A. Additional codes

The code to create the mesh in Chapter 5:

```
function foilgmsh(archi,alfa,yplus,eter,Re,M,T0,N,bump)
%foilgmsh(archi,alfa,yplus,eter,Re,M,T0,N,bump)
%
%foilgmsh will create a GEO file with all necessary instructions to mesh an airfoil geometry only with hexahedra in Gmsh, departuring from a set of points defining the airfoil section contour. So far this code can only deal with monoelement airfoils and 2D simulations for finite volume CFD codes.
%The on-screen output displays a summary of the input, some statistics of the mesh to be created and useful information for definition of the case in CFD softwares, specially Code_Saturne.
%
%- archi: string which defines the complete name of the airfoil coordinates file. The coordinates must be given in XFoil format starting from the trailing edge and circling counterclockwise. The coordinates don't need to be adimensionalized, but the program won't do it either, as the airfoil chord will be estimated automatically from it. The author consider's this is useful to compare further results with experimental data. The coordinates can be off-centered by small amounts.
%The file must be located in the active directory. The output file will have the same name with the string '.geo' appended. If you wish to include text in your coordinates file, please add a percentage sign on the first column, otherwise an error will occur.
%
%- alfa: desired angle of attack in degrees. The mesh generated will be so in wind axis system, where wind velocity coincides in sense and direction with the positive X axis, so when defining inlet speed, Y and Z components should be zero. Also, the section plane coincides with the mesh XY plane.
%
%- yplus: desired y+ value around the airfoil. Spacing between susecquent cells normal to the airfoil surface is done with an authomatically defined geometric progression.
%
%- eter: string defining material medium where airfoil is submerged. Only three posibilities are allowed, 'a' for normal air, 'h' for distilled water and 'n' for nitrogen.
%
%- Re: Reynolds number based on airfoil chord and freestream speed. The reference chord and that of the airfoil in the coordinates file must coincide.
%
%- M: if the medium is air or nitrogen, it is the Mach number. If the medium is water, it is the freestream speed magnitude in m/s. Although the applied formulas hold for transonic and supersonic flows, the resulting mesh might not be suitable for those cases. Compressible fully subsonic flows should not be a problem.
%
```

```
%- T0: if the medium is air or nitrogen, it is the stagnation temperature in Kelvin degrees. If the medium is water, it is the non-stagnated flow temperature in Celsius degrees.
%
%- N: four integers vector whre N(1) is the number of hexahedra along the airfoil chord, therefore the whole contour of the airfoil will be discretized in 2*N(1) elements. N(2) is the number of elements normal to the chord and inside a semiellipse closely enclosing the airfoil (a value between 25 and 100 should suffice for most applications). N(3) is the number of elements into which the horizontal leading edge-inlet and trailing edge-outlet gaps will be discretized. N(4) is similar to N(3) but applied to the vertical gaps between the airfoil and the top/bottom walls defining the box.
%
%- bump: a value which defines the mesh concentration degree around the leading and trailing edge. If bump=1 the cell lenghts will be uniform along the airfoil contour, if bump>1 the elements will be concentrated around the midchord. If 0<bump<1 the elements will concentrate around the edges, and don't be surprised if you need very low values like 0.1 or less to achieve a noticeable concentration.
%
%To mesh, simply open in Gmsh the generated GEO file, go to Mesh with the menu or by pressing 'm' and click on "3D". Save the mesh as a MED file for use with Code_Saturne (remember to apply 'check cell orientation' in the GUI or preprocessor). The generated groups are "inlet", "outlet", "airfoil", "symmetry" and "walls".
%The on-screen output text provides information to define all necessary variables in CS's GUI. The hydraulic diameter value is just a dummy number suitable to initialize properly the turbulence model for external flows.
%
%Send some feedback if you wish to cesar_vecchio@gmx.com (I also accept Ferraris and Porsches). I hope you find this software useful.
%----------------------
%Cesar A. Vecchio Toloy
%----------------------
%Disclaimer: I am giving you this software as is fully for free. I will not be responsible for any harm of any kind this code and the uses you give to it may cause. You are using this code under your own responsability and risk.

more off

N = N+1; %number of nodes on upper and lower surface
alfa = -alfa*pi/180; %conversion to radians and Gmsh references
inic = load(archi); %loading coordinates file...
[m void] = size(inic);
inic(1,2)=(inic(1,2)+inic(m,2))/2; %the trailing edge is closed. Sorry for the inconvenience.
percor=inic(1:m-1,:); %the (now) extra trailing edge point is removed.
m=m-1;disp(m)
z0 = 0;
[maxx posmaxx] = max(percor(:,1));
[minx posminx] = min(percor(:,1));
cuerda = maxx-minx; %computation of airfoil chord
[maxy posmaxy] = max(percor(:,2));
[miny posminy] = min(percor(:,2));

fid = fopen(strcat(archi,'.geo'),'w'); %opening the output file
```

```
%writing the points which define the airfoil
for i = 1:m
fprintf(fid,'Point(%i) = {%.10g,%.10g,%.10g,%.10g};\n',i,percor(i,1),percor(i,2),z0,cuerda/100);
end

%writing the points which define the enclosing semiellipse
fprintf (fid, 'Point(%i) = {%.10g,%.10g,%.10g,%.10g};\n', m+1,minx-cuerda/20,0,z0,cuerda/25);
fprintf (fid, 'Point(%i) = {%.10g,%.10g,%.10g,%.10g};\n', m+2,maxx,maxy+cuerda/4,z0,cuerda/25);
fprintf (fid, 'Point(%i) = {%.10g,%.10g,%.10g,%.10g};\n', m+3,maxx,miny-cuerda/4,z0,cuerda/25);

fprintf (fid, 'Spline(1) = {'); %defining an interpolating spline for the upper surface
for i = posminx:m
fprintf (fid, '%i,', i);
end
%The following and all the lines where you see 'Transfinite' is a way of indicating to Gmsh that a structured mesh will be made.
fprintf (fid, '%i}; Transfinite Line{1} = %i Using Bump %f;\n', 1,N(1),bump);
fprintf (fid, 'Spline(2) = {'); %lower surface interpolating spline
for i = 1:posminx-1
fprintf (fid, '%i,', i);
end
fprintf (fid, '%i}; Transfinite Line{2} = %i Using Bump %f;\n', posminx,N(1),bump);

%Defining the lines of our enclosing semiellipse
fprintf (fid, 'Ellipse(3) = {%i,%i,%i,%i}; Transfinite Line{3} = %i Using Progression 1;\n', m+1,1,posminx,m+2,N(1));
fprintf (fid, 'Ellipse(4) = {%i,%i,%i,%i}; Transfinite Line{4} = %i Using Progression 1;\n', m+1,1,posminx,m+3,N(1));
%Calculating minimum cell distance from wall and geometric progression with subfunctions
Ymin = ypar (yplus,cuerda,Re,M,T0,eter); Prog5 = mindist(Ymin,norm(percor(posmaxx,:)-[maxx,maxy+cuerda/4]),N(2));
fprintf (fid, 'Line(5) = {%i,%i}; Transfinite Line{5} = %i Using Progression %.10g;\n', 1,m+2,N(2),Prog5);
Prog6 = mindist(Ymin,norm(percor(posmaxx,:)-[maxx,miny-cuerda/4]),N(2));
fprintf (fid, 'Line(6) = {%i,%i}; Transfinite Line{6} = %i Using Progression %.10g;\n', 1,m+3,N(2),Prog6);
Prog7 = mindist(Ymin,norm(percor(posminx,:)-[minx-cuerda/20,0]),N(2));
fprintf (fid, 'Line(7) = {%i,%i}; Transfinite Line{7} = %i Using Progression %.10g;\n', posminx,m+1,N(2),Prog7);

%2D surfaces are created from the available liens so far
fprintf (fid, 'Line Loop(1) = {-2,5,-3,-7};\n');
fprintf (fid, 'Ruled Surface(1) = {1};\n');
fprintf (fid, 'Transfinite Surface(1) = {%i,%i,%i,%i};\n', 1,posminx,m+1,m+2);
fprintf (fid, 'Line Loop(2) = {1,6,-4,-7};\n');
fprintf (fid, 'Ruled Surface(2) = {2};\n');
fprintf (fid, 'Transfinite Surface(2) = {%i,%i,%i,%i};\n', 1,posminx,m+1,m+3);

%Let's tell Gmsh to rotate the airfoil+semielipse our desired angle of attack
fprintf (fid, 'Rotate {{0,0,1},{%.10g,0,0},%.10g} {Surface{1,2};}\n', minx+cuerda/2,alfa);
```

```

%Now some points to define the flowfield boundaries. Notice the resulting box will be 15*chord long and 8*chord high.
fprintf (fid , 'Point(%i) = {%.10g,%.10g,%.10g,%.10g};\n', m+4,maxx-(1-cos(alfa))*cuerda/2-sin(alfa)*(cuerda/4+maxy),4*cuerda,z0,cuerda);
fprintf (fid , 'Point(%i) = {%.10g,%.10g,%.10g,%.10g};\n', m+5,minx-2*cuerda,4*cuerda,z0,cuerda);
fprintf (fid , 'Point(%i) = {%.10g,%.10g,%.10g,%.10g};\n', m+6,minx-4*cuerda,4*cuerda,z0,cuerda);
fprintf (fid , 'Point(%i) = {%.10g,%.10g,%.10g,%.10g};\n', m+7,minx-4*cuerda,0-0.55*cuerda*sin(alfa),z0,cuerda);
fprintf (fid , 'Point(%i) = {%.10g,%.10g,%.10g,%.10g};\n', m+8,minx-4*cuerda,-4*cuerda,z0,cuerda);
fprintf (fid , 'Point(%i) = {%.10g,%.10g,%.10g,%.10g};\n', m+9,minx-2*cuerda,-4*cuerda,z0,cuerda);
fprintf (fid , 'Point(%i) = {%.10g,%.10g,%.10g,%.10g};\n', m+10,maxx-(1-cos(alfa))*cuerda/2-sin(alfa)*(-cuerda/4+miny),-4*cuerda,z0,cuerda);
fprintf (fid , 'Point(%i) = {%.10g,%.10g,%.10g,%.10g};\n', m+11,maxx+10*cuerda,-4*cuerda,z0,cuerda);
fprintf (fid , 'Point(%i) = {%.10g,%.10g,%.10g,%.10g};\n', m+12,maxx+10*cuerda,percor(posmaxx,2)+cos(alfa)*(miny-cuerda/4)+cuerda/2*sin(alfa),z0,cuerda);
fprintf (fid , 'Point(%i) = {%.10g,%.10g,%.10g,%.10g};\n', m+13,maxx+10*cuerda,sin(alfa)*cuerda/2,z0,cuerda);
fprintf (fid , 'Point(%i) = {%.10g,%.10g,%.10g,%.10g};\n', m+14,maxx+10*cuerda,percor(posmaxx,2)+cos(alfa)*(maxy+cuerda/4)+cuerda/2*sin(alfa),z0,cuerda);
fprintf (fid , 'Point(%i) = {%.10g,%.10g,%.10g,%.10g};\n', m+15,maxx+10*cuerda,4*cuerda,z0,cuerda);

%Now we join the previous points with some lines
Prog = mindist(Ymin*Prog7^(N(2)-1),abs(3.95*cuerda+(1-cos(alfa))*0.55*cuerda),N(3));
fprintf (fid , 'Line(8) = {%i,%i}; Transfinite Line{8} = %i Using Progression %.10g;\n', m+1,m+7,N(3),Prog);

Prog = mindist(cuerda/N(1),abs(10*cuerda+(1+cos(alfa))*0.5*cuerda),N(3));
fprintf (fid , 'Line(9) = {%i,%i}; Transfinite Line{9} = %i Using Progression %.10g;\n', 1,m+13,N(3),Prog);

L = 4*cuerda-(percor(posmaxx,2)+cuerda/4)*cos(alfa)-cuerda/2*sin(alfa);
Prog = mindist(Ymin*Prog5^(N(2)-1),L,N(4));
fprintf (fid , 'Line(10) = {%i,%i}; Transfinite Line{10} = %i Using Progression %.10g;\n', m+2,m+4,N(4),Prog);

L = norm([minx-cuerda*2,cuerda*4]-[(minx-cuerda/20)-(1-cos(alfa))*0.55*cuerda,percor(posminx,2)-sin(
Prog = mindist(Ymin*Prog7^(N(2)-1),L,N(4));
fprintf (fid , 'Line(11) = {%i,%i}; Transfinite Line{11} = %i Using Progression %.10g;\n', m+1,m+5,N(4),Prog);

L = 4*cuerda+(miny-cuerda/4)*cos(alfa)+cuerda/2*sin(alfa);
Prog = mindist(Ymin*Prog6^(N(2)-1),L,N(4));
fprintf (fid , 'Line(12) = {%i,%i}; Transfinite Line{12} = %i Using Progression %.10g;\n', m+3,m+10,N(4),Prog);

L = norm([minx-cuerda*2,-cuerda*4]-[(minx-cuerda/20)-(1-cos(alfa))*0.55*cuerda,percor(posminx,2)-sin(
Prog = mindist(Ymin*Prog7^(N(2)-1),L,N(4));
```

```
fprintf (fid, 'Line(13) = {%i,%i}; Transfinite Line{13} = %i Using Progression %.10g;\n', m+1,m+9,N(4),Prog);

Prog = mindist(cuerda/N(1),abs(10*cuerda+(1-cos(alfa))*0.5*cuerda+sin(alfa)*(maxy+cuerda/4)),N(3));
fprintf (fid, 'Line(14) = {%i,%i}; Transfinite Line{14} = %i Using Progression %.10g;\n', m+2,m+14,N(3),Prog);

Prog = mindist(cuerda/N(1),abs(10*cuerda+(1-cos(alfa))*0.5*cuerda+sin(alfa)*(miny-cuerda/4)),N(3));
fprintf (fid, 'Line(15) = {%i,%i}; Transfinite Line{15} = %i Using Progression %.10g;\n', m+3,m+12,N(3),Prog);

fprintf (fid, 'Line(16) = {%i,%i}; Transfinite Line{16} = %i Using Progression 1.00;\n', m+4,m+5,N(1));

fprintf (fid, 'Line(17) = {%i,%i}; Transfinite Line{17} = %i Using Progression 1.00;\n', m+10,m+9,N(1));

Prog = mindist((3*cuerda-cuerda/2*cos(alfa)-sin(alfa)*(maxy+cuerda/4))/N(1),2*cuerda,N(3));
fprintf (fid, 'Line(18) = {%i,%i}; Transfinite Line{18} = %i Using Progression %.10g;\n', m+5,m+6,N(3),Prog);

Prog = mindist((3*cuerda-cuerda/2*cos(alfa)-sin(alfa)*(miny-cuerda/4))/N(1),2*cuerda,N(3));
fprintf (fid, 'Line(19) = {%i,%i}; Transfinite Line{19} = %i Using Progression %.10g;\n', m+9,m+8,N(3),Prog);

Prog = mindist(cuerda/N(1),4*cuerda+cuerda*0.55*sin(alfa),N(4));
fprintf (fid, 'Line(20) = {%i,%i}; Transfinite Line{20} = %i Using Progression %.10g;\n', m+7,m+6,N(4),Prog);

Prog = mindist(cuerda/N(1),4*cuerda-cuerda*0.55*sin(alfa),N(4));
fprintf (fid, 'Line(21) = {%i,%i}; Transfinite Line{21} = %i Using Progression %.10g;\n', m+7,m+8,N(4),Prog);

Prog = mindist((3*cuerda-cuerda/2*cos(alfa)-sin(alfa)*(maxy+cuerda/4))/N(1),abs(10*cuerda+(1-cos(alfa))
fprintf (fid, 'Line(22) = {%i,%i}; Transfinite Line{22} = %i Using Progression %.10g;\n', m+4,m+15,N(3),Prog);

Prog = mindist((3*cuerda-cuerda/2*cos(alfa)-sin(alfa)*(miny-cuerda/4))/N(1),abs(10*cuerda+(1-cos(alfa))
fprintf (fid, 'Line(23) = {%i,%i}; Transfinite Line{23} = %i Using Progression %.10g;\n', m+10,m+11,N(3),Prog);

L = 4*cuerda+(miny-cuerda/4)*cos(alfa)+cuerda/2*sin(alfa);
Prog = mindist(Ymin*Prog6^(N(2)-1),L,N(4));
fprintf (fid, 'Line(24) = {%i,%i}; Transfinite Line{24} = %i Using Progression %.10g;\n', m+12,m+11,N(4),Prog);

Prog = mindist(Ymin,abs(miny-cuerda/4)*cos(alfa),N(2));
fprintf (fid, 'Line(25) = {%i,%i}; Transfinite Line{25} = %i Using Progression %.10g;\n', m+13,m+12,N(2),Prog);

Prog = mindist(Ymin,abs(maxy+cuerda/4)*cos(alfa),N(2));
```

```
fprintf (fid, 'Line(26) = {%i,%i}; Transfinite Line{26} = %i Using Progression %.10g;\n', m+13,m+14,N(2),Prog);

L = 4*cuerda-(maxy+cuerda/4)*cos(alfa)-cuerda/2*sin(alfa);
Prog = mindist(Ymin*Prog5^(N(2)-1),L,N(4));
fprintf (fid, 'Line(27) = {%i,%i}; Transfinite Line{27} = %i Using Progression %.10g;\n', m+14,m+15,N(4),Prog);

%2D surfaces are defined from the previous lines.
fprintf (fid, 'Line Loop(3) = {3,10,16,-11};\n');
fprintf (fid, 'Ruled Surface(3) = {3};\n');
fprintf (fid, 'Transfinite Surface(3) = {%i,%i,%i,%i};\n', m+1,m+2,m+4,m+5);

fprintf (fid, 'Line Loop(4) = {4,12,17,-13};\n');
fprintf (fid, 'Ruled Surface(4) = {4};\n');
fprintf (fid, 'Transfinite Surface(4) = {%i,%i,%i,%i};\n', m+1,m+3,m+10,m+9);

fprintf (fid, 'Line Loop(5) = {-18,-11,8,20};\n');
fprintf (fid, 'Ruled Surface(5) = {5};\n');
fprintf (fid, 'Transfinite Surface(5) = {%i,%i,%i,%i};\n', m+1,m+5,m+6,m+7);

fprintf (fid, 'Line Loop(6) = {-19,-13,8,21};\n');
fprintf (fid, 'Ruled Surface(6) = {6};\n');
fprintf (fid, 'Transfinite Surface(6) = {%i,%i,%i,%i};\n', m+1,m+7,m+8,m+9);

fprintf (fid, 'Line Loop(7) = {5,14,-26,-9};\n');
fprintf (fid, 'Ruled Surface(7) = {7};\n');
fprintf (fid, 'Transfinite Surface(7) = {%i,%i,%i,%i};\n', m+2,m+14,m+13,1);

fprintf (fid, 'Line Loop(8) = {6,15,-25,-9};\n');
fprintf (fid, 'Ruled Surface(8) = {8};\n');
fprintf (fid, 'Transfinite Surface(8) = {%i,%i,%i,%i};\n', m+3,m+12,m+13,1);

fprintf (fid, 'Line Loop(9) = {14,27,-22,-10};\n');
fprintf (fid, 'Ruled Surface(9) = {9};\n');
fprintf (fid, 'Transfinite Surface(9) = {%i,%i,%i,%i};\n', m+2,m+14,m+15,m+4);

fprintf (fid, 'Line Loop(10) = {15,24,-23,-12};\n');
fprintf (fid, 'Ruled Surface(10) = {10};\n');
fprintf (fid, 'Transfinite Surface(10) = {%i,%i,%i,%i};\n', m+3,m+10,m+11,m+12);

fprintf (fid, 'Recombine Surface{1,2,3,4,5,6,7,8,9,10}=0;\n'); %This is important, it tells Gmsh to attemp to join the default triangles into quadrangles (2triangles=1quadrangle)

%The next lines extrude a small height the 2D surface to have a one-cell-depth volume, necessary for finite volume codes. It is not necessary for finite elements, but who uses them? :D
fprintf (fid, 'j1[] = Extrude {0,0,%.10g} {Surface{1};Layers{1};Recombine;};\n', cuerda/10);
fprintf (fid, 'j2[] = Extrude {0,0,%.10g} {Surface{2};Layers{1};Recombine;};\n', cuerda/10);
fprintf (fid, 'j3[] = Extrude {0,0,%.10g} {Surface{3};Layers{1};Recombine;};\n', cuerda/10);
fprintf (fid, 'j4[] = Extrude {0,0,%.10g} {Surface{4};Layers{1};Recombine;};\n', cuerda/10);
```

```
fprintf (fid, 'j5[] = Extrude {0,0,%.10g} {Surface{5};Layers{1};Recombine;};\n',
    cuerda/10);
fprintf (fid, 'j6[] = Extrude {0,0,%.10g} {Surface{6};Layers{1};Recombine;};\n',
    cuerda/10);
fprintf (fid, 'j7[] = Extrude {0,0,%.10g} {Surface{7};Layers{1};Recombine;};\n',
    cuerda/10);
fprintf (fid, 'j8[] = Extrude {0,0,%.10g} {Surface{8};Layers{1};Recombine;};\n',
    cuerda/10);
fprintf (fid, 'j9[] = Extrude {0,0,%.10g} {Surface{9};Layers{1};Recombine;};\n',
    cuerda/10);
fprintf (fid, 'j10[] = Extrude {0,0,%.10g} {Surface{10};Layers{1};Recombine;};\n',
    cuerda/10);

%Grouping the faces and the volumes...
fprintf (fid, 'Physical Surface("inlet") = {j5[5],j6[5]};\n');
fprintf (fid, 'Physical Surface("outlet") = {j7[4],j8[4],j9[3],j10[3]};\n');
fprintf (fid, 'Physical Surface("airfoil") = {j1[2],j2[2]};\n');
fprintf (fid, 'Physical Surface("walls") =
    {j3[4],j4[4],j5[2],j6[2],j9[4],j10[4]};\n');
fprintf (fid, 'Physical Surface("symmetry") =
    {1,2,3,4,5,6,7,8,9,10,j1[0],j2[0],j3[0],j4[0],j5[0],j6[0],j7[0],j8[0],j9[0],j10[0]};\n');

fprintf (fid, 'Physical Volume("Volumen") =
    {j1[1],j2[1],j3[1],j4[1],j5[1],j6[1],j7[1],j8[1],j9[1],j10[1]};\n');

fclose(fid);

N = N-1;
%In case you wonder, the info is just below:
printf ('The mesh is made of %i linear hexahedra.\n',
    2*(N(1)*(N(2)+N(4))+N(4)*2*N(3)+N(2)*N(3)))
printf
    ('\n-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_\n\n')

end



%%%%%%%%%%----%%%%%%%%%%----%%%%%%%%%%----%%%%%%%%%%----%%%%%%%%%%----%%%%%%%%%%



function Prog = mindist(Ymin,L,N)

%For a line of length L to be discretized in N elements, of which the shortest is
    at the beggining and has a length Ymin, we compute the Prog such that
    L=Ymin*sum(Prog^n, from n=0 to n=N). Ymin can be but is not limited to the
    Ymin defined in the next function.

e = 1;
Prog = 1.001;

while e > 0.001

  Prog_t = Prog +
      ((Prog^4-2*Prog^3+Prog^2)*L+(Prog^3-Prog^2+Prog^N*(Prog-Prog^2))*Ymin)/(((Prog-1)*Prog^N*Ymin*N-
  e = abs((Prog_t-Prog)/Prog);
```

```
  Prog = Prog_t;

end

if Prog < 1

  Prog = 1;

end

end



%%%%%%%%%%-----%%%%%%%%%%-----%%%%%%%%%%-----%%%%%%%%%%-----%%%%%%%%%%-----%%%%%%%%%%



function Ymin = ypar (yplus,cuerda,Re,M,T0,eter)

%This function computes the minimum cell distance from a wall, according to equations for turbulent flow over a flat plate at zero incidence. It also computes some bonus data to define simulation parameters.

switch eter

  case 'a'
    T = T0/(1+0.2*M^2);
    V = M*sqrt(1.4*287.074*T);
    muT = 1.458e-6*T^1.5/(110.4+T);
    rho = Re*muT/(V*cuerda);
    P = rho*287.074*T;
    P0 = P*(T0/T)^(1.4/0.4);
    printf ('\n-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_\n\
    printf ('Medium: air\nRe = %g\nChord = %f [m]\nDensity = %g [Kg/m^3]\nDynamic viscosity = %g [Kg/(ms)]\nFreestream speed = %f [m/s]\nFreestream Mach = %f\nStatic pressure (absolute) = %f [Pa]\nStagnation pressure = %f [Pa]\nTemperature = %f [K]\nStagnation temeprature = %f[K]\nY+ = %f\n\n', Re,cuerda,rho,muT,V,M,P,P0,T,T0,yplus)

  case 'n'
    T = T0/(1+0.2*M^2);
    V = M*sqrt(1.4*297*T);
    muT = 1.781e-5*(111+300.55)/(111+T)*(T/300.55)^1.5;
    rho = Re*muT/(V*cuerda);
    P = rho*297*T;
    P0 = P*(T0/T)^(1.4/0.4);
    printf ('\n-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_\n\
    printf ('Medium: nitrogen\nRe = %g\nChord = %f [m]\nDensity = %g [Kg/m^3]\nDynamic viscosity = %g [Kg/(ms)]\nFreestream speed = %f [m/s]\nFreestream Mach = %f\nStatic pressure (absolute) = %f [Pa]\nStagnation pressure = %f [Pa]\nTemperature = %f [K]\nStagnation temperature = %f[K]\nY+ = %f\n\n', Re,cuerda,rho,muT,V,M,P,P0,T,T0,yplus)

  case 'h'
```

```
    V = M;
    rho = 1000*(1-(T0+288.9414)/(508929.2*(T0+68.12963))*(T0-3.9863)^2);
    muT = rho*V*cuerda/Re;
    printf
        ('\n-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_.\n\
    printf ('Medium: water\nRe = %g\nChord = %f [m]\nDensity = %g
        [Kg/m^3]\nDynamic viscosity = %g [Kg/(ms)]\nFreestream speed = %f
        [m/s]\nTemperature = %f [K]\nY+ = %f\n\n', Re,cuerda,rho,muT,V,yplus)

end

Cf = 0.02;

while i<10

  funcion = 4.15*sqrt(Cf)*log10(Re*Cf)+1.7*sqrt(Cf)-1.0;
  derfunc = (4.15*log10(exp(1.0))+0.5*4.15*log10(Re*Cf)+1.7/2.0)/sqrt(Cf);
  fsd = funcion/derfunc;

  if abs(fsd/Cf) <= exp(-5.0)
    break
  end

  Cfo = Cf-fsd;

  if Cfo <= 0.0
    Cf = 0.5*Cf;
  else
    Cf = Cfo;
  end

  i=i+1;

end

%Cfo = (1./(4.15*log10(Re*Cf)+1.7))^2;
%tau = 0.5*rho*V*V*Cf
%aus = sqrt(tau/rho)

Ymin = yplus*muT/(V*sqrt(Cf/2));

printf ('To obtain CFL(max) <= 20 across the whole flowfield, a timestep dt <=
    %.10gs is recomended for transient simulations.\n\n', 20*Ymin/V)
printf ('The following values are recomended to initialize external flowfield
    variables:\nK = %g [m^2/s^2]\nEpsilon = %g [m^2/s^3]\nOmega = %g
    [1/s]\nTurbulent intensity = 6.6667e-7 [%%]\nHydraulic diameter = %f [m]\n\n',
    1e-6*V^2,4.5e-7*V^3/cuerda,0.45*V/cuerda,0.0052164*cuerda)

end
```

Bibliography

[1] *OpenFOAM, the open source CFD toolbox User Guide*. OpenFOAM, 2013.

[2] B Mutlu Sumer and Jorgen Fredsoe. *Hydrodynamics around cylindrical structures*. World Scientific, 2006.