# Table of Contents



# 0

```
------------ HW1 - Josh Oates ------------
```

# 2.23 find gamma(theta) and z(theta)

```
Warning: joshCOE will assume that R and V are normal if the inputs are scalar
ie: the craft is in a circular orbit or is at periapse or apoapse
------------P2.23------------
My calculations have the following results:
Flight path angle (gamma): 44.5973 degrees
H/C: Theta is between 0 and 180, so gamma should be positive
Altitude at theta = 120 degrees: 12246.7575 km
H/C: Altitude is in LEO range
```

# 2.36 excess V

```
Warning: joshCOE will assume that R and V are normal if the inputs are scalar
ie: the craft is in a circular orbit or is at periapse or apoapse
------------P2.36------------
My calculations have the following results:
Excess escape velocity: 0.84994 km/s
H/C: Hyperbolic tragector should have positive excess escape velocity
r at theta = 100 degrees: 16178.7779 km
H/C: z > 250
Azmuthal velocity at theta = 100 degrees: 4.5064 km/s
H/C: Azmuthal velocity should be positive
Radial velocity at theta = 100 degrees: 5.4488 km/s
H/C: At theta = 100 object is on departure so radial velocity should be
 positive
```

# 2.37 meteroid

```
------------P2.37------------
```

```
My calculations have the following results:
Eccentricity: 1.086
H/C: For hyperbolic ecc should be above 1
Altitude at periapse: 5087.5854 km
H/C: Altitude at closest approach is lower than initial velocity but high
 enough to orbit
Velocity at closest approach: 8.5158 km/s
H/C: v at closest approach is higher than v initial
```

# 3.8 time above 400

```
------------P3.8------------
My calculations have the following results:
Time spent over 400 km: 47.1482 min
H/C: time over 400 km is less than period
Intermediate H/C used:
      ecc < 1
      period resonable for LEO
```

# 3.10

```
------------P3.10------------
My calculations have the following results:
radial position: 42354.9211 km
H/C: r is between rp and ra
speed: 2.3034 km/s
H/C: MEO orbit resonable velocity seems beleivable
radial velocity: -1.2709 km/s
H/C: since period is 14hrs and this is 10hrs in, the orbit is past apoapse, so
 radial velocity should be negative
```

# 3.20

```
------------P3.20------------
My calculations have the following results:
The final position vector in km:
   1.0e+05 *

    0.2634   -1.2875   -0.2966

The final velocity vector in km/s:
    0.8628   -3.2116   -1.4613

H/C: f*g_dot-f_dot*g value: 1
```

# 4.5

```
------------P4.5------------
My calculations have the following results:
Semimajor axis: 9081.4773 km
Eccentricity: 0.22261
```

```
True anomaly: 134.7259 degrees
Inclination: 32.445 degrees
Right ascention of ascending node: 107.5713 degrees
Argument of periapse: 72.3586 degrees
h: 58655.7755m^2/s
```

## 4.7

```
------------P4.7------------
My calculations have the following results:
Inclination of this orbit: 43.2661 degrees
H/C: r vector has a relatively large z component so this makes sense nominally
```

## dependencies

*Published with MATLAB® R2022a*

# Table of Contents



```
% Joshua Oates - AERO351 - HW2


% Feel free to use MATLAB (or similar product) when appropriate but please
 submit a pdf/html
% of the published code.
%
% Chapter 2 problems: 2.23, 2.36, 2.37
%
% Chapter 3 problems: 3.8, 3.10, and 3.20 (check this answer with using ODE45
 as well, do they
% agree?)
%
% Chapter 4 problems: 4.5, 4.7
```

# 0

```
clear all;
close all;
clc

addpath('C:\joshFunctionsMatlab\')

disp("------------ HW1 - Josh Oates ------------")
```

# 2.23 find gamma(theta) and z(theta)

```
mu_e = 398600;
r_e = 6378; % km
zp = 500; % km
rp=zp+r_e; % km
v = 10; % km/s
theta = 120; % degrees
theta = deg2rad(theta); % rad
```

```
[a,ecc,~,~,~,~,h,T]=joshCOE(rp,v,mu_e); % COEs
[vaz,vr,gamma] = joshVazVr(theta,ecc,h,mu_e); % Velocity components
gamma = rad2deg(gamma);
T = T/3600;

r=a*((1-ecc^2)/(1+ecc*cos(theta)));
z120=r-r_e;
disp("------------P2.23------------")
disp("My calculations have the following results:")
disp("Flight path angle (gamma): "+string(gamma)+" degrees")
disp("H/C: Theta is between 0 and 180, so gamma should be positive")
disp("Altitude at theta = 120 degrees: "+string(z120)+" km")
disp("H/C: Altitude is in LEO range")
```

# 2.36 excess V

```
clear all;
mu_e = 398600;
r_e = 6378; % km
zp = 250; % km
rp=zp+r_e; % km
v = 11; % km/s
theta = 100; %degrees
theta = deg2rad(theta);% rad


[a,ecc,~,~,~,~,h,T,E]=joshCOE(rp,v,mu_e);
v_inf = sqrt(mu_e/-a);
r100 = (h^2/mu_e)/(1+ecc*cos(theta));
z100 = r100-r_e;
[vaz,vr,gamma] = joshVazVr(theta,ecc,h,mu_e);

disp("------------P2.36------------")
disp("My calculations have the following results:")
disp("Excess escape velocity: "+string(v_inf)+" km/s")
disp("H/C: Hyperbolic tragector should have positive excess escape velocity")
disp("r at theta = 100 degrees: "+string(r100)+" km")
disp("H/C: z > 250")
disp("Azmuthal velocity at theta = 100 degrees: "+string(vaz)+" km/s")
disp("H/C: Azmuthal velocity should be positive")
disp("Radial velocity at theta = 100 degrees: "+string(vr)+" km/s")
disp("H/C: At theta = 100 object is on departure so radial velocity should be
 positive")
```

# 2.37 meteroid

```
clear all;
mu_e = 398600;
r_e = 6378; % km
r0=402000;%km
theta=150;%deg
theta=deg2rad(theta);
v0 = 2.23;%km/s
```

```
E=(v0^2/2)-(mu_e/r0); % Energy
a = mu_e/(2*E);

syms h % set up h for symbolic solving
assume(h,'real')
assumeAlso(h>0)
eqn = sym(r0== (h^2/mu_e)*(1+(2*E*(h^2/mu_e^2)+1)^.5*cos(theta))^-1); %
 symbolic equation to solve for h
h = solve(eqn,h);
h = double(h);
ecc = (1/(r0*(mu_e/h^2))-1)/cos(theta); % solve for ecc

rp = a*(ecc-1);
zp = rp-r_e;

v = h/rp;

disp("------------P2.37------------")
disp("My calculations have the following results:")
disp("Eccentricity: "+string(ecc))
disp("H/C: For hyperbolic ecc should be above 1")
disp("Altitude at periapse: "+string(zp)+" km")
disp("H/C: Altitude at closest approach is lower than initial velocity but
 high enough to orbit")
disp("Velocity at closest approach: "+string(v)+" km/s")
disp("H/C: v at closest approach is higher than v initial")

% [a,ecc,~,~,~,~,h,T,E]=joshCOE(ri,vi,mu_e);
```

## 3.8 time above 400

```
clear all;
mu_e = 398600;
r_e = 6378; % km
ra = r_e + 600;
rp = r_e + 200;
a = (rp+ra)/2;
ecc = (ra-rp)/(rp+ra); % definition of ecc

h = sqrt(a*(1-ecc^2)*mu_e); % solve for h

r = 400 +r_e; % solve r
theta = acos((1/(r*(mu_e/h^2))-1)/ecc); % solve theta

[Me,Ean]=joshAnomalyCalculator(ecc,theta); % convert TA to Me
P=((2*pi)/sqrt(mu_e))*a^1.5;
n=sqrt(mu_e/a^3); % definition n
tsp = Me/n;
t400k = P-2*tsp; % time above 400 km
t400k = t400k/60;

disp("------------P3.8------------")
```

```
disp("My calculations have the following results:")
disp("Time spent over 400 km: "+string(t400k)+" min")
disp("H/C: time over 400 km is less than period")
disp("Intermediate H/C used:")
disp("      ecc < 1")
disp("      period resonable for LEO")
```

# 3.10

```
clear all;
mu_e = 398600;
rp = 10000; % km
P = 14*60*60;
a = ((sqrt(mu_e)/(2*pi))*P)^(2/3); % equation for a
ra = a*2-rp;
ecc = (ra-rp)/(rp+ra);
h = sqrt(a*(1-ecc^2)*mu_e);
v0= h/rp; % this is because v0 is at periapse and is perpendicular to r at
 periapse
r0 = rp;
pr = 0; % dot product of v0 and r0 as vectors, 0 b/c they are perpendicular

dt = 10*60*60;
coefs = 15; % 15 terms in stumpff functions
[Cc,Sc]=joshStumpffCoeffs(coefs); % generate stumpff coeffecients
C = @(z) sum(Cc.*joshStumpffZ(z,coefs)); % generate stumpff functions
S = @(z) sum(Sc.*joshStumpffZ(z,coefs));
[fX,fpX]=joshfChi(r0,v0,mu_e,a,dt,Cc,Sc,pr); % generate f and fp which can be
 put into a newton solver for X
X0 = sqrt(mu_e)*dt*abs(1/a); % initial guess
[X] = joshNewtons(fX,fpX,X0,1e-14); % newton solver for X

Ean = X/sqrt(a); % Eccentric anomaly
theta = 2*atan(tan(Ean/2)/sqrt((1-ecc)/(1+ecc)));
% [Me,Ean2] = joshAnomalyCalculator(ecc,theta)
% [~,Ean2]=joshQuadrant(Ean2)

r = (h^2/mu_e)/(1+ecc*cos(theta)); % radial distance
[vaz,vr] =joshVazVr(theta,ecc,h,mu_e); % azmuthal and radial velocity
speed = sqrt(vaz^2 + vr^2);
disp("------------P3.10------------")
disp("My calculations have the following results:")
disp("radial position: "+string(r)+" km")
disp("H/C: r is between rp and ra")
disp("speed: "+string(speed)+" km/s")
disp("H/C: MEO orbit resonable velocity seems beleivable")
disp("radial velocity: "+ string(vr)+" km/s")
disp("H/C: since period is 14hrs and this is 10hrs in, the orbit is past
 apoapse, so radial velocity should be negative")
```

# 3.20

```
clear all;
```

```
mu_e = 398600;
dt = 2*60*60;
r0 = [20 -105 -19]*1000; % km
v0 = [.9 -3.4 -1.5]; % km/s
[a]=joshCOE(r0,v0,mu_e);


%%%%%%%%% same as above %%%%%%%%%
coefs = 15;
[Cc,Sc]=joshStumpffCoeffs(coefs);
C = @(z) sum(Cc.*joshStumpffZ(z,coefs));
S = @(z) sum(Sc.*joshStumpffZ(z,coefs));
[fX,fpX]=joshfChi(r0,v0,mu_e,a,dt,Cc,Sc);
X0 = sqrt(mu_e)*dt*abs(1/a);
[X] = joshNewtons(fX,fpX,X0,1e-14);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


f = 1-(X^2/norm(r0))*C(X^2/a); % f and g functions are possible because v0 and
 r0 are vectors
g = dt-((1/sqrt(mu_e))*X^3*S(X^2/a));
r = f*r0+g*v0; % position
f_dot = (sqrt(mu_e)/(norm(r)*norm(r0)))*X*((X^2/a)*S(X^2/a)-1);
g_dot = 1-(X^2/norm(r))*C(X^2/a);
v = f_dot*r0 + g_dot*v0; % velocity

shouldBe1 = f*g_dot-f_dot*g; % H/C
disp("-------------P3.20-------------")
disp("My calculations have the following results:")
disp("The final position vector in km:")
disp(r)
disp("The final velocity vector in km/s:")
disp(v)
disp("H/C: f*g_dot-f_dot*g value: "+string(shouldBe1))
```

# 4.5

```
clear all;
mu_e = 398600;
R = [6.5 -7.5 -2.5]*1000;
V = [4 3 -3];
[a,ecc,theta,inc,raan,aop,h,T,E] = joshCOE(R,V,mu_e); % COEs
inc = rad2deg(inc);
raan = rad2deg(raan);
aop = rad2deg(aop);
theta = rad2deg(theta);
disp("-------------P4.5-------------")
disp("My calculations have the following results:")
disp("Semimajor axis: "+string(a)+" km")
disp("Eccentricity: "+string(ecc))
disp("True anomaly: "+string(theta)+" degrees")
disp("Inclination: "+string(inc)+" degrees")
disp("Right ascention of ascending node: "+ string(raan)+" degrees")
```

```
disp("Argument of periapse: "+string(aop)+" degrees")
disp("h: "+string(h)+"m^2/s")
```

# 4.7

```
clear all;
r = [-6.6 -1.3 -5.2]*1000;
ecc = [-.4 -.5 -.6];
mu_e = 398600;

theta = acos(dot(r,ecc)/(norm(ecc)*norm(r))); % solve for true anomaly
h = sqrt(mu_e*norm(r)*(1+norm(ecc)*cos(theta))); % use TA and get h
h = h*(cross(r,ecc)/norm(cross(r,ecc)));
inc = acos((dot([0 0 1],h))/norm(h)); % inclination degrees
inc = rad2deg(inc);

disp("------------P4.7------------")
disp("My calculations have the following results:")
disp("Inclination of this orbit: "+string(inc)+" degrees")
disp("H/C: r vector has a relatively large z component so this makes sense
 nominally")
```

# dependencies

```
depends = matlab.codetools.requiredFilesAndProducts("C:
\AERO351\A351HW2\HW2.m");
depends = depends';
%     {'C:\AERO351\A351HW2\HW2.m'                         }
%     {'C:\joshFunctionsMatlab\joshAnomalyCalculator.m'}
%     {'C:\joshFunctionsMatlab\joshCOE.m'                 }
%     {'C:\joshFunctionsMatlab\joshIsOnes.m'              }
%     {'C:\joshFunctionsMatlab\joshNewtons.m'             }
%     {'C:\joshFunctionsMatlab\joshStumpffCoeffs.m'       }
%     {'C:\joshFunctionsMatlab\joshStumpffZ.m'            }
%     {'C:\joshFunctionsMatlab\joshVazVr.m'               }
%     {'C:\joshFunctionsMatlab\joshfChi.m'                }
```

*Published with MATLAB® R2022a*

```
function [vaz,vr,gamma] = joshVazVr(theta,ecc,h,mu)
% gives magnitude of azmuthal velocity and radial velocity
% takes theta ecc and h
% optionally takes mu for the center body
% assumes shperical body and 2 body
% angles in rad
arguments
    theta (1,1) double {mustBeReal}
    ecc (1,1) double {mustBeReal, mustBeNonnegative}
    h (1,1) double {mustBeReal,mustBePositive}
    mu (1,1) double {mustBeReal} = 3.986004418 * (10^5) %km^3/s^2 mu_earth
end
vr  = (mu/h)*ecc*sin(theta);
vaz = (mu/h)*(1+ecc*cos(theta));
gamma = atan2(vr,vaz);
end
```

*Published with MATLAB® R2022a*

```
function [isOnes] = joshIsOnes(M)
% takes a value (persumably a logical type matrix) and returns true iff all
% entries are true
[m,n] = size(M);
isOnes = true;
for i = 1:m
    for j = 1:n
        if M(i,j) ~= 1
            isOnes = false;
        end
    end
end
end
```

*Published with MATLAB® R2022a*

```
function [a,ecc,theta,inc,raan,aop,h,T,E] = joshCOE(R,V,u,magOrVec)

%%%%%%%%%%%%%%%%%%
% Revamped to do rads and fit new naming convention %
%%%%%%%%%%%%%%%%%%


%COESOATESJOSHUA Takes postion and velocity vector and returns COES, all in
%ECI frame of reffrenece, km and seconds as units and degrees
% a = semi major axis
% ecc = eccentricity
% i = inclination
% raan = right accention acending node
% aop = argument of periapsis
% theta = true anomaly

% will return T in s as a period
% and E (sometimes epsilon) in km^2/s^2 as specific mechanical energy
% h is agular momentum

% magOrVec is a parameter that can be set for vector inputs to return
% vector h and ecc

% scalar entry should only be used if the spacecraft is at apoapse or
% periapse

arguments
    R {mustBeNumeric, mustBeReal}
    V  {mustBeNumeric, mustBeReal}
    u (1,1) {mustBeNumeric, mustBeReal, mustBePositive} = 3.986004418 *
 (10^5) %km^3/s^2
    magOrVec {mustBeMember(magOrVec,{'magnitude','vector'})} = 'magnitude'
end

[m1,n1] = size(R);
[m2,n2] = size(V);

if joshIsOnes([m1 n1 m2 n2])
    magOrVec = 'magnitude';
    R = [0 0 R];
    V = [V 0 0];
    warning("joshCOE will assume that R and V are normal if the inputs are
 scalar ie: the craft is in a circular orbit or is at periapse or apoapse")
elseif (~joshIsOnes([m1 n1] == [m2 n2]))|~((n1==1&m1==3)|(n1==3&m1==1))
    throw(MException("COEsOatesJoshua:invalidInput","R and V must be either
 1x3 vectors or scalars"))
end

% uearth = 3.986004418(8) x 10^14 m^3/s^2
ihat=[1,0,0];
khat=[0,0,1];
```

```
Rm = norm(R);
Vm = norm(V);

%calculate orbital constants
h = cross(R,V); %angular momentum vector
hm = norm(h);
E = ( ( Vm^2 ) / 2) - ( u / Rm ) ; %specfic mechanical energy
%calculate COEs

a = -u / (2 * E ); %semi major axis in km
T = 2*pi*sqrt((a^3)/u); %period in s

e = (1/u) * (((Vm^2)-(u/Rm) ) * R - (dot(R,V) * V)); %eccentricity vector
em = norm (e); %magnitude of e

inc = acos((dot(khat,h))/hm); %inclination
n = cross(khat,h); %node vector
nm = norm(n); %magnitude n

%raan
raan = acos(dot(ihat,n)/nm);
if n(2) < 0  %checks the vector relative to j to see if angle is positive or
 negative
    raan = 360 - raan;
end

%aop
aop = acos(dot(n,e)/(nm*em));
if e(3) < 0
    aop = 360 - aop;
end

%theta
theta = acos(dot(e,R)/(em*Rm));
if(dot(R,V) < 0) % cehck flight path angle to see if it is postive or negative
    theta = 360 -theta;
end

if strcmp(magOrVec, 'magnitude') %for magnitude mode, vectors will not be
 returned
    h = hm;
    e = em;
    if joshIsOnes([m1 n1 m2 n2]) % for scalar inputs it is not possible to
 calculate these values
        theta = NaN;
        inc = NaN;
        raan = NaN;
        aop = NaN;
    end
end
ecc = e;

end
```

*Published with MATLAB® R2022a*

```
function [fX,fpX] = joshfChi(r0,v0,mu,a,dt,C,S,pr)
% note fX = sqrt(mu)*dt - integral(rdX)
% fpX = -r

% it is highly recommended that C and S are passed in
arguments
    r0 double {mustBeReal}
    v0 double {mustBeReal}
    mu (1,1) double {mustBeReal}
    a (1,1) double {mustBeReal}
    dt (1,1) double {mustBePositive}
    C (1,:) double {mustBeReal} = nan
    S (1,:) double {mustBeReal} = nan
    pr (1,1) double {mustBeReal} = nan
end

if isnan(C)|isnan(S)
    [C,S] = joshStumpffCoeffs();
end

[m1,n1] = size(r0);
[m2,n2] = size(v0);

if joshIsOnes([m1 n1 m2 n2])
    if isnan(pr)
        throw(MException("joshfChi:invalidInput","If r0 and v0 are given as
 vectors, pr should be the dot product of them"))
    end

elseif(~joshIsOnes([m1 n1] == [m2 n2]))|~((n1==1&m1==3)|(n1==3&m1==1))
    throw(MException("joshfChi:invalidInput","r0 and v0 must be either 1x3
 vectors or scalars"))
else
    pr = dot(r0,v0);
end


coefs = length(C);
if length(S)~=coefs
    throw(MException("joshfChi:invalidInput","S and C should be the same
 length"))
end

sm = sqrt(mu);
r0 = norm(r0);
% v0 = norm(v0);

% fX = @(X) (pr/sm)*X^2*C(z) + (1-a*r0)*X^3*S(z) + r0*X - sm*dt;
% fpX = @(X) (pr/sm)*X*(1-a*X^2*S(z)) + (1-a*r0)*X^2*C(z) + r0;
% (sum(C.*joshStumpffZ(X^2/a)))
```

```
% fX = @(X) (pr/sm)*X^2*(sum(C.*joshStumpffZ(X^2*a,coefs))) + (1-
a*r0)*X^3*(sum(S.*joshStumpffZ(X^2*a,coefs))) + r0*X - sm*dt;
% fpX = @(X) (pr/sm)*X*(1-a*X^2*(sum(S.*joshStumpffZ(X^2*a,coefs)))) + (1-
a*r0)*X^2*(sum(C.*joshStumpffZ(X^2*a,coefs))) + r0;

a = 1/a;
fX = @(X) (pr/sm)*X^2*sum(C.*joshStumpffZ(X^2*a,coefs)) + (1-
a*r0)*X^3*sum(S.*joshStumpffZ(X^2*a,coefs)) + r0*X - sm*dt;
fpX = @(X) (pr/sm)*X*(1-a*X^2*sum(S.*joshStumpffZ(X^2*a,coefs)))+(1-
a*r0)*X^2*sum(C.*joshStumpffZ(X^2*a,coefs))+r0;



% X^2*(sum(S.*joshStumpffZ(((X^2)/a)))) = X^2*C(z)
%
%
end
```

*Published with MATLAB® R2022a*

```
function [M,E] = joshAnomalyCalculator(ecc,theta)
% M will be Me,Mp or Mh depending on ecc
% E will be Eccentric Anomoly when applicable or F: hyperbolic Ecctric
% anomoly. E will be set to
% values in Rads
arguments
    ecc (1,1) double {mustBeReal}
    theta (1,1) double {mustBeReal}
end

if ecc <1 % Me & E
    E = 2*atan(sqrt((1-ecc)/(1+ecc))*tan(theta/2)); % definintion of E,
 rewriten to solve E
    M = E-ecc*sin(E); % definition of M
elseif ecc > 1 % Mh & F
    E = log((sqrt(ecc+1)+sqrt(ecc-1)*tan(theta/2))/(sqrt(ecc+1)-
sqrt(ecc-1)*tan(theta/2)));
    M = ecc(sinh(F)-F);
else % ecc == 1 Mp
    E = nan; % This is a rare case and E doesnt have a definition for ecc == 1
    M = .5*tan(theta/2)+(1/6)*tan(theta/2)^3;
end
end
```

*Published with MATLAB® R2022a*

```
function [r, count, xVector, errorVector, errorRatioVector] = joshNewtons(f,
 fp, x0, TOL, maxI)

% Aero300 code updated 10/12
% see A300 HW3

% this function will check if newtons method will converge
% this function takes a diff'able function and its derivative as well as a
% guess and a tolerance.
% this function is not gaurranted to converge and will exit early if it
% does not returning an empty array for r

% this algorithim taken from canvas has been cleaned up for readablity and
% commented for understanding

arguments
    f
    fp
    x0 (1,1) {mustBeNumeric,mustBeReal}
    TOL (1,1) {mustBeNumeric,mustBeReal} = .0001
    maxI (1,1) {mustBeNumeric,mustBeReal,mustBePositive,mustBeInteger} = 200
end

if ~(isa(f,'function_handle')&isa(fp,'function_handle'))
    throw(MException("joshNewtons:invalidInput":"f and fp must be function
 handles"))
end

x1 = x0 - f(x0)/fp(x0); % create x1 for use in algorithim
count = 1;
xVector = [x0; x1];
error = abs(x1 - x0);
errorVector = error;
errorRatioVector = [];

while (error > TOL) && (count <= maxI)
    % set x0 to x1 and recalculate x1 using function and functionPrime
    x0 = x1;
    x1 = x1 - f(x0)/fp(x0); % newtons algorithim

    count = count + 1;

    % for the sake of the error ratio compute errorSquared
    errorSquared = error^2; % this is the previous error since it hasnt been
 updated
    error = abs(x1 - x0); % update error
    errorRatio = error/errorSquared; % update errorRatio

    xVector = [xVector; x1]; % append new data to old
    errorVector = [errorVector; error];
    errorRatioVector = [errorRatioVector; errorRatio];
end
```

```
if (count > maxI)
    r = NaN; % r DNE for these inputs
    warn = "Did not converge in " + num2str(maxI) + " intervals";
    warning(warn)
elseif isnan(error)
    r = NaN; % r DNE for these inputs
    warn = "Value exploded in "+ num2str(maxI)+ " intervals";
    warning(warn)
else
    r = xVector(end); % last guess is also best prediction for the root
end
```

*Published with MATLAB® R2022a*

```
function [C,S] = joshStumpffCoeffs(n)
% AERO 351 code
% Generates the first n terms of the stumpff coeffcients for @S(z) and @C(z)
 in a vector

% these coeffs are used for the universal variable approach to orbital
 mechanics

% for use as companion function with joshStrumpffZ
% coeffs should be saved to workspace and reused to save compute time
% @S(z) == sum(S.*Z) == polyval(flip(S),z) : where Z = [z^0 z^1 ... z^n]
% @C(z) == sum(C.*Z) == polyval(flip(C),z) : where Z = [z^0 z^1 ... z^n]
arguments
    n (1,1) {mustBePositive,mustBeInteger} = 15;
end

C = zeros(1,n);
S = C;
for i = 1:n
    k = i-1;
    C(i) = (-1)^k*(1/factorial(2*k+2));
    S(i) = (-1)^k*(1/factorial(2*k+3));
end
end
```

*Published with MATLAB® R2022a*

```
function [Z] = joshStumpffZ(z,n)
% AERO 351 code
% Generates the first n terms of the stumpff coeffcients for @S(z) and @C(z)
 in a vector
% for use as companion function with joshStrumpffCoeffs
% @S(z) == sum(S.*Z) == polyval(flip(S),z) : where S given by (-1)^k*(1/
factorial(2*k+2))
% @C(z) == sum(C.*Z) == polyval(flip(C),z) : where C given by (-1)^k*(1/
factorial(2*k+3))
arguments
    z (1,1)
    n (1,1) {mustBePositive,mustBeInteger} = 15;
end
Z = ones(1,n);
for i = 2:n
    Z(i) = z*Z(i-1);
end
end
```

*Published with MATLAB® R2022a*