Numerical Mini-Group Project.
Solving Conical Flow using R-K and Modified Euler Methods

Dr. Joseph Homer Saleh

By: Neilay Amin, Wonhee Lee, Rohan Malik

AE 4803 NUM

Spring 2022

Abstract

This report investigates the faster than sound phenomenon known as conical waves. The shocks that result from them affect the supersonic and hypersonic aircraft responsible for them. Implementing numerical methods allow for a different approach to calculating the factors of the shocks. With the use of MATLAB, a program was created based on the Runge-Kutta method and modified Euler methods. The Taylor-Maccoll ODEs were computed and executed with the methods, giving out the desired characteristics of a conical shock, such as the shock angle. With the use of the function program, various scenarios were tested, and the appropriate plots. It was found that the Euler method delivered desirable results but the R-K was more accurate.

Introduction

This mini-project entails solving a common aerospace problem with the use of numerical methods learned in this class. The subject is conical flows, which are ever-present and vital for analyzing supersonic and hypersonic aircraft. The shock wave can be estimated with Taylor-Maccoll ODEs. Both the Runge-Kutta method and the modified Euler method are employed to provide a unique approach. MATLAB was employed for the operating of these functions. Several test cases were iterated numerous times to test the validity of the program and whether the results were acceptable.

Background

Conical Flow

For this analysis context is first provided by going over the methodology and implementation of Taylor- MacColl equations in MATLAB used to solve conical shock problems. These equations consider when a cone with a known angle is placed into supersonics flow, the shock angle that will be produced can be estimated. These are vital factors to know for supersonic and hypersonic vehicles as they employ conical shapes to travel at high speeds. The equations below are the basic necessary ODE needed to establish the proper physical relations. With the free stream Mach number, and initial shock wave angle, we can calculate the velocity, deflection angle, and the Mach number of the shock wave from these relations. From those equations and their outputs, we then find the radial and tangential velocities behind the shock. All these values serve as boundary conditions to solve the Taylor-Maccoll equation below.

$$M_2^2 = \frac{1 + \frac{\gamma - 1}{2} M_1^2}{\gamma M_1^2 \sin^2\ \theta_s - \frac{(\gamma - 1)}{2}} + \frac{M_1^2 \cos^2\ \theta_s}{1 + \frac{\gamma - 1}{2} M_1^2 \sin^2\ \theta_s}$$

Equation 1: Downstream Mach

$$\bar{V}_2 = \frac{1}{\sqrt{1 + \frac{2}{(\gamma - 1)M_2^2}}}$$

Equation 2: Normalized Velocity

$$\tan\ \delta_s = \cot\ \theta_s \frac{M_1^2 \sin^2\ \theta_s - 1}{\frac{\gamma + 1}{2} M_1^2 - (M_1^2 \sin^2\ \theta_s - 1)}$$

Equation 3: Deflection Angle

Taylor-Maccoll Equation

$$\bar{v}_\theta = -\bar{V}_{n2} = -\bar{V}_2 \sin(\theta_s - \delta_s)$$
$$\bar{v}_p = \bar{V}_t = \bar{V}_2 \cos(\theta_s - \delta_s)$$
$$\frac{d\bar{v}_r}{d\theta} = \bar{v}_\theta$$
$$\frac{d\bar{v}_\theta}{d\theta} = f(\bar{v}_r, \bar{v}_\theta)$$

$$f(\bar{v}_r, \bar{v}_\theta) = -\bar{v}_r + \frac{\frac{\gamma - 1}{2}\left[1 - \left(\bar{v}_r^2 + \bar{v}_\theta^2\right)\right](\bar{v}_r + \bar{v}_\theta \cot\ \theta)}{\left\{\bar{v}_\theta^2 - \frac{\gamma - 1}{2}\left[1 - \left(\bar{v}_r^2 + \bar{v}_\theta^2\right)\right]\right\}}$$

Modified Euler Method

To solve the Taylor-Maccoll equation a modified Euler method is used. It is very similar to Heun's method in that it uses a predictor and corrector scheme to solve for the cone angle.

Shown below is the process that is implemented within MATLAB to find the cone angle.

The predictor and corrector steps are shown below.

$$(\bar{\boldsymbol{v}}_{\boldsymbol{r}})_{\boldsymbol{p}} = (\bar{\boldsymbol{v}}_{\boldsymbol{r}})_{\boldsymbol{i}} - (\bar{\boldsymbol{v}}_{\boldsymbol{\theta}})_{\boldsymbol{i}} \boldsymbol{\Delta\theta}$$
$$(\bar{\boldsymbol{v}}_{\boldsymbol{\theta}})_{\boldsymbol{p}} = (\bar{\boldsymbol{v}}_{\boldsymbol{\theta}})_{\boldsymbol{i}} - \boldsymbol{f}[(\bar{\boldsymbol{v}}_{\boldsymbol{r}})_{\boldsymbol{i}}, (\bar{\boldsymbol{v}}_{\boldsymbol{\theta}})_{\boldsymbol{i}}] \boldsymbol{\Delta\theta}$$

Equation 4: Predictor Step

$$(\bar{\boldsymbol{v}}_{\boldsymbol{r}})_{\boldsymbol{i+1}} = (\bar{\boldsymbol{v}}_{\boldsymbol{r}})_{\boldsymbol{i}} - \frac{(\bar{\boldsymbol{v}}_{\boldsymbol{\theta}})_{\boldsymbol{p}} + (\bar{\boldsymbol{v}}_{\boldsymbol{\theta}})_{\boldsymbol{i}}}{\mathbf{2}} \boldsymbol{\Delta\theta}$$
$$(\bar{\boldsymbol{v}}_{\boldsymbol{\theta}})_{\boldsymbol{i+1}} = (\bar{\boldsymbol{v}}_{\boldsymbol{\theta}})_{\boldsymbol{i}} - \frac{\boldsymbol{f}[(\bar{\boldsymbol{v}}_{\boldsymbol{r}})_{\boldsymbol{i}}, (\bar{\boldsymbol{v}}_{\boldsymbol{\theta}})_{\boldsymbol{i}}] + \boldsymbol{f}\left[(\bar{\boldsymbol{v}}_{\boldsymbol{r}})_{\boldsymbol{p}}, (\bar{\boldsymbol{v}}_{\boldsymbol{\theta}})_{\boldsymbol{p}}\right]}{\mathbf{2}} \boldsymbol{\Delta\theta}$$

Equation 5: Corrector Step

This modified Euler method is very similar to the Heun Method other than the subtraction in the predictor and corrector steps. This is because rather than integrating from the cone to the shock angle the solution is to start from the shock and go to the cone angle. This means to forward step in the method a subtraction is needed. Figure 1 is a visual depiction of the process.

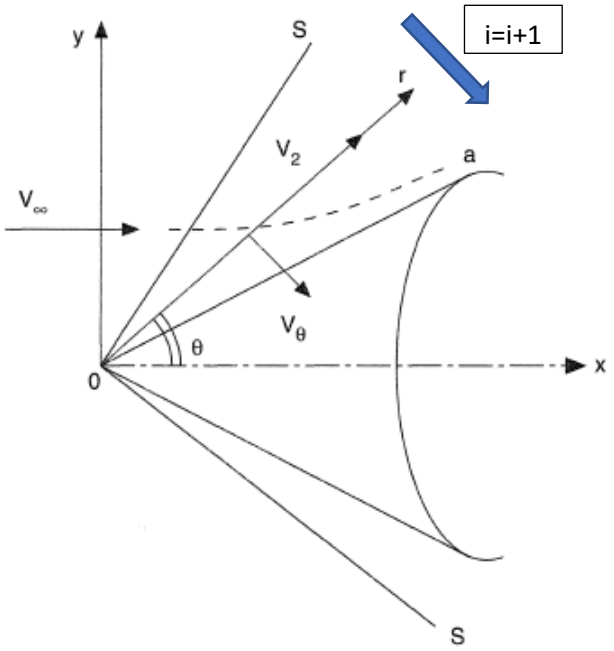

Figure 1. Diagram of Conical Flow and its Solving Process

The cone angle is determined by the halt condition of $(\bar{v}_\theta)_p = 0$ as the there can be no velocity going into the cone as it is a surface. The step size of 0.1 degrees was chosen which was converted into radians. This method will deliver a local error of $O(h^3)$ and a global error of $O(h^2)$.

Runge-Kutta Method

The Runge-Kutta method of n^{th} order approximates the solution to an ODE by taking a weighted average of gradients evaluated at n points within one step. In this report, the most common 4^{th} order Runge-Kutta method was used and is given as Equation 6.

$$k = \frac{1}{6}(k_1 + 2 \cdot k_2 + 2 \cdot k_3 + k_4)$$

Where,

$$\boldsymbol{k_1 = f(x_i, \vec{y}_i)}$$

$$\boldsymbol{k_2 = f\left(x_i + \frac{1}{2}h, \vec{y}_i + \frac{1}{2}h \cdot k_1\right)}$$

$$\boldsymbol{k_3 = f\left(x_i + \frac{1}{2}h, \vec{y}_i + \frac{1}{2}h \cdot k_2\right)}$$

$$\boldsymbol{k_4 = f(x_i + h, \vec{y}_i + h \cdot k_3)}$$

Equation 6: Constant Step 4th order Runge-Kutta Method

Runge-Kutta Method: Adaptive Step Size

The Runge-Kutta method described above may be equipped with the adaptive step sizing feature. Adaptive step size allows to capture more accurate solutions when there are drastic changes independent variables over a short range of independent variables. It is not a very suitable method for the Taylor-McColl equation, considering the gradual nature of the solution. Still, in this report, the 4th order adaptive step size Runge-Kutta Method was implemented to investigate its behavior compared to other methods. There are numerous ways to evaluate the next step size. One way is to use two-step sizes to evaluate the next step solution, and by comparing them, one can reasonably find the approximate expression for the local truncation error. In this report, two steps with one twice the other were implemented to the 4th order Runge-Kutta method given in Equation 0, and for this specific case, the next step size can be determined for the desired tolerance as in Equation 1.

$$\boldsymbol{h_{new} = h \cdot \left|\frac{15}{16} \cdot \frac{\varepsilon_{tol}}{y(h_2) - y(h_1)}\right|^{1/5}}$$

Where,

$\boldsymbol{h_2 = \frac{1}{2} \cdot h_1}$ and $\boldsymbol{\varepsilon_{tol}}$ is the tolerance for the local truncation error, which was set to 10^{-10} for this report.

Equation 7: Step Size Update for the 4th order Runge-Kutta Method

$\boldsymbol{h_2 = \frac{1}{2} \cdot h_1}$ and $\boldsymbol{\varepsilon_{tol}}$ is the tolerance for the local truncation error, which was set to 10^{-10} for this report

Solution Procedure

The implementation of the R-K 4th order and Modified Euler was done in MATLAB. To compare the two methods 4 test cases were performed in which the accuracy and the number of iterations were tabulated. Shown below is the Modified Euler implementation.

```
while Vt_ipl < 0
    %Predictor Stepping
    Vr_p = Vr_i - (Vt_i .* steprad);
    V_theta_p = Vt_i - (Taymac(theta_g,Vt_i,Vr_i).*steprad);
    %Corrector Stepping
    Vr_ipl = Vr_i - ((V_theta_p + Vt_i)./2) .* steprad;
    Vt_ipl = Vt_i - ((Taymac(theta_g,Vt_i,Vr_i) + Taymac(theta_g, V_theta_p,Vr_p))./2).*steprad;
    Vr_i = Vr_ipl;
    Vt_i = Vt_ipl;
    theta_g = theta_g - .1;
    j = j + 1;
```

The while loop enforces the halt condition as when $\bar{v}_\theta$ changes from negative to positive the cone angle has been found. The predictor step and corrector steps are identical to the equations in the section above. Afterward, the shock angle is updated and the iteration is counted.

The overview of the implanted Runge-Kutta method is as follows.

```
k1 = f(x, v);
k2 = f(x + (h / 2), v + (h / 2) .* k1);
k3 = f(x + (h / 2), v + (h / 2) .* k2);
k4 = f(x + h, v + h .* k3);

k = (1 / 6) .* (k1 + 2 .* k2 + 2 .* k3 + k4);
v = v + h .* k;

% march
x = x + h;
```

The overview of the implanted adaptive step size Runge-Kutta method is as follows.

```
% 1. Step Size: h1 = h
h1 = h;
v1 = v;

k1 = f(x, v1);
k2 = f(x + (h1 / 2), v1 + (h1 / 2) .* k1);
k3 = f(x + (h1 / 2), v1 + (h1 / 2) .* k2);
k4 = f(x + h1, v1 + h1 .* k3);

k = (1 / 6) .* (k1 + 2 .* k2 + 2 .* k3 + k4);
v1 = v1 + h1 .* k;

% 2. Step Size: h2 = h1 / 2
h2 = h1 / 2;
v2 = v;

for J = 1 : 2
    k1 = f(x, v2);
    k2 = f(x + (h2 / 2), v2 + (h2 / 2) .* k1);
    k3 = f(x + (h2 / 2), v2 + (h2 / 2) .* k2);
    k4 = f(x + h2, v2 + h2 .* k3);

    k = (1 / 6) .* (k1 + 2 .* k2 + 2 .* k3 + k4);
    v2 = v2 + h2 .* k;

end
v = v1;

% Update Step Size
h = h1 * (abs((15 / 16) * epsTol / max(v2 - v1))) ^ (1 / 5);

% march
x = x + h;
```

To compare the two methods shown above a total of 4 test cases were conducted. The 4 cases go over a 15- and 30-degree cone at Mach 2 and 5. For a baseline datum, the Conical Flow calculator by Stephen Krauss is used. This is because the calculator solves the Taylor-Maccoll equations to a very high degree of accuracy. Using the calculated shock angle from the calculator it is then put into our MATLAB codes to see the deviation in the calculated cone and iteration count. This will allow us to see which method is more accurate and takes less computational power. Additionally, plots of the Mach number versus the shock angle are shown. These plots are useful because they can help determine the flow properties' between the shock and the surface. This is done by using the isentropic flow relations.

Results & Discussion

Table 1. Simulation Result for Modified Euler Method.

	15 Degree		30 Degree	
Mach	2	5	2	5
Cone Angle	15.0805	15.07	30.06	29.91
Percent Error	0.536667%	0.46667%	0.2%	0.3%
Iter. Count	189	51	181	57

Table 2. Simulation Result for Runge-Kutta Method.

	15 Degree		30 Degree	
Mach	2	5	2	5
Cone Angle	14.9305	14.9511	30.0112	29.9356
Percent Error	0.4633%	0.3260%	0.0373%	0.2147%
Iter. Count	190	51	181	57

Table 3. Simulation Result for Adaptive Step Runge-Kutta Method.

	15 Degree		30 Degree	
Mach	2	5	2	5
Cone Angle	15.0366	15.0037	30.1092	29.9229
Percent Error	0.2440%	0.0247%	0.3640%	0.2570%
Iter. Count	19,103	3,918	9,947	2,353

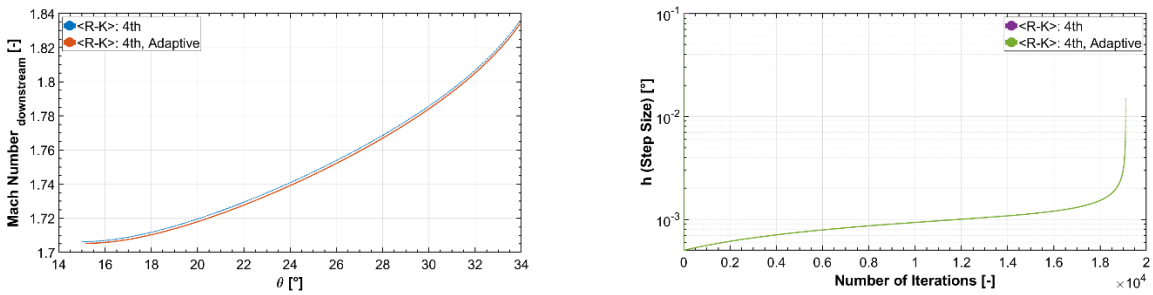

Figure 2. Downstream Mach Number against Tangential Direction and Step Size against Number of Iterations for 4th order Runge-Kutta Method and its Adaptive Step Size Version for the Test Case of Upstream Mach Number of 2 and Expected Cone Angle of 15°.

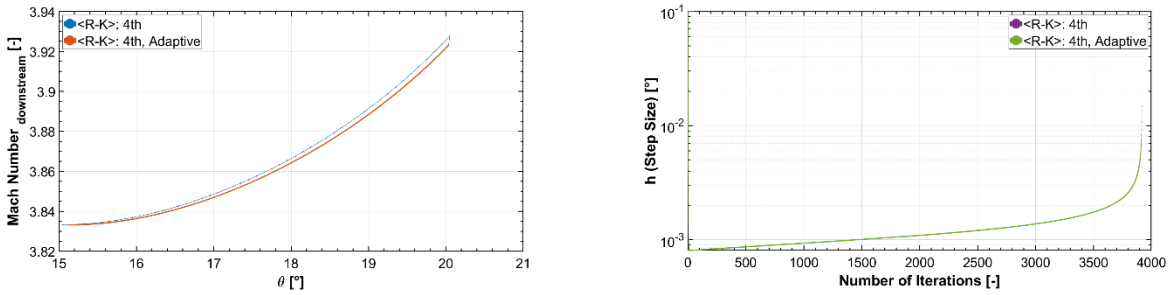

Figure 3. Downstream Mach Number against Tangential Direction and Step Size against Number of Iterations for 4th order Runge-Kutta Method and its Adaptive Step Size Version for the Test Case of Upstream Mach Number of 2 and Expected Cone Angle of 30°.

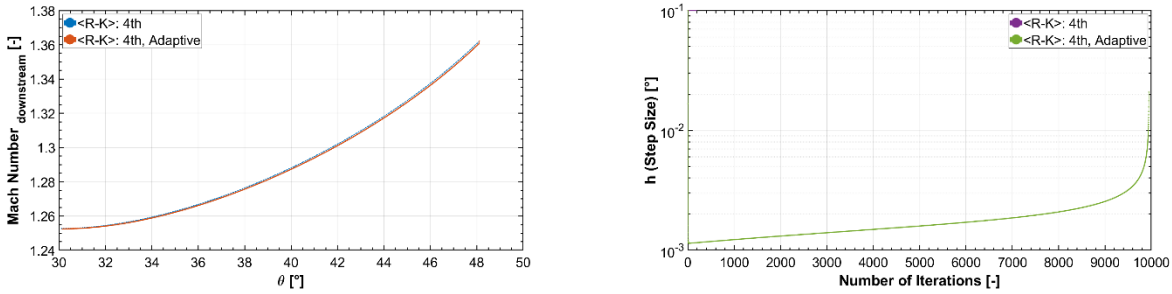

Figure 4. Downstream Mach Number against Tangential Direction and Step Size against Number of Iterations for 4th order Runge-Kutta Method and its Adaptive Step Size Version for the Test Case of Upstream Mach Number of 5 and Expected Cone Angle of 15°.

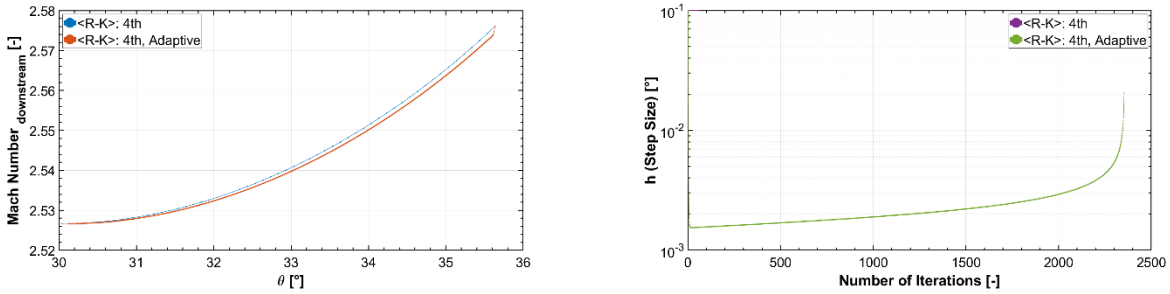

Figure 5. Downstream Mach Number against Tangential Direction and Step Size against Number of Iterations for 4th order Runge-Kutta Method and its Adaptive Step Size Version for the Test Case of Upstream Mach Number of 5 and Expected Cone Angle of 30°.

Tables 1 and 2 show that the 4th stage R-K was the most accurate while keeping the same number of iterations as the Modified Euler Method. However, the Modified Euler method is less

taxing on the computer's resources due to the smaller number of operations being done. But the added operations in the R-K help aid the accuracy.

According to Table 2 and Table 3, the adaptive Runge-Kutta method does not provide a clear advantage over the constant step size one. The percentage errors for the 15-degree cases are significantly lower for the former, but for the 30-degree cases, the latter prevails. Also, the computational cost for the former is much higher.

One reason for the poor performance of the adaptive step-size method can be deduced from the problem statement. The termination condition for the problem is condition-based, not domain-based. Therefore, the resulting cone angle is very sensitive to the last part of the iteration process. According to Figures 2, 3, 4, and 5, the step size grows at a very fast rate in the last part of the iteration, and this behavior may affect the accuracy of the numerical solution.

However, the more serious problem is the initial giant drop in the step size. According to Figures 2, 3, 4, and 5, the Mach number obtained from the adaptive step-size method shows a notable, abnormal initial deviation from that from the constant step size method. This behavior is more severe for the higher cone angle. Therefore, some modifications to update the step size seems needed to deradicalize the behavior of the numerical scheme.

Conclusion

In conclusion, the conical flow problem can easily be tackled using numerical methods. It was found that the modified Euler method was less accurate when compared to R-K but was easier to compute. This application of numerical methods allows for the computation of flow properties to be found with ease. With the flow properties known design conditions can be found which help the engineer immensely.

As for the adaptive step-size method, its effectiveness compared to the constant step size method really depends on the problem statement as well as the initial conditions. Abrupt changes in the step size can cause unexpected behavior for the numerical scheme and, in turn, a faulty solution.

Therefore, it is important to use a variety of numerical methods for their suited situations. To solve the Taylor McColl equation, the modified Euler method may be used to obtain the initial estimation of the shock angle from the desired cone angle. Then, for cone angles less than some critical point between 15 and 30 degrees, the adaptive step size Runge-Kutta method is a good

result. Otherwise, the constant step size can be adopted. Still, if the computational cost is affordable, crosscheck of results among various numerical methods is recommended.

References

Krauss, Stephen. “Conical Flow Calculator .” *Compressible Aerodynamics Calculator*, http://www.dept.aoe.vt.edu/~devenpor/aoe3114/calc.html.

Polezhaev, Yury V., and I.V. Chircov. “Inviscid Flow.” *THERMOPEDIA*, Begel House Inc., https://www.thermopedia.com/content/883/.

Appendix A: Code

Modified Euler Method

```
clc
clear all
close all
global g
g = 1.4;
guess =   35.6355638;
ConeAngleDiff=1;
wantedcone=30;
M1=5;
[ShockAngle, ConeAngle, iter] = conesolver(M1,guess);
i=1;
%%
while ConeAngleDiff > 0.001
    [ShockAngle, ConeAngle, iter] = conesolver(M1,guess);
    jtemp(i)=iter;
    if ConeAngle > wantedcone
        guess=guess-.01;
    else
        guess=guess+.01;
    end
    ConeAngleDiff = abs(ConeAngle-wantedcone)/wantedcone;
    i=i+1;
end
sum(jtemp)

function [ShockAngle, ConeGuess, iter] = conesolver(M,theta_s)
global g
%Initialize the Shock angle guess
theta_g = theta_s;
thetarad = theta_s .* (pi./180);
steprad = .1 .* pi./180;
j = 0;
%Solve for downsteam mach, deflection angle and vmax
%Mach calculation
num1 = 1 + ((g-1)./2) .* (M.^2);
den1 = (g .* (M.^2) .* (sind(theta_s)).^2) - (g-1)./2;
A = num1./den1;
```

```
num2 = M.^2 .* (cosd(theta_s)).^2;
den2 = 1 + ((g-1)./2).*(M.^2).*(sind(theta_s)).^2;
B = num2./den2;
M2 = sqrt(A + B)
%finding the vmax
V2 = 1./sqrt(1+ (2./((g-1).* M2.^2)));
num = M^2 .* sind(theta_s).^2 - 1;
den = ((g+1)./2).*M.^2 - num;
int = (1/tand(theta_s)).* (num./den);
D = atand(int);
%getting inital for vt and vr
Vt_i = -V2 .* sind(theta_s - D);
Vr_i = V2 .* cosd(theta_s - D);
Vt_ip1 = Vt_i;
Vr_ip1 = Vr_i;

%looping through
while Vt_ip1 < 0
    %Predictor Stepping
    Vr_p = Vr_i - (Vt_i .* steprad);
    V_theta_p = Vt_i - (Taymac(theta_g,Vt_i,Vr_i).*steprad);
    %Corrector Stepping
    Vr_ip1 = Vr_i - ((V_theta_p + Vt_i)./2) .* steprad;
    Vt_ip1 = Vt_i - ((Taymac(theta_g,Vt_i,Vr_i) + Taymac(theta_g,
V_theta_p,Vr_p))./2).*steprad;
    Vr_i = Vr_ip1;
    Vt_i = Vt_ip1;
    theta_g = theta_g - .1;
    j = j + 1;
end
ConeGuess = (theta_g + (theta_g +.1))./2;
ShockAngle = theta_s;
iter=j;
end
function cheese = Taymac(theta,vt,vr)
global g
num = ((g-1)/2)*(1-(vr.^2+vt.^2))*(vr + vt*cotd(theta));
dem = (vt.^2 - (g-1)/2 * (1-(vr.^2 + vt.^2)));
cheese = -vr+(num/dem);
end
%This is the Taylor Mccoll eqs
```

4th order Runge-Kutta and its Adaptive Step Size Version

```
%% Flow Condition
gma = 1.4;          % [-]   Specific Heat Ratio

% Simplification
pu = (gma - 1) / 2;

M1 = 2;             % [-]   Inlet Mach Number
phiShock = 33.9304904;      % [deg] Shock Wave Angle % 39.7841

% Unit Conversion
phiShock = phiShock * (pi / 180);
```

```
%% Initial Conditions
% delta: Flow Angle right After the Shock (Oblique Shock Relations)
delta = atan2((2 * cot(phiShock) * (M1^2 * (sin(phiShock))^2 - 1)),...
        (M1^2 * (gma + cos(2 * phiShock))) + 2);

% M2   : Flow Speed right After the Shock (Oblique Shock Relations)
Mn1 = M1 * sin(phiShock);
Mn2 = sqrt((1 + pu * Mn1^2) / (gma * Mn1^2 - pu));
M2  = Mn2 / sin(phiShock - delta);

% Normalized Speed
V_2    = MtoV_(M2, pu);
V_r2   =   V_2 * cos(phiShock - delta);
V_phi2 = - V_2 * sin(phiShock - delta);


%% ODE Info.
% ODE: <Taylor-Maccoll>
% x is phi
% y is V_r
% z is dV_r / dphi
syms x y z v
v = [y z]';
f = @(x, v) [v(2)
             ((1 / pu) * v(1) * v(2)^2 - (1 - v(1)^2 - v(2)^2) * (2 * v(1) +
v(2) * cot(x))) /...
             (1 - v(1)^2 - v(2)^2 * (1 + (1 / pu)))];

% Bounds
a = phiShock;      v_a = [V_r2 V_phi2]';


%% Numerical Method Settings
% Step Size: phi
h = - 10 ^ (- 1) * (pi / 180);


%% Main
% ------------------------------------------------------------------------
% <R-K> 4th Order
[phiConeRK4, xListRK4, vListRK4, numIterRK4] =...
rk4(h, v, f, a, v_a);

% V_ & M
V_ListRK4   = sqrt(vListRK4(:, 1).^2 + vListRK4(:, 2).^2);
MListRK4    = V_toM(V_ListRK4, pu);


% <R-K> 4th Order, Adaptive
epsTol = 10 ^ (- 10); % 12.6
[phiConeRK4A, xListRK4A, vListRK4A, hListRK4A, numIterRK4A] =...
rk4Adapt(h, v, f, a, v_a, epsTol);
```

```
% V_ & M
V_ListRK4A  = sqrt(vListRK4A(:, 1).^2 + vListRK4A(:, 2).^2);
MListRK4A   = V_toM(V_ListRK4A, pu);


% ------------------------------------------------------------------------
%% Helpers
function V_ = MtoV_(M, pu)

    V_ = (1 ./ (pu .* M.^2) + 1).^(- 1 / 2);

end

function M = V_toM(V_, pu)

    M  = sqrt(1 ./ (pu .* (V_.^(- 2) - 1)));

end

function [phiCone, xList, vList, numIter] = rk4(h, v, f, a, v_a)

    % Iteration
    xList = [];
    vList = [];

    I = 0;
    while true

        if I == 0
            x = a;
            v = v_a;

        else
            k1 = f(x, v);
            k2 = f(x + (h / 2), v + (h / 2) .* k1);
            k3 = f(x + (h / 2), v + (h / 2) .* k2);
            k4 = f(x + h, v + h .* k3);

            k = (1 / 6) .* (k1 + 2 .* k2 + 2 .* k3 + k4);
            v = v + h .* k;

            % march
            x = x + h;

        end

        % V_phi
        V_phi = v(2);

        if V_phi > 0
            break;
        end
```

```
        % log
        xList = [xList x];
        vList = [vList v];

        % Count Iterations
        I = I + 1;

    end

    % Transpose
    xList = xList' * (180 / pi);
    vList = vList';

    % Cone Angle & Unit Conversion
    phiCone = x * (180 / pi);

    % Number of Iterations
    numIter = I;

end

function [phiCone, xList, vList, hList, numIter] = rk4Adapt(h, v, f, a, v_a,
epsTol)

    % Iteration
    xList = [];
    vList = [];
    hList = [];

    I = 0;
    while true

        if I == 0
            x = a;
            v = v_a;

        else
            % 1. Step Size: h1 = h
            h1 = h;
            v1 = v;

            k1 = f(x, v1);
            k2 = f(x + (h1 / 2), v1 + (h1 / 2) .* k1);
            k3 = f(x + (h1 / 2), v1 + (h1 / 2) .* k2);
            k4 = f(x + h1, v1 + h1 .* k3);

            k = (1 / 6) .* (k1 + 2 .* k2 + 2 .* k3 + k4);
            v1 = v1 + h1 .* k;

            % 2. Step Size: h2 = h1 / 2
            h2 = h1 / 2;
            v2 = v;

            for J = 1 : 2
```

```
                k1 = f(x, v2);
                k2 = f(x + (h2 / 2), v2 + (h2 / 2) .* k1);
                k3 = f(x + (h2 / 2), v2 + (h2 / 2) .* k2);
                k4 = f(x + h2, v2 + h2 .* k3);

                k = (1 / 6) .* (k1 + 2 .* k2 + 2 .* k3 + k4);
                v2 = v2 + h2 .* k;

            end

            % Evaluation
            v = v1;

            % Update Step Size
            h = h1 * nthroot((15 / 16) * epsTol / max(v2 - v1), 5);
            h = h1 * (abs((15 / 16) * epsTol / max(v2 - v1))) ^ (1 / 5);    %
abs(): is it right to use hear?

            % march
            x = x + h;

        end

        % V_phi
        V_phi = v(2);

        if V_phi > 0
            break;
        end

        % log
        xList = [xList x];
        vList = [vList v];
        hList = [hList h];

        % Count Iterations
        I = I + 1;

    end

    % Transpose
    xList = xList'  * (180 / pi);
    vList = vList';
    hList = hList';

    % Cone Angle & Unit Conversion
    phiCone = x * (180 / pi);

    % Number of Iterations
    numIter = I;

end
```