AAE 694: Computational Aeroacoustics

Project 3

Michael Crawley

3/10/2011

Introduction

In this project, the two dimensional linearized Euler equations are solved numerically given initial conditions and a mean flow Mach number along a solid wall. This numerical solution is accomplished using an explicit finite difference method; the coefficients for the spatial derivatives derived from Tam's fourth order Dispersion-Relation-Preserving scheme, and the coefficients for the temporal derivatives are derived from a third order optimized scheme. The nondimensionalized governing equations for this study are therefore

$$\frac{\partial \rho}{\partial t} = -\frac{\partial}{\partial x}(M_x \rho + u) - \frac{\partial v}{\partial y}$$
$$\frac{\partial u}{\partial t} = -\frac{\partial}{\partial x}(M_x u + p)$$
$$\frac{\partial v}{\partial t} = -\frac{\partial}{\partial x}(M_x v) - \frac{\partial p}{\partial y}$$
$$\frac{\partial p}{\partial t} = -\frac{\partial}{\partial x}(M_x p + u) - \frac{\partial v}{\partial y}$$

These equations are subsequently discretized as

$$\rho_{l,m}^{n+1} = \rho_{l,m}^{n} + \Delta t \sum_{j=0}^{3} b_j \left\{ - \sum_{j=-3}^{3} a_j \left(M_x \rho_{l+j,m}^{n-j} + u_{l+j,m}^{n-j} \right) - \sum_{j=-3}^{3} a_j v_{l,m+j}^{n-j} \right\}$$
$$u_{l,m}^{n+1} = u_{l,m}^{n} + \Delta t \sum_{j=0}^{3} b_j \left\{ - \sum_{j=-3}^{3} a_j \left(M_x u_{l+j,m}^{n-j} + p_{l+j,m}^{n-j} \right) \right\}$$
$$v_{l,m}^{n+1} = v_{l,m}^{n} + \Delta t \sum_{j=0}^{3} b_j \left\{ - \sum_{j=-3}^{3} a_j \left(M_x v_{l+j,m}^{n-j} \right) - \sum_{j=-3}^{3} a_j p_{l,m+k}^{n-j} \right\}$$
$$p_{l,m}^{n+1} = p_{l,m}^{n} + \Delta t \sum_{j=0}^{3} b_j \left\{ - \sum_{j=-3}^{3} a_j \left(M_x p_{l+j,m}^{n-j} + u_{l+j,m}^{n-j} \right) - \sum_{j=-3}^{3} a_j v_{l,m+k}^{n-j} \right\}$$

The initial conditions for this simulation take the form of a single perturbation in the nondimensionalized density and pressure fields, centered at (0,20); no perturbation initially exists in the velocity fields.

Boundary Conditions

Special care must be taken at the boundaries of the domain to ensure that the disturbances properly evolve as they reach the edges of the computational domain. In order to accomplish this, radiation (for acoustic only waves) and outflow boundary (for acoustic, entropy and vorticity waves) conditions are imposed at the left, right and top boundaries of the computational domain. The lower boundary is specified to be a solid wall, where the normal velocity at the wall is zero.

Radiation

In the derivation of the radiation boundary conditions, it is assumed that the boundaries of the computational domain are sufficiently far from the acoustic source so that they may be considered to be

in the source's far field. The radiation boundary conditions can be found from the asymptotic solution to the linearized Euler equations, and in nondimensionalized form are therefore

$$\frac{\partial \phi}{\partial t} = -A\frac{\partial \phi}{\partial x} - B\frac{\partial \phi}{\partial y} - C\phi$$
$$\phi = \{\rho; u; v; p\}$$
$$A = \cos\theta\left[M_x\cos\theta + \sqrt{1-(M_x\sin\theta)^2}\right]$$
$$B = \sin\theta\left[M_x\cos\theta + \sqrt{1-(M_x\sin\theta)^2}\right]$$
$$C = \frac{1}{2}(\mathrm{x}^2+\mathrm{y}^2)^{-1/2}\left[M_x\cos\theta + \sqrt{1-(M_x\sin\theta)^2}\right]$$

These boundary conditions can therefore be discretized as

$$\phi_{l,m}^{n+1} = \phi_{l,m}^{n} + \Delta t\sum_{j=0}^{3} b_j\left\{-A_{l,m}\sum_{j=-3}^{3} a_j\phi_{l+j,m}^{n-j} - B_{l,m}\sum_{j=-3}^{3} a_j\phi_{l,m+j}^{n-j} - C_{l,m}\phi_{l,m}\right\}$$

Outflow

Unlike the far field region, the outflow region contains entropy and vorticity waves in addition to the acoustic waves. The outflow boundary conditions can therefore be found from the asymptotic solutions to the acoustic, entropy and vorticity waves. In nondimensionalized form, they are

$$\frac{\partial \rho}{\partial t} = \frac{\partial}{\partial x}(p - M_x\rho) + \frac{\partial p}{\partial y} - A\frac{\partial p}{\partial x} - B\frac{\partial p}{\partial y} - Cp$$
$$\frac{\partial u}{\partial t} = -\frac{\partial}{\partial x}(M_x u + p)$$
$$\frac{\partial v}{\partial t} = -\frac{\partial}{\partial x}(M_x v) - \frac{\partial p}{\partial y}$$
$$\frac{\partial p}{\partial t} = -A\frac{\partial p}{\partial x} - B\frac{\partial p}{\partial y} - Cp$$

These equations are then discretized as

$$\rho_{l,m}^{n+1} = \rho_{l,m}^{n} + \Delta t\sum_{j=0}^{3} b_j\left\{\sum_{j=-3}^{3} a_j\left(p_{l+j,m}^{n-j} - M_x\rho_{l+j,m}^{n-j}\right) - \sum_{j=-3}^{3} a_j p_{l,m+j}^{n-j} - A_{l,m}\sum_{j=-3}^{3} a_j p_{l+j,m}^{n-j} - B_{l,m}\sum_{j=-3}^{3} a_j p_{l,m+j}^{n-j} - C_{l,m}p_{l,m}\right\}$$

$$u_{l,m}^{n+1} = u_{l,m}^{n} + \Delta t\sum_{j=0}^{3} b_j\left\{-\sum_{j=-3}^{3} a_j\left(M_x u_{l+j,m}^{n-j} + p_{l+j,m}^{n-j}\right)\right\}$$

$$v_{l,m}^{n+1} = v_{l,m}^{n} + \Delta t \sum_{j=0}^{3} b_j \left\{ - \sum_{j=-3}^{3} a_j \left(M_x v_{l+j,m}^{n-j} \right) - \sum_{j=-3}^{3} a_j p_{l,m+k}^{n-j} \right\}$$

$$p_{l,m}^{n+1} = p_{l,m}^{n} + \Delta t \sum_{j=0}^{3} b_j \left\{ -A_{l,m} \sum_{j=-3}^{3} a_j p_{l+j,m}^{n-j} - B_{l,m} \sum_{j=-3}^{3} a_j p_{l,m+j}^{n-j} - C_{l,m} p_{l,m} \right\}$$

Solid Wall

At the solid wall boundary, the normal velocity to the wall is driven to zero (as this simulation is for an inviscid flow, the parallel velocity is unaffected by the wall). In the physical realm, this condition is accomplished by the pressure the wall exerts on the fluid; likewise, in this simulation the pressure field was modified to produce the same result. This is achieved by the addition of a ghost point below the wall to the pressure field, the value of this ghost point being set so that $\frac{\partial p}{\partial y} = 0$ at the wall.

Grid Resolution

The effects of the spatial and temporal grid resolution are shown in Figure 1, where the density contours for two time periods and two different spatial/temporal grid resolutions are shown. As the results for the fine and coarse grids are indistinguishable, the coarse grid will be used throughout the rest of this study to reduce computational costs.

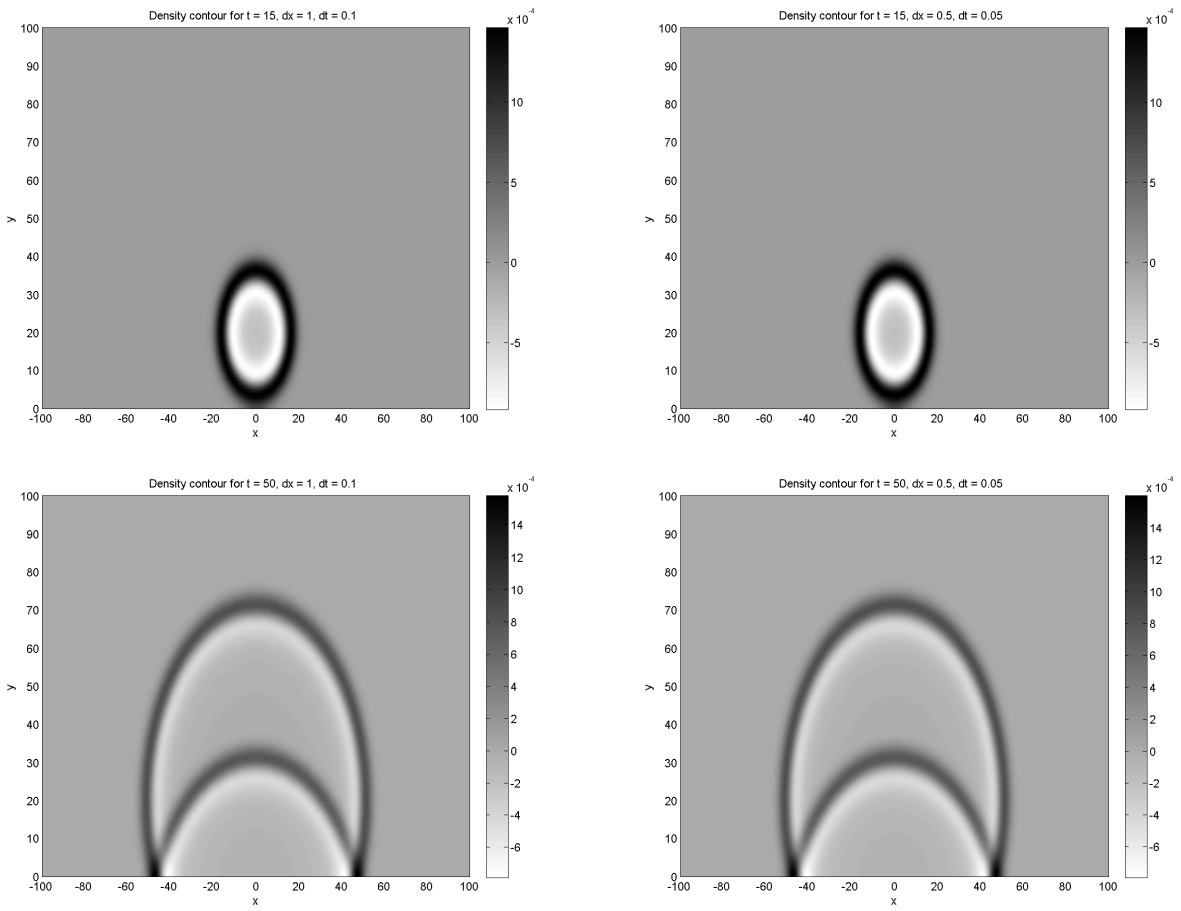

Figure 1: Density contours at 15 and 50 seconds for two different spatial-temporal grid resolutions.

Results

Case 1

For the first case, the mean flow Mach number along the solid wall is set to zero and the disturbance wave propagates in all directions equally. Figures 2, 3, 4 and 5 show the density, pressure, x-direction velocity, and y-direction velocity fields, respectively. From these plots, it is clear that the numerical scheme is accurately replicating the evolution of the disturbance. The high amplitude perturbation evolves into an expanding wave front with decreasing amplitude, which expands at sonic speed, while the center of the wave front remains stationary. As the wave front comes in contact with the solid wall boundary, it is reflected and begins traveling in the opposite direction of its initial propagation. As this simulation is for inviscid flow, the energy of the wave is conserved throughout the propagation and reflection processes.

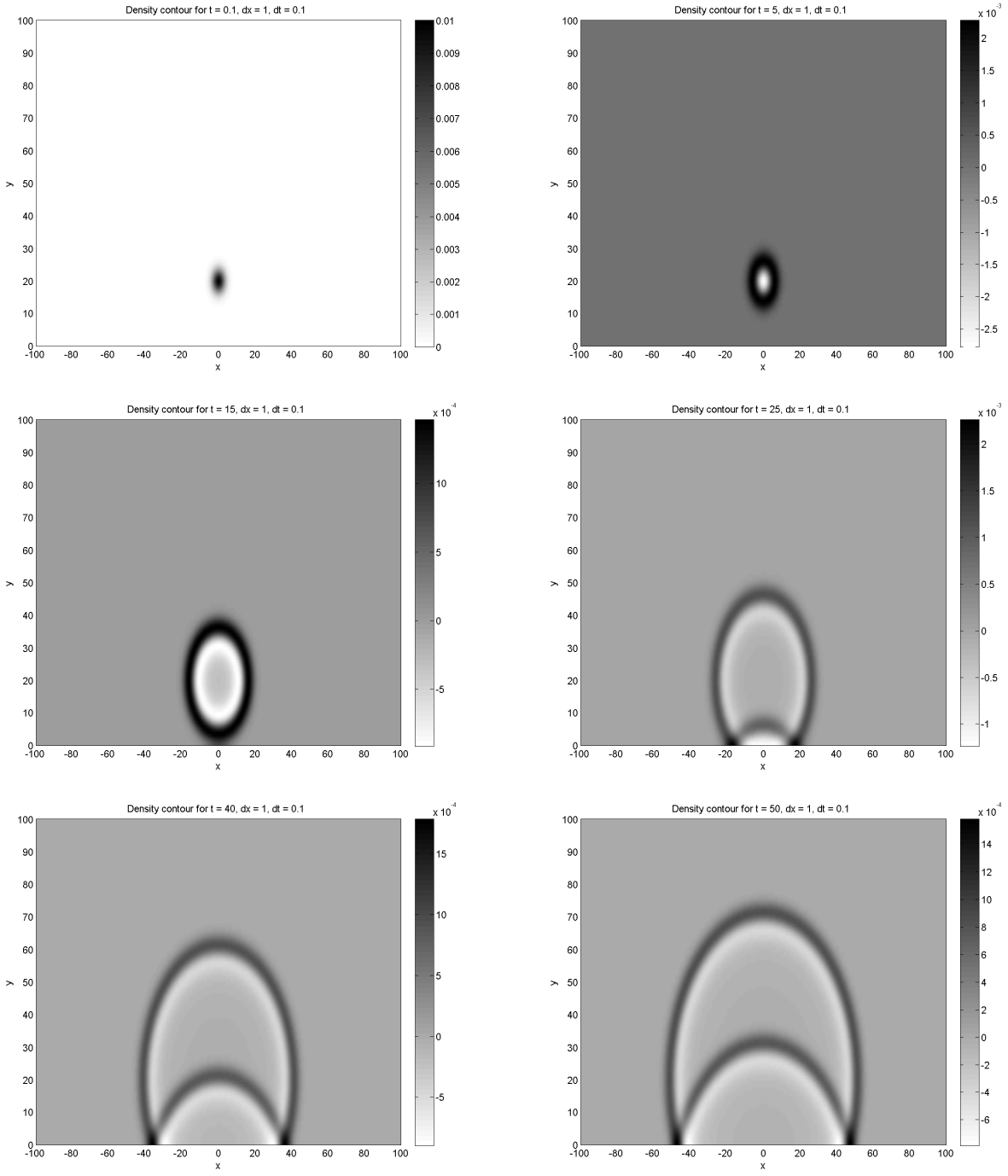

Figure 2: Density contour at 0.1, 5, 15, 25, 40 and 50 seconds.

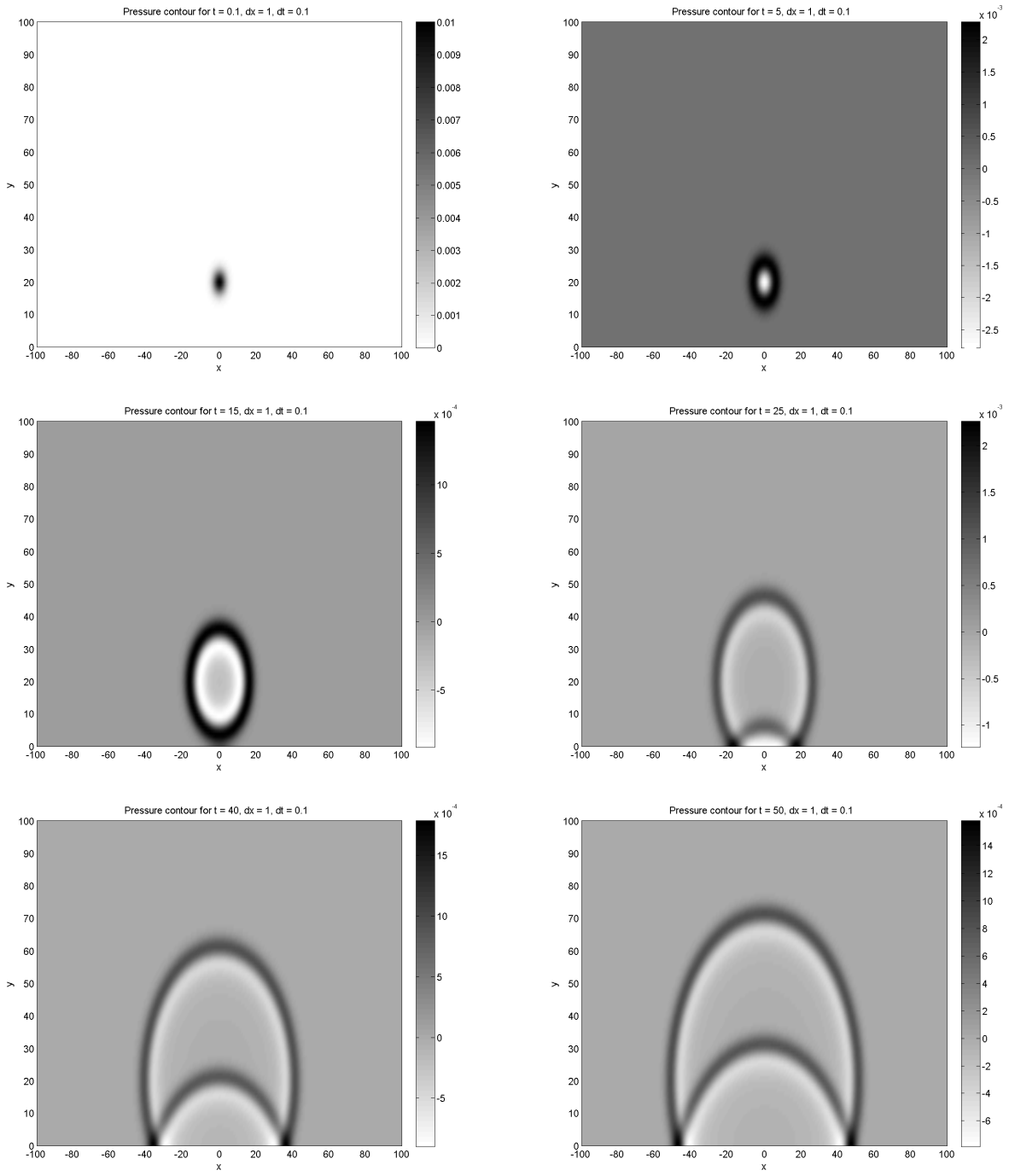

Figure 3: Pressure contour at 0.1, 5, 15, 25, 40 and 50 seconds.

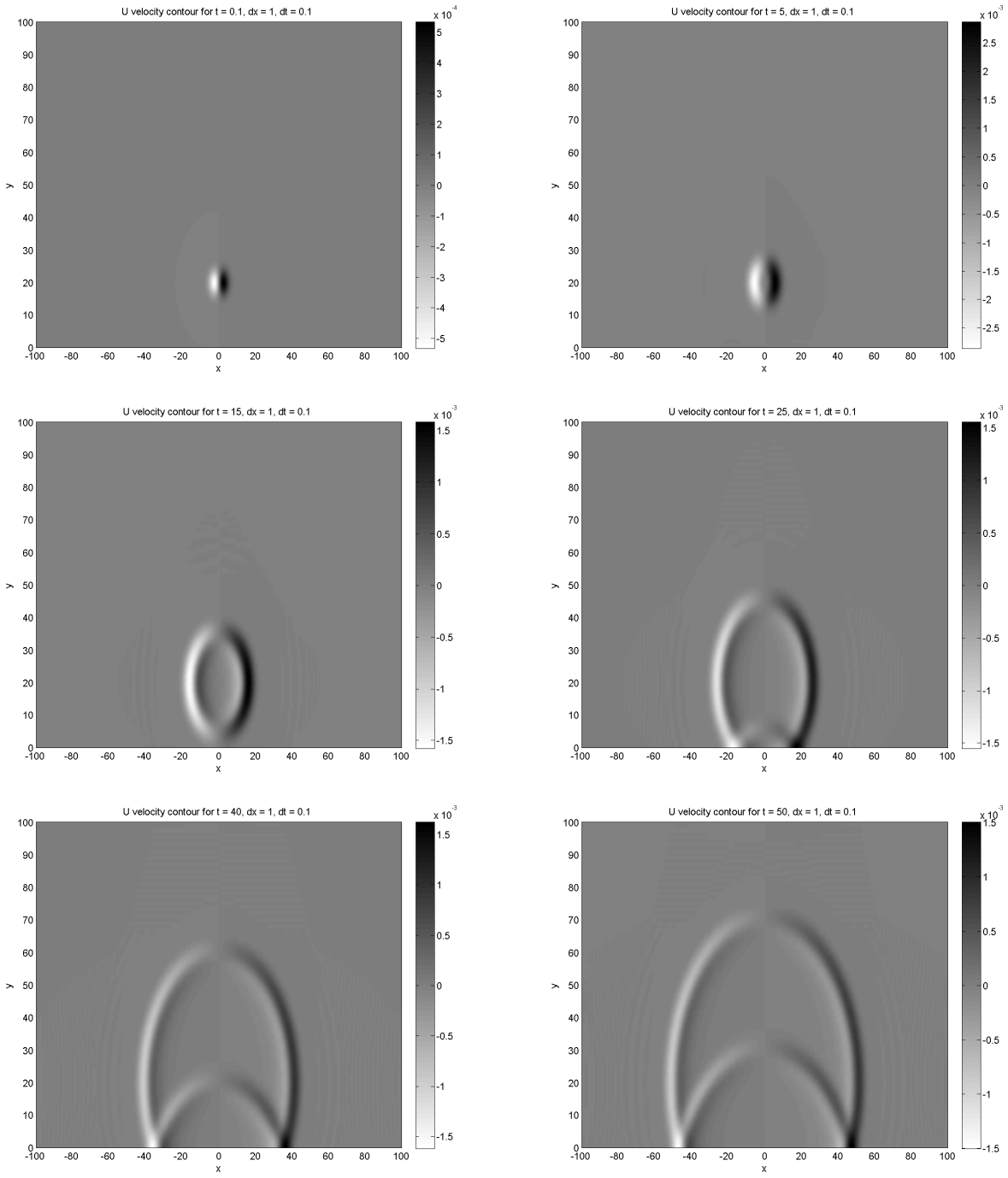

Figure 4: u velocity contour at 0.1, 5, 15, 25, 40 and 50 seconds.

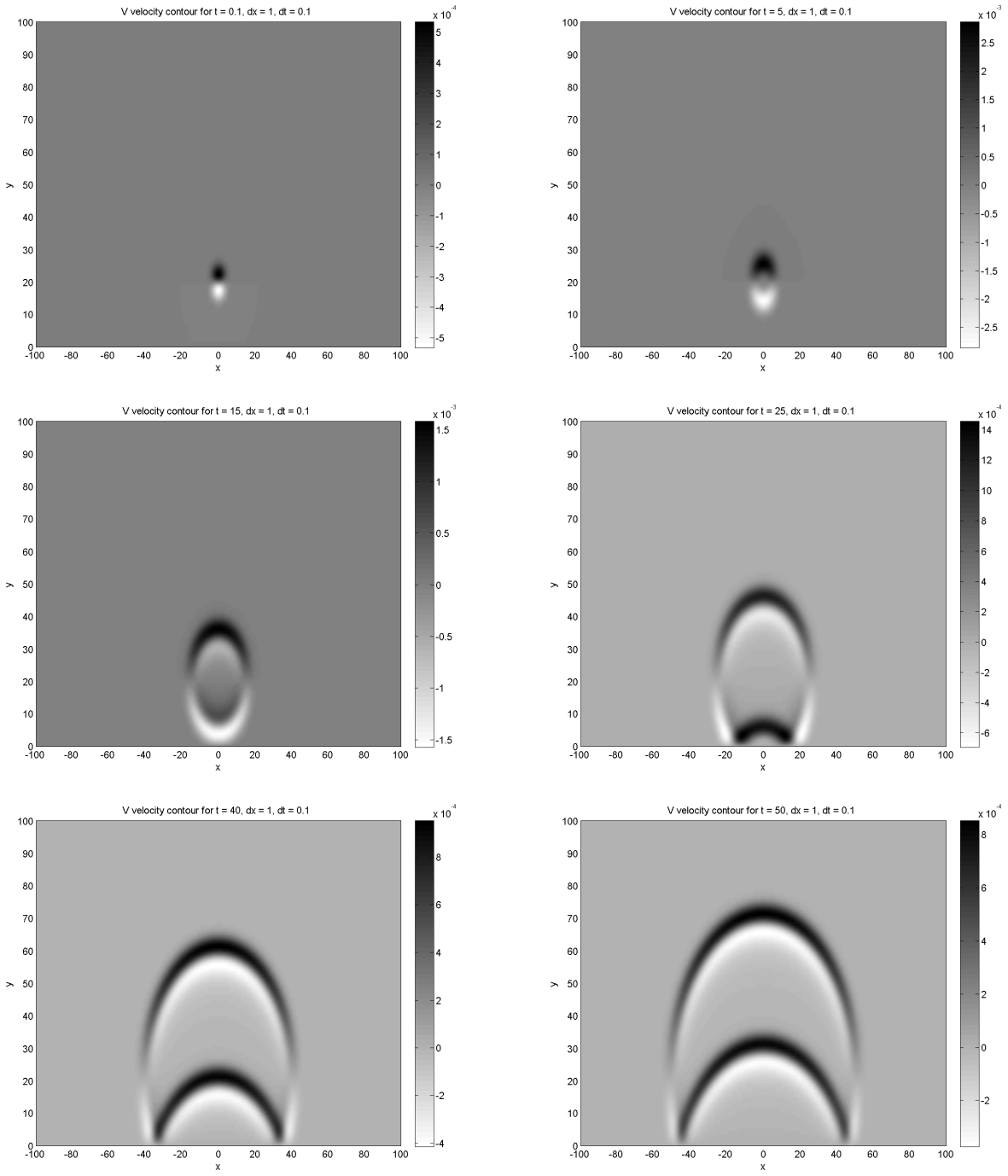

Figure 5: v velocity contour at 0.1, 5, 15, 25, 40 and 50 seconds.

Case 2

For the second case, the mean flow Mach number is set to 0.3, and is directed along the solid wall boundary. The density, pressure, x-direction velocity and y-direction velocity contours for selected time periods for the second case are shown in Figure 6, Figure 7, Figure 8 and Figure 9, respectively. These results are nearly identical to those of the first case, with the exception being that the center of the wave front is no longer stationary and moves with the flow in the positive x-direction.

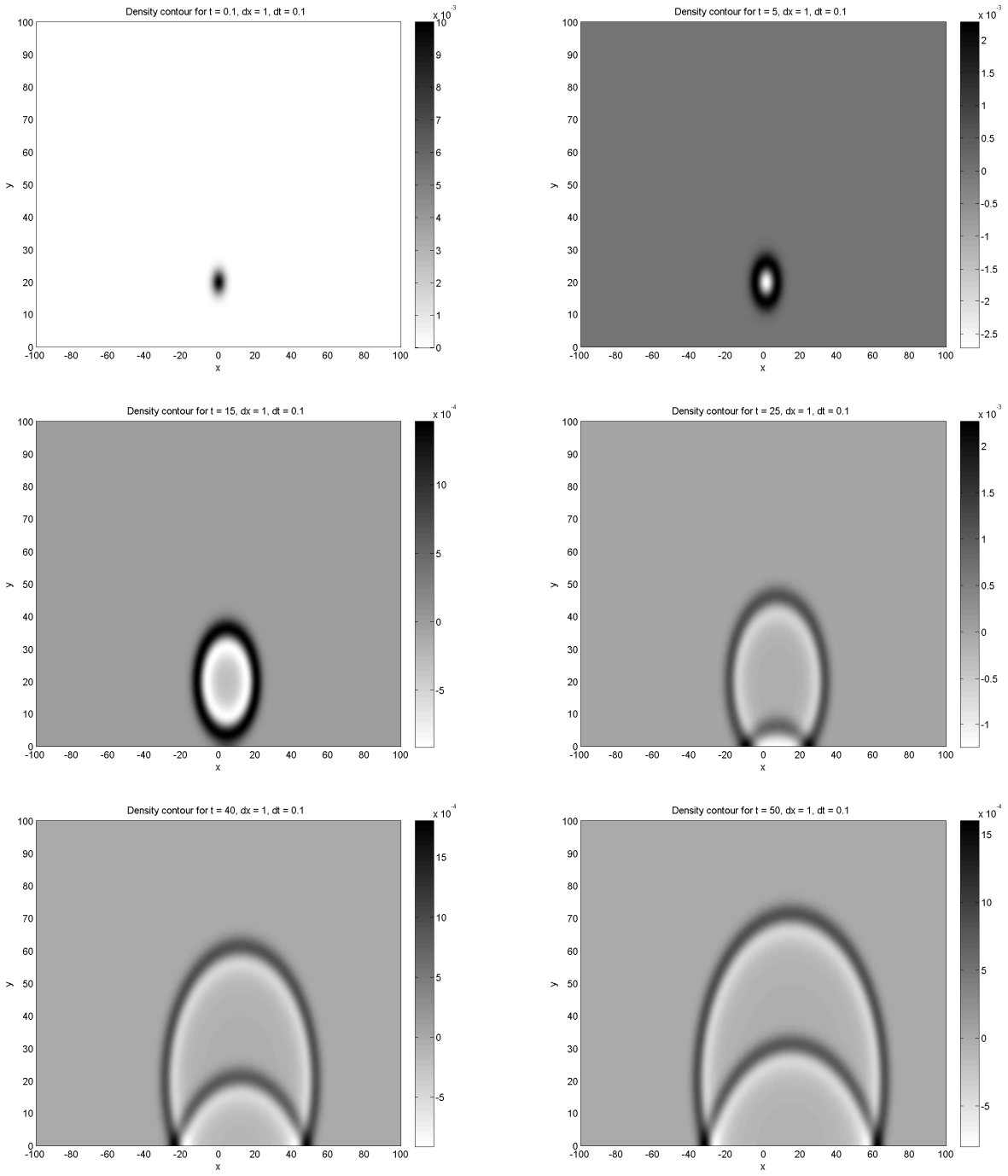

Figure 6: Density contour at 0.1, 5, 15, 25, 40 and 50 seconds.

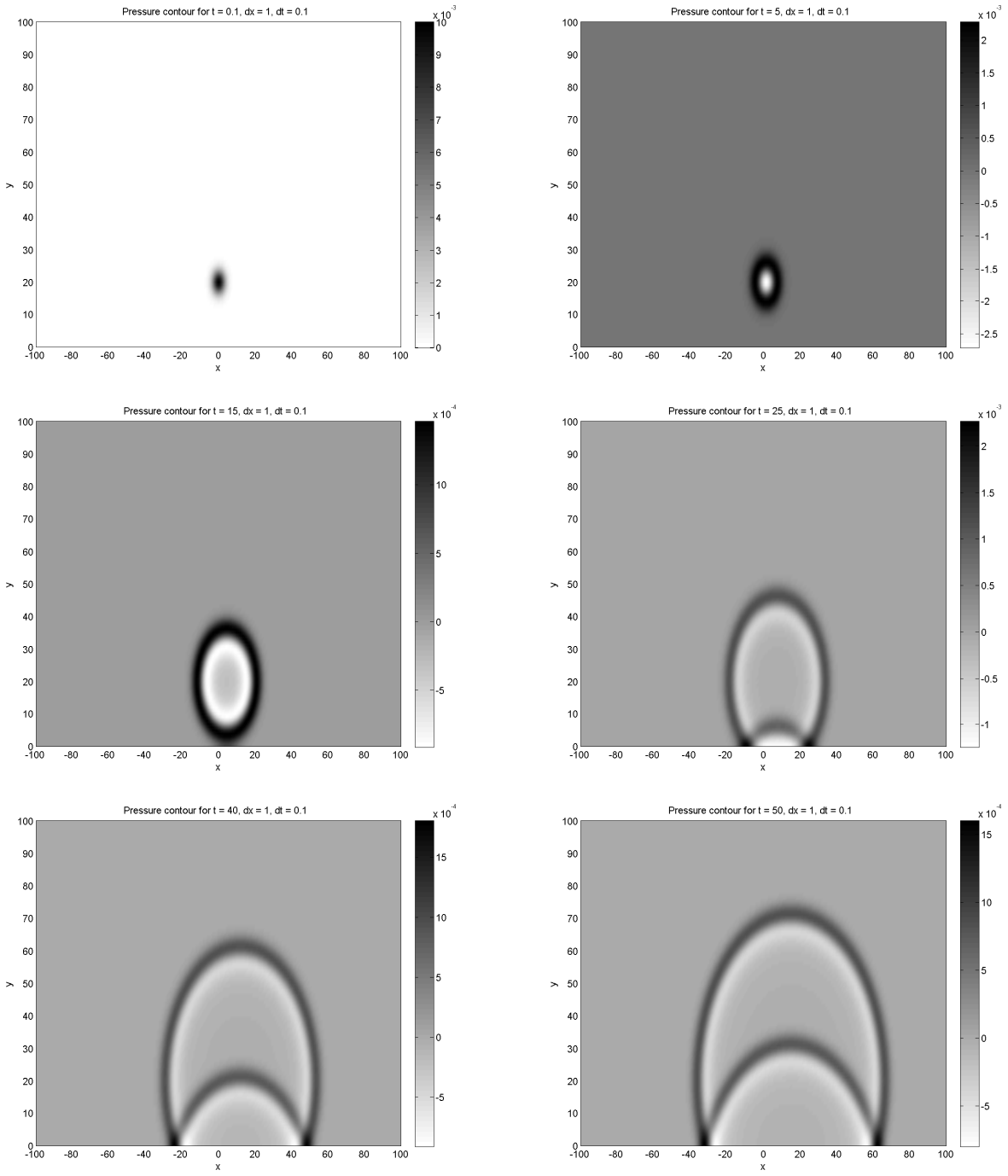

Figure 7: Pressure contour at 0.1, 5, 15, 25, 40 and 50 seconds.

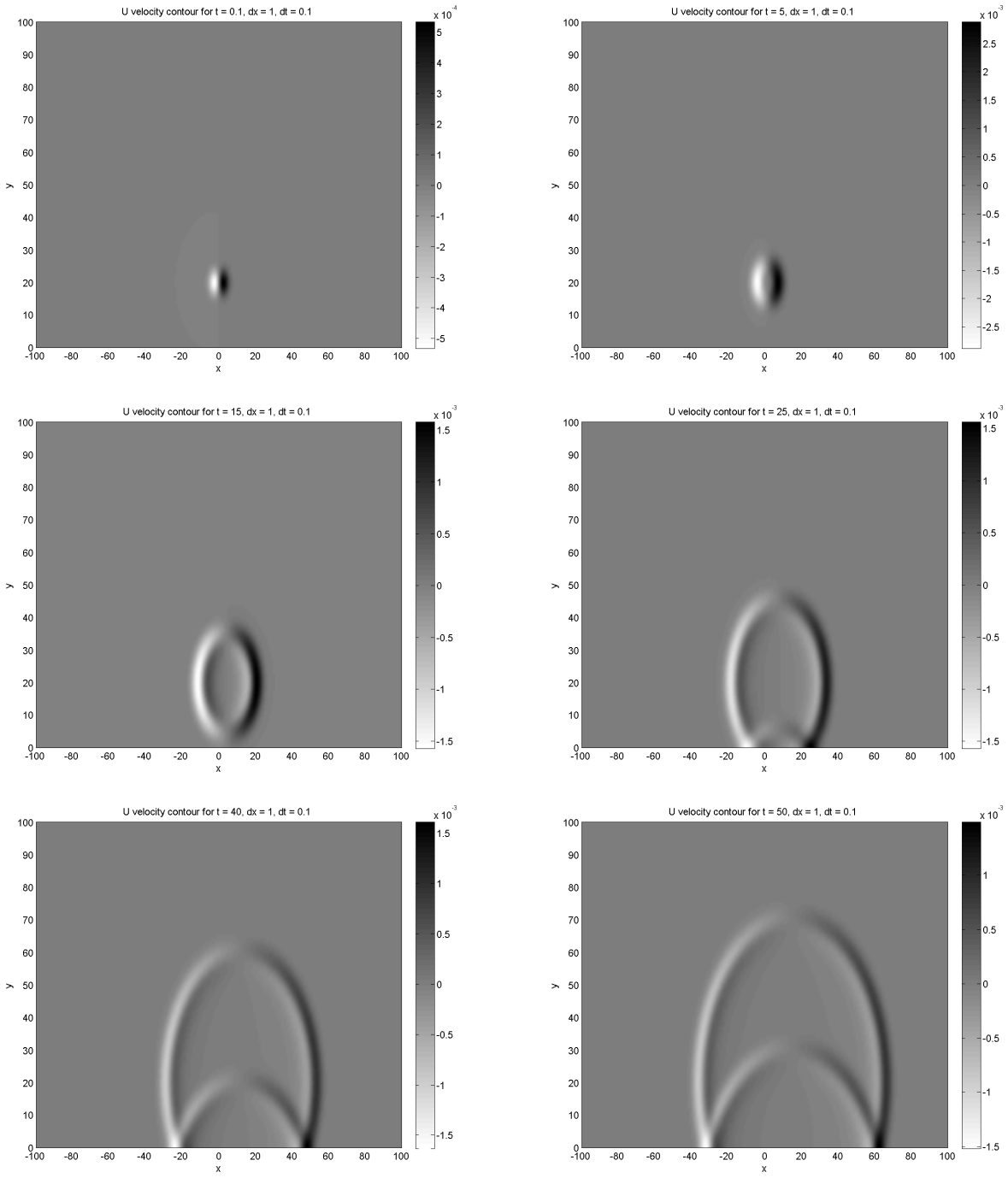

Figure 8: u velocity contour at 0.1, 5, 15, 25, 40 and 50 seconds.

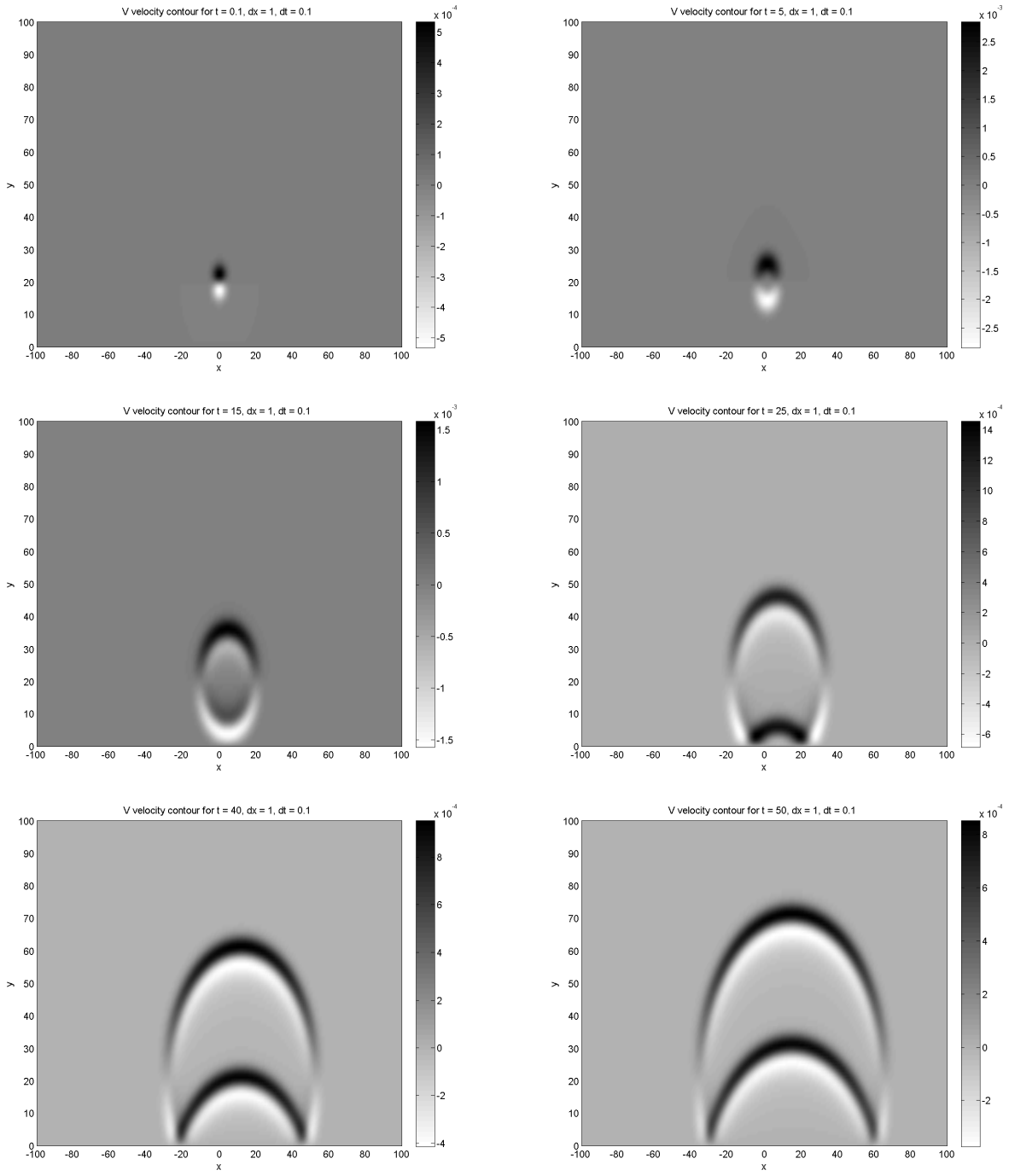

Figure 9: v velocity contour at 0.1, 5, 15, 25, 40 and 50 seconds.

Appendix

Appendix A: Main Program

```
function [] = Project3(dx,dt,Mx)
    %Euler equation solver, completed for AAE 694 Project 3 by
    %Michael Crawley
    %Inputs:
    %       dx: spatial step size
    %       dt: temporal step size
    %       Mx: mean flow in x-Direction
    tic;
    dy = dx;
    Nghost = 1;

    %Set up grid
    X = -100:dx:100;
    Y = 0:dy:100; %add ghost point to domain
    [x, y] = meshgrid(X,Y);
    foldername = [num2str(length(X)),' x ',num2str(length(Y)), ' Mx',
strrep(num2str(Mx),'.','_')];
    [Ny, Nx] = size(x);

    %Set maximum time for simulation to run
    plot_times = [5 15 25 40 50];
    time_itr_max = max(plot_times);
    imax = length(0:dt:time_itr_max)-1;

    %Boundary conDftion coefficients
    r = sqrt(x.^2+y.^2);
    V = Mx*(x./r)+sqrt(1-(Mx*(y./r)).^2);
    A = V.*(x./r);
    B = V.*(y./r);
    C = V./(2*r);

    %3rd order optimized time Dfscretization coefficients (b0, b1, b2, b3)
    b = [2.3025580883830 -2.4910075998482 1.5743409331815 -0.38589142217162];

    %initialize variables
    rho = zeros(length(Y),length(X),4);
    u = rho; v = rho; p = zeros(length(Y)+Nghost,length(X),4); %add ghost
point to pressure domain
    Drho = rho;
    Du = u;
    Dv = v;
    Dp = p;
    pswitch = p(:,:,1);

    %initial conditions
    rho(:,:,4) = 0.01*exp(-log(2)*(x.^2+(y-20).^2)/9);
    p(2:end,:,4) = 0.01*exp(-log(2)*(x.^2+(y-20).^2)/9);
    coefs = DRPlookup(1,5,dy); %create DRP coefficients to compute ghost
value
    p(1,:,4) = (-(1/coefs(1))*p(2:7,:,4)'*coefs(2:end))';
```

```
    %create numerical differentiation matrices and cells
    mDx = mNumericalDerivative(1,6,dx,Nx,'-DRP');
    mDy = mNumericalDerivative(1,6,dy,Ny,'-DRP')';
    mDyp = mNumericalDerivative(1,6,dx,Ny+Nghost,'-DRP')';
    dqdx = cell(4,1); dqdy = dqdx;

    if exist(foldername,'file') ~= 7
        mkdir(pwd,foldername);
    end
    cd(foldername);
    for i = 1:imax %time step loop
        %time shift spatial derivative matrices
        Drho(:,:,1:3) = Drho(:,:,2:4);
        Du(:,:,1:3) = Du(:,:,2:4);
        Dv(:,:,1:3) = Dv(:,:,2:4);
        Dp(:,:,1:3) = Dp(:,:,2:4);

        dqdx{1} = rho(:,:,4)*mDx;
        dqdy{1} = mDy*rho(:,:,4);
        dqdx{2} = u(:,:,4)*mDx;
        dqdy{2} = mDy*u(:,:,4);
        dqdx{3} = v(:,:,4)*mDx;
        dqdy{3} = mDy*v(:,:,4);
        dqdx{4} = p(:,:,4)*mDx;
        dqdy{4} = mDyp*p(:,:,4);

        %interior nodes
        Drho(:,:,4) = -(Mx*dqdx{1}+dqdx{2})-dqdy{3};
        Du(:,:,4) = -(Mx*dqdx{2}+dqdx{4}(2:end,:));
        Dv(:,:,4) = -(Mx*dqdx{3})-dqdy{4}(2:end,:);
        Dp(2:end,:,4) = -(Mx*dqdx{4}(2:end,:)+dqdx{2})-dqdy{3};

        %left boundary (radiation)
        Drho(:,1:3,4) = A(:,1:3).*(-dqdx{1}(:,1:3))+B(:,1:3).*(-
dqdy{1}(:,1:3))-C(:,1:3).*rho(:,1:3,4);
        Du(:,1:3,4) = A(:,1:3).*(-dqdx{2}(:,1:3))+B(:,1:3).*(-
dqdy{2}(:,1:3))-C(:,1:3).*u(:,1:3,4);
        Dv(:,1:3,4) = A(:,1:3).*(-dqdx{3}(:,1:3))+B(:,1:3).*(-
dqdy{3}(:,1:3))-C(:,1:3).*v(:,1:3,4);
        Dp(2:end,1:3,4) = A(:,1:3).*(-dqdx{4}(2:end,1:3))+B(:,1:3).*(-
dqdy{4}(2:end,1:3))-C(:,1:3).*p(2:end,1:3,4);

        %right boundary (outflow)
        Dp(2:end,end-2:end,4) = A(:,end-2:end).*(-dqdx{4}(2:end,end-
2:end))+B(:,end-2:end).*(-dqdy{4}(2:end,end-2:end))-C(:,end-
2:end).*p(2:end,end-2:end,4);
        Drho(:,end-2:end,4) = Mx*(-dqdx{1}(:,end-2:end)+dqdx{4}(2:end,end-
2:end))+A(:,end-2:end).*(-dqdx{4}(2:end,end-2:end))+B(:,end-2:end).*(-
dqdy{4}(2:end,end-2:end))-C(:,end-2:end).*p(2:end,end-2:end,4);

        %top boundary (radiation)
        Drho(end-2:end,4:end-3,4) = A(end-2:end,4:end-3).*(-dqdx{1}(end-
2:end,4:end-3))+B(end-2:end,4:end-3).*(-dqdy{1}(end-2:end,4:end-3))-C(end-
2:end,4:end-3).*rho(end-2:end,4:end-3,4);
```

```
        Du(end-2:end,4:end-3,4) = A(end-2:end,4:end-3).*(-dqdx{2}(end-
2:end,4:end-3))+B(end-2:end,4:end-3).*(-dqdy{2}(end-2:end,4:end-3))-C(end-
2:end,4:end-3).*u(end-2:end,4:end-3,4);
        Dv(end-2:end,4:end-3,4) = A(end-2:end,4:end-3).*(-dqdx{3}(end-
2:end,4:end-3))+B(end-2:end,4:end-3).*(-dqdy{3}(end-2:end,4:end-3))-C(end-
2:end,4:end-3).*v(end-2:end,4:end-3,4);
        Dp(end-2:end,4:end-3,4) = A(end-2:end,4:end-3).*(-dqdx{4}(end-
2:end,4:end-3))+B(end-2:end,4:end-3).*(-dqdy{4}(end-2:end,4:end-3))-C(end-
2:end,4:end-3).*p(end-2:end,4:end-3,4);

        %calculate physical variables at new time period
        rhoswitch =
rho(:,:,4)+dt*(b(1)*Drho(:,:,4)+b(2)*Drho(:,:,3)+b(3)*Drho(:,:,2)+b(4)*Drho(:
,:,1));
        uswitch =
u(:,:,4)+dt*(b(1)*Du(:,:,4)+b(2)*Du(:,:,3)+b(3)*Du(:,:,2)+b(4)*Du(:,:,1));
        vswitch =
v(:,:,4)+dt*(b(1)*Dv(:,:,4)+b(2)*Dv(:,:,3)+b(3)*Dv(:,:,2)+b(4)*Dv(:,:,1));
        pswitch(2:end,:) =
p(2:end,:,4)+dt*(b(1)*Dp(2:end,:,4)+b(2)*Dp(2:end,:,3)+b(3)*Dp(2:end,:,2)+b(4
)*Dp(2:end,:,1));
        pswitch(1,:) = (-(1/coefs(1))*p(2:7,:,4)'*coefs(2:end))'; %set ghost
value given solid wall requirements

        %%shift physical variable matrices
        rho(:,:,1:3) = rho(:,:,2:4);
        u(:,:,1:3) = u(:,:,2:4);
        v(:,:,1:3) = v(:,:,2:4);
        p(:,:,1:3) = p(:,:,2:4);

        %update physical variables
        rho(:,:,4) = rhoswitch;
        u(:,:,4) = uswitch;
        v(:,:,4) = vswitch;
        p(:,:,4) = pswitch;

        %plot variables at specified times
        if any(abs(dt*i - plot_times) <= eps) || i ==1
            pcolor(X,Y,rho(:,:,end)); colorbar; shading interp;
colormap(flipud(gray)); xlabel('x'); ylabel('y'); title(['Density contour for
t = ',num2str(dt*i),', dx = ', num2str(dx),', dt = ', num2str(dt)]);

saveas(gcf,['density_i',num2str(i)],'fig');saveas(gcf,['density_i',num2str(i)
],'png');

            pcolor(X,Y,u(:,:,end)); colorbar; shading interp;
colormap(flipud(gray)); xlabel('x'); ylabel('y'); title(['U velocity contour
for t = ',num2str(dt*i),', dx = ', num2str(dx),', dt = ', num2str(dt)]);

saveas(gcf,['u_i',num2str(i)],'fig');saveas(gcf,['u_i',num2str(i)],'png');

            pcolor(X,Y,v(:,:,end)); colorbar; shading interp;
colormap(flipud(gray)); xlabel('x'); ylabel('y'); title(['V velocity contour
for t = ',num2str(dt*i),', dx = ', num2str(dx),', dt = ', num2str(dt)]);

saveas(gcf,['v_i',num2str(i)],'fig');saveas(gcf,['v_i',num2str(i)],'png');
```

```
            pcolor(X,Y,p(2:end,:,end)); colorbar; shading interp;
colormap(flipud(gray)); xlabel('x'); ylabel('y'); title(['Pressure contour
for t = ',num2str(dt*i),', dx = ', num2str(dx),', dt = ', num2str(dt)]);

saveas(gcf,['p_i',num2str(i)],'fig');saveas(gcf,['p_i',num2str(i)],'png');
            close all;
        end
    end
    cd ..
    compute_time = toc
end
```

Appendix B: Numerical Derivative Matrix Generator

```
function [coefm] = mNumericalDerivative(Dorder, Horder, h, N, varargin)
    %Generate matrix to numerically derivate matrix given constant step size
    %Code Version: 1.0 @ 2011-03-04
    %This is not complete; DRP scheme is only available for central
    %differencing
    %Inputs:
    %   Dorder: Derivative Order
    %   Horder: Number of terms to use
    %   h: step size
    %   N: size of required matrix
    %   options:    '-center' for central difference only
    %               '-upstream' for backward difference only
    %               '-downstream' for forward difference only
    %               '-DRP' for 4th order DRP scheme
    %Outputs:
    %   coefm: matrix for Dorder derivative of phi with Horder accuracy,
    %   Dphi = phi*coefm

    %set default scheme (if not given by user)
    if any(strcmp(varargin,'-DRP'))
        if Dorder ~= 1 || Horder ~= 6
            warning('DRP scheme only available for 1st derivative, 4th order.
Switching to standard Taylor series expansion mode.'); %#ok<WNTAG>
            varargin(strcmp(varargin,'-DRP')) = '';
        end
        if length(varargin) ==1
            varargin{2} = '-center';
        end
    end

    if isempty(varargin)
       varargin{1} = '-center';
    end

    %calculate coefficients for differencing operations
    if any(strcmp(varargin,'-center'))
        if mod(Horder,2)
            error('Given accuracy order cannot be accomplished with central
finite difference method; please provide even accuracy order');
        end
        %central differencing coefficients
        n = Horder/2+floor((Dorder-1)/2);
        if any(strcmp(varargin,'-DRP'))
            coefs = DRPlookup(3,3,h);
            coefm = spdiags(repmat(coefs',N,1),-3:3,N,N);
        else
            coefs = TSE(n,n,h,Dorder);
            coefm = spdiags(repmat(coefs',N,1),-n:n,N,N);
        end

        %left bound coefficients
        for i = 1:Horder/2+floor((Dorder-1)/2)
            if any(strcmp(varargin,'-DRP'))
                coefm(i,1:7) = DRPlookup(i-1,7-i,h);
```

```
            else
                coefm(i,1:2*n+Dorder) = TSE(i-1,Horder+Dorder-i,h,Dorder)';
            end
        end

        %right bound coefficients
        for i = N-Horder/2+floor((Dorder-1)/2)+1:N
            if any(strcmp(varargin,'-DRP'))
                coefm(i,end-6:end) = DRPlookup(-N+6+i,N-i,h);
            else
                coefm(i,end+1-(2*n+Dorder):end) = TSE(-N+Horder+Dorder-1+i,N-
i,h,Dorder)';
            end
        end

    elseif any(strcmp(varargin,'-upstream'));
        %Upstream differencing coefficients
        coefs = TSE(Horder+Dorder-1,0,h,Dorder);
        coefm = spdiags(repmat(coefs',N,1),-(Horder+Dorder-1):0,N,N);

        %Left bound coefficients
        for i = 1:Horder+Dorder-1
           coefm(i,1:Horder+Dorder) = TSE(i-1,Horder+Dorder-i,h,Dorder)';
        end

    elseif any(strcmp(varargin,'-downstream'));
        %Downstream differencing coefficients
        coefs = TSE(0,Horder+Dorder-1,h,Dorder);
        coefm = spdiags(repmat(coefs',N,1),0:Horder+Dorder-1,N,N);

        %Right bound coefficients
        for i = N-Horder-Dorder+1:N
           coefm(i,end-(Horder+Dorder-1):end) = TSE(-N+Horder+Dorder-1+i,N-
i,h,Dorder)';
        end
    end
    coefm = coefm';
end
```

Appendix C: DRP Lookup

```
function [coefs] = DRPlookup(n,m,h)
%Looks up coefficients for 4th Order, 1st Derivative DRP scheme
    a{33} = [-0.02084314277031176 ...
                0.166705904414580469 ...
                -0.77088238051822552 ...
                0 ...
                0.77088238051822552 ...
                -0.166705904414580469 ...
                0.02084314277031176]';

    a{42} = [0.02636943100 ...
                -0.16613853300 ...
                0.518484526d0 ...
                -1.27327473700 ...
                0.47476091400 ...
                0.46884035700 ...
                -0.049041958d0]';

    a{51} = [-0.048230454 ...
                0.281814650 ...
                -0.768949766 ...
                1.388928322 ...
                -2.147776050 ...
                1.084875676 ...
                0.209337622]';

    a{60} = [0.203876371 ...
                -1.128328861 ...
                2.833498741 ...
                -4.461567104 ...
                5.108851915 ...
                -4.748611401 ...
                2.192280339]';

    a{24} = -flipud(a{42});
    a{15} = -flipud(a{51});
    a{06} = -flipud(a{60});

    coefs = a{str2double(strcat([num2str(n),num2str(m)]))}/h;
end
```