Development of Numerical Analysis for Two-Dimensional Inviscid Burgers Equation using Finite Difference-based MacCormack Method

B.B, Terang, A.G, Syahrahman

Department of Aerospace Engineering, Bandung Institute of Technology
St. Ganesha No.10 Bandung 40132
terangbrantas@gmail.com

Abstract— **Nonlinear partial differential equations hold a significant role in applied numerical engineering, serving as crucial tools to depict and explore real-world phenomena. Among these equations, the classical Burgers equation, belonging to the nonlinear partial differential category, has captivated researchers. It serves as a focal point in the study of diverse physical concepts including shock wave theory, fluid dynamics, turbulent flow, and gas dynamics. Focused on harnessing the Finite Difference-based MacCormack Method, this study aims to offer a comprehensive understanding of its application in simulating and analyzing the behaviour of the Two-Dimensional Inviscid Burgers Equation. The investigation further examines the computational efficiency and accuracy of the method in handling discontinuities within the equation framework.**

Keywords— **Numerical Simulation, Inviscid Burger Equation, MacCormack Method**

I. INTRODUCTION

The 2D Burgers' equation is one of the fundamental mathematical constructions in fluid mechanics. This equation shares same convective and diffusion traits akin to the famous Navier-Stokes equation. It serves as a prevalent model across multiple domains, offering a simplified perspective on diverse physical flows and complexities. Its applications span various arenas, encompassing dynamic modelling, turbulence phenomena, the depiction of shock wave movement within viscous fluids, and the analysis of traffic flow dynamics.

However, within the traditional numerical method, the computational expenses elevate notably with the escalation of the Reynolds number (Re), especially noticeable when a smooth wave transforms into a single-shock wave. Furthermore, the costs rise as the spatial discretization mesh undergoes refinement. This challenge intensifies when tackling control or PDE optimization problems across a broad spectrum of engineering applications through Direct Numerical Simulation (DNS) of the complete 2D Burgers' equations model. This arises due to the necessity for rapid and recurrent numerical simulations, presenting significant mathematical and computational hurdles, impacting both CPU and memory requisites.

To address the challenges in simulating, controlling, and optimizing complex systems, reduced-order modeling emerges as a promising solution. This approach enables the representation of high-dimensional system dynamics through a reduced number of degrees of freedom, offering a powerful and practical means to tackle complexity. A current area of active research involves the development of low-dimensional models for partial differential equations (PDEs). One method employed in simulating these reduced-dimensional PDE models is the finite difference method, demonstrating its effectiveness in capturing the dynamics of these systems while operating on a reduced scale.

II. LITTERATURE REVIEW

A. Grid Generation and Transfinite Interpolation

Grid generation based on interpolation has two basic advantages:

1. Rapid computation of grids
2. Direct control over grid point locations

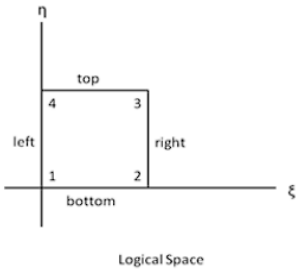
Despite their benefits, interpolation method may encounter a limitation where smooth grids might not be produced, particularly in problem domains featuring steep curves or bends. In such instances the grid tends to fold along the bends within the domain. These grid generation techniques, also referred to as algebraic method, primarily involve a standard approach known as transfinite interpolation (TFI) for creating grids algebraically.

Any 2D grid generation problem requires the description of four boundaries of the region, i.e. four parametric equations for the boundaries.

$$X_b(\xi), X_t(\xi), \quad 0 \le \xi \le 1 \tag{1}$$
$$X_b(\eta), X_t(\eta), \quad 0 \le \eta \le 1 \tag{2}$$

The vector notation shown in Eq. (1) should be converted to component form to be implemented in a computer program. There should be four important consistency check for the boundary formulas at four corners of the region, as show below

$$\begin{aligned} X_b(0) &= X_l(0) \\ X_b(1) &= X_r(0) \\ X_t(1) &= X_r(1) \\ X_t(0) &= X_l(1) \end{aligned} \tag{3}$$



Logical Space

Figure 1 Computational **Domain Boundary**

The generalized coordinates calculated based on following equations

$$\xi_{i,j} = \frac{1}{P_{i,j}}\left(\xi_{i,0} + \eta_{0,i}\left(\xi_{i,jmax} - \xi_{i,0}\right)\right) \tag{4}$$

$$\eta_{i,j} = \frac{1}{P_{i,j}}\left(\eta_{0,j} + \xi_{i,0}\left(\eta_{imax,j} - \eta_{0,j}\right)\right) \tag{5}$$

$$P_{i,j} = 1 - \left(\xi_{i,jmax} - \xi_{i,0}\right) * \left(\eta_{imax,j} - \eta_{0,j}\right) \tag{6}$$

Therefore, using the obtained generalized coordinates, the transfinite interpolation result for cartesian coordinates can be calculated using

$$\begin{aligned} R_{i,j}(\xi,\eta) = {} & (1-\xi)R_{0,j} + \xi R_{imax,j} \\ & + (1-\eta)R_{i,0} + \eta R_{i,jmax} \\ & - [(1-\xi)(1-\eta)R_{0,0} \\ & + \xi(1-\eta)R_{imax,0} \\ & + (1-\xi)R_{0.jmax} \\ & + \xi\eta R_{imax,jmax}\ ] \end{aligned} \tag{7}$$

Where the $imax$ and $jmax$ representing the maximum number of grid at i and j, respectively.

B. Inviscid Burgers Equation

The two-dimensional Burgers' equation, a foundational mathematical framework in fluid mechanics, shares similar convective and diffusion characteristics with the Navies-Stokes equation. It serves as a widely adopted, simplistic model across multiple domains, aiding in comprehending various physical flows and complexities. Its applications span from dynamic modelling, turbulence phenomena, analysis of shockwave movement within viscous fluids, and the study of traffic flow dynamics.

The used two-dimensional Burgers' Equation in the developed numerical model is

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y} = 0 \tag{8}$$

For domain : $0 \le x \le 1$ and $0 \le y \le 1$
With boundary conditions

$$\begin{aligned} u(0,y) &= 1.5 \\ u(1.y) &= -0.5 \\ u(x,0) &= 1.5 - 2x \end{aligned} \tag{9}$$

C. Space and Time Discretization

Space and Time discretization are pivotal aspect in numerical method, serving as the basis for solving complex mathematical equations representing real-world phenomena. Space discretization involves dividing the spatial domain into smaller elements, enabling the representation of continuous space in a discrete form. Time discretization, on the other hand, breaks down the temporal evolution of a system into discrete time steps. Together, these discretization techniques allow mathematical model to be approximated and solved computationally. Proper selection and refinement of these discretization methods significantly impact the accuracy, stability, and efficiency of numerical simulation across various scientific and engineering disciplines.

In this numerical model for solving 2D Inviscid Burgers equation, the used discretization techniques are selected mainly based on the boundary conditions. The used discretization techniques are FTCS (Forward Time Centred Space) at x-axis variables and FTBS (Forward Time Backward Space) for y-axis variables. The general one-dimensional form of FTCS and FTBS discretization given by Eq. (5) and Eq. (6), respectively.

$$U_i^{n+1} = U_i^n + \alpha\,\Delta t\,\frac{U_{i+1}^n - U_{i-1}^n}{2\Delta x} \tag{10}$$

$$U_i^{n+1} = U_i^n + \alpha\,\Delta t\,\frac{U_i^n - U_{i-1}^n}{\Delta x} \tag{11}$$

The FTCS is selected for x-axis iteration since the boundary condition types for x-axis variables are both Dirichlet (given value) at both downstream ($x = 0$) and upstream ($x = 1$). Therefore, the numerical scheme needs to be the one that capable to consider downstream and upstream value simultaneously, which narrow down to the FTCS scheme considering its explicit in time scheme to reduce complexity during code development. The similar explicit in time scheme, FTBS scheme is selected for variables at y-axis due to the boundary

conditions. In y-axis, the available Dirichlet boundary conditions is only at the downstream ($y = 0$) meanwhile the upstream ($y = 1$) is a free value which means Neumann boundary conditions. This conditions, require a numerical scheme that only consider the downstream value since the upstream value itself is "unknown". Therefore, the options fall to FTBS scheme.

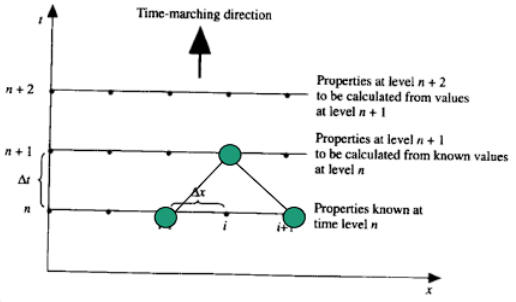


Figure 2 FTCS Scheme

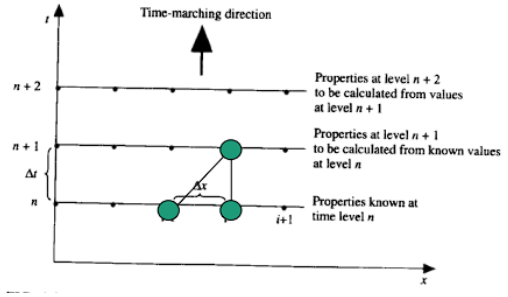


Figure 3 FTBS Scheme

D. McCormack Method

The MacCormack method stands as a significant numerical approach for solving partial differential equations, particularly those describing fluid dynamics and wave propagations. This method operates by describing both space and time, employing a two-step process involving predictor and corrector steps. The MacCormack method offers accuracy and stability in approximating solutions for fluid flow problems and has proven effective in simulating shock waves, turbulent flows, and other nonlinear physical phenomena. To utilize the MacCormack Method, the used Burgers's Equation in Eq. (4) needs to be modified as follow

$$\frac{\partial u}{\partial t} + \frac{\partial E}{\partial x} + \frac{\partial F}{\partial y} = 0$$

with (12)

$$E = f(u) = \frac{u^2}{2}, \quad F = g(u) = u$$

Applying the discretization technique, the solver scheme given as follows

Predictor Step

$$u^*_{i,j} = u^n_{i,j} - \frac{\Delta t}{2\Delta x}\left(E^n_{i+1,j} - E^n_{i-1,j}\right) - \frac{\Delta t}{\Delta y}\left(F^n_{i,j} - F^n_{i,j-1}\right) \quad (13)$$

Corrector Step

$$u^{**}_{i,j} = u^n_{i,j} - \frac{\Delta t}{2\Delta x}\left(E^*_{i+1,j} - E^*_{i-1,j}\right) - \frac{\Delta t}{\Delta y}\left(F^*_{i,j} - F^*_{i,j-1}\right) \quad (14)$$

Solution

$$u^{n+1}_{i,j} = \frac{1}{2}\left(u^*_{i,j} + u^{**}_{i,j}\right) \quad (15)$$

E. Artificial Viscosity

Artificial diffusivity serves as a crucial tool in numerical method for accurately simulating phenomena involving discontinuities. It is a technique that artificially introduces a controlled amount of viscosity into calculations to mitigate numerical instabilities that arise when handling sharp gradients or discontinuities in the solution. By mimicking the effects of physical viscosity, artificial viscosity helps smooth out abrupt changes in the solution, ensuring stability and preventing spurious oscillation. The general form of artificial viscosity in one-dimensional case given as follows

$$S^n_i = D_{i+\frac{1}{2}}(U_{i+1} - U_i) - D_{i-\frac{1}{2}}(U_i - U_{i-1})$$

$$D_{i+\frac{1}{2}} = \varepsilon(|u| + a)_{i+\frac{1}{2}} \frac{|p_{i+1} - 2p_i + p_{i-1}|}{p_{i+1} - 2p_i + p_{i-1}} \quad (16)$$

D = *artificial dissipation*,
p = Shock Sensor
e = *Switching*

In the used two-dimensional Inviscid Burgers' equation, there is only one primitive variable, therefore the artificial viscosity is only a function of u. Further in the developed code, the artificial viscosity amplified by a constant value to reduce the computational complexity, the adaptive amplification can be added later as an improvement of the code. Therefore the artificial viscosity can be represented as

$$S^n_{i,j} = S^n_i + S^n_j$$

$$S^n_i = \nu \frac{\left| u^n_{i+1,j} - 2u^n_{i,j} + u^n_{i-1,j} \right|}{u^n_{i+1,j} + 2u^n_{i,j} + u^n_{i-1,j}} \left(u^n_{i+1,j} - 2u^n_{i,j} + u^n_{i-1,j}\right) \quad (17)$$

$$S_j^n = v\frac{\left|u_{i,j+1}^n - 2u_{i,j}^n + u_{i,j-1}^n\right|}{u_{i,j+1}^n + 2u_{i,j}^n + u_{i,j-1}^n}\left(u_{i,j+1}^n - 2u_{i,j}^n + u_{i,j-1}^n\right)$$

With the v is a constant viscosity value.

III. METHODOLOGY

The general flowchart of developed numerical scheme given by Figure 4. After that, the computational domain created by utilizing Transfinite Interpolation. The detailed flowchart of transfinite interpolation given by Figure 5.

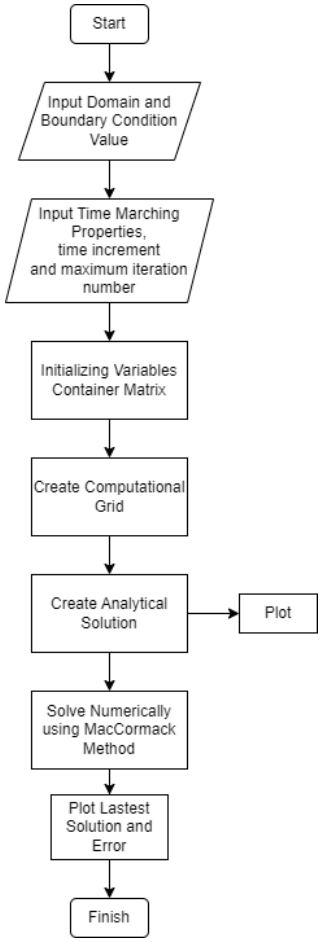


Figure 4 General Flowchart

A. Input Value

The program starts by inputting the domain and boundary conditions. Continued by inputting the time marching properties, which are time increment, maximum time step, convergent criterion, and artificial viscosity constant. After all required variables inputted, the programs start by initializing value/result container.

B. Grid Generation

The computational domain created by utilizing Transfinite Interpolation. The detailed flowchart of transfinite interpolation given by Figure 5.

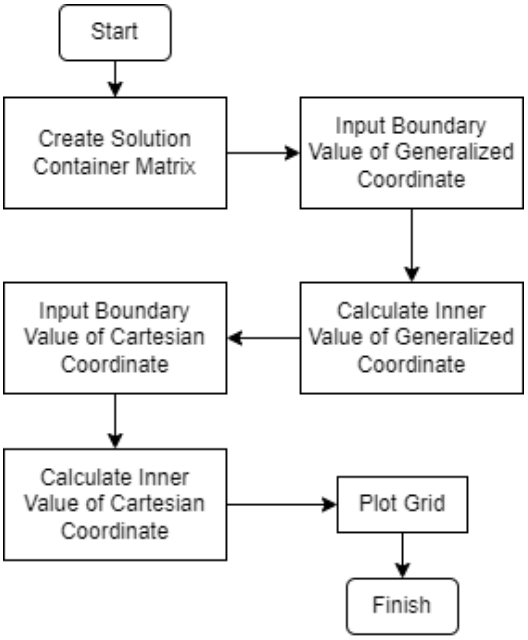


Figure 5 Grid Generation Flowchart

C. Analytical Solution

The Analytical Solution created based on following piecewise equation

for $y \leq 0.5$

$$u(x,y) = \begin{cases} 1.5 & if \quad x \leq 1.5y \\ \dfrac{1.5-2x}{1-2y} & if \quad 1.5y \leq x \leq (1-0.5y) \\ -0.5 & if \quad x \geq (1-0.5y) \end{cases} \tag{18}$$

for $y \geq 0.5$

$$u(x,y) = \begin{cases} 1.5 & if \quad x \leq (0.5+0.5y) \\ -0.5 & if \quad x \geq (0.5+0.5y) \end{cases}$$

D. MacCormac Method

The MacCormac method is calculated using predictor and corrector step with adding boundary condition evaluation at each step. The flowchart for MacCormac method given by Figure 6

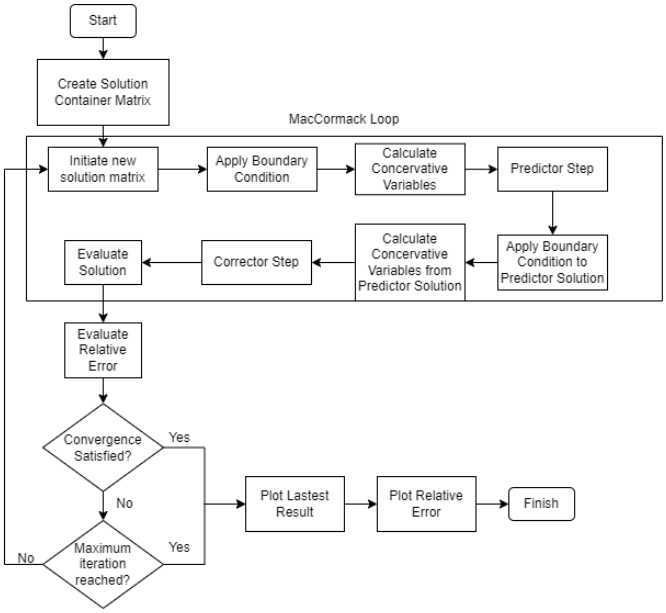


Figure 6 MacCormack Method

IV. RESULTS & ANALYSIS

Following is the selected time marching restrictions in the developed numerical scheme

Time Step	$10^{-3}\ s$
Maximum iteration	10000
Convergence criterion	10^{-4}

A. Grid Generation Result

The grid generation carried out to two different number of grid, which are using $N = 21$ and $N = 41$ number of grid at x and y axis. Following is the result for each grid generation process.

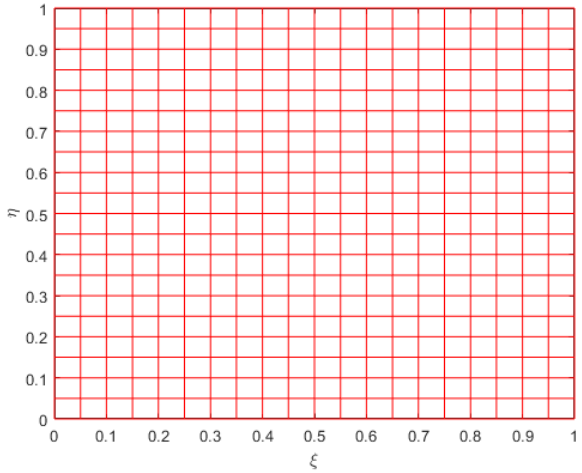


Figure 7 Generalized Coordinate for N=21

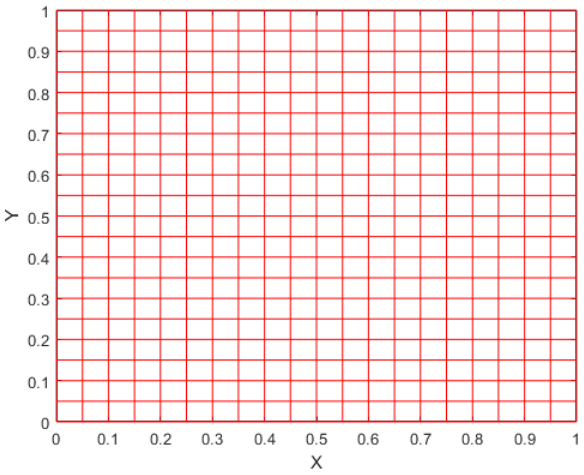


Figure 8 Cartesian Coordinate for N=21

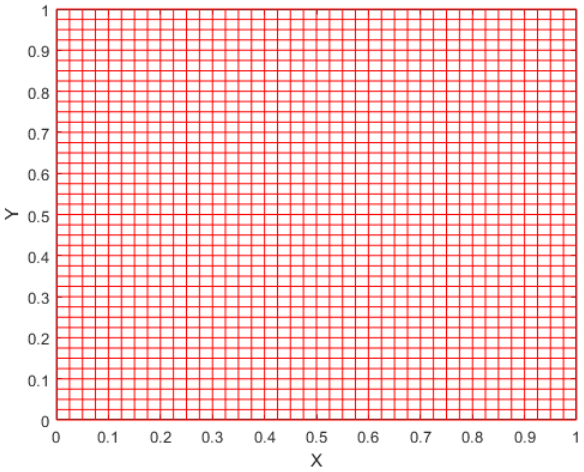


Figure 9 Generalized Coordinate for N=41

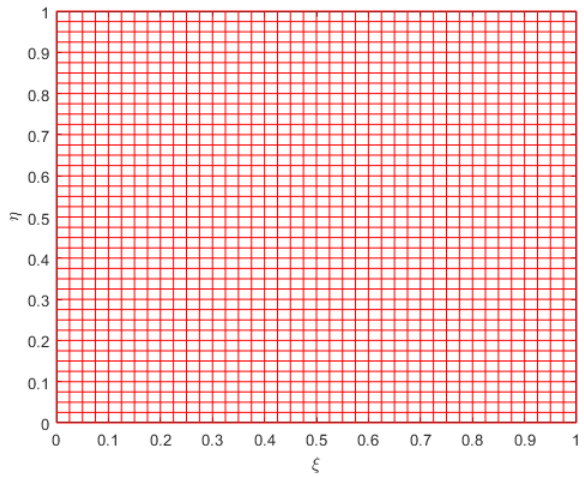


Figure 10 Cartesian Coordinate for N=41

As can be seen, the parallel grid has been created with different spatial increments. Since in this case the cartesian grid itself already parallel each other, there is no difference between the generalized coordinates and the cartesian coordinates grid.

B. Analytical Solution

The analytical solution created based on Eq. (17) . Following is the result for each grid size.

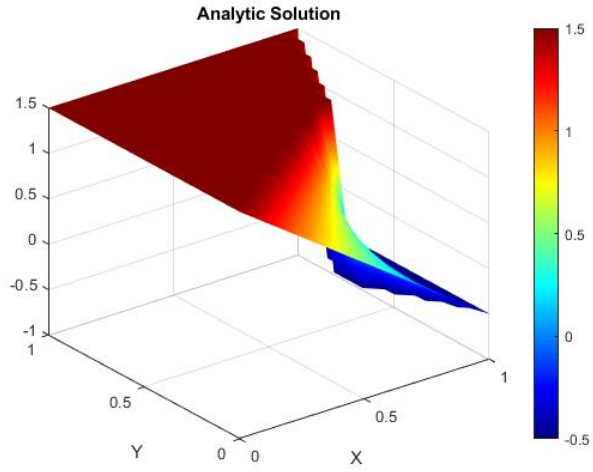


Figure 11 3D view of analytical solution evaluated at N=21

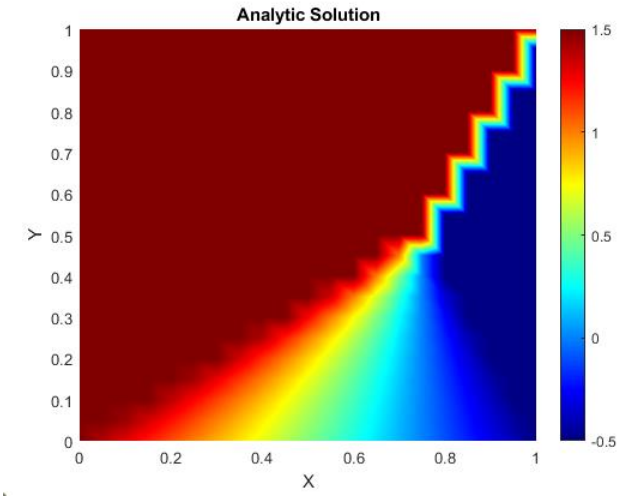


Figure 12 Top view of analytical solution evaluated at N=21

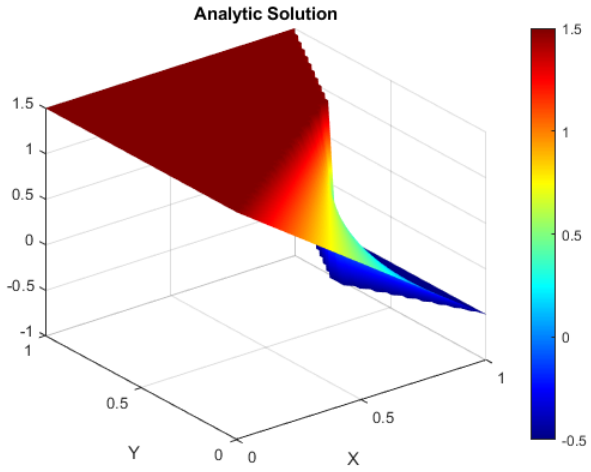


Figure 13 3D view of analytical solution evaluated at N=41

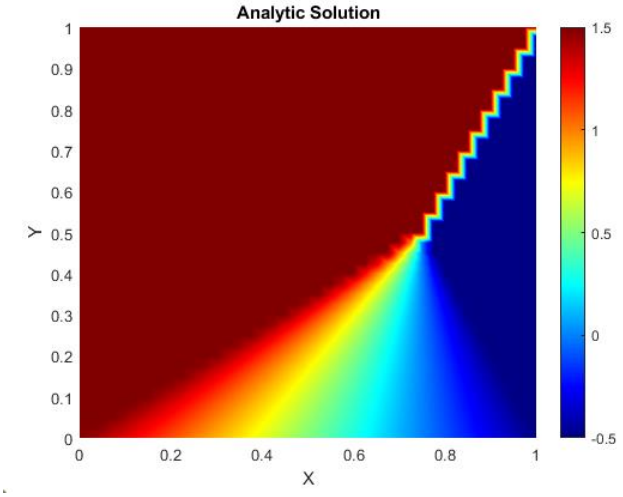


Figure 14 Top view of analytical solution evaluated at N=41

As the native of Burgers' Equation mentioned previously, there is a discontinuity at its solution. However, the increase of grid number shown to reduce the non-smooth area around discontinuity. It can be expected that the same results will appear in numerical simulations where increasing the number of grids will smooth out discontinuous areas.

C. Steady Solution for N=21

Following is the steady solution for number of grid $N = 21$

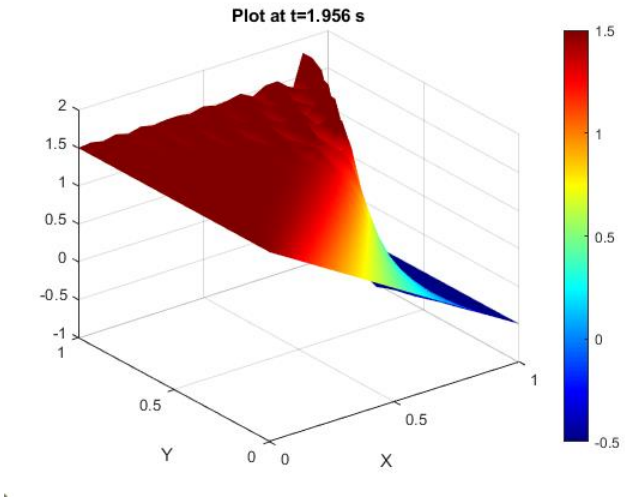


Figure 15 3D view of numerical solution evaluated at N=21

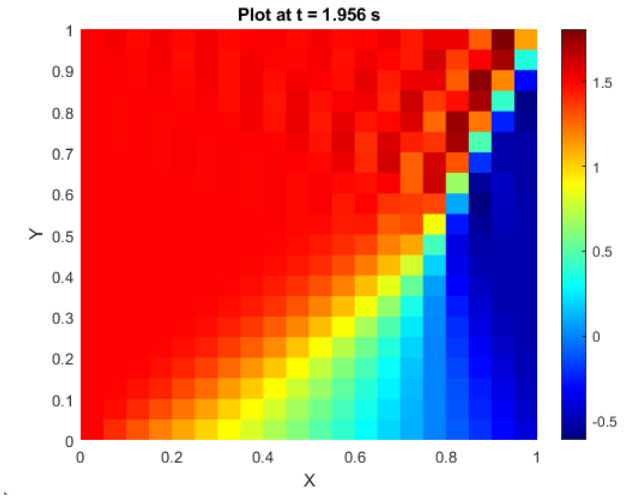


Figure 16 Top view of numerical solution evaluated at N=21

Meanwhile the relative error for each iteration is shown by following figure

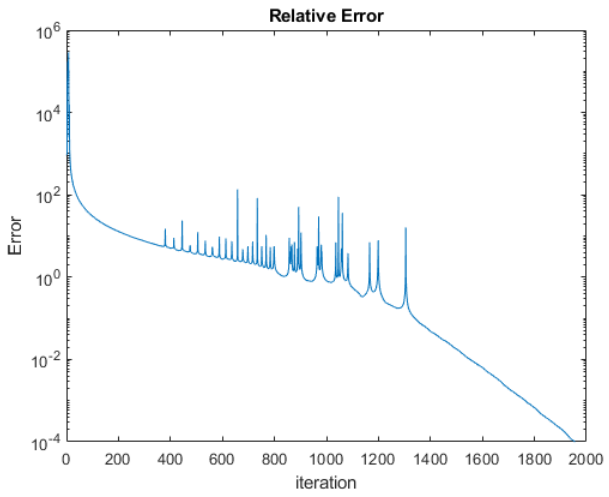


Figure 17 Relative error over iteration for N=21

Using $N = 21$ grid, there is still some unsmooth area around the discontinuity regime. At this number of grid, the steady criterion satisfied at $t = 1.956\ s$. Also there is noticeable oscillation in error value during iteration.

D. Steady Solution for N=41

Following is the steady solution for number of grid $N = 41$

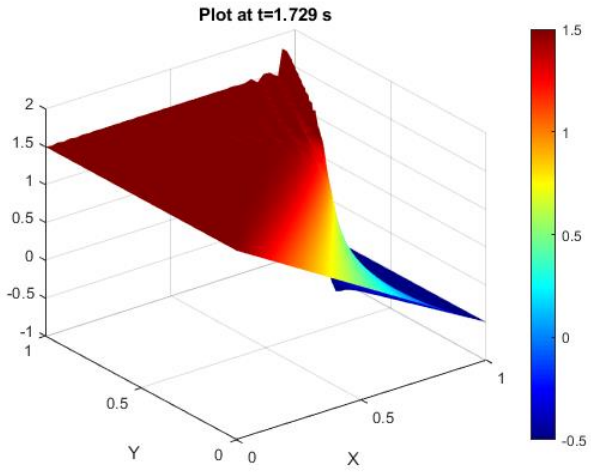


Figure 18 3D view of numerical solution evaluated at N=41

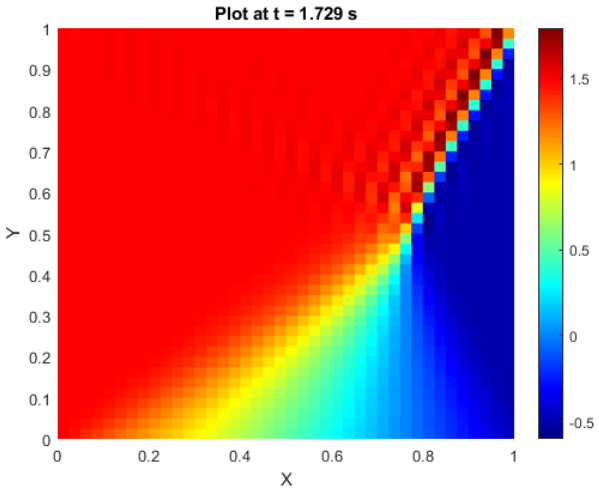


Figure 19 Top view of numerical solution evaluated at N=41

Meanwhile the relative error for each iteration is shown by following figure

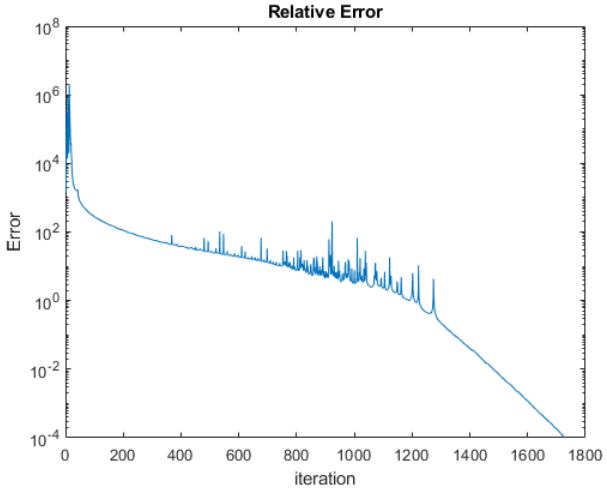


Figure 20 Relative error over iteration for N=41

Using $N = 41$ grid, there is still some unsmooth area around the discontinuity regime. At this number of grid, the steady criterion satisfied at $t = 1.729\,s$. Compared with the N=21, this result shown smoother area around the discontinuity regime, proofing the increase of numerical accuracy with the increase of number of grids. Also the convergency criterion satisfied faster with this grid configuration compared to the N=41.

E. Artificial Viscosity Variation

The usage of artificial will reduce the oscillation during iteration. Therefore, the artificial viscosity added in both cases, N=21 and N=41. The used artificial viscosity is $\nu = 10^{-5}$. This constant viscosity value selected considering very high viscosity value will produce divergent numerically meanwhile the lower value produces small effect. Following is the result contour plot and relative error for numerical scheme using artificial viscosity.

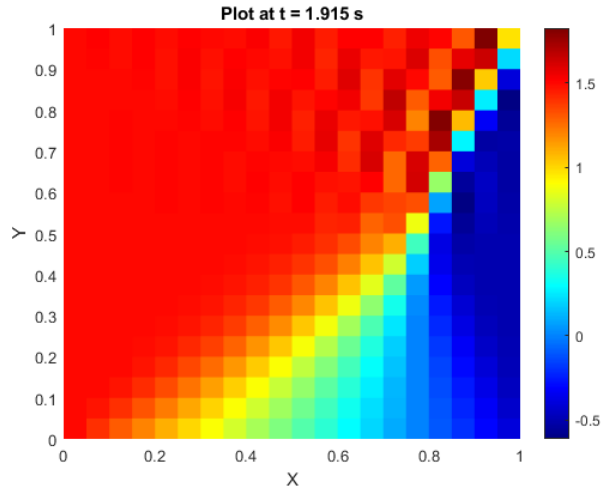


Figure 21 Top view of numerical solution using artificial visc at N=21

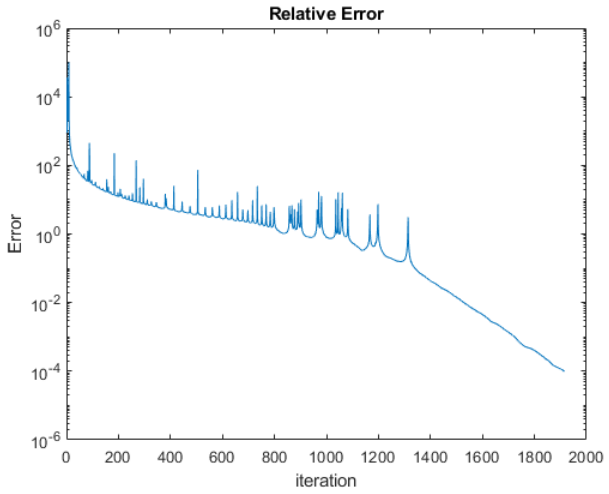


Figure 22 Relative error over iteration using artificial visc for N=21

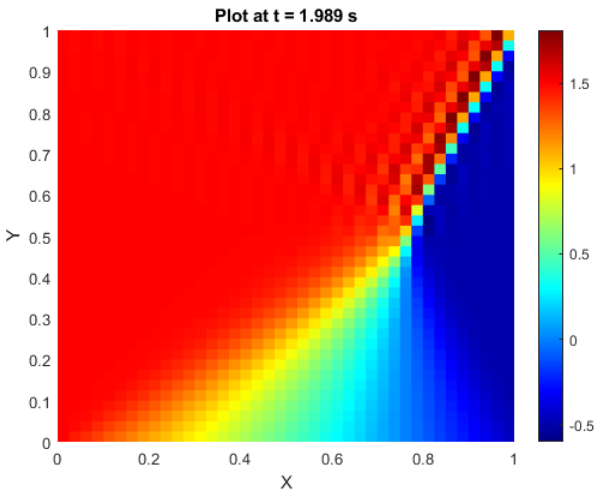


Figure 23 Top view of numerical solution using artificial visc at N=41

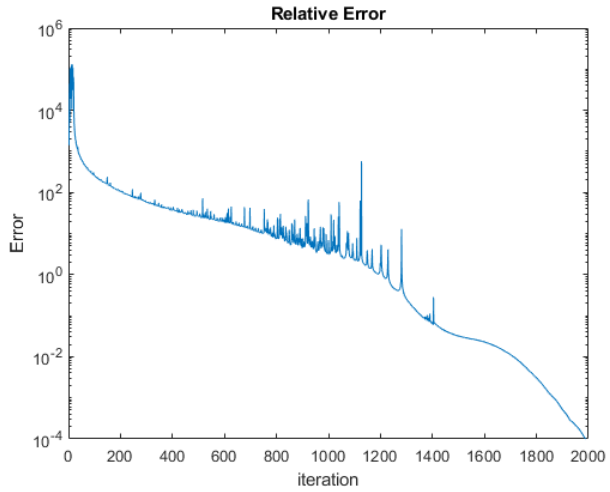


Figure 24 Relative error over iteration using artificial visc for N=41

As can be seen the artificial viscosity application produces smoother result at area above discontinuity regime. Also, the application of artificial viscosity increases the number of iterations required to achieve convergence criterion. However, the oscillation amplitude at relative error is smaller than without artificial oscillation. This numerical stability can be improved with using adaptive artificial viscosity that not covered in this numerical scheme development.

V. Conclusions

It can be concluded that the code was successfully created and able to execute the McCormack Method to solving two-dimensional Burgers' Equation. The method uses predictor and corrector scheme that proves time efficient to solve and capture discontinuity that occurs in the solution. The numerical stability during iteration oscillates due to the discontinuity. This unstability can be improved by applying artificial viscosity to the MacCormack Scheme.

VI. References

[1.] Mochammad Agoes Moelyadi, Slide Kuliah, Computational Fluid Dynamics II, 2023

[2.] Chung, T.J. (2010). Computational Fluid Dynamics (2nd ed.). Cambridge

[3.] Boyang, Li et.al. (2021) Numerical Study on Subsonic-Supersonic Laval Nozzle Using MacCormack Scheme. Journal of Physics: Conference Series

ATTACHMENT

```
clc
clear

%% Properties
% Grid Size
L = 1; % Length at X axis
H = 1; % Height at Y axis
N = 41; % Number of Uniform Nodes
dx = L/(N-1); % Spatial Increment at X axis
dy = H/(N-1); % Spatial Increment at Y axis

% Initial Condition
U1 = 1.5;
U2 = -0.5;
U3f1 = 1.5;
U3f2 = -2;

% Time marching Properties
dt = 10^-3; % Temporal Increment
nt = 5000; % Maximum Iteration
Courant = U1*0.5*((dt/dx)+(dt/dy)) %
Courant Number

% Numerical Properties
savestep = 1; % Result saved after certain
iteration
v = 10^(-4); % Artificial Viscosity
conv = 10^(-4); % Convergent Criterion

%% Analytic Solution
M = N; % Number of nodal
ua = zeros(M,M);
xp = linspace(0,L,M);
yp = linspace(0,H,M);
[X, Y] = meshgrid(xp,yp);

% Analytical Function
for i = 1:M
    for j = 1:M
        xval = X(i,j);
        yval = Y(i,j);
        if yval <= 0.5*H
            if xval <= 1.5*yval
                ua(i,j) =  1.5;
            elseif xval <= (1-0.5*yval)
                ua(i,j) = (1.5 - 2*xval)/(1
- 2*yval);
            else
                ua(i,j) = -0.5;
            end
        else
            if xval <= (0.5+0.5*yval)
                ua(i,j) = 1.5;
            else
                ua(i,j) = -0.5;
            end
        end
    end
end

% Plot Figure
figure(1)
surf(X, Y, ua,'EdgeColor', 'None',
'facecolor', 'interp');
view(2)
colorbar
colormap jet;
title('Analytic Solution')
xlabel('X')
ylabel('Y');

%% Transfinite Interpolation
% Number of nodes every side
IM = N;
JM = N;

% Generalized Coordinate
sidexi = linspace(0,1,IM);
sideeta = linspace(0,1,JM);
xi = zeros(IM,JM);
eta = zeros(IM,JM);
P = zeros(IM,JM);

% Boundary Value at Generalized Coordinate
xi(:,1) = sidexi;
xi(:,JM) = sidexi;
eta(1,:) = sideeta;
eta(IM,:) = sideeta;

% Calculate Inner Value at Generalized
Coordinate
for i = 1:IM
    for j = 1:JM
        P(i,j) = 1- (xi(i,JM) -
xi(i,1))*(eta(IM,j) - eta(1,j));
        xi(i,j) = (1/P(i,j)) *
(xi(i,1)+eta(1,j)*(xi(i,JM)-xi(i,1)));
        eta(i,j) = (1/P(i,j)) *
(eta(1,j)+xi(i,1)*(eta(IM,j)-eta(1,j)));
    end
end

% Plot Generalized Coordinate
figure(2)
plot(xi,eta,'r',xi',eta','r')
xlabel('\xi')
ylabel('\eta')

% Cartesian Coordinate
sideX = linspace(0,L,IM);
```

```
sideY = linspace(0,H,JM);
X = zeros(IM,JM);
Y = zeros(IM,JM);

% Boundary Value at Cartesian Coordinate
X(:,1) = sideX;
X(IM,:) = sideX(IM)*ones(1,JM);
X(:,JM) = sideX;
Y(1,:) = sideY;
Y(:,JM) = sideX(JM)*ones(JM,1);
Y(IM,:) = sideY;

% Calculate Inner Value at Cartesian
Coordinate
for i = 1:IM
    for j = 1:JM
        xival = xi(i,j);
        etaval = eta(i,j);
        X(i,j) = (1-xival)*X(1,j) +
xival*X(IM,j) ...
                + (1-etaval)*X(i,1) +
etaval*X(i,JM) ...
                -((1-xival)*(1-
etaval)*X(1,1) + xival*(1-
etaval)*X(IM,1) ...
                + (1-xival)*etaval*X(1,JM)
+ xival*etaval*X(IM,JM));
        Y(i,j) = (1-xival)*Y(1,j) +
xival*Y(IM,j) ...
                + (1-etaval)*Y(i,1) +
etaval*Y(i,JM) ...
                -((1-xival)*(1-
etaval)*Y(1,1) + xival*(1-
etaval)*Y(IM,1) ...
                + (1-xival)*etaval*Y(1,JM)
+ xival*etaval*Y(IM,JM));
    end
end

% Plot Cartesian Coordinate Value
figure(3)
refreshdata
plot(X,Y,'r',X',Y','r')
xlabel('X')
ylabel('Y')

%% Initial Condition
% Initial Condition Matrix
u_init = zeros(N,N);

% Initial Condition at X==0
[I_x0,J_x0] = find(X==0);
u_init(I_x0,J_x0) = U1;

% Initial Condition at X=L
[I_xL,J_xL] = find(X==L);
u_init(I_xL,J_xL) = U2;

% Initial Condition at Y=0
[I_y0,J_y0] = find(Y==0);
u_init(I_y0,J_y0) = U3f1 +
U3f2.*X(I_y0,J_y0);

% Initial Condition at Y=H
[I_y1,J_y1] = find(Y==H);
I_y2 = I_y1;
J_y2 = J_y1-1;
u_init(I_y1,J_y1) = u_init(I_y2,J_y2);

% Plot Initial Condition
figure(4)
surf(X, Y, u_init,'EdgeColor', 'None');
view(2)
colormap jet;
xlabel('X')
ylabel('Y');

%% Create time marching data storage
u_sol = cell(nt/savestep,1); % Time
marching solution
err = zeros(2,nt/savestep); % Time marching
relative error
u_sol{1} = u_init; % Assign initial
condition to the solution

%% Mac Cormac Scheme
E = zeros(N,N); % concervative variable
matrix
F = zeros(N,N); % concervative variable
matrix
up = zeros(N,N);  % predictor primitive
variable
Ep = zeros(N,N); % predictor concervative
variable
Fp = zeros(N,N); % predictor concervative
variable
uc = u_init; % corrector primitive variable

u_old = u_init; % Assign Initial Condition
u_new = zeros(N,N);
errt = 1;
% Looping
for t = 1:nt
    % Replace Old Solution
    if t~=1
        u_old = u_new;
    end

    % Calculate Concervative Variable
    for i = 1:IM
        for j = 1:JM
            E(i,j) = 0.5*u_old(i,j)^2;
```

```
            F(i,j) = u_old(i,j);
        end
    end

    % Predictor Loop
    check1 = 0;
    for j = 2:JM-1
        for i = 2:floor(IM/2)
            Sij = artvisc(i,j,v,u_old);
            up(i,j) =
FTCBS(u_old,E,F,Sij,dx,dy,dt,i,j);

            Sij = artvisc(IM+1-
i,j,v,u_old);
            up(IM+1-i,j) =
FTCBS(u_old,E,F,Sij,dx,dy,dt,IM+1-i,j);
        end
        if mod(IM,2)==1
            i=i+1;
            Sij = artvisc(i,j,v,u_old);
            up(i,j) =
FTCBS(u_old,E,F,Sij,dx,dy,dt,i,j);
        end
    end

    % Apply Boundary Condition to Predictor
Result
    up(I_x0,J_x0) = U1; % Dirichlet at X=0
    up(I_xL,J_xL) = U2; % Dirichlet at X=L
    up(I_y0,J_y0) = U3f1 +
U3f2.*X(I_y0,J_y0); % Dirichlet at Y=0
    up(I_y1,J_y1) = up(I_y2,J_y2); %
Neumann at Y=L

    % Predictor Concervative Variable
    for i = 1:IM
        for j = 1:JM
            Ep(i,j) = 0.5*up(i,j)^2;
            Fp(i,j) = up(i,j);
        end
    end

    % Corrector Loop
    for j = 2:JM-1
        for i = 2:floor(IM/2)
            Sp = artvisc(i,j,v,up);
            uc(i,j) =
FTCBS(u_old,Ep,Fp,Sp,dx,dy,dt,i,j);

            Sp = artvisc(IM+1-i,j,v,up);
            uc(IM+1-i,j) =
FTCBS(u_old,Ep,Fp,Sp,dx,dy,dt,IM+1-i,j);
        end
        if mod(IM,2)==1
            i=i+1;
            Sp = artvisc(i,j,v,up);
            uc(i,j) =
FTCBS(u_old,Ep,Fp,Sp,dx,dy,dt,i,j);
        end
    end

    % Apply Boundary Condition to
Correction
    uc(I_x0,J_x0) = U1; % Dirichlet at X=0
    uc(I_xL,J_xL) = U2; % Dirichlet at X=L
    uc(I_y0,J_y0) = U3f1 +
U3f2.*X(I_y0,J_y0); % Dirichlet at Y=0
    uc(I_y1,J_y1) = uc(I_y2,J_y2); %
Neumann at Y=L

    % Calculate solution
    check2 = 0;
    for i = 1:IM
        for j = 1:JM
            u_new(i,j) = 0.5*(up(i,j) +
uc(i,j));
        end
    end

    % Calculate Error
    errt = 0;
    for j = 1:JM
        for i = 1:IM
            errc = abs((u_new(i,j)-
u_old(i,j))/u_old(i,j));
            if ~isnan(errc) && ~isinf(errc)
&& errc<10^5
                errt = errt+errc;
            end
        end
    end

    % Save time step result
    if round(t/savestep,0) == t/savestep
        count = t/savestep;
        % Save Data
        u_sol{count} = u_new;

        % Plot Contour
        figure(5)

h=surf(X,Y,u_sol{count},'EdgeColor',
'None');
        view(2)
        title(['Plot at t =
',num2str(t*dt),' s'])
        colormap jet
        colorbar
        xlabel('X')
        ylabel('Y');
        drawnow
        refreshdata(h)
```

```
        % Save Error
        err(1,count) = errt;
        err(2,count) = t;
        figure(6)

g=semilogy(err(2,1:count),err(1,1:count));
        title('Relative Error')
        xlabel('iteration')
        ylabel('Error');
        drawnow
        refreshdata(g)
    end

    % Stop Iteration when Convergence
Criterion Reached
    if errt~=0 && errt<conv
        break
    end

end
convres = u_new; % Save lastest Solution

%% Result Plot

figure(7)
h=surf(X,Y,convres,'EdgeColor', 'None',
'facecolor', 'interp');
%view(2)
title(['Plot at t=',num2str(t*dt),' s'])
colormap jet
colorbar
clim([-0.5 1.5])
xlabel('X')
ylabel('Y');
drawnow
refreshdata(h)
hold off
time=t*dt

%% Artificial Viscosity function
function S = artvisc(i,j,v,u)
ai = u(i+1,j) - 2*u(i,j) + u(i-1,j);
aj = u(i,j+1) - 2*u(i,j) + u(i,j-1);
bi = u(i+1,j) + 2*u(i,j) + u(i-1,j);
bj = u(i,j+1) + 2*u(i,j) + u(i,j-1);
Si = (abs(ai)/bi)*ai;
Sj = (abs(aj)/bj)*aj;
S = v*0.5*(Si+Sj);
if isnan(S)
    S=0;
end
end

%% Forward Time - Central X space -
Backward Y space Numerical Scheme
```

```
function res = FTCBS(u,E,F,S,dx,dy,dt,i,j)
res = u(i,j) - (dt/dx)*0.5*(E(i+1,j) - E(i-
1,j)) ...
                - (dt/dy)*(F(i,j) - F(i,j-
1)) + S;
end
```