Andrew Albers

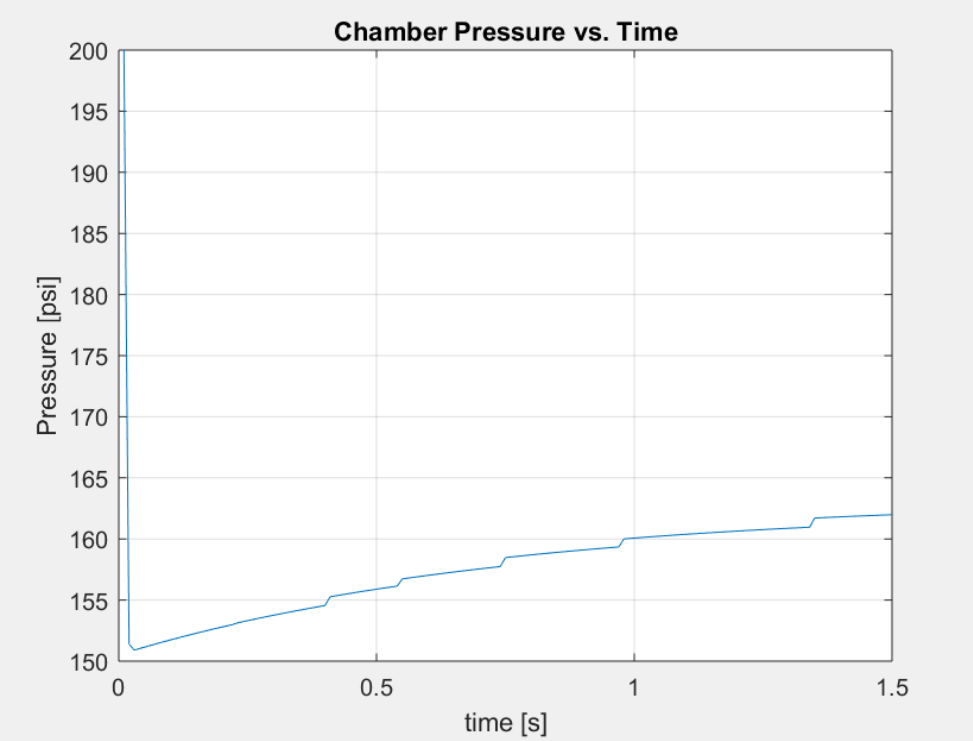
AAE 539 Homework 5:

Develop plots for Pc, Gox, R1, R2-6, mdot1, mdot2-6, OF1, OF2-6 for a burn time of 1.5 seconds.

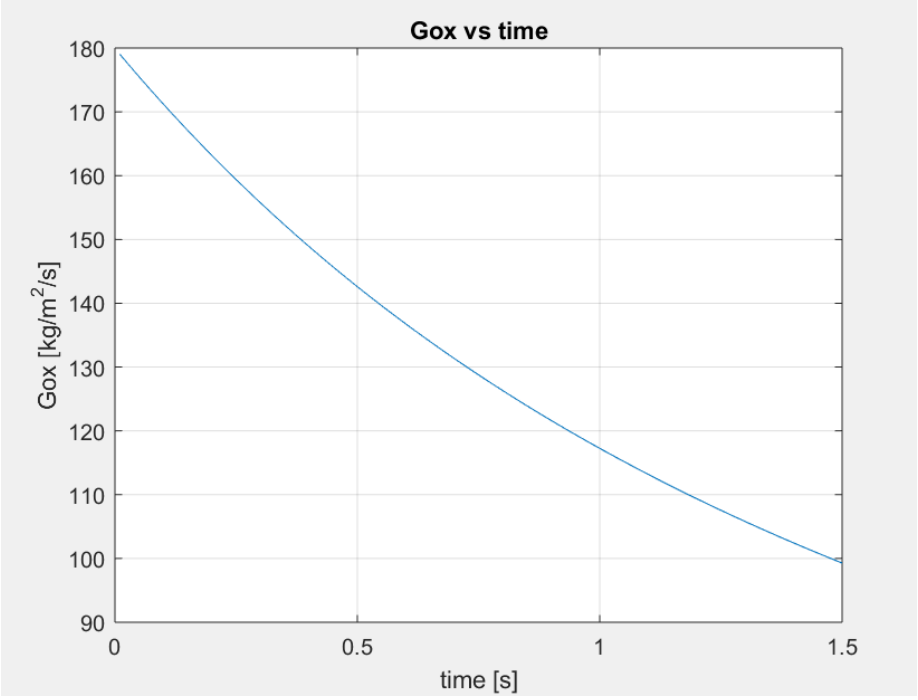
Procedure:

Case i):

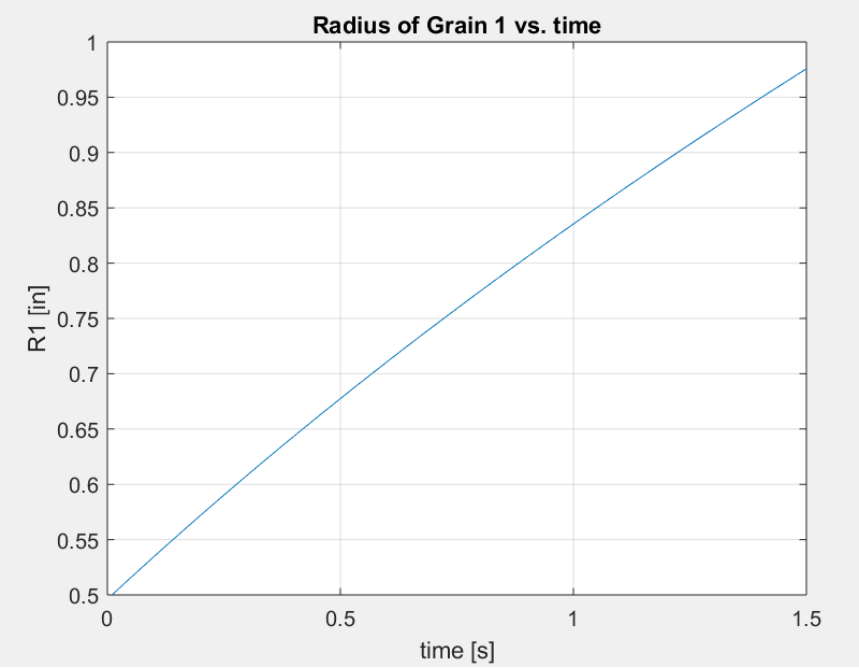
Pc vs. time:



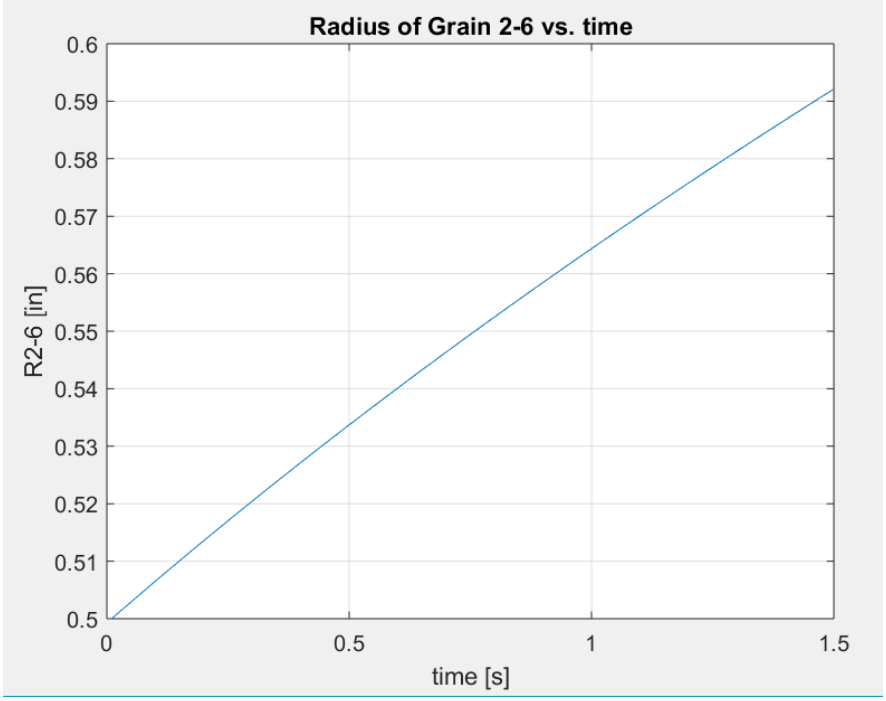
Gox vs. time:



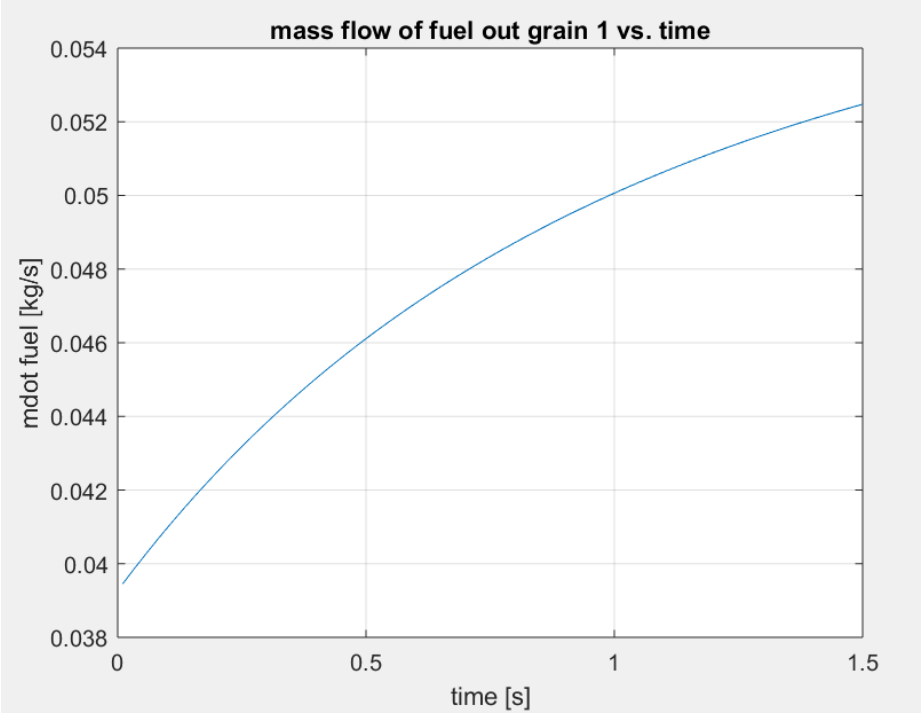
Grain 1 Radius vs time:



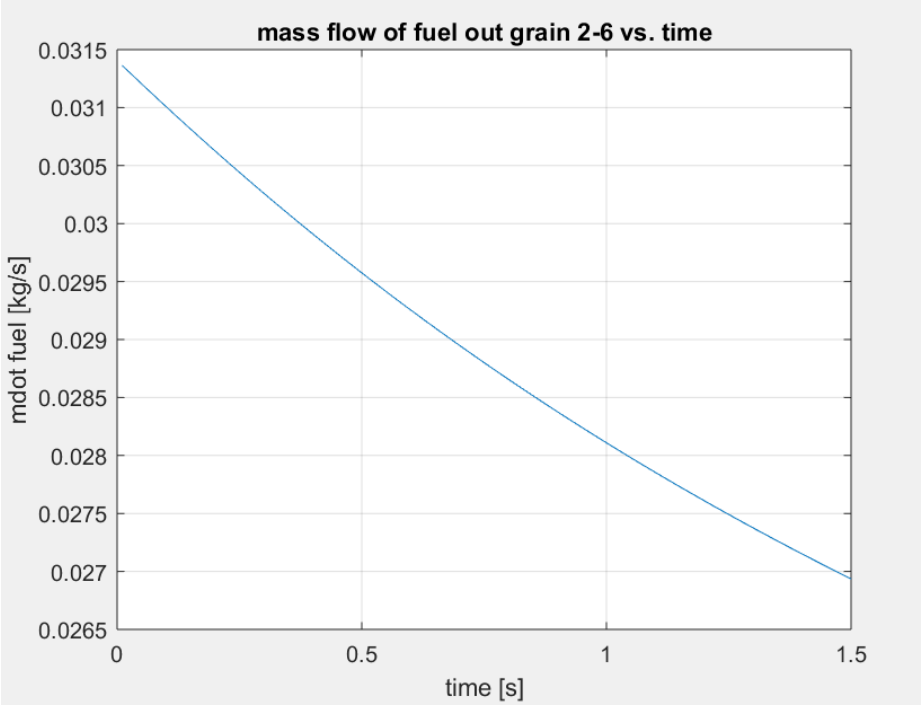
Grain2-6 Radius vs time:



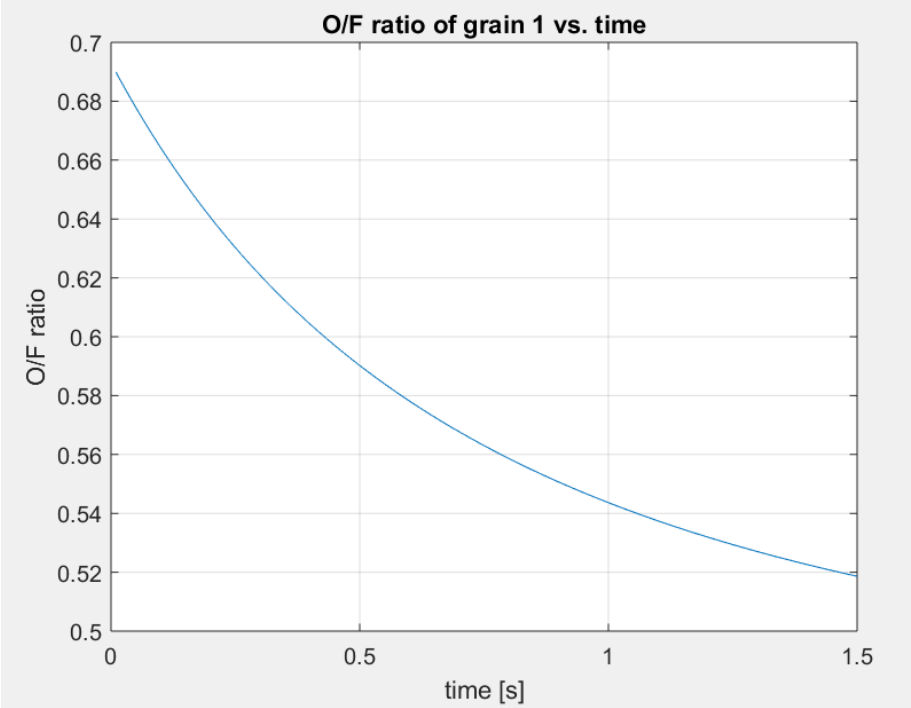
Mdot1 vs time:



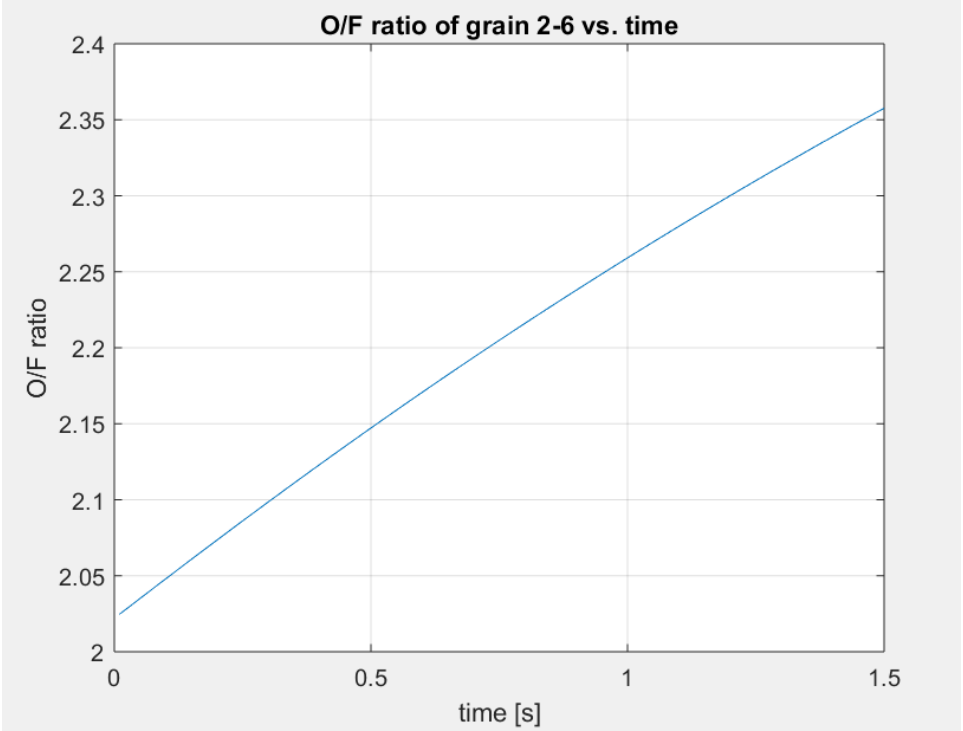
Mdot2-6 vs time:



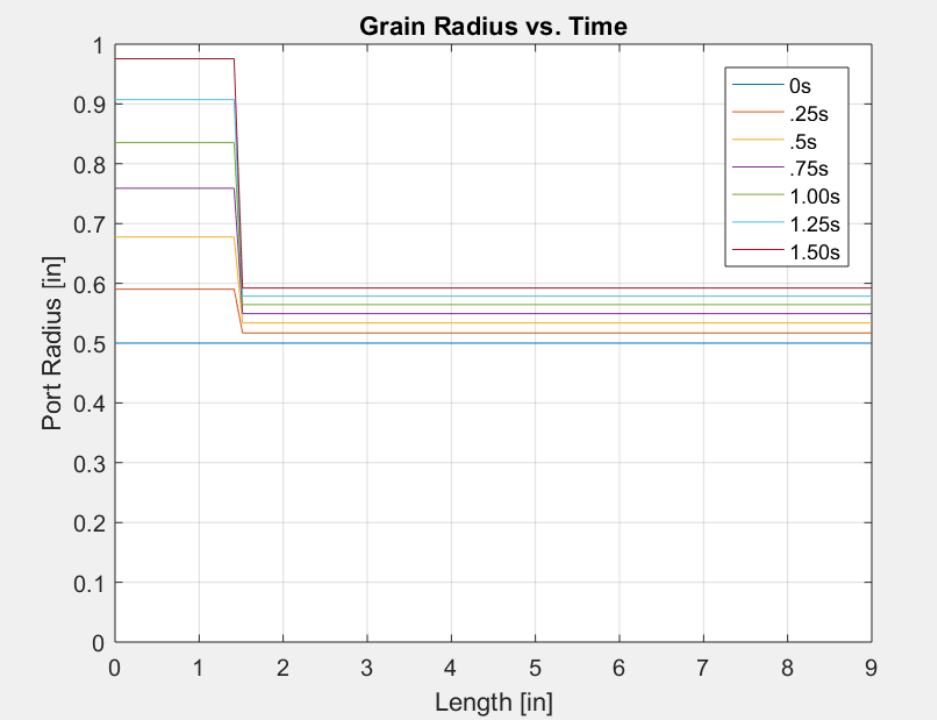
O/F of grain 1 vs time:



O/F of grain 2-6 vs time:

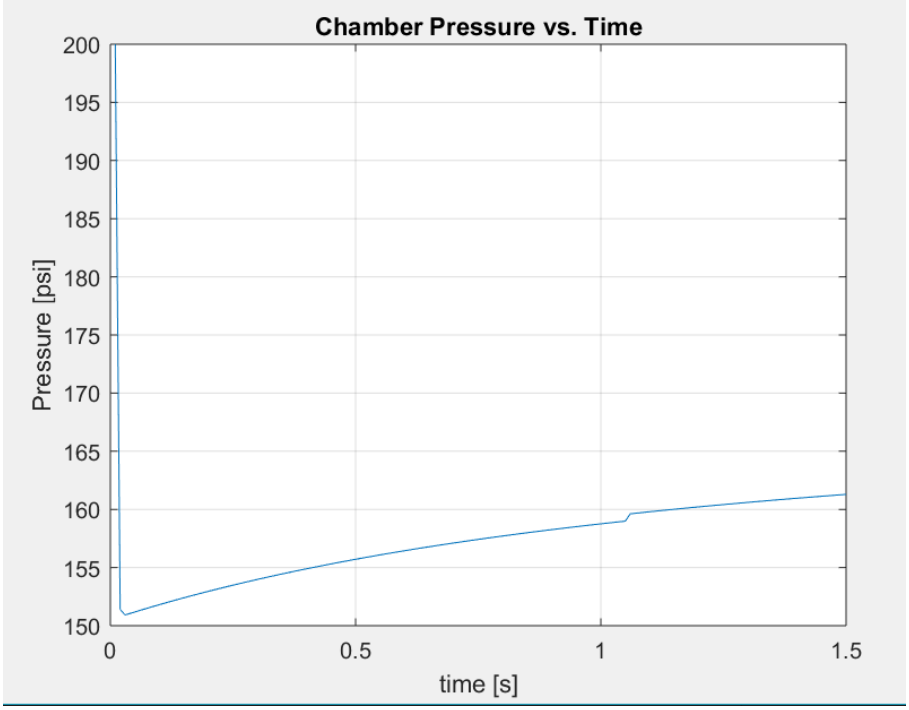


Fuel regression vs. Length

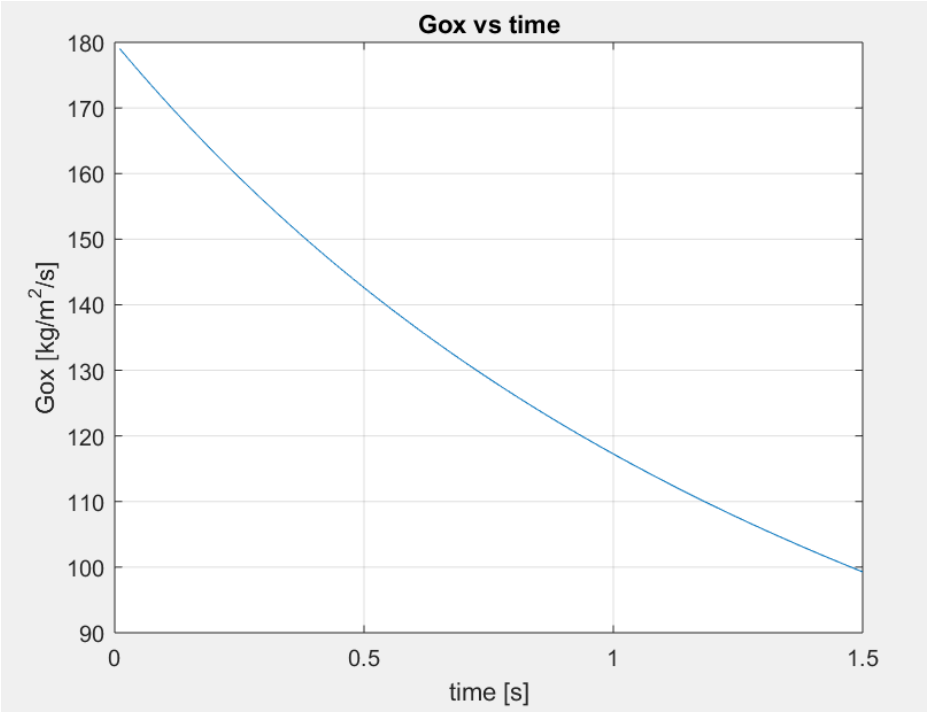


Case ii):

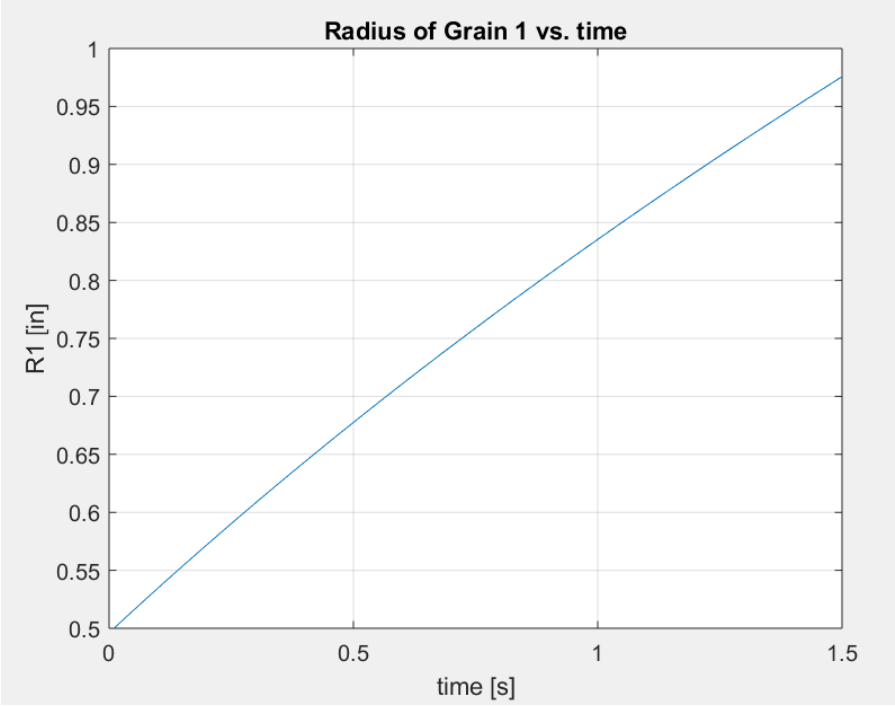
Pc vs. time:



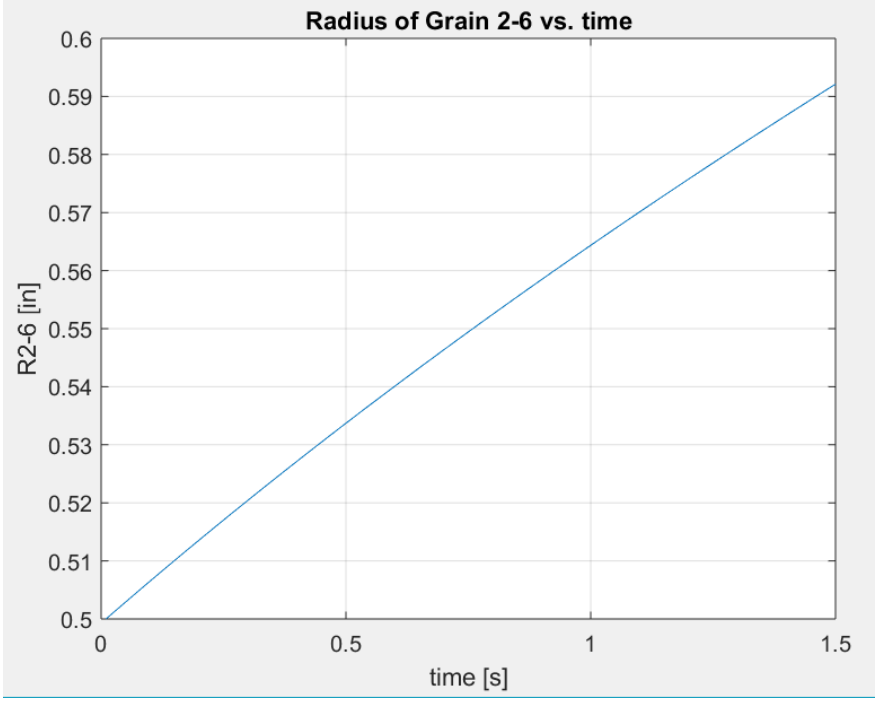
Gox vs time:



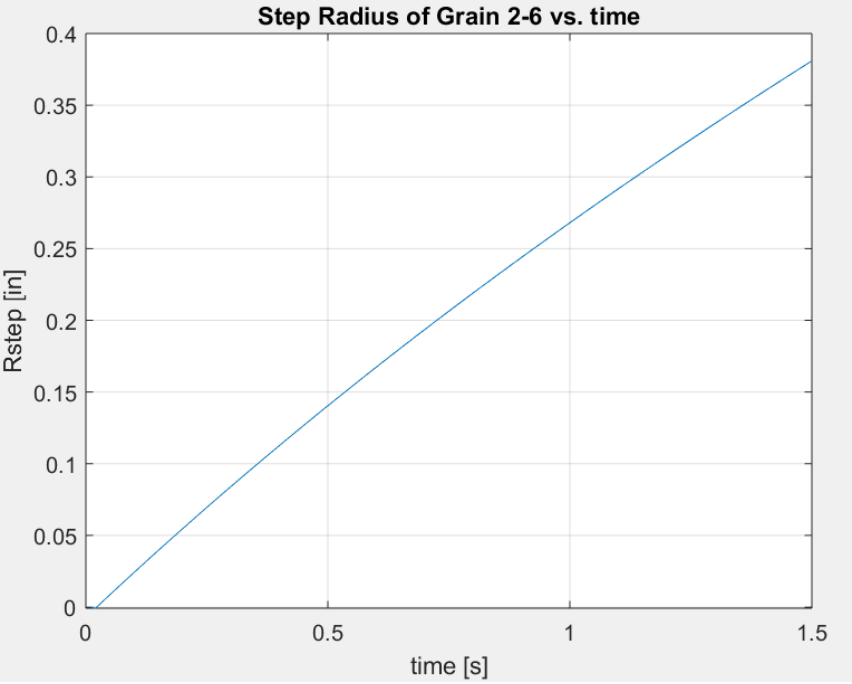
Radius of grain 1 vs. time:



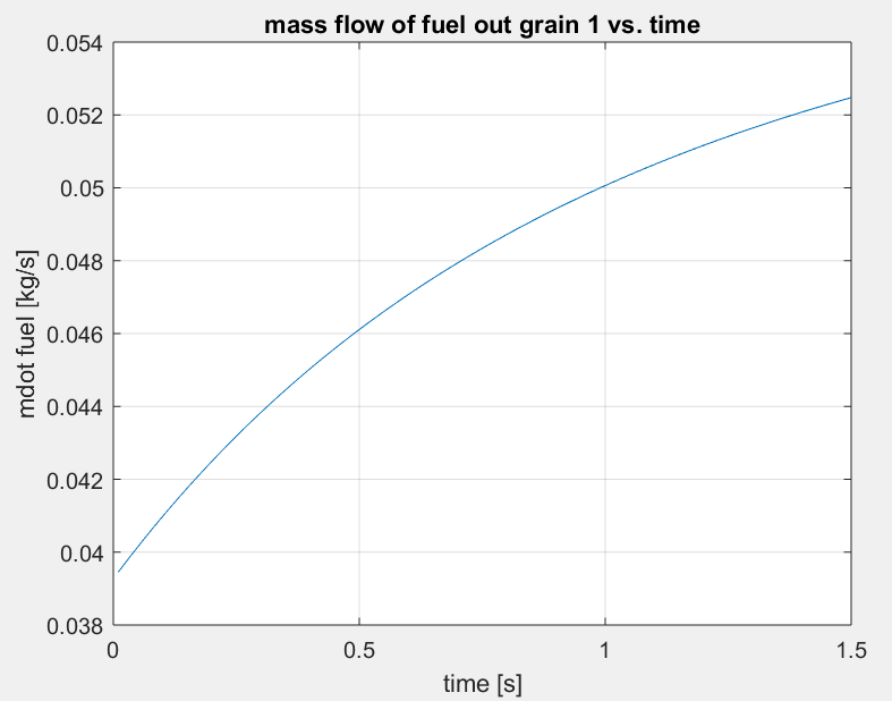
Radius of grain 2-6 vs time:



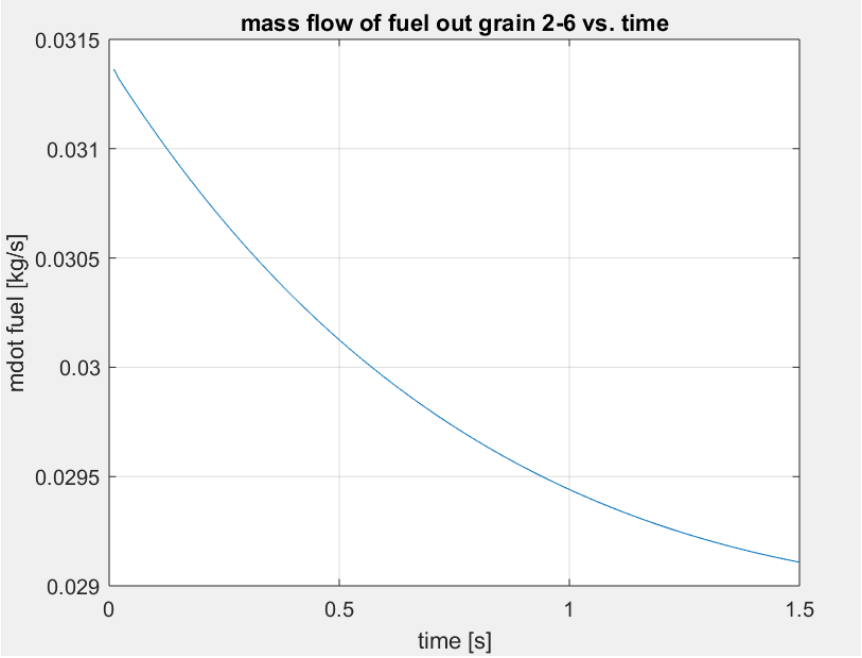
Rstep vs. time:



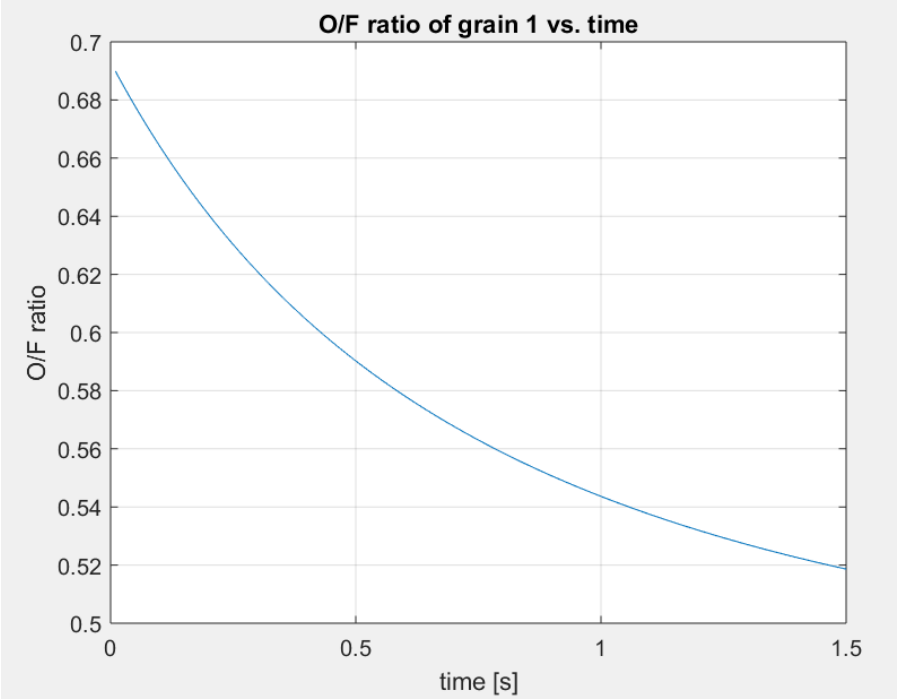
Mass flow of fuel out grain 1 vs time:



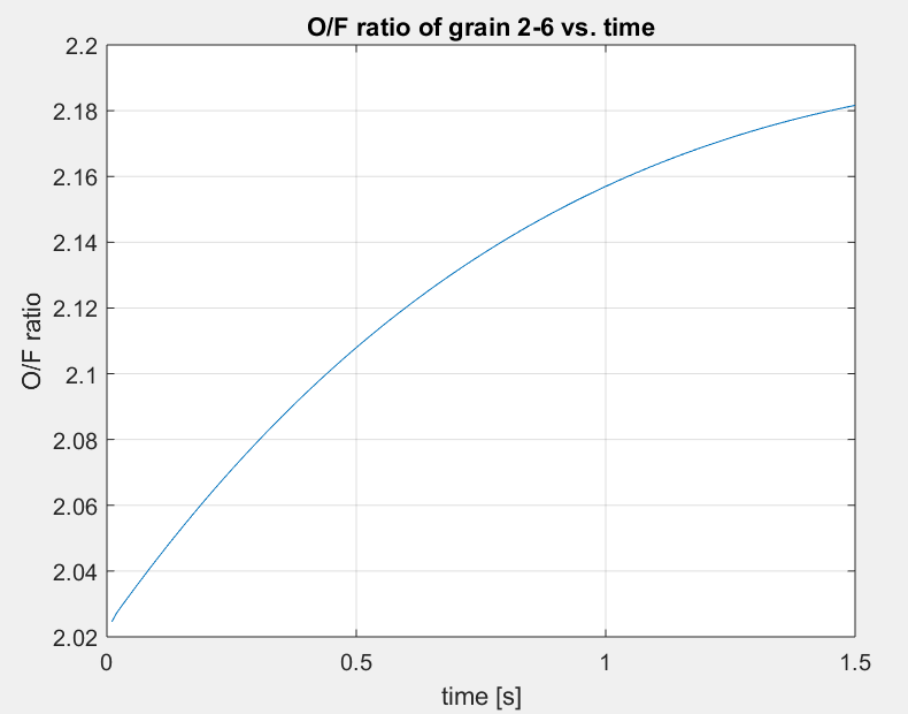
Mass flow of fuel out grain 2-6 vs. time:



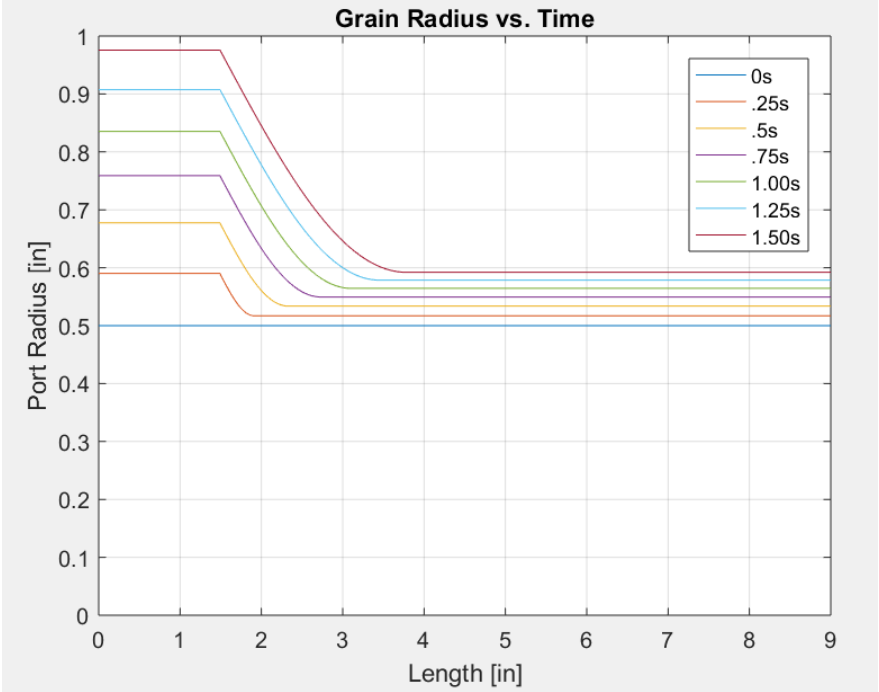
O/F of grain 1 vs. time:



O/F of grain 2-6 vs. time:



Fuel regression vs. Length



Matlab Codes:

Part i):

```
% AAE 539 Homework 5
% part i: assume step burn is inhibited
clear; close all; clc;
% Andrew Albers
% Plot Pc(t), Gox(t) R1(t) R26(t) Rstep(t, mdot1(t), mdot26(t), OF1(t),
% OF26(t) as functions of time over 1.5s

% known values:
mdot_NTO = 0.0907185; % assumed constant oxidizer mass flow rate, kg/s
% First section:
rhof1 = 1310; % kg/m^3
n = .65; % empirical regression constant
% 2-6 sections:
rhof26 = 1110; % kg/m^3
n = .545; % empirical regression constant

At = pi*(.603/2*.0254)^2; % m
Ae = pi*(1.5/2*.0254)^2; % m

burntime = 1.5; % s
totallength = 9*.0254; % m
ID = 2*.0254; % inner diameter, m

% declare initial variables prior to ignition:
R1 = [];
R1(1) = .5*.0254; % m
L1 = 9/6*.0254; % m

R26 = [];
R26(1) = .5*.0254; % m
L26 = 9*5/6*.0254; % m

Ap1 = [];
Ap1(1) = pi*R1(1)^2; % m^2

Ap26 = [];
Ap26(1) = pi*R26(1)^2; % m^2


mdot_total = [];
Pc = [];
Gox = [];

% initial guess for Pc: 200 psia:
% step 1): Run CEA with Pc of 200, gives you Tc... etc.
% initial O/F would be 1:1, just use this as a starting point.
% regression rate multiplied by timestep = regressed grain amount
% calculate G mass flow rate is total ox, calculate regression rate,  grain 1 will be
30% mout, grain 2 will be 70% mout calculate R, get mf
% run CEA every 10 timesteps or so


Pc(1) = 200; % initial assumed pressure of 200 psi, should not change drastically and
is a good starting point

Ab1(1) = 2*pi*R1(1)*L1; % m^2
Ab26(1) = 2*pi*R26(1)*L26; % m^2
Abtotal(1) = Ab1(1)+Ab26(1); % m^2

Gox(1) = mdot_NTO/Ap26(1);
r1(1) = 3.4e-4*Gox(1)^.65; % m/s
r26(1) = 1.1e-4*Gox(1)^.545; % m/s
```

```
mf1(1) = r1(1)*rhof1*Ab1(1); % kg/s
mf26(1) = r26(1)*rhof26*Ab26(1); % kg/s
mdot_total(1) = mdot_NTO; % initial mdot out, kg/s

o_f1(1) = .3*mdot_NTO/mf1(1);
o_f26(1) = .7*mdot_NTO/mf26(1);

Cstar1 = [];
Cstar26 = [];
Cstaravg = [];
Cstar1(1) = 0; % initialize Cstar1
Cstar26(1) = 0; % initialize Cstar26
Cstaravg(1) = 0; % initialize Cstaravg

t = linspace(0.01,1.5,150); % 0.01-1.5s in 150 timesteps

Pcheck = 250; % to determine when CEA should be run again, higher to allow first
iteration check to pass
for i = 2:length(t)

   if abs(Pc(i-1)-Pcheck) >= .01*Pcheck % if current pressure has deviated 10% from
previous CEA run, run CEA again, otherwise, just calculate to reduce runtime
% run CEA for
    Pcheck = Pc(i-1); % reset Pcheck
% Change this variable to true to rerun CEA instead of using saved values
CEA_RUN = true;
CEA_SAVE_FILE = 'cea.mat';

% The CEA MATLAB code takes a MATLAB map (called a dictionary in Python or
% hash in C) as input. The dictionary uses MATLAB character arrays as the
% keys, and the value data type varies by which key is used. Details of
% each key are listed in cea_rocket_run.m
% For example: inp('key') = value.
inp = containers.Map;
inp('type') = 'eq';              % Sets the type of CEA calculation
inp('p') = Pc(i-1);                % Chamber pressure
inp('p_unit') = 'psi';              % Chamber pressure units
inp('o/f') = o_f1(i-1);               % Mixture ratio
% inp('sup') = [Ae/At];               % Supersonic area ratios
% inp('sub') = [5.292 4.704 3.528 2.352 1.176];
inp('fuel') = {'Paraffin_Wax','NaNH2'};             % Fuel name from thermo.inp
inp('fuel_wt%') = [10 90];
inp('fuel_t') = [298 298];                % Fuel inlet temperature
inp('ox') = 'N2O(L),298.15';              % Ox name from thermo.inpj
inp('ox_t') = 298.15;                  % Ox inlet temperature
inp('file_name') = 'hw5_1.inp';    % Input/output file name
% inp('omit') = {'NaH(cr)','Na(L)','Na2O(c)'};
if CEA_RUN
    data = cea_rocket_run(inp);     % Call the CEA MATLAB code
    save(CEA_SAVE_FILE, 'data');
else
    load(CEA_SAVE_FILE);
end

% The output data structure, called 'data' in this case, is also a MATLAB
% map. 'data' contains a single entry for each of the CEA calculation types
% listed ('eq' and 'fr'). For instance, if only 'fr' is listed, then 'data'
% will only contain a single entry under data('fr').
data_eq_1 = data('eq');
Cdata1 = squeeze(data_eq_1('cstar'));
Cstar1(i) = Cdata1(1);

% run CEA for
% Change this variable to true to rerun CEA instead of using saved values
```

```
CEA_RUN = true;
CEA_SAVE_FILE = 'cea.mat';
% The CEA MATLAB code takes a MATLAB map (called a dictionary in Python or
% hash in C) as input. The dictionary uses MATLAB character arrays as the
% keys, and the value data type varies by which key is used. Details of
% each key are listed in cea_rocket_run.m
% For example: inp('key') = value.
inp = containers.Map;
inp('type') = 'eq';              % Sets the type of CEA calculation
inp('p') = Pc(i-1);               % Chamber pressure
inp('p_unit') = 'psi';             % Chamber pressure units
inp('o/f') = o_f26(i-1);              % Mixture ratio
% inp('sup') = [Ae/At];              % Supersonic area ratios
% inp('sub') = [5.292 4.704 3.528 2.352 1.176];
inp('fuel') = {'Paraffin_Wax','NaNH2'};             % Fuel name from thermo.inp
inp('fuel_wt%') = [60 40];
inp('fuel_t') = [298 298];                % Fuel inlet temperature
inp('ox') = 'N2O(L),298.15';              % Ox name from thermo.inpj
inp('ox_t') = 298.15;                   % Ox inlet temperature
inp('file_name') = 'hw5_26.inp';    % Input/output file name
% inp('omit') = {'NaH(cr)','Na(L)','Na2O(c)'};
if CEA_RUN
    data = cea_rocket_run(inp);     % Call the CEA MATLAB code
    save(CEA_SAVE_FILE, 'data');
else
    load(CEA_SAVE_FILE);
end

% The output data structure, called 'data' in this case, is also a MATLAB
% map. 'data' contains a single entry for each of the CEA calculation types
% listed ('eq' and 'fr'). For instance, if only 'fr' is listed, then 'data'
% will only contain a single entry under data('fr').
data_eq_26 = data('eq');
Cdata26= squeeze(data_eq_26('cstar'));
Cstar26(i) = Cdata26(1);

   else
       Cstar1(i) = Cstar1(i-1);
       Cstar26(i) = Cstar26(i-1);
   end
% calculate Gox, r1, r26, mf1, mf2

% Calculate new R at timestep i:
R1(i) = R1(i-1)+r1(i-1)*.01; % m
R26(i) = R26(i-1)+r26(i-1)*.01; % m

% calculate new burn area for grain 1:
Ab1(i) = 2*pi*R1(i)*L1; % m^2
% calculate new burn area for grain 2-6:
Ab26(i) = 2*pi*R26(i)*L26; % m^2
% calculate total burn area for Gox calculation: (ignore step burn area for
Ap1(i) = pi*R1(i)^2;
Ap26(i) = pi*R26(i)^2;
Apavg(i) = 1/6*Ap1(i) + 5/6*Ap26(i);
% now)
Abtotal(i) = Ab1(i)+Ab26(i);

% Calculate new Gox at timestep i:
Gox(i) = mdot_NTO/Apavg(i);

% Calculate new regression rate of grain 1:
r1(i) = 3.4e-4*Gox(i)^.65; % m/s
% Calculate new regression rate of grain 2-6:
r26(i) = 1.1e-4*Gox(i)^.545; % m/s
```

```
% calculate new fuel mass flow rate of grain 1:
mf1(i) = r1(i)*rhof1*Ab1(i); % kg/s
% calculate new fuel mass flow rate of grain 2-6:
mf26(i) = r26(i)*rhof26*Ab26(i); % kg/s
% calculate total mdot of fuel of system:
mftotal(i) = mf1(i)+mf26(i); % kg/s
% calculate total mdot out of system
mdot_total(i) = (mf1(i)+mf26(i)+mdot_NTO);
% calculate new o/f ratio to be used in CEA next iteration:
o_f1(i) = .3*mdot_NTO/mf1(i);
o_f26(i) = .7*mdot_NTO/mf26(i);
%
% weight1(i) = mf1(i)/mftotal(i);
% weight26(i) = mf26(i)/mftotal(i);
% must average out Cstar before calculating Pc:
Cstaravg(i) = 1/6*Cstar1(i) + 5/6*Cstar26(i);
% calculate new Pc to be used in CEA next iteration:
Pc(i) = Cstaravg(i)*mdot_total(i)/At; % Pa
Pc(i) = Pc(i)*0.000145038; % convert to psi
end

% plots:

figure
plot(t,Pc);
title('Chamber Pressure vs. Time')
xlabel('time [s]')
ylabel('Pressure [psi]')
grid on

figure
plot(t,Gox)
title('Gox vs time')
xlabel('time [s]')
ylabel('Gox [kg/m^2/s]')
grid on

figure
plot(t,R1*39.3701)
title('Radius of Grain 1 vs. time')
xlabel('time [s]');
ylabel('R1 [in]');
grid on

figure
plot(t,R26*39.3701)
title('Radius of Grain 2-6 vs. time')
xlabel('time [s]');
ylabel('R2-6 [in]');
grid on

figure
plot(t,mf1)
title('mass flow of fuel out grain 1 vs. time')
xlabel('time [s]');
ylabel('mdot fuel [kg/s]');
grid on

figure
plot(t,mf26)
title('mass flow of fuel out grain 2-6 vs. time')
xlabel('time [s]');
ylabel('mdot fuel [kg/s]');
grid on
```

```
figure
plot(t,o_f1)
title('O/F ratio of grain 1 vs. time')
xlabel('time [s]');
ylabel('O/F ratio');
grid on

figure
plot(t,o_f26)
title('O/F ratio of grain 2-6 vs. time')
xlabel('time [s]');
ylabel('O/F ratio');
grid on

% part ii) : Provide a graphical output of the regression rate along the
% length of the grain:

Lplot = linspace(0,9,90);
R1plot = [R1(1) R1(25) R1(50) R1(75) R1(100) R1(125) R1(150)]./.0254;
R26plot = [R26(1) R26(25) R26(50) R26(75) R26(100) R26(125) R26(150)]./.0254;

R_0s = [linspace(R1plot(1),R1plot(1),15) linspace(R26plot(1),R26plot(1),75)];
R_25s = [linspace(R1plot(2),R1plot(2),15) linspace(R26plot(2),R26plot(2),75)];
R_50s = [linspace(R1plot(3),R1plot(3),15) linspace(R26plot(3),R26plot(3),75)];
R_75s = [linspace(R1plot(4),R1plot(4),15) linspace(R26plot(4),R26plot(4),75)];
R_100s = [linspace(R1plot(5),R1plot(5),15) linspace(R26plot(5),R26plot(5),75)];
R_125s = [linspace(R1plot(6),R1plot(6),15) linspace(R26plot(6),R26plot(6),75)];
R_150s = [linspace(R1plot(7),R1plot(7),15) linspace(R26plot(7),R26plot(7),75)];

figure
plot(Lplot,R_0s,Lplot,R_25s,Lplot,R_50s,Lplot,R_75s,Lplot,R_100s,Lplot,R_125s,Lplot,R_
150s);
title('Grain Radius vs. Time')
xlabel('Length [in]')
ylabel('Port Radius [in]')
legend('0s','.25s','.5s','.75s','1.00s','1.25s','1.50s')
ylim([0 1])
grid on
```

Matab Part ii):

```
% AAE 539 Homework 5
% part i: assume step burn is inhibited
clear; close all; clc;
% Andrew Albers
% Plot Pc(t), Gox(t) R1(t) R26(t) Rstep(t, mdot1(t), mdot26(t), OF1(t),
% OF26(t) as functions of time over 1.5s

% known values:
mdot_NTO = 0.0907185; % assumed constant oxidizer mass flow rate, kg/s
% First section:
rhof1 = 1310; % kg/m^3
n = .65; % empirical regression constant
% 2-6 sections:
rhof26 = 1110; % kg/m^3
n = .545; % empirical regression constant

At = pi*(.603/2*.0254)^2; % m
Ae = pi*(1.5/2*.0254)^2; % m

burntime = 1.5; % s
totallength = 9*.0254; % m
ID = 2*.0254; % inner diameter, m

% declare initial variables prior to ignition:
R1 = [];
R1(1) = .5*.0254; % m
L1 = 9/6*.0254; % m

R26 = [];
R26(1) = .5*.0254; % m
L26 = 9*5/6*.0254; % m

Ap1 = [];
Ap1(1) = pi*R1(1)^2; % m^2

Ap26 = [];
Ap26(1) = pi*R26(1)^2; % m^2


mdot_total = [];
Pc = [];
Gox = [];

% initial guess for Pc: 200 psia:
% step 1): Run CEA with Pc of 200, gives you Tc... etc.
% initial O/F would be 1:1, just use this as a starting point.
% regression rate multiplied by timestep = regressed grain amount
% calculate G mass flow rate is total ox, calculate regression rate,  grain 1 will be 30% mout,
grain 2 will be 70% mout calculate R, get mf
% run CEA every 10 timesteps or so


Pc(1) = 200; % initial assumed pressure of 200 psi, should not change drastically and is a good
starting point

Ab1(1) = 2*pi*R1(1)*L1; % m^2
Ab26(1) = 2*pi*R26(1)*L26; % m^2
Abtotal(1) = Ab1(1)+Ab26(1); % m^2

Gox(1) = mdot_NTO/Ap26(1);
r1(1) = 3.4e-4*Gox(1)^.65; % m/s
r26(1) = 1.1e-4*Gox(1)^.545; % m/s

mf1(1) = r1(1)*rhof1*Ab1(1); % kg/s
mf26(1) = r26(1)*rhof26*Ab26(1); % kg/s
mdot_total(1) = mdot_NTO; % initial mdot out, kg/s

o_f1(1) = .3*mdot_NTO/mf1(1);
o_f26(1) = .7*mdot_NTO/mf26(1);

Cstar1 = [];
Cstar26 = [];
Cstaravg = [];
```

```
Cstar1(1) = 0; % initialize Cstar1
Cstar26(1) = 0; % initialize Cstar26
Cstaravg(1) = 0; % initialize Cstaravg

t = linspace(0.01,1.5,150); % 0.01-1.5s in 150 timesteps

Pcheck = 250; % to determine when CEA should be run again, higher to allow first iteration check
to pass
for i = 2:length(t)

   if abs(Pc(i-1)-Pcheck) >= .05*Pcheck % if current pressure has deviated 10% from previous CEA
run, run CEA again, otherwise, just calculate to reduce runtime
% run CEA for
    Pcheck = Pc(i-1); % reset Pcheck
% Change this variable to true to rerun CEA instead of using saved values
CEA_RUN = true;
CEA_SAVE_FILE = 'cea.mat';

% The CEA MATLAB code takes a MATLAB map (called a dictionary in Python or
% hash in C) as input. The dictionary uses MATLAB character arrays as the
% keys, and the value data type varies by which key is used. Details of
% each key are listed in cea_rocket_run.m
% For example: inp('key') = value.
inp = containers.Map;
inp('type') = 'eq';              % Sets the type of CEA calculation
inp('p') = Pc(i-1);                % Chamber pressure
inp('p_unit') = 'psi';              % Chamber pressure units
inp('o/f') = o_f1(i-1);               % Mixture ratio
% inp('sup') = [Ae/At];               % Supersonic area ratios
% inp('sub') = [5.292 4.704 3.528 2.352 1.176];
inp('fuel') = {'Paraffin_Wax','NaNH2'};             % Fuel name from thermo.inp
inp('fuel_wt%') = [10 90];
inp('fuel_t') = [298 298];               % Fuel inlet temperature
inp('ox') = 'N2O(L),298.15';             % Ox name from thermo.inpj
inp('ox_t') = 298.15;                  % Ox inlet temperature
inp('file_name') = 'hw5_1.inp';    % Input/output file name
inp('omit') = {'NaH(cr)','Na(L)','Na2O(c)'};
if CEA_RUN
    data = cea_rocket_run(inp);     % Call the CEA MATLAB code
    save(CEA_SAVE_FILE, 'data');
else
    load(CEA_SAVE_FILE);
end

% The output data structure, called 'data' in this case, is also a MATLAB
% map. 'data' contains a single entry for each of the CEA calculation types
% listed ('eq' and 'fr'). For instance, if only 'fr' is listed, then 'data'
% will only contain a single entry under data('fr').
data_eq_1 = data('eq');
Cdata1 = squeeze(data_eq_1('cstar'));
Cstar1(i) = Cdata1(1);

% run CEA for
% Change this variable to true to rerun CEA instead of using saved values
CEA_RUN = true;
CEA_SAVE_FILE = 'cea.mat';
% The CEA MATLAB code takes a MATLAB map (called a dictionary in Python or
% hash in C) as input. The dictionary uses MATLAB character arrays as the
% keys, and the value data type varies by which key is used. Details of
% each key are listed in cea_rocket_run.m
% For example: inp('key') = value.
inp = containers.Map;
inp('type') = 'eq';              % Sets the type of CEA calculation
inp('p') = Pc(i-1);                % Chamber pressure
inp('p_unit') = 'psi';              % Chamber pressure units
inp('o/f') = o_f26(i-1);               % Mixture ratio
% inp('sup') = [Ae/At];               % Supersonic area ratios
% inp('sub') = [5.292 4.704 3.528 2.352 1.176];
inp('fuel') = {'Paraffin_Wax','NaNH2'};             % Fuel name from thermo.inp
inp('fuel_wt%') = [60 40];
inp('fuel_t') = [298 298];               % Fuel inlet temperature
inp('ox') = 'N2O(L),298.15';             % Ox name from thermo.inpj
inp('ox_t') = 298.15;                  % Ox inlet temperature
```

```
inp('file_name') = 'hw5_26.inp';    % Input/output file name
inp('omit') = {'NaH(cr)','Na(L)','Na2O(c)'};
if CEA_RUN
    data = cea_rocket_run(inp);     % Call the CEA MATLAB code
    save(CEA_SAVE_FILE, 'data');
else
    load(CEA_SAVE_FILE);
end

% The output data structure, called 'data' in this case, is also a MATLAB
% map. 'data' contains a single entry for each of the CEA calculation types
% listed ('eq' and 'fr'). For instance, if only 'fr' is listed, then 'data'
% will only contain a single entry under data('fr').
data_eq_26 = data('eq');
Cdata26= squeeze(data_eq_26('cstar'));
Cstar26(i) = Cdata26(1);

   else
       Cstar1(i) = Cstar1(i-1);
       Cstar26(i) = Cstar26(i-1);
   end
% calculate Gox, r1, r26, mf1, mf2

% Calculate new R at timestep i:
R1(i) = R1(i-1)+r1(i-1)*.01; % m
R26(i) = R26(i-1)+r26(i-1)*.01; % m
Rstep(i) = R1(i-1)-R26(i-1)-r26(i-1)*.01; % m
L26(i) = L26(1)-Rstep(i); % calculate horizontal section length
% calculate new burn area for grain 1:
Ab1(i) = 2*pi*R1(i)*L1; % m^2
% calculate new burn area for grain 2-6:
Ab26(i) = 2*pi*R26(i)*L26(i) + pi^2*(R1(i)*Rstep(i)); % added surface area of quarter torus
region, m^2

% calculate total burn area for Gox calculation: (ignore step burn area for
Ap1(i) = pi*R1(i)^2;
Ap26(i) = pi*R26(i)^2;
Apavg(i) = 1/6*Ap1(i) + 5/6*Ap26(i);
% calculate Abstep:

Abtotal(i) = Ab1(i)+Ab26(i);

% Calculate new Gox at timestep i:
Gox(i) = mdot_NTO/Apavg(i);

% Calculate new regression rate of grain 1:
r1(i) = 3.4e-4*Gox(i)^.65; % m/s
% Calculate new regression rate of grain 2-6:
r26(i) = 1.1e-4*Gox(i)^.545; % m/s


% calculate new fuel mass flow rate of grain 1:
mf1(i) = r1(i)*rhof1*Ab1(i); % kg/s
% calculate new fuel mass flow rate of grain 2-6:
mf26(i) = r26(i)*rhof26*Ab26(i); % kg/s
% calculate total mdot of fuel of system:
mftotal(i) = mf1(i)+mf26(i); % kg/s
% calculate total mdot out of system
mdot_total(i) = (mf1(i)+mf26(i)+mdot_NTO);
% calculate new o/f ratio to be used in CEA next iteration:
o_f1(i) = .3*mdot_NTO/mf1(i);
o_f26(i) = .7*mdot_NTO/mf26(i);
%
% weight1(i) = mf1(i)/mftotal(i);
% weight26(i) = mf26(i)/mftotal(i);
% must average out Cstar before calculating Pc:
Cstaravg(i) = 1/6*Cstar1(i) + 5/6*Cstar26(i);
% calculate new Pc to be used in CEA next iteration:
Pc(i) = Cstaravg(i)*mdot_total(i)/At; % Pa
Pc(i) = Pc(i)*0.000145038; % convert to psi
end

% plots:
```

```
figure
plot(t,Pc);
title('Chamber Pressure vs. Time');
xlabel('time [s]');
ylabel('Pressure [psi]');
grid on

figure
plot(t,Gox)
title('Gox vs time')
xlabel('time [s]')
ylabel('Gox [kg/m^2/s]')
grid on

figure
plot(t,R1*39.3701)
title('Radius of Grain 1 vs. time')
xlabel('time [s]');
ylabel('R1 [in]');
grid on

figure
plot(t,R26*39.3701)
title('Radius of Grain 2-6 vs. time')
xlabel('time [s]');
ylabel('R2-6 [in]');
grid on

figure
plot(t, Rstep*39.3701)
title('Step Radius of Grain 2-6 vs. time')
xlabel('time [s]')
ylabel('Rstep [in]')
grid on

figure
plot(t,mf1)
title('mass flow of fuel out grain 1 vs. time')
xlabel('time [s]');
ylabel('mdot fuel [kg/s]');
grid on

figure
plot(t,mf26)
title('mass flow of fuel out grain 2-6 vs. time')
xlabel('time [s]');
ylabel('mdot fuel [kg/s]');
grid on

figure
plot(t,o_f1)
title('O/F ratio of grain 1 vs. time')
xlabel('time [s]');
ylabel('O/F ratio');
grid on

figure
plot(t,o_f26)
title('O/F ratio of grain 2-6 vs. time')
xlabel('time [s]');
ylabel('O/F ratio');
grid on

% part ii) : Provide a graphical output of the regression rate along the
% length of the grain:

Lplot = linspace(0,9,900);
R1plot = [R1(1) R1(25) R1(50) R1(75) R1(100) R1(125) R1(150)]./.0254;
R26plot = [R26(1) R26(25) R26(50) R26(75) R26(100) R26(125) R26(150)]./.0254;
Rstepplot = [Rstep(1) Rstep(25) Rstep(50) Rstep(75) Rstep(100) Rstep(125) Rstep(150)]./.0254;
% determine L distance points covered by rounded corner:
Lstep = [0 42 84 124 161 196 229];
dtheta = [0 90/42 90/84 90/124 90/161 90/196 90/229];
theta25s = linspace(0,90,42);
```

```
theta50s = linspace(0,90,84);
theta75s = linspace(0,90,124);
theta100s = linspace(0,90,161);
theta125s = linspace(0,90,196);
theta150s = linspace(0,90,229);

    Rcurve_25s = R26plot(2)+Rstepplot(2) - Rstepplot(2).*sind(theta25s);

    Rcurve_50s = R26plot(3)+Rstepplot(3) - Rstepplot(3).*sind(theta50s);

    Rcurve_75s = R26plot(4)+Rstepplot(4)- Rstepplot(4).*sind(theta75s);

    Rcurve_100s = R26plot(5)+Rstepplot(5)- Rstepplot(5).*sind(theta100s);

    Rcurve_125s = R26plot(6)+Rstepplot(6)- Rstepplot(6).*sind(theta125s);

    Rcurve_150s = R26plot(7)+Rstepplot(7)- Rstepplot(7).*sind(theta150s);


R_0s = [linspace(R1plot(1),R1plot(1),150)  linspace(R26plot(1),R26plot(1),750)];
R_25s = [linspace(R1plot(2),R1plot(2),150) Rcurve_25s linspace(R26plot(2),R26plot(2),750-
Lstep(2))];
R_50s = [linspace(R1plot(3),R1plot(3),150) Rcurve_50s linspace(R26plot(3),R26plot(3),750-
Lstep(3))];
R_75s = [linspace(R1plot(4),R1plot(4),150) Rcurve_75s linspace(R26plot(4),R26plot(4),750-
Lstep(4))];
R_100s = [linspace(R1plot(5),R1plot(5),150) Rcurve_100s linspace(R26plot(5),R26plot(5),750-
Lstep(5))];
R_125s = [linspace(R1plot(6),R1plot(6),150) Rcurve_125s linspace(R26plot(6),R26plot(6),750-
Lstep(6))];
R_150s = [linspace(R1plot(7),R1plot(7),150) Rcurve_150s linspace(R26plot(7),R26plot(7),750-
Lstep(7))];

figure
plot(Lplot,R_0s,Lplot,R_25s,Lplot,R_50s,Lplot,R_75s,Lplot,R_100s,Lplot,R_125s,Lplot,R_150s);
title('Grain Radius vs. Time')
xlabel('Length [in]')
ylabel('Port Radius [in]')
legend('0s','.25s','.5s','.75s','1.00s','1.25s','1.50s')
ylim([0 1])
grid on
```