

**دانشگاه صنعتی امیرکبیر**

**(پلی**‏**تکنیک تهران)**

**دانشکده مهندسی مکانیک**

پروژه دینامیک سیالات محاسباتی  
حل عددی مسئله ی جریان تراکم ناپذیر سیال  
در یک حفره ی دو بعدی

**نگارش**

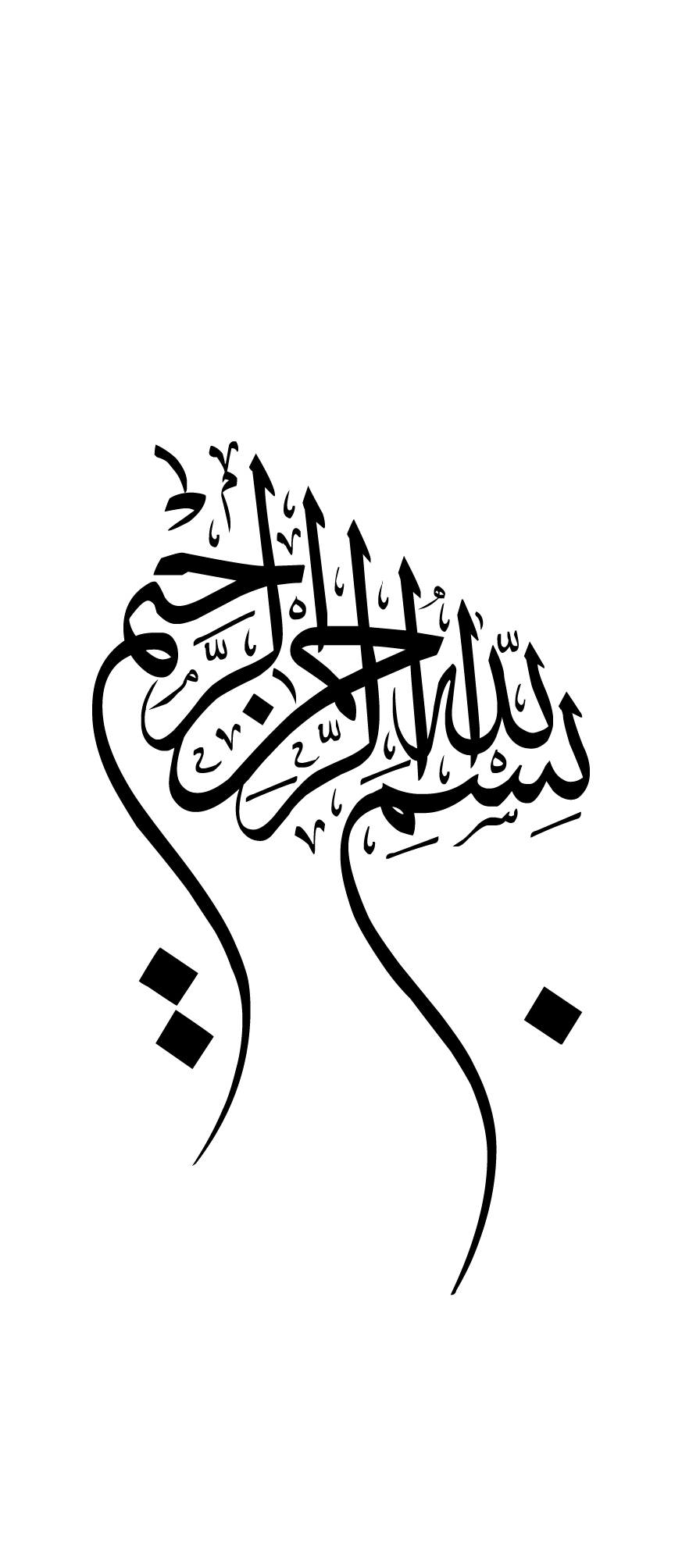
**هانیه عطریان**

**درسا طیبی**

**استاد درس**

**دکتر احمدپور**

**تیر ماه 1403**

****

فهرست مطالب

[**بخش اول : معادلات حاکم و کد ملوار** 1](#_Toc170420296)

[شرح معادلات ورتیسیه – تابع جریان 1](#_Toc170420297)

[خطوط جریان و خطوط تاوایی ثابت درون محفظه 3](#_Toc170420298)

[مقدار تاوایی بر روی تمامی دیوار ها 2](#_Toc170420299)

[**بخش دوم - اصلاح کد به روش SOR** 6](#_Toc170420300)

[شرح معادلات SOR 6](#_Toc170420301)

[پروفیل سرعت افقی بر روی خط عمودی و افقی میانی حفره 7](#_Toc170420302)

[پروفیل سرعت عمودی بر روی خط عمودی و افقی میانی حفره 9](#_Toc170420303)

[تنش برشی بر روی دیواره ی بالایی و پایینی 11](#_Toc170420304)

[**بخش سوم - مقایسه دو روش** 13](#_Toc170420305)

[کد ملوار 13](#_Toc170420306)

[روش SOR 13](#_Toc170420307)

[Under relaxation 13](#_Toc170420308)

[**بخش چهارم - معادله حاکم بر سیال دو بعدی** 15](#_Toc170420309)

[شرح معادلات دو بعدی 15](#_Toc170420310)

[**کد های متلب** 17](#_Toc170420311)

[کد ملوار اصلاح شده 17](#_Toc170420312)

[کد SOR 19](#_Toc170420313)

[کد بخش امتیازی 27](#_Toc170420314)

[**منابع و مراجع** 30](#_Toc170420315)

# **بخش اول : معادلات حاکم و کد ملوار**

## شرح معادلات ورتیسیه – تابع جریان

معادلات تراکم ناپذیر ناویر استوکس برای دو بعد به شرح زیر می باشند :

برای حذف کردن ترم فشار از معادلات، از معادله ی مومنتوم x نسبت به y و از معادله ی مومنتوم y نسبت به x مشتق گرفته شده و در نهایت با هم جمع می شوند.

ورتیسیه به صورت زیر تعریف می شود :

با جایگذاری ورتیسیته در معادله تفاضل شده معادله ی انتقال ورتیسیته به دست می آید :

هم چنین می توان سرعت ها را به صورت تابعی از تابع جریان نوشت :

با جایگذاری سرعت های فوق در معادله ی انتقال ورتیسیته این معادله به صورت زیر بازنویسی می شود.

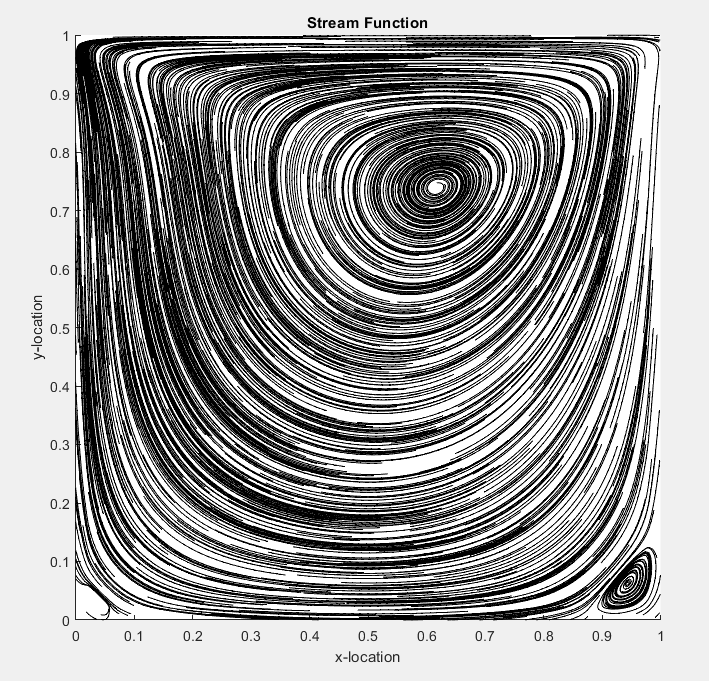
شرط های مرزی تابع جریان :

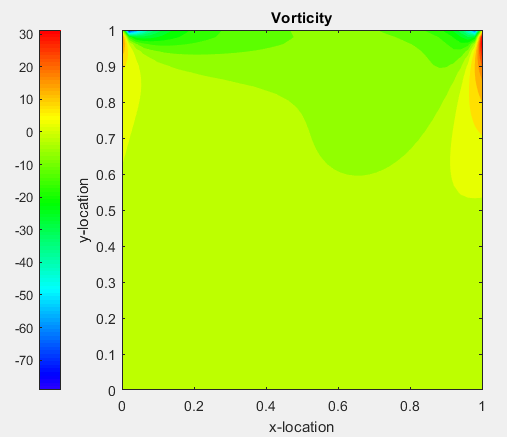
شرط های مرزی ورتیسیته :

گسسته سازی سرعت ها نیز به شکل زیر می باشد.

## خطوط جریان و خطوط تاوایی ثابت درون محفظه

خطوط جریان درون محفظه و خطوط تاوایی ثابت برای رینولدز برابر با 100 و سرعت دیواره برابر با 1 متربرثانیه و تعداد نودهای برابر با 52 به شرح زیر می باشد :

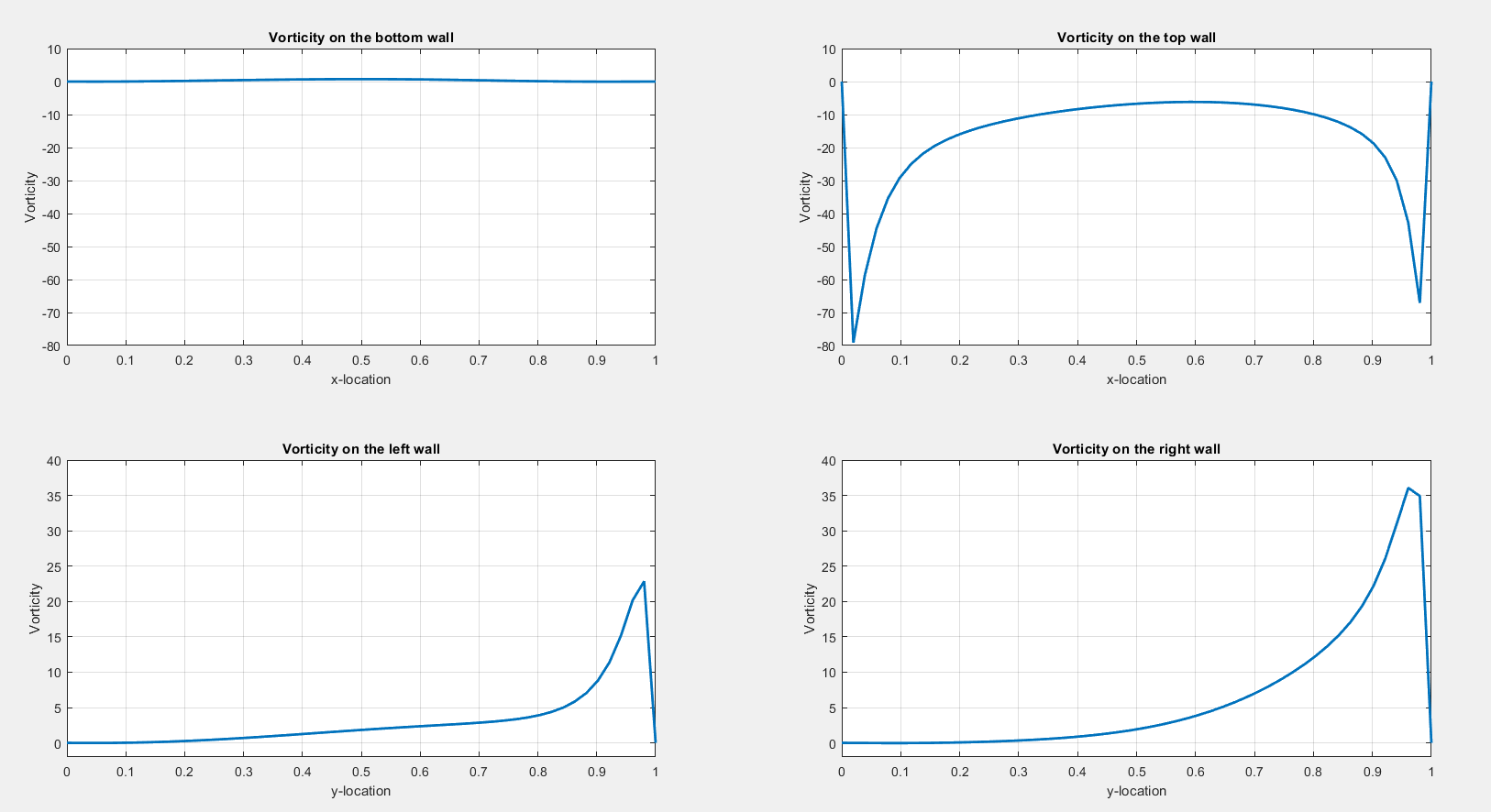


شکل 1 – خطوط جریان داخل محفظه

شکل 2 – خطوط تاوایی ثابت داخل محفظه

## مقدار تاوایی بر روی تمامی دیوار ها

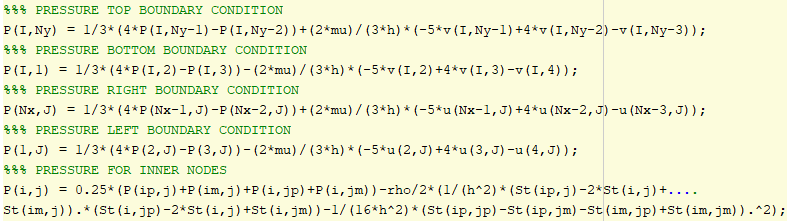
همچنین مقدار تاوایی بر روی 4 دیوار به تفکیک به صورت زیر می باشد :



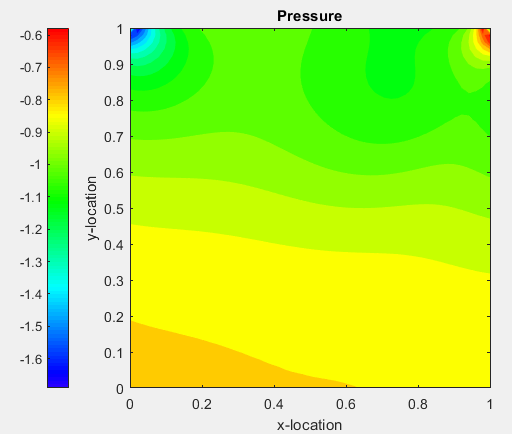
شکل 3 – مقدار تاوایی بر روی دیوارها

کانتور فشار

برای به دست آوردن فشارها باید از جاگذاری سرعت ها در معادله تکانه اصلی استفاده نمود.



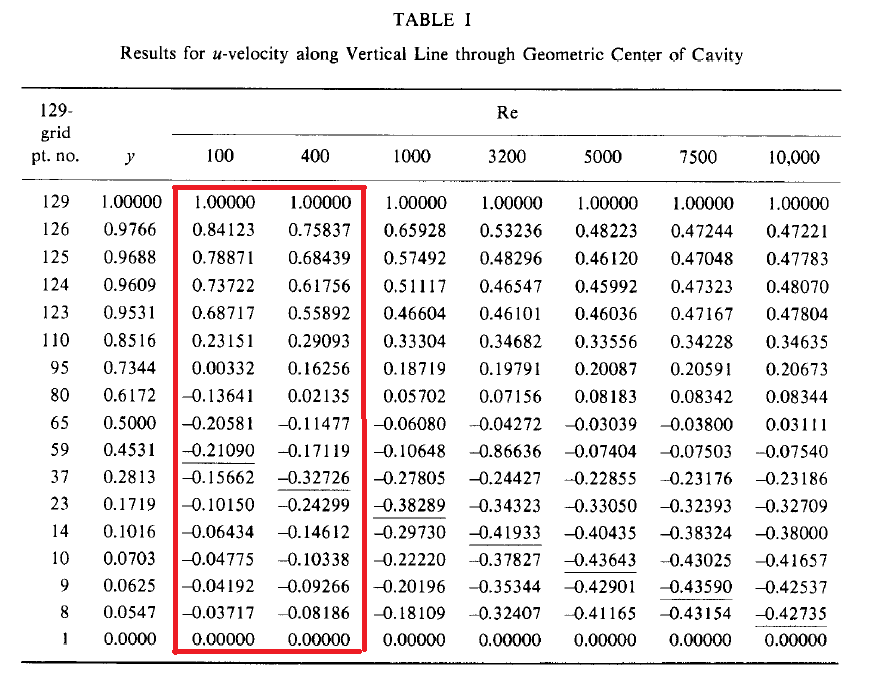
در نتیجه کانتور فشار حاصله با 52 نود و رینولدز 100 و سرعت دیواره 1 متر بر ثانیه به صورت زیر است :



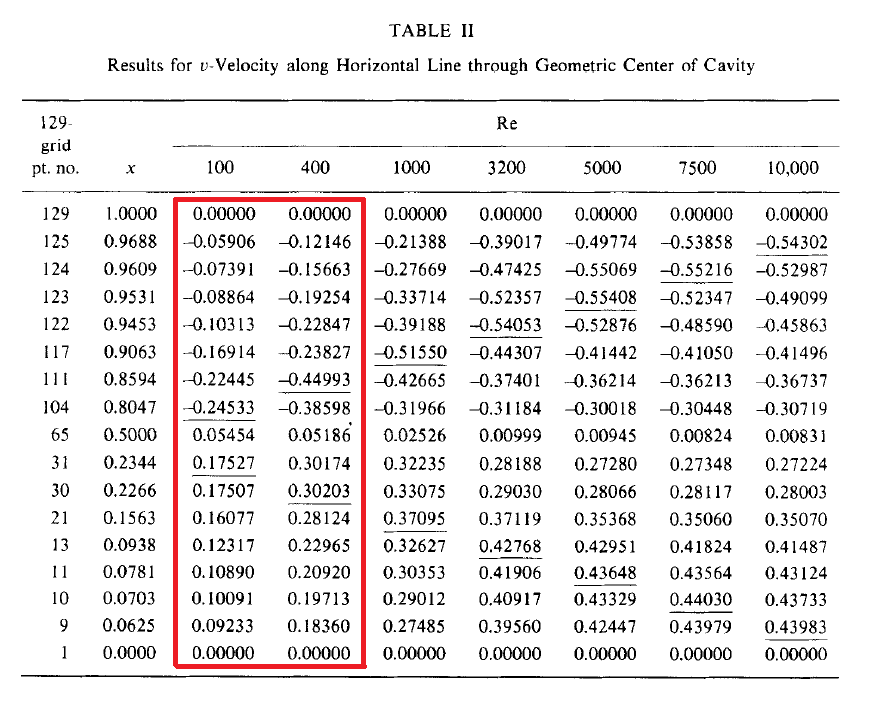
شکل 4 – کانتور فشار

مقایسه سرعت ها

سرعت های نگارش شده در مقاله قیا در رینولدز برابر با 100 و 400 و تعداد نود های 129 و سرعت دیواره برابر با 1 متر بر ثانیه به شرح زیر می باشد :

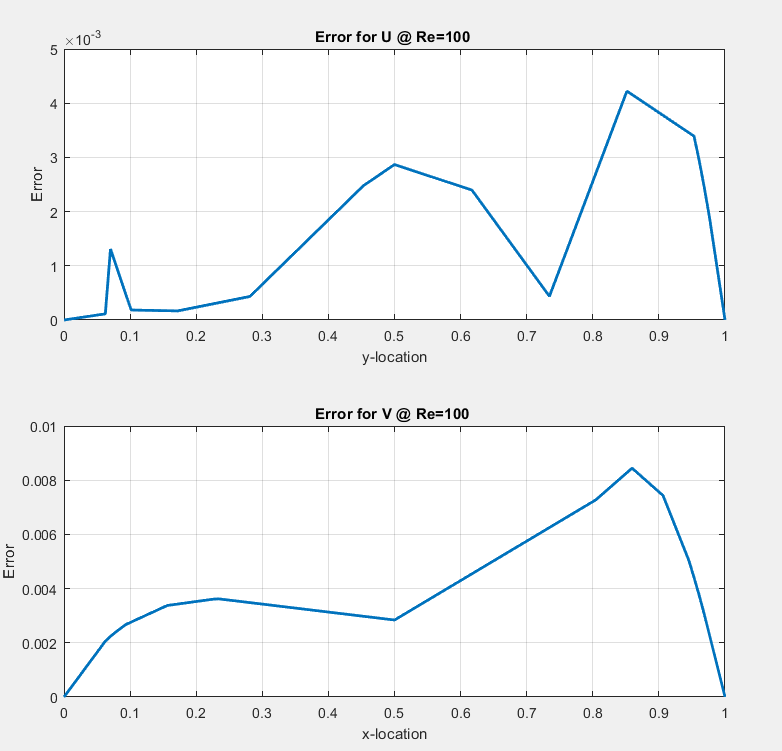


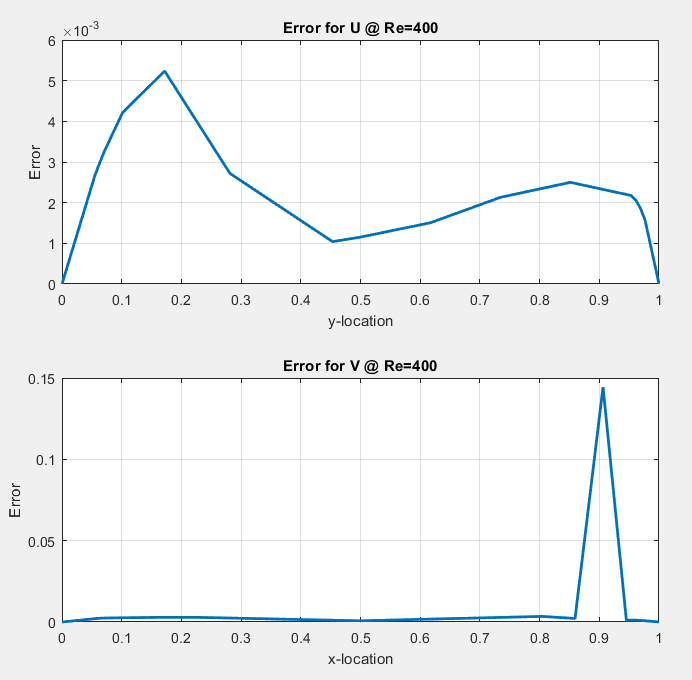
شکل 5 – مقادیر u برای رینولدز های 100 و 400 در مقاله قیا



شکل 6 – مقادیر v برای رینولدز های 100 و 400 در مقاله قیا

در نهایت مقایسه ی سرعت های به دست آمده و سرعت های مقاله ی قیا در رینولدز های 100 و 400 با تعداد نود 129 و سرعت دیواره برابر با 1 متر برثانیه به صورت زیر است :



شکل 7 – خطای مطلق سرعت ها در رینولدز برابر با 100

شکل 8 – خطای مطلق سرعت ها در رینولدز برابر با 400

شکل 8 – خطای مطلق سرعت ها در رینولدز برابر با 400

# **بخش دوم - اصلاح کد به روش SOR**

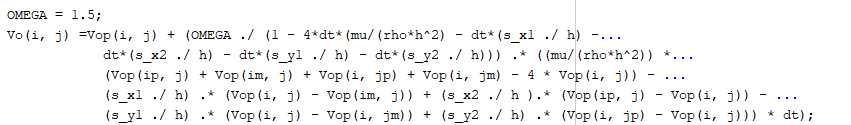
## شرح معادلات SOR

برای استخراج گسسته سازی برای اجرای کد مورد نظر نیاز به پیاده سازی روش upwind و SOR روی معادله انتقال ورتیسیته به طور همزمان می باشد.

جملات جابجایی در معادله زیر باید به گونه ای بازنویسی شوند که با توجه به جهت حرکت موج رونده، داده های گسسته سازی شده از بالادست برداشته شوند.

همچنین برای روش تکراری SOR ضریبی به اسم ضریب over relaxation(W) تعریف می شود که تعریف این روش به شرح زیر می باشد:

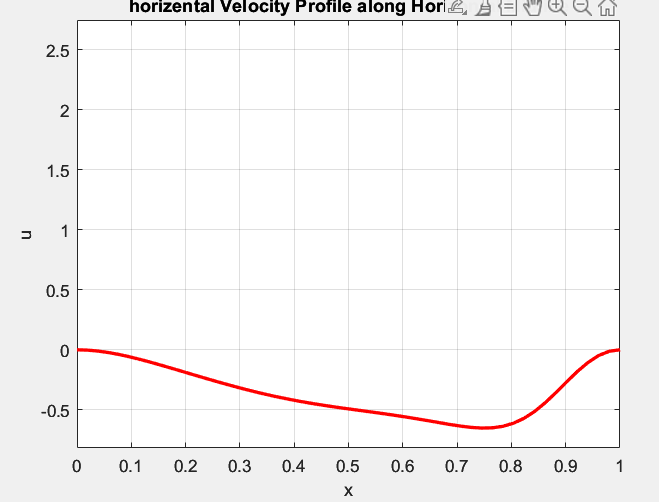
و در نهایت کد با به فرم زیر حاصل می شود :



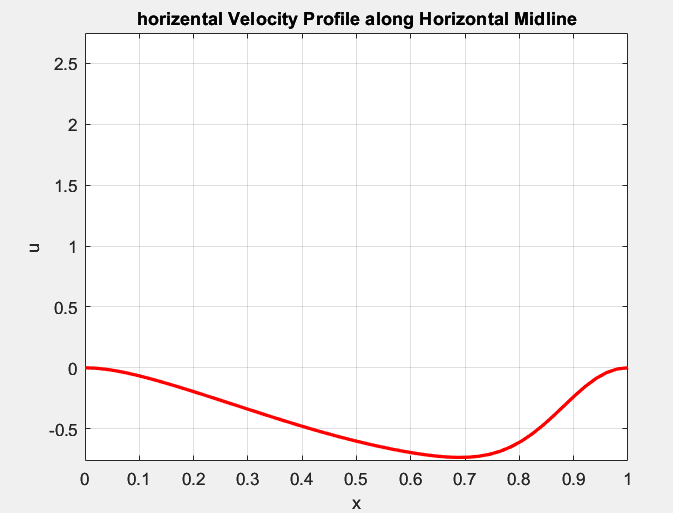
## پروفیل سرعت افقی بر روی خط عمودی و افقی میانی حفره

در این قسمت با تعداد نودهای 52 کد عددی اصلاح شده با روش بالادست و SOR با یکدیگر مقایسه شده اند.

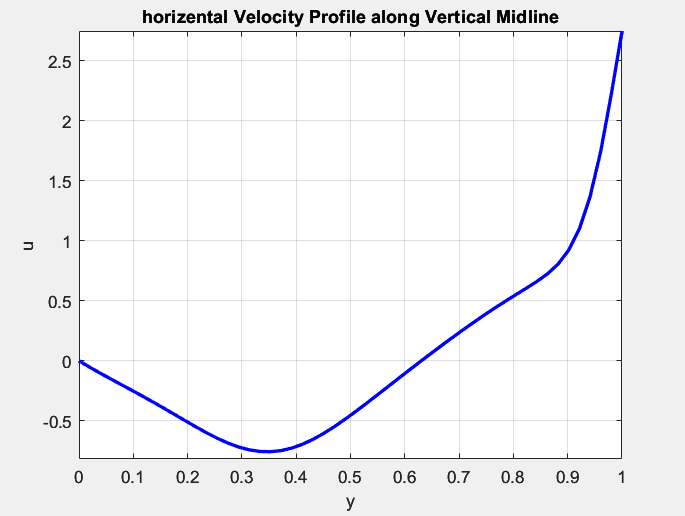
سرعت دیواره 2.75 متر بر ثانیه می باشد و نمودارهای مربوطه در زیر اورده شده است. لازم به ذکر است هر دو کد در حالت گذرا گسسته سازی شده اند.



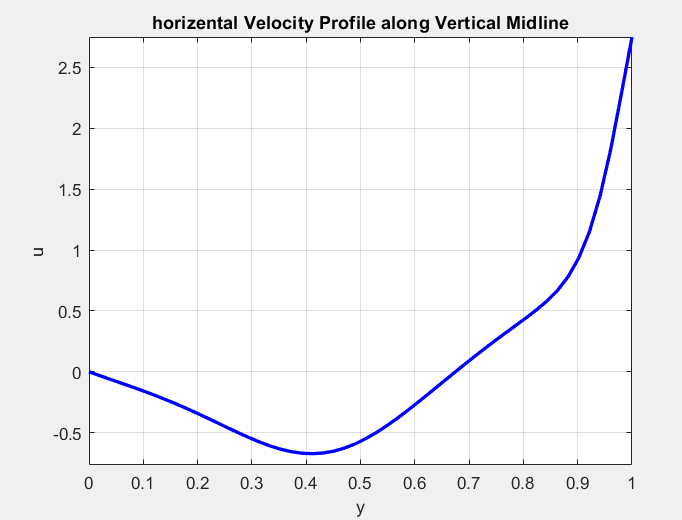
شکل 9 – سرعت در راستای افقی روی خط افقی میانی حفره در روش ملوار



شکل 10 – سرعت در راستای افقی روی خط افقی میانی حفره در روش SOR

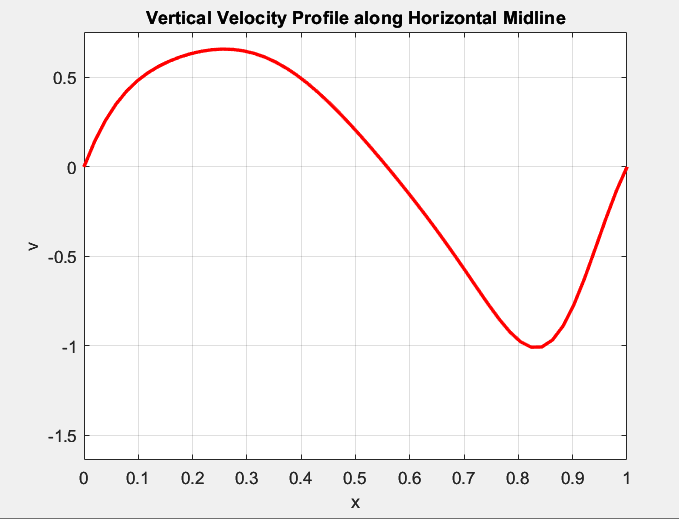


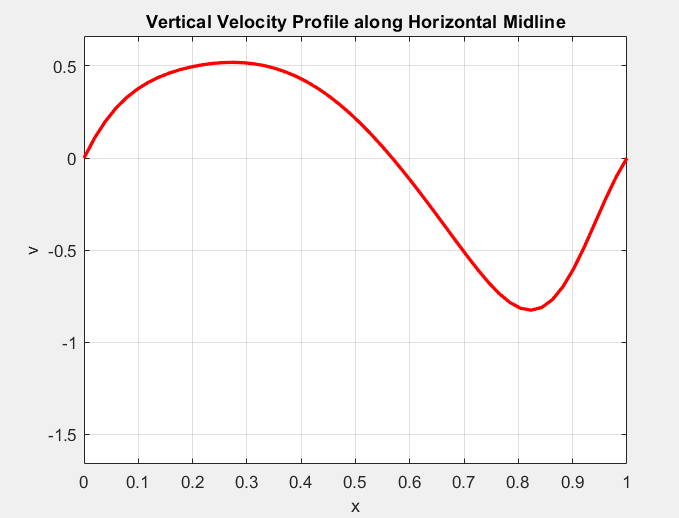
شکل 11 – سرعت در راستای افقی روی خط عمودی میانی حفره در روش ملوار



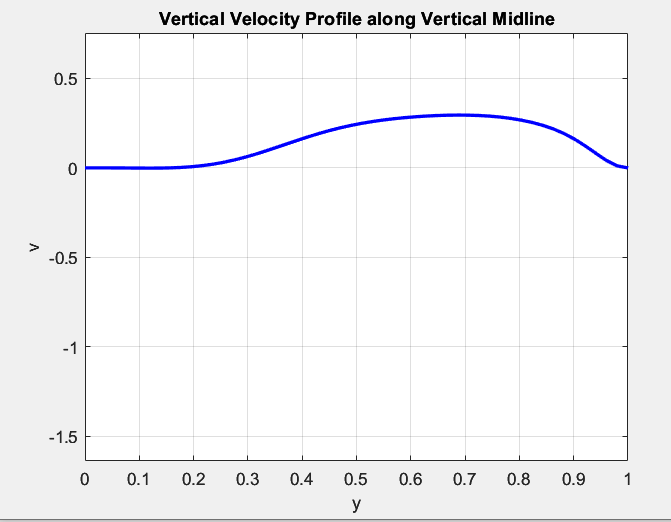
شکل 12 – سرعت در راستای افقی روی خط عمودی میانی حفره در روش SOR

## پروفیل سرعت عمودی بر روی خط عمودی و افقی میانی حفره

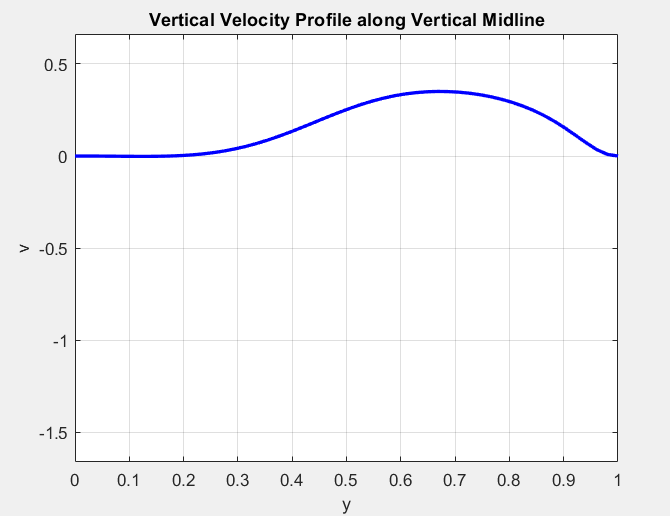


شکل 13 – سرعت در راستای عمودی روی خط افقی میانی حفره در روش ملوار

شکل 14 – سرعت در راستای عمودی روی خط افقی میانی حفره در روش SOR



شکل 15 – سرعت در راستای عمودی روی خط عمودی میانی حفره در کد ملوار



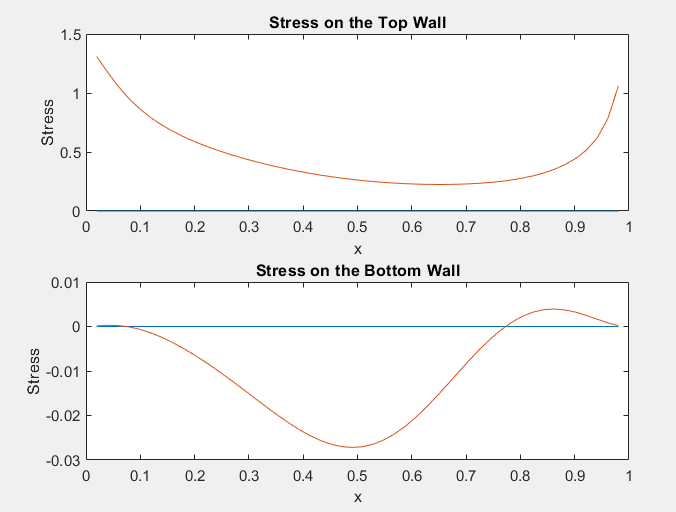
شکل 16 – سرعت در راستای افقی روی خط عمودی میانی حفره در روش SOR

## تنش برشی بر روی دیواره ی بالایی و پایینی

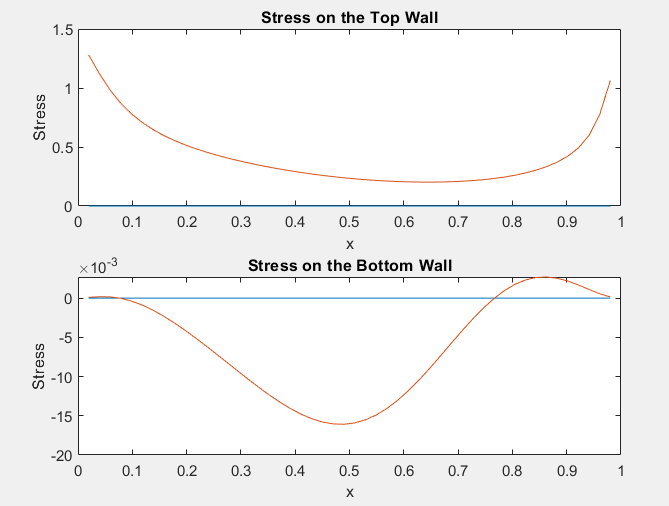
برای محاسبه ی تنش های برش روی دیواره های بالا و پایین به شکل زیر عمل میکنیم:

تنش برشی در دیواره ی پایینی و بالایی در کد :





شکل 17 – تنش برشی روی دیواره ی بالایی و پایینی در کد ملوار

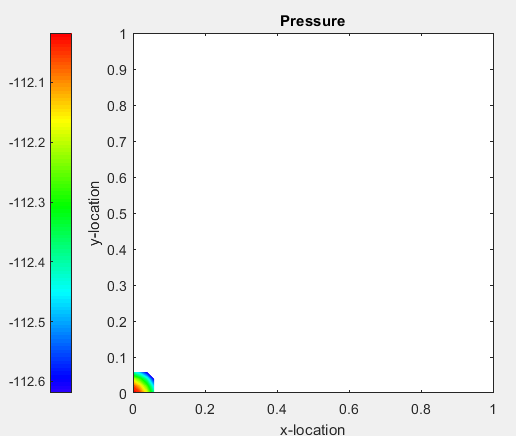


شکل 18 – تنش برشی روی دیواره ی بالایی و پایینی در در روش SOR

# **بخش سوم - مقایسه دو روش**

## کد ملوار

با تغییر عدد رینولدز و سرعت دیواره بالایی در کد ملوار از رینولدز برابر با 1326 و سرعت دیواره برابر با 13.26 متر بر ثانیه به بعد نمودار ها دچار اختلال می شوند و نتایجی ارائه نمی دهند.



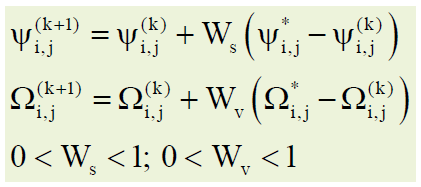
شکل 19 – نمونه کانتور فشار ناصحیح در رینولدز برابر با 1327

## روش SOR

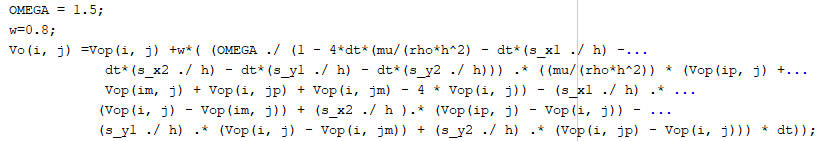
در کدمتلب با روش SOR نیز مشاهده می شود با گذر از عددرینولدز برابر با 1725 و سرعت دیواره بالایی برابر با 17.25 متر بر ثانیه، نمودار ها دچار اختلال می شوند.

## Under relaxation

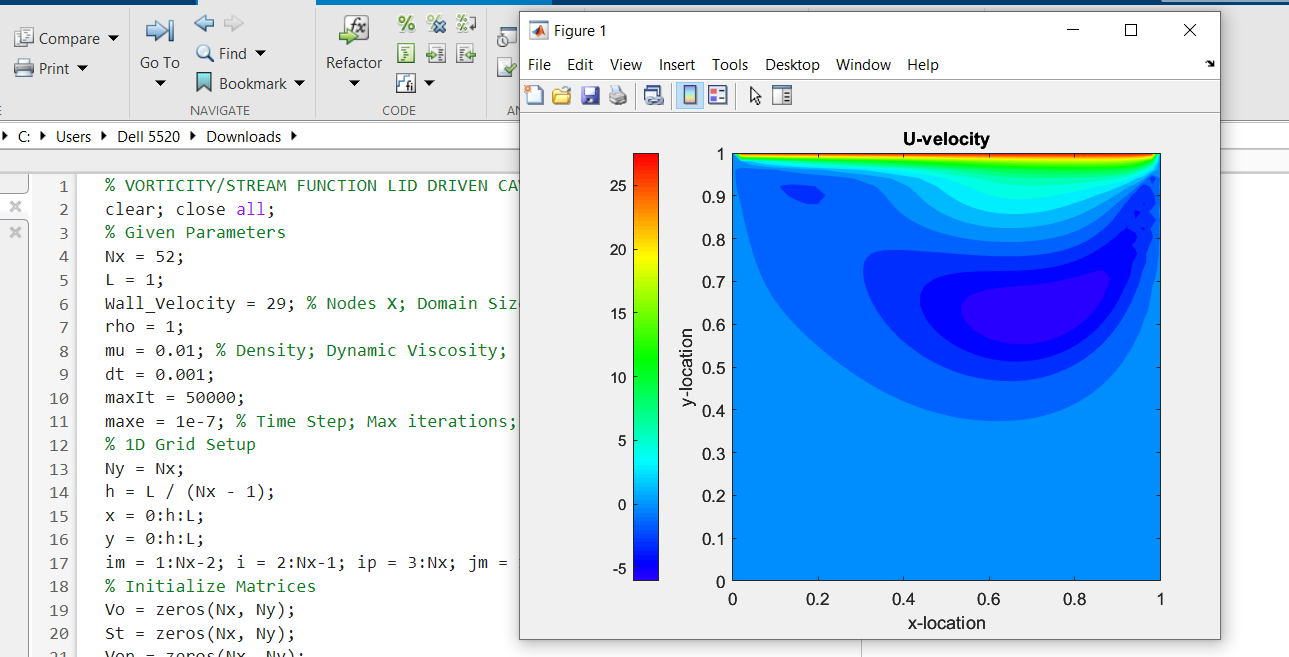
در این بخش از خواسته ی سوال، ضریب under relaxation بکار برده می شود که به همگرایی بهتر کمک میکند و مقداری بین صفر و یک دارد و به شکل زیر تعریف می شود:



شکل 20 – نحوه کارکرد ضریب under relaxation



همانطور که در شکل زیر پیداست با اضافه کردن ضریب under relaxation عدد رینولدز تا عدد 2900 نیز افزوه شده است اما در رینولدزهای بالاتر از 2925 دیگر نمودارها رسم نمی شوند .پس واضح است که این ضریب بازه ی سرعت دیواره ی بالا را افزایش داده است.



شکل 21 – نمونه ای از استفاده از ضریب under relaxation با رینولدز 2900

# **بخش چهارم - معادله حاکم بر سیال دو بعدی**

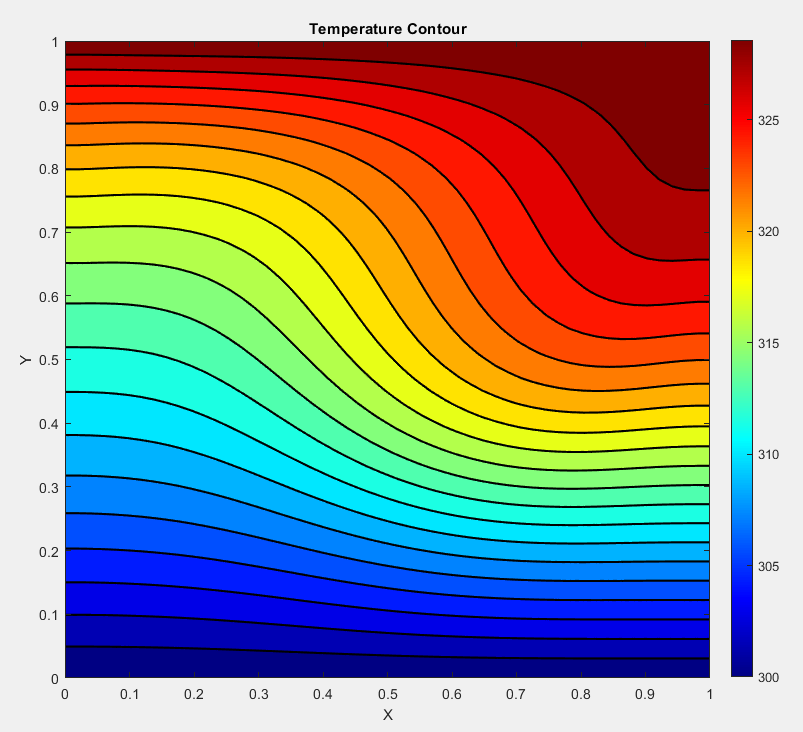
## شرح معادلات دو بعدی

در این قسمت معادله حاکم بر انتقال ورتیسیته به شکل دائمی حل می شود در نتیجه در کد قسمت SOR گسسته سازی زمانی در نظر گرفته نمی شود و معادله به شکل زیر با روش گوس سایدل حل شده است :

همچنین معادلات انرژی با این معادلات حل می شود، به این صورت که سرعت های (u و v) محاسبه شده از حل معادلات پواسون انتقال ورتیسیته و تابع جربان، در معادله دما جاگذاری می شود و دما در مش مشخص شده محاسبه می گردد.

نحوه حل شدن معادله انرژی همانند انتقال ورتیسیته است که به شکل دائمی و با روش تفاضل بالادست (upwinding) حل می شود. شرایط مرزی نیز برای دیوارهای اطراف با مشتق یک طرفه (نیومن) و دریشله در کد مشخص شده است.

درشکل زیر برای سرعت دیواره بالایی 10 متربر ثانیه کانتور دمایی رسم شده است همانطور که در شکل مشاهده می شود بر روی دیواره های جانبی شیب ثابت است که نشان دهنده ی شرط مرزی نیومن صفر می باشد و در دیواره های بالایی و پایینی دمای ثابت مشهود است.



شکل 22 – کانتور دما بر روی محفظه

بر اساس کانتور دمایی که از حل معادله انرژی به دست آمده است، توزیع عدد ناسلت (Nu) در دیواره بالایی و پایینی می‌تواند اطلاعات مهمی را درباره انتقال حرارت در این نواحی ارائه دهد. برای این منظور، ابتدا باید تعریف عدد ناسلت را بررسی کنیم. عدد ناسلت بیانگر نسبت انتقال حرارت همرفتی به انتقال حرارت هدایتی در یک سیال است.

* **دیواره بالایی) x=L)** دمای دیواره بالایی بالاتر است (330 K) و از آنجا که گرادیان دما به طور معمول در این ناحیه بالا خواهد بود، می‌توان نتیجه گرفت که انتقال حرارت در این ناحیه بیشتر است. بنابراین عدد ناسلت در این ناحیه احتمالاً بیشتر خواهد بود.
* **دیواره پایینی (x=0)**: دمای دیواره پایینی کمتر است (300 K) و گرادیان دما در این ناحیه نیز می‌تواند کمتر باشد. بنابراین عدد ناسلت در این ناحیه احتمالاً کمتر از دیواره بالایی خواهد بود.

بنابراین، به طور کلی می‌توان گفت که عدد ناسلت در دیواره بالایی به دلیل گرادیان دمایی بیشتر، بالاتر است و انتقال حرارت همرفتی در این ناحیه بیشتر خواهد بود. این نتیجه با مشاهده کانتور دمایی و توزیع دما در دیواره‌ها هماهنگ است.

# **کد های متلب**

## کد ملوار اصلاح شده

%% VORTICITY/STREAM FUNCTION LID DRIVEN CAVITY FLOW SOLVER JOE MOLVAR

clear;

clc;

close all

%%% GIVENS

Nx = 52; %Nodes

L = 1; %Domain Size

rho = 1;% Density

mu = 0.01; %Dynamic Viscosity

Re=100; %Reynolds

Wall\_Velocity =1; %Velocity

dt = 0.001;%Time Step

maxIt = 50000; %Max iter

maxe = 1e-7; %Max error

%%% SETUP 1D GRID

Ny = Nx; h=L/(Nx-1); x = 0:h:L; y = 0:h:L;

im = 1:Nx-2; i = 2:Nx-1; ip = 3:Nx; jm = 1:Ny-2; j = 2:Ny-1; jp = 3:Ny; I=1:Nx; J=1:Ny;

%%% PRELOCATE MATRIXES

Vo = zeros(Nx,Ny); St = Vo; Stp = Vo; Vop = Vo; u = Vo; v = Vo; P = Vo; epsilon = 0;

%%% VELOCITY ON THE UPPER WALL(NO SLIP CONDITION)

u(2:Nx-1,Ny) = Wall\_Velocity;

%%% SOLVE LOOP SIMILAR TO GAUSS-SIEDEL METHOD

for iter = 1:maxIt

%%% CREATE BOUNDARY CONDITIONS

Vo(1:Nx,Ny) = -2\*St(1:Nx,Ny-1)/(h^2) - Wall\_Velocity\*2/h; % Top

Vo(1:Nx,1) = -2\*St(1:Nx,2) /(h^2); % Bottom

Vo(1,1:Ny) = -2\*St(2,1:Ny) /(h^2); % Left

Vo(Nx,1:Ny) = -2\*St(Nx-1,1:Ny)/(h^2); % Right

%%% PARTIALLY SOLVE VORTICITY TRANSPORT EQUATION

Vop = Vo;

Stp = St;

Vo(i,j) = Vop(i,j) + ...

(-1\*(St(i,jp)-St(i,jm))/(2\*h) .\* (Vop(ip,i)-Vop(im,j))/(2\*h)+...

(St(ip,j)-St(im,j))/(2\*h) .\* (Vop(i,jp)-Vop(i,jm))/(2\*h)+...

mu/rho\*(Vop(ip,j)+Vop(im,j)-4\*Vop(i,j)+Vop(i,jp)+Vop(i,jm))/(h^2))\*dt;

%%% PARTIALLY SOLVE ELLIPTICAL VORTICITY EQUATION FOR STREAM FUNCTION

St(i,j) = (Vo(i,j)\*h^2 + St(ip,j) + St(i,jp) + St(i,jm) + St(im,j))/4;

%%% CREATE VELOCITY FROM STREAM FUNCTION

u(i,j) = (St(i,jp)-St(i,jm))/(2\*h); v(i,j) = (-St(ip,j)+St(im,j))/(2\*h);

%%% PRESSURE TOP BOUNDARY CONDITION

P(I,Ny) = 1/3\*(4\*P(I,Ny-1)-P(I,Ny-2))+(2\*mu)/(3\*h)\*(-5\*v(I,Ny-1)+4\*v(I,Ny-2)-v(I,Ny-3));

%%% PRESSURE BOTTOM BOUNDARY CONDITION

P(I,1) = 1/3\*(4\*P(I,2)-P(I,3))-(2\*mu)/(3\*h)\*(-5\*v(I,2)+4\*v(I,3)-v(I,4));

%%% PRESSURE RIGHT BOUNDARY CONDITION

P(Nx,J) = 1/3\*(4\*P(Nx-1,J)-P(Nx-2,J))+(2\*mu)/(3\*h)\*(-5\*u(Nx-1,J)+4\*u(Nx-2,J)-u(Nx-3,J));

%%% PRESSURE LEFT BOUNDARY CONDITION

P(1,J) = 1/3\*(4\*P(2,J)-P(3,J))-(2\*mu)/(3\*h)\*(-5\*u(2,J)+4\*u(3,J)-u(4,J));

%%% PRESSURE FOR INNER NODES

P(i,j) = 0.25\*(P(ip,j)+P(im,j)+P(i,jp)+P(i,jm))-rho/2\*(1/(h^2)\*(St(ip,j)-2\*St(i,j)+....

St(im,j)).\*(St(i,jp)-2\*St(i,j)+St(i,jm))-1/(16\*h^2)\*(St(ip,jp)-St(ip,jm)-St(im,jp)+St(im,jm)).^2);

%%% CALCULATING EPSILON

epsilon(1,iter) = max(abs(St-Stp),[],'all');

%%% CHECK FOR CONVERGENCE

if iter > 10

error = max(max(Vo - Vop));

if error < maxe

break;

end

end

end

%%% PLOTS

cm = hsv(ceil(100/0.7)); cm = flipud(cm(1:100,:));

figure(1); contourf(x,y,u',23,'LineColor','none');

title('U-velocity'); xlabel('x-location'); ylabel('y-location')

axis('equal',[0 L 0 L]); colormap(cm); colorbar('westoutside');

figure(2); plot(y,u(round(Ny/2),:));

title('Centerline x-direction velocity');

xlabel('y/L'); ylabel('u/U'); axis('square'); xlim([0 L]); grid on

N = 1000; xstart = max(x)\*rand(N,1); ystart = max(y)\*rand(N,1);

[X,Y] = meshgrid(x,y);

figure(3); h=streamline(X,Y,u',v',xstart,ystart,[0.1, 200]);

title('Stream Function'); xlabel('x-location'); ylabel('y-location')

axis('equal',[0 L 0 L]); set(h,'color','k')

figure(4);

contourf(x,y,P',20,'LineColor','none');

title('Pressure'); xlabel('x-location'); ylabel('y-location'); axis('equal',[0 L 0 L]); colormap(cm); colorbar('westoutside');

figure(5);

subplot(2,2,1);

plot(x,Vo(:,1),'LineWidth',2);

title('Vorticity on the bottom wall'); xlabel('x-location'); ylabel('Vorticity'); ylim([-80 10]);

grid on

subplot(2,2,2);

plot(x,Vo(:,Ny),'LineWidth',2);

title('Vorticity on the top wall'); xlabel('x-location'); ylabel('Vorticity'); ylim([-80 10]);

grid on

subplot(2,2,3);

plot(y,Vo(1,:),'LineWidth',2);

title('Vorticity on the left wall'); xlabel('y-location'); ylabel('Vorticity'); ylim([-2 40]);

grid on

subplot(2,2,4);

plot(y,Vo(Nx,:),'LineWidth',2);

title('Vorticity on the right wall'); xlabel('y-location'); ylabel('Vorticity'); ylim([-2 40]);

grid on

figure(6);

contourf(x,y,Vop',20,'LineColor','none');

title('Vorticity'); xlabel('x-location'); ylabel('y-location'); axis('equal',[0 L 0 L]); colormap(cm); colorbar('westoutside');

## کد SOR

% VORTICITY/STREAM FUNCTION LID DRIVEN CAVITY FLOW SOLVER BY JOE MOLVAR

clear; close all;

% Given Parameters

Nx = 52;

L = 1;

Wall\_Velocity = 2.75; % Nodes X; Domain Size; Wall Velocity

rho = 1;

mu = 0.01; % Density; Dynamic Viscosity;

dt = 0.001;

maxIt = 50000;

maxe = 1e-7; % Time Step; Max iterations; Max error

% 1D Grid Setup

Ny = Nx;

h = L / (Nx - 1);

x = 0:h:L;

y = 0:h:L;

im = 1:Nx-2; i = 2:Nx-1; ip = 3:Nx; jm = 1:Ny-2; j = 2:Ny-1; jp = 3:Ny;

% Initialize Matrices

Vo = zeros(Nx, Ny);

St = zeros(Nx, Ny);

Vop = zeros(Nx, Ny);

u = zeros(Nx, Ny);

v = zeros(Nx, Ny);

u(2:Nx-1, Ny) = Wall\_Velocity;

Vop = Vo;

% Solve Loop (Similar to Gauss-Siedel Method)

for iter = 1:maxIt

% Boundary Conditions

Vo(:, Ny) = -2 \* St(:, Ny-1) / (h^2) - Wall\_Velocity \* 2 / h; % Top

Vo(:, 1) = -2 \* St(:, 2) / (h^2); % Bottom

Vo(1, :) = -2 \* St(2, :) / (h^2); % Left

Vo(Nx, :) = -2 \* St(Nx-1, :) / (h^2); % Right

% Partially Solve Vorticity Transport Equation

s\_x1 = max(u(i, j), 0);

s\_x2 = max(-u(i, j), 0);

s\_y1 = max(v(i, j), 0);

s\_y2 = max(-v(i, j), 0);

% SOR Calculation

OMEGA = 1.5;

Vo(i, j) =Vop(i, j) + (OMEGA ./ (1 - 4\*dt\*(mu/(rho\*h^2) - dt\*(s\_x1 ./ h) -...

dt\*(s\_x2 ./ h) - dt\*(s\_y1 ./ h) - dt\*(s\_y2 ./ h))) .\* ((mu/(rho\*h^2)) \* (Vop(ip, j) + Vop(im, j) + Vop(i, jp) + Vop(i, jm) - 4 \* Vop(i, j)) - ...

(s\_x1 ./ h) .\* (Vop(i, j) - Vop(im, j)) + (s\_x2 ./ h ).\* (Vop(ip, j) - Vop(i, j)) - ...

(s\_y1 ./ h) .\* (Vop(i, j) - Vop(i, jm)) + (s\_y2 ./ h) .\* (Vop(i, jp) - Vop(i, j))) \* dt);

% Partially Solve Elliptical Vorticity Equation for Stream Function

St(i, j) = (Vo(i, j) \* h^2 + St(ip, j) + St(i, jp) + St(i, jm) + St(im, j)) / 4;

u(i, j) = (St(i, jp) - St(i, jm)) / (2 \* h);

v(i, j) = (-St(ip, j) + St(im, j)) / (2 \* h);

% Check for Convergence

if iter > 10

error = max(max(abs(Vo - Vop)));

if error < maxe

break;

end

end

Vop = Vo;

end

% Plots

cm = hsv(ceil(100 / 0.7));

cm = flipud(cm(1:100, :));

figure(1);

contourf(x, y, u', 23, 'LineColor', 'none');

title('U-velocity');

xlabel('x-location');

ylabel('y-location');

% contourf(x, y, v', 23, 'LineColor', 'none');

% title('V-velocity');

% xlabel('x-location');

% ylabel('y-location');

axis equal;

axis([0 L 0 L]);

colormap(cm);

colorbar('westoutside');

figure(2);

plot(y, u(round(Ny/2), :));

title('Centerline x-direction velocity');

xlabel('y/L');

ylabel('u/U');

axis square;

xlim([0 L]);

grid on;

N = 1000;

xstart = L \* rand(N, 1);

ystart = L \* rand(N, 1);

[X, Y] = meshgrid(x, y);

figure(3);

h = streamline(X, Y, u', v', xstart, ystart, [0.1, 200]);

title('Stream Function');

xlabel('x-location');

ylabel('y-location');

axis equal;

set(h, 'color', 'k');

% a and b

%Find indices for the middle vertical and horizontal lines

%Define the midpoints for vertical and horizontal midlines

midpoint\_x = round(Nx / 2);

midpoint\_y = round(Ny / 2);

% Extract the vertical velocity profile (v) along the midlines

v\_vertical\_midline = v(midpoint\_x, :);

v\_horizontal\_midline = v(:, midpoint\_y);

% Plot the vertical velocity profile along the vertical midline

figure;

plot(y, v\_vertical\_midline, 'b-', 'LineWidth', 2);

xlabel('y');

ylabel('v');

title('Vertical Velocity Profile along Vertical Midline');

grid on;

ylim([min(v(:)), max(v(:))]);

% Plot the vertical velocity profile along the horizontal midline

figure;

plot(x, v\_horizontal\_midline, 'r-', 'LineWidth', 2);

xlabel('x');

ylabel('v');

title('Vertical Velocity Profile along Horizontal Midline');

grid on;

ylim([min(v(:)), max(v(:))]);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%5

% Extract the vertical velocity profile (v) along the midlines

u\_vertical\_midline = u(midpoint\_x, :);

u\_horizontal\_midline = u(:, midpoint\_y);

% Plot the vertical velocity profile along the vertical midline

figure;

plot(y, u\_vertical\_midline, 'b-', 'LineWidth', 2);

xlabel('y');

ylabel('u');

title('horizental Velocity Profile along Vertical Midline');

grid on;

ylim([min(u(:)), max(u(:))]);

% Plot the vertical velocity profile along the horizontal midline

figure;

plot(x, u\_horizontal\_midline, 'r-', 'LineWidth', 2);

xlabel('x');

ylabel('u');

title('horizental Velocity Profile along Horizontal Midline');

grid on;

ylim([min(u(:)), max(u(:))]);

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*part c

stress\_top(2,:) = mu\*((u(i,Ny)-u(i,Ny-1))/h + (v(ip,Ny)-v(i,Ny))/h)

stress\_bottom(2,:) = mu\*((u(i,2)-u(i,1))/h + (v(ip,1)-v(i,1))/h)

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*part c

% stress\_top(2,:) = mu\*((u(i,Ny)-u(i,Ny-1))/h + (v(ip,Ny)-v(i,Ny))/h)

% stress\_bottom(2,:) = mu\*((u(i,2)-u(i,1))/h + (v(ip,1)-v(i,1))/h)

% %%% PLOT STRESS

% figure;

% subplot(2,1,1);

% plot(x(2:end-1), stress\_top);

% title('Stress on the Top Wall');

% xlabel('x');

% ylabel('Stress');

%

% subplot(2,1,2);

% plot(x(2:end-1), stress\_bottom);

% title('Stress on the Bottom Wall');

% xlabel('x');

% ylabel('Stress');

کد under relaxation

% VORTICITY/STREAM FUNCTION LID DRIVEN CAVITY FLOW SOLVER BY JOE MOLVAR

clear; close all;

% Given Parameters

Nx = 52;

L = 1;

Wall\_Velocity = 29.25; % Nodes X; Domain Size; Wall Velocity

rho = 1;

mu = 0.01; % Density; Dynamic Viscosity;

dt = 0.001;

maxIt = 50000;

maxe = 1e-7; % Time Step; Max iterations; Max error

% 1D Grid Setup

Ny = Nx;

h = L / (Nx - 1);

x = 0:h:L;

y = 0:h:L;

im = 1:Nx-2; i = 2:Nx-1; ip = 3:Nx; jm = 1:Ny-2; j = 2:Ny-1; jp = 3:Ny;

% Initialize Matrices

Vo = zeros(Nx, Ny);

St = zeros(Nx, Ny);

Vop = zeros(Nx, Ny);

u = zeros(Nx, Ny);

v = zeros(Nx, Ny);

u(2:Nx-1, Ny) = Wall\_Velocity;

Vop = Vo;

% Solve Loop (Similar to Gauss-Siedel Method)

for iter = 1:maxIt

% Boundary Conditions

Vo(:, Ny) = -2 \* St(:, Ny-1) / (h^2) - Wall\_Velocity \* 2 / h; % Top

Vo(:, 1) = -2 \* St(:, 2) / (h^2); % Bottom

Vo(1, :) = -2 \* St(2, :) / (h^2); % Left

Vo(Nx, :) = -2 \* St(Nx-1, :) / (h^2); % Right

% Partially Solve Vorticity Transport Equation

s\_x1 = max(u(i, j), 0);

s\_x2 = max(-u(i, j), 0);

s\_y1 = max(v(i, j), 0);

s\_y2 = max(-v(i, j), 0);

% SOR Calculation

OMEGA = 1.5;

w=0.8;

Vo(i, j) =Vop(i, j) +w\*( (OMEGA ./ (1 - 4\*dt\*(mu/(rho\*h^2) - dt\*(s\_x1 ./ h) -...

dt\*(s\_x2 ./ h) - dt\*(s\_y1 ./ h) - dt\*(s\_y2 ./ h))) .\* ((mu/(rho\*h^2)) \* (Vop(ip, j) +...

Vop(im, j) + Vop(i, jp) + Vop(i, jm) - 4 \* Vop(i, j)) - (s\_x1 ./ h) .\* ...

(Vop(i, j) - Vop(im, j)) + (s\_x2 ./ h ).\* (Vop(ip, j) - Vop(i, j)) - ...

(s\_y1 ./ h) .\* (Vop(i, j) - Vop(i, jm)) + (s\_y2 ./ h) .\* (Vop(i, jp) - Vop(i, j))) \* dt));

% Partially Solve Elliptical Vorticity Equation for Stream Function

Stp = St;

St(i,j) = Stp(i,j) + w\*((Vo(i,j)\*h^2 + St(ip,j) + St(i,jp) + St(i,jm) + St(im,j))/4 - Stp(i,j));

u(i, j) = (St(i, jp) - St(i, jm)) / (2 \* h);

v(i, j) = (-St(ip, j) + St(im, j)) / (2 \* h);

% Check for Convergence

if iter > 10

error = max(max(abs(Vo - Vop)));

if error < maxe

break;

end

end

Vop = Vo;

end

% Plots

cm = hsv(ceil(100 / 0.7));

cm = flipud(cm(1:100, :));

figure(1);

contourf(x, y, u', 23, 'LineColor', 'none');

title('U-velocity');

xlabel('x-location');

ylabel('y-location');

% contourf(x, y, v', 23, 'LineColor', 'none');

% title('V-velocity');

% xlabel('x-location');

% ylabel('y-location');

axis equal;

axis([0 L 0 L]);

colormap(cm);

colorbar('westoutside');

figure(2);

plot(y, u(round(Ny/2), :));

title('Centerline x-direction velocity');

xlabel('y/L');

ylabel('u/U');

axis square;

xlim([0 L]);

grid on;

N = 1000;

xstart = L \* rand(N, 1);

ystart = L \* rand(N, 1);

[X, Y] = meshgrid(x, y);

figure(3);

h = streamline(X, Y, u', v', xstart, ystart, [0.1, 200]);

title('Stream Function');

xlabel('x-location');

ylabel('y-location');

axis equal;

set(h, 'color', 'k');

% a and b

%Find indices for the middle vertical and horizontal lines

%Define the midpoints for vertical and horizontal midlines

midpoint\_x = round(Nx / 2);

midpoint\_y = round(Ny / 2);

% Extract the vertical velocity profile (v) along the midlines

v\_vertical\_midline = v(midpoint\_x, :);

v\_horizontal\_midline = v(:, midpoint\_y);

% Plot the vertical velocity profile along the vertical midline

figure;

plot(y, v\_vertical\_midline, 'b-', 'LineWidth', 2);

xlabel('y');

ylabel('v');

title('Vertical Velocity Profile along Vertical Midline');

grid on;

ylim([min(v(:)), max(v(:))]);

% Plot the vertical velocity profile along the horizontal midline

figure;

plot(x, v\_horizontal\_midline, 'r-', 'LineWidth', 2);

xlabel('x');

ylabel('v');

title('Vertical Velocity Profile along Horizontal Midline');

grid on;

ylim([min(v(:)), max(v(:))]);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%5

% Extract the vertical velocity profile (v) along the midlines

u\_vertical\_midline = u(midpoint\_x, :);

u\_horizontal\_midline = u(:, midpoint\_y);

% Plot the vertical velocity profile along the vertical midline

figure;

plot(y, u\_vertical\_midline, 'b-', 'LineWidth', 2);

xlabel('y');

ylabel('u');

title('horizental Velocity Profile along Vertical Midline');

grid on;

ylim([min(u(:)), max(u(:))]);

% Plot the vertical velocity profile along the horizontal midline

figure;

plot(x, u\_horizontal\_midline, 'r-', 'LineWidth', 2);

xlabel('x');

ylabel('u');

title('horizental Velocity Profile along Horizontal Midline');

grid on;

ylim([min(u(:)), max(u(:))]);

کد بخش امتیازی

clear; close all

%%\*\*\*\*\*\*\*\*\*\*\*\*\*\*steady state equation\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

%%% GIVENS

Nx = 52; L = 1; Wall\_Velocity = 10; % Nodes X; Domain Size; Velocity

rho = 1; mu = 0.01; k=0.3; % Density; Dynamic Viscosity;

maxIt = 50000; maxe = 1e-7; % Time Step; Max iter; Max error

%%% SETUP 1D GRID

Ny = Nx; h = L / (Nx - 1); x = 0:h:L; y = 0:h:L;

im = 1:Nx-2; i = 2:Nx-1; ip = 3:Nx; jm = 1:Ny-2; j = 2:Ny-1; jp = 3:Ny;

%%% PRELOCATE MATRICES

Vo = zeros(Nx, Ny); St = Vo; Vop = Vo; u = Vo; v = Vo; T = 300 \* ones(Nx, Ny); % Initial temperature

%%% VELOCITY ON THE UPPER WALL (NO-SLIP CONDITION)

u(2:Nx-1, Ny) = Wall\_Velocity;

%%% SOLVE LOOP SIMILAR TO GAUSS-SEIDEL METHOD

for iter = 1:maxIt

%%% CREATE VELOCITY FROM STREAM FUNCTION

u(i, j) = (St(i, jp) - St(i, jm)) / (2\*h);

v(i, j) = (-St(ip, j) + St(im, j)) / (2\*h);

T(1, 1:Ny) = T(2, 1:Ny);

T(Nx, 1:Ny) = T(Nx-1, 1:Ny);

T(1:Nx, 1) = 300;

T(1:Nx, Ny) = 330;

%%% TEMPERATURE EQUATION (ENERGY EQUATION)

% Temporal discretization

Tn = T;

T(i,j) =(k \* (Tn(ip, j) + Tn(im, j) + Tn(i, jp) + Tn(i, jm)) + ...

max(u(i, j), 0) .\* h .\* Tn(im, j) + max(-u(i, j), 0) .\* h .\* Tn(ip, j) + ...

max(v(i, j), 0) .\* h .\* Tn(i, jm) + max(-v(i, j), 0) .\* h .\* Tn(i, jp)) ./ ...

(max(u(i, j), 0) .\* h + max(-u(i, j), 0) .\* h + max(v(i, j), 0) .\* h + max(-v(i, j), 0) .\* h + 4 \* k);

%%% SAVING THE VALUE OF VORTICITY AT THE LAST STEP

Vop = Vo;

%%% CREATE BOUNDARY CONDITIONS

Vo(1:Nx, Ny) = -2 \* St(1:Nx, Ny-1) / (h^2) - Wall\_Velocity \* 2 / h; % Top

Vo(1:Nx, 1) = -2 \* St(1:Nx, 2) / (h^2); % Bottom

Vo(1, 1:Ny) = -2 \* St(2, 1:Ny) / (h^2); % Left

Vo(Nx, 1:Ny) = -2 \* St(Nx-1, 1:Ny) / (h^2); % Right

%%% PARTIALLY SOLVE VORTICITY TRANSPORT EQUATION

Vo(i, j) = (mu / rho \* (Vop(ip, j) + Vop(im, j) + Vop(i, jp) + Vop(i, jm)) + ...

max(u(i, j), 0) .\* h .\* Vop(im, j) + max(-u(i, j), 0) .\* h .\* Vop(ip, j) + ...

max(v(i, j), 0) .\* h .\* Vop(i, jm) + max(-v(i, j), 0) .\* h .\* Vop(i, jp)) ./ ...

(max(u(i, j), 0) .\* h + max(-u(i, j), 0) .\* h + max(v(i, j), 0) .\* h + max(-v(i, j), 0) .\* h + 4 \* mu / rho);

%%% PARTIALLY SOLVE ELLIPTIC VORTICITY EQUATION FOR STREAM FUNCTION

St(i, j) = (Vo(i, j) \* h^2 + St(ip, j) + St(i, jp) + St(i, jm) + St(im, j)) / 4;

%%% CHECK FOR CONVERGENCE

if iter > 10

error = max(max(Vo - Vop));

if error < maxe

break;

end

end

end

% Plot temperature contour

contourf(x, y, T', 20, 'LineWidth', 1.5);

colorbar;

colormap('jet');

xlabel('X');

ylabel('Y');

title('Temperature Contour');

# **منابع و مراجع**

[1] High-Re Solutions for Incompressible Flow Using the Navier-Stoles Equations and a Multigrid Method – U. Ghia, K.N. Ghia and C.T. Shin

[2]A fast and short Matlab code to solve the lid driven cavity flow problem using the vorticity-stream function formulation – Joe Molvar