;

A.ra = 16000;
A.rp = 8000;
B.ra = 21000;
B.rp = 7000;
eta = 25*pi/180;

A.e = (A.ra-A.rp)/(A.ra+A.rp);
B.e = (B.ra-B.rp)/(B.ra+B.rp);
A.h = sqrt(A.rp*mu*(1+A.e));
B.h = sqrt(B.rp*mu*(1+B.e));

a = A.e*B.h^2-B.e*A.h^2*cos(eta);
b = -B.e*A.h^2*sin(eta);
c = A.h^2 - B.h^2;
phi = atan(b/a);
theta1 = phi + acos(c/a*cos(phi));
theta2 = theta1-eta;
r11 = A.h^2/mu/(1+A.e*cos(theta1));
r22 = B.h^2/mu/(1+B.e*cos(theta2));
A.Vtheta = A.h/r11;
A.Vr = mu/A.h*A.e*sin(theta1);
A.V = sqrt(A.Vtheta^2 + A.Vr^2);
A.gamma = atan(A.Vr/A.Vtheta);
B.Vtheta = B.h/r22;
B.Vr = mu/B.h*B.e*sin(theta2);
B.gamma = atan(B.Vr/B.Vtheta);
B.V = sqrt(B.Vtheta^2 + B.Vr^2);
dV = sqrt(A.V^2 + B.V^2 -2*A.V*B.V*cos(B.gamma-A.gamma));

fprintf('Total delta V: %0.3f km/s \n',dV)
fprintf('Inital True Anomaly: %0.3f degrees \n',theta1*180/pi)
fprintf('Phi is oriented towards the 3rd quadrant: %0.1f degrees \n',phi*180/pi)
```

Total delta V: 1.503 km/s

Inital True Anomaly: 153.036 degrees

Phi is oriented towards the 3rd quadrant: 59.4 degrees