The Pennsylvania State University

The Graduate School

# **Unsteady Dynamics Modeling of Helicopters**

**Project 3**

By Zarif Rahman

Submitted by

Zarif Rahman

Contents

[**Unsteady Dynamics Modeling of Helicopters** 1](#_Toc101805264)

[Introduction: 3](#_Toc101805265)

[Objective: 3](#_Toc101805266)

[Technical Approach 3](#_Toc101805267)

[Results 4](#_Toc101805268)

[Conclusion 7](#_Toc101805269)

[Code 8](#_Toc101805270)

[References: 37](#_Toc101805271)

## Introduction:

## Unsteady aerodynamics is a change of the forcing over a period. Here the hysteresis effects of the blade become increasingly important, hence the previous wake calculations or inflow calculations are not accurately representing the complete physics. Aspects such as higher harmonics or flutter on the dynamics structure often led to the total equation of motion to be impacted significantly and cause multiple issues. These issues are often very critical with little natural damping which is why accurate calculations are necessary. These higher-order calculations are meant to allow for the future design choices for the aircraft that are not heavier than needed, all the while having sufficient damping and structure to be able to withstand these aerodynamic forces.

## Objective:

## The objective is to characterize and calculate these unsteady effects on the blade elements. The primary goal for my code was to calculate for the the effect of a step change on the angle of attack. This then would allow for the calculation of coefficient of lifts as it approached the quasi lift that is expected at an angle of attack. Another aspect of the code was to see the impact of the higher harmonics and how this impact is different for the angle of attack result from the wake calculations.

## Technical Approach

For the third project, my goal was to create a code that could use Beddoes’s algorithm for unsteady aerodynamics but does not account for compressibility. Another primary assumption of this code is that it assumes trim using the same parameters but utilizes only up to the fifth harmonic for the collective pitch. Firstly, the code takes a high harmonics calculation from the initial trim solution and reruns the blade element code for the new forces. Initially, the code from the wake calculations solves for the angle of attack using the blade element code and that angle of attack is used for the calculation of the indicial responses. Here I also used W.P. Jones constants for the indicial response, and I plotted the phi function. This then utilized this information to solve for the coefficient of lift which was compared with harmonic calculations as seen in figure 5. Overall, convergence here was not a problem but the bigger issue was that the wake calculations were still having issues. All in all, the codes utilize the previous blade element code and wake code, and then implement the unsteady forcing to it.

Specifications (Shinoda, Yeo, Norman):

* Airfoils: SC1095 & SC1094R8
* Blades: 4
* Radius: 26.83 [ft]
* Angular velocity: 258 [rpm]
* Rotor Disk Area: 2261.5 [ft2]
* Blade Area: 186.9 [ft2]
* Root Cut out = 4.5611 [ft]
* f = 32.95 ft^2 to 36.34 ft^2 Chord, from 20.76/20.965 in using 20.76 ignoring taper
* Clα = 2𝛑
* Cd0 = 0.01
* V∞ = 174 mph
* No twist distribution
* Constant Circulation along the blade (averaged)
* Only up to the 5th harmonic
* Ran only time steps up to s = 100 for run time

## Results

Chart, line chart

Description automatically generated

Figure (1)

The first plot for the collective of the fifth harmonic with the total sum of the harmonics together and plots it as a function of the azimuth. It shows the solutions to be like a sinusoidal graph but is a collection of all the harmonics.

Chart, bar chart, histogram

Description automatically generated

Figure (2) This plot is meant to be the collective pitches at different harmonics. There seems to be an issue with this plot and why it fluctuates so often.

Chart

Description automatically generated

Figure (3)

Chart

Description automatically generated

Figure (4) Here the air load at a constant angle of attack after a quasi-steady change is seen in the blue, while the green is representing the lift coefficient change while accounting for in unsteady effects

Chart

Description automatically generated

Figure (5) This figure compares the lift coefficient with and without harmonics. The plot colors are off but the yellow is meant to be the one without harmonics and the purple is with harmonics.

Chart, surface chart

Description automatically generated

Figure (6)

## The fourth plot calculates the lift without and with collective calculations. The initial value for the lift drastically varied but both seemed to approach 0.1 to 0.18. The primary reason for this is how the initial conditions are set for it but overall give the same magnitude of lift coefficient. At a higher time, domain, the lift is likely to approach the 0.18 value.

## Conclusion

In general, the unsteady aerodynamics causes time-based loading on the blade which impacts the dynamics and the lift of each blade element. The Beddoes algorithm shows that as time passes the lift varies drastically, due to the fluctuations in the angle of attack. The variation is caused by both flapping and pitching motion. At a higher fidelity model, all the coupling of the motions would output a lift more accurately of the flow. The result of the code showed that the changes in the angle of attack caused on the lift. Instead of quasi-steady assumptions, this unsteady solution requires a set amount of time because it converges to the new lift. All in all, the code results are similar to expectation, but there remains an error from the previous code that causes convergence issues. To fix this I made a few changes to the code, but this also causes other problems in the wake calculation as seen in figure (6). The problem was it caused a spike at the retreating and advancing ends but generally had the previous shape. The code does give a rough estimate of the unsteady forcing and the impact of collective changes over time.

## Code

The code is also on Github at: <https://github.com/zarif55/UH60-Modeling>

clc

close

clear

%% UH 60

%Constants

% Airfoil SC1095 & SC1094R8

[Nb, N\_psi, R\_i, a, omega\_i,A\_i,A\_blade\_i,cd0,f\_i,T\_i,c\_i,rho,k,...

L\_tr,x\_cg,y\_cg,h,vb, lock, theta\_tw] = find\_constants;

[R,A,A\_blade,f,v\_inf\_mph,v\_inf,c,sigma,omega,v\_tip,alpha\_d,alpha,...

mu,T,CT\_constant\_T,lambda\_hover,psi,e,r,Cl\_alpha,nu,dr,...

linear\_inflow\_power,MS\_inflow\_power,w] = find\_general\_inflow(Nb, R\_i,...

a, omega\_i,A\_i,A\_blade\_i,f\_i,T\_i,c\_i,rho);

%% Assignment 1 %%

%% Inflow Functions

for Nb = 4

[lambda\_TPP\_UH60\_FF,lambda\_i\_TPP\_UH60\_FF,lambda\_output]...

= find\_linear\_inflow(mu, alpha, CT\_constant\_T);

[trim\_inflow, CH\_TPP\_UH60\_FF, CY\_TPP\_UH60\_FF, beta, beta\_dot,...

beta\_star, theta, theta\_dot, theta\_star]...

= find\_trim\_inflow(mu, psi, CT\_constant\_T, sigma, f, A,L\_tr, R, vb,...

linear\_inflow\_power, theta\_tw,lambda\_TPP\_UH60\_FF, omega, x\_cg, y\_cg,...

h, T, lock, a, cd0);

[cn\_1] = find\_MS\_1\_inflow(mu(174), psi, nu, alpha, CT\_constant\_T);

[cn\_3] = find\_MS\_3\_inflow(mu(174), psi, nu, alpha, CT\_constant\_T);

[w,lambda\_MS\_total] = find\_MS\_real\_inflow(cn\_1, cn\_3, w, psi,nu,...

CT\_constant\_T, mu(174));

[u\_t, u\_p\_linear, u\_p\_MS, T\_linear\_bar, Q\_linear\_bar, T\_MS\_bar,...

Q\_MS\_bar,P\_linear\_bar, P\_MS\_bar] = ...

find\_bemt(r, psi, mu(174), beta(:,174), beta\_star(:,174),v\_inf, ...

lambda\_TPP\_UH60\_FF(174),Cl\_alpha,lambda\_MS\_total, theta(:,174),...

rho, c, R, cd0, N\_psi, dr, Nb, omega, v\_tip);

epsilon\_t\_linear\_it = 1;

epsilon\_p\_linear\_it = 1;

while (epsilon\_t\_linear\_it > 0.004) && (epsilon\_p\_linear\_it > 0.004)

T = T\_linear\_bar;

linear\_inflow\_power = P\_linear\_bar;

CT\_constant\_T = (T./(rho\*A\*v\_tip^2)) \* ones(1, 223);

[lambda\_TPP\_linear\_it,lambda\_i\_linear\_it,lambda\_output\_linear\_it] ...

= find\_linear\_inflow(mu, alpha, CT\_constant\_T);

[trim\_inflow\_linear\_it, CH\_TPP\_UH60\_FF\_linear\_it,...

CY\_TPP\_UH60\_FF\_linear\_it, beta\_linear\_it, beta\_dot\_linear\_it, ...

beta\_star\_linear\_it, theta\_linear\_it, theta\_dot\_linear\_it, ...

theta\_star\_linear\_it] = find\_trim\_inflow(mu, psi, CT\_constant\_T,...

sigma, f, A,L\_tr, R, vb, linear\_inflow\_power, theta\_tw,...

lambda\_TPP\_linear\_it, omega, x\_cg, y\_cg, h, T, lock, a, cd0);

[u\_t\_linear\_it, u\_p\_linear\_linear\_it, u\_p\_MS\_linear\_it, ...

T\_linear\_bar\_linear\_it, Q\_linear\_bar\_linear\_it, T\_MS\_bar\_linear\_it,...

Q\_MS\_bar\_linear\_it,P\_linear\_bar, P\_MS\_bar\_linear\_it, alpha\_linear,alpha\_MS] = ...

find\_bemt(r, psi, mu(174), beta\_linear\_it(:,174), beta\_star\_linear\_it(:,174),v\_inf, ...

lambda\_TPP\_linear\_it(174),Cl\_alpha,lambda\_MS\_total, theta(:,174),...

rho, c, R, cd0, N\_psi, dr, Nb, omega, v\_tip);

epsilon\_t\_linear\_it = abs(T\_linear\_bar - T);

epsilon\_p\_linear\_it = abs(P\_linear\_bar - linear\_inflow\_power);

end

epsilon\_t\_MS\_it = 1;

epsilon\_p\_MS\_it = 1;

while (epsilon\_t\_MS\_it > 0.004) && (epsilon\_p\_MS\_it > 0.004)

T = T\_MS\_bar;

MS\_inflow\_power = P\_MS\_bar;

CT\_constant\_T = (T./(rho\*A\*v\_tip^2)) \* ones(1, 223);

[lambda\_TPP\_MS\_it,lambda\_i\_TPP\_MS\_it,lambda\_output] = ...

find\_linear\_inflow(mu, alpha, CT\_constant\_T);

[trim\_inflow\_MS\_it, CH\_TPP\_UH60\_FF\_MS\_it, CY\_TPP\_UH60\_FF\_MS\_it, beta\_MS\_it,...

beta\_dot\_MS\_it, beta\_star\_MS\_it, theta\_MS\_it,...

theta\_dot\_MS\_it, theta\_star\_MS\_it] = find\_trim\_inflow(mu, psi, CT\_constant\_T, sigma,...

f, A,L\_tr, R, vb, MS\_inflow\_power, theta\_tw,lambda\_TPP\_MS\_it, omega,...

x\_cg, y\_cg, h, T, lock, a, cd0);

[cn\_1] = find\_MS\_1\_inflow(mu(174), psi, nu, alpha, CT\_constant\_T);

[cn\_3] = find\_MS\_3\_inflow(mu(174), psi, nu, alpha, CT\_constant\_T);

[w,lambda\_MS\_total] = find\_MS\_real\_inflow(cn\_1, cn\_3, w, psi,nu, CT\_constant\_T, mu(174));

[u\_t, u\_p\_linear, u\_p\_MS, T\_linear\_bar, Q\_linear\_bar, T\_MS\_bar,...

Q\_MS\_bar, P\_linear\_bar, P\_MS\_bar, dL\_MS, alpha\_linear,alpha\_MS, dFx\_MS, dFz\_MS]...

= find\_bemt(r, psi, mu(174), beta\_MS\_it(:,174), ...

beta\_star\_MS\_it(:,174),v\_inf, lambda\_TPP\_MS\_it(174),...

Cl\_alpha,lambda\_MS\_total, theta\_MS\_it(:,174), rho, c,...

R, cd0, N\_psi, dr, Nb, omega, v\_tip);

epsilon\_t\_MS\_it = T\_MS\_bar - T;

epsilon\_p\_MS\_it = P\_MS\_bar - MS\_inflow\_power;

P\_MS = P\_MS\_bar;

end

end

%% Display

for Nb = 4

disp("Thrust Linear"); disp(T\_linear\_bar)

disp("Thrust Mangler Squire"); disp(T\_MS\_bar\_linear\_it)

disp("Power Linear"); disp(P\_linear\_bar)

disp("Power Mangler Squire"); disp(P\_MS\_bar\_linear\_it)

end

%% Plots P1

for Plots = Nb

figure (1)

plot (mu, lambda\_TPP\_UH60\_FF)

title("Uniform inflow vs advance ratio")

xlabel("advance ratio mu"); ylabel("uniform inflow lambda")

figure (2)

[r\_grid, psi\_grid] = meshgrid(r,psi);

surf(r\_grid.\*cos(psi\_grid), r\_grid.\*sin(psi\_grid), abs(lambda\_MS\_total));

title("Non uniform Inflow")

zlabel("Non uniform Inflow lambda")

legend("Mangler & Squire Total Inflow")

figure(3)

plot(psi, real(dFx\_MS))

title("Force in x vs azimuth at all r locations")

xlabel("psi"); ylabel("dFx\_MS")

figure(4)

plot(psi, real(dFz\_MS))

title("Force in z vs azimuth at all r locations")

xlabel("psi"); ylabel("dFz\_MS")

figure(5)

plot(r, real(dFx\_MS'))

title("Force in x vs radial location at all azimuth locations")

xlabel("r"); ylabel("dFx\_MS")

figure(6)

plot(r, real(dFz\_MS'))

title("Force in z vs radial location at all azimuth locations")

xlabel("r"); ylabel("dFz\_MS")

figure(7)

uniform\_crt = ones(361, 100);

surf(r\_grid.\*cos(psi\_grid), r\_grid.\*sin(psi\_grid), uniform\_crt\*lambda\_TPP\_UH60\_FF(174))

title("Uniform Inflow"); zlabel("Uniform Inflow lambda")

end

%% Assignment 2 %%

%% Wake model

%% Constants

%defintions

%psi\_w = wake age;

%psi\_b = blade azimuthal position

del\_psi\_w = 30;

psi\_b = psi;

psi\_w = (0:del\_psi\_w:3\*360)\*(pi/180);

epsilon\_wake = 1000;

%% Wake functions

while epsilon\_wake > 1

T\_old\_store = T\_MS\_bar;

[constant\_circulation,constant\_circulation\_bar] = ...

find\_constant\_circulation(rho, omega, R, Nb, T\_MS\_bar, v\_tip, A);

circulation\_v = constant\_circulation;

[xv, yv, zv, mu\_z, mu\_x, E, r\_wake] = ...

find\_prescribed\_wake(lambda\_TPP\_linear\_it(174), mu(174), alpha(174),...

v\_tip, r(100), psi, psi\_w, psi\_b);

[xv2, yv2, zv2, mu\_z2, mu\_x2, E2, r\_wake2] = ...

find\_prescribed\_wake(lambda\_TPP\_linear\_it(15), mu(15), alpha(15),...

v\_tip, r(100), psi, psi\_w, psi\_b);

[r\_rotor] = find\_rotor(psi\_b, r, c);

disp("display checkpoint1")

[V, lambda\_wake] = find\_biot\_savart(circulation\_v,...

v\_tip, r\_wake, psi\_w, psi\_b, r\_rotor, r);

disp("display checkpoint2")

lambda\_MS\_wake = lambda\_wake(:,:,3);

for P\_B = 1:length(psi\_b)

T\_old\_store = T\_MS\_bar;

[u\_t, u\_p\_linear, u\_p\_MS, T\_linear\_bar, Q\_linear\_bar, T\_MS\_bar,...

Q\_MS\_bar, P\_linear\_bar, P\_MS\_bar, dL\_MS, alpha\_linear,alpha\_wake,...

dFx\_MS, dFz\_MS] = find\_bemt(r, psi(P\_B), mu(174), beta\_MS\_it(P\_B,174), ...

beta\_star\_MS\_it(P\_B,174),v\_inf, lambda\_TPP\_MS\_it(174),...

Cl\_alpha,lambda\_MS\_wake(:,P\_B), theta\_MS\_it(P\_B,174), rho, c,...

R, cd0, N\_psi, dr, Nb, omega, v\_tip);

P\_wake = P\_MS\_bar;

end

epsilon\_wake = 100\*(T\_MS\_bar - T\_old\_store)/T\_old\_store;

disp(epsilon\_wake)

end

%% Plot P2

for Plots = Nb

figure(8)

plot3(xv, yv, zv)

xlabel("x"); ylabel("y"); zlabel("z");

figure(9)

plot3(xv(:,15), yv(:,15), zv(:,15))

title("Beddoes Wake Geometry at Blade azimuth of 15");

xlabel("x"); ylabel("y"); zlabel("z")

figure(10)

plot(psi\_w, zv(:,15))

title("Beddoes Wake Geometry 2D x vs. z")

figure(11)

plot(psi\_w, zv(:,1))

title("Beddoes Wake Geometry 2D wake angle vs. z; mu = 0.3496")

figure(12)

plot(psi\_w, zv2(:,1))

title("Beddoes Wake Geometry 2D wake angle vs. z; mu = 0.02829")

disp("Thrust Wake ="), disp(T\_MS\_bar\_linear\_it);

disp("Power Wake ="), disp(P\_MS\_bar\_linear\_it);

figure(13)

[r\_grid\_2, psi\_b\_grid] = meshgrid(r,psi\_b);

surf(r\_grid\_2.\*cos(psi\_b\_grid), r\_grid\_2.\*sin(psi\_b\_grid), abs(lambda\_MS\_wake'));

title("Inflow over blade azimuth with constant Circulation");

xlabel("x"); ylabel("y"); zlabel("lambda")

end

%% Assignment 3 %%

%% Unsteady Aerodynamics

u\_p = real(u\_p\_MS);

v = v\_inf;

[theta\_5, theta\_root, theta\_all] = find\_harmonics(beta, beta\_dot,...

beta\_star, theta, theta\_dot, theta\_star, trim\_inflow(:,174), u\_p, u\_t, psi);

[Cl, L, phi\_wp\_jones] = find\_beddoes\_mdpt(A, v(174), rho, alpha\_wake, theta);

[u\_t\_unsteady, u\_p\_linear\_linear\_it, u\_p\_MS\_linear\_it, ...

T\_linear\_bar\_linear\_it, Q\_linear\_bar\_linear\_it, T\_MS\_unsteady,...

Q\_MS\_unsteady,P\_unsteady, P\_MS\_unsteady, alpha\_unsteady1,alpha\_unsteady2] = ...

find\_bemt(r, psi, mu(174), beta\_linear\_it(:,174), beta\_star\_linear\_it(:,174),v\_inf, ...

lambda\_TPP\_linear\_it(174),Cl\_alpha,lambda\_MS\_total, theta\_5(:),...

rho, c, R, cd0, N\_psi, dr, Nb, omega, v\_tip);

[Cl\_5, L\_5, phi\_wp\_jones\_5] = find\_beddoes\_mdpt(A, v(174), rho, alpha\_unsteady2, theta\_5);

t\_md = 1:length(Cl);

%% Plot P3

for Plots = Nb

figure(14)

plot(theta(:,174))

title("Collective with Hamonics(5)");

xlabel("psi"); ylabel("theta");

figure(15)

hold on

plot(psi/(pi/180), theta\_all(1, :))

plot(psi/(pi/180), theta\_all(2, :))

plot(psi/(pi/180), theta\_all(3, :))

plot(psi/(pi/180), theta\_all(4, :))

plot(psi/(pi/180), theta\_all(5, :))

hold off

figure(16)

hold on

title("Lift vs time");

xlabel("s"); ylabel("C\_L");

plot(t\_md, Cl)

plot(t\_md, Cl(100)\*ones(1, 100))

hold off

figure(17)

hold on

title("Lift vs time");

xlabel("s"); ylabel("C\_L");

plot(t\_md, Cl)

plot(t\_md, Cl\_5)

legend("Lift coefficient without harmonics","Lift coefficient with 5th harmonics")

hold off

figure(18)

hold on

title("W.P. Jones Phi Indicial Response");

xlabel("s"); ylabel("Indical Response");

plot(t\_md, phi\_wp\_jones)

hold off

end

%% constants

function [Nb, N\_psi, R\_i, a, omega\_i,A\_i,A\_blade\_i,cd0,f\_i,T\_i,c\_i,rho,k,...

L\_tr,x\_cg,y\_cg,h,vb, lock, theta\_tw] = find\_constants

Nb = 4;

N\_psi = 360;

R\_i = 26.85; %ft

a = 2\*pi; %1/rad

omega\_i = 258; %rpm

A\_i = 2261.5; %ft^2

A\_blade\_i = 186.8; %ft^2

cd0 = 0.01;

f\_i = 35.04; %ft

T\_i = 20000; %lbs

c\_i = 20.76; %in

rho = 1.225; %kg/m^3

k = 1.15;

L\_tr = 45;

x\_cg = 0;

y\_cg = 0;

h = 5;

vb = 1;

lock = 8;

theta\_tw = 0;

%% find\_linear\_inflow function

% mu: advance ratio

% alpha: angle of attack

% CT\_constant\_T: thrust coefficient

function[lambda\_TPP\_UH60\_FF,lambda\_i\_TPP\_UH60\_FF, lambda\_output] = find\_linear\_inflow(mu, alpha, CT\_constant\_T)

%% defining constants

%alpha = -2 \* ones(1, 223);

%% defining functions

lambda\_H\_60 = @(CT\_constant\_T) sqrt(CT\_constant\_T./2);

lambda\_i\_FF\_60 = @(lambda\_FF\_60\_old) CT\_constant\_T./ ...

(2\*sqrt((((mu.\*sin(alpha)-lambda\_FF\_60\_old)).^2)...

+(mu.\*cos(alpha)).^2));

lambda\_FF\_60 = @(lambda\_i\_FF\_60) (-mu.\*sin(alpha) + lambda\_i\_FF\_60);

%% define lambda\_TPP\_UH60\_FF

% assuming CT\_constant\_T is 1 x N

% lambda\_TPP\_UH60\_FF would be 1 x N

lambda\_TPP\_UH60\_FF = lambda\_H\_60(CT\_constant\_T);

lambda\_output = lambda\_TPP\_UH60\_FF;

%% Finding Linear inflow of lambda\_TPP\_UH60\_FF

% Using 100 iterations to find 223 different values of mu

% mu should be 1 x 223

% lambda\_TPP\_UH60\_FF should be 1 x 223 as well

for n=1:100

lambda\_TPP\_UH60\_FF\_old = lambda\_TPP\_UH60\_FF; %lambda\_TPP\_UH60\_FF\_old is 1 x N

lambda\_i\_TPP\_UH60\_FF= lambda\_i\_FF\_60(lambda\_TPP\_UH60\_FF\_old);

lambda\_TPP\_UH60\_FF = lambda\_FF\_60(lambda\_i\_TPP\_UH60\_FF);

end

%% find\_trim\_inflow function

function[trim\_inflow, CH\_TPP\_UH60\_FF, CY\_TPP\_UH60\_FF, beta, beta\_dot, beta\_star, theta, theta\_dot, theta\_star]...

= find\_trim\_inflow(mu, psi, CT\_constant\_T, sigma, f, A,L\_tr, R, vb, CP\_UH60\_FF, theta\_tw,lambda\_TPP\_UH60\_FF, ...

omega, x\_cg, y\_cg, h, T, lock, a, cd0)

%% constants

M\_yF\_UH60\_FF = 0;

M\_xF\_UH60\_FF = 0;

CH\_TPP\_UH60\_FF = 0;

CY\_TPP\_UH60\_FF = 0;

epsilon = 1;

theta\_tw = 0;

%% defining functions

beta\_1c = @(CH\_TPP\_UH60\_FF) ((-x\_cg./h)+(M\_yF\_UH60\_FF./(T.\*h))+...

(CH\_TPP\_UH60\_FF./CT\_constant\_T))./(1+(((vb.^2)-1)./(lock.\*(2.\*h.\*CT\_constant\_T./R./sigma./a))));

beta\_1s = @(CY\_TPP\_UH60\_FF) ((y\_cg/h)-(M\_xF\_UH60\_FF/(T\*h))+...

(CY\_TPP\_UH60\_FF./CT\_constant\_T ))./(1+(((vb.^2)-1)...

./(lock.\*(2.\*h.\*CT\_constant\_T./R./sigma./a))));

alpha\_shaft = @(CH\_TPP\_UH60\_FF) ((((x\_cg./h)-(M\_yF\_UH60\_FF./(T.\*h))+...

(((((vb.^2)-1)./(lock.\*(2.\*h.\*CT\_constant\_T./R./sigma./a)))).\*...

CH\_TPP\_UH60\_FF./CT\_constant\_T )))./(1+(((vb.^2)-1)./...

(lock.\*(2.\*h.\*CT\_constant\_T./R./sigma./a)))))+...

((1/2).\*(f./A).\*(mu.^2)./CT\_constant\_T );

phi\_shaft = @(CY\_TPP\_UH60\_FF)((((y\_cg./h)-(M\_xF\_UH60\_FF./(T.\*h))-...

(((((vb.^2)-1)./(lock.\*(2.\*h.\*CT\_constant\_T./R./sigma./a)))).\*CY\_TPP\_UH60\_FF./...

CT\_constant\_T )))./...

(1+(((vb^2)-1)./(lock.\*(2.\*h.\*CT\_constant\_T./R./sigma./a))))) -...

(CP\_UH60\_FF.\*R./CT\_constant\_T./L\_tr);

theta\_knot\_fn = @(beta\_1s\_UH60\_FF) (((6.\*CT\_constant\_T./sigma./a).\*...

(1+(3/2.\*mu.^2)))- ((0.75.\*theta\_tw).\*(1-(1.5.\*mu.^2)+...

(1.5\*mu.^4))) +(1.5.\*lambda\_TPP\_UH60\_FF).\*(1-(0.5.\*mu.^2))) + ...

((4./lock).\*mu.\*((vb^2)-1).\*beta\_1s\_UH60\_FF)./ ...

(1 - (mu.^2) +(9/4\*mu.^4)) ;

theta\_1s = @(beta\_1c\_UH60\_FF, beta\_1s\_UH60\_FF,theta\_knot\_UH60\_FF)-...

beta\_1c\_UH60\_FF + ((1./(1+(1.5\*mu.^2))).\*((-8/3\*mu.\*...

(theta\_knot\_UH60\_FF+ (0.75.\*theta\_tw) - (0.75\*lambda\_TPP\_UH60\_FF)))+...

( (8/lock).\*((vb^2)-1).\*(beta\_1s\_UH60\_FF) ) ));

beta\_knot = @(beta\_1c\_UH60\_FF, theta\_1s\_UH60\_FF, theta\_knot\_UH60\_FF)...

(lock/vb.^2)\*(((1+(mu.^2)).\*theta\_knot\_UH60\_FF/8)+...

((1+mu.^2).\*theta\_tw./10)+((1/6).\*mu.\*...

(beta\_1c\_UH60\_FF+theta\_1s\_UH60\_FF))-(lambda\_TPP\_UH60\_FF/6));

theta\_1c = @(beta\_1s\_UH60\_FF, beta\_knot\_UH60\_FF, beta\_1c\_UH60\_FF)...

beta\_1s\_UH60\_FF + ((1./(1+(0.5\*mu.^2))).\*...

((4/3.\*mu.\*beta\_knot\_UH60\_FF) + ((8./lock).\*...

((vb.^2)-1).\*beta\_1c\_UH60\_FF)));

CH\_TPP = @(theta\_knot\_UH60\_FF, theta\_1s\_UH60\_FF,...

theta\_1c\_UH60\_FF, beta\_knot\_UH60\_FF) ((sigma.\*a./2).\* ...

((((1/2).\*mu.\*lambda\_TPP\_UH60\_FF.\*...

(theta\_knot\_UH60\_FF+(0.5.\*theta\_tw)))) - ...

(((1/6).\*beta\_knot\_UH60\_FF.\*theta\_1c\_UH60\_FF) + ...

((1/4).\*theta\_1s\_UH60\_FF.\*lambda\_TPP\_UH60\_FF) +...

((1/4).\*mu.\*beta\_knot\_UH60\_FF.^2)))) + (sigma.\*cd0.\*mu./4);

CY\_TPP = @(theta\_knot\_UH60\_FF, beta\_knot\_UH60\_FF,...

theta\_1s\_UH60\_FF, theta\_1c\_UH60\_FF) -(sigma.\*a./2).\*...

( ((3/4).\*mu.\*beta\_knot\_UH60\_FF.\*(theta\_knot\_UH60\_FF+((2/3).\*...

theta\_tw))) +((1/4).\*theta\_1c\_UH60\_FF.\*lambda\_TPP\_UH60\_FF) +...

((1/6).\*beta\_knot\_UH60\_FF.\*theta\_1s\_UH60\_FF.\*(1+(3.\*mu.^2) )) - ...

((3/2).\*mu.\*beta\_knot\_UH60\_FF.\*lambda\_TPP\_UH60\_FF));

theta\_fn = @(theta\_knot\_UH60\_FF, theta\_1s\_UH60\_FF, theta\_1c\_UH60\_FF, psi)...

theta\_knot\_UH60\_FF + theta\_1c\_UH60\_FF.\*cos(psi) + (theta\_1s\_UH60\_FF.\*sin(psi));

theta\_dot\_fn = @(theta\_1s\_UH60\_FF, theta\_1c\_UH60\_FF, psi) -...

theta\_1c\_UH60\_FF.\*sin(psi) + theta\_1s\_UH60\_FF.\*cos(psi);

theta\_star\_fn = @(theta\_dot) theta\_dot./omega;

beta\_fn = @(beta\_knot\_UH60\_FF, beta\_1s\_UH60\_FF, beta\_1c\_UH60\_FF, psi) beta\_knot\_UH60\_FF + (beta\_1c\_UH60\_FF.\*cos(psi)) + (beta\_1s\_UH60\_FF.\*sin(psi));

beta\_dot\_fn = @(beta\_1s\_UH60\_FF, beta\_1c\_UH60\_FF, psi) - beta\_1c\_UH60\_FF.\*sin(psi) + beta\_1s\_UH60\_FF.\*cos(psi);

beta\_star\_fn = @(beta\_dot) beta\_dot./omega;

% Based on mu., lambda, generate the data for plots

while epsilon > 0.004

CY\_TPP\_old = CY\_TPP\_UH60\_FF;

beta\_1c\_UH60\_FF = beta\_1c(CH\_TPP\_UH60\_FF);

beta\_1s\_UH60\_FF = beta\_1s(CY\_TPP\_UH60\_FF);

alpha\_shaft\_UH60\_FF = alpha\_shaft(CH\_TPP\_UH60\_FF);

phi\_shaft\_UH60\_FF = phi\_shaft(CY\_TPP\_UH60\_FF);

theta\_knot\_UH60\_FF = theta\_knot\_fn(beta\_1s\_UH60\_FF);

theta\_1s\_UH60\_FF = theta\_1s(beta\_1c\_UH60\_FF, beta\_1s\_UH60\_FF,theta\_knot\_UH60\_FF);

beta\_knot\_UH60\_FF = beta\_knot(beta\_1c\_UH60\_FF, theta\_1s\_UH60\_FF, theta\_knot\_UH60\_FF);

theta\_1c\_UH60\_FF = theta\_1c(beta\_1s\_UH60\_FF, beta\_knot\_UH60\_FF, beta\_1c\_UH60\_FF);

CH\_TPP\_UH60\_FF = CH\_TPP(theta\_knot\_UH60\_FF, theta\_1s\_UH60\_FF,theta\_1c\_UH60\_FF, beta\_knot\_UH60\_FF);

CY\_TPP\_UH60\_FF = CY\_TPP(theta\_knot\_UH60\_FF, beta\_knot\_UH60\_FF, theta\_1s\_UH60\_FF, theta\_1c\_UH60\_FF);

epsilon = CY\_TPP\_UH60\_FF - CY\_TPP\_old;

end

[beta\_knot\_grid, psi\_grid] = meshgrid(beta\_knot\_UH60\_FF, psi);

[beta\_1s\_grid, psi\_grid] = meshgrid(beta\_1s\_UH60\_FF, psi);

[beta\_1c\_grid, psi\_grid] = meshgrid(beta\_1c\_UH60\_FF, psi);

[theta\_knot\_grid, psi\_grid] = meshgrid(theta\_knot\_UH60\_FF, psi);

[theta\_1s\_grid, psi\_grid] = meshgrid(theta\_1s\_UH60\_FF, psi);

[theta\_1c\_grid, psi\_grid] = meshgrid(theta\_1c\_UH60\_FF, psi);

theta = theta\_fn(theta\_knot\_grid, theta\_1s\_grid, theta\_1c\_grid, psi\_grid);

theta\_dot = theta\_dot\_fn(theta\_1s\_grid, theta\_1c\_grid, psi\_grid);

theta\_star = theta\_star\_fn(theta\_dot);

beta = beta\_fn(beta\_knot\_grid, beta\_1s\_grid, beta\_1c\_grid, psi\_grid);

beta\_dot = beta\_dot\_fn(beta\_1s\_grid, beta\_1c\_grid, psi\_grid);

beta\_star = beta\_star\_fn(beta\_dot);

% beta = zeros(length(mu), length(psi));

% beta\_dot = zeros(length(mu), length(psi));

% beta\_star = zeros(length(mu), length(psi));

% for i = 1: length(mu)

% for j = 1: length(psi)

% beta(i,j) = beta\_fn(beta\_knot\_UH60\_FF(i), beta\_1s\_UH60\_FF(i), beta\_1c\_UH60\_FF(i), psi(j));

% beta\_dot(i,j) = beta\_dot\_fn(beta\_1s\_UH60\_FF(i), beta\_1c\_UH60\_FF(i), psi(j));

% beta\_star(i,j) = beta\_star\_fn(beta\_dot(i, j));

% end

% end

trim\_inflow = [beta\_1c\_UH60\_FF; beta\_1s\_UH60\_FF; alpha\_shaft\_UH60\_FF; phi\_shaft\_UH60\_FF; theta\_knot\_UH60\_FF; theta\_1s\_UH60\_FF; beta\_knot\_UH60\_FF; theta\_1c\_UH60\_FF];

end

%% find\_MS\_1\_inflow

function[cn\_1] = find\_MS\_1\_inflow(mu, psi, nu, alpha, CT\_constant\_T)

%% constants

%epsilon = 1;

cn\_even = @(n, nu, alpha)((-1)^((n-2)/2))\*(3/4)\* ((n+nu)/((n^2)-1)) ...

\* (((1-nu)/(1+nu))^(n/2)) \* ...

(((1+sin(alpha))/(1-sin(alpha)) )^(n/2));

cn\_1 = zeros(length(nu), 6);

% lambda\_1\_i\_MS = zeros(length(psi), length(nu));

% for i = 1:length(psi)

for j = 1:length(nu)

n=5;

% epsilon = 1;

% while epsilon > 0.004

c1 = zeros(1, n+1);

for cn = 0:n

if cn == 0

c1(1) = (3/4).\*nu(j);

elseif cn == 1

c1(2) = -(3\*pi/16).\*(sqrt(1-nu(j).^2))\*((1+sin(alpha))/(1-sin(alpha))).^(1/2);

elseif (mod(cn,2)==0)

c1(cn+1) = cn\_even(n,nu(j), alpha);

elseif mod(cn,2)==1

c1(cn+1) = 0;

end

end

% lambda\_1\_i\_MS\_old = lambda\_1\_i\_MS(i,j);

% lambda\_1\_i\_MS(i,j) = ((2\*CT\_constant\_T(1))/mu) \*( c1(1) + sum(c1(2:length(c1)).\*cos((1:n).\*psi(j)))) ;

% epsilon = abs(lambda\_1\_i\_MS(i,j)-lambda\_1\_i\_MS\_old);

% n = n+1;

cn\_1(j,:) = c1;

end

% end

% end

end

%% find\_MS\_3\_inflow

function[cn\_3] = find\_MS\_3\_inflow(mu, psi, nu, alpha, CT\_constant\_T)

%% constants

%epsilon = 1;

cn\_even = @(n, nu, alpha)((-1)^((n-2)/2)).\*(15/8).\* ...

((((n+nu)/((n^2)-1))\* (((9\*(nu.^2) + ...

(n^2) - 6))/((n^2)-9)))+(3\*nu/((n^2)-9))).\*...

(((1-nu)/(1+nu))^(n/2)) \* ...

(((1+sin(alpha))/(1-sin(alpha)) )^(n/2));

lambda\_3\_i\_MS = zeros(length(psi), length(nu));

cn\_3 = zeros(length(nu), 6);

% for i = 1:length(psi)

for j = 1:length(nu)

n=5;

% epsilon = 1;

% while epsilon > 0.004

c1 = zeros(1, n+1);

for cn = 0:n

if cn == 0

c1(1) = (15/8)\*nu(j).\*(1-(nu(j)).^2);

elseif cn == 1

c1(2) = -(15\*pi/256)\*(5-(9\*(nu(j)).^2)).\*(sqrt(1-(nu(j)).^2)).\*(((1+sin(alpha))/(1-sin(alpha))).^0.5);

elseif cn == 3

c1(4) = (45\*pi/256)\*((1-((nu(j)).^2)).^(3/2)).\*(((1+sin(alpha))/(1-sin(alpha))).^1.5);

elseif (mod(cn,2)==0)

c1(cn+1) = cn\_even(n,nu(j), alpha);

elseif mod(cn,2)==1

c1(cn+1) = 0;

end

% end

% lambda\_3\_i\_MS\_old = lambda\_3\_i\_MS(i,j);

% lambda\_3\_i\_MS(i,j) = ((2.\*CT\_constant\_T(1))/mu).\*( c1(1) + sum(c1(2:length(c1)).\*cos((1:n).\*psi(j)))) ;

% epsilon = abs(lambda\_3\_i\_MS(i,j)-lambda\_3\_i\_MS\_old);

% n = n+1;

% end

cn\_3(j,:) = c1;

end

end

end

%% find\_MS\_real\_inflow

function[w,lambda\_MS\_total] = find\_MS\_real\_inflow(cn\_1, cn\_3, w, psi,nu, CT\_constant\_T, mu)

%% Adding padding of zeros

length\_cn1 = length(cn\_1);

length\_cn3 = length(cn\_3);

padding = [];

if length\_cn1 > length\_cn3

padding = zeros(1, length\_cn1 - length\_cn3);

cn\_3 = [cn\_3 padding];

elseif length\_cn3 > length\_cn1

padding = zeros(1, length\_cn3 - length\_cn1);

cn\_1 = [cn1 padding];

end

%% defining functions

cn\_fn = @(cn\_1, cn\_3, range\_w) (range\_w.\*cn\_1) + ((1-range\_w).\*cn\_3);

cn\_total\_fn = @(cn) sum(cn);

lambda\_MS\_total = zeros(length(psi), length(nu));

w\_fn = @(min\_i) 0.3 + min\_i.\*(0.4/100);

epsilon = 1;

while epsilon > 0.004

lambda\_MS\_old = lambda\_MS\_total;

for range\_w = w:0.0001:0.7

cn = cn\_fn(cn\_1, cn\_3, range\_w);

cn\_total = cn\_total\_fn(cn)/length(range\_w);

[val, min\_i] = min(cn\_total);

w = w\_fn(min\_i);

end

for i = 1:length(psi)

lambda\_MS\_total(i,:) = ((2\*CT\_constant\_T(1))./mu).\*( cn(:,1) + ...

sum(cn(:,2:size(cn, 2))...

.\*cos((1:(size(cn, 2)-1))...

.\*psi(i)), 2)) ;

end

epsilon = abs(lambda\_MS\_total-lambda\_MS\_old);

end

end

%% find\_bemt function

% r: radial station

% psi: azimuth angle

% v\_inf: free stream velocity

% v\_tip: tip velocity

% alpha: angle of attack

function[u\_t, u\_p\_linear, u\_p\_MS, T\_linear\_bar, Q\_linear\_bar, T\_MS\_bar,...

Q\_MS\_bar, P\_linear\_bar, P\_MS\_bar, dL\_MS, alpha\_linear,alpha\_MS, dFx\_MS, dFz\_MS] = ...

find\_bemt(r, psi, mu, beta, beta\_star,v\_inf, lambda\_TPP\_UH60\_FF,...

Cl\_alpha,lambda\_MS\_total, theta, rho, c, R, cd0, N\_psi, dr, Nb, omega, v\_tip)

% mu is a scalar

% lambda\_TPP\_UH\_60\_FF is a scalar

% want lambda\_1\_i\_MS to be length(r) x length(psi)

%% constants

xpr = r.\*R;

%% defining functions

u\_t\_fn = @(r,psi) v\_tip\*(r + mu.\*sin(psi));

u\_p\_linear\_fn = @(r, psi, beta, beta\_star) v\_tip\*(lambda\_TPP\_UH60\_FF + ...

r.\*beta\_star + mu.\*beta.\*cos(psi));

u\_p\_MS\_fn = @(r, psi, beta, beta\_star) v\_tip\*(lambda\_MS\_total + ...

r.\*beta\_star...

+ mu.\*beta...

.\*cos(psi));

phi\_linear\_fn = @(u\_t\_linear, u\_p\_linear) u\_p\_linear./u\_t\_linear;

phi\_MS\_fn = @(u\_p\_MS, u\_t\_MS) u\_p\_MS./u\_t\_MS;

alpha\_linear\_fn = @(u\_t\_linear, u\_p\_linear) theta - u\_p\_linear./u\_t\_linear;

alpha\_MS\_fn = @(u\_p\_MS, u\_t\_MS) theta - u\_p\_MS./u\_t\_MS;

Cl\_linear\_fn = @(alpha\_linear) Cl\_alpha.\*alpha\_linear;

Cl\_MS\_fn = @(alpha\_MS) Cl\_alpha.\*alpha\_MS;

dL\_linear\_fn = @(u\_t, Cl\_linear) 0.5.\*rho.\*(u\_t.^2).\*c.\*dr.\*Cl\_linear;

dL\_MS\_fn = @(u\_t, Cl\_MS) 0.5.\*rho.\*(u\_t.^2).\*c.\*dr.\*Cl\_MS;

dD\_fn = @(u\_t) 0.5.\*rho.\*(u\_t.^2).\*c.\*dr.\*cd0; % is it cd0 NOT SURE

dFz\_linear\_fn = @(dL\_linear, phi\_linear, dD) dL\_linear.\*cos(phi\_linear) - dD.\*sin(phi\_linear);

dFz\_MS\_fn = @(dL\_MS, phi\_MS, dD) dL\_MS.\*cos(phi\_MS) - dD.\*sin(phi\_MS);

dFx\_linear\_fn = @(dL\_linear, phi\_linear, dD) dL\_linear.\*sin(phi\_linear) + dD.\*cos(phi\_linear);

dFx\_MS\_fn = @(dL\_MS, phi\_MS, dD) dL\_MS.\*sin(phi\_MS) + dD.\*cos(phi\_MS);

T\_linear\_psi\_fn = @(psi, dFz\_linear) sum(dFz\_linear, 2);

T\_linear\_bar\_fn = @(T\_linear\_psi) (Nb/N\_psi)\*sum(T\_linear\_psi);

T\_MS\_psi\_fn = @(psi, dFz\_MS) sum(dFz\_MS);

T\_MS\_bar\_fn = @(T\_MS\_psi) (Nb/N\_psi)\*sum(T\_MS\_psi);

Q\_linear\_psi\_fn = @(r, psi, dFx\_linear) sum(R\*r.\*dFx\_linear, 1);

Q\_linear\_bar\_fn = @(Q\_linear\_psi) (Nb/N\_psi)\*sum(Q\_linear\_psi);

Q\_MS\_psi\_fn = @(r, psi, dFx\_MS\_fn) (sum(R\*r.\*dFx\_MS\_fn));

Q\_MS\_bar\_fn = @(Q\_MS\_psi) (Nb/N\_psi)\*sum(Q\_MS\_psi);

P\_linear\_bar\_fn = @(Q\_linear\_bar) Q\_linear\_bar\*omega;

P\_MS\_bar\_fn = @(Q\_MS\_bar) abs(Q\_MS\_bar)\*omega;

%%

[r\_bemt\_grid, beta\_grid] = meshgrid(r, beta);

[r\_bemt\_grid, beta\_star\_grid] = meshgrid(r, beta\_star);

[r\_bemt\_grid, psi\_bemt\_grid] = meshgrid(r, psi);

u\_t = u\_t\_fn(r\_bemt\_grid,psi\_bemt\_grid);

u\_p\_linear = u\_p\_linear\_fn(r\_bemt\_grid,psi\_bemt\_grid,beta\_grid, beta\_star\_grid);

u\_p\_MS = u\_p\_MS\_fn(r\_bemt\_grid,psi\_bemt\_grid,beta\_grid, beta\_star\_grid);

phi\_linear = phi\_linear\_fn(u\_t, u\_p\_linear);

phi\_MS = phi\_MS\_fn(u\_p\_MS, u\_t);

alpha\_linear = alpha\_linear\_fn(u\_t, u\_p\_linear);

alpha\_MS = alpha\_MS\_fn(u\_p\_MS, u\_t);

Cl\_linear = Cl\_linear\_fn(alpha\_linear);

Cl\_MS = Cl\_MS\_fn(alpha\_MS);

dL\_linear = dL\_linear\_fn(u\_t, Cl\_linear);

dL\_MS = dL\_MS\_fn(u\_t, Cl\_MS);

dD = dD\_fn(u\_t);

dFz\_linear = dFz\_linear\_fn(dL\_linear, phi\_linear,dD);

dFz\_MS = dFz\_MS\_fn(dL\_MS, phi\_MS, dD);

dFx\_linear = dFx\_linear\_fn(dL\_linear, phi\_linear, dD);

dFx\_MS = dFx\_MS\_fn(dL\_MS, phi\_MS, dD);

T\_linear\_psi = T\_linear\_psi\_fn(psi, dFz\_linear);

T\_linear\_bar = T\_linear\_bar\_fn(T\_linear\_psi);

T\_MS\_psi = T\_linear\_psi\_fn(psi, dFz\_linear);

T\_MS\_bar = T\_linear\_bar\_fn(T\_MS\_psi);

Q\_linear\_psi = Q\_linear\_psi\_fn(r, psi, dFx\_linear);

Q\_linear\_bar = Q\_linear\_bar\_fn(Q\_linear\_psi);

Q\_MS\_psi = Q\_MS\_psi\_fn(r, psi, dFx\_MS);

Q\_MS\_bar = Q\_MS\_bar\_fn(Q\_MS\_psi);

P\_linear\_bar = P\_linear\_bar\_fn(Q\_linear\_bar);

P\_MS\_bar = P\_MS\_bar\_fn(Q\_MS\_bar);

%% find\_constant\_circulation

function[constant\_circulation,constant\_circulation\_bar] = find\_constant\_circulation(rho, omega, R, Nb, T\_MS\_bar, v\_tip, A)

%% constants

T = T\_MS\_bar;

CT = T/(0.5\*rho\*A\*(v\_tip)^2);

%% defining functions

constant\_circulation\_fn = @(T)(2\*T) / (Nb\*rho\*omega\*R^2);

constant\_circulation\_bar\_fn = @(CT) 2\*pi\*CT/Nb;

%%

constant\_circulation = constant\_circulation\_fn(T);

constant\_circulation\_bar = constant\_circulation\_bar\_fn(CT);

%% find\_prescribed\_wake

function[xv, yv, zv, mu\_z, mu\_x, E, r\_wake] =...

find\_prescribed\_wake(lambda\_TPP\_linear\_it, mu, alpha, v\_tip,...

r, psi, psi\_w, psi\_b)

%% constants

lambda\_io = lambda\_TPP\_linear\_it;

X = atan(-lambda\_io./mu);

E = X/2;

mu\_x = mu.\*cos(alpha)./v\_tip;

mu\_z = mu.\*sin(alpha)./v\_tip;

rv = 1;

%% defining functions

zv1\_fn = @(lambda\_io, mu\_z, psi\_b, psi\_w, mu\_x, E, y0, x0) mu\_z\*psi\_w - ...

lambda\_io.\*psi\_w.\*(1+(E.\*(x0+(0.5.\*mu\_x.\*psi\_w)-(abs(y0).^3))));

zv2\_fn = @(lambda\_io, mu\_z, psi\_b, psi\_w, mu\_x, E, y0, x0) mu\_z.\*psi\_w - ...

((2.\*lambda\_io.\*psi\_w).\*(1-(E.\*(abs(y0).^3))));

zv3\_fn = @(lambda\_io, mu\_z, psi\_b, psi\_w, mu\_x, E, y0, x0) mu\_z.\*psi\_w - ...

((2.\*lambda\_io.\*x0./mu\_x).\*(1-(E.\*(abs(y0).^3))));

%%

%%zv = %% initialize

zv1\_count = 0;

zv2\_count = 0;

zv3\_count = 0;

xv = zeros(length(psi\_w), length(psi\_b));

zv = zeros(length(psi\_w), length(psi\_b));

for n = 1:length(psi\_w)

for m = 1:length(psi\_b)

x0(n, m) = rv.\*cos(psi\_b(m)-psi\_w(n));

y0(n, m) = rv.\*sin(psi\_b(m)-psi\_w(n));

yv(n, m) = y0(n, m);

xv(n, m) = x0(n, m) + mu\_x\*psi\_w(n);

if xv(n, m) < -cos(psi\_b(m)-psi\_w(n))

zv(n, m) = zv1\_fn(lambda\_io, mu\_z, psi\_b(m), psi\_w(n), mu\_x, E, yv(n, m), xv(n, m));

zv1\_count = zv1\_count + 1;

elseif xv(n, m) > cos(psi\_b(m)-psi\_w(n))

zv(n, m) = zv2\_fn(lambda\_io, mu\_z, psi\_b(m), psi\_w(n), mu\_x, E, yv(n, m), xv(n, m));

zv2\_count = zv2\_count + 1;

else

zv(n, m) = zv3\_fn(lambda\_io, mu\_z, psi\_b(m), psi\_w(n), mu\_x, E, yv(n, m), xv(n, m));

zv3\_count = zv3\_count + 1;

end

end

end

zv(1, :) = 0;

% disp(size(xv))

% disp(size(yv))

% disp(size(zv))

% disp(zv1\_count)

% disp(zv2\_count)

% disp(zv3\_count)

r\_wake = cat(3, xv, yv, zv);

%% find\_biot\_savart

function[V, lambda\_wake] = find\_biot\_savart(circulation\_v,...

v\_tip, r\_wake, psi\_w, psi\_b, r\_rotor, r)

%% constants

% rv = 1;

% lv = 30\*pi\*rv;

% %dlv = rv\*(pi/12)/(2\*pi);

r\_biot = @(r\_wake) r\_wake;

lambda\_wake = zeros(length(r),length(psi\_b),3);

%% defining functions

%%

for PB = 1:length(psi\_b)

for R\_R = 1:length(r)

V = zeros(length(psi\_w),3);

for PW = 1:length(psi\_w)-1

r1 = r\_wake(PW,PB,:) - (r\_rotor(R\_R,PB,:));

r2 = r\_wake(PW+1,PB,:) - (r\_rotor(R\_R,PB,:));

lv = r2 - r1;

% disp("display 3")

h = abs(cross(r1, lv));

% disp("display 4")

V(PW, :) = (circulation\_v/(4.\*pi.\*h)).\*...

(dot(lv, r1-r2).\*cross(r1, r2));

end

V\_total = sum(V);

lambda\_wake(R\_R, PB, :) = V\_total./v\_tip;

end

end

%% find\_linear\_inflow\_power function

% sigma: blade solidity

% f:

% A: rotor disk area

% L\_tr: moment arm tail rotor

% R: total radius

function[linear\_inflow\_power] = find\_linear\_inflow\_power(mu, CT\_constant\_T, sigma, f, A,k)

%% constants

lambda\_climb\_FF = 0;

cd0 = 0.001;

%% defining functions

% c(r) is constant because assume rectangular blade

% blade elements are all assumed as the same with the cd0 being for airfoil SC1095

CP\_L\_H\_60\_lambda = @(CT\_constant\_T) (k.\*(CT\_constant\_T.^(3/2))/sqrt(2)) + (sigma.\*cd0./8);

CP\_L\_FF\_60\_lambda = @(mu, CT\_constant\_T)...

((k\*CT\_constant\_T.^2)./(2.\*mu)) + ((sigma.\*cd0./8)\*(1+(4.6\*mu.^2)))...

+ ((1./2).\*(f./A).\*(mu.^3)) + (lambda\_climb\_FF.\*CT\_constant\_T); %%% POWER EQUATION IS RIGHT BUT GRAPH IS WRONG

%YF\_W = @(CP\_UH60\_FF) CP\_UH60\_FF.\*R./CT\_constant\_T./L\_tr;

%% Finding Linear Coefficient of Power Required CP\_UH60\_FF based on lambda\_TPP\_UH60\_FF

CP\_UH60\_H = CP\_L\_H\_60\_lambda(CT\_constant\_T(1));

CP\_UH60\_FF = CP\_L\_FF\_60\_lambda(mu(2:223), CT\_constant\_T(2:223));

CP\_UH60\_FF = [CP\_UH60\_H CP\_UH60\_FF];

%YF\_W\_FF = YF\_W(CP\_UH60\_FF);

linear\_inflow\_power = CP\_UH60\_FF;

end

%% find\_beddoes\_mdpt

function [Cl, L, phi\_wp\_jones] = find\_beddoes\_mdpt(A, v, rho, alpha, theta)

% approximate solutions from W.P. Jones

%% constants

A1 = 0.165;

b1 = 0.0455;

A2 = 0.335;

b2 = 0.3;

%% zeros

phi\_wp\_jones = zeros(1, 100);

X = 0;

Y = 0;

ds = 1;

for s = 2:length(alpha)

X\_old = X;

Y\_old = Y;

d\_alpha = alpha(s) - alpha(s-1);

X = (X\_old\*exp(-b1\*ds)) + (A1\*d\_alpha\*exp(-b1\*ds));

Y = (Y\_old\*exp(-b2\*ds)) + (A2\*d\_alpha\*exp(-b2\*ds));

alpha\_eq = alpha - X - Y;

cl\_eq = 2\*pi\*alpha\_eq;

epsilon\_mdpt = 2 - ((b1\*ds\*exp(-b1\*ds))/(1-exp(-b1\*ds)))...

- ((b2\*ds\*exp(-b2\*ds))/(1-exp(-b2\*ds)));

phi\_wp\_jones(s) = 1 - (A1\*exp(-b1\*s)) - (A2\*exp(-b2\*s));

end

Cl = cl\_eq(1, :);

L = 0.5.\*rho.\*(v.^2).\*A.\*Cl;

%% find\_harmonics

function[theta, theta\_root, theta\_all] = find\_harmonics(beta, beta\_dot,...

beta\_star, theta, theta\_dot, theta\_star, trim\_inflow, u\_p, u\_t, psi);

theta\_old = 0;

psi\_n = 1:361;

for n = 1:5

theta\_knot = trim\_inflow(5, :);

theta\_nc = trim\_inflow(8, :);

theta\_ns = trim\_inflow(6, :);

theta\_sum(psi\_n) = (theta\_nc.\*cos(n.\*psi\_n))...

+ (theta\_ns.\*sin(n.\*psi\_n))...

+ theta\_old;

theta\_root(psi\_n) = theta\_knot + theta\_sum(psi\_n);

theta\_old = theta\_sum(psi\_n);

theta\_all(n, psi\_n) = theta\_root(psi\_n);

end

theta = theta\_root;

end

## References:

1. Beddoes, T.S. “A Wake Model for High Resolution Airloads.” 2nd International Conference on Basic Rotorcraft Research, 1985.
2. Leishman, J. Gordon. *Principles of Helicopter Aerodynamics*. Cambridge University Press, 2017.
3. Shinoda, Patrick M., et al. “Rotor Performance of a UH‐60 Rotor System in the NASA Ames 80‐ by 120‐Foot Wind Tunnel.” *Latest TOC RSS*, Vertical Flight Society, 1 Oct. 2004, https://www.ingentaconnect.com/content/ahs/jahs/2004/00000049/00000004/art00003.
4. Totah, Joseph. *A Critical Assessment of UH-60 Main Rotor Blade Airfoil Data - NASA*. https://ntrs.nasa.gov/api/citations/19940027557/downloads/19940027557.pdf.