ASEN 2004 - Lab 2
Static Test Stand Performance Analysis
April 19, 2017

Marshall Herr, [*] Richard Moon, [†] Nicholas Renninger, [‡]
University of Colorado - Boulder

Specific impulse and thrust determination is an important concept. Specific impulse allows rocket scientists to determine the efficiency of their rockets while thrust allows them to determine the force their rockets put out. These two combined are what gives rockets the ability to fly. Without a thrust to weight ratio greater than one, a rocket would never leave the ground. However, without a respectable specific impulse, that rocket will never make it to orbit. This makes finding thrust and specific impulse paramount in the aerospace industry. So too is it important for this lab. In order to predict a bottle rocket's flight path, the thrust and specific impulse must be known. This report characterizes an algorithm for determining a specific impulse for static tests which reports a mean specific impulse of 1.25 seconds, mean maximum thrust of 266 N, and mean thrust time of 0.284 seconds. Furthermore, with a standard deviation of the specific impulse found to be 0.184 seconds, it is predicted that 14 datasets would decrease the 95% confidence interval to within 0.1 seconds.

Nomenclature

$\bar{X}$	= Sample Mean of the Data Set [s]
δ	= Confidence Interval (C.I.)
σ	= Standard Deviation of the Data Set [s]
I_{SP}	= Specific Impulse [s]
N	= Number of Trials Conducted
SEM	= Standard Error of the Mean [s]
T	= Thrust [N]
t	= Time [s]
z	= Statistical Constant Relating SEM to a δ

[*]104725592
[†]105675495
[‡]105492876

Contents

List of Figures

List of Tables

I. Introduction and Theory

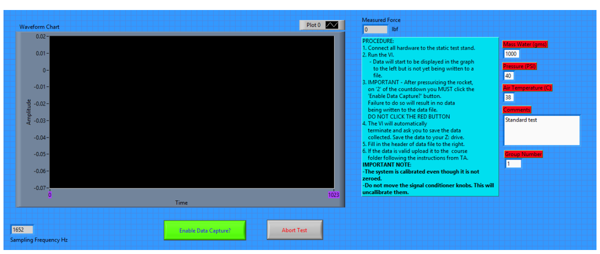
Rockets use thrust to move. Rocket thrust is the reaction force, from Newton's Third Law of Motion, that acts upon the main rocket body in equal magnitude and opposite direction of the action force. The action force comes from expelling propellant mass away from the rocket, and is calculated from Newton's Second Law of Motion, as the net force is equal to the product of the mass and the acceleration of the mass relative to the inertial frame. Rocket thrust is measured in order to find the magnitude of force acting upon the rocket, which is important as usually rockets are used to move a system upwards, which requires the rocket to generate enough vertical thrust to overcome the force due to gravity. Typically, rocket trust is measured with static test stands, which are testing apparatuses that hold down the rocket to prevent a change in position and measures the force the static stand applies to the rocket to keep the net force on the rocket as 0. Measuring thrust is important, as thrust is used to find the specific impulse of a rocket.

Specific impulse is a type of measurement used to describe how effective a rocket is in generating thrust per unit mass of propellant. It can be defined as the thrust generated per weight flow rate of the propellant used. Weight flow rate can also be written as the product of mass flow rate and the standard gravity constant.[1] It is also equivalent to the total impulse over the weight of the propellant used. It is usually found by measuring the thrust of a rocket over an amount of time, finding the approximate total impulse by integrating the force function in respects to time, and by dividing the total impulse with the weight of the propellant. The total weight of the propellant is found by finding the mass of the propellant used and multiplying by the standard gravity constant.

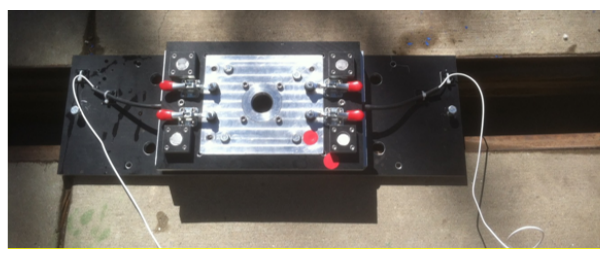
The standard error of the mean (SEM) is used to show how accurate a sample data point is in representing the mean of the data. The SEM is defined as the standard deviation of the mean over the square root of the number of trials, and the resulting SEM can be used to find the confidence interval. Both the SEM and confidence interval are explained in the results section. The number of tests required to establish the desired degree of confidence in the results was 14 trials.

II. Materials and Methods

Below in Fig. 1 are two pictures detailing the Labview Code and Static Test Stand Apparatus.



(a) Labview Software. *Data collection system.*



(b) Static Test Stand. *Load sensor apparatus.*

Figure 1: Static Test Elements

The LabVIEW^{TM} software is collecting the data from the Test Stand's load cells. It plots this data on the graph as a Thrust vs Time curve. Water mass, pressure, temperature, group number, and any comments one has are entered into the comment fields on the right. Two seconds before the launch, the "Enable Data Capture?" button is pressed, enabling data capture. Data is recorded by the software, which ends the data collection automatically and requests a name and location for the data save.

The Static Test Stand is composed of two sections: the launch clamp and the base. The launch clamp is the silver piece that may be seen in Fig. 1. Its purpose is to provide a housing where the bottle may be attached. This launch clamp is attached to the base using the four red clamps which restricts its movement. When the launch plug is inserted into the rocket, the release bracket assembly may be installed allowing for quick release of the launch plug. The launch plug contains a one way valve attached to a hose which may be used to pressurize the rocket. The base contains the load cells which measure the forces being applied by the base on the launch clamp. These sensors attach to the computer running the LabVIEW^{TM} Software. The base is also attached to the ground so that it cannot move.

When the rocket is properly attached to the launch clamp and pressurized, launch may begin. In order to launch, safety glasses must be put on, and a countdown must be started. At 2 seconds, the data collection is enabled. When the countdown ends, the release bracket it pulled from the launch clamp, releasing the launch plug. the pressure in the rocket ejects the launch plug from the throat of the bottle, beginning the rocket thrust. The rocket thrust causes the a force on the rocket, which causes a force on the launch clamp, which causes a force on the base. this force on the base is measured by the load cells and recorded by the LabVIEW^{TM} Software until the thrusting ends.

Since the load cells automatically zero themselves to ambient conditions, as the rocket is thrusting, the load cells will begin to adjust themselves to the rocket thrust and will under-report the magnitude of thrust the rocket is producing. In order to compensate for this, a line is drawn between the first data point at no thrust and the final thrust data point, which is negative due to the load cell adjustment. By shifting all of the data points up by the amount the line beneath them is below zero, the load cell adjustment is corrected. Then, by using `trapz.m` to find the area under the curve of the entire thrusting section, the total impulse is calculated. If the total impulse is divided by the weight of the water ejected and gravity at sea level, the specific impulse is found.

III. Results

Before any analysis could begin, the data was processed using the algorithm described in Section II. This processed data, while generally acceptable, still included some outlying trial data. For two of the data sets, the total time to thrust was far below the rest of the data, with less than half of the time to thrust of the rest of the data. With these outlying data sets removed, the ISP for each data set was calculated, as well as the peak thrust for each trial. This data can be found in Table 1 in Appendix B. The average ISP was calculated to be 1.25 s with a standard deviation of 0.184s, and the average peak thrust was found to be 266N with a standard deviation of 39N. The total time to thrust, the amount of time between firing the rocket and the stoppage of thrust, is plotted in a histogram in Figure 3b. The time to thrust is approximately normally distributed about 0.285s, and demonstrates the effectiveness of the data processing algorithm used. By consistently processing the data, the thrust curves were well aligned, as shown in Figure 2, and allowed the accurate calculation of the pertinent values listed previously discussed.

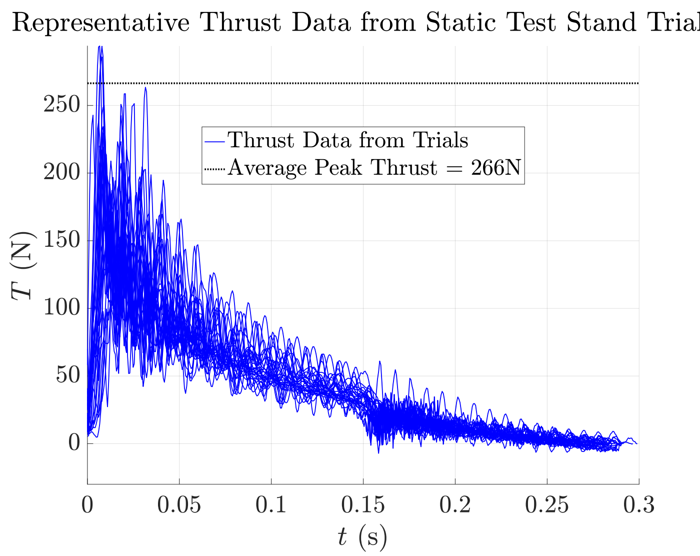
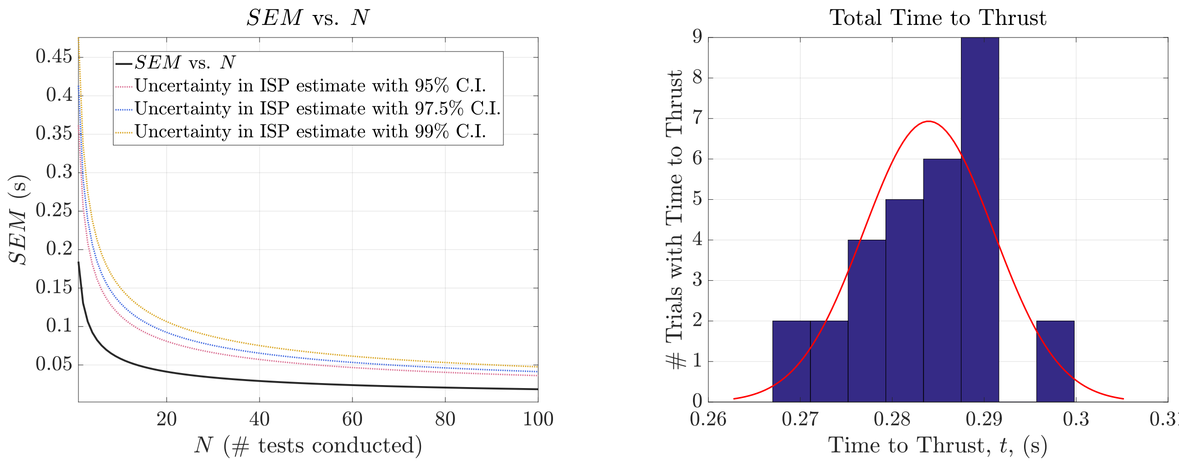


Figure 2: Plot of Thrust Curves from Representative Static Test Data.

Another important part of the results of analysis was the statistical analysis of the data. This primarily consisted of computation of the SEM, and the corresponding confidence intervals (C.I.s) about the mean of the calculated ISPs. The SEM, formally defined as $\frac{\sigma}{\sqrt{N}}$, describes the variance of the sample mean of a data set with respect to means computed using other data sets; SEM is used to determine the validity of using a sample mean to estimate the actual mean of the data.[2] This measure of variance of the sample means during multiple trials can be used to calculate a confidence interval about the mean. Confidence intervals, δ, are defined as the area about a sampled mean that encompass a certain probability that this area about the sampled mean contains the actual mean of the data. More formally the confidence interval about a mean value is defined as: $\bar{X} \pm \delta = \bar{X} \pm z * SEM$, where the δ term essentially captures how much variance there must be in the mean to ensure that the mean captures the actual mean for a given probability. This allows the estimation of how many trials need to be collected to have a given probability of capturing the data for a given δ.

For this lab, the SEM was computed for the full data set, and was found to be 0.0321s, and the SEM computed as a function of N is shown in Figure 3a. This plot shows that as more trials are completed, a given confidence interval can be reached with higher probability. Using this analysis, the ISP can be calculated to 0.1s of the real mean with only 14 trials (C.I. = 95%), 18 trials (C.I. = 97.5%), and 23 trials (C.I. =

99%). To estimate the ISP to within 0.01s of the real mean with 1307 trials (C.I. = 95%), 1707 trials (C.I. = 97.5%), and 2265 trials (C.I. = 99%). As can be seen from these results and Figure 3a, getting very precise measurements of the mean requires exponentially more trials for a given increase in precision.



(a) Plot of SEM of the ISP as a function of N. *The uncertainty in the mean of the ISP values is plotted with different C.I.s.*

(b) Distribution of the total amount of time the rocket thrusted. *The binning shows here that the data approximately fits a normal dist. about $t = 0.285s$.*

Figure 3: Statistics of Test Stand Data

IV. Conclusions

Through our use of the SEM, the number of data points needed to have a reasonably accurate prediction of the ISP can be found. As the Monte Carlo Simulation predicts up to 3σ of uncertainty in the trajectory of the bottle rocket, choosing the 95% confidence interval for a 0.1s uncertainty in the ISP (a reasonably high accuracy) would mean that about 14 trials would be needed to ensure the 0.1s accuracy desired.

This 95% confidence interval relates to the flight performance model as such: when the ISP data is varied by 0.1 s about the mean I_{SP}, the 95% confidence interval that the Monte Carlo analysis returns should contain 95% of the I_{SP} data points collected.

References

[1]Price, E. W., and Biblarz, O., Rocket, Jet-Propulsion Device and Vehicle, Jan. 2011, pp. 12.

[2]Taylor, J. R., An Introduction to Error Analysis: the Study of the Uncertainties in Physical Measurements, Sausalito, CA: University Science, 1997.

Acknowledgments

We would like to thank Prof. Clark for his help in developing our algorithms, as well as Trudy, Bobby and the class staff for setting up and maintaining all of the test equipment.

V. Appendix A: Group Member Contributions

The primary contributions of each lab member are as follows:

Marshall Herr: Code, Document, Transportation.

Richard Moon: Derivations, Intro/theory, Research.

Nick Renninger: Code, Document, All-natural Soda Provider.

VI. Appendix B: Table

Table 1: Table of Specific Impulse and Peak Thrust for each Lab Group's data

Lab Group Number	I_{SP} $[s]$	**Peak Thrust** $[N]$
3	1.158	263.693
4	0.939	185.506
4	1.190	195.215
5	1.126	245.490
6	1.088	203.661
7	1.084	240.724
8	1.139	276.948
9	1.169	263.872
10	1.202	288.661
11	1.268	272.935
13	1.103	231.533
14	1.175	274.513
15	1.256	255.286
17	1.158	261.624
18	1.137	239.757
19	1.132	244.419
20	1.195	263.956
21	1.183	243.353
22	1.118	282.423
22	1.084	265.526
23	1.523	239.683
24	1.577	314.872
25	1.482	321.832
26	1.352	292.471
27	1.522	323.12
28	1.302	315.898
29	1.337	240.232
29	0.874	213.519
30	1.594	269.073
31	1.488	300.771
32	1.482	283.042
33	1.408	336.118
34	1.470	341.319

VII. Appendix C: MATLAB Code

VII.A. STS_Main.m – Main Code

```
%%% Main code to Analyze the Static Test Stand data, made by the
%%% bomb-digiest lab group.
%%%
%%% Authors: Nicholas Renninger & Marshall Herr
%%% Date Created: 04/17/2017
%%% Last Modified: 04/19/2017

clear
clc
close all

%% Setup
N_BINS = 8;
shouldSaveFigures = true;

saveLocation = '../Figures/';

figName1 = 'Thrust Plot';
saveTitle1 = cat(2, saveLocation, sprintf('%s.pdf', figName1));

figName2 = 'Total Time to Thrust';
saveTitle2 = cat(2, saveLocation, sprintf('%s.pdf', figName2));

figName3 = 'SEM Plot';
saveTitle3 = cat(2, saveLocation, sprintf('%s.pdf', figName3));

set(0, 'defaulttextinterpreter', 'latex');

colorVecs =     {[0.156863 0.156863 0.156863], ... % sgivery dark grey
                 [0.858824 0.439216 0.576471], ... % palevioletred
                 [0.254902 0.411765 0.882353], ... % royal blue
                 [0.854902 0.647059 0.12549]}; % golden rod


FONTSIZE = 28;


%% Load in Data from Each Test
disp('Reading in Data...')
data = loadInData;
N = length(data); % number of tests performed

idx_offset = 0;
hFig = figure('name', figName1);
scrz = get(groot, 'ScreenSize');
set(hFig, 'Position', scrz)

% loop through every test
for i = 1:N

    % Get Total Time of Thrust
    total_time_thrust_vec(i + idx_offset) = max(data{i}.time) - ...
```

```
                                                min(data{i}.time);

    current_time = total_time_thrust_vec(i + idx_offset);

    % Pull Out Data
    ISP_vec(i) = data{i}.ISP;
    GRP_nums(i) = data{i}.group_num;
    water_mass_vec(i) = data{i}.water_mass;
    pressure_vec(i) = data{i}.pressure;
    temp_vec(i) = data{i}.temp;

    % Get Peak Thrust
    peak_thrust_vec(i) = max(data{i}.thrust);

    % dont record data that takes longer than 0.32s or shorter than 0.22s
    % to thrust
    if  current_time < 0.3 && current_time > 0.25

        thrust_vecs{i + idx_offset} = data{i}.thrust;
        time_vecs{i + idx_offset} = data{i}.time;

        % Plot F vs. t
        hold on
        p1 = plot(time_vecs{i + idx_offset}, ...
             smooth(thrust_vecs{i + idx_offset}), 'b', 'linewidth', 1);


    else % delete data point and reindex one data point less
        idx_offset = idx_offset - 1;
    end



end

disp('Finished')



%% ISP for Each Test
GRP_nums = GRP_nums';
ISP_vec = ISP_vec';
table(GRP_nums, ISP_vec)
avg_ISP = mean(ISP_vec);
std_dev_ISP = std(ISP_vec);


%% Peak Thrust for Each Test
peak_thrust_vec =  peak_thrust_vec';
table(GRP_nums, peak_thrust_vec)

avg_peak_thrust = mean(peak_thrust_vec);
std_dev_peak_thrust = std(peak_thrust_vec);
```

```
%% Print these Results after Table

% print to command window
fprintf('Average ISP: %0.3gs\n', avg_ISP)
fprintf('Std. Dev. of Data Set: %0.3gs\n\n', std_dev_ISP)

% print to command window
fprintf('Average Peak Thrust is: %0.3gN\n', avg_peak_thrust)
fprintf('Std. Dev. of Peak Thrust is: %0.3gN\n\n', std_dev_peak_thrust)



%% Format Plot of F vs. t
xMax = 0.3; % [s]

%%% plot avg. peak thrust
t_vec = linspace(0, xMax, 100);
peak_thrust_plot_vec = avg_peak_thrust .* ones(100, 1);

p2 = plot(t_vec, peak_thrust_plot_vec, ':k', 'linewidth', 2);

% legend
legend([p1, p2], {'Thrust Data from Trials', ...
        sprintf('Average Peak Thrust = %0.3gN', avg_peak_thrust)}, ...
       'interpreter', 'latex', 'location', 'best')
% give title
title('Representative Thrust Data from Static Test Stand Trials')
% give x and y labels
xlabel('$t$ (s)')
ylabel('$T$ (N)')
ylim([-30 inf])
xlim([0, xMax])
set(gca, 'fontsize', FONTSIZE)
set(gca, 'defaulttextinterpreter', 'latex')
set(gca, 'TickLabelInterpreter', 'latex')
grid on

%%% setup and save figure as .pdf
if shouldSaveFigures
    curr_fig = gcf;
    set(curr_fig, 'PaperOrientation', 'landscape');
    set(curr_fig, 'PaperUnits', 'normalized');
    set(curr_fig, 'PaperPosition', [0 0 1 1]);
    [fid, errmsg] = fopen(saveTitle1, 'w+');
    if fid < 1 % check if file is already open.
       error('Error Opening File in fopen: \n%s', errmsg);
    end
    fclose(fid);
    print(gcf, '-dpdf', saveTitle1);
end


%% Total Time to Thrust
%%% Plotting
hFig = figure('name', figName2);
```

```matlab
scrz = get(groot, 'ScreenSize');
set(hFig, 'Position', scrz)

histfit(total_time_thrust_vec, N_BINS)

% give title
title('Total Time to Thrust')
% give x and y labels
xlabel('Time to Thrust, $t$, (s)')
ylabel('\# Trials with Time to Thrust')
set(gca, 'fontsize', FONTSIZE)
set(gca, 'defaulttextinterpreter', 'latex')
set(gca, 'TickLabelInterpreter', 'latex')
grid on

%%% setup and save figure as .pdf
if shouldSaveFigures
    curr_fig = gcf;
    set(curr_fig, 'PaperOrientation', 'landscape');
    set(curr_fig, 'PaperUnits', 'normalized');
    set(curr_fig, 'PaperPosition', [0 0 1 1]);
    [fid, errmsg] = fopen(saveTitle2, 'w+');
    if fid < 1 % check if file is already open.
       error('Error Opening File in fopen: \n%s', errmsg);
    end
    fclose(fid);
    print(gcf, '-dpdf', saveTitle2);
end


%%% printing
avg_time_thrust = mean(total_time_thrust_vec);
std_dev_time_thrust = std(total_time_thrust_vec);

% print to command window
fprintf('Average Time to Thrust is: %0.3gs\n', avg_time_thrust)
fprintf('Std. Dev. of Peak Thrust is: %0.3gs\n\n', std_dev_time_thrust)


%% Plot of SEM vs. N
maxN = 1e4;
N = 1:1:maxN;
SEM_ISP = std_dev_ISP ./ sqrt(N);

SEM_Data_set_ISP = SEM_ISP(length(ISP_vec));

fprintf('The SEM for the ISP data set is: %0.3gs\n\n', SEM_Data_set_ISP)


%%% Plotting
hFig = figure('name', figName3);
scrz = get(groot, 'ScreenSize');
set(hFig, 'Position', scrz)

plot(N, SEM_ISP, 'color', colorVecs{1}, ...
```

```
      'linewidth', 2.5)
hold on
plot(N, SEM_ISP .* 1.96, ':', 'color', colorVecs{2}, ...
      'linewidth', 2)
plot(N, SEM_ISP .* 2.24, ':', 'color', colorVecs{3}, ...
      'linewidth', 2)
plot(N, SEM_ISP .* 2.58, ':', 'color', colorVecs{4}, ...
      'linewidth', 2)

% assign legend
legend({'$SEM$ vs. $N$', ...
         'Uncertainty in ISP estimate with 95\% C.I.', ...
         'Uncertainty in ISP estimate with 97.5\% C.I.', ...
         'Uncertainty in ISP estimate with 99\% C.I.'}, ...
        'interpreter', 'latex', 'location', 'best')
% give title
title('$SEM$ vs. $N$')
% give x and y labels
xlabel('$N$ (\# tests conducted)')
ylabel('$SEM$ (s)')
xlim([1 100])
ylim([-inf inf])
set(gca, 'fontsize', FONTSIZE)
set(gca, 'defaulttextinterpreter', 'latex')
set(gca, 'TickLabelInterpreter', 'latex')
grid on

%%% setup and save figure as .pdf
if shouldSaveFigures
    curr_fig = gcf;
    set(curr_fig, 'PaperOrientation', 'landscape');
    set(curr_fig, 'PaperUnits', 'normalized');
    set(curr_fig, 'PaperPosition', [0 0 1 1]);
    [fid, errmsg] = fopen(saveTitle3, 'w+');
    if fid < 1 % check if file is already open.
       error('Error Opening File in fopen: \n%s', errmsg);
    end
    fclose(fid);
    print(gcf, '-dpdf', saveTitle3);
end


%% N for 95% confidence level for mean ISP = 0.1s & mean ISP 0.01s

% set C.I. z-value
z = 1.96;

%%% 95% CI for ISP estimate within 0.1s
confidence_idx = find(z .* SEM_ISP < 0.1, 1, 'first');
N_needed = N(confidence_idx);
fprintf(['N needed to ISP estimate', ...
          ' to within 0.1s with 95%% confidence: N = %d\n'], N_needed);

%%% 95% CI for ISP estimate within 0.01s
confidence_idx = find(z .* SEM_ISP < 0.01, 1, 'first');
```

```
N_needed = N(confidence_idx);
fprintf(['N needed to ISP estimate', ...
         ' to within 0.01s with 95%% confidence: N = %d\n\n'], N_needed);


%% N for 97.5% confidence level for mean ISP = 0.1s & mean ISP 0.01s

% set C.I. z-value
z = 2.24;

%% 97.5% CI for ISP estimate within 0.1s
confidence_idx = find(z .* SEM_ISP < 0.1, 1, 'first');
N_needed = N(confidence_idx);
fprintf(['N needed to ISP estimate', ...
         ' to within 0.1s with 97.5%% confidence: N = %d\n'], N_needed);

%% 97.5% CI for ISP estimate within 0.01s
confidence_idx = find(z .* SEM_ISP < 0.01, 1, 'first');
N_needed = N(confidence_idx);
fprintf(['N needed to ISP estimate', ...
         ' to within 0.01s with 97.5%% confidence: N = %d\n\n'], N_needed);


%% N for 99% confidence level for mean ISP = 0.1s & mean ISP 0.01s

% set C.I. z-value
z = 2.58;

%% 99% CI for ISP estimate within 0.1s
confidence_idx = find(z .* SEM_ISP < 0.1, 1, 'first');
N_needed = N(confidence_idx);
fprintf(['N needed to ISP estimate', ...
         ' to within 0.1s with 99%% confidence: N = %d\n'], N_needed);

%% 99% CI for ISP estimate within 0.01s
confidence_idx = find(z .* SEM_ISP < 0.01, 1, 'first');
N_needed = N(confidence_idx);
fprintf(['N needed to ISP estimate', ...
         ' to within 0.01s with 99%% confidence: N = %d\n\n'], N_needed);
```

VII.B. loadInData.m – Builds Data Directory and Loads Data

```
function data = loadInData
    %% Author: Nicholas Renninger
    %%
    %% Purpose: Creates directory of data, creates array of structs from
    %% each data file
    %%
    %% Inputs:
    %% none
    %%
    %% Outputs:
    %% data = cell array of structs, each containing the following data
    %% for each valid static test performed:
    %%
    %% water_mass: the mass of water used [kg]
    %%
    %% pressure: the pressure inside of the bottle [Pa]
    %%
    %% temp: the temperature [K]
    %%
    %% group_num: the group number
    %%
    %% time: the time vector for the data [s]
    %%
    %% thrust: the thrust vector for the data [N]
    %%
    %% ISP: the specific impulse [s]
    %%
    %% Date Created: 17 April 2017
    %%
    %% Last Editted: 17 April 2017


    relPathData = '../Data/';
    absPathData = cat(2, relPathData, '*.csv');

    test_dir = dir(absPathData);

    for i = 1:length(test_dir) - 2

        filename = cat(2, relPathData, test_dir(i + 2, 1).name);

        [ data{i}.water_mass, data{i}.pressure, data{i}.temp, ...
          data{i}.group_num,  data{i}.time, data{i}.thrust, data{i}.ISP ] = ...
        somethingNew( filename );


    end

end
```

VII.C. somethingNew.m – Processes Each Data File for Analysis

```
function [ water_mass, pressure, temp, group_num, time, thrust, ISP ] = ...
    somethingNew( filename )

    %% Author: Marshall Herr & Nicholas Renninger
    %%
    %% Purpose: To load a STS datafile, extract the relavent information,
    %% correct the thrust for the sensors, and calculate ISP
    %%
    %% Inputs:
    %% filename: the file location name for the datafile
    %%
    %% Outputs:
    %% water_mass: the mass of water used [kg]
    %%
    %% pressure: the pressure inside of the bottle [Pa]
    %%
    %% temp: the temperature [K]
    %%
    %% group_num: the group number
    %%
    %% time: the time vector for the data [s]
    %%
    %% thrust: the thrust vector for the data [N]
    %%
    %% ISP: the specific impulse [s]
    %%
    %% Date Created: 12 April 2017
    %%
    %% Last Editted: 19 April 2017

    %% Error Checking

    % generating file ID
    fID = fopen( filename );

    % extracting as string
    string = textscan( fID, '%c' );

    % while the file is being stored in a 1x1 cell array
    while isa( string, 'cell' )

        string = string{1};

    end

    % if it is a column vector instead of a row vector
    if ( size( string, 2 ) == 1 ) && ( length( string ) ~= 1 )

        % transpose it
        string = string';

    end

    % to search for numbers
```

```
    number_string = double( string );

    % extracting the char_to_double for numbers
    num_chars = '0123456789.';

    % initialization
    num_local = zeros( 1, length( string ) );

    for i = 1: length( num_chars )

        % determining the double equivalent
        num = double( num_chars(i) );

        % finding any location with the number
        % by adding them together it becomes effectively a big or statement
        num_local = num_local + ( number_string == num );

    end

    % for simplicity
    string = lower( string );

    % find where the data rate is located
    rate_local = strfind( string, 'frequency' );

    % if not reported for some reason
    if ~any( rate_local )

        error( 'Data rate not reported.' )

    end

    % finding the first number after 'water'
    num_start = find( num_local( rate_local : end ), 1, 'first' ) + ...
        rate_local - 1;

    % finding the first non-number after the first number
    num_end = find( ~num_local( num_start : end ), 1, 'first' ) + ...
        num_start - 2;

    % extracting the water mass ( should be kHz when extracted )
    rate = str2double( string( num_start : num_end ) ) * 1000; % s^-1

    % find where the water mass is located
    water_local = strfind( string, 'water' );

    % if not reported for some reason
    if ~any( water_local )

        error( 'Water mass not reported.' )

    end

    % finding the first number after 'water'
    num_start = find( num_local( water_local : end ), 1, 'first' ) + ...
```

```
        water_local - 1;

    % finding the first non-number after the first number
    num_end = find( ~num_local( num_start : end ), 1, 'first' ) + ...
        num_start - 2;

    % extracting the water mass ( should be grams when extracted )
    water_mass = str2double( string( num_start : num_end ) ) / 1000; % kg

    % test if the data is of the TA baseline or not
    if water_mass > 1.05
        error('Test Data in %s used too much water: %0.3gkg', filename, ...
                                                            water_mass)
    elseif water_mass < 0.98
        error('Test Data in %s used too little water: %0.3gkg', filename, ...
                                                              water_mass)
    end

    % find where the pressure is located
    pressure_local = strfind( string, 'pressure' );

    % if not reported for some reason
    if ~any( pressure_local )

        error( 'Pressure not reported.' )

    end

    % finding the first number after 'pressure'
    num_start = find( num_local( pressure_local : end ), 1, 'first' ) + ...
        pressure_local - 1;

    % finding the first non-number after the first number
    num_end = find( ~num_local( num_start : end ), 1, 'first' ) + ...
        num_start - 2;

    % extracting the pressure ( should be psi when extracted )
    pressure = str2double( string( num_start : num_end ) ) * 6894.76; % Pa

    % find where the temperature is located
    temp_local = strfind( string, 'temperature' );

    % if not reported for some reason
    if ~any( temp_local )

        error( 'Temperature not reported.' )

    end

    % finding the first number after 'temperature'
    num_start = find( num_local( temp_local : end ), 1, 'first' ) + ...
        temp_local - 1;

    % finding the first non-number after the first number
    num_end = find( ~num_local( num_start : end ), 1, 'first' ) + ...
```

```matlab
        num_start - 2;

    % extracting the temperature ( should be C when extracted )
    temp = str2double( string( num_start : num_end ) ) + 273.15; % K

    % find where the group number is located
    group_num_local = strfind( string, 'group' );

    % if not reported for some reason
    if ~any( group_num_local )

        error( 'Group Number not reported.' )

    end

    % extracting the group number
    group_num = str2double( filename(end-22:end-21) );

    %% Pull Out Data

    % importing the thrust data
    data = load( filename );

    % is in lbf when extracted
    thrust = data( :, 3 )' * 4.44822; % N

    % finding max thrust and location
    [ ~, max_local ] = max( thrust );

    % finding when the thrusting began (last negative is right before
    % thrust)
    thrust_start = find( thrust( 1 : max_local ) < 0, 1, 'last' );

    % cutting out pre-thrust data
    thrust = thrust( thrust_start : end ); % N

    thrust_start = find( thrust > 5, 1, 'first' );

    thrust = thrust( thrust_start : end ); % N

    % inverting data and smoothing
    search_thrust = -thrust;

    % really, really smooth it out

    search_thrust = smooth( search_thrust );

    [ ~, thrust_end ] = max( search_thrust );

    % cutting out post thrust data
    thrust = thrust( 1 : thrust_end ); % N

    time = linspace( 0, length( thrust ), length( thrust ) ) ./ rate; % s

    % creating points for sensor adjustment correction
```

```
    point1 = [ time(1), 0 ];

    point2 = [ time(end), thrust(end) ];

    slope = ( point2(2) - point1(2) ) / ( point2(1) - point1(1) ); % N s^-1

    % the amount the sensor has adjusted
    adjustment = slope .* time; % N


%     figure()
%
%     plot( time, thrust, 'b' )
%
%     hold on
%
%     plot( time, adjustment, 'r' )


    % adjusting thrust to correct for sensors
    thrust = thrust - adjustment; % N

    water_weight = water_mass * 9.81; % N

    % total I [N s] divided by water weight [N] to get ISP [s]
    ISP = trapz( time, thrust ) / water_weight; % s

    % plot( time, thrust, 'm' )

    fclose(fID);

end
```