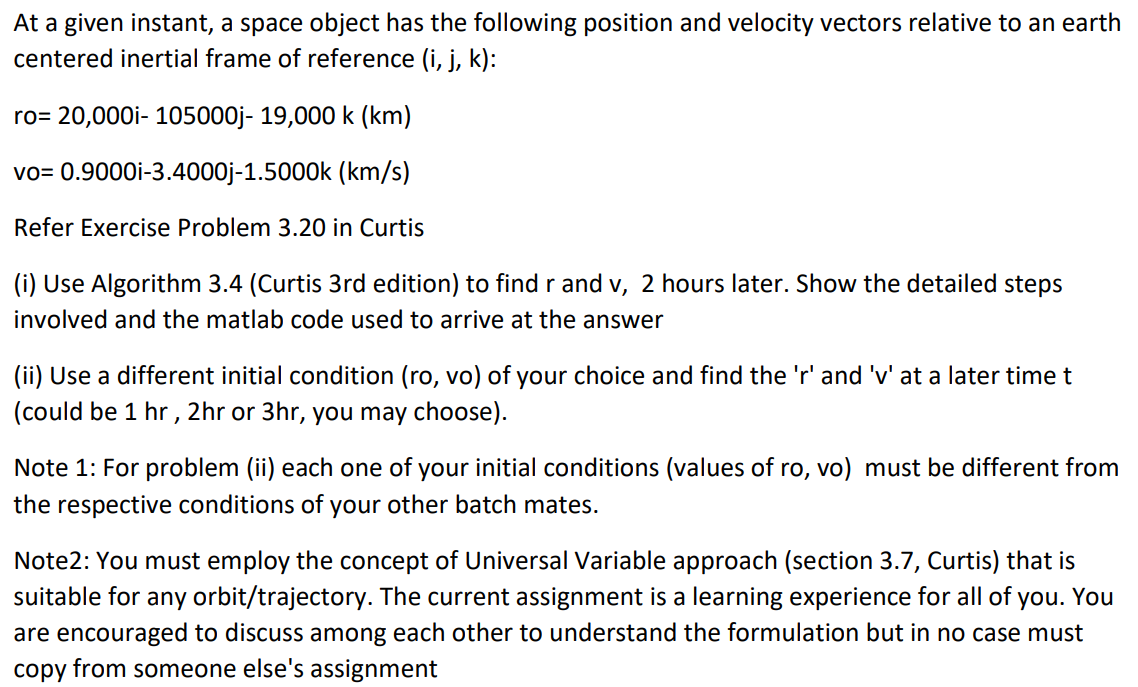
**Problem Statement:**



Kepler’s Equation for:

1. Ellipse: M\_e = E – esin(E)
2. Hyperbola: M\_h = esinh(F) – F

Universal Variables gives us Universal Kepler’s Equation:



where,

χ: universal anomaly

r0: radius at t=t0

vr0: radial component of velocity at t=t0

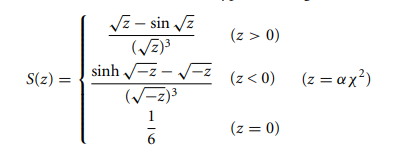
α: 1/a [α >0 for ellipse, α <0 for hyperbola, α =0 for parabola]

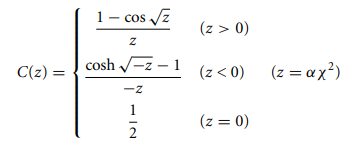
C(z) and S(z): They belong to a class of function called Stumpff functions and are defined by infinite series





C(z) and S(z) are also related to circular and hyperbolic trigo functions as





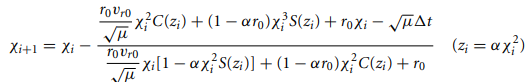
Newton’s Method is used to solve Universal Kepler’s Equation for universal anomaly χ, given the time interval. A function f(χ) is formed such that:



And



Now applying Newton’s algorithm:



A reasonable estimate for the starting value χ 0 is given by:



Following two algorithms were used to solve the problem statement.

*Algorithm 1*: To solve for universal anomaly given Δ T, r0, vr0 and alpha.

* Find the value of initial estimate χ 0
* Find f(χ i) and f’(χ i). Also calculate the ratio\_i = f/f’
* If |ratio\_i| > chosen tolerance, then update χ as χ \_i+1 = χ \_i – ratio\_i. return to step 2
* If |ratio\_i|<tolerance, then that value of χ i is chosen as the solution.

*Algorithm 2*: Given r0 and v0 vector, calculate r and v vector at a time ΔT later.

* Find magnitude r0 and v0



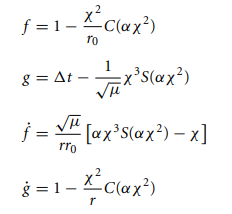
* Find vr0 using



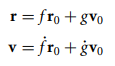
* Find alpha using



* Use value of r0, vr0, alpha and Δ T, along with Algorithm 1 to find the value of universal anomaly χ
* Find lagrange coefficients using



* Compute r and v vector using



**Appendix: MATLAB Codes**

1. **Code for finding C**

function c = C(z)

if z > 0

c = (1 - cos(sqrt(z)))/z;

elseif z < 0

c = (cosh(sqrt(-z)) - 1)/(-z);

else

c = 1/2;

end

1. **Code for finding S**

function s = S(z)

if z > 0

s = (sqrt(z) - sin(sqrt(z)))/(sqrt(z))^3;

elseif z < 0

s = (sinh(sqrt(-z)) - sqrt(-z))/(sqrt(-z))^3;

else

s = 1/6;

end

1. **Code for finding orbital elements**

% input to the function are the components of initial position and velocity vector

% output are the orbit parameters, that is, eccentricity (e), angular

% momentum of the orbit (h) and true anomaly (thetha0)

function [e,h,thetha0] = orbital\_parameters(r0x,r0y,r0z,v0x,v0y,v0z)

r0 = sqrt((r0x\*r0x) + (r0y\*r0y) + (r0z\*r0z)); % initial position vector magnitude

v0 = sqrt((v0x\*v0x) + (v0y\*v0y) + (v0z\*v0z)); % initial velocity vector magnitude

vr0 = ((r0x\*v0x) + (r0y\*v0y) + (r0z\*v0z))/r0; % radial component of velocity vector magnitude

vp0 = sqrt((v0\*v0) - (vr0\*vr0)); % perpendicular component of velocity vector magnitude

v0 = sqrt((v0x\*v0x) + (v0y\*v0y) + (v0z\*v0z));

mu = 398600;

A = (((v0\*v0) - (vr0\*vr0))\*r0/mu) - 1; % e\*cos(thetha0) value

B = (vr0\*r0/mu)\*sqrt(((v0\*v0) - (vr0\*vr0))); % e\*sin(thetha0) value

e = sqrt((A\*A) + (B\*B));

tt = acosd(A/e); % true anomaly

% Above we will get two values of tt. Now which one to accept depends upon

% the flight path angle, which is calculated as shown below.

gamma = atand(vr0/vp0);

if gamma > 0

if A > 0

thetha0 = tt;

fprintf("1st quad, t = %d", thetha0);

elseif A < 0

thetha0 = tt;

fprintf("2nd quad, t = %d", thetha0);

end

elseif gamma < 0

if A > 0

thetha0 = (-360+tt);

fprintf("4th quad, t = %d", thetha0);

elseif A < 0

thetha0 = (-360+tt);

fprintf("3rd quad, t = %d", thetha0);

end

end

h = (mu\*e\*sind(thetha0))/vr0; % angular momentum calculation

end

1. **Code for finding r and v vector using Algorithm 1 and 2 discussed above**

% known parameters

mu = 398600;

delThorus = 2; % time (in hours) after which s/c state is needed

r0v = [20000, -105000, -19000]; % given initial position vector

v0v = [0.9, -3.4, -1.5]; % given initial velocity vector

r0 = norm(r0v); % r0 is magnitude of initial position

v0 = norm(v0v); % v0 is magnitude of initial velocity

tolerance = 10^-6;

% derived parameters

[e,h, thetha0] = orbital\_parameters(r0v(1), r0v(2), r0v(3), v0v(1), v0v(2), v0v(3)); % calculating e, h and thetha0

vr0 = (dot(r0v, v0v))/r0; % magnitude of radial component of velocity vector

alpha = (2/r0) - (v0\*v0/mu);

fprintf("\n")

if alpha < 0

fprintf("Hyperbolic Trajectory")

elseif alpha > 0

fprintf("Elliptical Trajectory")

else

fprintf("Parabolic Trajectory")

end

a = 1/alpha; % semi major axis

delt = delThorus\*3600; % time (in seconds) after which s/c state is needed

% Algorithm 3.3

x = sqrt(mu)\*abs(alpha)\*delt; % initial guess

% coefficients of function f(x)

c0 = (r0\*vr0)/sqrt(mu);

c1 = 1-(alpha\*r0);

c2 = r0;

c3 = sqrt(mu)\*delt;

ratio = 1; % initializing ratio

n = 1; % iterations

while abs(ratio) > tolerance

z = alpha\*x\*x;

f = c0\*x\*x\*C(z) + c1\*x\*x\*x\*S(z) + c2\*x - c3; % f

fd = c0\*x\*((1-(alpha\*x\*x\*S(z)))) + c1\*x\*x\*C(z) + r0; % f'

ratio = f/fd;

x = x - ratio;

n = n + 1;

end

% lagrange coefficient and Algorithm 3.4

fL = 1 - (x\*x\*C(alpha\*x\*x)/r0);

gL = delt - (x\*x\*x\*S(alpha\*x\*x)/sqrt(mu));

rv = fL\*r0v + gL\*v0v; % position vector of s/c after time t

r = norm(rv); % magnitude of position vector

fLd = sqrt(mu)\*((alpha\*x\*x\*x\*S(alpha\*x\*x)) - x)/(r\*r0);

gLd = 1 - (x\*x\*C(alpha\*x\*x)/r);

vv = fLd\*r0v + gLd\*v0v; % velocity vector of s/c after time t

v = norm(vv); % magnitude of velocity vector

% displaying outputs

fprintf("\nNumber of iterations= %d", n)

fprintf("\n")

fprintf("\nValue of universal variable= %d", x)

fprintf("\n")

fprintf("r = %d i + %d j + %d k", rv(1), rv(2), rv(3))

fprintf("\n")

fprintf("v = %d i + %d j + %d k", vv(1), vv(2), vv(3))

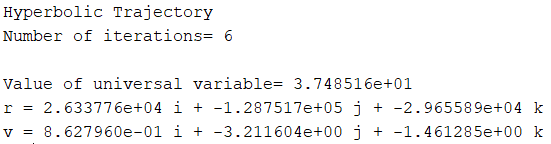
fprintf("\n")

1. **Solution 1**

Initial Conditions:



Solution obtained:

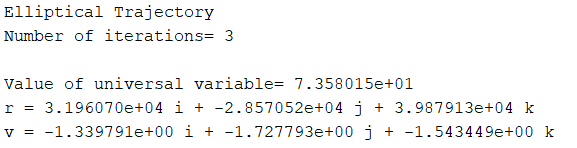


1. **Solution 2**

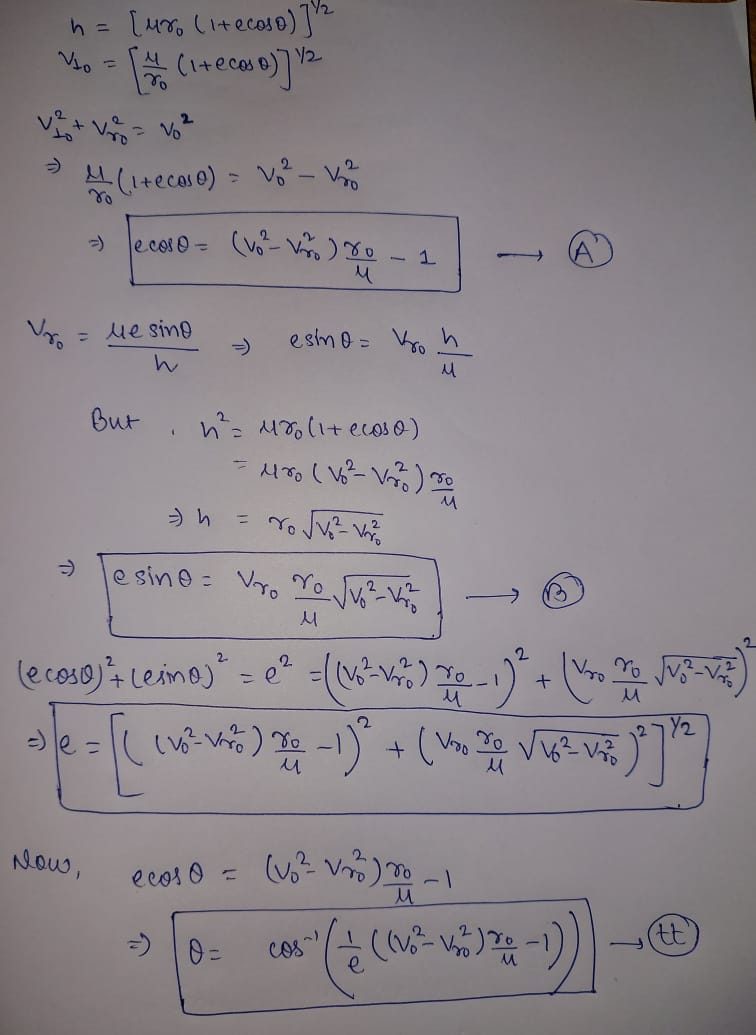
Initial Conditions:



Solution obtained:



1. **Derivation of formulas for finding orbital parameters**



**References**

* Orbital Mechanics for Engineering Students by Howard Curtis. Elsevier Aerospace Engineering Series