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

```
%--------------------Author and course info----------------------------%
% Name: Suryanarayan Ramachandran
% Course: AEM5253 Computational Fluid Mechanics
% Date: Dec 26th, 2022
% E-Mail: ramac106@umn.edu
% Course instructor: Dr.Pramod K Subbareddy

%--------------------Preamble------------------------------------------%
clear all; close all; clc; %#ok<CLALL>

%---------------------Mesh generation----------------------------------%
%# of points along xi/eta
N=100;

%generate xi, eta
xi = linspace(0,2*pi,N);
eta = linspace(atanh(0.25),6.2,N);

%meshgrid to get structured mesh
[eta, xi] = meshgrid(eta, xi);

%get physical space coords
x = cosh(eta).*cos(xi);
y = sinh(eta).*sin(xi);

%--------------------Mesh Metrics--------------------------------------%
%get Delta(xi), Delta(eta)
dxi = xi(2,1)-xi(1,1);
deta = eta(1,2)-eta(1,1);

%Declare arrays for mesh-metrics and Jacobians
xi_x = zeros(N,N);
xi_y = zeros(N,N);
eta_x = zeros(N,N);
eta_y = zeros(N,N);
J = zeros(N,N);

%Compute mesh-metrics and Jacobians
for i=1:N
    for j=1:N
        if(i≥2 && i≤N-1)
            x_xi = (x(i+1,j)-x(i-1,j))/2/dxi;
            y_xi = (y(i+1,j)-y(i-1,j))/2/dxi;
        elseif(i==1)
            x_xi = (x(i+1,j)-x(i,j))/dxi;
            y_xi = (y(i+1,j)-y(i,j))/dxi;
        elseif(i==N)
            x_xi = (x(i,j)-x(i-1,j))/dxi;
            y_xi = (y(i,j)-y(i-1,j))/dxi;


        end

        if(j≥2 && j≤N-1)
            x_eta = (x(i,j+1)-x(i,j-1))/2/deta;
```

```
            y_eta = (y(i,j+1)-y(i,j-1))/2/deta;
        elseif(j==1)
            x_eta = (x(i,j+1)-x(i,j))/deta;
            y_eta = (y(i,j+1)-y(i,j))/deta;
        elseif(j==N)
            x_eta = (x(i,j)-x(i,j-1))/deta;
            y_eta = (y(i,j)-y(i,j-1))/deta;
        end

        J(i,j) = 1/(x_xi*y_eta - x_eta*y_xi);
        xi_x(i,j) = y_eta*J(i,j);
        xi_y(i,j) = -y_xi*J(i,j);
        eta_x(i,j)= -x_eta*J(i,j);
        eta_y(i,j)= x_xi*J(i,j);
    end
end

%----------------------Declare matrices for the solver--------------------%
%Time-level: n arrays
rho = zeros(N,N);
u = zeros(N,N);
v = zeros(N,N);
p = zeros(N,N);
%Time-level:n+1 arrays
rho_n = zeros(N,N);
u_n = zeros(N,N);
v_n = zeros(N,N);
p_n = zeros(N,N);

%matrices for storing norms
u_norm = [];
v_norm = [];
rho_norm = [];
p_norm = [];
time = [];

%--------------------Initial conditions-------------------------------%
%freestream conditions
M=6;
gamma = 1.4;
p_inflow = 1/gamma;
rho_inflow=1;
u_inflow = M*sqrt(gamma*p_inflow/rho_inflow);
cfl=0.5;

%Initialize
for i=1:N
    for j=1:N
        u(i,j)= u_inflow;
        p(i,j)=p_inflow;
        rho(i,j)=rho_inflow;
        v(i,j)=0;
    end
end

%------------------Main Solution Loop--------------------------------%
%set time:
t=0;
%set max time of simulation:
t_max=10;
```

```

%Solve:
%Index notation followed: i for xi direction, j for eta direction
while(t<t_max)
    %compute minimum dt:
    dt_min = cfl*compute_dt(u,v,p,rho, xi_x, xi_y, eta_x, eta_y, dxi, deta);

    %advance i=1 and i=N, in the interior j's
    % For i=1, i+1=2, i-1 = N-1
    for j=2:N-1
            %Compute RHS
            R_ij =(F_plus(u(1,j),...
                          v(1,j),...
                          p(1,j),...
                          rho(1,j),...
                          xi_x(1,j),...
                          xi_y(1,j),...
                          J(1,j))...
                  -F_plus(u(N-1,j),...
                          v(N-1,j),...
                          p(N-1,j),...
                          rho(N-1,j),...
                          xi_x(N-1,j),...
                          xi_y(N-1,j),...
                          J(N-1,j)))/dxi...
                  +(F_minus(u(2,j),...
                            v(2,j),...
                            p(2,j),...
                            rho(2,j),...
                            xi_x(2,j),...
                            xi_y(2,j),...
                            J(2,j))...
                    -F_minus(u(1,j),...
                            v(1,j),...
                            p(1,j),...
                            rho(1,j),...
                            xi_x(1,j),...
                            xi_y(1,j),...
                            J(1,j)))/dxi...
                    +(F_plus(u(1,j),
                            v(1,j),...
                            p(1,j),...
                            rho(1,j),...
                            eta_x(1,j),...
                            eta_y(1,j),...
                            J(1,j))...
                    -F_plus(u(1,j-1),...
                            v(1,j-1),...
                            p(1,j-1),...
                            rho(1,j-1),...
                            eta_x(1,j-1),...
                            eta_y(1,j-1),...
                            J(1,j-1)))/deta...
                    +(F_minus(u(1,j+1),...
                            v(1,j+1),...
                            p(1,j+1),...
                            rho(1,j+1),...
                            eta_x(1,j+1),...
                            eta_y(1,j+1),...
                            J(1,j+1))...
```

```
                        -F_minus(u(1,j),...
                                 v(1,j),...
                                 p(1,j),...
                                 rho(1,j),...
                                 eta_x(1,j),...
                                 eta_y(1,j),...
                                 J(1,j)))/deta;

                %Locally construct conserved state vector
                U_old = [rho(1,j);...
                         rho(1,j)*u(1,j);...
                         rho(1,j)*v(1,j);...
                         rho(1,j)*(0.5*u(1,j)^2+0.5*v(1,j)^2)+p(1,j)/(gamma-1)];

                %advance
                U_new = U_old -dt_min*J(1,j)*R_ij;

                %get back new primitive vars
                rho_new = U_new(1);
                u_new = U_new(2)/rho_new;
                v_new = U_new(3)/rho_new;
                q_new = 0.5*(u_new^2+v_new^2);
                p_new = (gamma-1)*(U_new(4)-rho_new*q_new);

                %assign to new arrays
                % since xi-direction is periodic, we note i=1 and i=N are the same
                rho_n(1,j) = rho_new;
                rho_n(N,j) = rho_new;
                u_n(1,j) = u_new;
                u_n(N,j) = u_new;
                v_n(1,j) = v_new;
                v_n(N,j) = v_new;
                p_n(1,j) = p_new;
                p_n(N,j) = p_new;
        end

        %advance all other interior j's and i's i.e. 2:N-1
        for j=2:N-1 %iterate through the interior ellipses-outer loop, we will fill ... j=1, N later
            for i=2:N-1 %iterate along the interior ellipses-inner loop, we will fill ... i=1,N later
                %compute RHS
                R_ij =(F_plus(u(i,j),
                            v(i,j),...
                            p(i,j),...
                            rho(i,j),...
                            xi_x(i,j),...
                            xi_y(i,j),...
                            J(i,j))...
                      -F_plus(u(i-1,j),...
                              v(i-1,j),...
                              p(i-1,j),...
                              rho(i-1,j),...
                              xi_x(i-1,j),...
                              xi_y(i-1,j),...
                              J(i-1,j)))/dxi...
                      +(F_minus(u(i+1,j),...
                                v(i+1,j),...
                                p(i+1,j),...
                                rho(i+1,j),...
```

```
                            xi_x(i+1,j),...
                            xi_y(i+1,j),...
                            J(i+1,j))...
                   -F_minus(u(i,j),...
                            v(i,j),...
                            p(i,j),...
                            rho(i,j),...
                            xi_x(i,j),...
                            xi_y(i,j),...
                            J(i,j)))/dxi...
                  +(F_plus(u(i,j),...
                           v(i,j),...
                           p(i,j),...
                           rho(i,j),...
                           eta_x(i,j),...
                           eta_y(i,j),...
                           J(i,j))...
                   -F_plus(u(i,j-1),...
                           v(i,j-1),...
                           p(i,j-1),...
                           rho(i,j-1),...
                           eta_x(i,j-1),...
                           eta_y(i,j-1),...
                           J(i,j-1)))/deta...
                  +(F_minus(u(i,j+1),...
                            v(i,j+1),...
                            p(i,j+1),...
                            rho(i,j+1),...
                            eta_x(i,j+1),...
                            eta_y(i,j+1),...
                            J(i,j+1))...
                   -F_minus(u(i,j),...
                            v(i,j),...
                            p(i,j),...
                            rho(i,j),...
                            eta_x(i,j),...
                            eta_y(i,j),...
                            J(i,j)))/deta;

            %construct U_old
            U_old = [rho(i,j);...
                     rho(i,j)*u(i,j);...
                     rho(i,j)*v(i,j);...
                     0.5*rho(i,j)*(u(i,j)^2+v(i,j)^2)+p(i,j)/(gamma-1)];

            %advance
            U_new = U_old -dt_min*J(i,j)*R_ij;

            %extract fields from solution
            rho_new = U_new(1);
            u_new = U_new(2)/rho_new;
            v_new = U_new(3)/rho_new;
            q_new = 0.5*(u_new^2+v_new^2);
            p_new = (gamma-1)*(U_new(4)-rho_new*q_new);

            %assign to new arrays:
            rho_n(i,j) = rho_new;
            u_n(i,j) = u_new;
            v_n(i,j) = v_new;
            p_n(i,j) = p_new;
```

```
        end
    end

    %advance j=1 and j=N
    for i=1:N
       %j=1 i.e. airfoil surface
       M = [xi_x(i,1) xi_y(i,1);eta_x(i,1) eta_y(i,1)];
       D = [xi_x(i,2)*u_n(i,2)+xi_y(i,2)*v_n(i,2); 0];
       A=M\D;
       %get velocities
       u_n(i,1)= A(1); v_n(i,1)=A(2);
       %get density
       rho_n(i,1) = rho_n(i,2);
       %get pressure
       q_1 = 0.5*(u_n(i,1)^2 + v_n(i,1)^2);
       q_2 = 0.5*(u_n(i,2)^2 + v_n(i,2)^2);
       p_n(i,1) = p_n(i,2) + (gamma-1)*(rho_n(i,2)*q_2-rho_n(i,1)*q_1);

       %j=N, i.e. outer boundary
        if(i>N/2 && i≤3*N/2)
            %one half of the domain is an inlet
            u_n(i,N)=u_inflow;
            v_n(i,N)=0;
            p_n(i,N)=p_inflow;
            rho_n(i,N)=rho_inflow;
        else
            %other half is an outlet
            u_n(i,N)=u_n(i,N-1);
            v_n(i,N)=v_n(i,N-1);
            p_n(i,N)=p_n(i,N-1);
            rho_n(i,N)=rho_n(i,N-1);
        end
    end

    %impose periodicity along i once again
    u_n(N,:)=u_n(1,:);
    v_n(N,:)=v_n(1,:);
    p_n(N,:)=p_n(1,:);
    rho_n(N,:)=rho_n(1,:);

    %compute norms
    L2_u = norm(u-u_n,2);
    L2_v = norm(v-v_n,2);
    L2_p = norm(p-p_n,2);
    L2_rho = norm(rho-rho_n,2);

    %update norm vectors
    u_norm = [L2_u, u_norm];
    v_norm = [L2_v, v_norm];
    p_norm = [L2_p, p_norm];
    rho_norm = [L2_rho, rho_norm];

    %update solution arrays
    u = u_n;
    v = v_n;
    p = p_n;
    rho = rho_n;

    %advance time
    t = t+dt_min;
```

```
    %display time
    t
    %store time
    time = [t,time];
end

%plot residuals of primitive variables vs time
loglog(time,p_norm,...
        time,rho_norm,...
        time,u_norm,...
        time,v_norm);
xlabel("time");legend("pressure","rho","u","v")
```

2 F^+, G^+

```
    function fp = F_plus(u, v, p, rho, xi_x, xi_y,J)
    %returns the 3 column F^+ vector for the advance step
    %all inputs are scalars hence values at a point and not vectors
    r_xi = sqrt(xi_x^2+xi_y^2);
    gamma = 1.4;
    n_x = xi_x/r_xi;
    n_y = xi_y/r_xi;
    u_d = u*n_x + v*n_y;
    q = (u^2+v^2)/2;
    a = sqrt(gamma*p/rho);
    h0 = q + a^2/(gamma-1);
    e = 1e-6;
    %construct lambda_plus matrix
    L_p = zeros(4,4);
    L_p(1,1) = r_xi*0.5*(u_d-a+sqrt((u_d-a)^2+e^2));
    L_p(2,2) = r_xi*0.5*(u_d+sqrt(u_d^2+e^2));
    L_p(3,3) = r_xi*0.5*(u_d+a+sqrt((u_d+a)^2+e^2));
    L_p(4,4) = r_xi*0.5*(u_d+sqrt(u_d^2+e^2));

    %construct left eigen vector
    R = zeros(4,4);
    R(1,1) = 1;
    R(1,2) = 1;
    R(1,3) = 1;

    R(2,1) = u-a*n_x;
    R(2,2) = u;
    R(2,3) = u+a*n_x;
    R(2,4) = n_y;

    R(3,1) = v-a*n_y;
    R(3,2) = v;
    R(3,3) = v+a*n_y;
    R(3,4) = -n_x;

    R(4,1) = h0 - a*u_d;
    R(4,2) = q;
    R(4,3) = h0 + a*u_d;
    R(4,4) = u*n_y - v*n_x;

    %construct left-egien vector
```

```
    R_inv = zeros(4,4);

    R_inv(1,1) = ((gamma-1)*q+a*u_d)/(2*a*a);
    R_inv(1,2) = ((1-gamma)*u-a*n_x)/(2*a*a);
    R_inv(1,3) = ((1-gamma)*v-a*n_y)/(2*a*a);
    R_inv(1,4) = ((gamma-1))/(2*a*a);

    R_inv(2,1) = (a^2-(gamma-1)*q)/(a*a);
    R_inv(2,2) = ((gamma-1)*u)/(a*a);
    R_inv(2,3) = ((gamma-1)*v)/(a*a);
    R_inv(2,4) = ((1-gamma))/(a*a);

    R_inv(3,1) = ((gamma-1)*q-a*u_d)/(2*a*a);
    R_inv(3,2) = ((1-gamma)*u+a*n_x)/(2*a*a);
    R_inv(3,3) = ((1-gamma)*v+a*n_y)/(2*a*a);
    R_inv(3,4) = ((gamma-1))/(2*a*a);

    R_inv(4,1) = v*n_x-u*n_y;
    R_inv(4,2) = n_y;
    R_inv(4,3) = -n_x;
    R_inv(4,4) = 0;

    %construct state vector:
    U = [rho;rho*u;rho*v;rho*q+p/(gamma-1)];
    U_Tilde = U/J;

    A_p = R*L_p*R_inv;
    fp = A_p*U_Tilde;
end
```

3 F^-, G^-

```
    function fm = F_minus(u, v, p, rho, xi_x, xi_y,J)
    %returns the 3 column F^+ vector for the advance step
    %all inputs are scalars hence values at a point and not vectors
    r_xi = sqrt(xi_x^2+xi_y^2);
    gamma = 1.4;
    n_x = xi_x/r_xi;
    n_y = xi_y/r_xi;
    u_d = u*n_x + v*n_y;
    q = (u^2+v^2)/2;
    a = sqrt(gamma*p/rho);
    h0 = q + a^2/(gamma-1);
    e = 1e-6;
    %construct lambda_plus matrix
    L_m = zeros(4,4);
    L_m(1,1) = r_xi*0.5*(u_d-a-sqrt((u_d-a)^2+e^2));
    L_m(2,2) = r_xi*0.5*(u_d-sqrt(u_d^2+e^2));
    L_m(3,3) = r_xi*0.5*(u_d+a-sqrt((u_d+a)^2+e^2));
    L_m(4,4) = r_xi*0.5*(u_d-sqrt(u_d^2+e^2));

    %construct left eigen vector
    R = zeros(4,4);
    R(1,1) = 1;
    R(1,2) = 1;
    R(1,3) = 1;
```

```
    R(1,4) = 0;

    R(2,1) = u-a*n_x;
    R(2,2) = u;
    R(2,3) = u+a*n_x;
    R(2,4) = n_y;

    R(3,1) = v-a*n_y;
    R(3,2) = v;
    R(3,3) = v+a*n_y;
    R(3,4) = -n_x;

    R(4,1) = h0 - a*u_d;
    R(4,2) = q;
    R(4,3) = h0 + a*u_d;
    R(4,4) = u*n_y - v*n_x;

    %construct left-egien vector
    R_inv = zeros(4,4);

    R_inv(1,1) = ((gamma-1)*q+a*u_d)/(2*a*a);
    R_inv(1,2) = ((1-gamma)*u-a*n_x)/(2*a*a);
    R_inv(1,3) = ((1-gamma)*v-a*n_y)/(2*a*a);
    R_inv(1,4) = ((gamma-1))/(2*a*a);

    R_inv(2,1) = (a^2-(gamma-1)*q)/(a*a);
    R_inv(2,2) = ((gamma-1)*u)/(a*a);
    R_inv(2,3) = ((gamma-1)*v)/(a*a);
    R_inv(2,4) = ((1-gamma))/(a*a);

    R_inv(3,1) = ((gamma-1)*q-a*u_d)/(2*a*a);
    R_inv(3,2) = ((1-gamma)*u+a*n_x)/(2*a*a);
    R_inv(3,3) = ((1-gamma)*v+a*n_y)/(2*a*a);
    R_inv(3,4) = ((gamma-1))/(2*a*a);

    R_inv(4,1) = v*n_x-u*n_y;
    R_inv(4,2) = n_y;
    R_inv(4,3) = -n_x;
    R_inv(4,4) = 0;

    %construct state vector:
    U = [rho;rho*u;rho*v;rho*q+p/(gamma-1)];
    U_Tilde = U/J;

    A_m = R*L_m*R_inv;
    fm = A_m*U_Tilde;
end
```

4 Compute stable dt

```
function dt = compute_dt(u,v,rho,p,xi_x, xi_y, eta_x, eta_y, dxi, deta)
 N = length(u);
 dt = 1e2;
 for i = 1:N
        for j=1:N
            lmax = lambda_max(u(i,j),...
                              v(i,j),...
                              rho(i,j),...
                              p(i,j),...
                              xi_x(i,j),...
                              xi_y(i,j),...
                              eta_x(i,j),...
                              eta_y(i,j));
            dt_min = min(dxi/lmax(1),deta/lmax(2));
            if(dt_min<dt)
                dt = dt_min;
            end
        end
 end
end
```

5 Get λ_{max}

```
function l_max = lambda_max(u,v, p, rho, xi_x, xi_y, eta_x, eta_y)
    gamma = 1.4;
    a = sqrt(gamma*p/rho);
    if (¬isreal(a))
        a
        rho
        p
        return;
    end
    l_xi = abs(xi_x*u+xi_y*v) + a*sqrt(xi_x^2 + xi_y^2);
    l_eta = abs(eta_x*u+eta_y*v) + a*sqrt(eta_x^2 + eta_y^2);
    l_max = [l_xi, l_eta];
end
```

6 Post Processing

`post_process.m`

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

```
set(groot,'defaultAxesTickLabelInterpreter','latex');
set(gca,...
    "FontSize", 18, ...
    "FontName", "Computer Modern Roman");
saveas(gcf,"Final_Project/u","epsc");

contourf(x,y,v_n,'LevelStep',0.1,'LineStyle','--')
xlim([-1.5,1.5])
ylim([-1.5,1.5])
title("$v$","Interpreter","latex")
xlabel("$x$","interpreter","latex");
ylabel("$y$","interpreter","latex");
c=colorbar;
c.TickLabelInterpreter="latex";
set(groot,'defaultAxesTickLabelInterpreter','latex');
set(gca,...
    "FontSize", 18, ...
    "FontName", "Computer Modern Roman");
saveas(gcf,"Final_Project/v","epsc");

contourf(x,y,sqrt(u_n.^2+v_n.^2),'LevelStep',0.1,'LineStyle','--')
xlim([-1.5,1.5])
ylim([-1.5,1.5])
title("$|U| = \sqrt{u^2+v^2}$","Interpreter","latex")
xlabel("$x$","interpreter","latex");
ylabel("$y$","interpreter","latex");
c=colorbar;
c.TickLabelInterpreter="latex";
set(groot,'defaultAxesTickLabelInterpreter','latex');
set(gca,...
    "FontSize", 18, ...
    "FontName", "Computer Modern Roman");
saveas(gcf,"Final_Project/velmag","epsc");

contourf(x,y,p_n,'LevelStep',0.1,'LineStyle','--')
xlim([-1.5,1.5])
ylim([-1.5,1.5])
title("$p$","Interpreter","latex")
xlabel("$x$","interpreter","latex");
ylabel("$y$","interpreter","latex");
c=colorbar;
c.TickLabelInterpreter="latex";
set(groot,'defaultAxesTickLabelInterpreter','latex');
set(gca,...
    "FontSize", 18, ...
    "FontName", "Computer Modern Roman");
saveas(gcf,"Final_Project/p","epsc");

contourf(x,y,rho_n,'LevelStep',0.1,'LineStyle','--')
xlim([-1.5,1.5])
ylim([-1.5,1.5])
title("$\rho$","Interpreter","latex")
xlabel("$x$","interpreter","latex");
ylabel("$y$","interpreter","latex");
c=colorbar;
c.TickLabelInterpreter="latex";
set(groot,'defaultAxesTickLabelInterpreter','latex');
set(gca,"FontSize", 18, ...
    "FontName", "Computer Modern Roman"); saveas(gcf,"Final_Project/rho","epsc");
```

7 Compute Drag Coefficient

compute_drag.m

```
M = 0.4; rho_inf = 1; U_inf = M; F_D =0;
for i=1:N-1
    M = [xi_x(i,1) xi_y(i,1);eta_x(i,1) eta_y(i,1)];
    if (i==1)
        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)];
    else
        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)];
    end
    A = M\D;
    F_x_i = A(1);
    F_D = F_D + F_x_i;
end

L=0;
for i=1:N-1
    L = L+sqrt((x(i+1,1)-x(i,1))^2+(y(i+1,1)-y(i,1))^2);
end

C_D = F_D/0.5/rho_inf/U_inf^2;
```