Table of Contents

ASEN 3111 - Computational Assignment 04 - Main

This script performs three tasks. First, it uses the function PLLT to calculate the lift and induced drag of a given finite wing. Second, it analyzes how the error of this PLLT function corresponds with the number of terms. Third, the PLLT function is used to show the relationship between span efficiency factor and the geometry of the wing planform (Aspect Ratio and Taper Ratio)

```
Author: Grant Norman
Collaborators: Jake McGrath, Daniel Crook
Date: 11/21/2019
```

```
clc; clear all; close all;

% Timing for TA grading
tic;
```

Problem #1

Create an implementation of Prandtl Lifting Line Theory

```
fprintf('Problem #1: PLLT Function:\n');
help PLLT;
```

```
Problem #1: PLLT Function:
 PLLT Prandtl Lifting Line Theory implementation for a general lift
  distribution, of a trapezoidal wing with linear geometric and
 aerodynamic
  twist


    Inputs:
        - b = span length of the wing
        - a0_t = 2D lift slope of the airfoil at the tip of the wing
        - a0_r = 2D lift slope of the airfoil at the root of the wing
        - c_t = chord length of the airfoil at the tip of the wing
        - c_r = chord length of the airfoil at the root of the wing
        - aero_t = 0 Lift angle of the airfoil at the tip of the wing,
 in
        radians, corresponding to aerodynamic twist
        - aero_r = 0 Lift angle of the airfoil at the root of the
 wing, in
        radians, corresponding to aerodynamic twist
```

```
        - geo_t = geometric angle of attack of the airfoil at the tip
of the
        wing, in radians, corresponding to geometric twist
        - geo_r = geometric angle of attack of the airfoil at the root
of
        the wing, in radians, corresponding to geometric twist
        - N = number of odd terms to use in the Fourier expansion to
        approximate the circulation

   Outputs:
        e = span efficiency factor, 1/(1+delta), where 1 is a
perfectly
        elliptical lift distribution
        c_L = finite wing lift coefficient
        c_Di = finite wing induced drag coefficient


   Author: Grant Norman
   Collaborators: Jake McGrath
   Date: 11/21/2019
```

Problem #2

Analyze the given wing, and perform error analysis

```
b = 100; % ft
c_r = 15; % ft
c_t = 5; % ft

N_vortex_panels = 1000;
% Airfoil at root: NACA 2412
[a0_r, aero_r] = NACA_cl_Props(2412,N_vortex_panels);

% Airfoil at tip: NACA 0012
[a0_t, aero_t] = NACA_cl_Props(0012,N_vortex_panels);

% Geometric Angle of Attack
geo_r = deg2rad(5); % 5 degrees in radians
geo_t = deg2rad(0); % 0 degrees in radians

% Number of Fourier terms to find
N = 1500;

[e,c_L,c_Di] = PLLT(b,a0_t,a0_r,c_t,c_r,aero_t,aero_r,geo_t,geo_r,N);
fprintf('\n\n\n\n\nProblem #2: c_L and c_Di\n');
fprintf('\nFor the given finite wing, c_L = %f, c_Di = %f, and e = %f
\n',...
    c_L, c_Di, e);

% Conditions at Sea Level
rho = 0.002377; % slug/ft^3
V_inf = 150*5280/3600; % ft/s
```

```
S = (c_r+c_t)/2 * b;
L = rho*V_inf^2/2 * S * c_L;
Di = rho*V_inf^2/2 * S * c_Di;
fprintf('\nAt 150 mi/hr, Lift = %3.3f lbf, Induced Drag = %3.3f lbf
\n',...
    L, Di);

nTerms = [0; 0; 0];
relErrors = [0.05; 0.01; 0.001];
for i=2:1:1000
    [e_i,c_L_i,c_Di_i] =
 PLLT(b,a0_t,a0_r,c_t,c_r,aero_t,aero_r,geo_t,geo_r,i);
    c_D_error = abs(c_Di_i - c_Di)/c_Di;
    c_L_error = abs(c_L_i - c_L)/c_L;
    error = max(c_L_error, c_D_error);
    if error < relErrors(1)
        if error < relErrors(2)
            if error < relErrors(3)
                nTerms(3) = i;
            end
            if  ~nTerms(2)
                nTerms(2) = i;
            end
        end
        if  ~nTerms(1)
            nTerms(1) = i;
        end
    end

    if all(nTerms)
        break;
    end
end

q2Tab = table(relErrors,nTerms,'VariableNames',{'Relative Error',...
    'Number of Odd Terms in Circulation Series Expansion'});
disp(q2Tab);
```

Problem #3

PLLT for e vs. ct/cr for a thin wing

```
AR = [4; 6; 8; 10;];

% For a thin wing
a0 = 2*pi;

n = 30;
% Taper ratio
tr = linspace(0,1,n);
% Span efficiency factors
e_mat = zeros(length(tr),length(AR));
```

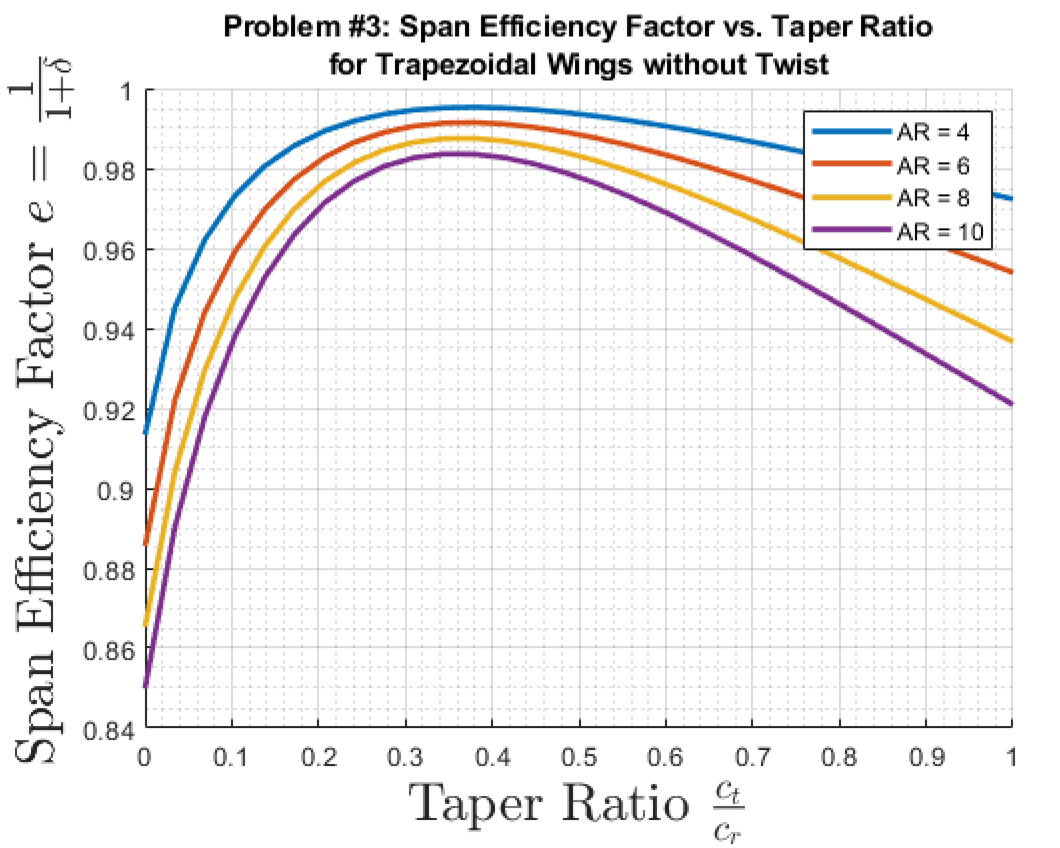
```
for j=1:length(AR)
    AR_i = AR(j);
    for i=1:length(tr)
        % Trapezoidal Wing
        c_r = 1;
        c_t = c_r*tr(i);
        b = AR_i*(c_t+c_r)/2;
        e_mat(i,j) = PLLT(b,a0,a0,c_t,c_r,0.1,0.1,0.2,0.2,N);
    end
end

deltas = 1./(e_mat) - 1;

figure; hold on;
for j=1:length(AR)
    plot(tr,e_mat(:,j),'Linewidth',2,'DisplayName',...
        ['AR = ',num2str(AR(j))]);
end
grid on; grid minor;
xlabel('Taper Ratio $\frac{c_t}
{c_r}$','Interpreter','latex','Fontsize',20)
ylabel('Span Efficiency Factor $e = \frac{1}{1+
\delta}$','Interpreter','latex','Fontsize',20)
legend show;
title({'Problem #3: Span Efficiency Factor vs. Taper Ratio','for
 Trapezoidal Wings without Twist'});
hold off;

toc;
```

Elapsed time is 26.984090 seconds.



Functions

```
function [c_l, Cp, X] = Vortex_Panel(x,y,alpha,tle,plt)
    %Vortex_Panel Uses the vortex panel method to analyze a closed
 body, based
    %on code provided in Kuethe and Chow
    %
    %
    %   Inputs:
    %       - x = x-coordinates of body, with the first and last
 points at the
    %       trailing edge
    %       - y = y-coordinates of body, with the first and last
 points at the
    %       trailing edge
    %       - alpha = angle of attack of the body's centerline
    %       - tle = title corresponding to Coefficient plot
    %       - plt = boolean as to whether or not plot X vs. Cp
 automatically
    %
    %   Outputs:
    %       - c_l = coefficient of lift of the body
    %       - Cp = Pressure Coefficient vector
    %       - X = X locations for Cp vector
    %
```

```matlab
%
%   Author: Grant Norman
%   Collaborators: Jake McGrath
%   Date: 11/7/2019

XB = x;
YB = y;

MP1 = length(XB);
M = MP1-1;

X = zeros(M,1);
Y = zeros(M,1);
S = zeros(M,1);
Theta = zeros(M,1);

for i=1:M
    X(i) = 0.5*(XB(i) + XB(i+1));
    Y(i) = 0.5*(YB(i) + YB(i+1));
    S(i) = sqrt((XB(i+1)-XB(i))^2 + (YB(i+1)-YB(i))^2);
    Theta(i) = atan2((YB(i+1)-YB(i)), (XB(i+1)-XB(i)));
end
sine = sin(Theta);
cosine = cos(Theta);
rhs = sin(Theta-alpha);

CN1 = zeros(M);
CN2 = zeros(M);
CT1 = zeros(M);
CT2 = zeros(M);

for i=1:M
    for j=1:M
        if i==j
            CN1(i,j) = -1;
            CN2(i,j) = 1;
            CT1(i,j) = 0.5*pi;
            CT2(i,j) = 0.5*pi;
        else
            A = -(X(i)-XB(j))*cosine(j) - (Y(i)-YB(j))*sine(j);
            B = (X(i)-XB(j))^2 + (Y(i)-YB(j))^2;
            C = sin(Theta(i)-Theta(j));
            D = cos(Theta(i)-Theta(j));
            E = (X(i)-XB(j))*sine(j) - (Y(i)-YB(j))*cosine(j);
            F = log(1+S(j)*(S(j)+2*A)/B);
            G = atan2(E*S(j), B+A*S(j));
            P = (X(i)-XB(j)) * sin(Theta(i)-2*Theta(j))...
                + (Y(i)-YB(j)) * cos(Theta(i)-2*Theta(j));
            Q = (X(i)-XB(j)) * cos(Theta(i)-2*Theta(j))...
                - (Y(i)-YB(j)) * sin(Theta(i)-2*Theta(j));
            CN2(i,j) = D + 0.5*Q*F/S(j) - (A*C+D*E)*G/S(j);
            CN1(i,j) = 0.5*D*F + C*G - CN2(i,j);
            CT2(i,j) = C + 0.5*P*F/S(j) + (A*D-C*E)*G/S(j);
            CT1(i,j) = 0.5*C*F - D*G - CT2(i,j);
```

```
            end
        end
    end

    AN = zeros(M);
    AT = zeros(M);

    for i=1:M
        AN(i,1) = CN1(i,1);
        AN(i,MP1) = CN2(i,M);
        AT(i,1) = CT1(i,1);
        AT(i,MP1) = CT2(i,M);
        for j=2:M
            AN(i,j) = CN1(i,j) + CN2(i,j-1);
            AT(i,j) = CT1(i,j) + CT2(i,j-1);
        end
        AN(MP1,1) = 1;
        AN(MP1,MP1) = 1;
        for j=2:M
            AN(MP1,j) = 0;
        end
        rhs(MP1) = 0;
    end

    % Solve Linear System:
    Gamma = AN\rhs;

    V = zeros(M,1);
    Cp = zeros(M,1);

    for i=1:M
        V(i) = cos(Theta(i) - alpha);
        for j=1:MP1
            V(i) = V(i) + AT(i,j)*Gamma(j);
            Cp(i) = 1.0 - V(i)^2;
        end
    end

    if (plt==1)
        f = figure();
        ax1 = gca;
        plotCp(X,Cp,alpha,tle,f,ax1,1);
    end

    % From Kutta–Joukowski
    c_l = 2*sum(S.*V);
end

function [e,c_L,c_Di] =
 PLLT(b,a0_t,a0_r,c_t,c_r,aero_t,aero_r,geo_t,geo_r,N)
    %PLLT Prandtl Lifting Line Theory implementation for a general
 lift
    % distribution, of a trapezoidal wing with linear geometric and
 aerodynamic
```

```
    % twist
    %
    %
    %   Inputs:
    %       - b = span length of the wing
    %       - a0_t = 2D lift slope of the airfoil at the tip of the
wing
    %       - a0_r = 2D lift slope of the airfoil at the root of the
wing
    %       - c_t = chord length of the airfoil at the tip of the wing
    %       - c_r = chord length of the airfoil at the root of the
wing
    %       - aero_t = 0 Lift angle of the airfoil at the tip of the
wing, in
    %       radians, corresponding to aerodynamic twist
    %       - aero_r = 0 Lift angle of the airfoil at the root of the
wing, in
    %       radians, corresponding to aerodynamic twist
    %       - geo_t = geometric angle of attack of the airfoil at the
tip of the
    %       wing, in radians, corresponding to geometric twist
    %       - geo_r = geometric angle of attack of the airfoil at the
root of
    %       the wing, in radians, corresponding to geometric twist
    %       - N = number of odd terms to use in the Fourier expansion
to
    %       approximate the circulation
    %
    %   Outputs:
    %       e = span efficiency factor, 1/(1+delta), where 1 is a
perfectly
    %       elliptical lift distribution
    %       c_L = finite wing lift coefficient
    %       c_Di = finite wing induced drag coefficient
    %
    %
    %   Author: Grant Norman
    %   Collaborators: Jake McGrath
    %   Date: 11/21/2019

    N = 2*N;

    n = 1:2:N;
    N_odd = length(n);
    i=1:N_odd;
    thetas = (i)*pi/(2*N_odd);

    A1 = zeros(N_odd);
    A2 = zeros(N_odd);
    b_vec = zeros(N_odd,1);

    for i=1:N_odd
        a0 = evalAtTheta(a0_r, a0_t, thetas(i), b);
        c = evalAtTheta(c_r, c_t, thetas(i), b);
```

```
        geo = evalAtTheta(geo_r, geo_t, thetas(i), b);
        aero = evalAtTheta(aero_r, aero_t, thetas(i), b);

        for j=1:N_odd
            A1(i,j) = 1/(a0 * c)*(sin(n(j)*thetas(i)));
            A2(i,j) = n(j) * (sin(n(j)*thetas(i))/sin(thetas(i)));
        end
        b_vec(i) = geo - aero;
    end
    A1 = A1*4*b;
    A = (A1+A2);
    four_coeffs = A\b_vec;

    delta = 0;
    for i=2:length(four_coeffs)
        delta = delta + n(i)*(four_coeffs(i)/four_coeffs(1))^2;
    end

    e = 1/(1+delta);
    AR = b^2 / ((c_r+c_t)*b/2);
    c_L = four_coeffs(1)*pi*AR;
    c_Di = c_L^2 * (1+delta) / (pi*AR);
end

function v = evalAtTheta(v_r, v_t, theta, b)
    %evalAtTheta Evaluates wing properties at a given theta, for use
 within
    % Prandtl Lifting Line Theory implementation.
    %
    %   Inputs:
    %       - v_r = Values of properties at the root of the wing
 (vector or
    %       scalar)
    %       - v_t = Values of properties at the tip of the wing (same
 dimension
    %       as v_r)
    %       - theta = transformed parameters corresponding to where on
 the wing
    %       to evaluate the properties
    %       - b = span length of the wing
    %
    %   Author: Grant Norman
    %   Collaborators:
    %   Date: 11/21/2019

    v = v_r - 2*(v_r-v_t)/(b)*abs(-b/2*cos(theta));
end

function [a0,aL0] = NACA_cl_Props(NACA_num,N)
    %NACA_CL_PROPS uses the tools from Computational Assignment 3, the
 vortex
    % panel method, to calculate the 0-Lift angle, and the coefficient
    % of lift slope
    %
```

```
    %
    %   Inputs:
    %       - NACA_num = 4 digit number, mpth with the following
 properties:
    %           - m = maximum camber in hundredths of chord
    %           - p = location of max. camber from the Leading Edge,
 in tenths
    %           of chord
    %           - th = 2 digits corresponding to max thickness, in
 hundredths of
    %           chord
    %       - N = number of requested unique points on the airfoil
    %
    %   Outputs:
    %       - a0 = lift coefficient slope in 1/radians
    %       - aL0 = 0 lift angle of attack in radians
    %
    %
    %   Author: Grant Norman
    %   Collaborators:
    %   Date: 11/16/2019

    c=1;
    tle = '';
    plt = 0;
    [x,y,~] = NACA_Airfoils(NACA_num,c,N);

    alpha1 = deg2rad(5);
    alpha2 = deg2rad(-5);

    c_l1 = Vortex_Panel(x,y,alpha1,tle,plt);
    c_l2 = Vortex_Panel(x,y,alpha2,tle,plt);

    a0 = (c_l1 - c_l2)/(alpha1 - alpha2);
    aL0 = alpha1 - c_l1/a0;
end

function [x,y,al0] = NACA_Airfoils(NACA_num,c,N)
    %NACA_Airfoils returns the coordinates of the designated NACA
 airfoil's
    %surface
    %
    %
    %   Inputs:
    %       - NACA_num = 4 digit number, mpth with the following
 properties:
    %           - m = maximum camber in hundredths of chord
    %           - p = location of max. camber from the Leading Edge,
 in tenths
    %           of chord
    %           - th = 2 digits corresponding to max thickness, in
 hundredths of
    %           chord
    %       - c = chord length
```

```
    %       - N = number of requested unique points on the airfoil
    %
    %   Outputs:
    %       - x = N+1 x coordinates describing the airfoils surface,
 beginning
    %       at the trailing edge and going to the leading edge, and
 back again,
    %       with the first and last points at the trailing edge
    %       - y = N+1 y coordinates describing the airfoils surface,
    %       corresponding with the x output; i.e. (x,y) give the
 coordinates of
    %       the airfoils surface
    %       - al0 = zero lift angle from thin airfoil theory
    %
    %
    %   Author: Grant Norman
    %   Collaborators: Daniel Crook
    %   Date: 11/7/2019

    % Get airfoil properties from NACA number.
    m = floor(NACA_num/1000) / 100;
    p = mod(floor(NACA_num/100),10) / 10;
    t = mod(NACA_num,100) / 100;

    % For redundant points at the trailing edge
    N = N+2;
    x = linspace(0,c,N/2)';

    % Thickness
    yt = t/0.2*c* (0.2969*sqrt(x./c) - 0.1260*(x./c) - 0.3516*(x./
c).^2 + 0.2843*(x./c).^3 -0.1036*(x./c).^4);

    % Camber
    yc = zeros(length(x),1);
    for i=1:length(x)
        if 0<=x(i) && x(i)<=p*c
            yc(i) = m * x(i)/(p^2) * (2*p - x(i)/c);
        else
            if p*c<=x(i) && x(i)<=c
                yc(i) = m * (c - x(i))/(1 - p)^2 * (1 + x(i)/c - 2*p);
            else
                disp('Error in NACA_Airfoils.m')
            end
        end
    end

    dx = c/N;
    dyc = yc(2:end)-yc(1:end-1);
    xi = atan(dyc/dx);

    xu = x(1:end-1)-yt(1:end-1).*sin(xi);
    xl = x(1:end-1)+yt(1:end-1).*sin(xi);

    yu = yc(1:end-1)+yt(1:end-1).*cos(xi);
```

```
    yl = yc(1:end-1)-yt(1:end-1).*cos(xi);

    % If it's a symmetric airfoil, do not consider the camber, which
 includes
    % factors that cause Nans.
    if (m==0 && p==0)
        xu = x(1:end-1);
        xl = x(1:end-1);

        yu = 0+yt(1:end-1);
        yl = 0-yt(1:end-1);
        al0 = 0;
    else
        theta0 = linspace(0.001,pi-0.001,length(dyc))';
%         intgrl = trapz(dyc*2./(c*sin(theta0)) .* (cos(theta0)-1));
        dtheta0 = theta0(2)-theta0(1);
        intgrl = trapz((dyc/dx).*(cos(theta0)-1)*dtheta0);
        al0 = rad2deg(intgrl/-pi);
    end

    % Organize so that we start at the TE, go counterclockwise, and
 end at the
    % trailing edge.
    x = [c; flip(xl(2:end)); xu; c];
    y = [0; flip(yl(2:end)); yu; 0];
end
```

```
Problem #2: c_L and c_Di

For the given finite wing, c_L = 0.409067, c_Di = 0.007607, and e =
 0.700207

At 150 mi/hr, Lift = 23530.939 lbf, Induced Drag = 437.580 lbf
```

Relative Error	Number of Odd Terms in Circulation Series Expansion
0.05	7
0.01	16
0.001	48

Published with MATLAB® R2019b