

AEM 5253—Computational Fluid Mechanics

Fall Semester 2022

Final Project

Suryanarayan(Surya) Ramachandran

July 23, 2023

A MATLAB code is written to solve the compressible Euler equations using Steger-Warming flux-splitting methodology in curvilinear (elliptic) coordinates. The code consists 7 subroutines:

1. `Euler_Finite_Diff.m`-the main solver routine
2. `F_plus.m`-computes the F^+ and G^+ flux locally.
3. `F_minus.m`-computes the F^- and G^- flux locally
4. `compute_dt.m`-computes the minimum stable dt possible for advance
5. `lambda_max.m`-helper function for `compute_dt.m` to retrieve max local eigen value
6. `post_process.m`-to generate contours
7. `compute_drag.m`-to compute drag coefficient

The contents of each of these routines are presented below:

1 Main solver:

Euler_Finite_Diff.m

```
1  %-----Author and course info-----%
2  % Name: Suryanarayan Ramachandran
3  % Course: AEM5253 Computational Fluid Mechanics
4  % Date: Dec 26th, 2022
5  % E-Mail: ramac106@umn.edu
6  % Course instructor: Dr.Pramod K Subbareddy
7
8  %-----Preamble-----%
9  clear all; close all; clc; %#ok<CLALL>
10
11 %-----Mesh generation-----%
12 %# of points along xi/eta
13 N=100;
14
15 %generate xi, eta
16 xi = linspace(0,2*pi,N);
17 eta = linspace(atanh(0.25),6.2,N);
18
19 %meshgrid to get structured mesh
20 [eta, xi] = meshgrid(eta, xi);
21
22 %get physical space coords
23 x = cosh(eta).*cos(xi);
24 y = sinh(eta).*sin(xi);
25
26 %-----Mesh Metrics-----%
27 %get Delta(xi), Delta(eta)
28 dxi = xi(2,1)-xi(1,1);
29 deta = eta(1,2)-eta(1,1);
30
31 %Declare arrays for mesh-metrics and Jacobians
32 xi_x = zeros(N,N);
33 xi_y = zeros(N,N);
34 eta_x = zeros(N,N);
35 eta_y = zeros(N,N);
36 J = zeros(N,N);
37
38 %Compute mesh-metrics and Jacobians
39 for i=1:N
40     for j=1:N
41         if(i>2 && i<=N-1)
42             x_xi = (x(i+1,j)-x(i-1,j))/2/dxi;
43             y_xi = (y(i+1,j)-y(i-1,j))/2/dxi;
44         elseif(i==1)
45             x_xi = (x(i+1,j)-x(i,j))/dxi;
46             y_xi = (y(i+1,j)-y(i,j))/dxi;
47         elseif(i==N)
48             x_xi = (x(i,j)-x(i-1,j))/dxi;
49             y_xi = (y(i,j)-y(i-1,j))/dxi;
50
51
52     end
53
54     if(j>2 && j<=N-1)
55         x_eta = (x(i,j+1)-x(i,j-1))/2/deta;
```

```

56         y_eta = (y(i,j+1)-y(i,j-1))/2/deta;
57     elseif(j==1)
58         x_eta = (x(i,j+1)-x(i,j))/deta;
59         y_eta = (y(i,j+1)-y(i,j))/deta;
60     elseif(j==N)
61         x_eta = (x(i,j)-x(i,j-1))/deta;
62         y_eta = (y(i,j)-y(i,j-1))/deta;
63     end
64
65     J(i,j) = 1/(x_xi*y_eta - x_eta*y_xi);
66     xi_x(i,j) = y_eta*J(i,j);
67     xi_y(i,j) = -y_xi*J(i,j);
68     eta_x(i,j) = -x_eta*J(i,j);
69     eta_y(i,j) = x_xi*J(i,j);
70 end
71 end
72
73 %-----Declare matrices for the solver-----%
74 %Time-level: n arrays
75 rho = zeros(N,N);
76 u = zeros(N,N);
77 v = zeros(N,N);
78 p = zeros(N,N);
79 %Time-level:n+1 arrays
80 rho_n = zeros(N,N);
81 u_n = zeros(N,N);
82 v_n = zeros(N,N);
83 p_n = zeros(N,N);
84
85 %matrices for storing norms
86 u_norm = [];
87 v_norm = [];
88 rho_norm = [];
89 p_norm = [];
90 time = [];
91
92 %-----Initial conditions-----%
93 %freestream conditions
94 M=6;
95 gamma = 1.4;
96 p_inflow = 1/gamma;
97 rho_inflow=1;
98 u_inflow = M*sqrt(gamma*p_inflow/rho_inflow);
99 cfl=0.5;
100
101 %Initialize
102 for i=1:N
103     for j=1:N
104         u(i,j)= u_inflow;
105         p(i,j)=p_inflow;
106         rho(i,j)=rho_inflow;
107         v(i,j)=0;
108     end
109 end
110
111 %-----Main Solution Loop-----%
112 %set time:
113 t=0;
114 %set max time of simulation:
115 t_max=10;

```

```

116
117 %Solve:
118 %Index notation followed: i for xi direction, j for eta direction
119 while(t<t_max)
120     %compute minimum dt:
121     dt_min = cfl*compute_dt(u,v,p,rho, xi_x, xi_y, eta_x, eta_y, dxi, deta);
122
123     %advance i=1 and i=N, in the interior j's
124     % For i=1, i+1=2, i-1 = N-1
125     for j=2:N-1
126         %Compute RHS
127         R_i_j =(F_plus(u(1,j),...
128                        v(1,j),...
129                        p(1,j),...
130                        rho(1,j),...
131                        xi_x(1,j),...
132                        xi_y(1,j),...
133                        J(1,j))...
134                -F_plus(u(N-1,j),...
135                        v(N-1,j),...
136                        p(N-1,j),...
137                        rho(N-1,j),...
138                        xi_x(N-1,j),...
139                        xi_y(N-1,j),...
140                        J(N-1,j))/dxi...
141                + (F_minus(u(2,j),...
142                        v(2,j),...
143                        p(2,j),...
144                        rho(2,j),...
145                        xi_x(2,j),...
146                        xi_y(2,j),...
147                        J(2,j))...
148                -F_minus(u(1,j),...
149                        v(1,j),...
150                        p(1,j),...
151                        rho(1,j),...
152                        xi_x(1,j),...
153                        xi_y(1,j),...
154                        J(1,j))/dxi...
155                + (F_plus(u(1,j),
156                        v(1,j),...
157                        p(1,j),...
158                        rho(1,j),...
159                        eta_x(1,j),...
160                        eta_y(1,j),...
161                        J(1,j))...
162                -F_plus(u(1,j-1),...
163                        v(1,j-1),...
164                        p(1,j-1),...
165                        rho(1,j-1),...
166                        eta_x(1,j-1),...
167                        eta_y(1,j-1),...
168                        J(1,j-1))/deta...
169                + (F_minus(u(1,j+1),...
170                        v(1,j+1),...
171                        p(1,j+1),...
172                        rho(1,j+1),...
173                        eta_x(1,j+1),...
174                        eta_y(1,j+1),...
175                        J(1,j+1))...

```

```

176         -F_minus(u(1,j),...
177                 v(1,j),...
178                 p(1,j),...
179                 rho(1,j),...
180                 eta_x(1,j),...
181                 eta_y(1,j),...
182                 J(1,j))/deta;
183
184     %Locally construct conserved state vector
185     U_old = [rho(1,j);...
186             rho(1,j)*u(1,j);...
187             rho(1,j)*v(1,j);...
188             rho(1,j)*(0.5*u(1,j)^2+0.5*v(1,j)^2)+p(1,j)/(gamma-1)];
189
190     %advance
191     U_new = U_old -dt_min*J(1,j)*R_ij;
192
193     %get back new primitive vars
194     rho_new = U_new(1);
195     u_new = U_new(2)/rho_new;
196     v_new = U_new(3)/rho_new;
197     q_new = 0.5*(u_new^2+v_new^2);
198     p_new = (gamma-1)*(U_new(4)-rho_new*q_new);
199
200     %assign to new arrays
201     % since xi-direction is periodic, we note i=1 and i=N are the same
202     rho_n(1,j) = rho_new;
203     rho_n(N,j) = rho_new;
204     u_n(1,j) = u_new;
205     u_n(N,j) = u_new;
206     v_n(1,j) = v_new;
207     v_n(N,j) = v_new;
208     p_n(1,j) = p_new;
209     p_n(N,j) = p_new;
210 end
211
212 %advance all other interior j's and i's i.e. 2:N-1
213 for j=2:N-1 %iterate through the interior ellipses-outer loop, we will fill ...
214     j=1, N later
215     for i=2:N-1 %iterate along the interior ellipses-inner loop, we will fill ...
216         i=1,N later
217         %compute RHS
218         R_ij = (F_plus(u(i,j),
219                       v(i,j),...
220                       p(i,j),...
221                       rho(i,j),...
222                       xi_x(i,j),...
223                       xi_y(i,j),...
224                       J(i,j))...
225               -F_plus(u(i-1,j),...
226                       v(i-1,j),...
227                       p(i-1,j),...
228                       rho(i-1,j),...
229                       xi_x(i-1,j),...
230                       xi_y(i-1,j),...
231                       J(i-1,j)))/dxi...
232               + (F_minus(u(i+1,j),...
233                       v(i+1,j),...
234                       p(i+1,j),...
235                       rho(i+1,j),...

```

```

234         xi_x(i+1,j),...
235         xi_y(i+1,j),...
236         J(i+1,j))...
237     -F_minus(u(i,j),...
238         v(i,j),...
239         p(i,j),...
240         rho(i,j),...
241         xi_x(i,j),...
242         xi_y(i,j),...
243         J(i,j))/dxi...
244     +(F_plus(u(i,j),...
245         v(i,j),...
246         p(i,j),...
247         rho(i,j),...
248         eta_x(i,j),...
249         eta_y(i,j),...
250         J(i,j))...
251     -F_plus(u(i,j-1),...
252         v(i,j-1),...
253         p(i,j-1),...
254         rho(i,j-1),...
255         eta_x(i,j-1),...
256         eta_y(i,j-1),...
257         J(i,j-1))/deta...
258     +(F_minus(u(i,j+1),...
259         v(i,j+1),...
260         p(i,j+1),...
261         rho(i,j+1),...
262         eta_x(i,j+1),...
263         eta_y(i,j+1),...
264         J(i,j+1))...
265     -F_minus(u(i,j),...
266         v(i,j),...
267         p(i,j),...
268         rho(i,j),...
269         eta_x(i,j),...
270         eta_y(i,j),...
271         J(i,j))/deta;
272
273     %construct U_old
274     U_old = [rho(i,j);...
275         rho(i,j)*u(i,j);...
276         rho(i,j)*v(i,j);...
277         0.5*rho(i,j)*(u(i,j)^2+v(i,j)^2)+p(i,j)/(gamma-1)];
278
279     %advance
280     U_new = U_old -dt_min*J(i,j)*R_ij;
281
282     %extract fields from solution
283     rho_new = U_new(1);
284     u_new = U_new(2)/rho_new;
285     v_new = U_new(3)/rho_new;
286     q_new = 0.5*(u_new^2+v_new^2);
287     p_new = (gamma-1)*(U_new(4)-rho_new*q_new);
288
289     %assign to new arrays:
290     rho_n(i,j) = rho_new;
291     u_n(i,j) = u_new;
292     v_n(i,j) = v_new;
293     p_n(i,j) = p_new;

```

```

294     end
295 end
296
297 %advance j=1 and j=N
298 for i=1:N
299     %j=1 i.e. airfoil surface
300     M = [xi_x(i,1) xi_y(i,1); eta_x(i,1) eta_y(i,1)];
301     D = [xi_x(i,2)*u_n(i,2)+xi_y(i,2)*v_n(i,2); 0];
302     A=M\D;
303     %get velocities
304     u_n(i,1)= A(1); v_n(i,1)=A(2);
305     %get density
306     rho_n(i,1) = rho_n(i,2);
307     %get pressure
308     q_1 = 0.5*(u_n(i,1)^2 + v_n(i,1)^2);
309     q_2 = 0.5*(u_n(i,2)^2 + v_n(i,2)^2);
310     p_n(i,1) = p_n(i,2) + (gamma-1)*(rho_n(i,2)*q_2-rho_n(i,1)*q_1);
311
312     %j=N, i.e. outer boundary
313     if(i>N/2 && i<=3*N/2)
314         %one half of the domain is an inlet
315         u_n(i,N)=u_inflow;
316         v_n(i,N)=0;
317         p_n(i,N)=p_inflow;
318         rho_n(i,N)=rho_inflow;
319     else
320         %other half is an outlet
321         u_n(i,N)=u_n(i,N-1);
322         v_n(i,N)=v_n(i,N-1);
323         p_n(i,N)=p_n(i,N-1);
324         rho_n(i,N)=rho_n(i,N-1);
325     end
326 end
327
328 %impose periodicity along i once again
329 u_n(N,:)=u_n(1,:);
330 v_n(N,:)=v_n(1,:);
331 p_n(N,:)=p_n(1,:);
332 rho_n(N,:)=rho_n(1,:);
333
334 %compute norms
335 L2_u = norm(u-u_n,2);
336 L2_v = norm(v-v_n,2);
337 L2_p = norm(p-p_n,2);
338 L2_rho = norm(rho-rho_n,2);
339
340 %update norm vectors
341 u_norm = [L2_u, u_norm];
342 v_norm = [L2_v, v_norm];
343 p_norm = [L2_p, p_norm];
344 rho_norm = [L2_rho, rho_norm];
345
346 %update solution arrays
347 u = u_n;
348 v = v_n;
349 p = p_n;
350 rho = rho_n;
351
352 %advance time
353 t = t+dt_min;

```

```

354     %display time
355     t
356     %store time
357     time = [t,time];
358 end
359
360 %plot residuals of primitive variables vs time
361 loglog(time,p_norm,...
362         time,rho_norm,...
363         time,u_norm,...
364         time,v_norm);
365 xlabel("time");legend("pressure","rho","u","v")

```

2 F^+, G^+

```

1     function fp = F_plus(u, v, p, rho, xi_x, xi_y,J)
2     %returns the 3 column  $F^+$  vector for the advance step
3     %all inputs are scalars hence values at a point and not vectors
4     r_xi = sqrt(xi_x^2+xi_y^2);
5     gamma = 1.4;
6     n_x = xi_x/r_xi;
7     n_y = xi_y/r_xi;
8     u_d = u*n_x + v*n_y;
9     q = (u^2+v^2)/2;
10    a = sqrt(gamma*p/rho);
11    h0 = q + a^2/(gamma-1);
12    e = 1e-6;
13    %construct lambda_plus matrix
14    L_p = zeros(4,4);
15    L_p(1,1) = r_xi*0.5*(u_d-a+sqrt((u_d-a)^2+e^2));
16    L_p(2,2) = r_xi*0.5*(u_d+sqrt(u_d^2+e^2));
17    L_p(3,3) = r_xi*0.5*(u_d+a+sqrt((u_d+a)^2+e^2));
18    L_p(4,4) = r_xi*0.5*(u_d+sqrt(u_d^2+e^2));
19
20    %construct left eigen vector
21    R = zeros(4,4);
22    R(1,1) = 1;
23    R(1,2) = 1;
24    R(1,3) = 1;
25
26    R(2,1) = u-a*n_x;
27    R(2,2) = u;
28    R(2,3) = u+a*n_x;
29    R(2,4) = n_y;
30
31    R(3,1) = v-a*n_y;
32    R(3,2) = v;
33    R(3,3) = v+a*n_y;
34    R(3,4) = -n_x;
35
36    R(4,1) = h0 - a*u_d;
37    R(4,2) = q;
38    R(4,3) = h0 + a*u_d;
39    R(4,4) = u*n_y - v*n_x;
40
41    %construct left-eigen vector

```



```

42     R_inv = zeros(4,4);
43
44     R_inv(1,1) = ((gamma-1)*q+a*u_d)/(2*a*a);
45     R_inv(1,2) = ((1-gamma)*u-a*n_x)/(2*a*a);
46     R_inv(1,3) = ((1-gamma)*v-a*n_y)/(2*a*a);
47     R_inv(1,4) = ((gamma-1))/(2*a*a);
48
49     R_inv(2,1) = (a^2-(gamma-1)*q)/(a*a);
50     R_inv(2,2) = ((gamma-1)*u)/(a*a);
51     R_inv(2,3) = ((gamma-1)*v)/(a*a);
52     R_inv(2,4) = ((1-gamma))/(a*a);
53
54     R_inv(3,1) = ((gamma-1)*q-a*u_d)/(2*a*a);
55     R_inv(3,2) = ((1-gamma)*u+a*n_x)/(2*a*a);
56     R_inv(3,3) = ((1-gamma)*v+a*n_y)/(2*a*a);
57     R_inv(3,4) = ((gamma-1))/(2*a*a);
58
59     R_inv(4,1) = v*n_x-u*n_y;
60     R_inv(4,2) = n_y;
61     R_inv(4,3) = -n_x;
62     R_inv(4,4) = 0;
63
64     %construct state vector:
65     U = [rho;rho*u;rho*v;rho*q+p/(gamma-1)];
66     U_Tilde = U/J;
67
68     A_p = R*L_p*R_inv;
69     fp = A_p*U_Tilde;
70 end

```

3 F^-, G^-

```

1     function fm = F_minus(u, v, p, rho, xi_x, xi_y, J)
2     %returns the 3 column  $F^+$  vector for the advance step
3     %all inputs are scalars hence values at a point and not vectors
4     r_xi = sqrt(xi_x^2+xi_y^2);
5     gamma = 1.4;
6     n_x = xi_x/r_xi;
7     n_y = xi_y/r_xi;
8     u_d = u*n_x + v*n_y;
9     q = (u^2+v^2)/2;
10    a = sqrt(gamma*p/rho);
11    h0 = q + a^2/(gamma-1);
12    e = 1e-6;
13    %construct lambda_plus matrix
14    L_m = zeros(4,4);
15    L_m(1,1) = r_xi*0.5*(u_d-a-sqrt((u_d-a)^2+e^2));
16    L_m(2,2) = r_xi*0.5*(u_d-sqrt(u_d^2+e^2));
17    L_m(3,3) = r_xi*0.5*(u_d+a-sqrt((u_d+a)^2+e^2));
18    L_m(4,4) = r_xi*0.5*(u_d-sqrt(u_d^2+e^2));
19
20    %construct left eigen vector
21    R = zeros(4,4);
22    R(1,1) = 1;
23    R(1,2) = 1;
24    R(1,3) = 1;

```

```

25     R(1,4) = 0;
26
27     R(2,1) = u-a*n_x;
28     R(2,2) = u;
29     R(2,3) = u+a*n_x;
30     R(2,4) = n_y;
31
32     R(3,1) = v-a*n_y;
33     R(3,2) = v;
34     R(3,3) = v+a*n_y;
35     R(3,4) = -n_x;
36
37     R(4,1) = h0 - a*u_d;
38     R(4,2) = q;
39     R(4,3) = h0 + a*u_d;
40     R(4,4) = u*n_y - v*n_x;
41
42     %construct left-egien vector
43     R_inv = zeros(4,4);
44
45     R_inv(1,1) = ((gamma-1)*q+a*u_d)/(2*a*a);
46     R_inv(1,2) = ((1-gamma)*u-a*n_x)/(2*a*a);
47     R_inv(1,3) = ((1-gamma)*v-a*n_y)/(2*a*a);
48     R_inv(1,4) = ((gamma-1))/(2*a*a);
49
50     R_inv(2,1) = (a^2-(gamma-1)*q)/(a*a);
51     R_inv(2,2) = ((gamma-1)*u)/(a*a);
52     R_inv(2,3) = ((gamma-1)*v)/(a*a);
53     R_inv(2,4) = ((1-gamma))/(a*a);
54
55     R_inv(3,1) = ((gamma-1)*q-a*u_d)/(2*a*a);
56     R_inv(3,2) = ((1-gamma)*u+a*n_x)/(2*a*a);
57     R_inv(3,3) = ((1-gamma)*v+a*n_y)/(2*a*a);
58     R_inv(3,4) = ((gamma-1))/(2*a*a);
59
60     R_inv(4,1) = v*n_x-u*n_y;
61     R_inv(4,2) = n_y;
62     R_inv(4,3) = -n_x;
63     R_inv(4,4) = 0;
64
65     %construct state vector:
66     U = [rho;rho*u;rho*v;rho*q+p/(gamma-1)];
67     U_Tilde = U/J;
68
69     A_m = R*L_m*R_inv;
70     fm = A_m*U_Tilde;
71 end

```

4 Compute stable dt

```
1 function dt = compute_dt(u,v,rho,p,xi_x, xi_y, eta_x, eta_y, dxi, deta)
2   N = length(u);
3   dt = 1e2;
4   for i = 1:N
5       for j=1:N
6           lmax = lambda_max(u(i,j),...
7                               v(i,j),...
8                               rho(i,j),...
9                               p(i,j),...
10                              xi_x(i,j),...
11                              xi_y(i,j),...
12                              eta_x(i,j),...
13                              eta_y(i,j));
14           dt_min = min(dxi/lmax(1),deta/lmax(2));
15           if(dt_min<dt)
16               dt = dt_min;
17           end
18       end
19   end
20 end
```

5 Get λ_{max}

```
1 function l_max = lambda_max(u,v, p, rho, xi_x, xi_y, eta_x, eta_y)
2   gamma = 1.4;
3   a = sqrt(gamma*p/rho);
4   if (~isreal(a))
5       a
6       rho
7       p
8       return;
9   end
10  l_xi = abs(xi_x*u+xi_y*v) + a*sqrt(xi_x^2 + xi_y^2);
11  l_eta = abs(eta_x*u+eta_y*v) + a*sqrt(eta_x^2 + eta_y^2);
12  l_max = [l_xi, l_eta];
13 end
```

6 Post Processing

post_process.m

```
1 contourf(x,y,u_n, 'LevelStep',0.1, 'LineStyle','--')
2 xlim([-1.5,1.5])
3 ylim([-1.5,1.5])
4 title("$u$", "Interpreter", "latex")
5 xlabel("$x$", "interpreter", "latex");
6 ylabel("$y$", "interpreter", "latex");
7 c=colorbar;
8 c.TickLabelInterpreter="latex";
```

```

9  set(groot, 'defaultAxesTickLabelInterpreter', 'latex');
10 set(gca, ...
11     "FontSize", 18, ...
12     "FontName", "Computer Modern Roman");
13 saveas(gcf, "Final_Project/u", "eps");
14
15 contourf(x,y,v_n, 'LevelStep', 0.1, 'LineStyle', '--')
16 xlim([-1.5, 1.5])
17 ylim([-1.5, 1.5])
18 title("$v$", "Interpreter", "latex")
19 xlabel("$x$", "interpreter", "latex");
20 ylabel("$y$", "interpreter", "latex");
21 c=colorbar;
22 c.TickLabelInterpreter="latex";
23 set(groot, 'defaultAxesTickLabelInterpreter', 'latex');
24 set(gca, ...
25     "FontSize", 18, ...
26     "FontName", "Computer Modern Roman");
27 saveas(gcf, "Final_Project/v", "eps");
28
29 contourf(x,y,sqrt(u_n.^2+v_n.^2), 'LevelStep', 0.1, 'LineStyle', '--')
30 xlim([-1.5, 1.5])
31 ylim([-1.5, 1.5])
32 title("$|U| = \sqrt{u^2+v^2}$", "Interpreter", "latex")
33 xlabel("$x$", "interpreter", "latex");
34 ylabel("$y$", "interpreter", "latex");
35 c=colorbar;
36 c.TickLabelInterpreter="latex";
37 set(groot, 'defaultAxesTickLabelInterpreter', 'latex');
38 set(gca, ...
39     "FontSize", 18, ...
40     "FontName", "Computer Modern Roman");
41 saveas(gcf, "Final_Project/velmag", "eps");
42
43 contourf(x,y,p_n, 'LevelStep', 0.1, 'LineStyle', '--')
44 xlim([-1.5, 1.5])
45 ylim([-1.5, 1.5])
46 title("$p$", "Interpreter", "latex")
47 xlabel("$x$", "interpreter", "latex");
48 ylabel("$y$", "interpreter", "latex");
49 c=colorbar;
50 c.TickLabelInterpreter="latex";
51 set(groot, 'defaultAxesTickLabelInterpreter', 'latex');
52 set(gca, ...
53     "FontSize", 18, ...
54     "FontName", "Computer Modern Roman");
55 saveas(gcf, "Final_Project/p", "eps");
56
57 contourf(x,y,rho_n, 'LevelStep', 0.1, 'LineStyle', '--')
58 xlim([-1.5, 1.5])
59 ylim([-1.5, 1.5])
60 title("$\rho$", "Interpreter", "latex")
61 xlabel("$x$", "interpreter", "latex");
62 ylabel("$y$", "interpreter", "latex");
63 c=colorbar;
64 c.TickLabelInterpreter="latex";
65 set(groot, 'defaultAxesTickLabelInterpreter', 'latex');
66 set(gca, "FontSize", 18, ...
67     "FontName", "Computer Modern Roman"); saveas(gcf, "Final_Project/rho", "eps");

```

7 Compute Drag Coefficient

compute_drag.m

```
1 M = 0.4; rho_inf = 1; U_inf = M; F_D = 0;
2 for i=1:N-1
3     M = [xi_x(i,1) xi_y(i,1); eta_x(i,1) eta_y(i,1)];
4     if (i==1)
5         D = [0; p_n(i,1)*sqrt((x(i+1,1)-x(N-1,1))^2+(y(i+1,1)-y(N-1,1))^2)];
6     else
7         D = [0; p_n(i,1)*sqrt((x(i+1,1)-x(i-1,1))^2+(y(i+1,1)-y(i-1,1))^2)];
8     end
9     A = M\D;
10    F_x_i = A(1);
11    F_D = F_D + F_x_i;
12 end
13
14 L=0;
15 for i=1:N-1
16     L = L+sqrt((x(i+1,1)-x(i,1))^2+(y(i+1,1)-y(i,1))^2);
17 end
18
19 C_D = F_D/0.5/rho_inf/U_inf^2;
```