UNIVERSITY OF TURKISH AERONAUTICAL ASSOCIATION

AERODYNAMIC SHAPE OPTIMIZATION

AEE 444

Final Project

DUE DATE

14/01/2019

GROUP- 1

MEMBERS

Mehmet EROL – 140121024

Alican COŞKUN – 140121014

Murat Batuhan GÜNAYDIN - 140121028

Onur KARAASLAN 150221035

The Maximum Fuel Tank Volume of a Wing with Considering the Airfoil that has Maximum l/d

ABSTRACT

Optimization has an important place for engineering processes. Designing a new project needs to have optimal features. In this project, it is aimed to have most efficient wing so that airplane can fly to further. First the airfoil that has maximum (Cl/Cd) was obtained, then by using this airfoil another optimization process was done, which is a fuel tank dimensioned to have maximum volume which will be placed into a finite wing. MATLAB software was used to determine the volume. XFOIL was used for obtain (L/D)max. After this step, Endurance was calculated. Due to this calculation, the maximum dimension of fuel tank that capable to aircraft flight for maximum time.

INTRODUCTION

In this century, aircraft industry is increasing step by step. Designing an aircraft need to follow many processes. While designing the wing of the aircraft, there will be a problem to design fuel tank. In this project, of fuel tank that can be fit in different type of wings by using optimization methods. Generalized Reduced Gradient Method was used, and a MATLAB code was written to take NACA 4-digit airfoils as input and gives the location of fuel tank's edges as output.

NACA 4-DIGIT PLOTTER

Naca 4-digit airfoil is generated by considering 2 main design variables:

1. Mean Camber
2. Thickness distribution.

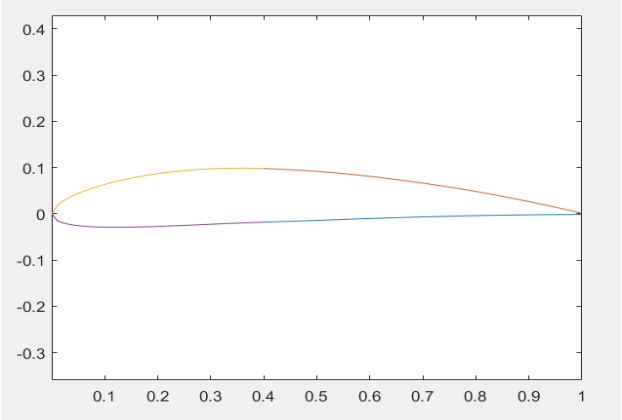


Figure 1: Matlab Drawing

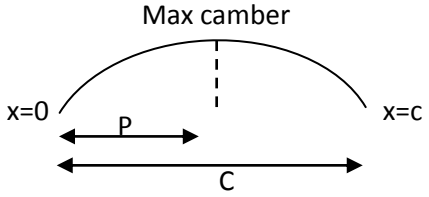
The plot was done by using the formulas which will be covered in the next section. Naca 4-digit nomenclature can be declared as follows: NACA **M P XX**

M= Max Camber in percentage (M/100)

P= Chordwise position of max camber (P/10)

XX= Max Section Thickness (XX/100)

Figure 2: Camberline



$y_c = \frac{M}{P^2}[2P_x - x^2]$ 0≤x<P (1)

$y_c = \frac{M}{P^2}[1 - 2P + 2P_x - x^2]$ P≤x<1 (2)

Derivatives of these functions shown below

$\frac{dy_c}{dx} = \frac{2M}{P^2}[P - x]$ 0≤x<P (3)

$\frac{dy_c}{dx} = \frac{2M}{(1-P)^2}[P - x]$ P≤x<1 (4)

The slope of airfoil is defined as

$\theta = \tan^{-1}(\frac{dy_c}{dx})$ (5)

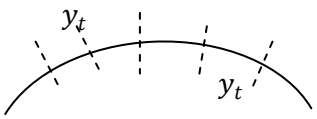


Figure 3: Thickness Distribution

In this figure, dashed lines represent thickness distribution perpendicular to mean camber line. The formula of thickness is follows:

$y_t = \frac{XX}{0.2}[a_0x^{0.5} + a_1x + a_2x^2 + a_3x^3 + a_4x^4]$ (6)

y_t is half-thickness, namely, the top and bottom of camberline both have y_t. Constants in the formulas are listed below.

$$a_0 = 0.2969$$

$$a_1 = -0.1260$$

$$a_2 = -0.3516$$

$$a_3 = 0.2843$$

$$a_4 = 0.2843$$

$$a_5 = -0.1015$$

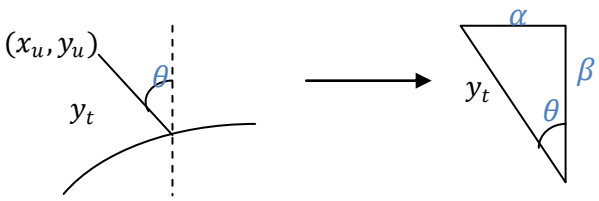


Figure 4: Upper Surface Location

Upper Surface	Lower Surface
$x_u = x - y_t \sin\theta$	$x_l = x + y_t \sin\theta$
$y_u = y_c + y_t \cos\theta$	$y_l = y_c - y_t \cos\theta$

Table 1: Upper & Lower surface distribution

The trapezoidal Fuel tank calculation of area was found by the formula

$(x_{3-}x_2)[\frac{(y_1-y_2)+(y_4-y_3)}{2}]$ (6)

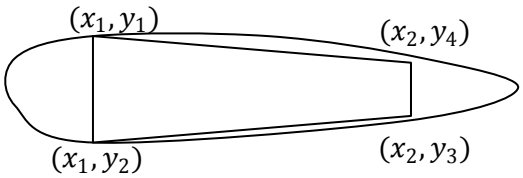


Figure 5: Airfoil cross-section with trapezoidal fuel tank

METHOD of SOLUTION to MATLAB CODE

Firstly, the NACA 4 digit airfoils were defined in the Matlab code. Camber, camber location and thickness were taken as input. And airfoils were plotted. The aim of the project is to create a maximum fuel tank and obtaining the maximum CL/CD. In order to obtain maximum volume of fuel tank, trapezoid area with maximum volume on airfoil has been selected.

Design variables are x1, x2, x3, x4, y1, y2, y3, y4. The airfoil shape was divided into 4 boundary zones. These boundaries were selected as constrain functions. These functions are g1 = 0, g2 = 0, g3 = 0, g4 = 0. The design variable must provide the function. If values provide constraints, they are called feasible. The objective function is the trapezoidal area as shown in the figure.

Function was created on Matlab for objective function. And scripts were created for constraints. Generalized Reduced Gradient Method was used to estimate optimum locations of the fuel tank edges, for this reason first of all initial design variables were selected then GRGM was applied to find new roots of the wing, when the roots are violating the constraints back to Feasible Method was used to take the values again into the constraints. Afterward Optimization of the fuel tank edge locations were completed.

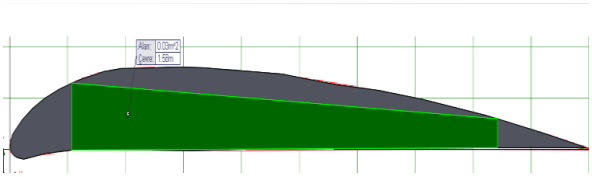


Figure 6: Optimized Airfoil & Maximum Fuel Tank Cross-Section

RESULTS

(Cl/Cd) optimization code, Max Camber =0.05, Max camber location = 0.4, Max Thickness =0.08 was found. So, the airfoil was NACA5408.(Cl/Cd)max equals 88.09. The maximized fuel tank area was calculated as 0.034m² for 1m chord.

Conclusion

To conclude, maximum fuel tank volume was obtained by code written on MATLAB. User has chance to enter different types of airfoil and then, program gives the location of fuel tank's edges as output. As a result of this paper, the fuel tank volume was maximized by considering the condition of maximum range.

```
output =

  struct with fields:

         iterations: 11
          funcCount: 50
    constrviolation: 0
           stepsize: 9.51015793177708e-11
          algorithm: 'interior-point'
      firstorderopt: 0.29396121454576
       cgiterations: 38
            message: 'Local minimum possible. Constraints satisfied....'

Elapsed time is 132.865973 seconds.

x =

        0.0599275507965391        0.434750654730652        0.0800015844417715        2.99978914076308

         -88.0948275862069
```

Figure7: Output of Optimized (Cl/Cd)

REFERENCES

[1] MathworksXfoilMatlabInterfacem.file.

[2] GeneralizedGradientReducedMethod.

[3]Walid Ibrahim, John W. Chinneck, improving solving success in reaching feasibility in sets of nonlinear constraints.

[4] Engineering Optimization Singiresu S. Rao.

APPENDIX

-Maximize (Cl/Cd)

```
function [xu,zu]=aa(t,mm,pp,panel)
air=1;
format longg
   A0=0.2969;
   A1=-0.126;
   A2=-0.3516;
   A3=0.2843;
   A4=-0.1036;
   inc=0.;
      for i=1:panel/2+1;
      xsy(i)=(1-cos(inc))/2.;

zsy(i)=(t/0.2)*(A0*sqrt(xsy(i))+A1*xsy(i)+A2*(x
sy(i))^2.+A3*(xsy(i))^3.+A4*(xsy(i))^4.);
    xsy(panel/2+1+i)=xsy(i);
zsy(panel/2+1+i)=-zsy(i);
      inc=inc+2.*acos(-1.0)/panel;
      end
   for i=1:panel/2+1;
   if xsy(i)<pp
   zc=(mm/(pp^2.))*(2.*pp*xsy(i)-xsy(i)^2.);
   zc1=((2.*mm)/(pp^2.))*(pp-xsy(i));
tet=atan(zc1);
   xsy(i)=xsy(i)-zsy(i)*sin(tet);
xsy(panel/2+1+i)=xsy(panel/2+1+i)-
zsy(panel/2+1+i)*sin(tet);
zsy(i)=zc+zsy(i)*cos(tet);
zsy(panel/2+1+i)=zc+zsy(panel/2+1+i)*cos(tet)
;

end
   if xsy(i) >= pp;
   zc=(mm/(1.-pp)^2.)*(1.-2.*pp+2.*pp*xsy(i)-
xsy(i)^2.);
   zc1=(2.*mm/(1.-pp)^2.)*(pp-xsy(i));
   tet=atan(zc1);
   xsy(i)=xsy(i)-zsy(i)*sin(tet);
   xsy(panel/2+1+i)=xsy(panel/2+1+i)-
zsy(panel/2+1+i)*sin(tet);
   zsy(i)=zc+zsy(i)*cos(tet);
zsy(panel/2+1+i)=zc+zsy(panel/2+1+i)*cos(tet)
;
   end
   end
   for i=1:panel/2+1
xu(i)=xsy(panel/2+2-i);
   zu(i)=zsy(panel/2+2-i);
   end
   for i=panel/2+2:panel+1;
xu(i)=xsy(i+1);

   zu(i)=zsy(i+1);
   end
end
```

```
function [CL,CD] = callaa(mm,pp,t,alpha)
%t=x(3)
%mm=x(1)
%pp=x(2)
panel=200
[xu,zu]=aa(t,mm,pp,panel);
xu=transpose(xu);
zu=transpose(zu);
coord=[xu,zu];
Re=200000
Mach=0.1
[pol,foil] =
xfoil(coord,alpha,Re,Mach,'oper iter 200')
CMF=isempty(pol.CD)
while(CMF==1)
[xu,zu]=aa(t,mm,pp,panel);
xu=transpose(xu);
zu=transpose(zu);
coord=[xu,zu];
Re=200000;
Mach=0.1;
[pol,foil] =
xfoil(coord,alpha,Re,Mach,'oper iter 200')
CMF=isempty(pol.CD)
panel=panel+4
end
CD=pol.CD;
```

```
CL=pol.CL;
%pol.CDp ;
%pol.Cm;
%pol.Top_xtr;
%pol.Bot_xtr;
end
```

```
function [f1,c,ceq,g,gradc,gradceq] =
computeall(x)
    ceq = [];
    gradceq =[];


    [CL,CD] = callaa(x(1),x(2),x(3),x(4));
    c(1) = 0.5-CL ;
    f1 = -CL/CD

    for i=1:1:4
        for j=1:1:4
           xf(j)=x(j);
           xb(j)=x(j);
        end
        xf(i)=x(i)*(1.01);
        xb(i)=x(i)*(0.99);
        if x(i)==0
            xf(i)=0.01;
            xb(i)=-0.01;
        end

       [CLF,CDF] =
callaa(xf(1),xf(2),xf(3),xf(4)) ;
       [CLB,CDB] =
callaa(xb(1),xb(2),xb(3),xb(4)) ;


  FF(i)=-CLF/CDF;
  FB(i)=-CLB/CDB;
  g(i)=(FF(i)-FB(i))/(xf(i)-xb(i));
  cf(i)=0.5-CLF;
  cb(i)=0.5-CLB;
    for k=1:1
        gradc(i,k)=(cf(i)-cb(i))/(xf(i)-
xb(i));
    end
    end
end
```

```
function [pol,foil] =
xfoil(coord,alpha,Re,Mach,varargin)
% Run XFoil and return the results.
% [polar,foil] =
xfoil(coord,alpha,Re,Mach,{extra commands})
%
% Xfoil.exe needs to be in the same
directory as this m function.
% For more information on XFoil visit these
websites;
%
http://web.mit.edu/drela/Public/web/xfoil
%
% Inputs:
%    coord: Normalised foil co-ordinates (n
by 2 array, of x & y
%           from the TE-top passed the LE
to the TE bottom)
%           or a filename of the XFoil co-
ordinate file
%           or a NACA 4 or 5 digit
descriptor (e.g. 'NACA0012')
%    alpha: Angle-of-attack, can be a
vector for an alpha polar
%       Re: Reynolds number (use Re=0 for
inviscid mode)
%     Mach: Mach number
% extra commands: Extra XFoil commands
%           The extra XFoil commands need
to be proper xfoil commands
%           in a character array. e.g.
'oper/iter 150'
%
% The transition criterion Ncrit can be
specified using the
% 'extra commands' option as follows,
% foil =
xfoil('NACA0012',10,1e6,0.2,'oper/vpar n
12')
%
%   Situation           Ncrit
%   -----------------   -----
%   sailplane           12-14
%   motorglider         11-13
%   clean wind tunnel   10-12
%   average wind tunnel    9 <= standard
"e^9 method"
%   dirty wind tunnel    4-8
%
% A flap deflection can be added using the
following command,
% 'gdes flap {xhinge} {yhinge}
{flap_defelction} exec'
%
% Outputs:
%  polar: structure with the polar
coefficients (alpha,CL,CD,CDp,CM,
%          Top_Xtr,Bot_Xtr)
%   foil: stucture with the specific aoa
values (s,x,y,UeVinf,
%          Dstar,Theta,Cf,H,cpx,cp) each
column corresponds to a different
%          angle-of-attack.
%         If only one left hand operator is
specified, only the polar will be parsed
and output
%
% If there are different sized output
```

```
arrays for the different incidence
% angles then they will be stored in a
structured array, foil(1),foil(2)...
%
% If the output array does not have all
alphas in it, that indicates a convergence
failure in Xfoil.
% In that event, increase the iteration
count with 'oper iter ##;
%
% Examples:
%    % Single AoA with a different number
of panels
%    [pol foil] =
xfoil('NACA0012',10,1e6,0.0,'panels n 330')
%
%    % Change the maximum number of
iterations
%    [pol foil] =
xfoil('NACA0012',5,1e6,0.2,'oper iter 50')
%
%    % Deflect the trailing edge by 20deg
at 60% chord and run multiple incidence
angles
%    [pol foil] = xfoil('NACA0012',[-
5:15],1e6,0.2,'oper iter 150','gdes flap
0.6 0 5 exec')
%    % Deflect the trailing edge by 20deg
at 60% chord and run multiple incidence
angles and only
%    parse or output a polar.
%    pol = xfoil('NACA0012',[-
5:15],1e6,0.2,'oper iter 150','gdes flap
0.6 0 5 exec')
%    % Plot the results
%    figure;
%    plot(pol.alpha,pol.CL); xlabel('alpha
[\circ]'); ylabel('C_L'); title(pol.name);
%    figure; subplot(3,1,[1 2]);
%
plot(foil(1).xcp(:,end),foil(1).cp(:,end));
xlabel('x');
%    ylabel('C_p'); title(sprintf('%s @
%g\\circ',pol.name,foil(1).alpha(end)));
%    set(gca,'ydir','reverse');
%    subplot(3,1,3);
%    I = (foil(1).x(:,end)<=1);
%
plot(foil(1).x(I,end),foil(1).y(I,end));
xlabel('x');
%    ylabel('y'); axis('equal');
%
% Some default values
if ~exist('coord','var'), coord =
'NACA0012'; end;
if ~exist('alpha','var'), alpha = 0;
end;
if ~exist('Re','var'),    Re = 1e6;
end;
if ~exist('Mach','var'),  Mach = 0.2;
end;
Nalpha = length(alpha); % Number of alphas
swept
% default foil name
foil_name = mfilename;
% default filenames
wd = fileparts(which(mfilename)); % working
directory, where xfoil.exe needs to be
fname = mfilename;
file_coord= [foil_name '.foil'];
% Save coordinates
if ischar(coord),  % Either a NACA string
or a filename
  if isempty(regexpi(coord,'^NACA *[0-
9]{4,5}$')) % Check if a NACA string
%     foil_name = coord; % some redundant
code removed to go green ( ~isempty if
uncommented)
%   else             % Filename supplied
    % set coord file
    file_coord = coord;
  end;
else
  % Write foil ordinate file
  if exist(file_coord,'file'),
delete(file_coord); end;
  fid = fopen(file_coord,'w');
  if (fid<=0),
    error([mfilename ':io'],'Unable to
create file %s',file_coord);
  else
    fprintf(fid,'%s\n',foil_name);
    fprintf(fid,'%9.5f   %9.5f\n',coord');
    fclose(fid);
  end;
end;
% Write xfoil command file
fid = fopen([wd filesep fname '.inp'],'w');
fprintf(fid,'PLOP\nG\n\n');
if (fid<=0),
  error([mfilename ':io'],'Unable to create
xfoil.inp file');
else
  if ischar(coord),
    if ~isempty(regexpi(coord,'^NACA *[0-
9]{4,5}$')),  % NACA string supplied
      fprintf(fid,'naca
%s\n',coord(5:end));
    else  % filename supplied
      fprintf(fid,'load %s\n',file_coord);
    end;
  else % Coordinates supplied, use the
default filename
    fprintf(fid,'load %s\n',file_coord);
```

```
  end;
  % Extra Xfoil commands
  for ii = 1:length(varargin),
    txt = varargin{ii};
    txt = regexprep(txt,'[ \\\/]+','\n');
    fprintf(fid,'%s\n\n',txt);
  end;
  fprintf(fid,'\n\noper\n');
  % set Reynolds and Mach
  fprintf(fid,'re %g\n',Re);
  fprintf(fid,'mach %g\n',Mach);

  % Switch to viscous mode
  if (Re>0)
    fprintf(fid,'visc\n');
  end;
  % Polar accumulation
  fprintf(fid,'pacc\n\n\n');
  % Xfoil alpha calculations
  [file_dump, file_cpwr] =
deal(cell(1,Nalpha)); % Preallocate cell
arrays

  for ii = 1:Nalpha
    % Individual output filenames
    file_dump{ii} =
sprintf('%s_a%06.3f_dump.dat',fname,alpha(i
i));
    file_cpwr{ii} =
sprintf('%s_a%06.3f_cpwr.dat',fname,alpha(i
i));
    % Commands
    fprintf(fid,'alfa %g\n',alpha(ii));
    fprintf(fid,'dump %s\n',file_dump{ii});
    fprintf(fid,'cpwr %s\n',file_cpwr{ii});
  end;
  % Polar output filename
  file_pwrt = sprintf('%s_pwrt.dat',fname);
  fprintf(fid,'pwrt\n%s\n',file_pwrt);
  fprintf(fid,'plis\n');
  fprintf(fid,'\nquit\n');
  fclose(fid);
  % execute xfoil
  cmd = sprintf('cd %s && xfoil.exe <
xfoil.inp > xfoil.out',wd);
  [status,result] = system(cmd);
  if (status~=0),
    disp(result);
    error([mfilename ':system'],'Xfoil
execution failed! %s',cmd);
  end;
  % Read dump file
  %    #    s        x        y     Ue/Vinf
Dstar     Theta      Cf       H
  jj = 0;
  ind = 1;
% Note that
foil.alpha = zeros(1,Nalpha); % Preallocate
alphas
% Find the number of panels with an inital
run
only = nargout; % Number of outputs
checked. If only one left hand operator
then only do polar
if only >1 % Only do the foil calculations
if more than one left hand operator is
specificed
  for ii = 1:Nalpha
    jj = jj + 1;
    fid = fopen([wd filesep
file_dump{ii}],'r');
    if (fid<=0),
      error([mfilename ':io'],'Unable to
read xfoil output file %s',file_dump{ii});
    else
      D =
textscan(fid,'%f%f%f%f%f%f%f%f','Delimiter'
,'
','MultipleDelimsAsOne',true,'CollectOutput
',1,'HeaderLines',1);
      fclose(fid);
      delete([wd filesep file_dump{ii}]);

      if ii == 1 % Use first run to
determine number of panels (so that NACA
airfoils work without vector input)
        Npanel = length(D{1}); % Number of
airfoil panels pulled from the first angle
tested
        % Preallocate Outputs
        [foil.s, foil.x, foil.y,
foil.UeVinf, foil.Dstar, foil.Theta,
foil.Cf, foil.H] =
deal(zeros(Npanel,Nalpha));
      end

      % store data
      if ((jj>1) &&
(size(D{1},1)~=length(foil(ind).x)) &&
sum(abs(foil(ind).x(:,1)-size(D{1},1)))>1e-
6 ),
        ind = ind + 1;
        jj = 1;
      end;
      foil.s(:,jj) = D{1}(:,1);
      foil.x(:,jj) = D{1}(:,2);
      foil.y(:,jj) = D{1}(:,3);
      foil.UeVinf(:,jj) = D{1}(:,4);
      foil.Dstar(:,jj) = D{1}(:,5);
      foil.Theta(:,jj) = D{1}(:,6);
      foil.Cf(:,jj) = D{1}(:,7);
      foil.H (:,jj)= D{1}(:,8);
    end;
    foil.alpha(1,jj) = alpha(jj);
```

```
    % Read cp file
    fid = fopen([wd filesep
file_cpwr{ii}],'r');
    if (fid<=0),
      error([mfilename ':io'],'Unable to
read xfoil output file %s',file_cpwr{ii});
    else
      C = textscan(fid, '%10f%9f%f',
'Delimiter', '', 'WhiteSpace', '',
'HeaderLines' ,3, 'ReturnOnError', false);
      fclose(fid);
      delete([wd filesep file_cpwr{ii}]);
      % store data
      if ii == 1 % Use first run to
determine number of panels (so that NACA
airfoils work without vector input)
        NCp = length(C{1}); % Number of
points Cp is listed for pulled from the
first angle tested
        % Preallocate Outputs
        [foil.xcp, foil.cp] =
deal(zeros(NCp,Nalpha));
        foil.xcp = C{1}(:,1);
      end
      foil.cp(:,jj) = C{3}(:,1);
    end;
  end;
end
if only <= 1% clear files for default run
  for ii=1:Nalpha % Clear out the xfoil
dump files not used
      delete([wd filesep file_dump{ii}]);
      delete([wd filesep file_cpwr{ii}]);
  end
end
  % Read polar file
  %
  %       XFOIL         Version 6.96
  %
  % Calculated polar for: NACA 0012
  %
  % 1 1 Reynolds number fixed          Mach
number fixed
  %
  % xtrf =   1.000 (top)        1.000
(bottom)
  % Mach =   0.000     Re =     1.000 e 6
Ncrit =  12.000
  %
  %   alpha    CL        CD       CDp
CM     Top_Xtr  Bot_Xtr
  %  ------ -------- --------- --------- --
------ -------- --------
  fid = fopen([wd filesep file_pwrt],'r');
  if (fid<=0),
    error([mfilename ':io'],'Unable to read
xfoil polar file %s',file_pwrt);
  else
    % Header
    % Calculated polar for: NACA 0012
    P = textscan(fid,' Calculated polar
for: %[^\n]','Delimiter','
','MultipleDelimsAsOne',true,'HeaderLines',
3);
    pol.name = strtrim(P{1}{1});
    % xtrf =   1.000 (top)        1.000
(bottom)
    P = textscan(fid,
'%*s%*s%f%*s%f%s%s%s%s%s%s', 1,
'Delimiter', ' ', 'MultipleDelimsAsOne',
true, 'HeaderLines', 2, 'ReturnOnError',
false);
    pol.xtrf_top = P{1}(1);
    pol.xtrf_bot = P{2}(1);
    % Mach =   0.000     Re =     1.000 e 6
Ncrit =  12.000
    P = textscan(fid,
'%*s%*s%f%*s%*s%f%*s%f%*s%*s%f', 1,
'Delimiter', ' ', 'MultipleDelimsAsOne',
true, 'HeaderLines', 0, 'ReturnOnError',
false);
    pol.Re = P{2}(1) * 10^P{3}(1);
    pol.Ncrit = P{4}(1);
    % data
    P = textscan(fid,
'%f%f%f%f%f%f%*s%*s%*s%*s', 'Delimiter',
' ', 'MultipleDelimsAsOne', true,
'HeaderLines' , 4, 'ReturnOnError', false);
    fclose(fid);
    delete([wd filesep file_pwrt]);
    % store data
    pol.alpha = P{1}(:,1);
    pol.CL  = P{2}(:,1);
    pol.CD  = P{3}(:,1);
    pol.CDp = P{4}(:,1);
    pol.Cm  = P{5}(:,1);
    pol.Top_xtr = P{6}(:,1);
    pol.Bot_xtr = P{7}(:,1);
  end
%   if length(pol.alpha) ~= Nalpha % Check
if xfoil failed to converge
%     warning('One or more alpha values
failed to converge. Last converged was
alpha = %f. Rerun with ''oper iter ##''
command.\n', pol.alpha(end))
%   end

end
```

_TRAPEZOIDAL CROSS-SECTIONAL

```
function [ g1 ] = FSolUst( x,yu,M,P,T )

a0=  0.2969;
```

```
a1= -0.1260;
a2= -0.3516;
a3=  0.2843;
a4= -0.1015;


 yc = (M./P.^2).*((2.*P.*x)-x.^2) ;
 dyc_dx = ((2.*M)./(P.^2)).*(P-x);

 theta = atan(dyc_dx);

    term0 = a0*sqrt(x);
    term1 = a1.*x;
    term2 = a2.*x.^2 ;
    term3 = a3.*x.^3 ;
    term4 = a4.*x.^4 ;

    yt =
5.*T.*(term0+term1+term2+term3+term4);



    g1=yu-(yc+yt.*cos(theta));
end
```

```
function [ yl ] = SolAlt( x,M,P,T )
%UNTİTLED6 Summary of this function goes
here
%   Detailed explanation goes here

%coefficients
a0=  0.2969;
a1= -0.1260;
a2= -0.3516;
a3=  0.2843;
a4= -0.1015;

%camber
 yc = (M./P.^2).*((2.*P.*x)-x.^2) ;
 dyc_dx = ((2.*M)./(P.^2)).*(P-x);

 theta = atan(dyc_dx);

    term0 = a0*sqrt(x);
    term1 = a1.*x;
    term2 = a2.*x.^2 ;
    term3 = a3.*x.^3 ;
    term4 = a4.*x.^4 ;

    %thickness
    yt =
5.*T.*(term0+term1+term2+term3+term4);

    %lower line constraint function)
        yl= yc-yt.*cos(theta);

       % g4=yl-(yc-yt.*cos(theta));
end
```

```
function [ fx ] = ObjectiveFunc(
x1,x2,x3,x4,y1,y2,y3,y4)
%UNTİTLED7 Summary of this function goes
here
%   Detailed explanation goes here

 fx= ((((y1-y2)+(y4-y3))*((x3-x2))))./2) ;
end
```

```
function [ yu ] = Sagust(x,M,P,T)
%UNTİTLED Summary of this function goes
here
%   Detailed explanation goes here

a0=  0.2969;
a1= -0.1260;
a2= -0.3516;
a3=  0.2843;
a4= -0.1015;


yu=(M/(1-P).^2).*(1-(2.*P)+(2.*P.*x)-
(x.^2))+5.*T.*(a0.*sqrt(x)+a1.*x+a2.*x.^2+a
3.*x.^3+a4.*x.^4).*cos(atan(((2.*M)/((1-
P).^2)).*(P-x))) ;

%g1=@(x,yu) yu - ((M/(1-P).^2).*(1-
(2.*P)+(2.*P.*x)-
(x.^2))+5.*T.*(a0.*sqrt(x)+a1.*x+a2.*x.^2+a
3.*x.^3+a4.*x.^4).*cos(atan(((2.*M)/((1-
P).^2)).*(P-x)))) ;


end
```

```
function [ yl ] = SagAlt(x,M,P,T)
%UNTİTLED3 Summary of this function goes
here
%   Detailed explanation goes here

a0=  0.2969;
a1= -0.1260;
a2= -0.3516;
a3=  0.2843;
a4= -0.1015;



yl = (M/(1-P).^2).*(1-(2.*P)+(2.*P.*x)-
(x.^2))-
5.*T.*(a0.*sqrt(x)+a1.*x+a2.*x.^2+a3.*x.^3+
a4.*x.^4).*cos(atan(((2.*M)/((1-
P).^2)).*(P-x))) ;

%func=yl-((M/(1-P).^2).*(1-
(2.*P)+(2.*P.*x)-(x.^2))-
5.*T.*(a0.*sqrt(x)+a1.*x+a2.*x.^2+a3.*x.^3+
a4.*x.^4).*cos(atan(((2.*M)/((1-
P).^2)).*(P-x)))) ;
```

```
end
```

```
function [x] = back2feasible(x0, C, GC, u,
l)
%   x0: infeasible vector.
%   C:  nonlinear constraint.
%   GC: constraint gradient.
%   u:  upper bound for desgin vars.
%   l:  lower bound for design vars.
syms x1 x2 x3 x4 y1 y2 y3 y4
alpha = 1e-8;
beta = 1e-8;
mu = 100;
J = length(x0);
I = length(C);


for k = 1:mu
    NINF = 0;
    n = zeros(size(x0));
    s = zeros(size(x0));
    for i = 1:I
        c(i) = eval(subs(C,
[x1;x2;x3;x4;y1;y2;y3;y4], x0));
        gc(i,:) = eval(subs(GC, [x1 x2 x3
x4 y1 y2 y3 y4], x0'));
        if c(i) ~= 0
            if norm(gc(i,:)) > 0
                d(i) = -1;
            else
                d(i) = 1;
            end
            fv(i,:) =
c(i).*d(i).*gc(i,:)./norm(gc(i,:)).^2;
            if norm(fv(i,:)) > alpha
                NINF = NINF + 1;
                for j = 1:J
                    n(j) = n(j) + 1;
                    s(j) = s(j) + fv(i,j);
                end
            end
        end
    end
    if NINF == 0
        x = x0;
        disp('BackToFeasible')
        return
    end
    for j = 1:J
        if n(j) ~= 0
            t(j) = s(j)/n(j);
        else
            t(j) = 0;
        end
    end
    if norm(t) <= beta
        x = x0;
        disp('Couldn''t go
BackToFeasible!')
        return
    end
    x0 = x0 + t';
    for j = 1:J
        if x0(j) > u
            x0(j) = u;
        elseif x0(j) < l
            x0(j) = l;
        end
    end
end
x = x0;
disp('Couldn''t go BackToFeasible!')
end
```

```
clc;
clear;
clear all;

camber=0.04;
locamber=0.4;
thickness=0.12;
syms coef;

syms a1 a2 b1 b2 b3 b4;

syms x_3 y_3 x_4 y_4 x_1 y_1 x_2 y_2;

x=  0.4:0.01:1;
x1= 0.4:0.01:1;
x2= 0:0.01:0.4;
x3= 0:0.01:0.4;

g1=SagAlt(x,camber,locamber,thickness);
g2=Sagust(x1,camber,locamber,thickness);
g3=SolUst(x2,camber,locamber,thickness);
g4=SolAlt(x3,camber,locamber,thickness);

G3=FSagAlt(x_3,y_3,camber,locamber,thicknes
s);

G4=FSagUst(x_4,y_4,camber,locamber,thicknes
s);

G2=FSolAlt(x_2,y_2,camber,locamber,thicknes
s);

G1=FSolUst(x_1,y_1,camber,locamber,thicknes
s);


Fx=ObjectiveFunc(x_1,y_1,x_2,y_2,x_3,y_3,x_
4,y_4);

first=diff(Fx,x_1);
second=diff(Fx,y_1);
```

```
third=diff(Fx,x_2);
fourth=diff(Fx,y_2);
fifth=diff(Fx,x_3);
sixth=diff(Fx,y_3);
seventh=diff(Fx,x_4);
eigth=diff(Fx,y_4);

GradY =
[first;diff(Fx,x_2);diff(Fx,x_3);diff(Fx,x_
4)] ;
GradZ = [second;fourth;sixth;eigth];

g1first=diff(G1,x_1);
g1second=diff(G1,y_1);
g2first=diff(G2,x_2);
g2second=diff(G2,y_2);
g3first=diff(G3,x_3);
g3second=diff(G3,y_3);
g4first=diff(G4,x_4);
g4second=diff(G4,y_4);
%4 tane arbitrarily dependent value

C=[g1first,0,0,0;0,g2second,0,0;0,0,g3first
,0;0,0,0,g4first];
D=[g1second,0,0,0;0,g2second,0,0;0,0,g3seco
nd,0;0,0,0,g4second];

plot(x,g1);
hold on;
plot(x1,g2);
hold on;
plot(x2,g3);
hold on;
plot(x3,g4);
axis equal;

Gr=GradY-((inv(D).*C)')*GradZ ;


S=-Gr;

k=1;

%initial values
x_1=0.3 ; x_2= 0.4; x_3= 0.5; x_4= 0.7;
y_1=0.2 ; y_2=-0.2 ; y_3=0.1 ; y_4= 0.15;

S1=double(subs(S))  ;

lambda= ones(4,1);

 xu=1; xl=0.2;


 for i=1:1:4;
    if S1(i,1)> 0
        lambda(i,1)= (xu-x_1)/S1(i,1) ;

    else

        lambda(i,1)= (xl-x_1)/S1(i,1) ;

    end

 end



M=abs([lambda(1);lambda(2);lambda(3);lambda
(4)]);

   minimum= min(M);

   T=-inv(D)*C*S;

   T1=double(subs(T));



   X0=[0.3;0.4;0.5;0.7;0.2;-0.2;0.1;0.15];

   lambdam=ones(4,1);

   for i=1:1:4
    if T1(i,1)> 0
        lambdam(i,1)= (xu-X0(i+4))/T1(i,1)
;

    else

        lambdam(i,1)= (xl-X0(i+4))/T1(i,1)
;

    end

   end


M=abs([lambdam(1);lambdam(2);lambdam(3);lam
bdam(4)]);


        W=((x_3+coef*S1(3))-
(x_2+coef*S1(2))).*(((y_1+coef*T1(1)-
y_2+coef*T1(2))+(-y_3-
coef*T1(3)+y_4+coef*T1(4))))/2;

     dW=diff(W,coef);

     subs(dW);
```

```
 minimum

 Xnew=ones(8,1);

Xnew(1)=X0(1)+minimum*S1(1);
Xnew(2)=X0(2)+minimum*S1(2);
Xnew(3)=X0(3)+minimum*S1(3);
Xnew(4)=X0(4)+minimum*S1(4);
Xnew(5)=X0(5)+minimum*T1(1);
Xnew(6)=X0(6)+minimum*T1(2);
Xnew(7)=X0(7)+minimum*T1(3);
Xnew(8)=X0(8)+minimum*T1(4);

Xnew
```