Table of Contents

Preamble

```
close all;
clear variables;
clc;

set(groot, 'defaultTextInterpreter', 'Latex');
set(groot, 'defaultLegendInterpreter', 'Latex');
set(groot, 'defaultAxesTickLabelInterpreter','latex');
set(groot, 'defaultLegendInterpreter','latex');
set(groot, 'defaultTextFontSize', 12);
set(groot, 'defaultAxesFontSize', 16);
set(groot, 'defaultLineLineWidth', 2);
```

Load data

```
%fileDir = '/Users/christopherbianco/Desktop/School_Code/Wind Physics/HW1';
%Mac

fileDir = 'C:\Users\Christopher\Desktop\School_Code\Wind Physics\Midterm';
%Windows

data = load(fullfile(fileDir, 'NWTC_M5_mystery_dataset.mat'));
```

Data Analysis from sonic

```
sonic_heights = [15;61;74;100;119];
% Load variables
TKE = zeros(size(sonic_heights));
TKE_stddev = zeros(size(sonic_heights));
uw = zeros(size(sonic_heights));
uw_stddev = zeros(size(sonic_heights));
```

```
vw = zeros(size(sonic_heights));
buoyFlux = zeros(size(sonic_heights));
buoyFlux_stddev = zeros(size(sonic_heights));
u_mean = zeros(size(sonic_heights));
Temp =  zeros(size(sonic_heights));
wT = zeros(size(sonic_heights));
wp = zeros(size(sonic_heights));


for ii = 1:length(uw)
    % Get time series of u, v, w, and T at each height
    % Note that u,v are already rotated so that u is in the direction
    % of the mean wind over the 10-minute interval
    u_vals = data.(['Sonic_u_', num2str(sonic_heights(ii)), 'm']).val;
    v_vals = data.(['Sonic_v_', num2str(sonic_heights(ii)), 'm']).val;
    w_vals = data.(['Sonic_w_', num2str(sonic_heights(ii)), 'm']).val;
    T_vals = data.(['Sonic_Temp_clean_', num2str(sonic_heights(ii)),
'm']).val + 273.15;

    % Compute requested quantities
    u_mean(ii) = mean(sqrt(u_vals.^2 + v_vals.^2), 'omitnan');
    mean_u_vals = movmean(u_vals, length(u_vals)/20); % ~30s lowpass filter
    mean_v_vals = movmean(v_vals, length(v_vals)/20); % (600 data points /
20 Hz)
    mean_w_vals = movmean(w_vals, length(w_vals)/20);
    mean_T_vals = movmean(T_vals, length(T_vals)/20);

    TKE_vals = 0.5 * ((u_vals - mean_u_vals).^2 + (v_vals - mean_v_vals).^2
+ (w_vals - mean_w_vals).^2);

    uw(ii) = mean((u_vals - mean_u_vals) .* (w_vals - mean_w_vals),
'omitnan');
    vw(ii) = mean((v_vals - mean_v_vals) .* (w_vals - mean_w_vals),
'omitnan');

    uw_stddev(ii) = std((u_vals - mean_u_vals) .* (w_vals - mean_w_vals),
[], 'omitnan');

    buoyFlux(ii) = mean((T_vals - mean_T_vals) .* (w_vals - mean_w_vals),
'omitnan') ...
        * 9.81 / mean(T_vals, 'omitnan');
    buoyFlux_stddev(ii) = std((T_vals - mean_T_vals) .* (w_vals -
mean_w_vals), [], 'omitnan') ...
        * 9.81 / mean(T_vals, 'omitnan');

    TKE(ii) = mean(TKE_vals, 'omitnan');
    TKE_stddev(ii) = std(TKE_vals, [], 'omitnan');

    Temp(ii) = mean(T_vals, 'omitnan');
    wT(ii) = mean((T_vals - mean_T_vals) .* (w_vals - mean_w_vals),
'omitnan');

    wp(ii) = mean ((w_vals - mean_w_vals), 'omitnan');
```

```
end
```

Data Analysis from cup

```
cup_heights = [3, 10, 30, 38, 55, 80, 87, 105, 122, 130];
U_av_cup = NaN(1, length(cup_heights));
U_std_cup = NaN(1, length(cup_heights));
TKE_cup = NaN(1, length(cup_heights));
TKE_std_cup = NaN(1, length(cup_heights));

for i = 1:length(cup_heights)
    cup_height = cup_heights(i);

    %Now, we need to take into account the naming convention
    fn = fieldnames(data);
    matchIdx = contains(fn, 'Cup_WS_') & contains(fn,
strcat(num2str(cup_height),'m'));
    match = fn(matchIdx);

    %Extract U
    U = data.(match{1}).val;

    %Do mean and standard deviation
    U_av_cup(i) = mean(U);
    U_std_cup(i) = std(U);

    up = U - mean(U);
    TKE_cup(i) = mean(up.^2);
    TKE_std_cup(i) = std(up.^2);
end
```

Part a code

```
k = 0.4;
g = 9.81;
u_star = (uw(1)^2 + vw(1)^2)^(1/4);

L = -(u_star^3 * Temp(1)/(k*g*wT(1)))
```

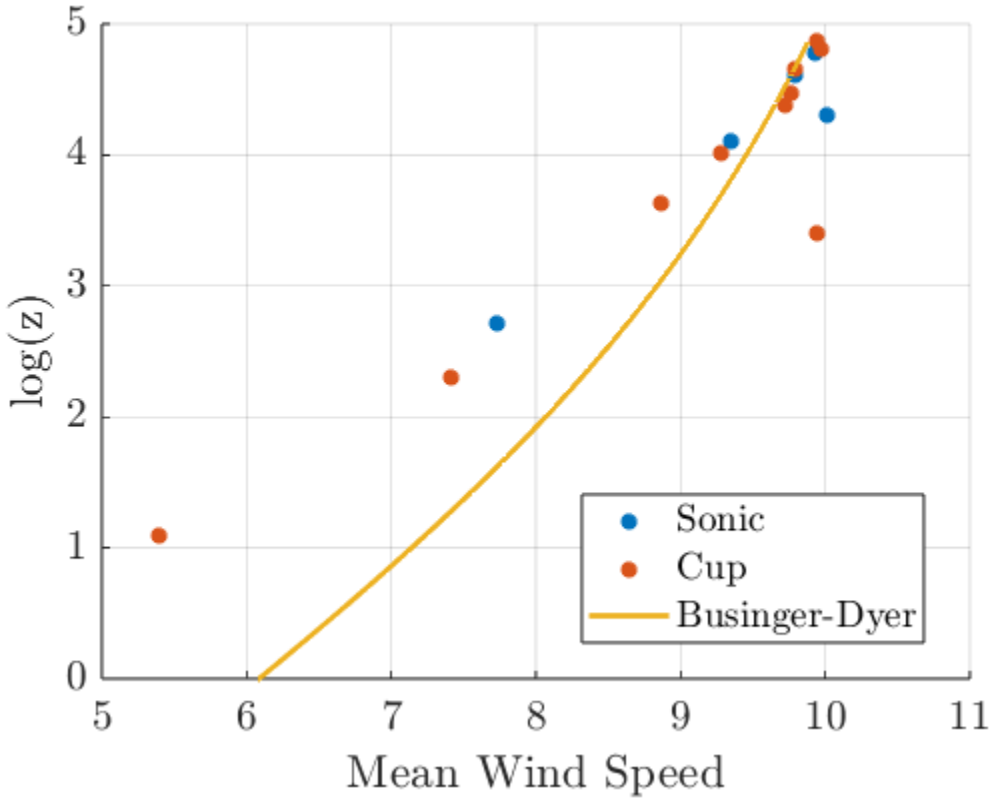
```
L =

  -36.7389
```

Part a short answer

Our Obukhov length is legative, indicating the flow is convective (statically unstable)

Part a code

```
%Plotting
figure(1); hold on;
xlabel('Mean Wind Speed');
ylabel('log(z)');
grid on

plot(u_mean, log(sonic_heights), '*');
plot(U_av_cup, log(cup_heights), '*');


%Fitting
z_all = [cup_heights(:); sonic_heights(:)];
U_all = [U_av_cup(:); u_mean(:)];
z0_guess = 2.5;
b0 = [z0_guess];

%Define psi
psi = @(z,L) -2*log((1+ ((1-(15.*z./L)).^(1/4)))/2) - log((1+((1-(15.*z./
L)).^(1/4)).^2)) + 2*atan(((1-(15.*z./L)).^(1/4))) - pi/2;

%Function for U
U_BD = @(b0, z) (u_star/k)*(log(z./b0(1)) + psi(z,L));

%Set robust fitting options
opts = statset('nlinfit');
opts.RobustWgtFun = 'bisquare';

b = nlinfit(z_all, U_all, U_BD, b0, opts);

z_fit = linspace(1, max(z_all), 1000);
U_fit = U_BD(b, z_fit);

plot(U_fit, log(z_fit))

legend({'Sonic', 'Cup', 'Businger-Dyer'}, 'Location','Best')

hold off
```



Part c

```
%Our roughness length is 0.0029 m. According to Stull fig 9.6, this value
%would be expected for something between off-sea winds in coastal areas and
%ntural snow surface (farmland). Based on fig ES-1 from the NWTC report,
%the site is quite flat with little to no vegetation. According to Stull,
%this should have a rougness legnth of around 0.006 m. So, the length we
%found here is smaller than we would expect, but is still reasonable.
```

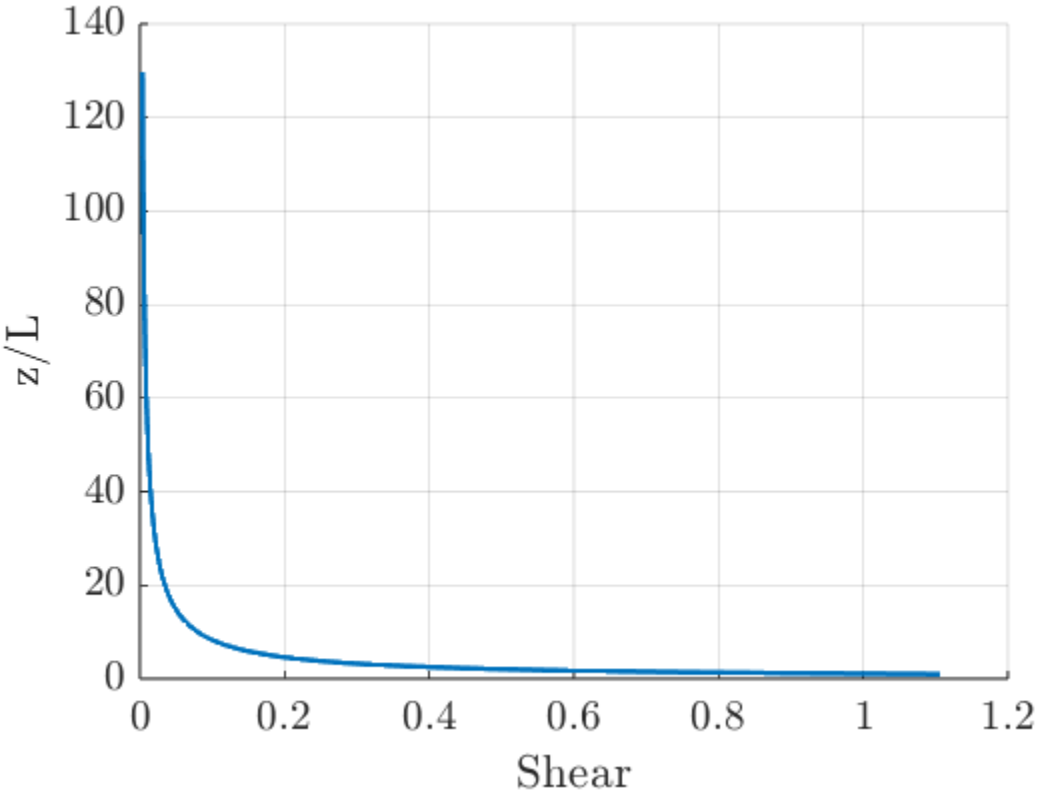
Part d

If the tower only had cup anemometers, we would not be able to calculate ustar as that required vertical temperature fluxes, which cup anemeometers cannot measure (they don't giv vertical velocity measurements)

```
%This means we wouldn't have the Obukhov length L. We could set L as a
parameter
%in our fit and do a two term fit. This would give us a value for L fit
%from the horizontal wind speed data, which we would still have. We could
%also neglect the contibution from psi alltogether, and sau U = ustar/k
%log(z/z0). Then, you could take two measurements for U at two differenmt z
%values, divide them by each other, and get a value for z0 without
%considering ustar or L. You could even do this for several values of U and
%z and average all the z0 values you obtain.
```

Part e

```
%We can estimate shear using the Businger-Dyer relations as well! Stull
%equation 9.7.5c details the function phi_M to use in this case.

shear = ((u_star)./(k.*z_fit)).*(1-(15.*z_fit/L)).^(-1/4);

figure(2); hold on;
xlabel('Shear');
ylabel('z/L');
grid on

plot(shear, z_fit);
hold off
```



Part f code

```
Shear_15 = ((u_star)./(k.*15)).*(1-(15.*15/L)).^(-1/4);
Rif = buoyFlux(1)/(uw(1) *Shear_15)
```

```
Rif =

   -0.6701
```

Part f short answer

Our flux Richardson number is -0.6701, which indicates static instability. This matches with our result from part a.

Part g

```
%Base on the fact that our Obokhov length and flux Richardson number are
%less that zero, we would expect these measurements to be taken sometime in
%the afternoon when the ABL is vbery unstable. We can see that bouyant
%production is clearly dominating, so there must be a lot of sun heating
%the ground to promote that.
```

Part h

We would expect vertical velocity fluctuations to be dominated by bouyancy flux because, as mentioned above, bouyant production is dominating in this flow. We hypothesize that these fluctuations only depend on flux and vertical position z. Bouyancy flux has units m^2/s^3 and z0 has units of m, so we can construct a velocity

```
%w_ref = ((g/theta)(w'theta')z)^(1/3)

%Checking our scale

for k = 1:length(sonic_heights)
    w_ref(k) = (buoyFlux(k)*sonic_heights(k))^(1/3);
    w_ref_check(k) = sqrt(wp(k)^2);
end

r = w_ref./w_ref_check

%W're off by two order of magnitudes, so our hypothesis is not great.
%However, the error is relatively consistent so a correction factor of about
% 115 would help.

r_corr = (w_ref./w_ref_check)/115

%This is a lot better. Interestingly, if we set our characterstic scale as
%our roughness lengthm we might get better results.


for k = 1:length(sonic_heights)
    w_ref_new(k) = (buoyFlux(k)*.0029)^(1/3);
    w_ref_check(k) = sqrt(wp(k)^2);
end

r_new = (w_ref_new./w_ref_check)

% This is a lot better! So, our final vertical velocity scale is
% w_ref = ((g/theta)(w'theta')z0)^(1/3), where z0 is the roughness length


r =
```

```
  204.0574  115.1282  162.8947  139.1566  114.3621

r_corr =

    1.7744    1.0011    1.4165    1.2101    0.9945

r_new =

   11.7993    4.1707    5.5330    4.2753    3.3156
```

Published with MATLAB® R2024b