```matlab
% Author: Leon Tannenbaum
% Date: June 6th 2021
% Purpose: Analyze a simple swept wing, use this code as basis for transonic cruise analysis.
% Written for Aero 601 Fall 2020 Advanced Aerodynamics - 2020

% This edition of VLM code has some parallelization written into it so as to speed
% up execution time.  Alone, a single run is fairly quick.  However, when iterated
% >1000 times, integrated over time, execution can be fairly time intensive.
%_wp - with parallelization




% -- Constants -------------------------------------------------------------CONSTANTS
errors_on = 1;
sideslip = 0;
% errors on = 1 turn on, 0 off

if ~(errors_on == 0 || errors_on == 1)
    error("Error entry must = 1 or 0.");
end % if ~(errors_on == 0 || errors_on == 1)

% -- Get number of CPU cores ----------------------------------------------CPU core amount
num_cpu = feature('numcores'); % number of available CPU cores
if isempty(gcp('nocreate')) == 1 % no parallel pool is running
    parpool('local', num_cpu);
end % if isempty

% -------------------------------------------------------------------------CPU core amount



% if B_pg is = 1, then the PG transform isn't done, if <= 1, then PG transform is used.
B_pg = 1;
eps = 1.456619e-16;
Uinf  = 1;
rho    = 1; % density
gamma  = 1; % circulation
AR     = 5;
b      = 1;  % wingspan normalized
lambda = .5; % taper
N      = 64; % number of panels
c_root   = (2*b) / (AR * (1+lambda)); % length
sweep    = deg2rad(25);   % sweep  = deg2rad(25);  % 25 degrees of sweep
alfa     = deg2rad(6);     % alfa   = deg2rad(6);   % angle of attack
beta     = deg2rad(0); % beta   = deg2rad(0);   % sideslip
Sref     = (b^2)/AR;
washout = deg2rad(2); % washout  = deg2rad(2);
% -- Input Check ----------------------------------------------------------Input check
if lambda <= 0
    error(" lambda cannot be less than or equal to zero.");
end % lambda <= 0

if lambda > 1
    error("lambda cannot be greater than 1");
end % lambda > 1

if AR <=0
    error("AR can't be less than or equal to zero.");
end % AR <=0
```

```matlab
if mod(N,num_cpu) ~= 0
    error("N is NOT divisble by the # of available CPU cores, parallelization problem!!");
end % (mod(N,4) ~= 0)

if (N <= 0) || (mod(N,2) == 1) % check if N is <= 0 or is odd
    error("ERROR N can't be less than/equal to zero or odd.");
end % (N <= 0) || (mod(N,2) == 1) % check if N is <= 0 or is odd
% -------------------------------------------------------------------------Input check
len_y_pts = (N/2) + 1;     % number of points (N/2) is number of panels
% -------------------------------------------------------------------------CONSTANTS


% -- FIND Y POSITIONS AND CHORD LENGTH EDGES-------------------------------YandCHORD
y_VEC = linspace(-b/2, b/2, len_y_pts);
chord_len_edges = zeros(1, len_y_pts);
for j =1:len_y_pts
    chord_len_edges(j) = chord_x(y_VEC(j), c_root, lambda,b);
    %fprintf("chord_len_edges(%d) = %e\n",j, chord_len_edges(j));

    if isnan(chord_len_edges(j)) == 1
        error("chord_len_edges(%d) = Nan\n", j);
    end % isnan(x_lead_vort_VEC(m)) == 1
end % for j = 1:len_y_pts

% check to make sure chord_len_edges is sym about y
for m = 1:N/2
    if chord_len_edges(m) ~= chord_len_edges(len_y_pts - m+1)
        error("Non Symmetry on chord_len_edges at %d and %d!\n", m, len_y_pts - m+1);
    end
end %
%--------------------------------------------------------------------------YandCHORD

% -- GENERATE THE LEAD/TRAIL EDGE POINTS, LEAD/TRAIL VOR PTS --------------LE/TE_LE/TE_VOR
x_leading_VEC = zeros(1, len_y_pts);
x_central_VEC = zeros(1, len_y_pts);
x_trail_VEC   = zeros(1, len_y_pts);

x_lead_vort_VEC  = zeros(1, len_y_pts);
x_trail_vort_VEC = zeros(1, len_y_pts);
for k = 1:len_y_pts
    h = (c_root - chord_len_edges(k))*.5;
    if isnan(h) == 1
        error("h at (%d) = Nan\n", k, h);
    end % isnan(x_lead_vort_VEC(m)) == 1

    h = add_sweep(sweep, h, y_VEC(k));

    if isnan(h) == 1
        error("h at (%d) = Nan\n", k, h);
    end % isnan(x_lead_vort_VEC(m)) == 1

    x_leading_VEC(k) = h / B_pg;
    x_central_VEC(k) = (h + (chord_len_edges(k)*.5))/B_pg;
    x_trail_VEC  (k) = (h +  chord_len_edges(k))/B_pg;

    x_lead_vort_VEC(k)  = (h + (1/8)* chord_len_edges(k))/B_pg;
    x_trail_vort_VEC(k) = (h + (5/8)* chord_len_edges(k))/B_pg;

    if isnan(x_lead_vort_VEC(k)) == 1
        error("x_lead_vort_VEC(%d) = Nan\n", k);
    end % isnan(x_lead_vort_VEC(m)) == 1
```

```
end % for k = 1:len_y_pts
%----------------------------------------------------------------------------LE/TE_LE/TE_VOR

% -- GET CONTROL POINTS Y LOCATION IN THE CENTER OF EACH PANEL-------------Y_ctrl
y_ctrl_pts_VEC = zeros(1, (N/2));
for j = 1:(len_y_pts -1)
    y_ctrl_pts_VEC(j) = (y_VEC(j) + y_VEC(j+1))/2;
end %for i = 1:(length(Y_vec) -1)
% -------------------------------------------------------------------------Y_ctrl


% -- Freestream Vel w/ Washout --------------------------------------------Freestream/Washout
% Uinf_x = Uinf * cos(alfa);
% Uinf_y = Uinf * 0;
% Uinf_z = Uinf * sin(alfa);
% U_inf_vec = [Uinf_x Uinf_y Uinf_z];


alfa_VEC   = ones(N/2, 1) .* alfa;
wash_f_VEC = zeros(N/2, 1);
parfor m = 1:N/2
    wash_f_VEC(m) = (-2*washout / b) *( abs(y_ctrl_pts_VEC(m)) - (b/2));
end % wash_f_VEC = zeros(N/2, 1);
alfa_w_wash_VEC_n2 = alfa_VEC + wash_f_VEC; % alfa for each panel
alfa_w_wash_VEC_n2(1) = alfa; alfa_w_wash_VEC_n2(N/2) = alfa; % correct numerical computation issues
alfa_wash_VEC_N    = [alfa_w_wash_VEC_n2; alfa_w_wash_VEC_n2]; % full alfa vector

% set up matrix with freestream velocity values
U_inf_VEC = zeros(N, 3);
parfor m = 1:N
    Uinf_x = Uinf * cos(alfa_wash_VEC_N(m));
    Uinf_y = Uinf * 0;
    Uinf_z = Uinf * sin(alfa_wash_VEC_N(m));
    U_inf_VEC(m, :) = [Uinf_x Uinf_y Uinf_z];
end % m = 1:N
% -------------------------------------------------------------------------Freestream/Washout

% -- GET X CONTROL POINTS LOCATION FOR LEADING AND TRAILING EDGE PANELS ---X_ctrl
x_ctrl_lead_VEC  = zeros(1, (N/2));
x_ctrl_trail_VEC = zeros(1, (N/2));

for m = 1:(N/2)
    chord = chord_x(y_ctrl_pts_VEC(m), c_root, lambda,b);
    h = (c_root - chord) * .5;
    h = add_sweep(sweep, h, y_ctrl_pts_VEC(m));


    x_ctrl_lead_VEC(m)  = (h + (5/16)*chord)/ B_pg;
%    fprintf("chord = %e,\t m = %d,\th = %e\t x_ctrl_lead_VEC(%d) = %e\n", chord, m, h, m,  x_ctrl_lead_VEC(m));
    x_ctrl_trail_VEC(m) = (h + (13/16)*chord)/B_pg;

end % m = 1:(N/2)
% -------------------------------------------------------------------------X_ctrl

% -- Set up Z positions ---------------------------------------------------Z-ctrl/vort
z_edges_VEC      = zeros(1, len_y_pts);
z_lead_vort_VEC  = zeros(1, len_y_pts);
z_trail_vort_VEC = zeros(1, len_y_pts);
z_ctrl_lead_VEC  = zeros(1, N/2);
z_ctrl_trail_VEC = zeros(1, N/2);
% -------------------------------------------------------------------------Z-ctrl/vort

% == WARNING ==============================================================PANELS NO MORE VECTORS
```

```
% == HERE VECTORS ARE DONE AND PANELS ARE CREATED ========================PANELS NO MORE VECTORS
% =======================================================================PANELS NO MORE VECTORS


% -- Generate Vortex and Control Point Vectors ---------------------------Generate Vor / Control Vectors
lead_vortex   = zeros(len_y_pts, 3);
trail_vortex  = zeros(len_y_pts, 3);
% set up lead/trail vortex pts
for m = 1:(len_y_pts)
    if isnan(x_lead_vort_VEC(m)) == 1
        error("x_lead_vort_VEC(%d) = Nan\n", m);
    end % isnan(x_lead_vort_VEC(m)) == 1

    if isnan(y_VEC(m)) == 1
        error("y_VEC(%d) = Nan\n", m);
    end % isnan(x_lead_vort_VEC(m)) == 1

    if isnan(z_lead_vort_VEC(m)) == 1
        error("z_lead_vort_VEC(%d) = Nan\n", m);
    end % isnan(z_lead_vort_VEC(m)) == 1



    if isnan(x_trail_vort_VEC(m)) == 1
        error("x_trail_vort_VEC(%d) = Nan\n", m);
    end % isnan(x_lead_vort_VEC(m)) == 1

    if isnan(y_VEC(m)) == 1
        error("y_VEC(%d) = Nan\n", m);
    end % isnan(y_VEC(m)) == 1

    if isnan(z_lead_vort_VEC(m)) == 1
        error("z_trail_vort_VEC(%d) = Nan\n", m);
    end % isnan(z_trail_vort_VEC(m)) == 1

    lead_vortex(m,  :)  = [x_lead_vort_VEC(m)  y_VEC(m) z_lead_vort_VEC(m)];
    trail_vortex(m, :)  = [x_trail_vort_VEC(m) y_VEC(m) z_trail_vort_VEC(m)];
end % m = 1:(len_y_pts)

lead_control  = zeros(N/2, 3);
trail_control = zeros(N/2, 3);
% set up control points
parfor m = 1:(N/2)
    lead_control(m,  :)  = [x_ctrl_lead_VEC(m)  y_ctrl_pts_VEC(m) z_ctrl_lead_VEC(m)];
    trail_control(m, :) = [x_ctrl_trail_VEC(m) y_ctrl_pts_VEC(m) z_ctrl_trail_VEC(m)];
end % m = 1:(N/2)
control = [lead_control; trail_control];
% -----------------------------------------------------------------------Generate Vor / Control Vectors

% -- Generate Midpoints -------------------------------------------------Generate - Midpoint Matrix
lead_mdpts_ctl  = zeros(N/2, 3);
trail_mdpts_ctl = zeros(N/2, 3);
for m = 1:(N/2)
    lead_mdpts_ctl(m, 1) = .5*(x_lead_vort_VEC(m) + x_lead_vort_VEC(m+1));
    lead_mdpts_ctl(m, 2) = .5*(y_VEC(m)           + y_VEC(m+1));
    lead_mdpts_ctl(m, 3) = .5*(z_lead_vort_VEC(m) + z_lead_vort_VEC(m+1));

    trail_mdpts_ctl(m, 1) = .5*(x_trail_vort_VEC(m) + x_trail_vort_VEC(m+1));
    trail_mdpts_ctl(m, 2) = .5*(y_VEC(m)            + y_VEC(m+1));
    trail_mdpts_ctl(m, 3) = .5*(z_trail_vort_VEC(m) + z_trail_vort_VEC(m+1));
end % m = 1:(N/2)
midpoint_control = [lead_mdpts_ctl; trail_mdpts_ctl];
% -----------------------------------------------------------------------Generate - Midpoint Matrix
```

```matlab
% AT THIS POINT IN THE CODE THE GEOMETRY IS ACCURATE - CHECKED BY "Plots-Wingshape"

% CREATE PANELS TO STORE ALL RELEVANT POINTS: N PANELS, 4 POINTS PER PANEL,
% POINTS ARE ORGANIZED: XA, XB, XC (Control), XM (Midpoint)
% X,Y,Z will be the pages and the final component.

% -- Create Vortex Panels -------------------------------------------------Vortex Panels
% N/2 panels, xa/xb, xyz depth
lead_vor_panels  = zeros(N/2,2,3);
trail_vor_panels = zeros(N/2,2,3);

for m = 1:(N/2)
    % read in xa
    lead_vor_panels (m, 1, :) = lead_vortex(m, :);
    trail_vor_panels(m, 1, :) = trail_vortex(m, :);

    % read in xb
    lead_vor_panels (m, 2, :) = lead_vortex(m+1, :);
    trail_vor_panels(m, 2, :) = trail_vortex(m+1, :);
end % m = 1:(N/2)
vor_panels = [lead_vor_panels; trail_vor_panels];

% check for any value in vor_panels == NAN
for m = 1:N
    if isnan(vor_panels(m,1,1)) == 1
        error('vor_panels = NaN, @(%d,1,1)', m);
    end

    if isnan(vor_panels(m,1,2)) == 1
        error('vor_panels = NaN, @(%d,1,2)', m);
    end

    if isnan(vor_panels(m,1,3)) == 1
        error('vor_panels = NaN, @(%d,1,3)', m);
    end

    if isnan(vor_panels(m,2,1)) == 1
        error('vor_panels = NaN, @(%d,2,1)', m);
    end

    if isnan(vor_panels(m,2,2)) == 1
        error('vor_panels = NaN, @(%d,2,2)', m);
    end

    if isnan(vor_panels(m,2,3)) == 1
        error('vor_panels = NaN, @(%d,2,3)', m);
    end

end % m = 1:N


% check if xb(m) = xa(m+1) ow not set up right
for m = 1:(N-1)
    if vor_panels(m+1, 1, :) ~= vor_panels(m, 2, :)
        error("ERROR xb != xa\n");
    end % vor_panels(m+1, 1, :) ~= vor_panels(m, 2, :)
end % m = 1:(N-1)
% -------------------------------------------------------------------------Vortex Panels

% -- Generate g_hat and n_hat ---------------------------------------------g_hat and n_hat
lead_g  = zeros((N/2), 3);
trail_g = zeros((N/2), 3);
```

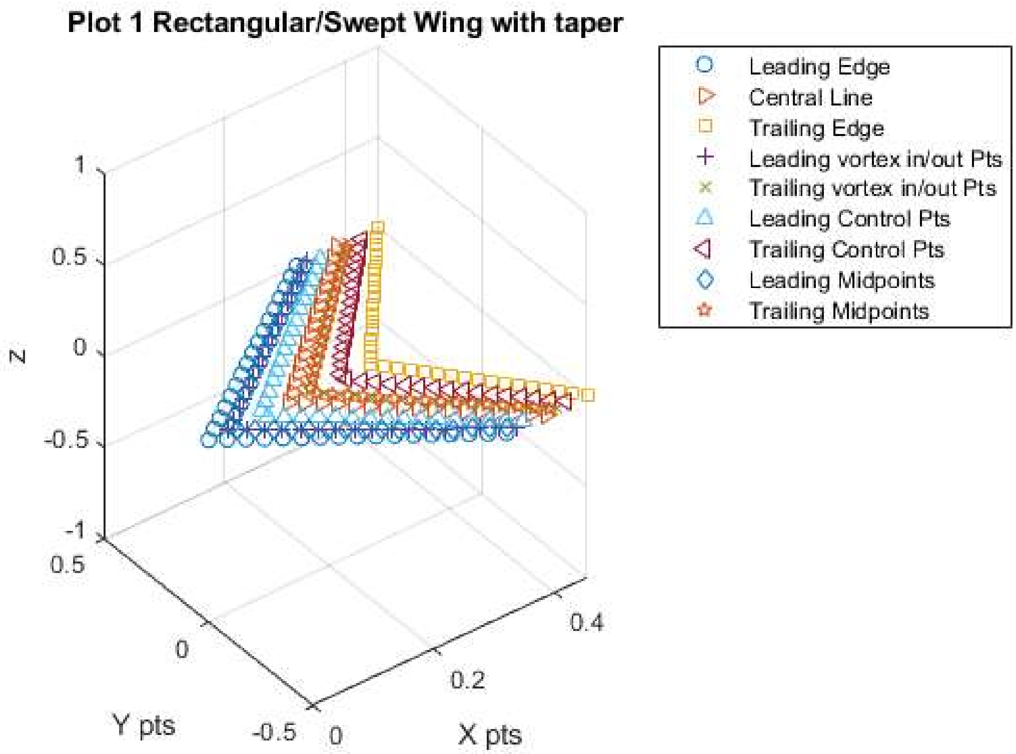
```
lead_n_hat  = zeros((N/2), 3);
trail_n_hat = zeros((N/2), 3);

for m = 1:(N/2)
    % pattern: g_hat = (xb - xa) x (xc - xa)
    lead_g(m, :) = cross (lead_vortex (m+1, :) - lead_vortex (m, :), lead_control (m, :) - lead_vortex (m, :));
    trail_g(m,:) = cross (trail_vortex(m+1, :) - trail_vortex(m, :), trail_control(m, :) - trail_vortex(m, :));

    lead_n_hat (m, :) = lead_g (m, :)/norm(lead_g (m, :));
    trail_n_hat(m, :) = trail_g(m, :)/norm(trail_g(m, :));

     if (norm(lead_n_hat(m, :)) ~= 1) || (norm(trail_n_hat(m, :)) ~= 1)
        disp(norm(lead_g (m, :)));
        disp(norm(trail_g(m, :)));
        disp(lead_g(m,:));
        disp(trail_g(m,:));
        disp(lead_n_hat(m, :));
        disp(trail_n_hat(m,:));
        error("Norm vectors lead_n_hat and trail_n_hat have a norm != 1 !!!\n");
    end
end % m = 1:(N/2)

% -- Check for lead_n_hat and trail_n_hat norms all = 1 -------------------CHECK NORMS
for m = 1:(N/2)
    if (norm(lead_n_hat(m, :)) ~= 1) || (norm(trail_n_hat(m, :)) ~= 1)
        disp(lead_n_hat(m, :));
        disp(trail_n_hat(m,:));
        error("Norm vectors lead_n_hat and trail_n_hat have a norm != 1 !!!\n");
    end
end % m = 1:(N/2)
n_hat = [lead_n_hat; trail_n_hat];
% ------------------------------------------------------------------------CHECK NORMS

% ------------------------------------------------------------------------g_hat and n_hat

% -- Generate Q Influence Matrix ------------------------------------------Create Qinfluence
Q_influence = zeros(N,N,3); % N by N by xyz, page 1 is x, page 2 y, page 3 z
for i = 1:N
    for j = 1:N
%         fprintf("%d, %d\n", i, j);
%         disp(control(i, :));
%         disp(vor_panels(j, 1, :));
%         disp(vor_panels(j, 2, :));
%
        xa(:) = vor_panels(j, 1, :);
        xb(:) = vor_panels(j, 2, :);
        xc(:) = control(i, :);
        Q_influence(i,j, :) = horse(gamma, xc, xa, xb);

    end % j = 1:N
end %for i = 1:N

% diagonal check
for m = 1:(N/2)
    for n = 1:(N/2)
        if (Q_influence(m,m,3) > Q_influence(m,n,3)) && m ~= n && (dihedral == 0)
            fprintf("Q_influence m = %d, n = %d\n", m, n);
            fprintf("Q_influence(%d,%d,3) = %f, ", m,m, Q_influence(m,m,3));
            fprintf("Q_influence(%d,%d,3) = %f\n ", m,n, Q_influence(m,n,3));

            error("something is fishy in Q_influence!");
        end % if
```

```matlab
    end % n = 1:(N/2)

end %m = 1:(N/2)
% -----------------------------------------------------------------------Create Qinfluence

% -- Solve A_mn*Circ_n = B_n, for Circ_n----------------------------------Solve for Circ_n
B_n = zeros(N,1);
parfor m = 1:N
  B_n(m) = dot(U_inf_VEC(m,:), n_hat(m,:));
end % m = 1:N
Amn = zeros(N,N);
q = [0 0 0];
for i = 1:N
    for j = 1:N
        q(1) = Q_influence(i, j, 1); q(2) = Q_influence(i, j, 2); q(3) = Q_influence(i, j, 3);
        Amn(i, j) = dot(q(:), n_hat(i, :));

    end % for j = 1:N
end % for i = 1:N
Circ_n = Amn\(-1*B_n);
% -----------------------------------------------------------------------Solve for Circ_n

% NOW WE WILL WORK ON THE VLM FLOW FIELD AND FORCES

% -- Generate Rn, Qnm w/ midpoint and Vn ---------------------------------RN, Qmn w/ midpoint Vn
Rn = zeros(N, 3);
for m = 1:N
    Rn(m, :) = vor_panels(m, 2, :) - vor_panels(m,1,:);
end % for m = 1:N

%disp(norm(Rn(1,:)));

Qinf_mid = zeros(N, N, 3);

% create the Qinf matrix using midpoints as the xc input
for i = 1:N
    for j = 1:N
        xa(:) = vor_panels(j, 1, :);
        xb(:) = vor_panels(j, 2, :);
        xc(:) = midpoint_control(i, :);
        Qinf_mid(i,j, :) = horse(gamma, xc, xa, xb);

    end % j = 1:N
end % i = 1:N

% diagonal check
for m = 1:(N/2)
    for n = 1:(N/2)
        if (Qinf_mid(m,m,3) >= Qinf_mid(m,n,3)) && m ~= n
            fprintf("Qinf_mid m = %d, n = %d\n", m, n);
            fprintf("Qinf_mid(%d,%d,3) = %f, ", m,m, Qinf_mid(m,m,3));
            fprintf("Qinf_mid(%d,%d,3) = %f\n ", m,n, Qinf_mid(m,n,3));

            error("something is fishy in Qinf_mid!");
        end % if
    end % n = 1:(N/2)

end %m = 1:(N/2)

% Vn_hat
cQnm_x = Circ_n' *  Qinf_mid(:,:, 1);
cQnm_y = Circ_n' *  Qinf_mid(:,:, 2);
cQnm_z = Circ_n' *  Qinf_mid(:,:, 3);
```

```
% N*xyz
cQnm = [cQnm_x'  cQnm_y' cQnm_z'];
Vel_mat_w_wash = U_inf_VEC + cQnm;
% -------------------------------------------------------------------------RN, Qmn w/ midpoint Vn

% -- Wind Fixed Frame Forces ---------------------------------------------WF forces
% Transformation matrix
% wf_bf_transform_mat = ...
%     [cos(beta)*cos(alfa)  sin(beta)*cos(alfa) -sin(alfa); ...
%     -sin(beta)            cos(beta)            0;          ...
%      cos(beta)*sin(alfa)  sin(beta)*sin(alfa)  cos(alfa)];
%
Wind_F_n = zeros(N,3);

parfor m = 1:N
   [bf_wf_transform_mat] = bf2wf(alfa_wash_VEC_N(m), beta);
    F_n = rho * Circ_n(m) * cross( Vel_mat_w_wash(m, :), Rn(m, :));
    Wind_F_n(m, :) = bf_wf_transform_mat\F_n';
end
Drag = sum(Wind_F_n(:,1)); Side = sum(Wind_F_n(:,2)); Lift = sum(Wind_F_n(:,3));
% fprintf("Lift  = %f,\t Drag = %f\n", Lift, Drag);
% -------------------------------------------------------------------------Wf forces


% -- Solve for Variables, CL, CDi, e, a ----------------------------------CL,CDi,e,a
CL  = Lift/(Sref*.5*rho*Uinf^2);
CDi = Drag/(Sref*.5*rho*Uinf^2);
e  = (CL)^2/(pi*AR*CDi);

if errors_on == 1
fprintf("CL    = %f,\t CDi = %f\n", CL, CDi);
fprintf("e     = %f\n", e);

if abs(CL) > .7 || abs(CL) < .1
    fprintf("Not an error, but weird CL values.\n");
end % if CL > .6 || CL < .3

if CDi <= 0
    error("CDi CANNOT be less than or equal to zero! Drag <= 0, & Axial < 0\n.");
end % Cdi < 0

if (e > 2) || (e < .4)
    error("e > 2 or e < .5.  These are BAD e values.");
end % if (e > 2) || (e < .5)
end % if errors_on == 1
% -------------------------------------------------------------------------CL,CDi,e,a

% -- Find Panel Areas ----------------------------------------------------Panel Areas
lead_panel_area  = zeros(1,N/2);
trail_panel_area = zeros(1,N/2);

% get panel areas
for m = 1:N/2
    lead_vec    = [x_leading_VEC(m)   y_VEC(m)   z_edges_VEC(m)] - ...
                  [x_central_VEC(m)   y_VEC(m)   z_edges_VEC(m)];

    lead_vec_2 = [x_leading_VEC(m+1)   y_VEC(m+1)   z_edges_VEC(m+1)] - ...
                 [x_central_VEC(m+1)   y_VEC(m+1)   z_edges_VEC(m+1)];

    center_vec = [x_central_VEC(m+1) y_VEC(m+1) z_edges_VEC(m+1)] - ...
                 [x_central_VEC(m)   y_VEC(m)   z_edges_VEC(m)];

    trail_vec   = [x_trail_VEC(m)     y_VEC(m)   z_edges_VEC(m)] - ...
```

```
                 [x_central_VEC(m)   y_VEC(m)   z_edges_VEC(m)];
    trail_vec_2 = [x_trail_VEC(m+1)     y_VEC(m+1)   z_edges_VEC(m+1)] - ...
                  [x_central_VEC(m+1)   y_VEC(m+1)   z_edges_VEC(m+1)];

    lead_panel_area(m)  = norm(lead_vec + lead_vec_2) * norm(center_vec) * .5;
    trail_panel_area(m) = norm(trail_vec + trail_vec_2) * norm(center_vec) * .5;

end % m = 1:N/2

% -------------------------------------------------------------------------Panel Areas
% -- PLOT1 -----------------------------------------------------------------Plot1-Wingshape
grid on
scatter3(x_leading_VEC, y_VEC, z_edges_VEC, 'o');
hold on
plot3(x_central_VEC,    y_VEC, z_edges_VEC,      '>');
plot3(x_trail_VEC,      y_VEC, z_edges_VEC,      's');

plot3(x_lead_vort_VEC,  y_VEC, z_lead_vort_VEC,  '+');
plot3(x_trail_vort_VEC, y_VEC, z_trail_vort_VEC, 'x');

plot3(x_ctrl_lead_VEC,  y_ctrl_pts_VEC, z_ctrl_lead_VEC, '^');
plot3(x_ctrl_trail_VEC, y_ctrl_pts_VEC, z_ctrl_trail_VEC, '<');

plot3(lead_mdpts_ctl(:, 1), lead_mdpts_ctl(:, 2), lead_mdpts_ctl(:, 3), 'd');
plot3(trail_mdpts_ctl(:, 1), trail_mdpts_ctl(:, 2), trail_mdpts_ctl(:, 3), 'p');


legend('Leading Edge', 'Central Line', 'Trailing Edge',...
       'Leading vortex in/out Pts', 'Trailing vortex in/out Pts', ...
       'Leading Control Pts', 'Trailing Control Pts', ...
       'Leading Midpoints'  , 'Trailing Midpoints');
xlabel('x');
ylabel('y');
zlabel('z');
title('Plot 1 Rectangular/Swept Wing with taper'); xlabel('X pts'); ylabel('Y pts');
hold off
% -------------------------------------------------------------------------Plot1-Wingshape

% -- Profiling -------------------------------------------------------------Profiling
s = profile('status');
disp(s);
% -------------------------------------------------------------------------Profiling
```

```
CL    = 0.394691,        CDi = 0.009109
e     = 1.088687
     ProfilerStatus: 'off'
        DetailLevel: 'mmex'
              Timer: 'performance'
    HistoryTracking: 'timestamp'
        HistorySize: 5000000
```



Published with MATLAB® R2020b