# Description of MATLAB Scripts

Daniel van den Berg[1*] and Daan van der Hoek[1]

[1*]Delft Center for Systems and Control, Delft University of Technology, Mekelweg 2, Delft, 2628 CD, Zuid Holland, The Netherlands.

*Corresponding author(s). E-mail(s): d.g.vandenberg@tudelft.nl;

## Introduction

This document will describe the four MATLAB scripts that are used to create the figures in the linked publication. Each MATLAB script creates one of the figures in the work. Also included in the repository is the data required to produce the figures. This data is already post-processed from the original data acquired during the experiments. The data that is included with the MATLAB figures is the phase averaged and time-averaged data, in both cases binned to 40 mm and 60 mm bins. The 40 mm data gives the most detail at the cost of accuracy, the 60 mm data is more accurate at the loss of details. The data can be found in their respective folders. We refer to the table with cases presented in the paper to link individual files to different experiments.

Every script starts with a similar piece of code to identify the files that need to be loaded. For the phase-averaged data the piece of code is as follows:

```
clear all; clc; %close all;
addpath(genpath('.\Functions'))

fileList = dir('Phase_Averaged_Data\');
allFileNames = {fileList.name};
pattern = 'PA';
p=0;
for k = 1 : length(allFileNames)
        thisFileName = fullfile(fileList(k).folder,
            allFileNames{k});
        if ~contains(thisFileName, pattern, 'IgnoreCase',
            true)
                continue;
    end
    p = p+1;
    folders{p} = thisFileName;
end

stringsort = {'BL','NoYaw','yaw_0deg','yaw_90deg','
   yaw_180deg','yaw_270deg'};

for i_0 = 1:length(folders)
    comparestring = contains(folders,stringsort{i_0}) ;
```

```
    indstore(i_0) = find(comparestring == 1);
end
```

For the time-averaged data only the pattern that is used to identify is different. The second part of the code sorts the indices such that later in the script post-processing the data can be done in the same order as the cases presented in the table in the work. This needs to be done because MATLAB automatically places the '90deg' case at the back of the vector. This step simplifies the plotting of the data later on in the code.

## Importing The Data

The following piece of code shows how to import and extract the phase-averaged data. The first piece of the code selects the 'Binsize'. For this data set this number can either be 40 (as in the example) and 60, The 'case_numb' is only used for the phase-averaged data. For the Helix cases, the phase-averaged data is divided into 72 'average' time steps. The baseline cases exists of 6 phase-averaged data steps. For the figures using phase-averaged data, single case numbers are used to show a phase-averaged snapshot. Importing the data is done over the 'folders' variable, which has the list of the cases that were found in the previous piece of code. Once the data is loaded the 'binned_data' variable contains all the binned data. In this code the coordinates are imported ($X - Y - Z$), the magnitude of vorticity, $W$, the velocity components, $U_x$, $U_y$, $U_z$, vorticity components, $W_x$, $W_y$, $W_z$ and Q-criterion. Finally, *binned_data.RS* contains the Reynold stressed in all directions.

```
    Binsize = '40';
    case_numb = [1006 1036 1036 1036 1036 1036];
for i_1 = 1:length(folders)

    load(strcat(folders{indstore(i_1)},'\binned_data_pa_',
        ...
    num2str(Binsize),'mm_bin30_',num2str(case_numb(i_1)),'.
        mat'));
    disp(strcat('Loaded Case:',folders{indstore(i_1)}))

    X = binned_data.A{1};
    Y = binned_data.A{2};
    Z = binned_data.A{3};
    W = binned_data.OMEGA_mag;
    Ux = binned_data.U{1};
    Uy = binned_data.U{2};
    Uz = binned_data.U{3};
    Wx = binned_data.OMEGA{1};
    Wy = binned_data.OMEGA{2};
    Wz = binned_data.OMEGA{3};
    Q = binned_data.Q;

    P = [3 2 1];
    Wx(isnan(Wx)) = 0;
    Wy(isnan(Wy)) = 0;
    Wz(isnan(Wz)) = 0;
    Ux(isnan(Ux)) = 0;
    W(isnan(W)) = 0;
    Q(isnan(Q)) = 0;
    Q_store(:,:,:,i_1) = Q;
end
```

When during the binning process no particles are found within the volume it will result in a NaN value. Every NaN value is changed to a zero value which is required for plotting the surfaces. Finally, the dimensions of the 3D data are given in $y-x-z$ orientation. Further post-processing can be done on this data. For example, in the script 'plot_Velocity_Vorticity_Fig3.m' the data is interpolated to enhance the details in the data. The second part of the script is dedicated to plotting the figure. This method of importing is used in the following scripts

- 'plot_Velocity_Vorticity_Fig3.m',
- 'plot_Entrainment_Fig5b.m',
- 'plot_HubVortex_Qcrit_Fig6ab.m',

after which further post-processing is done accordingly.

## Specific Remarks Scripts

The scripts

- 'plot_Windspeed_Fig4.m',
- 'plot_Entrainment_Fig5b.m',
- 'plot_HubVortex_Qcrit_Fig6ab.m',

all perform further steps once the data is imported, or in the case of 'plot_Windspeed_Fig4.m' a different data set is used. Specific remarks about the pieces of code used are given in this section.

### Remarks 'plot_Windspeed_Fig4.m'

This script generates Figure 4 in the paper which shows the evolution of wind speed as a function of downstream distance. This script uses a time-averaged data set whereby the whole domain is binned in one single run. The following code loads the data from one of the measurements and extracts the x-dimension data. The for-loop imports post-processed time-averaged wind speed data.

```
    load('Phase_Averaged_Data\PA_CCW_NoYaw\
       binned_data_pa_40mm_bin30_1001.mat');
    X = binned_data.A{1}/1000;
    plot_idx = 1:293;
    vel_idx = 11:276;

resultFolder = dir('Wind_Speed_Data/fAP_pa_*');
for i = 1:length(resultFolder)
    load(strcat(resultFolder(i).folder,'\',resultFolder(i).
       name))
    [fAPvar(i,:),fAPavg(i,:)] = var(cell2mat(fAP),1,'
       omitnan');
    for i2 = 1:length(UxInt)
    bla = cell2mat(UxInt(i2));
    Ux(i2,:) = bla(1:282);
    end
    [Uxvar(i,:),Uxavg(i,:)] = var(Ux,1,'omitnan');
    clear Ux
end
```

The first ten entries from the data are removed from the data set as they only contain NaN values after which it is stored in the appropriately named '`bla`' vector. The imported data also contains the rotor power. This value represents the total power of the flow going through the area of the rotor disk. Since the power a wind turbine produces scales with wind speed to the third power, a small change in wind speed leads to a large change in power. For the explanation of MATLAB functions we would refer to the Mathworks documentation.

### Remarks 'plot_Entrainment_Fig5b.m'

This script contains the following piece of code in the import script. This part of the script transforms the imported data from cartesian coordinates to radial coordinates. This is used in the original work to calculate the entrainment over the rotor edge which is selected using the `rotor_ind` variable. The reason for not transforming the entire domain to radial coordinates is given in the manuscript.

```
% Convert the time averaged data from cartesian to
   cylindrical
% coordinates.
k_phi = 1;
MKE_rad = [];
for phi = -45:1:225
    upvrp = cosd(phi).*upwp + sind(phi).*upvp;
    F_RS = griddedInterpolant(permute(X,[2 1 3]),
       permute(Y,[2 1 3]),permute(Z,[2 1 3]),permute(
       upvrp,[2 1 3]));
    F_U = griddedInterpolant(permute(X,[2 1 3]),permute
       (Y,[2 1 3]),permute(Z,[2 1 3]),permute(Ux,[2 1
       3]));

    % extract slice
    r = 0:10:400;
    [xGrid,rGrid] = ndgrid(X(1,:,1),r);
    yGrid = rGrid.*sind(phi);
    zGrid = rGrid.*cosd(phi);

    MKE_rad = cat(3,MKE_rad,-F_U(xGrid,yGrid,zGrid).*
       F_RS(xGrid,yGrid,zGrid)./Ufs^3);
    k_phi = k_phi + 1;
end

MKE_rad_avg_full = mean(MKE_rad,3,"omitnan");
MKE_rad_avg_full_store(i_1,:,:) = MKE_rad_avg_full;

rotor_ind = 28:30; % These indices correspond to the
   rotor edge in the cylindrical data.
MKE_sum_full(i_1,:) = sum(MKE_rad_avg_full(:,rotor_ind)
   ,2)/length(rotor_ind) ;
```

### Remarks 'plot_HubVortex_Qcrit_Fig6ab.m'

The final script considered is the script that is responsible for plotting the figures containing the traced hub vortex. This script examines four different phase-averaged time steps for three different cases. **Because it examines a few time steps this script takes a long time to run, and requires some RAM to be available to load and store the data for analysis.**

The tracing is done using a Gaussian convolution method. The following piece of code generates a grid, named `test_grid_in` that creates a grid of ones and zeros. Because we know that the hub vortex remains within one rotor area we exclude every piece of data outside that area by multiplying our data grid with `test_grid_in`. A convolution of this new grid with a Gaussian curve will indicate the location of the hub vortex based on the location where the convolution is highest. The following piece of code performs the analysis:

```
%% Full domain inner vortex
        test_grid_in = sqrt(zint.^2+yint.^2) < Drot
            *0.75;
        y_0 = round(length(yint)/2);
        z_0 = round(length(yint)/2);
        sig_y = 90;
        sig_z = 90;
        F_gauss = -1*exp(-(((yint-yint(z_0,y_0)).^2)
            /(2*sig_y.^2)+((zint-zint(z_0,y_0)).^2)/(2*
            sig_y.^2)));
        convolute = conv2(F_gauss,Wdata'.*test_grid_in,
            'same');
        [~,I] = max(convolute,[],'all','omitnan');
        [gaus_z,gaus_y] = ind2sub(size(zint),I);
        z_gaus_inner(p_i_0,p,i_1,:) = zint(gaus_z,
            gaus_y);
        y_gaus_inner(p_i_0,p,i_1,:) = yint(gaus_z,
            gaus_y);

        Wframe = fliplr(squeeze(Q(:,i2,:)));
        Wdata = interp2(Yframe',Zframe',Wframe',yint,
            zint,'spline');
```

A similar piece of code can be found that goes through the same steps but then for the tip vortices. This data is *not used in the manuscript* but the code is included for completeness sake.

```
        %% Full domain tip vortex
        test_grid_out = sqrt(zint.^2+yint.^2) > Drot
            *0.2;
        sig_y = 90;
        sig_z = 90;
        F_gauss = 1*exp(-(((yint-yint(z_0,y_0)).^2)/(2*
            sig_y.^2)+((zint-zint(z_0,y_0)).^2)/(2*
            sig_y.^2)));
        convolute = conv2(F_gauss,Wdata'.*test_grid_out
            ,'same');
        [~,I] = max(convolute,[],'all','omitnan');
        [gaus_z,gaus_y] = ind2sub(size(zint),I);
        z_gaus_outer(p_i_0,p,i_1,:) = zint(gaus_z,
            gaus_y);
        y_gaus_outer(p_i_0,p,i_1,:) = yint(gaus_z,
            gaus_y);
```