Table of Contents

Information

```
% Class: AAE 338
% Names:
%   - Matthew Currie
%   - Spence Troop
%   - Jonah Fouts
%   - Alex Porter
%   - Preetham Gowni
% Assignment: Final Project
```

Initialization

```
close all;
clear;
clc;
%lol
```

Graph Settings

```
chamberGraphs = 1;
```

Value Initialization

Rocket Engine Initial Conditions

```
pressureExit = 1; %psia
pressureChamber = 300; %psia
OF = 2.35;
mdot_engine = 0.7; %kg/s propellant mass flow rate (iterate this variable?)
contractionRatio = 3;
Lstar = .4; %meters, 9.84 in
T_hw = 1088; %K -Inconel X750 Wall Temperature (1500 F)
```

```
tensile = 5.537*10^8; %Pa - Tensile strength at 1088K (1500 F)
k_wall = 23; %W/m-K -Nozzle wall made from Inconel X750 @ 1500F
converge_num = 10000; %points in converging section
diverge_num = 10000; %points in diverging section

% Channel Initial Conditions
chan_ID = 0.003; %m (3 mm)
wall_thick = 0.001;
cham_chan_loops = 1; %Number of switch backs chamber has

%Initial Helium Conditons
heltemp_init = 120;
helpress_init = 80e6;
helmach_init = 0.4;
Cp_init = py.CoolProp.CoolProp.PropsSI("C","T",heltemp_init,"P",
 helpress_init,"Helium");
Cv_init = py.CoolProp.CoolProp.PropsSI("O","T",heltemp_init,"P",
 helpress_init,"Helium");
gamma_init = Cp_init/Cv_init;
```

CEA

```
CEAPath = append(pwd, '/PSP_CEA_function_wrapper');
INPPath = append(pwd, '/INP_OUT');
funcPath = append(pwd, '/funcs');
addpath(CEAPath);
addpath(INPPath);
addpath(funcPath);

nameString = strcat('338_estimates_pip_', num2str(int8(pressureChamber /
 pressureExit)), '_p_c_', num2str(pressureChamber), '_O_F_', num2str(OF));

inputName = append(nameString, '.inp');
outputName = append(nameString, '.out');

[Isp, CStar, expansionRatio, specificHeatRatio,
 combustionTemperature, Cpcea, ~, ~, rho0,Prcea,visccea,dataeq] =
 PSP_1DOF_CEA_function_wrapper(pressureChamber,pressureExit, OF, nameString,
 0);
movefile(inputName, 'INP_OUT');
movefile(outputName, 'INP_OUT');
delete(append(pwd, '\PSP_CEA_function_wrapper\', inputName));
Isp = Isp / 9.81; %seconds

gma = specificHeatRatio;
P0 = pressureChamber * 6894.76; %Pa
T_cns = combustionTemperature; %K

R = P0 / (rho0 * T_cns);

Aratio_sub = linspace(contractionRatio, 1, converge_num);
Aratio_sup = linspace(1.001, expansionRatio, diverge_num);
Aratio = [Aratio_sub Aratio_sup];
```

```
M_x = [];
rho_x = [];
T_x = [];
for subratio = Aratio_sub
    [mach_sub, T_sub, P_sub, rho_sub, area_sub] = flowisentropic(gma,
 subratio, 'sub');
    M_x = [M_x mach_sub];
    rho_x = [rho_x rho_sub];
    T_x = [T_x T_sub];
end
for supratio = Aratio_sup
    [mach_sup, T_sup, P_sup, rho_sup, area_sup] = flowisentropic(gma,
 supratio, 'sup');
    M_x = [M_x mach_sup];
    rho_x = [rho_x rho_sup];
    T_x = [T_x T_sup];
end
rho_x = rho_x .* rho0;
T_x = T_x .* T_cns;
V_x = M_x .* sqrt(gma .* R .* T_x);

Pr = (4 * gma) / (9 * gma - 5);
r = Pr ^ (0.33);

T_gas = T_cns .* (1 + (r .* ((gma - 1) ./ 2) .* M_x)) ./ (1 + (((gma - 1) ./
 2) .* M_x));
```

Chamber

```
At = (mdot_engine * CStar) / P0; %m^2

[chamber_L, contract_L, nozzle_L, cham_chan_num] = getChamberSize(At,
 contractionRatio, expansionRatio, Lstar, chan_ID, wall_thick);

Dt = 2 * sqrt(At/pi);
Dc = sqrt((At * contractionRatio) / pi) * 2;
De = sqrt((At * expansionRatio) / pi) * 2;
Ac = pi* (Dc/2)^2;

A = ((At ./ M_x) .* (((2 + (gma - 1) .* M_x .^ 2) ./ (gma + 1)) .^ ((gma +
 1) ./ (2 .* (gma - 1)))));

x1 = linspace(-contract_L, 0, converge_num);
x2 = linspace(0, nozzle_L, diverge_num);
xplot = [x1 x2];
x3 = linspace(-chamber_L, -contract_L, 5);
x4 = sqrt(A(1)/pi) * ones(length(x3));

hbartz = hgcalc(gma, visccea, Prcea, Cpcea, A./At, pressureChamber *
 0.0689476, CStar, Dt);
%h_g_x = (rho_x .* V_x) .^ 0.8; %Convection Coefficent Correlation
%h_g_x = hbartz/h_g_x(1)*h_g_x;
```

```
Qdot_x = hbartz .* (T_gas - T_hw);

Thrust = mdot_engine*V_x(end) + pressureExit*A(end);
```

Chamber Cooling Loop

```
%Initial Conditons for Chamber Loop
h_gas = hbartz(1);
Tstag_hel_init = heltemp_init * (1+((gamma_init-1)/2)*helmach_init);
T0stari = heltemp_init * ((1 + gamma_init * helmach_init^2)^2 / (2 *
 (gamma_init + 1) * helmach_init^2));


% For Loop Conditions
step = 0.001;
Tmax = 1088; % Temperature at which material properties fall apart
Mmax = 1; % Max Mach number allowed in code
steps = floor(2*cham_chan_loops*chamber_L / step); % Number of steps in the
 foor loop

% Array initialization
qdot_arr = zeros(1, steps);
T_cw_arr = zeros(1, steps);
T_hw_arr = zeros(1, steps);
T_hel_arr = zeros(1, steps);
P_hel_arr = zeros(1, steps);
M_hel_arr = zeros(1, steps);
mdot_arr = zeros(1, steps);
rho_arr = zeros(1, steps);
% Informs user Loop is Running
disp('Simulation Running...');

for i = 1:steps

    %Setting Helium Step Input Parameters
    if i == 1
        % Sets initial properties of helium
        Ti = heltemp_init;
        Pi = helpress_init;
        Mi = helmach_init;
        T0i = Tstag_hel_init;
    else
        Ti = Te;
        Pi = Pe;
        Mi = Me;
        T0i = T0e;
    end

    %Initializing Forced Convection
    Cp = py.CoolProp.CoolProp.PropsSI("C","T",Ti,"P", Pi,"Helium");
    Cv = py.CoolProp.CoolProp.PropsSI("O","T",Ti,"P", Pi,"Helium");
    gma_hel = Cp/Cv;
```

```
    R_i = py.CoolProp.CoolProp.PropsSI("gas_constant","T",Ti,"P",
Pi,"Helium");
    Vel_i = Mi * sqrt(gma_hel * R_i * Ti);
    [q_dot, T_cw, T_hw] = convergeTemp(T_cns, h_gas, k_wall, wall_thick, Ti,
Vel_i, chan_ID, Pi);

    %Checking Forced Convection Convergence
    if q_dot == 1

        disp('Forced Convection Failed to Converge')
        break
    end


    %Initialiing Ralyeigh Flow
    Qdot = q_dot * step * (chan_ID+2*wall_thick);
    rho_i = py.CoolProp.CoolProp.PropsSI("D","T",Ti,"P", Pi,"Helium");
    mdot = rho_i * Vel_i * (pi*(chan_ID/2)^2);

    % Rayleigh Flow Calculations
    [T0e] = getTempStagNew(Qdot, mdot, Mi, Ti, gma_hel, Cp);
    [Me,Te, Pe] = RayleighFlow(Pi, T0e, Mi, gma_hel, T0stari);

    % Places helium values into array
    T_hel_arr(i) = Ti;
    P_hel_arr(i) = Pi;
    M_hel_arr(i) = Mi;
    mdot_arr(i) = mdot;
    rho_arr(i) = rho_i;
    % Storing Forced Convection Results
    qdot_arr(i) = q_dot;
    T_cw_arr(i) = T_cw;
    T_hw_arr(i) = T_hw;

    % Break Conditions
    Tbreak = T_hw > Tmax;
    MmaxBreak = Me > Mmax;

    % Checks to see if the Mach Number has decreased
    if i > 1
        MdecBreak = Me < M_hel_arr(i-1);
    else
        MdecBreak = false;
    end

    % Breaks loop if conditions are met
    breakLoop = Tbreak || MmaxBreak || MdecBreak;
    if breakLoop
        % Displays data that could break loop
        disp('Max Value was reached and Loop Was Broken');
        disp('Final Values:');
        fprintf('Hot Wall Temperature:    %0.3f\n', T_hw)
        fprintf('Mach Number:             %0.3f\n', Me);
        if i ~= 1
```

```
            fprintf('Previous iteration Mach: %0.3f\n', M_hel_arr(i-1));
        else
            disp('Loop Broke on first iteration.');
        end

        % Breaks Loop
        break
    end
end

% Prints that the simulation is complete
disp('Simulation Complete');
```

Simulation Running...
Simulation Complete

Nozzle Cooling Loop

```
% For Loop Conditions
step_down = chan_ID + 2*wall_thick;
step_around = 0.001;

Tmax = 1088; % Temperature at which material properties fall apart
Mmax = 1; % Max Mach number allowed in code
steps_down = floor((contract_L + nozzle_L)/ step_down); % Number of steps in
 the foor loop

max_rad = sqrt(A(end)/pi);
max_tubelen = 2*pi*(max_rad + wall_thick + chan_ID/2);
max_steps_around = floor(max_tubelen/step_around);

% Array initialization
qdot_arr_cha = zeros(steps_down, max_steps_around);
T_cw_arr_cha = zeros(steps_down, max_steps_around);
T_hw_arr_cha = zeros(steps_down, max_steps_around);
T_hel_arr_cha = zeros(steps_down, max_steps_around);
P_hel_arr_cha = zeros(steps_down, max_steps_around);
M_hel_arr_cha = zeros(steps_down, max_steps_around);
rho_arr_cha = zeros(steps_down, max_steps_around);

% Informs user L0oop is Running
disp('Simulation Running...');

for j = 1:steps_down

    [hgas,area,Tgas,tubelen] = nozzleprops(chan_ID, A, hbartz,...
        wall_thick, j,T_gas,converge_num,contract_L,diverge_num, nozzle_L);


    steps_around = floor(tubelen/step_around);

    h_gas = hgas;
    Chambertemp = Tgas;
```

```
    for i = 1:steps_around

        %Setting Helium Step Input Parameters
        if i == 1
            % Sets initial properties of helium
            Ti = heltemp_init;
            Pi = helpress_init;
            Mi = helmach_init;
            T0i = Tstag_hel_init;

        else
            Ti = Te;
            Pi = Pe;
            Mi = Me;
            T0i = T0e;
        end

        %Initializing Forced Convection
        Cp = py.CoolProp.CoolProp.PropsSI("C","T",Ti,"P", Pi,"Helium");
        Cv = py.CoolProp.CoolProp.PropsSI("O","T",Ti,"P", Pi,"Helium");
        gma_hel = Cp/Cv;

        R_i = py.CoolProp.CoolProp.PropsSI("gas_constant","T",Ti,"P",
Pi,"Helium");
        Vel_i = Mi * sqrt(gma_hel * R_i * Ti);
        [q_dot, T_cw, T_hw] = convergeTemp(Chambertemp, h_gas, k_wall,
wall_thick, Ti,Vel_i,chan_ID,Pi);

        %Checking Forced Convection Convergence
        if q_dot == 1

            disp('Forced Convection Failed to Converge')
            break
        end

        %Initialiing Ralyeigh Flow
        Qdot = q_dot * step * (chan_ID+2*wall_thick);
        rho_i = py.CoolProp.CoolProp.PropsSI("D","T",Ti,"P", Pi,"Helium");
        mdot = rho_i * Vel_i * (pi*(chan_ID/2)^2);

        % Rayleigh Flow Calculations
        [T0e] = getTempStagNew(Qdot, mdot, Mi, Ti, gma_hel, Cp);
        [Me,Te, Pe] = RayleighFlow(Pi, T0e, Mi, gma_hel, T0stari);

        % Places helium values into array
        T_hel_arr_cha(j,i) = Ti;
        P_hel_arr_cha(j,i) = Pi;
        M_hel_arr_cha(j,i) = Mi;
        rho_arr_cha(j,i) = rho_i;
        % Storing Forced Convection Results
        qdot_arr_cha(j,i) = q_dot;
        T_cw_arr_cha(j,i) = T_cw;
        T_hw_arr_cha(j,i) = T_hw;
```

```
        % Break Conditions
        Tbreak = T_hw > Tmax;
        MmaxBreak = Me > Mmax;

        % Checks to see if the Mach Number has decreased
        if i > 1
            MdecBreak = Me < M_hel_arr_cha(i-1);
        else
            MdecBreak = false;
        end

        % Breaks loop if conditions are met
        breakLoop = Tbreak || MmaxBreak || MdecBreak;
        if breakLoop
            % Displays data that could break loop
            disp('Max Value was reached and Loop Was Broken');
            disp('Final Values:');
            fprintf('Hot Wall Temperature: %0.3f\n', T_hw)
            fprintf('Mach Number:          %0.3f\n', Me);
            if i ~= 1
                fprintf('Previous Mach:    %0.3f\n', M_hel_arr_cha(i-1));
            else
                disp('Loop Broke on first iteration.');
            end

            % Breaks Loop
            break
        end

    end

    percentDone = j / steps_down * 100;
    fprintf('%0.2f Percent Complete\n', percentDone);
end
```

```
Simulation Running...
2.13 Percent Complete
4.26 Percent Complete
6.38 Percent Complete
8.51 Percent Complete
10.64 Percent Complete
12.77 Percent Complete
14.89 Percent Complete
17.02 Percent Complete
19.15 Percent Complete
21.28 Percent Complete
23.40 Percent Complete
25.53 Percent Complete
27.66 Percent Complete
29.79 Percent Complete
31.91 Percent Complete
34.04 Percent Complete
36.17 Percent Complete
```

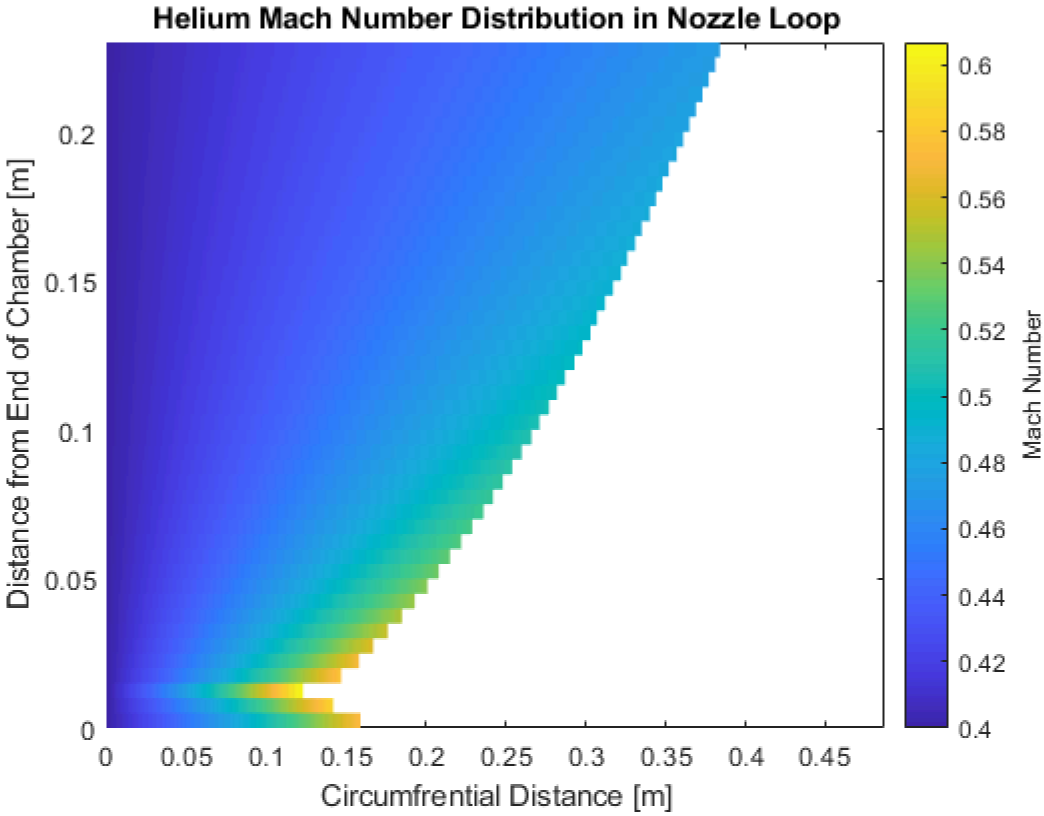
```
38.30 Percent Complete
40.43 Percent Complete
42.55 Percent Complete
44.68 Percent Complete
46.81 Percent Complete
48.94 Percent Complete
51.06 Percent Complete
53.19 Percent Complete
55.32 Percent Complete
57.45 Percent Complete
59.57 Percent Complete
61.70 Percent Complete
63.83 Percent Complete
65.96 Percent Complete
68.09 Percent Complete
70.21 Percent Complete
72.34 Percent Complete
74.47 Percent Complete
76.60 Percent Complete
78.72 Percent Complete
80.85 Percent Complete
82.98 Percent Complete
85.11 Percent Complete
87.23 Percent Complete
89.36 Percent Complete
91.49 Percent Complete
93.62 Percent Complete
95.74 Percent Complete
97.87 Percent Complete
100.00 Percent Complete
```

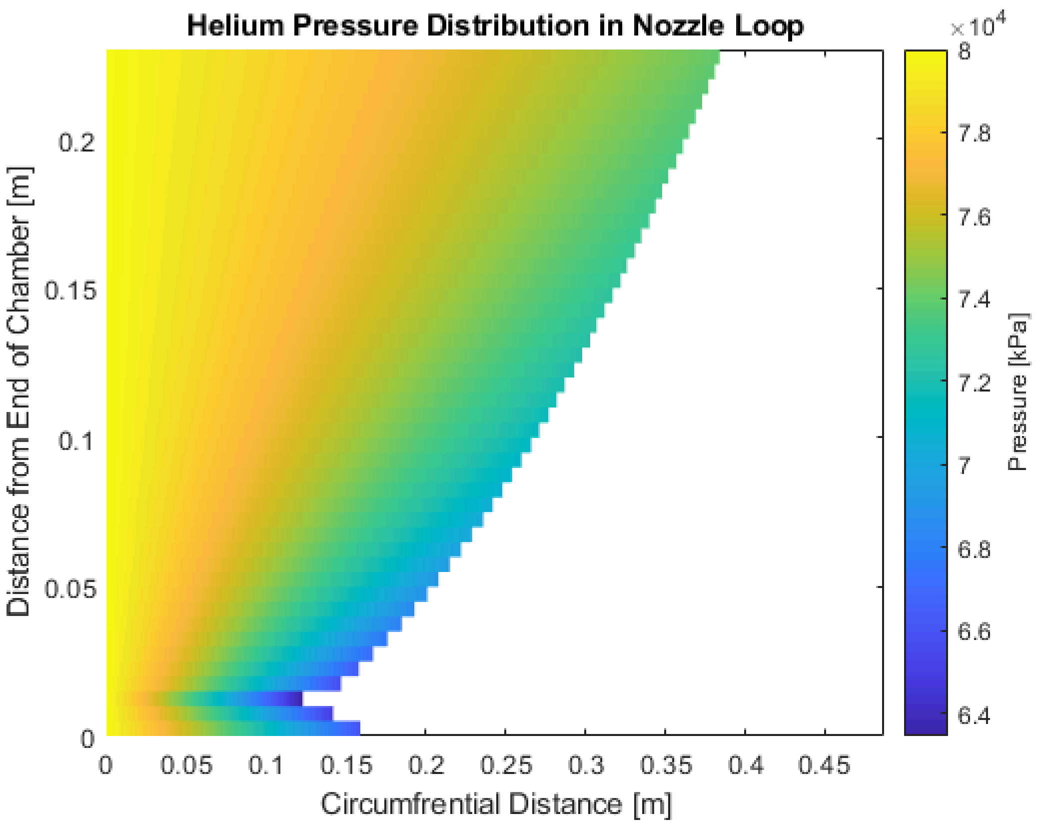
Thrust Calc and Plotting

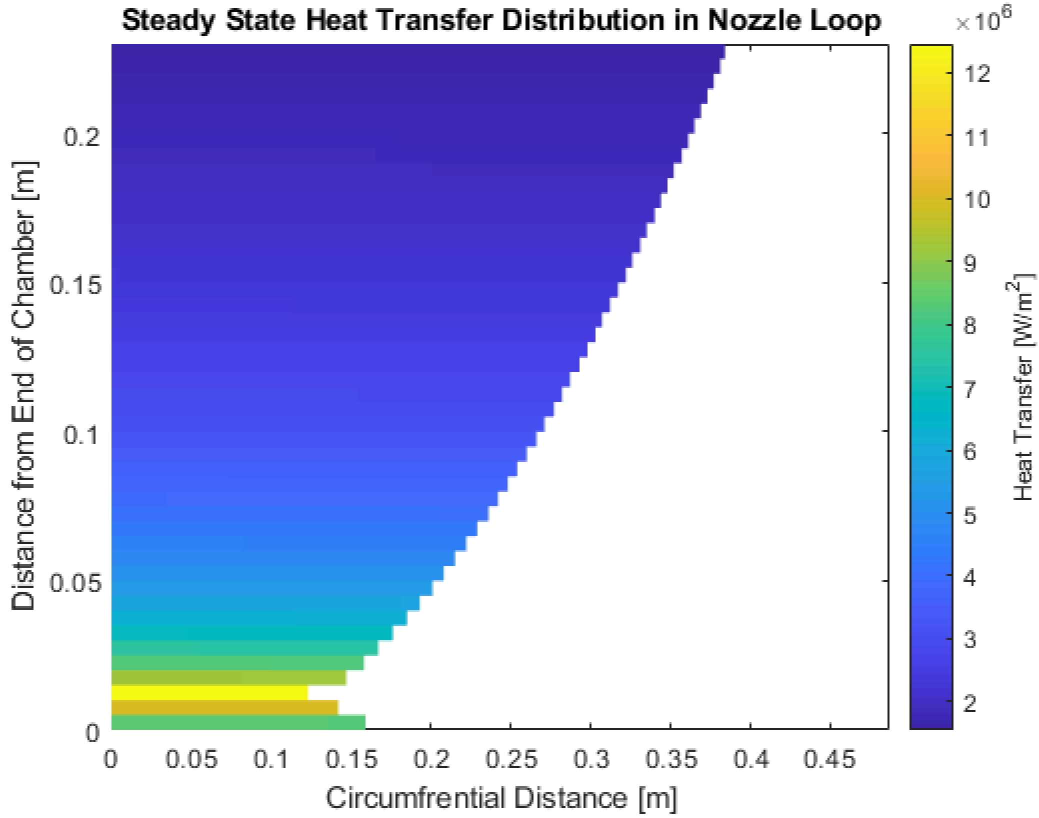
```
mdot_hel_total = (ceil(cham_chan_num)/(cham_chan_loops*2) + steps_down)*mdot;

fprintf("Engine Thrust:         %0.2f N \n", Thrust)
fprintf("Helium Mass Flow Rate: %0.2f kg/s \n", mdot_hel_total)

disp('Simulation Complete');

M_hel_arr_cha(M_hel_arr_cha==0) = NaN;
xscale = [0:step_around:(size(M_hel_arr_cha,2)-1)*step_around];
yscale = [0:step_down:(size(M_hel_arr_cha,1)-1)*step_down];
figure()
s = pcolor(xscale,yscale,M_hel_arr_cha);
colorbar
s.EdgeColor = 'none';
title('Helium Mach Number Distribution in Nozzle Loop')
xlabel('Circumfrential Distance [m]')
ylabel('Distance from End of Chamber [m]')
a=colorbar;
ylabel(a,'Mach Number')
```

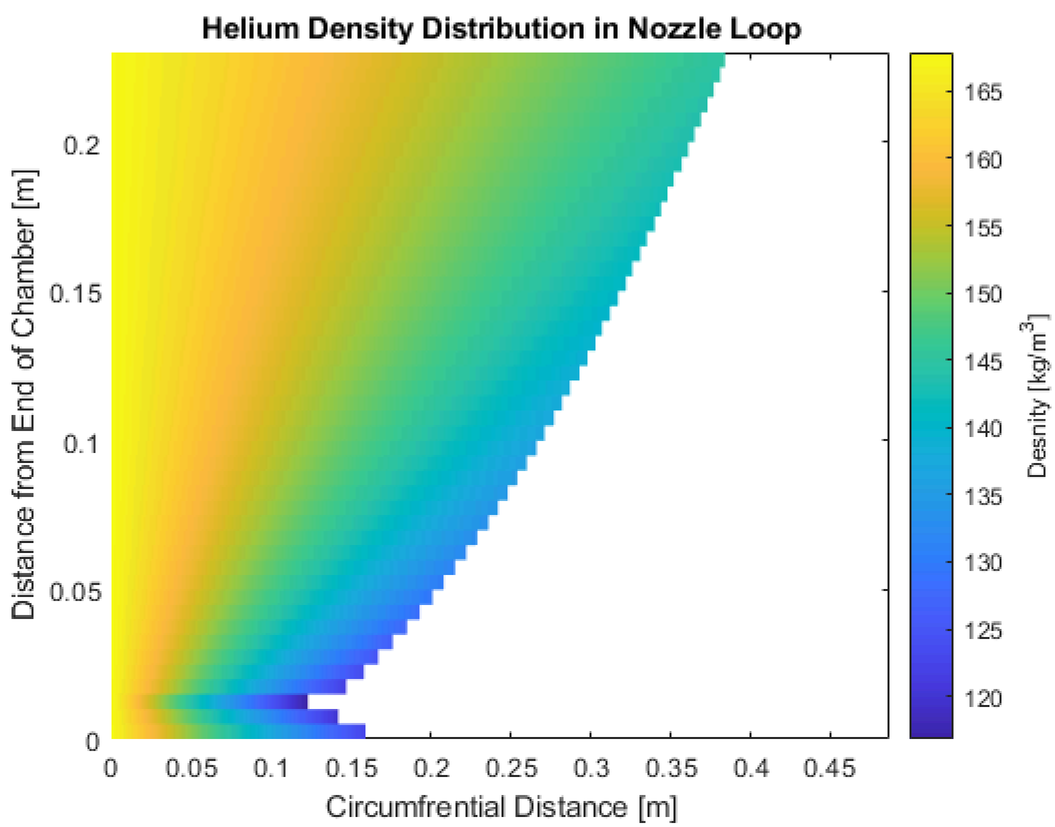
```
P_hel_arr_cha(P_hel_arr_cha==0) = NaN;
figure()
s = pcolor(xscale,yscale,P_hel_arr_cha./1000);
s.EdgeColor = 'none';
title('Helium Pressure Distribution in Nozzle Loop')
a=colorbar;
ylabel(a,'Pressure [kPa]')
xlabel('Circumfrential Distance [m]')
ylabel('Distance from End of Chamber [m]')

qdot_arr_cha(qdot_arr_cha==0) = NaN;
figure()
s = pcolor(xscale,yscale,qdot_arr_cha);
s.EdgeColor = 'none';
a=colorbar;
title('Steady State Heat Transfer Distribution in Nozzle Loop')
ylabel(a,'Heat Transfer [W/m^2]')
xlabel('Circumfrential Distance [m]')
ylabel('Distance from End of Chamber [m]')

rho_arr_cha(rho_arr_cha==0) = NaN;
figure()
s = pcolor(xscale,yscale,rho_arr_cha);
s.EdgeColor = 'none';
a=colorbar;
title('Helium Density Distribution in Nozzle Loop')
ylabel(a,'Desnity [kg/m^3]')
xlabel('Circumfrential Distance [m]')
ylabel('Distance from End of Chamber [m]')

T_hw_arr_nozz = T_hw_arr_cha;
T_hw_arr_nozz(T_hw_arr_nozz==0) = NaN;
figure()
s = pcolor(xscale,yscale,T_hw_arr_nozz);
s.EdgeColor = 'none';
a=colorbar;
title('Nozzle Wall Temperature Distribution in Nozzle Loop')
ylabel(a,'Temperature [K]')
xlabel('Circumfrential Distance [m]')
ylabel('Distance from End of Chamber [m]')


% Plots Graphs
if chamberGraphs
    coolinggrapher(M_hel_arr, qdot_arr, T_hw_arr, T_cw_arr, T_hel_arr,...
        P_hel_arr, Aratio, xplot, A, T_gas, hbartz, Qdot_x, M_x, rho_x,...
        T_x, V_x, x3, x4, T_hw_arr_cha, M_hel_arr_cha, P_hel_arr_cha,
 qdot_arr_cha, rho_arr_cha, T_hw_arr_nozz, rho_arr, hbartz)

end
```

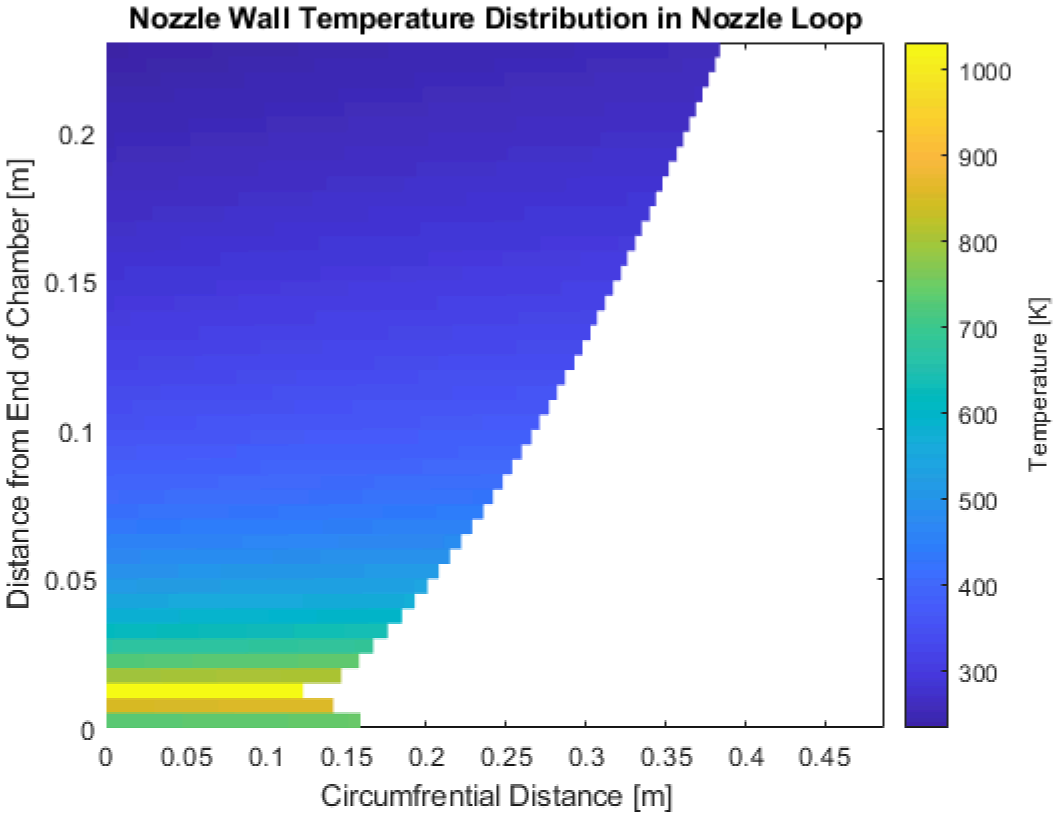
Engine Thrust: 2140.78 N
Helium Mass Flow Rate: 1.33 kg/s
Simulation Complete

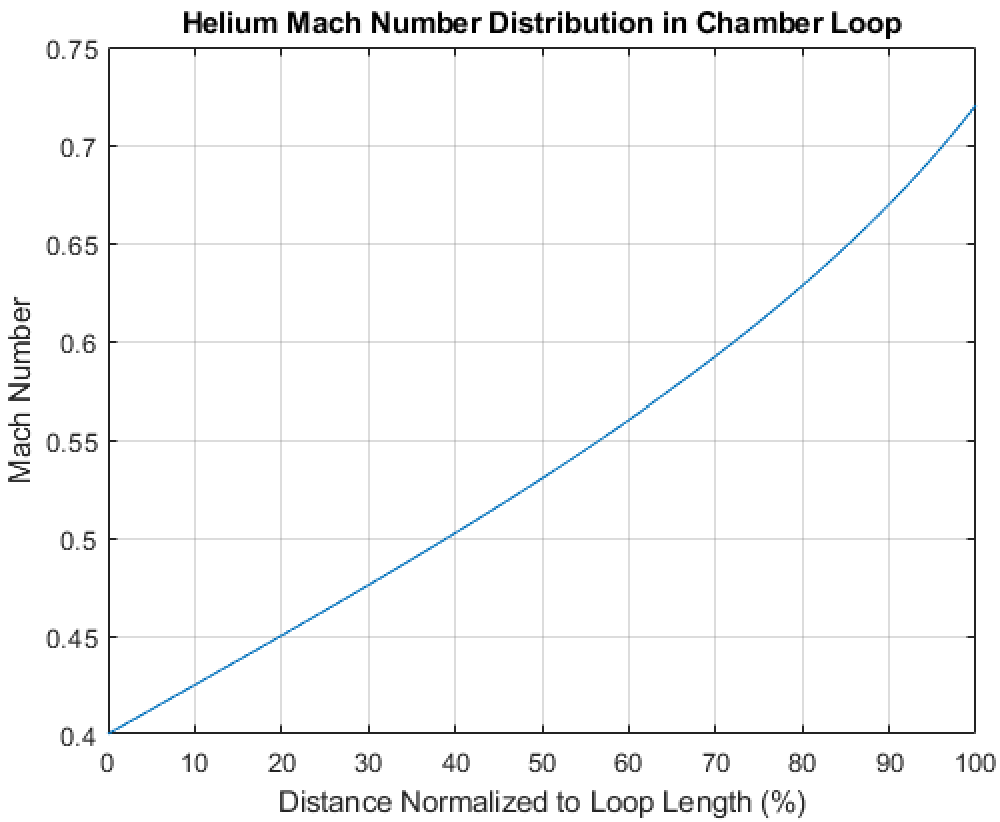

Helium Mach Number Distribution in Nozzle Loop
Distance from End of Chamber [m]
Circumfrential Distance [m]
Mach Number
0.4
0.42
0.44
0.46
0.48
0.5
0.52
0.54
0.56
0.58
0.6

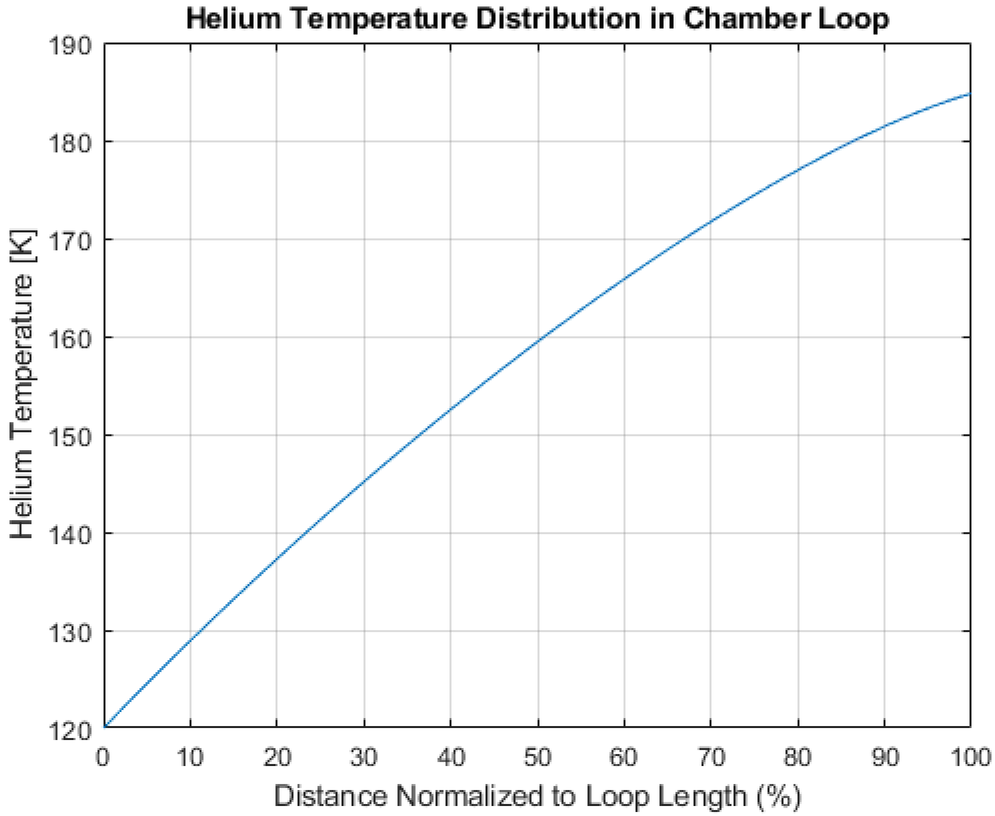

Helium Pressure Distribution in Nozzle Loop
Distance from End of Chamber [m]
Circumfrential Distance [m]
Pressure [kPa]
×10^4
6.4
6.6
6.8
7
7.2
7.4
7.6
7.8
8

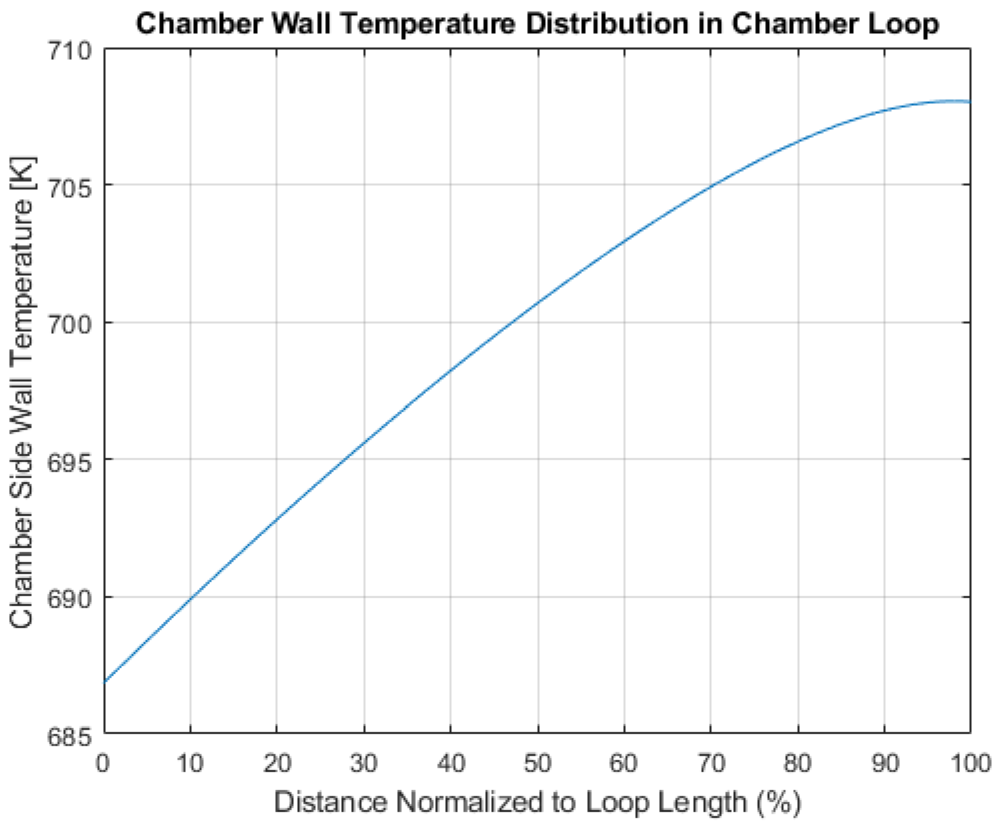

Steady State Heat Transfer Distribution in Nozzle Loop
Distance from End of Chamber [m]
Circumfrential Distance [m]
Heat Transfer [W/m^2]
$\times 10^6$

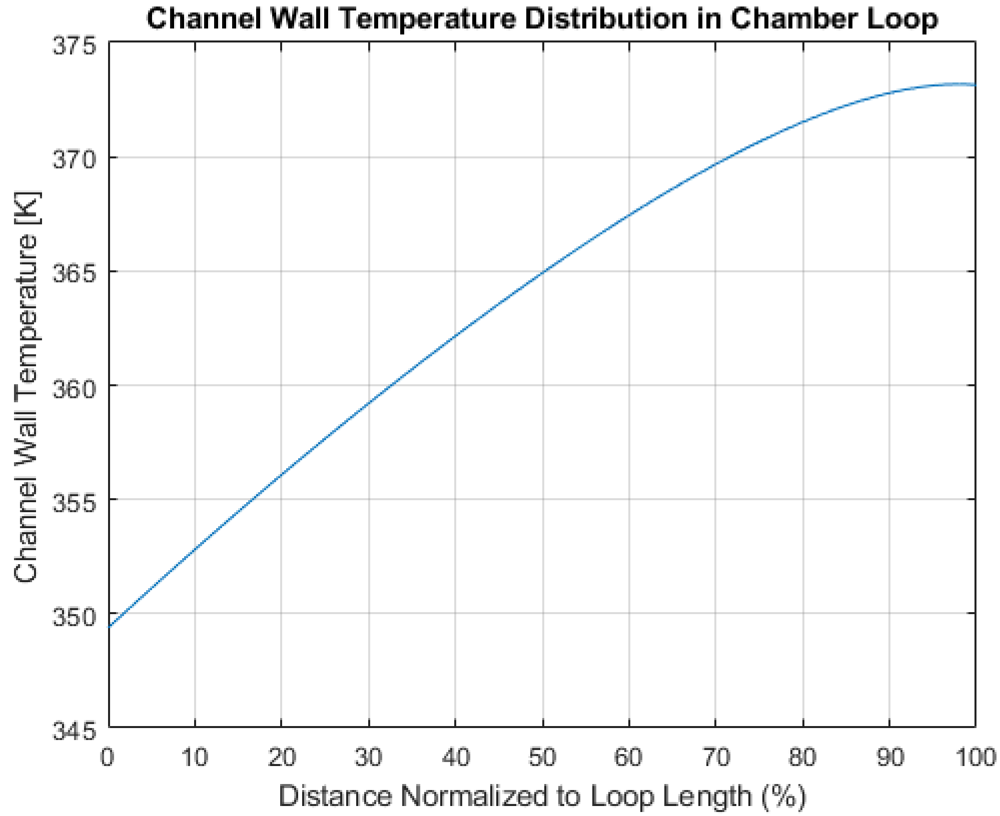

Helium Density Distribution in Nozzle Loop
Distance from End of Chamber [m]
Circumfrential Distance [m]
Desnity [kg/m^3]


Nozzle Wall Temperature Distribution in Nozzle Loop
Distance from End of Chamber [m]
Circumfrential Distance [m]
Temperature [K]
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
300
400
500
600
700
800
900
1000

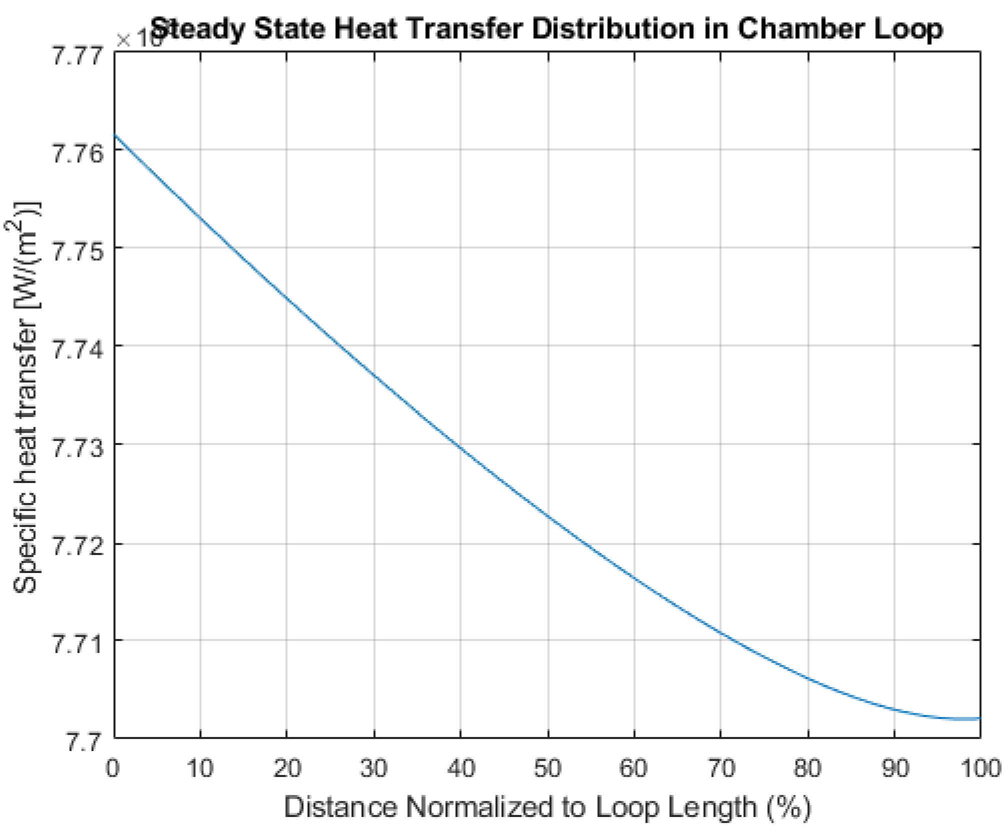

Helium Mach Number Distribution in Chamber Loop
Mach Number
Distance Normalized to Loop Length (%)
0.4
0.45
0.5
0.55
0.6
0.65
0.7
0.75
0
10
20
30
40
50
60
70
80
90
100


Helium Temperature Distribution in Chamber Loop
Helium Temperature [K]
Distance Normalized to Loop Length (%)
120
130
140
150
160
170
180
190
0
10
20
30
40
50
60
70
80
90
100

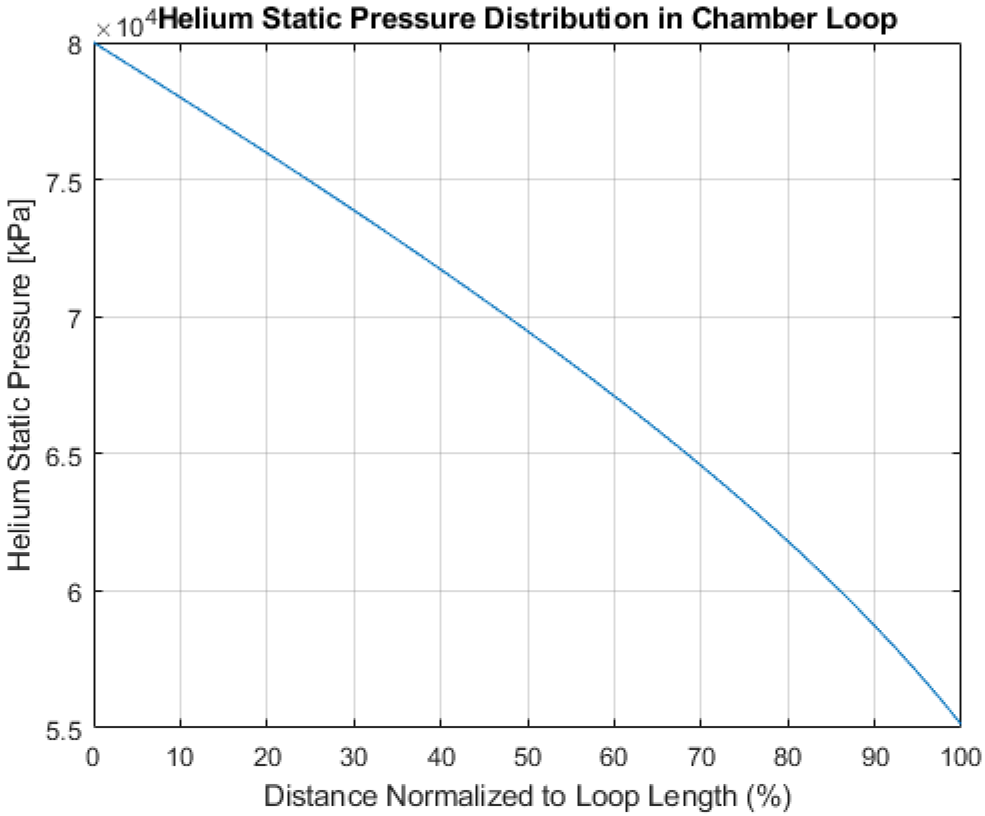

Chamber Wall Temperature Distribution in Chamber Loop
Chamber Side Wall Temperature [K]
Distance Normalized to Loop Length (%)
685
690
695
700
705
710
0
10
20
30
40
50
60
70
80
90
100

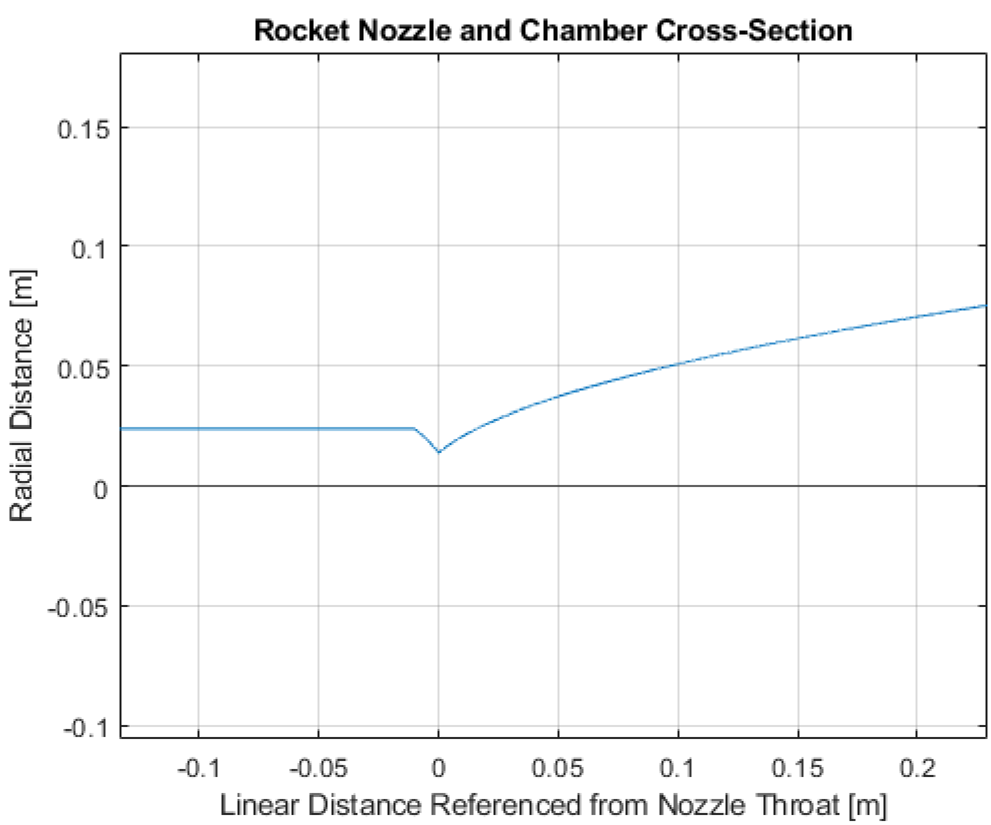


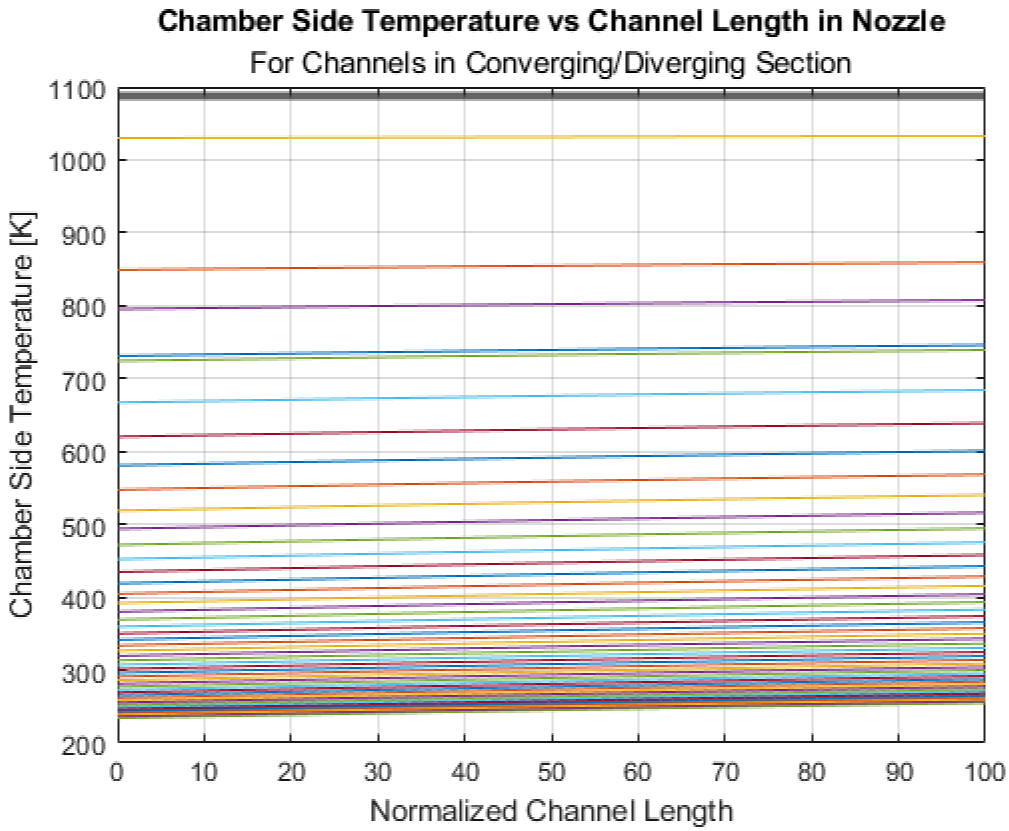
Channel Wall Temperature Distribution in Chamber Loop
Channel Wall Temperature [K]
Distance Normalized to Loop Length (%)

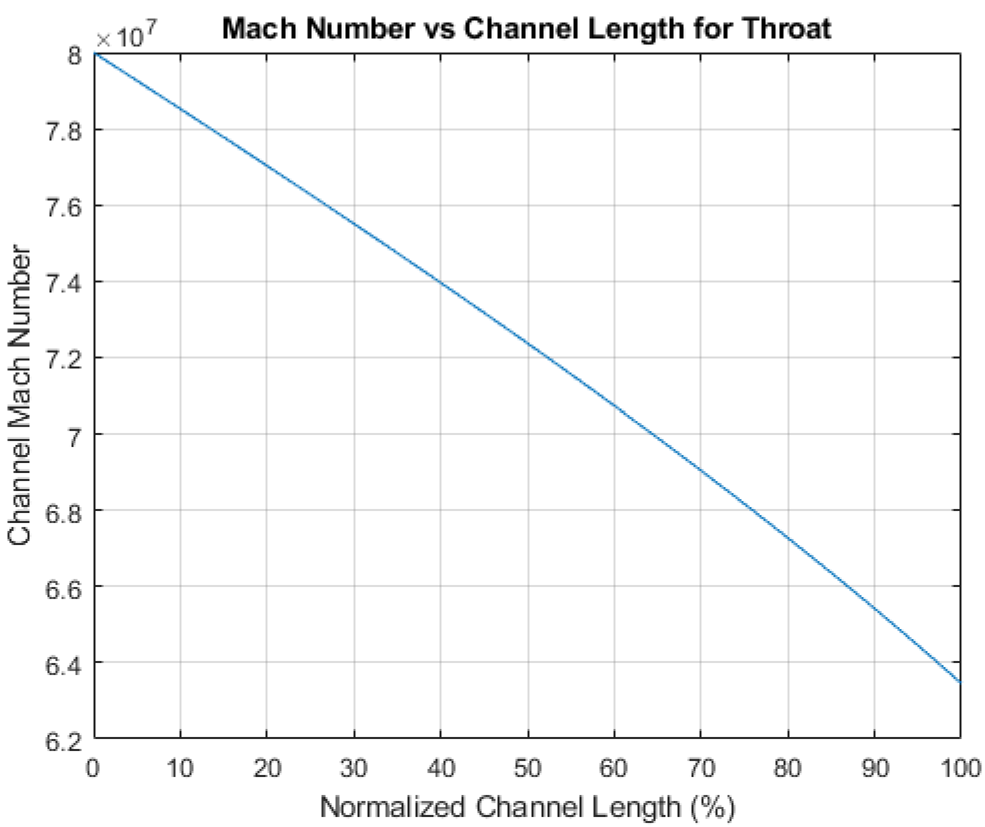


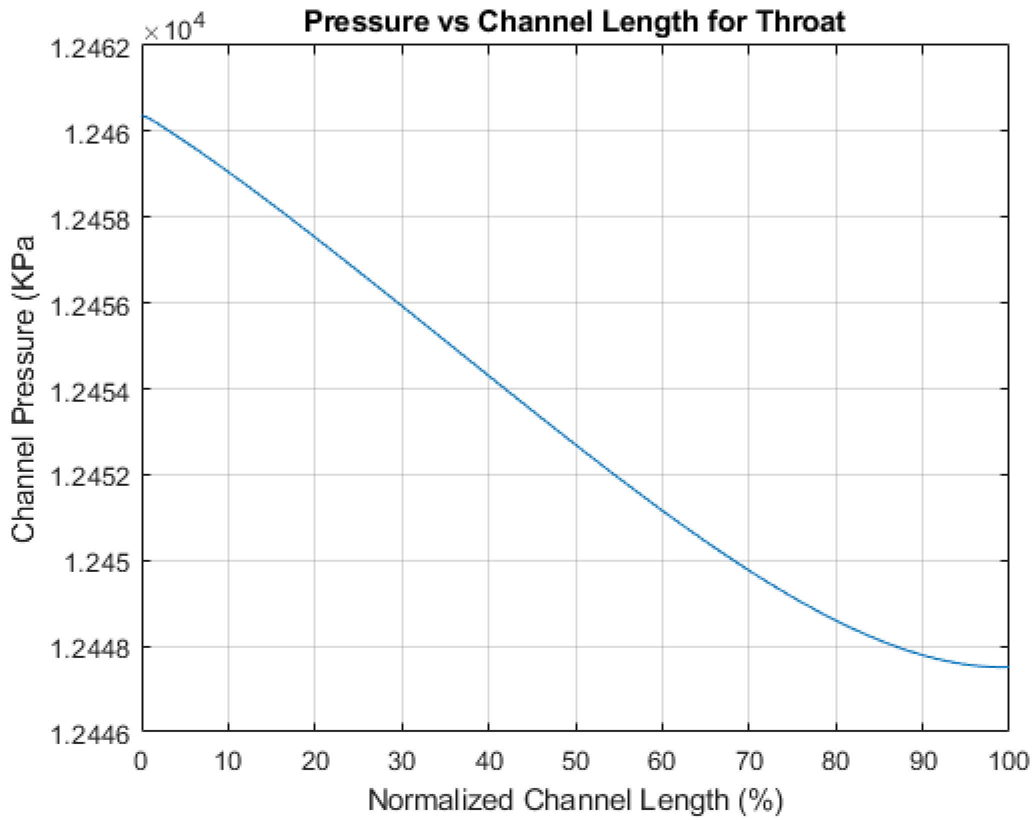
Steady State Heat Transfer Distribution in Chamber Loop
Specific heat transfer [W/(m2)]
Distance Normalized to Loop Length (%)

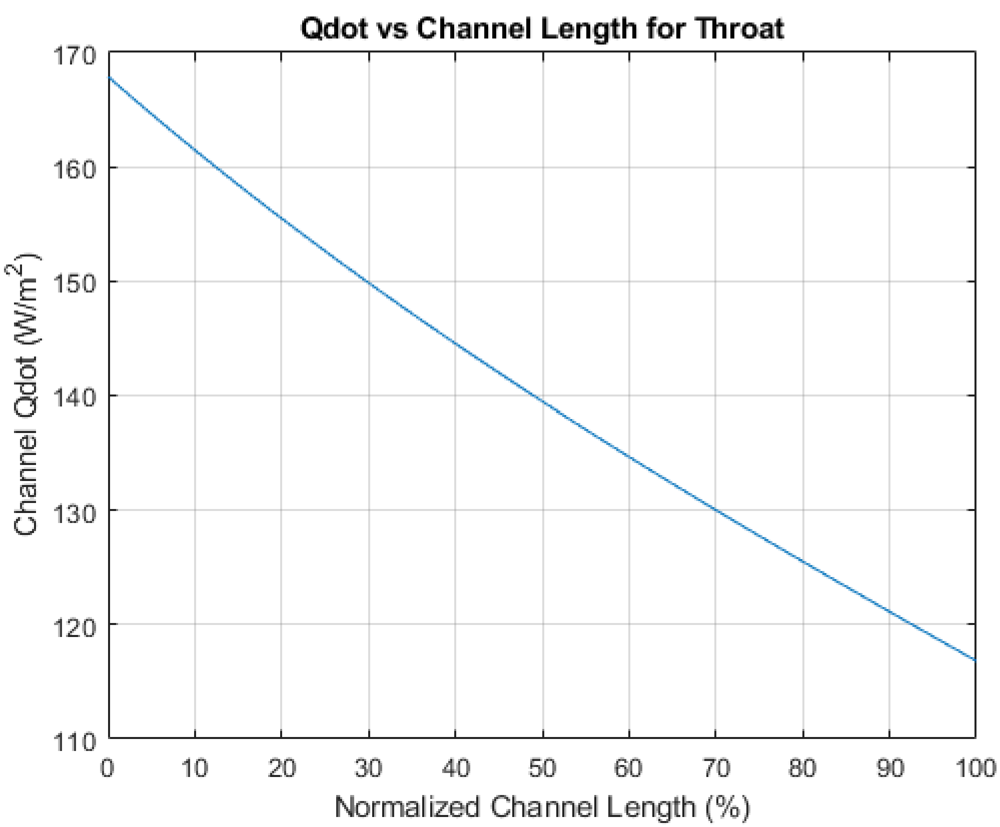

Helium Static Pressure Distribution in Chamber Loop
× 10^4
Helium Static Pressure [kPa]
8
7.5
7
6.5
6
5.5
0
10
20
30
40
50
60
70
80
90
100
Distance Normalized to Loop Length (%)

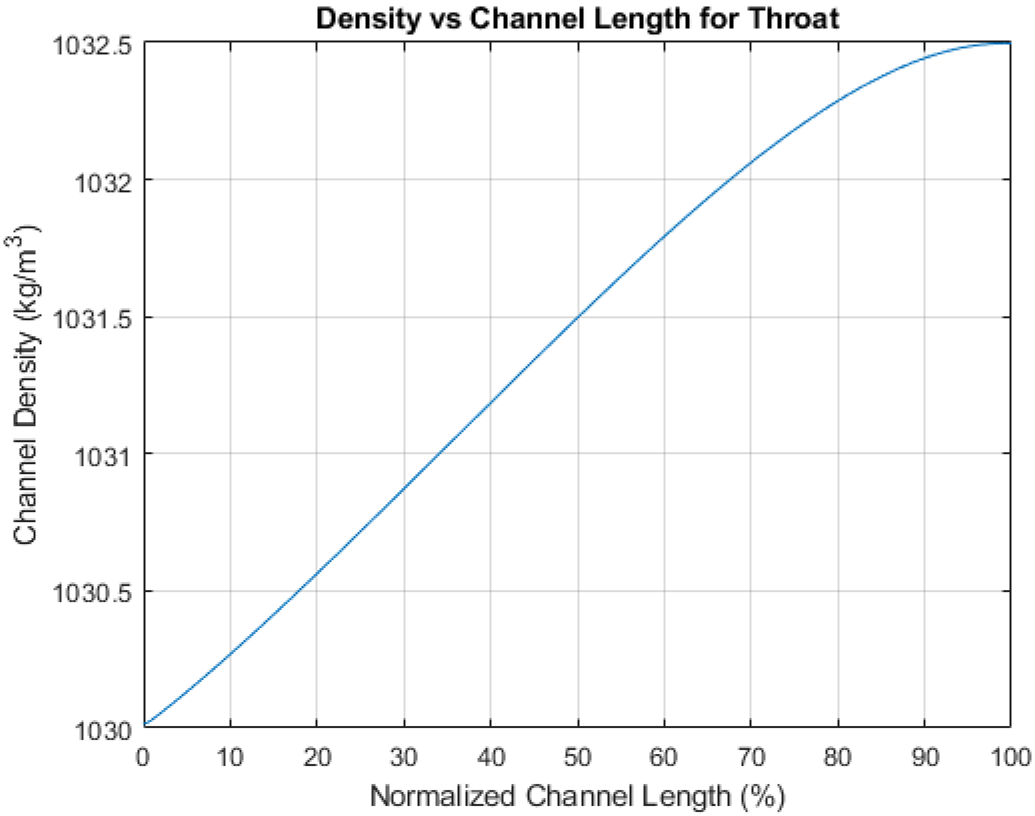

Rocket Nozzle and Chamber Cross-Section
Radial Distance [m]
0.15
0.1
0.05
0
-0.05
-0.1
-0.1
-0.05
0
0.05
0.1
0.15
0.2
Linear Distance Referenced from Nozzle Throat [m]

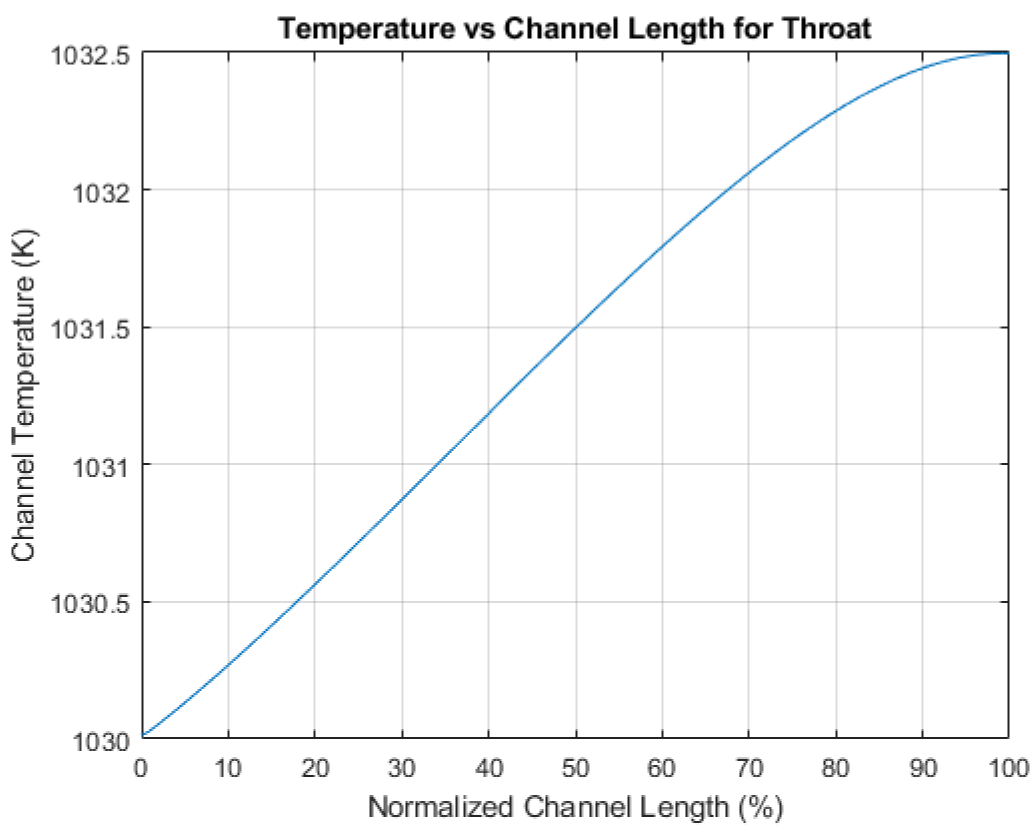

Chamber Side Temperature vs Channel Length in Nozzle
For Channels in Converging/Diverging Section
Chamber Side Temperature [K]
1100
1000
900
800
700
600
500
400
300
200
0
10
20
30
40
50
60
70
80
90
100
Normalized Channel Length


Mach Number vs Channel Length for Throat
×10^7
Channel Mach Number
8
7.8
7.6
7.4
7.2
7
6.8
6.6
6.4
6.2
0
10
20
30
40
50
60
70
80
90
100
Normalized Channel Length (%)


Pressure vs Channel Length for Throat
×10^4
1.2462
1.246
1.2458
1.2456
1.2454
1.2452
1.245
1.2448
1.2446
Channel Pressure (KPa
0
10
20
30
40
50
60
70
80
90
100
Normalized Channel Length (%)


Qdot vs Channel Length for Throat
170
160
150
140
130
120
110
Channel Qdot (W/m^2)
0
10
20
30
40
50
60
70
80
90
100
Normalized Channel Length (%)


Density vs Channel Length for Throat
Channel Density (kg/m3)
1032.5
1032
1031.5
1031
1030.5
1030
0
10
20
30
40
50
60
70
80
90
100
Normalized Channel Length (%)


Temperature vs Channel Length for Throat
Channel Temperature (K)
1032.5
1032
1031.5
1031
1030.5
1030
0
10
20
30
40
50
60
70
80
90
100
Normalized Channel Length (%)

Bottom of Script

Resets Matlabs Path preference so it doesn't mess up your matlab

```
clear CEApath INPPath funcPath

restoredefaultpath;
clear RESTOREDEFAULTPATH_EXECUTED
```

Referenced Functions

This example includes the external functions (No CEA program files

```
function [Me,Te, Pe] = RayleighFlow(Pi, T0e, Mi, gamma, T0star)
    % Calculates New Mach Number using Stagnation Temperature of the
    % exit flowing gas

    % Calculates T0* of the flowing gas and ratio
    T_ratio = T0e / T0star;

    % Solves for Mach Number
    frac1 = (-gamma * T_ratio) / (gamma^2*T_ratio - gamma^2 + 1);

    frac2num = sqrt(-4*gamma^2*T_ratio - 8*gamma*T_ratio - 4*T_ratio +...
        4*gamma^2 + 8*gamma + 4);
    frac2den = 2 * (gamma^2 * T_ratio - gamma^2 + 1);
    frac2 = frac2num / frac2den;

    frac3 = (gamma + 1) / (gamma^2*T_ratio - gamma^2 + 1);

    Me = abs(sqrt(frac1 - frac2 + frac3));

     % Calculates Static Presure at Exit
    Pe = Pi * ((gamma * Mi^2 + 1) / (gamma * Me^2 + 1));

    % Calculates Static Temperature at Exit
    Te = T0e / (1 + ((gamma - 1) / 2) * Me^2);

end


function [hgas,area,Tgas,tubelen] = nozzleprops(Dt, Area_arr, h_g_x, wallt,
 tubenum,T_gas_arr,converge_num,contract_L,diverge_num,nozzle_L)

%Find Area at a distance x from the start of the converging section
x = (tubenum*Dt) - (Dt/2);
% lenarrconv = linspace(0,contract_L, converge_num);
% lenarrdiv = linspace(0,nozzle_L, diverge_num);
% lenarr = [lenarrconv lenarrdiv];
disp(floor(converge_num * x / contract_L))
if x <= contract_L
```

```
    area = Area_arr(floor(converge_num * x / contract_L));
    Tgas = T_gas_arr(floor(converge_num * x / contract_L));
    hgas = h_g_x(floor(converge_num * x / contract_L));
elseif x <= contract_L + nozzle_L
    area = Area_arr(floor(diverge_num * x / nozzle_L)+converge_num);
    Tgas = T_gas_arr(floor(diverge_num * x / nozzle_L)+converge_num);
    hgas = h_g_x(floor(diverge_num * x / nozzle_L)+converge_num);
else
    disp("Spence Sucks")
end
nozzlerad = sqrt(area/pi);
tubelen = 2*pi*(nozzlerad + wallt + Dt/2);
disp(hgas)
disp(area)
disp(Tgas)
disp(tubelen)
end




function [Te0] = getTempStagNew(Qdot, mdot, Mi, Ti, gamma, Cp)
% Calculates T0e after the application of the heat
% Grabs Cp value and Calculates Exit Temperature
Ti0 = Ti * (1 + ((gamma - 1) / 2) * Mi^2);
Te0 = (Qdot/ (mdot * Cp)) + Ti0;
end



function [Re] = getRe(Vel, dynvisc, rho, Dh)
% Returns the reynolds number

Re = (rho*Vel*Dh)/dynvisc;

end



function [Hc] = getHc(vel,Dh,Ti,Pi)
dy_vsic_bulkT = py.CoolProp.CoolProp.PropsSI('V','T',Ti,'P',Pi,'Helium');
rho = py.CoolProp.CoolProp.PropsSI('D','T',Ti,'P',Pi,'Helium');
Pr = py.CoolProp.CoolProp.PropsSI('Prandtl','T',Ti,'P',Pi,'Helium');
k_bulkT = py.CoolProp.CoolProp.PropsSI('L','T',Ti,'P',Pi,'Helium');
Re = (rho*vel*Dh)/dy_vsic_bulkT;

Nu = 0.023*(Re^0.8)*(Pr^(0.4));
Hc = (k_bulkT/Dh)*Nu;
end



function [chamber_L, contract_L, nozzle_L, cham_chan_num] =
 getChamberSize(A_t, contrac, expand, L_star, chan_ID, chan_t);
```

```
%% Calculating Chamber Length
A_c = contrac * A_t;
V_c = L_star * A_t;

chamber_L = V_c/A_c;

%% Calculating Contacting Length
%Assumes simple 45 degree contracting angle

r_c = sqrt(A_c/pi);
r_t = sqrt(A_t/pi);

contract_L = r_c - r_t;

%%  Calculating Nozzle Length
%Assumes simple 15 degree conical nozzle

A_e = A_t * expand;

r_e = sqrt(A_e/pi);

nozzle_L = (r_e - r_t)/tand(15);

%% Chamber Channel Number
chan_OD = chan_ID + 2*chan_t;

new_diam = 2*r_c + chan_OD/2;
new_circum = new_diam * pi;
cham_chan_num = new_circum/chan_OD;



function [] = coolinggrapher(M_hel_arr, qdot_arr, T_hw_arr, T_cw_arr,
 T_hel_arr, P_hel_arr, Aratio, xplot, A, T_gas, h_g_x, Qdot_x, M_x, rho_x,
 T_x, V_x, x3, x4, T_hw_arr_cha, T_hw, M_hel_arr_cha, P_hel_arr_cha,
 qdot_arr_cha, rho_arr_cha, T_hw_arr_nozz, rho_arr, h_gas)

figure()
plot(linspace(0,100,length(M_hel_arr)), M_hel_arr)
grid on
xlabel("Distance Normalized to Loop Length (%)")
ylabel("Mach Number")
title('Helium Mach Number Distribution in Chamber Loop')

figure()
plot(linspace(0,100,length(rho_arr)), rho_arr)
grid on
xlabel("Distance Normalized to Loop Length (%)")
ylabel("Helium Density [kg/m^3]")
title('Helium Density Distribution in Chamber Loop')

plot(linspace(0,100,length(T_hel_arr)), T_hel_arr)
grid on
xlabel("Distance Normalized to Loop Length (%)")
```

```
ylabel("Helium Temperature [K]")
title('Helium Temperature Distribution in Chamber Loop')

figure()
plot(linspace(0,100,length(T_hel_arr)), T_hw_arr)
grid on
xlabel("Distance Normalized to Loop Length (%)")
ylabel("Chamber Side Wall Temperature [K]")
title('Chamber Wall Temperature Distribution in Chamber Loop')

figure()
plot(linspace(0,100,length(T_hel_arr)), T_cw_arr)
grid on
xlabel("Distance Normalized to Loop Length (%)")
ylabel("Channel Wall Temperature [K]")
title('Channel Wall Temperature Distribution in Chamber Loop')

figure()
plot(linspace(0,100,length(T_hel_arr)), qdot_arr)
grid on
xlabel("Distance Normalized to Loop Length (%)")
ylabel("Specific heat transfer [W/(m^2)]")
title("Steady State Heat Transfer Distribution in Chamber Loop")

figure()
plot(linspace(0,100,length(T_hel_arr)), P_hel_arr./1000)
grid on
xlabel("Distance Normalized to Loop Length (%)")
ylabel("Helium Static Pressure [kPa]")
title('Helium Static Pressure Distribution in Chamber Loop')



% Plot the cross-section
figure()
plot(xplot, sqrt(A/pi));
grid on
hold on
plot(x3, x4, 'color', [0, 0.4470, 0.7410])
yline(0)
xlabel('Linear Distance Referenced from Nozzle Throat [m]');
ylabel('Radial Distance [m]');
title('Rocket Nozzle and Chamber Cross-Section');
axis equal


% figure()
% plot(Aratio, T_gas)
% grid on;
% title('Nozzle Temperature Gas Distribution [Correlation]')
% xlabel("Area Ratio [A/At]")
% ylabel('Gas Temperature [K]')
%
```

```
% figure()
% plot(Aratio, h_g_x)
% grid on;
% title('Nozzle Convective Heat Transfer Coeff Distribution')
% xlabel("Area Ratio [A/At]")
%
% figure()
% plot(Aratio, Qdot_x)
% grid on;
% title('Nozzle Qdot Distribution')
% xlabel("Area Ratio [A/At]")
% ylabel('Qdot')
%
%
%
%
% plot(Aratio, M_x)
% grid on;
% title('Nozzle Mach Number Distribution')
% xlabel("Area Ratio [A/At]")
% ylabel('Mach Number')
%
% figure()
% plot(Aratio, rho_x)
% grid on;
% title('Nozzle Density Distribution')
% xlabel("Area Ratio [A/At]")
% ylabel('Density [kg/m^3]')
%
% figure()
% plot(Aratio, T_x)
% grid on;
% title('Nozzle Temperature Distribution [Isentropic]')
% xlabel("Area Ratio [A/At]")
% ylabel('Temperature [K]')
%
% figure()
% plot(Aratio, V_x)
% grid on;
% title('Nozzle Velocity Distribution')
% xlabel("Area Ratio [A/At]")
% ylabel('Velocity [m/s]')

figure()
find_zero = find(T_hw_arr_cha(1,:) == 0);
plot(linspace(0,100,length(T_hw_arr_cha(1,1:(find_zero(1) - 1)))),
 T_hw_arr_cha(1,1:(find_zero(1) - 1)))
grid on
hold on
i = 1;
while i < size(T_hw_arr_cha,1)
    find_zero = find(T_hw_arr_cha(i+1,:) == 0);
    if isempty(find_zero)
        find_zero = size(T_hw_arr_cha,2);
```

```
    end
    plot(linspace(0,100,length(T_hw_arr_cha(i+1,1:(find_zero(1) - 1)))),
 T_hw_arr_cha(i+1,1:(find_zero(1) - 1)))
    i = i+1;
end
yline(1088,"LineWidth",3)
xlabel("Normalized Channel Length")
ylabel("Chamber Side Temperature [K]")
title("Chamber Side Temperature vs Channel Length in Nozzle", "For Channels in
 Converging/Diverging Section")


find_zero_new = ones(10000,1);
for i = 1:size(M_hel_arr_cha,1)
    find_zero = find(isnan(M_hel_arr_cha(i,:)));
    if length(find_zero) > length(find_zero_new)
        find_zero_final = find_zero;
        i_final = i;
    end
    find_zero_new = find_zero;
end

figure()
plot(linspace(0,100,length(M_hel_arr_cha(i_final,1:(find_zero_final(1) -
 1)))),M_hel_arr_cha(i_final,1:(find_zero_final(1) - 1)))
grid on
xlabel("Normalized Channel Length (%)")
ylabel("Channel Mach Number")
title("Mach Number vs Channel Length for Throat")

figure()
plot(linspace(0,100,length(P_hel_arr_cha(i_final,1:(find_zero_final(1) -
 1)))),P_hel_arr_cha(i_final,1:(find_zero_final(1) - 1)) / 1000)
grid on
xlabel("Normalized Channel Length (%)")
ylabel("Channel Pressure (KPa")
title("Pressure vs Channel Length for Throat")

figure()
plot(linspace(0,100,length(qdot_arr_cha(i_final,1:(find_zero_final(1) -
 1)))),qdot_arr_cha(i_final,1:(find_zero_final(1) - 1)))
grid on
xlabel("Normalized Channel Length (%)")
ylabel("Channel Qdot (W/m^2)")
title("Qdot vs Channel Length for Throat")

figure()
plot(linspace(0,100,length(rho_arr_cha(i_final,1:(find_zero_final(1) -
 1)))),rho_arr_cha(i_final,1:(find_zero_final(1) - 1)))
grid on
xlabel("Normalized Channel Length (%)")
ylabel("Channel Density (kg/m^3)")
title("Density vs Channel Length for Throat")
```

```
figure()
plot(linspace(0,100,length(T_hw_arr_cha(i_final,1:(find_zero_final(1) -
 1)))),T_hw_arr_cha(i_final,1:(find_zero_final(1) - 1)))
grid on
xlabel("Normalized Channel Length (%)")
ylabel("Channel Temperature (K)")
title("Temperature vs Channel Length for Throat")
```

```
function [q_dot, T_cw, T_hw] = convergeTemp(T_gas, h_gas, k, wall_thick,...
    T_i,Vel,Dh,P_i)

% Grabs helium convection heat transfer coefficient
h_hel = getHc(Vel,Dh,T_i,P_i);

% Solves system of equations
num = (h_gas / h_hel) * T_gas + (wall_thick / k) * h_gas * T_gas + T_i;
den = (h_gas / h_hel) + 1 + (wall_thick / k) * h_gas;

% Derives other values
T_hw = num / den;
q_dot = h_gas * (T_gas - T_hw);
T_cw = -(q_dot * wall_thick / k) + T_hw;

end
```

```
File 'funcs/PSP_1DOF_CEA_function_wrapper.m' not found.
```

The file content above is properly syntax highlighted.

Published with MATLAB® R2023a