Error Propagation: given $R(x_1, x_2 \ldots x_n)$, and where W_R is the error in R,

$$W_R = \sqrt{\sum_{i=1}^{n} \left(\frac{\partial R}{\partial x_i} W_{x_i}\right)^2}$$ This is used in the MATLAB code.

Wing Area ($S=cb$): $S_{err} = \sqrt{(c \cdot b_{err})^2 + (b \cdot c_{err})^2}$

Dynamic Pressure ($q = \frac{1}{2}\rho v^2$): $q_{err} = \sqrt{\left(\frac{1}{2}v^2 \rho_{err}\right)^2 + (\rho v \cdot v_{err})^2}$

Lift ($L = -F_a \sin\alpha + F_n \cos\alpha$): $L_{err} = \sqrt{(-F_{a_{err}} \sin\alpha)^2 + (F_{n_{err}} \cos\alpha)^2 + [(-F_a \cos\alpha - F_n \sin\alpha)\alpha_{err}]^2}$

Drag ($D = F_a \cos\alpha + F_n \sin\alpha$): $D_{err} = \sqrt{(+F_{a_{err}} \cos\alpha)^2 + (F_{n_{err}} \sin\alpha)^2 + [(F_a \sin\alpha + F_n \cos\alpha)\alpha_{err}]^2}$

C_L ($C_L = \frac{L}{qS}$): $C_{L_{err}} = \sqrt{\left(\frac{L_{err}}{qS}\right)^2 + \left(-\frac{L}{q^2 S} q_{err}\right)^2 + \left(-\frac{L}{qS^2} S_{err}\right)^2}$

C_D ($C_D = \frac{D}{qS}$): $C_{D_{err}} = \sqrt{\left(\frac{D_{err}}{qS}\right)^2 + \left(-\frac{D}{q^2 S} q_{err}\right)^2 + \left(-\frac{D}{qS^2} S_{err}\right)^2}$

Re ($Re = \frac{\rho v_\infty c}{\mu}$): $Re_{err} = \sqrt{\left(\frac{v_\infty c}{\mu}\rho_{err}\right)^2 + \left(\frac{\rho c}{\mu} v_{\infty_{err}}\right)^2 + \left(\frac{\rho v_\infty}{\mu} c_{err}\right)^2 + \left(-\frac{\rho v_\infty c}{\mu^2} \mu_{err}\right)^2}$

L_o ($L_o = L - x\sin\alpha$): $L_{o_{err}} = \sqrt{(-x\sin\alpha \cdot L_{err})^2 + [(L - \sin\alpha)x_{err}]^2 + [(L - x\cos\alpha)\alpha_{err}]^2}$

$\frac{L_o - Y}{Y}$: $len\_scale_{err} = \sqrt{\left(\frac{Y L_{o_{err}}}{L_o^2}\right)^2 + \left(\frac{-Y_{err}}{L_o}\right)^2}$

In Spreadsheet King's Law Calibration, RMSE was calculated as such; $RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (\text{Predicted}_i - \text{Actual}_i)^2}$

King's Law fit

Lab data

```python
"""Script to store sting data in a more convenient .mat format."""

from scipy.io import savemat

ANGLES = ["-5", "00", "05", "10", "12", "14", "15", "18", "20"]

PREFIXES = {
    "force": "force_mes"
}

FOLDER = "Force Measurements"
src_fns = [f"./{FOLDER}/{PREFIXES['force']}_a_{angle}.dat" for angle in ANGLES]
dest_fns = [f"./{FOLDER}/{PREFIXES['force']}_a_{angle}.mat" for angle in ANGLES]

KEYS = {
    "Static Pressure": "P",
    "Density": "rho",
    "Fixed Pitot Probe Speed": "v",
    "Normal Force": "Fn",
    "Axial Force": "Fa",
    "Angle of Attack": "a",
}

for i, fn in enumerate(src_fns):
    with open(fn, "r") as f:
        lines = f.readlines()
        data = {}
        for j, line in enumerate(lines):
            key = line.split("=")[0].strip()
            if key in KEYS.keys():
                try:
                    val = line.split("=")[1].strip()
                    val = val.split("\t")[0] # Remove tabs if present
                except Exception:
                    # print(f"Error: Couldn't parse val of line {j}, setting as 'None'")
                    val = None
                data[KEYS[key]] = float(val) # Create dict with symbols
    print(data)
    savemat(dest_fns[i], data)
```

Contents

CL, CD vs alpha

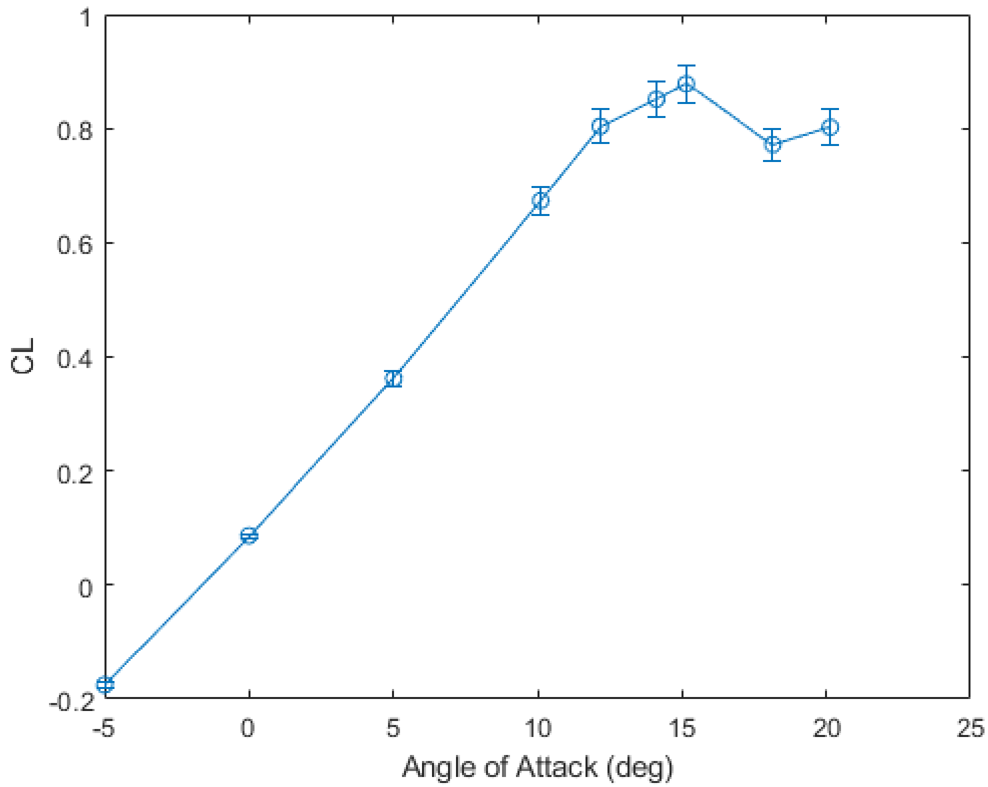
Trevor Burgoyne 16 Oct 2022

```
% Paths for data loading
ROOT_DIR = "C:/Users/Trevor/Desktop/AEM 4602W/Fluids Lab/Fluids Lab Data/";
FORCE_DIR = ROOT_DIR + "Force Measurements/";
ANGLES = ["-5", "00", "05", "10", "12", "14", "15", "18", "20"];

% Useful Conversions
LB_TO_N = 4.448; % lb -> N = (lb) * 4.448 N/lb
N_TO_LB = 1/LB_TO_N;
DEG_TO_RAD = pi/180; % degrees -> rad = (deg)* pi/180 rad/deg

% Base Uncertainties
F_ERR   =   0.1; % ± N, given error in sting measurements
A_ERR   =   0.2; % ± degrees, given error in sting measurements
C_ERR   = 0.001; % ± m, bias error from using a meter stick
B_ERR   = 0.001; % ± m, bias error from using a meter stick
V_ERR   =   0.4; % ± m/s, given error in pitot tube measurements
Y_ERR   =  1/16; % ± in, bias error from reading hot wire tape measure
RHO_ERR =  0.02; % *100 ± % of value, given error in pitot tube measurements
MU_ERR  =  0.01; % *100 ± % of value, given error in pitot tube measurements

F_ERR_LB  = F_ERR * N_TO_LB; % lb
A_ERR_RAD = A_ERR * DEG_TO_RAD; % rad

% Arrays to store CD, CL, and a
CL_arr     = zeros(1, length(ANGLES));
CD_arr     = zeros(1, length(ANGLES));
a_arr      = zeros(1, length(ANGLES));
CL_ERR_arr = zeros(1, length(ANGLES));
CD_ERR_arr = zeros(1, length(ANGLES));

% Airfoil properties
c = .254; % m +/- .005m, chord length
b = .670; % m +/- .005m, wing span
S =  b*c; % m^2, approx. wing area
S_ERR = sqrt( (c*B_ERR)^2 + (b*C_ERR)^2 ); % ± m^2

for i = 1:length(ANGLES)
    path = FORCE_DIR + "force_mes_a_" + ANGLES(i) + ".mat";
    data = load(path); % lab data, with P, rho, v, Fn, Fa, a

    % q  = .5*rho*v^2, dynamic pressure
    q = .5 * data.rho * data.v^2;
    Q_ERR = sqrt( (.5 * RHO_ERR * data.v^2)^2 + (data.rho*data.v*V_ERR)^2 );

    % L  = -Fa*sin(a) + Fn*cos(a), Lift Force
```

```
    L = -data.Fa*sind(data.a) + data.Fn*cosd(data.a);
    L = L * LB_TO_N;
    L_ERR = sqrt( (-F_ERR_LB*sind(data.a))^2 + (F_ERR_LB*cosd(data.a))^2 +...
        ( (-data.Fa*cosd(data.a) - data.Fn*sind(data.a))*A_ERR_RAD )^2 )...
        * LB_TO_N; % ± N

    % D = Fa*cos(a) + Fn*sin(a), Drag Force
    D = data.Fa*cosd(data.a) + data.Fn*sind(data.a);
    D = D * LB_TO_N;
    D_ERR = sqrt( (-F_ERR_LB*cosd(data.a))^2 + (F_ERR_LB*sind(data.a))^2 +...
        ( (data.Fa*cosd(data.a) + data.Fn*sind(data.a))*A_ERR_RAD )^2 )...
        * LB_TO_N; % ± N

    % CL = L / q*S, coefficient of lift
    CL = L / (q*S);
    CL_ERR = sqrt( (L_ERR/(q*S))^2 + ( (-Q_ERR*L)/(S*q^2) )^2 + ( (-S_ERR*L)/(q*S^2) )^2 ); % unitless

    % CD = D / (q*S), coefficient of drag
    CD = D / (q*S);
    CD_ERR = sqrt( (D_ERR/(q*S))^2 + ( (-Q_ERR*D)/(S*q^2) )^2 + ( (-S_ERR*D)/(q*S^2) )^2 ); % unitless

    % Store in arrays for graphing
    CL_arr(i) = CL;
    CD_arr(i) = CD;
    a_arr(i)  = data.a;
    CL_ERR_arr(i) = CL_ERR;
    CD_ERR_arr(i) = CD_ERR;
end
```

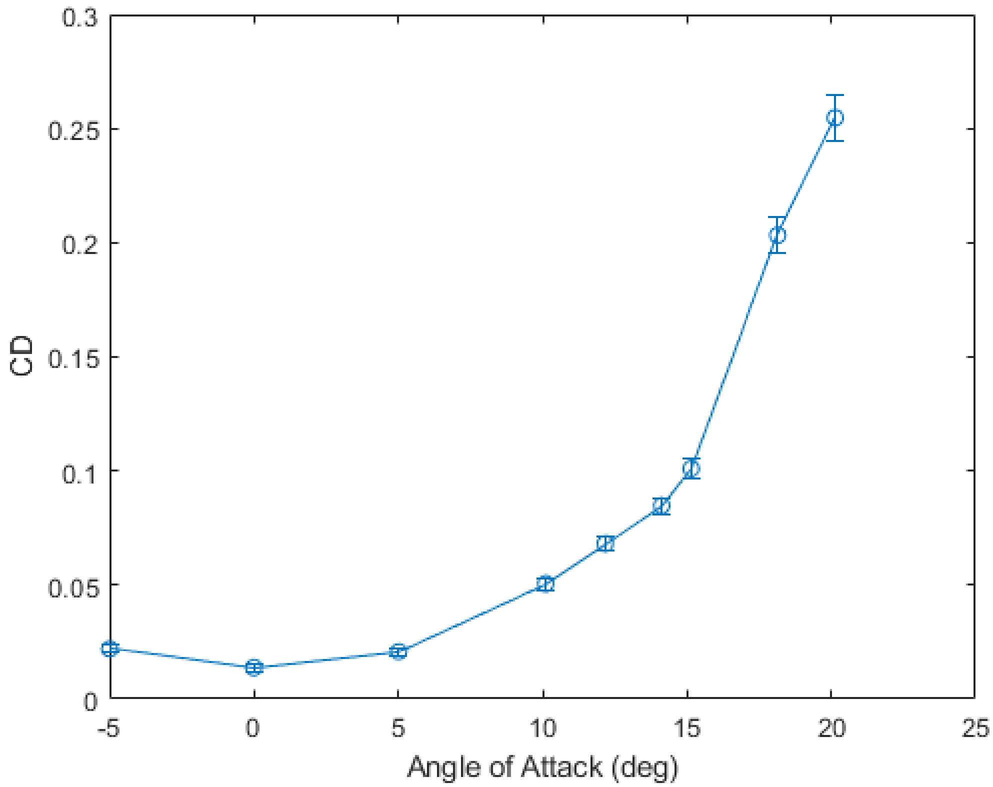
CL vs Angle of Attack

```
errorbar(a_arr, CL_arr, CL_ERR_arr, "-o")
xlabel("Angle of Attack (deg)")
ylabel("CL")
```



CD vs Angle of Attack

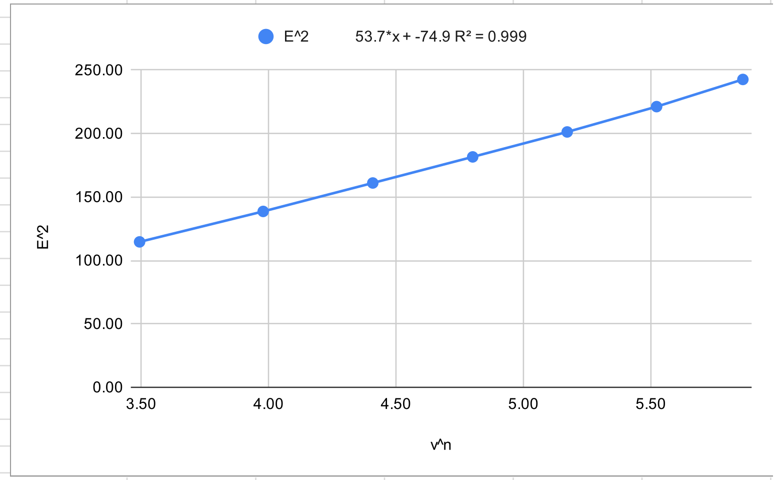
```
errorbar(a_arr, CD_arr, CD_ERR_arr, "-o")
xlabel("Angle of Attack (deg)")
ylabel("CD")
```



Published with MATLAB® R2020b

Power (% of 60 Hz):	25	30	35	40	45	50	55
pitot v	9.72312	12.3154	14.8422	17.3286	19.8391	22.354	24.9073
predicted v	9.90	12.30	14.75	17.17	19.64	22.30	25.32
diff^2	0.03	0.00	0.01	0.03	0.04	0.00	0.17
v^n	3.49	3.98	4.41	4.80	5.17	5.52	5.86
E^2	114.62	138.62	161.04	181.60	201.28	221.24	242.67
	-1.75964	-2.84302	-3.77075	-4.55261	-5.25543	-5.94055	-6.64825
	-1.76025	-2.8421	-3.75183	-4.55444	-5.26489	-5.94879	-6.64276
	-1.76056	-2.83325	-3.77167	-4.56482	-5.27313	-5.92865	-6.63879
	-1.77399	-2.84729	-3.76404	-4.54376	-5.2594	-5.94299	-6.64001
	-1.75537	-2.85492	-3.76068	-4.55414	-5.26794	-5.93475	-6.64917
	-1.75171	-2.84546	-3.7619	-4.57214	-5.2771	-5.93719	-6.63757
	-1.78833	-2.83997	-3.75336	-4.55963	-5.28046	-5.93414	-6.65802
	-1.75476	-2.85492	-3.76617	-4.5578	-5.28046	-5.95184	-6.66199
	-1.76605	-2.85187	-3.76221	-4.55017	-5.26093	-5.9613	-6.64337
	-1.76178	-2.84058	-3.76312	-4.5639	-5.2713	-5.93811	-6.63727
	-1.76056	-2.84698	-3.77899	-4.55017	-5.27985	-5.92957	-6.65009
	-1.78986	-2.84332	-3.78571	-4.55078	-5.26672	-5.9552	-6.64917
	-1.76117	-2.85492	-3.76251	-4.57092	-5.28595	-5.93597	-6.64276
	-1.76727	-2.85156	-3.7619	-4.53674	-5.28717	-5.92682	-6.64917
	-1.76514	-2.85126	-3.76251	-4.54407	-5.2774	-5.93781	-6.66595
	-1.75873	-2.8479	-3.76648	-4.54529	-5.26398	-5.93536	-6.64215
	-1.77246	-2.84698	-3.75824	-4.55841	-5.27924	-5.93842	-6.64856
	-1.77063	-2.84119	-3.77228	-4.55261	-5.27069	-5.94421	-6.63971
	-1.77673	-2.85095	-3.7851	-4.56299	-5.26215	-5.93872	-6.65344
	-1.76208	-2.85645	-3.76587	-4.56757	-5.27649	-5.94025	-6.63788
	-1.76758	-2.84698	-3.76495	-4.56207	-5.29236	-5.93842	-6.64581
	-1.78223	-2.84302	-3.78937	-4.54956	-5.27649	-5.95032	-6.65955
	-1.76727	-2.86835	-3.78174	-4.56024	-5.27649	-5.94055	-6.65344
	-1.77094	-2.84424	-3.74908	-4.56055	-5.27893	-5.95032	-6.65253
	-1.76056	-2.85583	-3.7674	-4.55627	-5.2713	-5.93872	-6.64246
	-1.7749	-2.84882	-3.77808	-4.55414	-5.25452	-5.9494	-6.66748
	-1.76422	-2.86011	-3.7738	-4.56696	-5.2655	-5.95001	-6.65039
	-1.76025	-2.84332	-3.77441	-4.56482	-5.26062	-5.94177	-6.633

OFFSET:	-8.92	N:	0.55
B:	53.7	A:	-74.9
RMSE	0.20		



```python
"""Script to store hotwire data in a more convenient .mat format."""

from scipy.io import savemat

ANGLES = ["-5", "00", "05", "20"]

FOLDER = "Hotwire Measurements"

FN_PREFIXES = {
    ANGLES[0]: "hotwire_mes_a_neg_5",
    ANGLES[1]: "hotwire_mes",
    ANGLES[2]: "hotwire_mes_a_5",
    ANGLES[3]: "hotwire_mes_a_20"
}

N_DATAPOINTS = {
    ANGLES[0]: 11,
    ANGLES[1]: 10,
    ANGLES[2]: 15,
    ANGLES[3]: 78,
}

USER_PREFIXES = {
    ANGLES[0]: "a_neg_5_",
    ANGLES[1]: "a_0_",
    ANGLES[2]: "a_5_",
    ANGLES[3]: "a_20_"
}

SRC_FNS = {
    angle: {
        "dat": [f"./{FOLDER}/a_{angle}/{FN_PREFIXES[angle]}-{i+1}.dat" for i in
range(N_DATAPOINTS[angle])],
        "txt": [f"./{FOLDER}/a_{angle}/{FN_PREFIXES[angle]}-{i+1}-1.txt" for i in
range(N_DATAPOINTS[angle])],
    }
    for angle in ANGLES
}

KEYS = {
    "Static Pressure": "P",
    "Density": "rho",
    "Fixed Pitot Probe Speed": "v",
    "AOA": "a",
    "User comment": "y",
    "Temperature": "T",
}

def main():
    """Parse .dat and .txt files for each angle and combine relevant info into a .mat file."""
    for angle in ANGLES:
        dat_fns = SRC_FNS[angle]["dat"]
        txt_fns = SRC_FNS[angle]["txt"]

        for i, fn in enumerate(dat_fns):
            with open(fn, "r") as f:
                lines = f.readlines()
                data = {}
                for line in lines:

                    # Parse data key
                    split_char = "=" # Most values use '='
                    if len(line.split(split_char)) == 1:
                        split_char = ":" # user comment uses ':'
                    key = line.split(split_char)[0].strip()

                    if key in KEYS.keys():
```

```
                        try:
                            val = line.split(split_char)[1].replace("degrees (inclinometer)",
"").strip() # Remove junk
                            val = val.split("\t")[0] # Remove tabs if present
                        except Exception:
                            val = None
                        data[KEYS[key]] = comment_to_distance(val, angle) if key == "User
comment" else float(val) # Create dict with symbols

            # Parse txt file with hotwire voltages
            with open(txt_fns[i], "r") as f:
                lines = f.readlines()
                voltages = []
                for line in lines:
                    try:
                        voltages.append(float(line.split("\t")[1])) # Get voltages
                    except Exception:
                        continue
            data["V_arr"] = voltages

            dest_fn = f"./{FOLDER}/a_{angle}/data_{i+1}.mat"
            savemat(dest_fn, data)


def comment_to_distance(comment: str, angle: str):
    """Turn the user comment into a numerical distance."""
    val = comment.replace(USER_PREFIXES[angle], "").replace("in","") # Get just the numbers
    val = val.split("_") # Split fraction into parts

    ret = float(val[0])
    if len(val) == 3: # Do fraction math if needed
        ret += float(val[1]) / float(val[2])

    return ret

if __name__ == "__main__":
    main()
```

Contents

Turbulence Profiles

Trevor Burgoyne 16 Oct 2022

```
% Paths for data loading
ROOT_DIR = "C:/Users/Trevor/Desktop/AEM 4602W/Fluids Lab/Fluids Lab Data/";
HOTWIRE_DIR = ROOT_DIR + "Hotwire Measurements/";
ANGLES = ["-5", "00", "05", "20"];
N_DATAPOINTS = [11, 10, 15, 78];
V_AVG   = 24.4247; % m/s, mean of all velocities, excluding nan
T_AVG   = 27.8641; % degrees C, mean of all temperatures
P_AVG   =   98769; % Pa, mean of all pressures, excluding nan
RHO_AVG =  1.1430; % kg/m^3, mean of all densities, excluding nan
MU_AVG  = 1.85e-5; % Pa*s, dynamic viscosity at T_AVG and ~atm pressure. Src: https://www.engineeringtoolbox.com/air-absolute-kinematic-viscosity-d_601.html

% Useful Conversions
LB_TO_N = 4.448; % lb -> N = (lb) * 4.448 N/lb
N_TO_LB = 1/LB_TO_N;
DEG_TO_RAD = pi/180; % degrees -> rad = (deg)* pi/180 rad/deg
IN_TO_M = 0.0254; % in -> m = (in) * .0254 m/in

% Base Uncertainties
F_ERR   =   0.1; % ± N, given error in sting measurements
A_ERR   =   0.2; % ± degrees, given error in sting measurements
C_ERR   = 0.001; % ± m, bias error from using a meter stick
B_ERR   = 0.001; % ± m, bias error from using a meter stick
V_ERR   =   0.4; % ± m/s, given error in pitot tube measurements
Y_ERR   =  1/16; % ± in, bias error from reading hot wire tape measure
X_ERR   =  1/16; % ± in, bias error from reading hot wire tape measure
L_ERR   =  1/16; % ± in, bias error from reading hot wire tape measure

RHO_ERR =  0.02*RHO_AVG; % *100 ± % of value, given error in pitot tube measurements
MU_ERR  =  0.01*MU_AVG; % *100 ± % of value, given error in pitot tube measurements
V_RMS_ERR = 0.2; % ± m/s, from calibration spreadsheet

F_ERR_LB  = F_ERR * N_TO_LB; % lb
A_ERR_RAD = A_ERR * DEG_TO_RAD; % rad
Y_ERR_M   = Y_ERR * IN_TO_M; % m
X_ERR_M   = X_ERR * IN_TO_M; % m
L_ERR_M   = L_ERR * IN_TO_M; % m

% Arrays to store data per angle
angle_data_arr = repmat(...
    struct(...
        "len_scale",[],...
        "v_normalized", [],...
        "v_rms", [],...
        "len_scale_ERR",[],...
        "v_normalized_ERR",[]...
    ), length(ANGLES), 1 ...
);

% Experiment properties
c =  .254; % m, airfoil chord length
x = .75*c; % m, distance from trailing edge to hot wire
L = 19.5*IN_TO_M; % m, distance from hot wire to top of tunnel

% Calibration constants: (E+offset)^2 = A + B*U^n
offset = -8.92; % V, voltage at zero flow
A      = -74.9; % constant from linear regression
B      =  53.7; % constant from linear regression
n      =  0.55; % exponent in King's Law that gave straightest fit

% Average Experiment Reynolds Number
Re_AVG = (RHO_AVG*V_AVG*c)/MU_AVG
Re_ERR = sqrt(...
    ( (RHO_ERR*V_AVG*c)/MU_AVG )^2 + ( (RHO_AVG*V_ERR*c)/MU_AVG )^2 + ...
    + ( (RHO_AVG*V_AVG*C_ERR)/MU_AVG )^2 + ( (-RHO_AVG*V_AVG*c*MU_ERR)/(MU_AVG^2) )^2 ...
)


for i = 1:length(ANGLES)
   angle = ANGLES(i);
   angle_data_arr(i) = struct(...
```

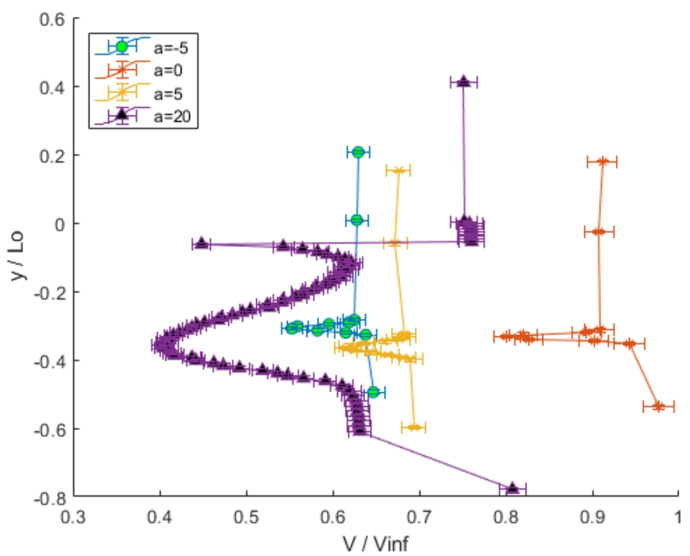
```
      "len_scale", zeros(1, N_DATAPOINTS(i)),...
      "v_normalized", zeros(1, N_DATAPOINTS(i)),...
      "v_rms", zeros(1,N_DATAPOINTS(i)),...
      "len_scale_ERR", zeros(1,N_DATAPOINTS(i)),...
      "v_normalized_ERR", zeros(1,N_DATAPOINTS(i))...
   );

   % LENGTH SCALE: the hot wire was positioned at x = .75c behind the
   % trailing edge of the airfoil, with the airfoil being L = 19.5in from
   % the top of the tunnel, as measured at zero angle of attack.
   % However, since moving the sting caused a change in the vertical
   % position of the TE, L0 was selected to be the distance from the top of
   % the tunnel to the airfoil, adjusted for angle of attack. Using trig,
   % this works out to be L - x*sin(a).
   a = str2double(angle); % deg
   L0 = L - x*sind(a); % m, height of TE adjusted for angle of attack
   L0_ERR = sqrt( (-x*sind(a)*L_ERR_M)^2 + ( (L-sind(a))*X_ERR_M )^2 + ( (L-x*cosd(a))*A_ERR_RAD )^2 ); % ± m

   for j = 1:N_DATAPOINTS(i)
      path = HOTWIRE_DIR + "a_" + ANGLES(i) + "/data_" + j + ".mat";
      data = load(path); % lab data, with P, T, rho, v, a, y, and V_arr

      % NOTE: for some reason, the pitot tube returned a speed of 'nan' for
      % all of our measurements at a = 0. This isn't a huge deal, since the
      % freestream was always set to the same speed, so a good
      % approximation for this case is to use the average of all other
      % velocities we measured
      if(isnan(data.v))
          data.v = V_AVG;
      end


      % NOTE: y as measured in the lab is the distance from the top of the
      % tunnel to the hot wire. L0, as discussed earlier, is the distance
      % from the top of the tunnel to the TE, adjusted for angle of attack.
      % To make values of y *above* the TE to be positive and *below* to be
      % negative, y was subtracted from L0 to transfrom y into the distance
      % of the hotwire above the TE. This was then nondimensionalized by L0

      % len_scale = (L0 - y) / L0
      len_scale = (L0 - (data.y * IN_TO_M)) / L0;
      len_scale_ERR = sqrt( ( (data.y*IN_TO_M*L0_ERR)/(L0^2) )^2 + (-Y_ERR_M/L0)^2 ); % unitless

      % hot wire velocity
      % from calibration: v_hotwire = (((E + offset)^2 - A)/B)^(1/n)
      v_hotwire_arr = (((data.V_arr + offset).^2 - A)./B).^(1/n);

      % v_rms = remove mean from velocities, take the average of their
      % squares, and then take the square root
      v_rms = sqrt( mean( ( v_hotwire_arr - mean(v_hotwire_arr) ).^2 ) );
      v_rms = v_rms / data.v; % normalized to be non-dimensional

      % v_hotwire / v_freestream
      v_normalized = mean(v_hotwire_arr) / data.v;
      v_normalized_ERR = sqrt( (V_RMS_ERR/data.v)^2 + ( (-V_ERR*mean(v_hotwire_arr))/(data.v^2) )^2  ); % unitless

       % Store in global arr
       angle_data_arr(i).len_scale(j) = len_scale;
       angle_data_arr(i).len_scale_ERR(j) = len_scale_ERR;
       angle_data_arr(i).v_normalized(j) = v_normalized;
       angle_data_arr(i).v_normalized_ERR(j) = v_normalized_ERR;
       angle_data_arr(i).v_rms(j) = v_rms;
   end
end
```

```
Re_AVG =

   3.8330e+05


Re_ERR =

   1.0730e+04
```

V/Vinf

```
colors = ["green", "red", "blue", "black"];
shapes = ["-o", "-*", "-x", "-^"];
```

```
hold on
for i = 1:length(ANGLES)
    errorbar(angle_data_arr(i).v_normalized, angle_data_arr(i).len_scale,...
        angle_data_arr(i).len_scale_ERR,... % yneg
        angle_data_arr(i).len_scale_ERR,... % ypos
        angle_data_arr(i).v_normalized_ERR,... % xneg
        angle_data_arr(i).v_normalized_ERR,... % xpos
        shapes(i), 'MarkerFaceColor', colors(i)...
   )
end
xlabel("V / Vinf")
ylabel("y / Lo")
xlim([.3, 1])
legend('a=-5', 'a=0', 'a=5', "a=20", 'AutoUpdate', 'off', 'Location', 'northwest')
```



Vrms/Vinf

```
colors = ["green", "red", "blue", "black"];
shapes = ["-o", "-*", "-x", "-^"];

clf % clear previous figure
hold on
for i = 1:length(ANGLES)
    errorbar(angle_data_arr(i).v_rms, angle_data_arr(i).len_scale, angle_data_arr(i).len_scale_ERR, shapes(i), 'MarkerFaceColor', colors(i))
end
xlabel("Vrms / Vinf")
ylabel("y / Lo")
legend('a=-5', 'a=0', 'a=5', "a=20", 'AutoUpdate', 'off', 'Location', 'northeast')
```