

پروژه درس مقدمه ای بر دینامیک سیالات محاسباتی (CFD)

استاد : دکتر زینب پورانصاری

گروه ۵ :

امیرمحمد اسدپور ۹۷۵۴۱۰۵۴

دارا رحمت سمیعی ۹۷۴۶۱۲۰۷

حسین کتابچی ۹۷۵۴۲۴۲۷

موضوع پروژه :

تحلیل جریان دو بعدی در حفره ای مستطیلی با استفاده از حل معادلات بیضوی

اهداف پروژه :

- محاسبه توزیع مولفه های افقی عمودی سرعت در حفره
- رسم خطوط جریان و گردابه در داخل حفره با استفاده از نرم افزار متلب

خلاصه روش حل عددی :

- انجام گسته سازی معادلات حاکم یعنی توابع جریان و تاوایی و سپس انجام شبیه سازی عددی به وسیله نرم افزار متلب

خواسته های بخش اول

سرفصل‌ها: حل معادلات بیضوی در دو بعد

با استفاده از معادلات تابع جریان-تاوابی می‌توان جریان‌های دو بعدی پایا مانند جریان سیال در درون حفره نشان داده شده در شکل زیر را محاسبه نمود. می‌دانیم معادلات حاکم به صورت زیر است

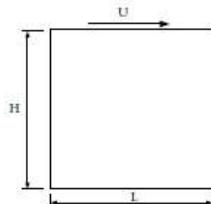
$$\omega = - \left(\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} \right), \quad u \frac{\partial \omega}{\partial x} + v \frac{\partial \omega}{\partial y} = \nu \left(\frac{\partial^2 \omega}{\partial x^2} + \frac{\partial^2 \omega}{\partial y^2} \right). \quad (1)$$

تابع جریان در مرزها به صورت زیر است

$$\psi(0, y) = \psi(L, y) = 0, \quad (\forall y), \quad \psi(x, 0) = \psi(x, H), \quad (\forall x).$$

تاوابی روی مرزها از روابط زیر محاسبه می‌باشد.

$$\begin{aligned} x = 0, \quad \omega &= -\frac{\partial^2 \psi}{\partial x^2}, \quad \frac{\partial \psi}{\partial x} = 0, \quad x = L, \quad \omega = -\frac{\partial^2 \psi}{\partial x^2}, \quad \frac{\partial \psi}{\partial x} = 0 \\ y = 0, \quad \omega &= -\frac{\partial^2 \psi}{\partial y^2}, \quad \frac{\partial \psi}{\partial y} = 0, \quad y = H, \quad \omega = -\frac{\partial^2 \psi}{\partial y^2}, \quad \frac{\partial \psi}{\partial y} = U \end{aligned} \quad (2)$$



برای حالتی که $H = L$ و عدد $Re = uL/\nu = 100$ باشد، معادلات حاکم و شرایط مرزی را گسترش‌سازی کرده و با انجام شبیه‌سازی عددی موارد زیر را محاسبه و گزارش کنید.

۱. مولقه افقی سرعت در $x = L/2$ بر حسب y و مقایسه با حل مرجع.

۲. مولقه عمودی سرعت در $y = H/2$ بر حسب x و مقایسه با حل مرجع.

۳. رسم خطوط جریان و گرداب.

۴. گزارش به صورت تایپ شده تحویل شود.

روش حل

مرحله اول : گستته سازی ω و ψ

$$\beta^2 = \left(\frac{\Delta x}{\Delta y} \right)^2$$

Discretized
Stream
Function

$$\begin{aligned}\omega &= - \left(\frac{\partial^2 \Psi}{\partial x^2} + \frac{\partial^2 \Psi}{\partial y^2} \right) \Rightarrow w_{i,j} = \left(\frac{\Psi_{i+1,j} - 2\Psi_{i,j} + \Psi_{i-1,j}}{\Delta x^2} + \frac{\Psi_{i,j+1} - 2\Psi_{i,j} + \Psi_{i,j-1}}{\Delta y^2} \right) \\ &\Rightarrow \Delta x^2 \omega_{i,j} = \Psi_{i+1,j} - 2\Psi_{i,j} + \Psi_{i-1,j} + \beta^2 \Psi_{i,j+1} - 2\beta^2 \Psi_{i,j} + \beta^2 \Psi_{i,j-1} \\ &\Rightarrow \Psi_{i,j} = \frac{1}{2 + 2\beta^2} \times (\Psi_{i+1,j} + \Psi_{i-1,j} + \beta^2 \Psi_{i,j+1} + \beta^2 \Psi_{i,j-1} + \Delta x^2 \omega_{i,j})\end{aligned}$$

مرحله دوم :

$$u \frac{\partial \omega}{\partial x} + v \frac{\partial \omega}{\partial y} = v \left(\frac{\partial^2 \omega}{\partial x^2} + \frac{\partial^2 \omega}{\partial y^2} \right)$$

$$\Rightarrow \overbrace{v \left(u \frac{\partial \omega}{\partial x} + v \frac{\partial \omega}{\partial y} \right)}^A = \frac{\partial^2 \omega}{\partial x^2} + \frac{\partial^2 \omega}{\partial y^2}$$

$$\Rightarrow \overbrace{v \left(u \frac{\partial \omega}{\partial x} + v \frac{\partial \omega}{\partial y} \right)}^A = \frac{\omega_{i+1,j} - 2\omega_{i,j} + \omega_{i-1,j}}{\Delta x^2} + \frac{\omega_{i,j+1} - 2\omega_{i,j} + \omega_{i-1,j}}{\Delta y^2}$$

$$\Rightarrow \omega_{i,j} = \frac{1}{2 + 2\beta^2} \times (\omega_{i+1,j} + \omega_{i-1,j} + \beta^2 \omega_{i,j+1} + \beta^2 \omega_{i,j-1} - \Delta x^2 A)$$

$$\left\{ \begin{array}{l} A = \frac{1}{v} \left(u \frac{\partial \omega}{\partial x} + v \frac{\partial \omega}{\partial y} \right) = \frac{1}{v} \left(\textcolor{red}{u_{i,j}} \frac{\omega_{i+1,j} - \omega_{i-1,j}}{2\Delta x} + \textcolor{red}{v_{i,j}} \frac{\omega_{i,j+1} - \omega_{i,j-1}}{2\Delta y} \right) \\ u = \frac{\partial \psi}{\partial y} \Rightarrow \textcolor{red}{u_{i,j}} = \frac{\psi_{i,j+1} - \psi_{i,j-1}}{2\Delta y} \\ v = -\frac{\partial \psi}{\partial x} \Rightarrow \textcolor{red}{v_{i,j}} = -\frac{\psi_{i+1,j} - \psi_{i-1,j}}{2\Delta x} \end{array} \right.$$

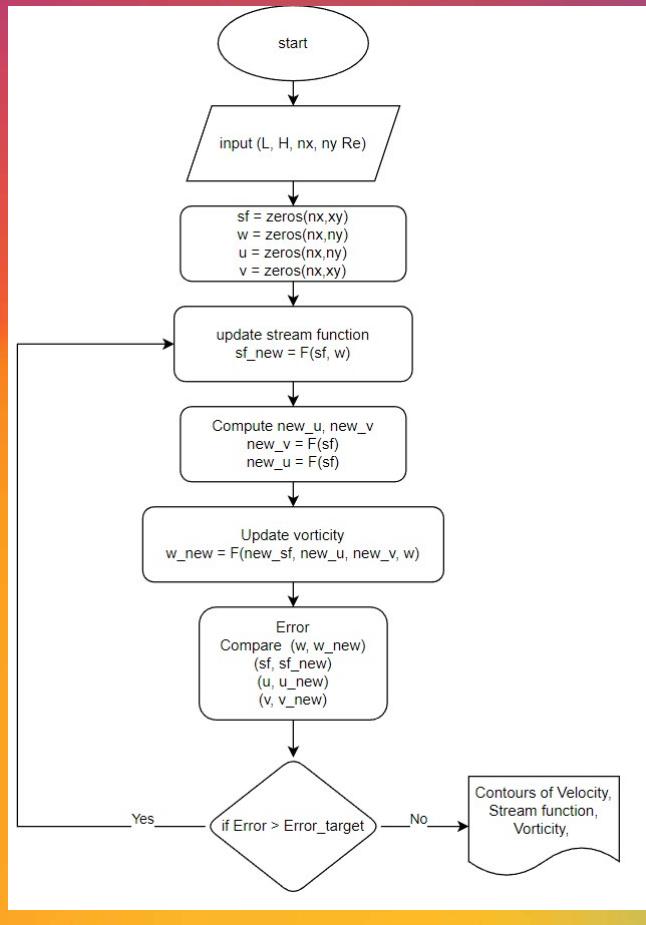
Discretized
u, v, Vorticity

شرایط مرزی

Discretized
Boundary
Condition Of
Vorticity

$$\begin{aligned}\omega_{1,j} &= -2 \frac{\psi_{2,j} - \Delta x \cdot v_{1,j}}{\Delta x^2} \xrightarrow{v_{1,j}=0} \omega_{1,j} = -2 \frac{\psi_{2,j}}{\Delta x^2} \\ \omega_{i_{max},j} &= -2 \frac{\psi_{i_{max}-1,j} + \Delta x \cdot v_{i_{max},j}}{\Delta x^2} \xrightarrow{v_{i_{max},j}=0} \omega_{i_{max},j} = -2 \frac{\psi_{i_{max}-1,j}}{\Delta x^2} \\ \omega_{i,1} &= -2 \frac{\psi_{i,2} - \Delta y \cdot u_{i,1}}{\Delta y^2} \xrightarrow{u_{i,1}=-U} \omega_{i,1} = -2 \frac{\psi_{i,2} + \Delta y \cdot U}{\Delta y^2} \\ \omega_{i,j_{max}} &= -2 \frac{\psi_{i,j_{max}-1} + \Delta y \cdot u_{i,j_{max}}}{\Delta y^2} \xrightarrow{u_{i,j_{max}}=U} \omega_{i,j_{max}} = -2 \frac{\psi_{i,j_{max}-1} + \Delta y \cdot U}{\Delta y^2}\end{aligned}$$

فلوچارت مراحل حل



شروع کدنویسی

بخش اول(عمومی برای همه گروه ها) : فقط مرز بالا به سمت راست حرکت میکند

مرحله اول : تعریف پارامتر های اساسی و ماتریس تابع جریان و گردابه

```
%>>> %% Defining Parameters
U = 1;
L = 1;
H = 1;
Re = 100;
nu = U * L / Re;
nx = 251;
ny = 251;
delta_x = L / (nx - 1);
delta_y = H / (ny - 1);
beta = delta_x / delta_y;
beta2 = beta.^2;

%>>> %% Define Matrices of Stream Function & Vorticity
sf = zeros(nx,ny);
sf0= zeros(nx,ny);
w = zeros(nx, ny);
w0 = zeros(nx, ny);
u = zeros(nx, ny); u(:,end) = U;
v = zeros(nx, ny);
```

ادامه کدنویسی : یافتن تابع جریان ، سرعت ، گرداب و خطای

مرحله دوم : یافتن تابع جریان و به روز رسانی شرایط مرزی برای گرداب

```
%% Finding Stream Function & Velocity & Vorticity & Error
Err_tot = 100;
step = 0;

while ( Err_tot > 1e-3)

    %Finding Stream Function
    for i=2:nx-1
        for j=2:ny-1
            sf(i,j) = (sf(i+1,j) + sf(i-1,j) +...
                        beta2* sf(i,j+1) + beta2* sf(i,j-1) + ...
                        (delta_x.^2) * w(i,j)) / (2 + 2* beta2);
        end
    end

    %Update Boundary Condition for Vorticity
    w(1,:)= -2 * sf(2,:)/delta_x^2;
    w(end,:)= -2 * sf(end-1,:)/delta_x^2;
    w(:,1)= -2 * sf(:,2)/delta_y^2;
    w(:,end)= -2 * (sf(:,end-1) + U*delta_y)/delta_y^2;
```

ادامه کدنویسی : یافتن تابع جریان ، سرعت ، گرداب و خطای

مرحله سوم : یافتن سرعت و گرداب و خطای

```
%Finding Velocity & Vorticity
for i=2:nx-1
    for j=2:ny-1
        u(i,j) = (sf(i,j+1) - sf(i,j-1)) / (2 * delta_y);
        v(i,j) = -(sf(i+1,j) - sf(i-1,j)) / (2 * delta_x);

        dwdx = (w(i+1,j) - w(i-1,j)) / (2 * delta_x);
        dwdy = (w(i,j+1) - w(i,j-1)) / (2 * delta_y);

        A = ((u(i,j) * dwdx) + (v(i,j) * dwdy)) / nu;

        w(i,j) = (w(i+1,j) + w(i-1,j) + beta2* w(i,j+1) + beta2* w(i,j-1) - (delta_x.^2) * A) / (2 + 2* beta2);
    end
end
V_tot = sqrt(u.^2 + v.^2);

%Finding Error
Err_sf = max(max(abs(sf0-sf)));
Err_w = max(max(abs(w0-w)));
Err_tot = max([Err_sf , Err_w]);
disp(Err_tot);

%Update sf0 & w0
w0 = w;
sf0 = sf;
step = step + 1;
```

مرحله پايانى کدنويسى: رسم کانتورها و بردار سرعت و استخراج فایل GIF

```
%Extract GIF
if (mod(step,100)==0)
    ugif=figure(1);
    contourf(transpose(u),50);
    title('Contour of velocity u');
    xlabel('x(m)');
    ylabel('y(m)');
    colorbar;
    colormap jet
    drawnow
    frame1 = getframe(ugif);
    im1 = frame2im(frame1);
    [imind1,cm1] = rgb2ind(im1,256);
    if step == 100
        imwrite(imind1,cm1,'Cavity_General_u.gif','gif', 'Loopcount',inf);
    else
        imwrite(imind1,cm1,'Cavity_General_u.gif','gif','WriteMode','append');
    end

    vgif=figure(2);
    contourf(transpose(v),50);
    title('Contour of velocity v');
    xlabel('x(m)');
    ylabel('y(m)');
    colorbar;
    colormap jet
    drawnow
    frame2 = getframe(vgif);
    im2 = frame2im(frame2);
    [imind2,cm2] = rgb2ind(im2,256);
    if step == 100
        imwrite(imind2,cm2,'Cavity_General_v.gif','gif', 'Loopcount',inf);
    else
        imwrite(imind2,cm2,'Cavity_General_v.gif','gif','WriteMode','append');
    end

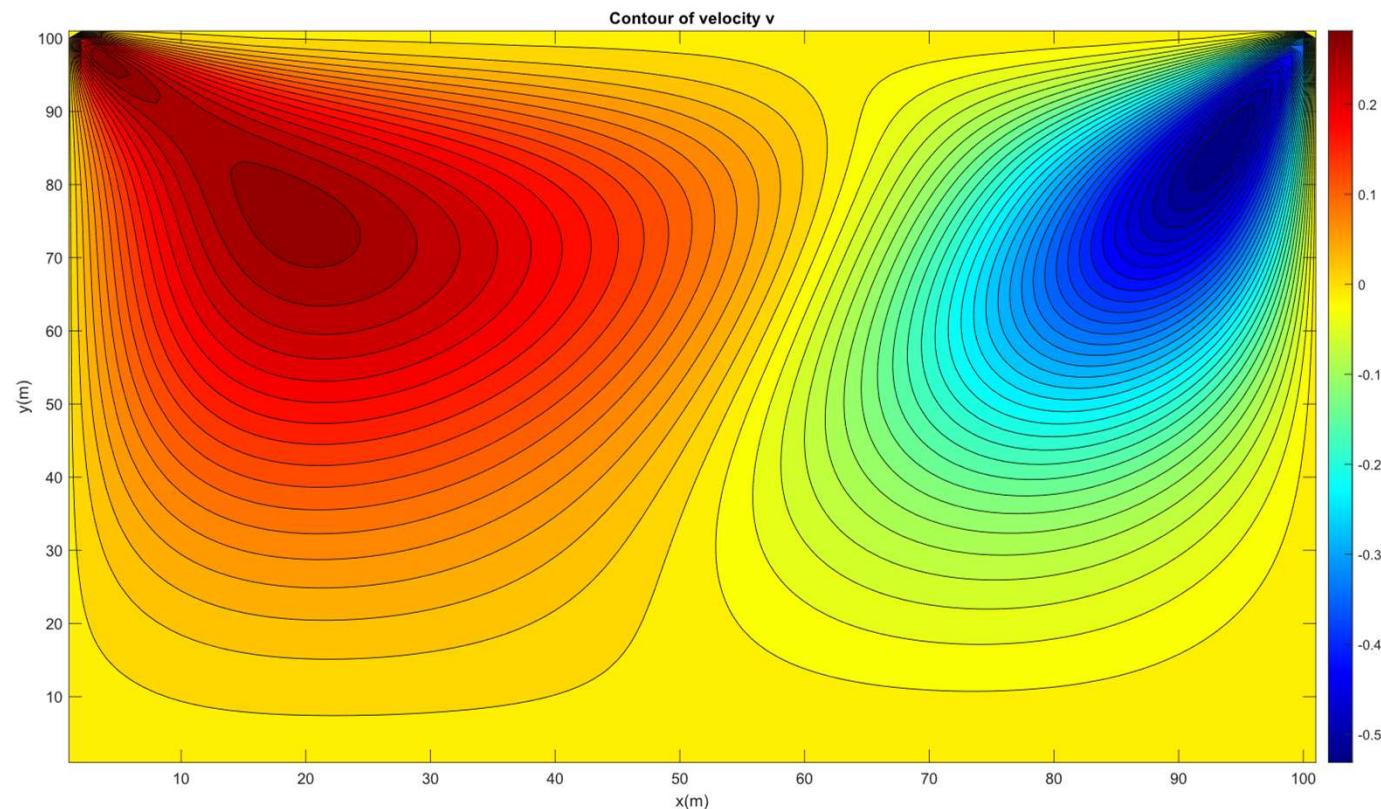
    vtgif=figure(3);
    contourf(transpose(V_tot));
    title('Contour of total velocity');
    xlabel('x(m)');
    ylabel('y(m)');
    colorbar
    drawnow
    frame3 = getframe(vtgif);
    im3 = frame2im(frame3);
    [imind3,cm3] = rgb2ind(im3,256);
    if step == 100
        imwrite(imind3,cm3,'Cavity_General_V-total.gif','gif', 'Loopcount',inf);
    else
        imwrite(imind3,cm3,'Cavity_General_V-total.gif','gif','WriteMode','append');
    end

    sfgif=figure(4);
    contourf(transpose(sf),50);
    title('Contour of Stream Lines');
    xlabel('x(m)');
    ylabel('y(m)');
    colorbar;
    colormap jet
    drawnow
    frame4 = getframe(sfgif);
    im4 = frame2im(frame4);
    [imind4,cm4] = rgb2ind(im4,256);
    if step == 100
        imwrite(imind4,cm4,'Cavity_General_Stream Lines.gif','gif', 'Loopcount',inf);
    else
        imwrite(imind4,cm4,'Cavity_General_Stream Lines.gif','gif','WriteMode','append');
    end

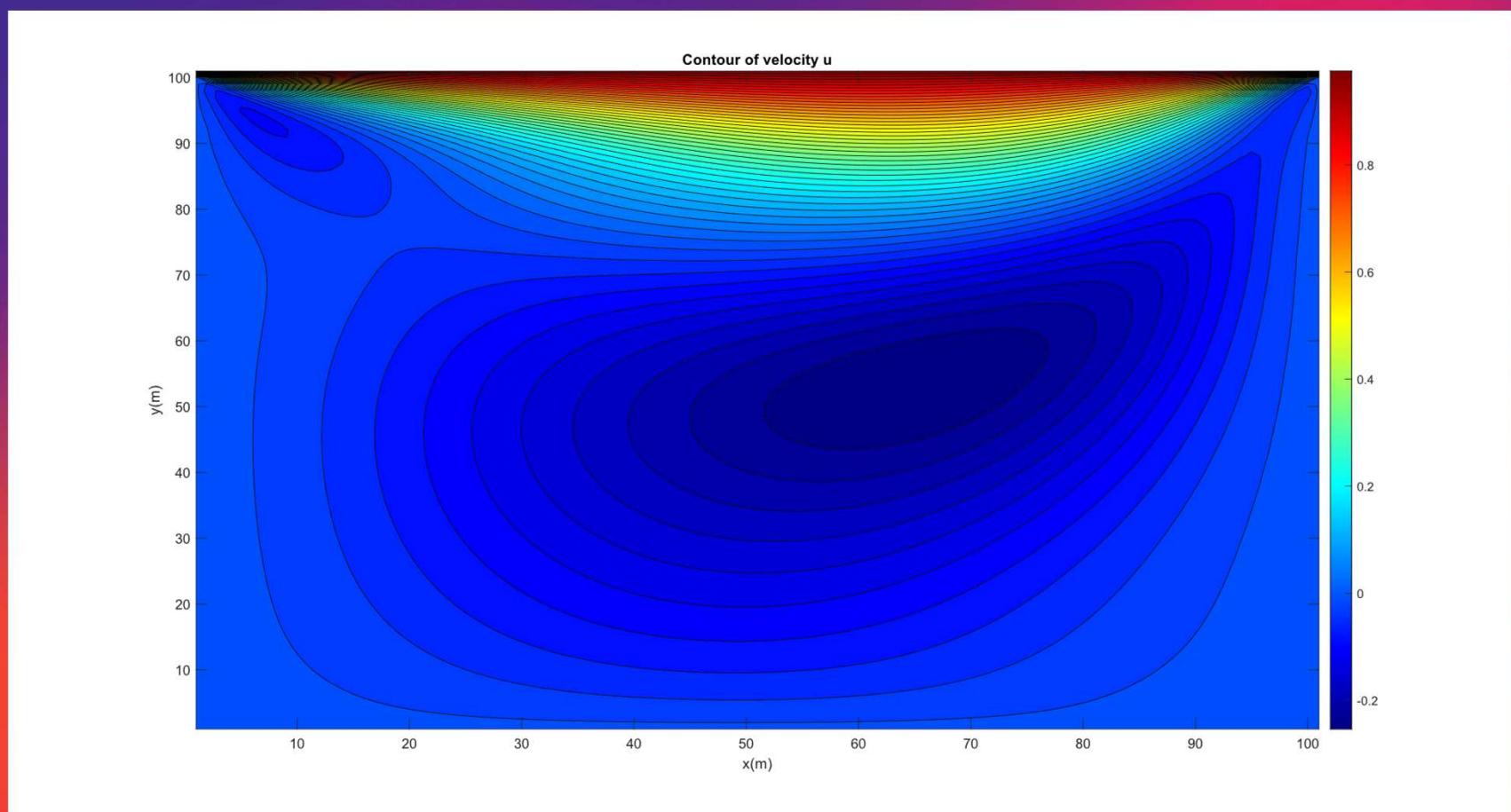
    wgif=figure(5);
    contourf(transpose(w),1000);
    title('Contour of Vorticity');
    xlabel('x(m)');
    ylabel('y(m)');
    colorbar;
    colormap jet
    drawnow
    frames = getframe(wgif);
    im5 = frame2im(frames);
    [iminds,cm5] = rgb2ind(im5,256);
    if step == 100
        imwrite(iminds,cm5,'Cavity_General_Vorticity.gif','gif', 'Loopcount',inf);
    else
        imwrite(iminds,cm5,'Cavity_General_Vorticity.gif','gif','WriteMode','append');
    end
end

figure(6);
hold on
contour(transpose(sf),10);
q = quiver(transpose(u),transpose(v),3);
q.Color = 'r';
xlabel('x(m)');
ylabel('y(m)');
title('Stream Line & Vector of velocity');
legend('Stream Line', 'Vector of velocity')
hold off
```

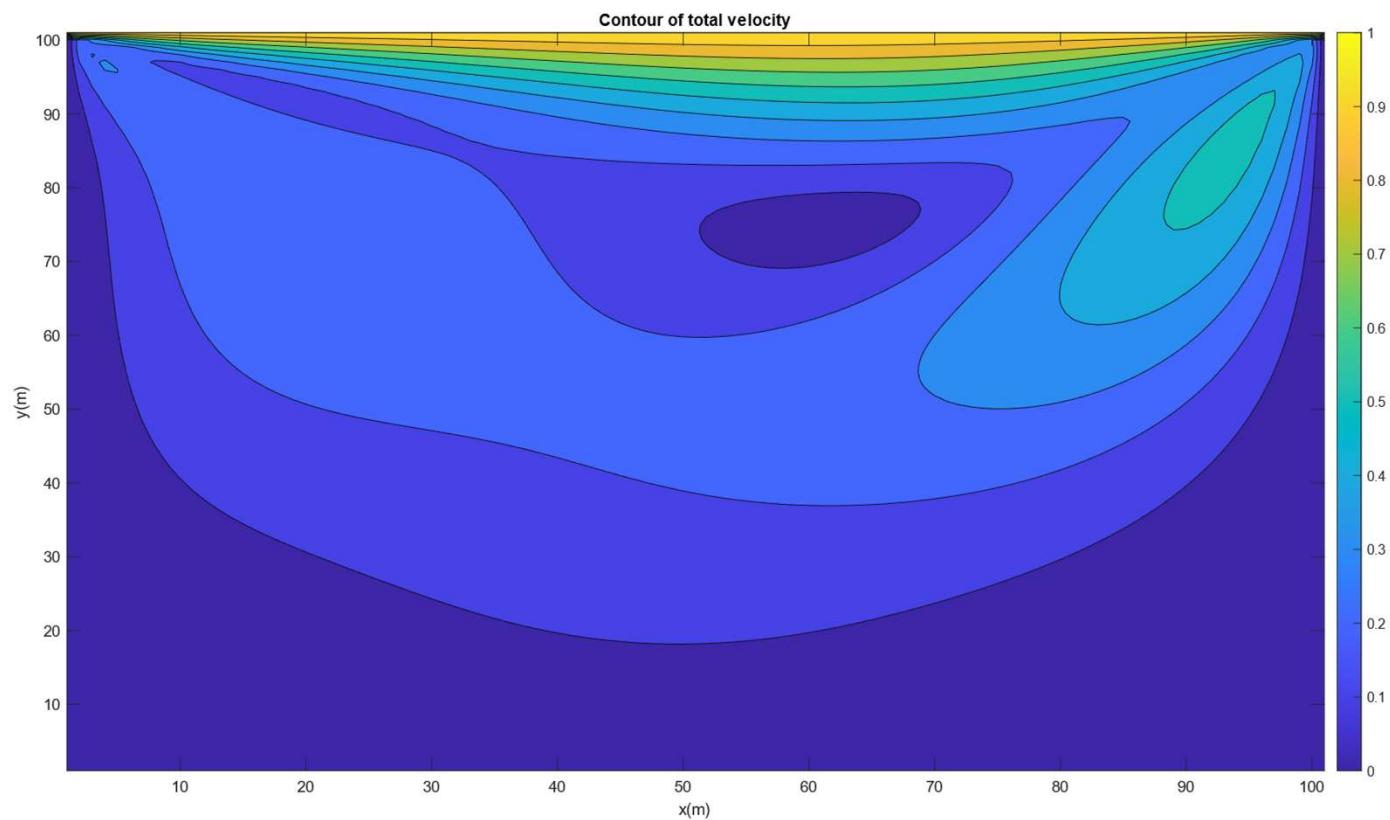
کانتور سرعت v



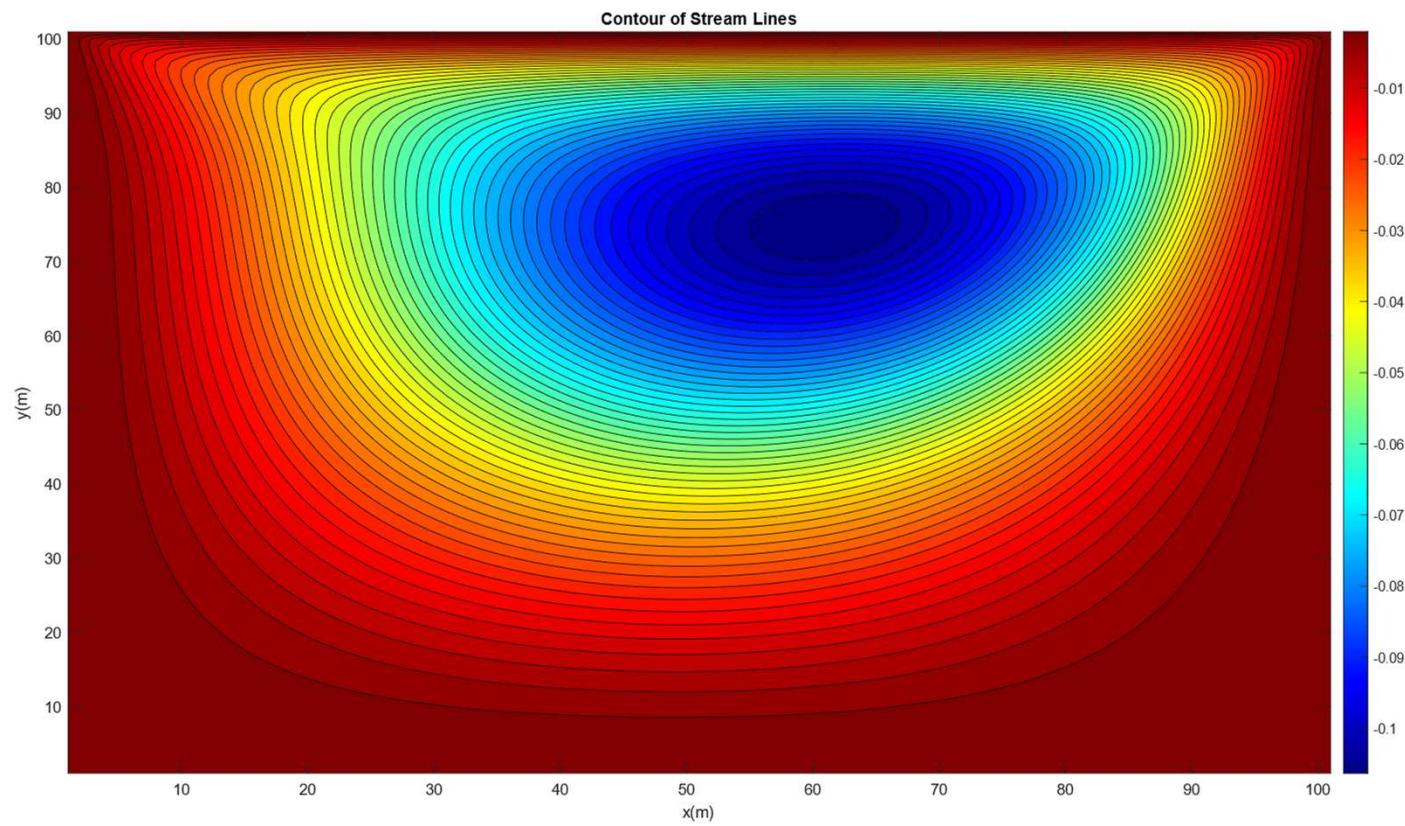
کانتور سرعت u



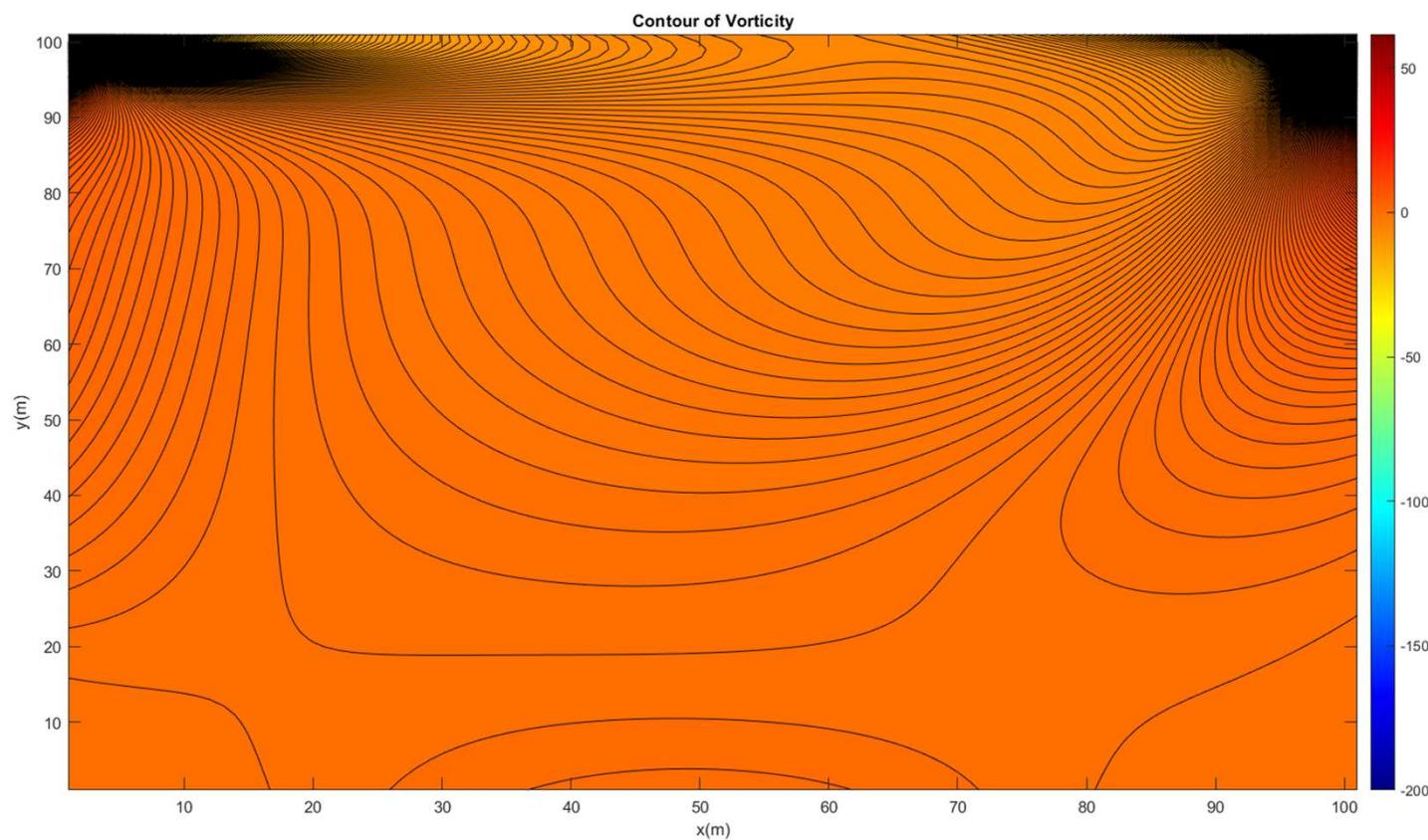
کانتور سرعت کل



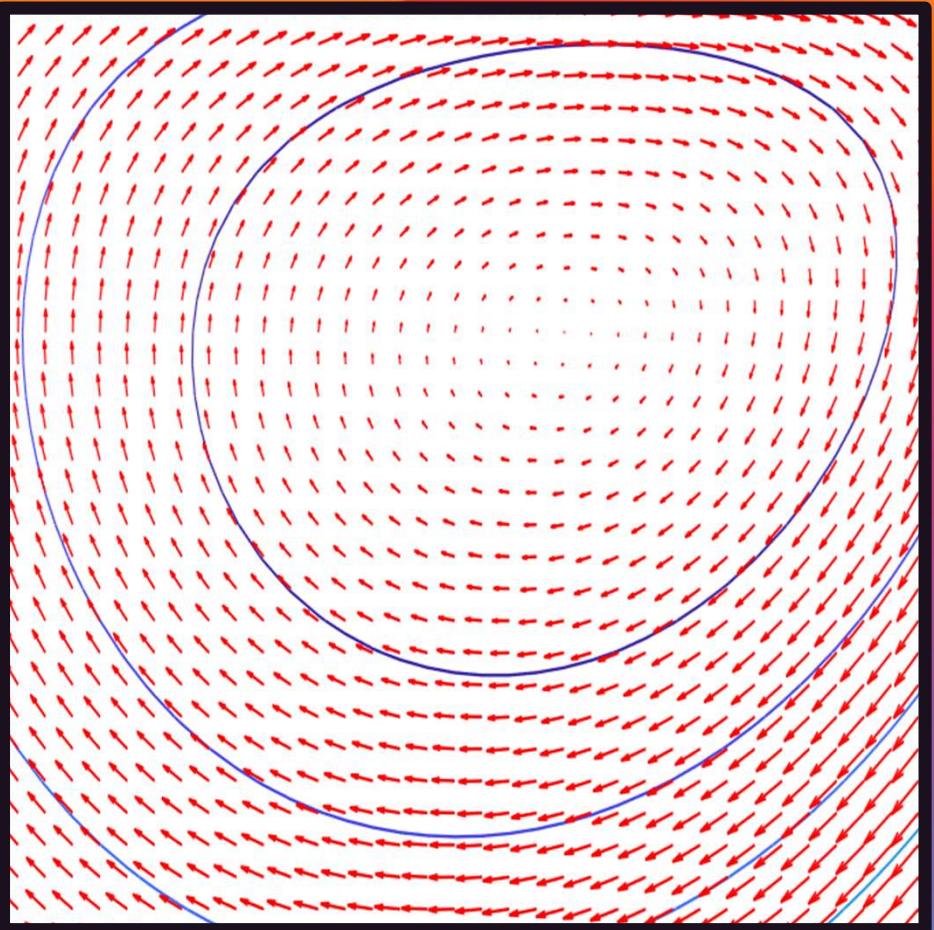
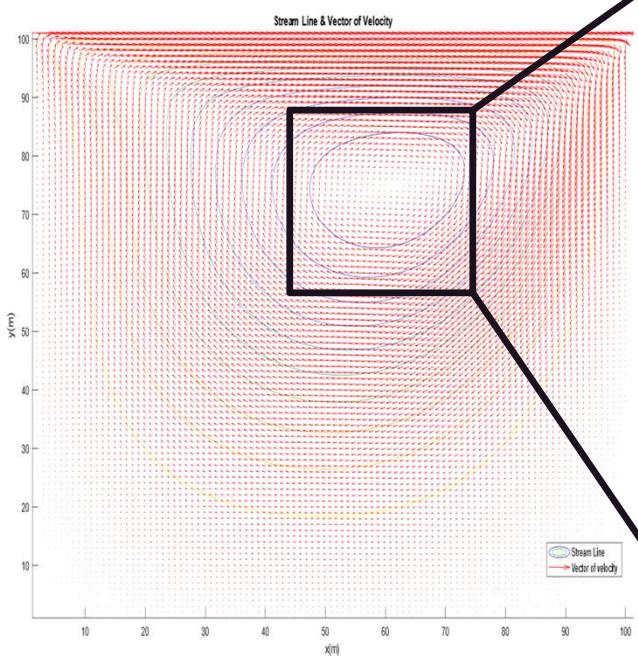
کانتور خطوط جریان



کانتور گردابه



بردار سرعت و خطوط جریان



بخش دوم (موضوع شماره ۸) : تحلیل جریان دو بعدی در حفره ای مستطیلی با فرض حرکت مرز بالا به سمت راست و حرکت مرز پایین به سمت چپ

در این مرحله نیز مراحل حل مانند قسمت قبل بوده و پس از تعریف ورودی ها و معادلات اساسی به محاسبه آن میپردازیم و صحت داده های بدست آمده را توسط مقایسه با مقدار خطأ می سنجیم

شروع کدنویسی

مرحله اول : تعریف پارامتر های اساسی و ماتریس قابع جریان و گردابه

```
%% Defining Parameters
U = 1;
L = 1;
H = 1;
Re = 100;
nu = U * L / Re;
nx = 101;
ny = 101;
delta_x = L / (nx - 1);
delta_y = H / (ny - 1);
beta = delta_x / delta_y;
beta2 = beta.^2;

%% Define Matrices of Stream Function & Vorticity
sf = zeros(nx,ny);
sf0= zeros(nx,ny);
w = zeros(nx, ny);
w0 = zeros(nx, ny);
u = zeros(nx, ny); u(:,1) = -U; u(:,end) = U; %Top moves right & Bottom moves left
v = zeros(nx, ny);
```

ادامه کدنویسی : یافتن تابع جریان ، سرعت ، گرداب و خطای

مرحله دوم : یافتن تابع جریان و بروزرسانی شرایط مرزی برای گرداب

```
%% Finding Stream Function & Velocity & Vorticity & Error
Err_tot = 100;
step = 0;

while ( Err_tot > 1e-3)

    %Finding Stream Function
    for i=2:nx-1
        for j=2:ny-1
            sf(i,j) = (sf(i+1,j) + sf(i-1,j) +...
                        beta2* sf(i,j+1) + beta2* sf(i,j-1) + ...
                        (delta_x.^2) * w(i,j)) / (2 + 2* beta2);
        end
    end

    %Update Boundary Condition for Vorticity
    w(1,:)= -2 * sf(2,:)/delta_x^2;
    w(end,:)= -2 * sf(end-1,:)/delta_x^2;
    w(:,1)= -2 * (sf(:,2) + U*delta_y)/delta_y^2;
    w(:,end)= -2 * (sf(:,end-1) + U*delta_y)/delta_y^2;
```

ادامه کدنویسی : یافتن تابع جریان ، سرعت ، گرداب و خطای

مرحله سوم : یافتن سرعت ، گرداب و خطای

```
%Finding Velocity & Vorticity
for i=2:nx-1
    for j=2:ny-1
        u(i,j) = (sf(i,j+1) - sf(i,j-1)) / (2 * delta_y);
        v(i,j) = -(sf(i+1,j) - sf(i-1,j)) / (2 * delta_x);

        dwdx = (w(i+1,j) - w(i-1,j)) / (2 * delta_x);
        dwdy = (w(i,j+1) - w(i,j-1)) / (2 * delta_y);

        A = ((u(i,j) * dwdx) + (v(i,j) * dwdy))/ nu;

        w(i,j) = (w(i+1,j) + w(i-1,j) + beta2* w(i,j+1) + beta2* w(i,j-1) - (delta_x.^2) * A) / (2 + 2* beta2);
    end
end
V_tot = sqrt(u.^2 + v.^2);

%Finding Error
Err_sf = max(max(abs(sf0-sf)));
Err_w = max(max(abs(w0-w)));
Err_tot = max([Err_sf , Err_w]);
disp(Err_tot);

%Update sf0 & w0
w0 = w;
sf0 = sf;
step = step + 1;
```

مرحله پایانی کدنویسی: رسم کانتورها و بردار سرعت و استخراج فایل GIF

```
%Extract GIF
if (mod(step,100)==0)
    ugif=figure(1);
    contourf(transpose(u),50);
    title('Contour of velocity u');
    xlabel('x(m)');
    ylabel('y(m)');
    colorbar;
    colormap jet
    drawnow
    frame1 = getframe(ugif);
    im1 = frame2im(frame1);
    [imind1,cm1] = rgb2ind(im1,256);
    if step == 100
        imwrite(imind1,cm1,'Cavity_Option8_u.gif','gif', 'Loopcount',inf);
    else
        imwrite(imind1,cm1,'Cavity_Options_u.gif','gif','WriteMode','append');
    end

    vgif=figure(2);
    contourf(transpose(v),50);
    title('Contour of velocity v');
    xlabel('x(m)');
    ylabel('y(m)');
    colorbar;
    colormap jet
    drawnow
    frame2 = getframe(vgif);
    im2 = frame2im(frame2);
    [imind2,cm2] = rgb2ind(im2,256);
    if step == 100
        imwrite(imind2,cm2,'Cavity_Option8_v.gif','gif', 'Loopcount',inf);
    else
        imwrite(imind2,cm2,'Cavity_Options_v.gif','gif','WriteMode','append');
    end

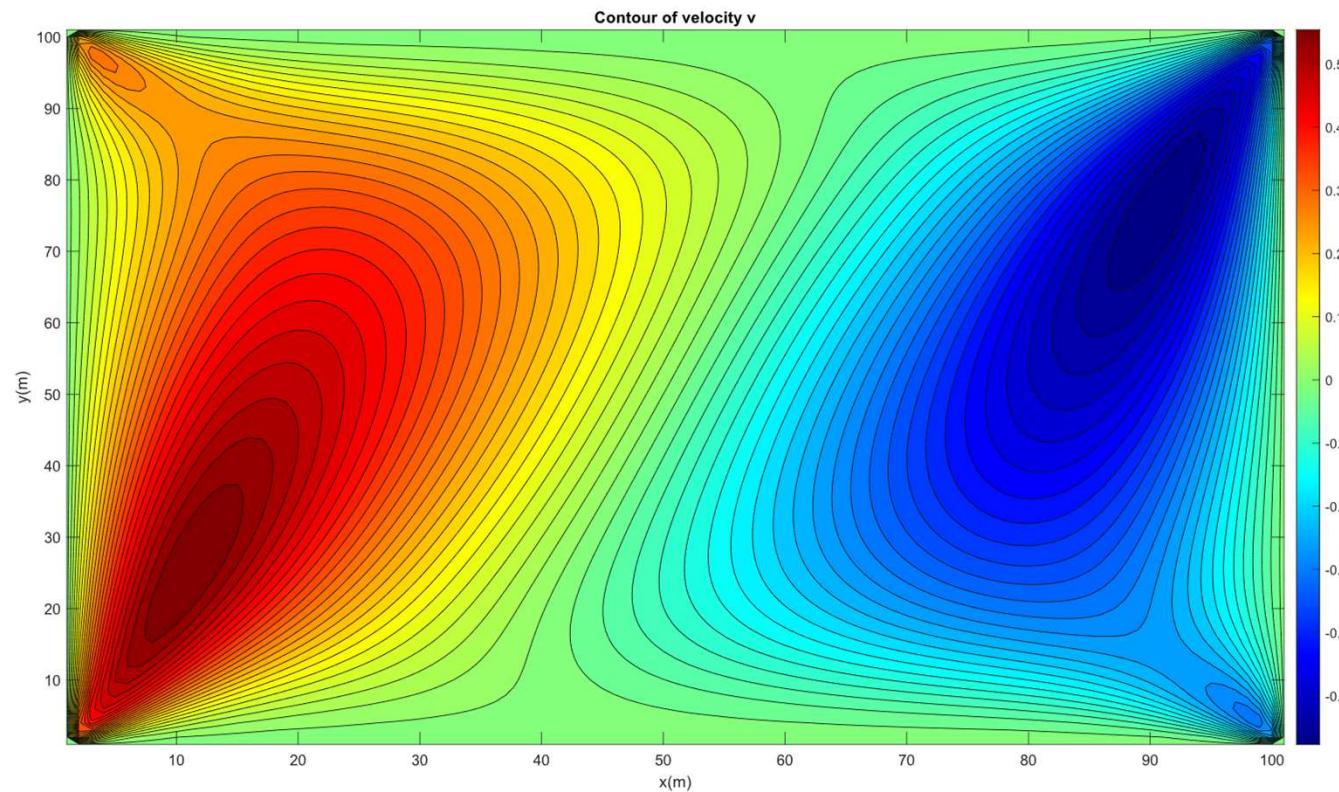
    vtgif=figure(3);
    contourf(transpose(V_tot));
    title('Contour of total velocity');
    xlabel('x(m)');
    ylabel('y(m)');
    colorbar;
    drawnow
    frame3 = getframe(vtgif);
    im3 = frame2im(frame3);
    [imind3,cm3] = rgb2ind(im3,256);
    if step == 100
        imwrite(imind3,cm3,'Cavity_Options_V-total.gif','gif', 'Loopcount',inf);
    else
        imwrite(imind3,cm3,'Cavity_Options_V-total.gif','gif','WriteMode','append');
    end

    sfgif=figure(4);
    contourf(transpose(sf),50);
    title('Contour of Stream Lines');
    xlabel('x(m)');
    ylabel('y(m)');
    colorbar;
    colormap jet
    drawnow
    frame4 = getframe(sfgif);
    im4 = frame2im(frame4);
    [imind4,cm4] = rgb2ind(im4,256);
    if step == 100
        imwrite(imind4,cm4,'Cavity_Option8_Stream Lines.gif','gif', 'Loopcount',inf);
    else
        imwrite(imind4,cm4,'Cavity_Option8_Stream Lines.gif','gif','WriteMode','append');
    end

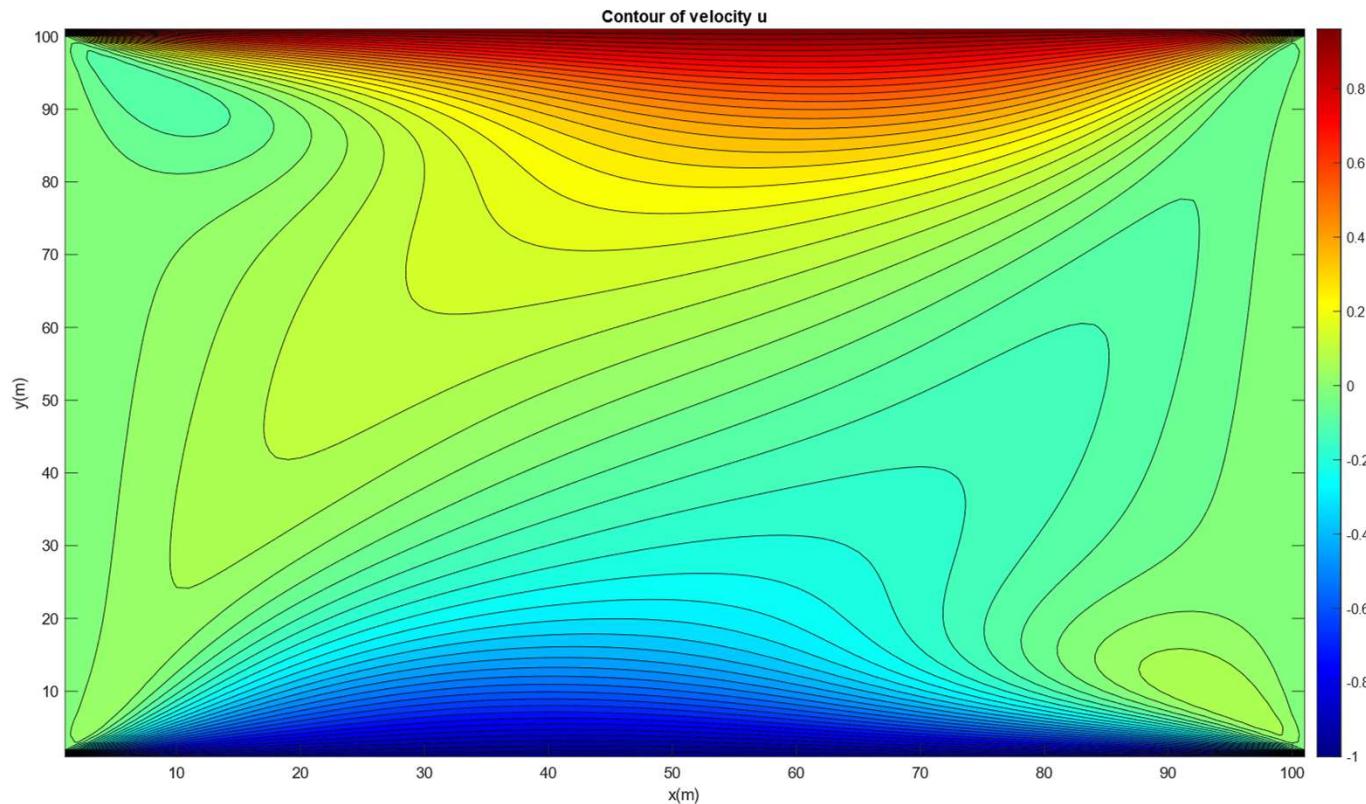
    wgif=figure(5);
    contourf(transpose(w),1000);
    title('Contour of Vorticity');
    xlabel('x(m)');
    ylabel('y(m)');
    colorbar;
    colormap jet
    drawnow
    frame5 = getframe(wgif);
    im5 = frame2im(frame5);
    [imind5,cm5] = rgb2ind(im5,256);
    if step == 100
        imwrite(imind5,cm5,'Cavity_Options_Vorticity.gif','gif', 'Loopcount',inf);
    else
        imwrite(imind5,cm5,'Cavity_Options_Vorticity.gif','gif','WriteMode','append');
    end
end

figure(6);
hold on
contour(transpose(sf),10);
q = quiver(transpose(u),transpose(v),3);
q.Color = 'r';
xlabel('x(m)');
ylabel('y(m)');
title('Stream Line & Vector of Velocity');
legend('Stream Line', 'Vector of velocity')
hold off
```

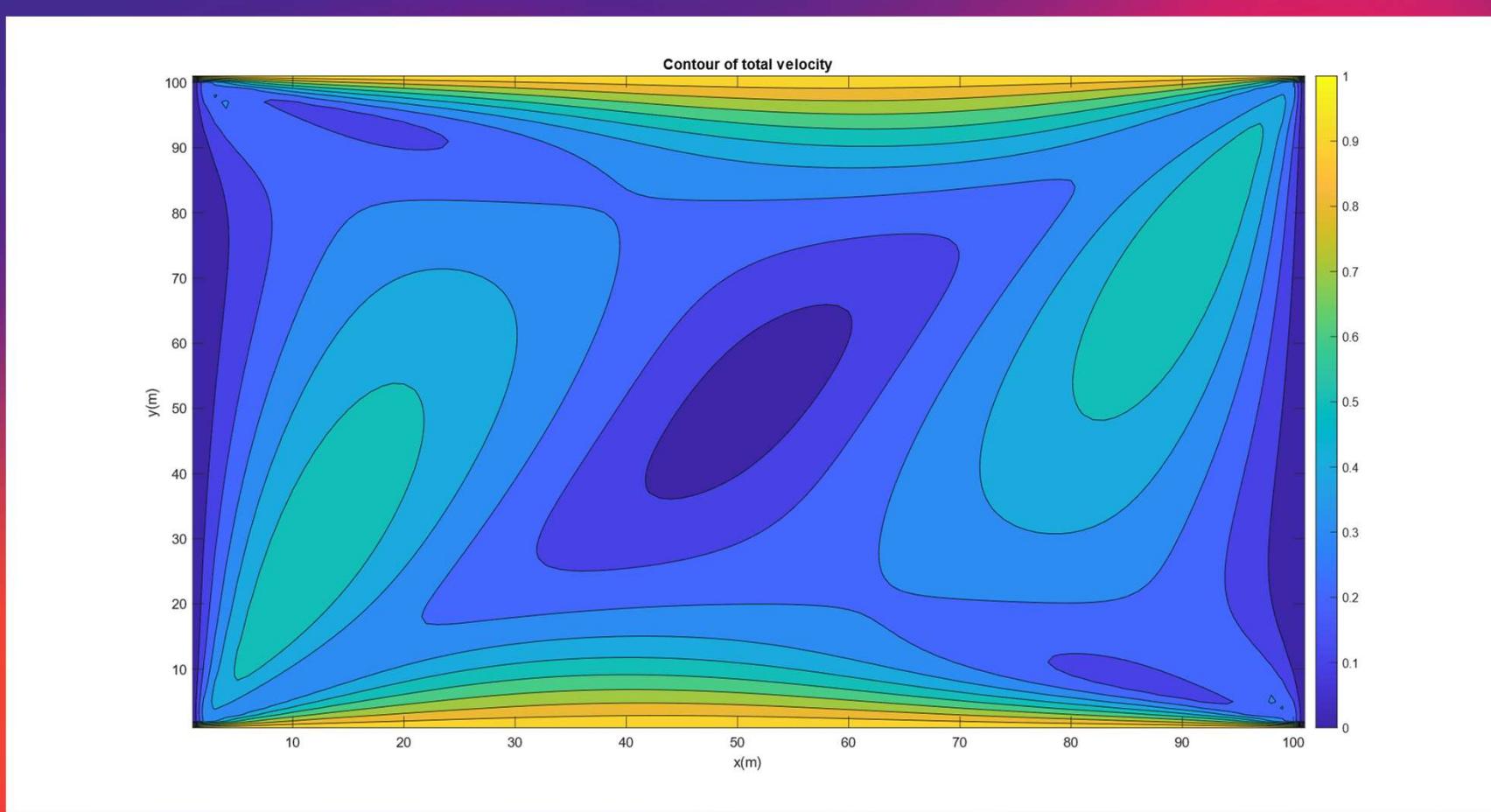
نمودار کانتور سرعت v



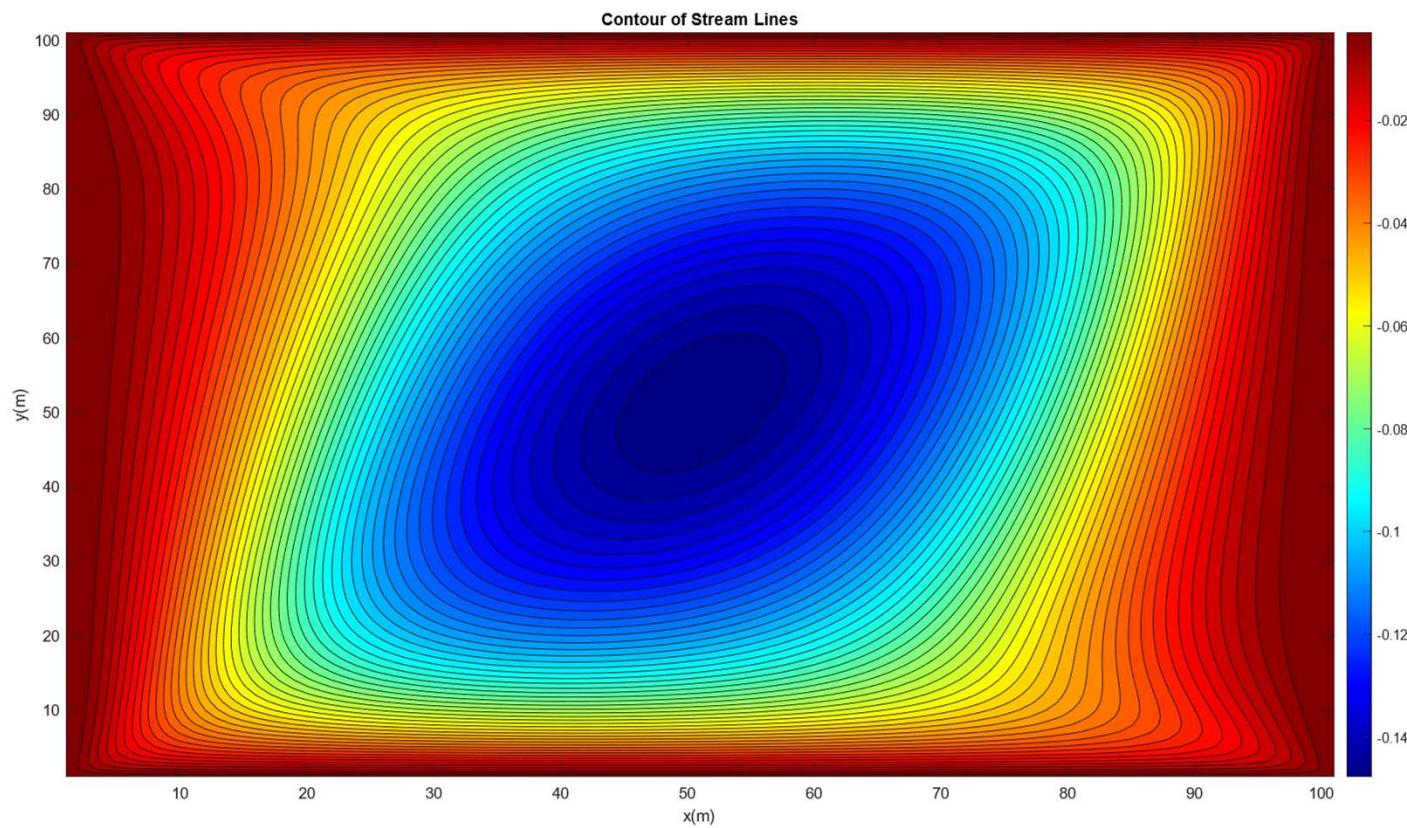
نمودار کانتور سرعت u



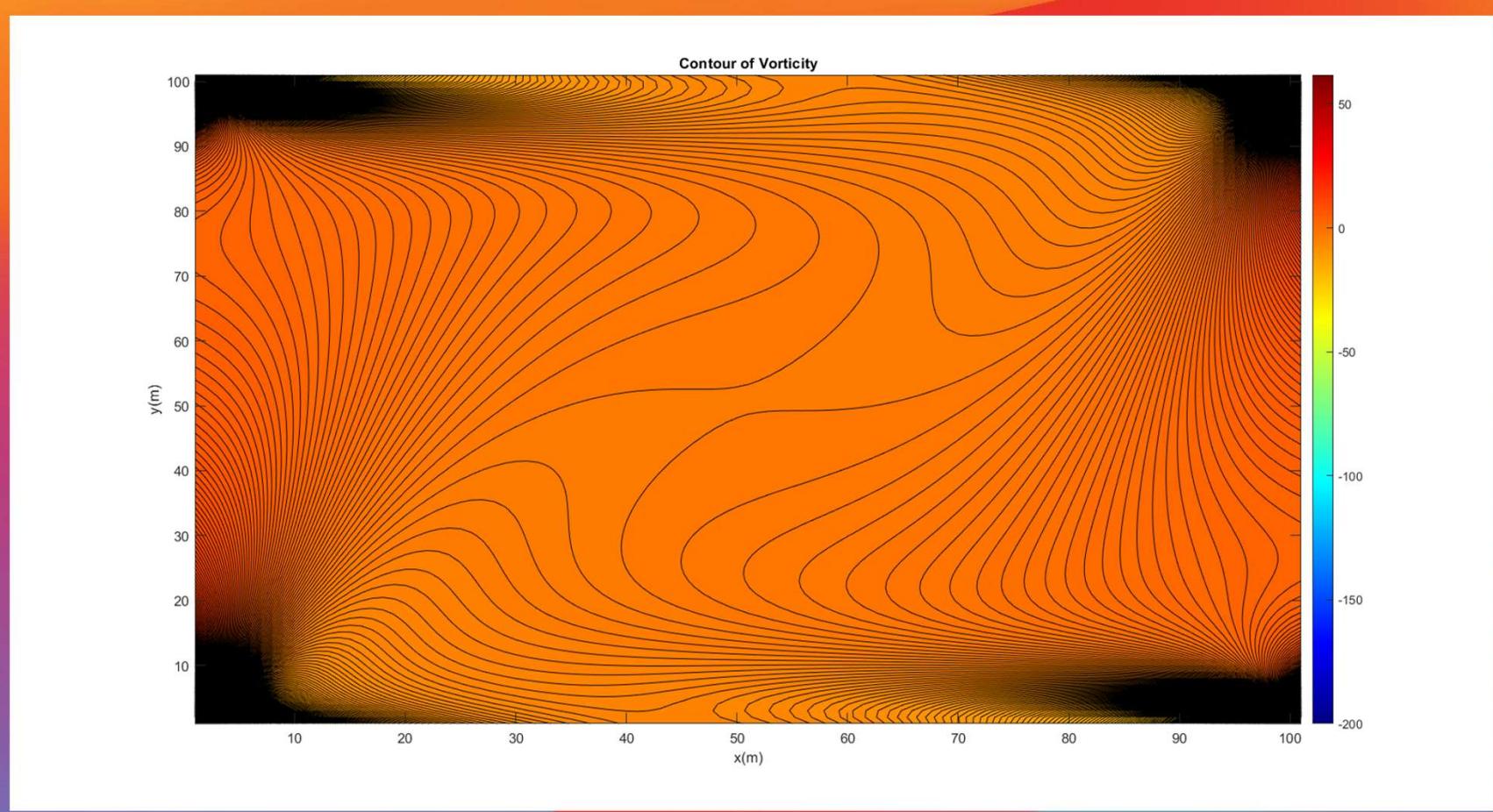
کانتور سرعت کل



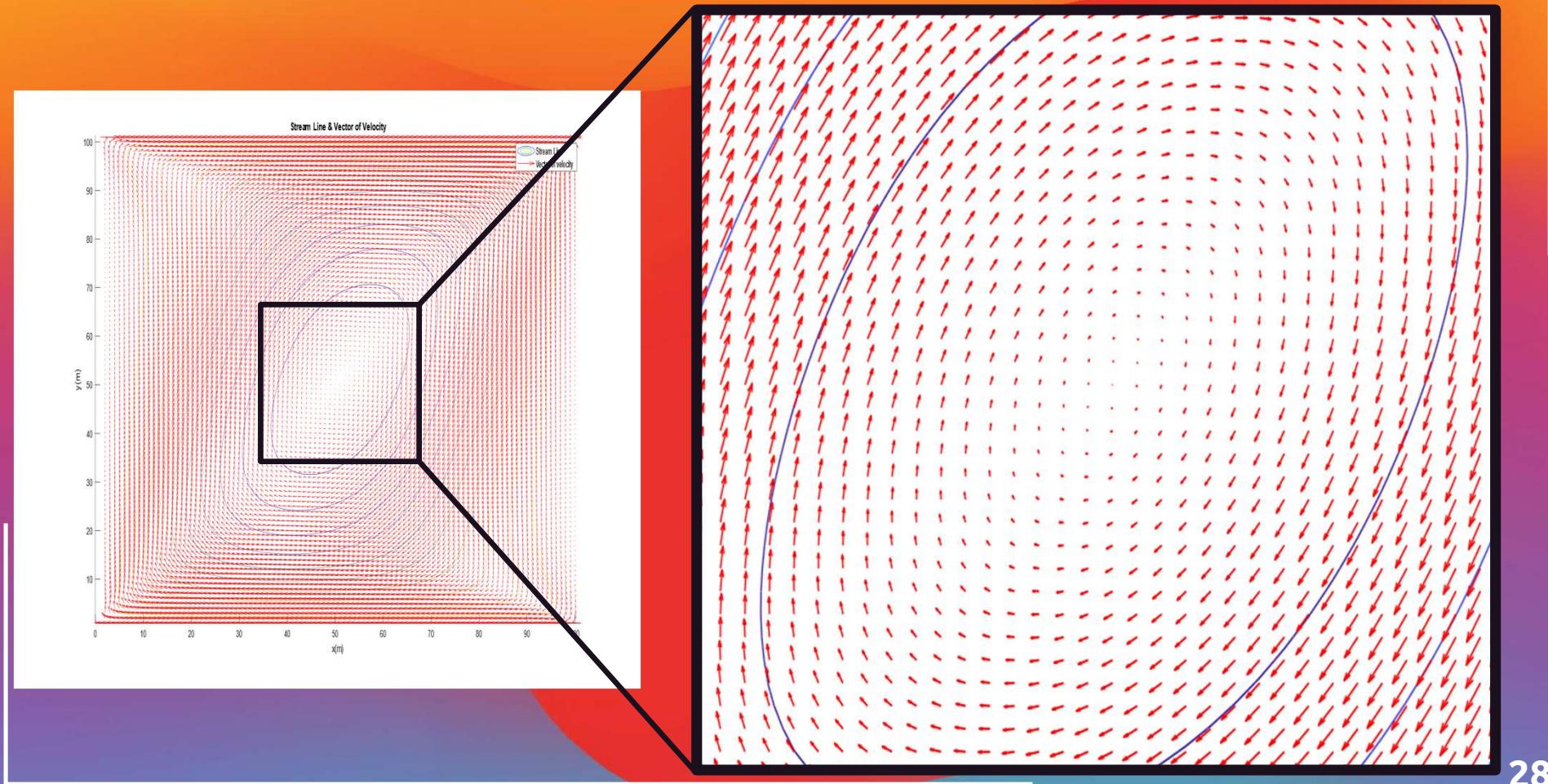
کانتور خطوط جریان



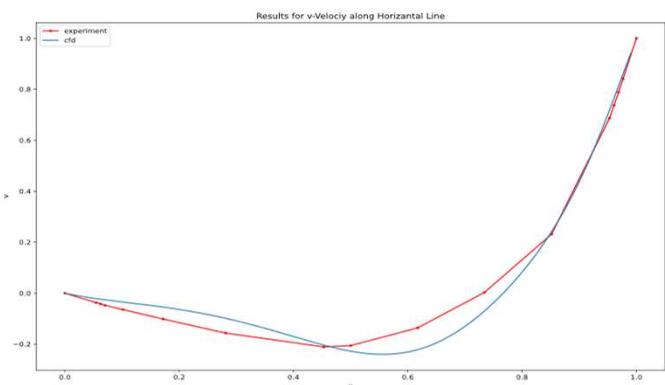
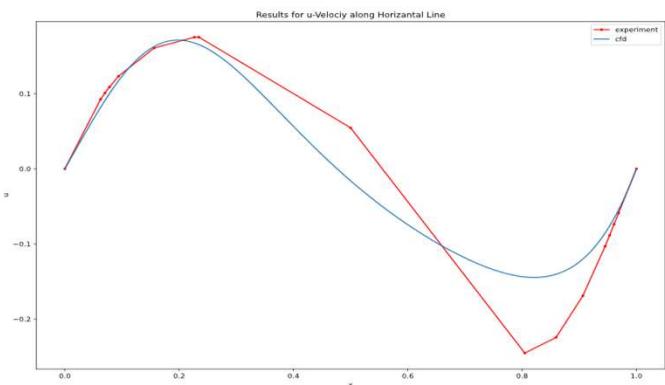
کانتور گردابه



بردار سرعت و خطوط جریان



اعتبار سنجی نسبت به داده های مقاله (با کد پایتون)



```
x = [0, 0.0547, 0.0625, 0.0703, 0.1016, 0.1719, 0.2813, 0.4531,
     0.5, 0.6172, 0.7344, 0.8516, 0.9531, 0.9609, 0.9688, 0.9766, 1]

y = [0.0, -0.03717, -0.04192, -0.04775, -0.06434, -0.10150, -0.15662, -0.2109,
     -0.20581, -0.13641, 0.00332, 0.23151, 0.68717, 0.73722, 0.78871, 0.84123, 1]

plt.figure(figsize=(15,10))
plt.plot(x,y,'r.-',label='experiment')
plt.plot(np.linspace(0,1,C3.nx)[ :-2],C3.v[ :,int(C3.v.shape[0]/2)][ :-2],label='cfd')
plt.legend()
plt.title("Results for v-Velociy along Horizontal Line")
plt.xlabel("x")
plt.ylabel("v")

plt.savefig("v.svg")
files.download("v.svg")
```

```
x = [1, 0.9688 ,0.9609 ,0.9531 ,0.9453 ,0.9063 ,0.8594 ,0.8047 ,0.5000
     ,0.2344 ,0.2266 ,0.1563 ,0.0938 ,0.0781 ,0.0703 ,0.0625 ,0.0000 ]

y = [0.00000 ,-0.05906 ,-0.07391 ,-0.08864 ,-0.10313 ,-0.16914 ,-0.22445 ,-0.24533
     ,0.05454 ,0.17527 ,0.17507 ,0.16077 ,0.12317 ,0.10890 ,0.10091 ,0.09233 ,0.0]

plt.figure(figsize=(15,10))
plt.plot(x,y,'r.-',label='experiment')
plt.plot(np.linspace(0,1,C3.nx),C3.u[ int(C3.u.shape[0]/2),:],label='cfd')
plt.legend()
plt.title("results for u-Velociy along Horizontal Line")
plt.xlabel("x")
plt.ylabel("u")

plt.savefig("u.svg")
files.download("u.svg")
```

همانطور که از نمودار های تحقیق شده مشاهده شد در حالت ثابت بودن حفره خطوط جریان از مرز بالایی که محل ورود جریان به حفره است شروع به تغییر کرده و سرعت های u و v به ترتیب از مرز بالا و گوشه ها به سرعت تغییر میکنند و با افزایش عمق این مقادیر کمتر میشود اما در حالت متحرک بودن مرز بالا و پایین این اتفاق از هر دو طرف رخ میدهد و با گذشت زمان گردابه های کوچک ایجاد شده به هم پیوسته و خطوط جریان به صورت یک گردابه ی یکتا در مرکز حفره شکل میگرد

انجام تحلیل اضافی : حل مسئله با الگوریتم های مختلف پایتونی و مقایسه زمان حل و حل با رینولذ های مختلف

در محاسبه ی مقادیر فوق با استفاده از الگوریتم حلقه ای ۵۰۰ ثانیه زمان مورد نیاز است که با استفاده از numpy و روش ماتریسی این زمان به ۲۰ ثانیه کاهش پیدا میکند همچنین در صورت استفاده از torch و بکارگیری gpu زمان صرف شده فقط ۵ ثانیه خواهد بود

بدیهی است نتایج تحقیق شده در مراحل قبل بر اثر تغییر عدد رینولذ دستخوش تغییر میشود که در اسلایدهای بعد مورد بررسی قرار خواهد گرفت

مقایسه الگوریتم های حل مسئله و تحقیق برای رینولدز های مختلف

```
import numpy as np
import matplotlib.pyplot as plt
import torch as T
import time

U = 1

U_up = U
U_down = -U
U_left = 0
U_right = 0
L = 1
H = 1
Re = 100
nu = U * L / Re
nx = 251
ny = 251
delta_x = L / (nx - 1)
delta_y = H / (ny - 1)
beta = delta_x / delta_y
beta2 = beta ** 2

sf = np.zeros((nx,ny))
sf0 = np.zeros((nx,ny))
w = np.zeros((nx,ny))
w0 = np.zeros((nx,ny))
u = np.zeros((nx,ny))
v = np.zeros((nx,ny))

u[0,:,:] = U_down
u[-1,:,:] = U_up
v[:,0] = U_left
v[:, -1] = U_right
```

- در این قسمت با استفاده از زبان برنامه نویسی پایتون مقایسه الگوریتم های حل مسئله و تحقیق برای رینولدز های مختلف می پردازیم ،
- مراحل حل مانند قسمت های قبل است و کد های هر الگوریتم در اسلاید های بعد آورده شده ،
- در این اسلاید کدهای مربوط به مقادیر اساسی و شرایط مرزی ذکر شده ،

الگوریتم حلقه ای (Iteration algorithm)

```

def sf_update(SF, w, delta_x, beta2):
    nx,ny = SF.shape
    sf = SF.copy()
    for i in range(1,nx-1):
        for j in range(1,ny-1):
            sf[i,j] = (sf[i+1,j] + sf[i-1,j] +\
                        beta2 * sf[i,j+1] + beta2 * sf[i,j-1] +\
                        (delta_x ** 2) * w[i,j]) / (2 + 2*beta2)

    return sf

def w_update(SF, W, U, V, delta_x, delta_y, nu):
    nx, ny = SF.shape

    sf = SF.copy()
    w = W.copy()
    u = U.copy()
    v = V.copy()

    w[0,:] = -2 * (sf[1,:]*u[0,:]) / (delta_x **2)
    w[-1,:] = -2 * (sf[-2,:]*u[-1,:]) / (delta_x **2)
    w[:,0] = -2 * (sf[:,1]*v[:,0]) / (delta_y **2)
    w[:, -1] = -2 * (sf[:, -2]*v[:, -1]) / (delta_y **2)

    beta2 = (delta_x / delta_y) **2
    for i in range(1,nx-1):
        for j in range(1,ny-1):
            u[i,j] = (sf[i,j+1] - sf[i,j-1]) / (2 * delta_y)
            v[i,j] = -(sf[i+1,j] - sf[i-1,j]) / (2 * delta_x)

            dwdx = (w[i+1,j] - w[i-1,j])/(2 * delta_x)
            dwdy = (w[i,j+1] - w[i,j-1])/(2 * delta_y)

            A = (u[i,j] * dwdx + v[i,j] * dwdy)/ nu

            w[i,j] = (w[i+1,j] + w[i-1,j] + beta2*w[i,j+1] + beta2 * w[i,j-1] - (delta_x**2) * A)/(2 + 2*beta2)

    return w,u,v

```

```

step = 0
Error_target = 1e-02
Error = 1000

sf_residuals = []
w_residuals = []
u_residuals = []
v_residuals = []

sf_imgs = []
w_imgs = []
v_imgs = []
u_imgs = []
V_total = []

startTime = time.time()
while (Error > Error_target):
    sf_new = sf_update(sf, w, delta_x, beta2)

    w_new, u_new, v_new = w_update(sf_new,w,u,v, delta_x, delta_y, nu)

    error_sf = np.max(np.abs(sf_new - sf))
    error_w = np.max(np.abs(w_new - w))
    error_u = np.max(np.abs(u_new - u))
    error_v = np.max(np.abs(v_new - v))

    Error = max([error_sf, error_w, error_v, error_u])
    print("{} - {:.4e} {:.4e} {:.4e} {:.4e}".format(step, error_sf,error_w,error_u, error_v))

    sf = sf_new
    w = w_new
    u = u_new
    v = v_new

    sf_residuals.append(error_sf)
    w_residuals.append(error_w)
    v_residuals.append(error_v)
    u_residuals.append(error_u)

    sf_imgs.append(sf)
    w_imgs.append(w)
    u_imgs.append(u)
    v_imgs.append(v)
    V_total.append( np.sqrt(u**2 + v**2))
    step += 1

```

Numpy Roll Algorithm

```
[ ] def sf_update_roll(SF, w, delta_x, beta2):
    nx,ny = SF.shape
    sf = SF.copy()
    sf[1:-1,1:-1] = ((np.roll(sf,1,0) + np.roll(sf,-1,0) +\n        beta2 * np.roll(sf,1,1) + beta2* np.roll(sf,-1,1) +\n        (delta_x ** 2) * w) / (2 + 2*beta2))[1:-1,1:-1]

    return sf

[ ] def w_update_roll(SF, W, U, V, delta_x, delta_y, nu):
    nx, ny = SF.shape

    sf = SF.copy()
    w = W.copy()
    u = U.copy()
    v = V.copy()

    w[0,:] = -2 * (sf[1,:]+ delta_x * u[0,:]) / (delta_x **2)
    w[-1,:] = -2 * (sf[-2,:]+ delta_x * u[-1,:]) / (delta_x **2)
    w[:,0] = -2 * (sf[:,1] + delta_y * v[:,0]) / (delta_y **2)
    w[:, -1] = -2 * (sf[:, -2] + delta_y * v[:, -1]) / (delta_y **2)

    beta2 = (delta_x / delta_y) **2

    u[1:-1,1:-1] = ((np.roll(sf,1,1) - np.roll(sf,-1,1)) / (2 * delta_y))[1:-1,1:-1]
    v[1:-1,1:-1] = -(np.roll(sf,1,0) - np.roll(sf,-1,0)) / (2 * delta_x))[1:-1,1:-1]

    dwdx = (np.roll(w,1,0) - np.roll(w,-1,0))/(2 * delta_x)
    dwdy = (np.roll(w,1,1) - np.roll(w,-1,1))/(2 * delta_y)

    A = (u * dwdx + v * dwdy)/ nu

    w[1:-1,1:-1] = ((np.roll(w,1,0) + np.roll(w,-1,0) + beta2*np.roll(w,1,1) + beta2 * np.roll(w,-1,1)- (delta_x**2) * A)/(2 + 2*beta2))[1:-1,1:-1]
    return w,u,v
```

ادامه کدهای الگوریتم numpy

```
step = 0
Error_target = 1e-02
Error = 1000

sf_residuals = []
w_residuals = []
v_residuals = []
u_residuals = []

sf_imgs = []
w_imgs = []
v_imgs = []
u_imgs = []
V_total = []

startTime = time.time()
while (Error > Error_target):
    sf_new = sf_update_roll(sf, w, delta_x, beta2)

    w_new, u_new, v_new = w_update_roll(sf,w,u,v, delta_x, delta_y, nu)

    error_sf = np.max(np.abs(sf_new - sf))
    error_w = np.max(np.abs(w_new - w))
    error_u = np.max(np.abs(u_new - u))
    error_v = np.max(np.abs(v_new - v))

    Error = max([error_sf, error_w, error_v, error_u])
    print("{} - {:.4e} {:.4e} {:.4e} {:.4e}".format(step, error_sf,error_w,error_u, error_v))

    sf = sf_new
    w = w_new
    u = u_new
    v = v_new

    sf_residuals.append(error_sf)
    w_residuals.append(error_w)
    v_residuals.append(error_v)
    u_residuals.append(error_u)

    sf_imgs.append(sf)
    w_imgs.append(w)
    u_imgs.append(u)
    v_imgs.append(v)
    V_total.append( np.sqrt(u**2 + v**2))
    step += 1
```

الگوریتم Torch roll

Torch roll algorithm

```
[ ] def sf_update_roll_T(SF, w, delta_x, beta2):
    device = T.device("cuda" if T.cuda.is_available() else "cpu")
    nx,ny = SF.shape
    sf = T.tensor(SF,device=device)
    w = T.tensor(w, device=device)
    sf[1:-1,1:-1] = ((T.roll(sf,1,0) + T.roll(sf,-1,0) +\n        beta2 * T.roll(sf,1,1) + beta2* T.roll(sf,-1,1) +\n        (delta_x ** 2) * w) / (2 + 2*beta2))[1:-1,1:-1]

    return sf.cpu().detach().numpy()

[ ] def w_update_roll_T(SF, W, U, V, delta_x, delta_y, nu):
    nx, ny = SF.shape
    device = T.device("cuda" if T.cuda.is_available() else "cpu")
    sf = T.tensor(SF, device=device)
    w = T.tensor(W, device=device)
    u = T.tensor(U, device=device)
    v = T.tensor(V, device=device)

    w[0,:] = -2 * (sf[1,:]+ delta_x * u[0,:]) / (delta_x **2)
    w[-1,:] = -2 * (sf[-2,:]+ delta_x * u[-1,:]) / (delta_x **2)
    w[:,0] = -2 * (sf[:,1] + delta_y * v[:,0]) / (delta_y **2)
    w[:, -1] = -2 * (sf[:, -2] + delta_y * v[:, -1]) / (delta_y **2)

    beta2 = (delta_x / delta_y) **2

    u[1:-1,1:-1] = ((T.roll(sf,1,1) - T.roll(sf,-1,1)) / (2 * delta_y))[1:-1,1:-1]
    v[1:-1,1:-1] = -((T.roll(sf,1,0) - T.roll(sf,-1,0)) / (2 * delta_x))[1:-1,1:-1]

    dwdx = (T.roll(w,1,0) - T.roll(w,-1,0))/(2 * delta_x)
    dwdy = (T.roll(w,1,1) - T.roll(w,-1,1))/(2 * delta_y)

    A = (u * dwdx + v * dwdy)/ nu

    w[1:-1,1:-1] = ((T.roll(w,1,0) + T.roll(w,-1,0) + beta2*T.roll(w,1,1) + beta2 * T.roll(w,-1,1)- (delta_x**2) * A)/(2 + 2*beta2))[1:-1,1:-1]
    return w.cpu().detach().numpy(), u.cpu().detach().numpy(), v.cpu().detach().numpy()
```

ادامه کد های الگوریتم Torch roll

```
step = 0
Error_target = 1e-02
Error = 1000

frame = 10

sf_residuals = []
w_residuals = []
u_residuals = []
v_residuals = []

sf_imgs = []
w_imgs = []
u_imgs = []
v_imgs = []
V_total = []

startTime = time.time()
while (Error > Error_target):
    sf_new = sf_update_roll_T(sf, w, delta_x, beta2)

    w_new, u_new, v_new = w_update_roll_T(sf,w,u,v, delta_x, delta_y, nu)

    error_sf = np.max(np.abs(sf_new - sf))
    error_w = np.max(np.abs(w_new - w))
    error_u = np.max(np.abs(u_new - u))
    error_v = np.max(np.abs(v_new - v))

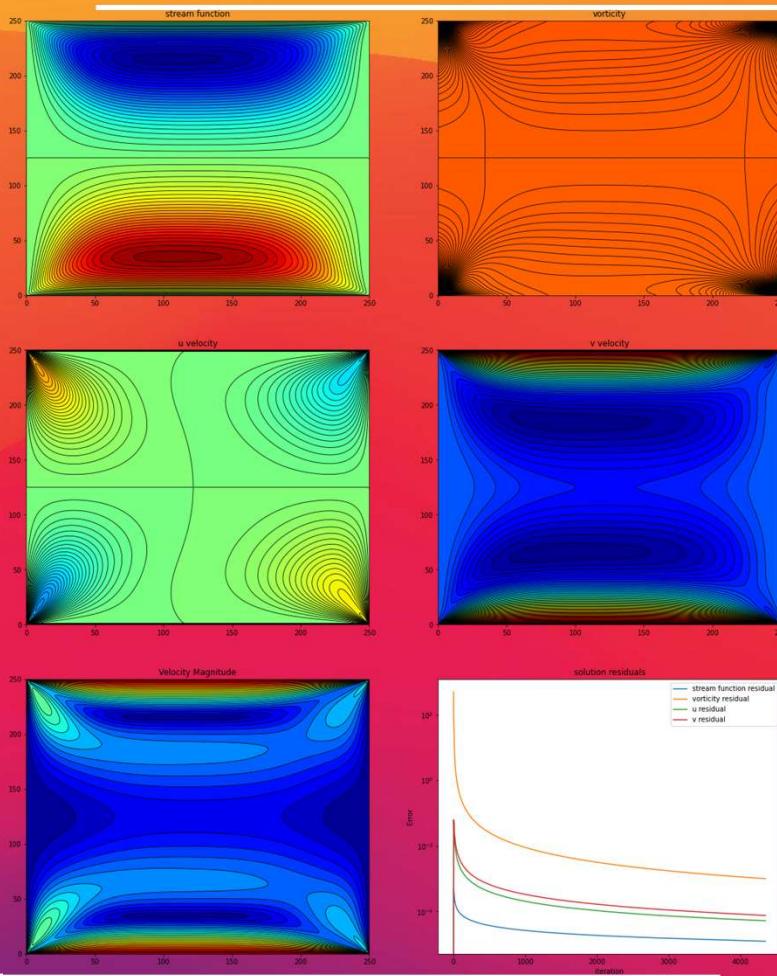
    Error = max([error_sf, error_w, error_v, error_u])
    print("{} - {:.4e} {:.4e} {:.4e} {:.4e}".format(step, error_sf,error_w,error_u, error_v))

    sf = sf_new
    w = w_new
    u = u_new
    v = v_new

    sf_residuals.append(error_sf)
    w_residuals.append(error_w)
    v_residuals.append(error_v)
    u_residuals.append(error_u)

    if (step % frame == 0):
        sf_imgs.append(sf)
        w_imgs.append(w)
        u_imgs.append(u)
        v_imgs.append(v)
        V_total.append( np.sqrt(u**2 + v**2))
    step += 1
```

نتایج بدست آمده برای $Re=100$



Class Cavity

```

class Cavity:
    def __init__(self, L = 1, H = 1, Re = 100, nx = 101, ny=101, U_up =1, U_down = 0, U_left = 0, U_right = 0):
        self.L = L
        self.H = H
        self.Re = Re
        self.nx = nx
        self.ny = ny

        self.U_up = U_up
        self.U_down = U_down
        self.U_left = U_left
        self.U_right = U_right

        self.nu = U_up * L / Re

        self.delta_x = L / (nx - 1)
        self.delta_y = H / (ny - 1)
        self.beta = self.delta_x / self.delta_y
        self.beta2 = self.beta ** 2

        self.sf = np.zeros((nx,ny))
        self.w = np.zeros((nx,ny))
        self.u = np.zeros((nx,ny))
        self.v = np.zeros((nx,ny))

        self.u[0,:] = U_down
        self.u[-1,:] = U_up
        self.v[:,0] = U_left
        self.v[:, -1] = U_right

        self.device = T.device("cuda" if T.cuda.is_available() else "cpu")

        self.sf_residuals = []
        self.w_residuals = []
        self.u_residuals = []
        self.v_residuals = []

        self.sf_imgs = []
        self.w_imgs = []
        self.v_imgs = []
        self.u_imgs = []
        self.V_total = []

        self.step = 0
    
```

```

def sf_update_roll_T(self):
    sf = T.tensor(self.sf,device=self.device)
    w = T.tensor(self.w, device=self.device)
    sf[1:-1,1:-1] = ((T.roll(sf,1,0) + T.roll(sf,-1,0) +\n                     self.beta2 * T.roll(sf,1,1) + self.beta2* T.roll(sf,-1,1) +\n                     (self.delta_x ** 2) * w) / (2 + 2*self.beta2))[1:-1,1:-1]

    return sf.cpu().detach().numpy()

def w_update_roll_T(self):
    sf = T.tensor(self.sf, device=self.device)
    w = T.tensor(self.w, device=self.device)
    u = T.tensor(self.u, device=self.device)
    v = T.tensor(self.v, device=self.device)

    w[0,:] = -2 * (sf[1,:]+ self.delta_x * u[0,:]) / (self.delta_x **2)
    w[-1,:] = -2 * (sf[-2,:]+ self.delta_x * u[-1,:]) / (self.delta_x **2)
    w[:,0] = -2 * (sf[:,1] + self.delta_y * v[:,0]) / (self.delta_y **2)
    w[:, -1] = -2 * (sf[:, -2] + self.delta_y * v[:, -1]) / (self.delta_y **2)

    u[1:-1,1:-1] = ((T.roll(sf,1,1) - T.roll(sf,-1,1)) / (2 * self.delta_y))[1:-1,1:-1]
    v[1:-1,1:-1] = -((T.roll(sf,1,0) - T.roll(sf,-1,0)) / (2 * self.delta_x))[1:-1,1:-1]

    dwdx = (T.roll(w,1,0) - T.roll(w,-1,0))/(2 * self.delta_x)
    dwdy = (T.roll(w,1,1) - T.roll(w,-1,1))/(2 * self.delta_y)

    A = (u * dwdx + v * dwdy)/ self.nu

    w[1:-1,1:-1] = ((T.roll(w,1,0) + T.roll(w,-1,0) + self.beta2*T.roll(w,1,1) +\n                     self.beta2 * T.roll(w,-1,1)- (self.delta_x**2) * A)/(2 + 2*self.beta2))[1:-1,1:-1]

    return w.cpu().detach().numpy(), u.cpu().detach().numpy(), v.cpu().detach().numpy()

```

Class Cavity

```
def solve(self,residual_target, frame):
    self.residual_target = residual_target

    Error = 1000

    startTime = time.time()
    while (Error > self.residual_target):
        sf_new = self.sf_update_roll_T()

        w_new, u_new, v_new = self.w_update_roll_T()

        error_sf = np.max(np.abs(sf_new - self.sf))
        error_w = np.max(np.abs(w_new - self.w))
        error_u = np.max(np.abs(u_new - self.u))
        error_v = np.max(np.abs(v_new - self.v))

        Error = max([error_sf, error_w, error_v, error_u])
        print("{} - {:.4e}      {:.4e}      {:.4e}      {:.4e}".format(self.step, error_sf,error_w,error_u, error_v))

        self.sf = sf_new
        self.w = w_new
        self.u = u_new
        self.v = v_new

        self.sf_residuals.append(error_sf)
        self.w_residuals.append(error_w)
        self.v_residuals.append(error_v)
        self.u_residuals.append(error_u)

    if (self.step % frame == 0):
        self.sf_imgs.append(self.sf)
        self.w_imgs.append(self.w)
        self.u_imgs.append(self.u)
        self.v_imgs.append(self.v)
        self.V_total.append( np.sqrt(self.u**2 + self.v**2))

    self.step += 1

print('-----')
print("Run time: {:e}".format(time.time() -startTime))
```

```
def plot_residuals(self):
    plt.figure(figsize=(10,5))
    plt.yscale('log')
    plt.plot(self.sf_residuals,label='stream function residual')
    plt.plot(self.w_residuals, label='vorticity residual')
    plt.plot(self.u_residuals, label='u residual')
    plt.plot(self.v_residuals, label='v residual')
    plt.legend()
    plt.xlabel('iteration')
    plt.ylabel('Error')
    _ = plt.title("solution residuals")

def plot_stream(self):
    plt.figure(figsize=(12,10))
    plt.rcParams['contour.negative_linestyle'] = 'solid'
    plt.contour(self.sf, 50,colors='k',linewidths=0.8)
    plt.contourf(self.sf,50,cmap='jet')
    plt.title("Stream function - Re:{} , {}x{}".format(self.Re,self.nx, self.ny))
    plt.colorbar()

def plot_vorticity(self):
    plt.figure(figsize=(12,10))
    plt.rcParams['contour.negative_linestyle'] = 'solid'
    plt.contour(self.w,1000,colors='k',linewidths=0.8)
    plt.contourf(self.w,1000,cmap='hot')
    plt.title("Vorticity")
    plt.colorbar()

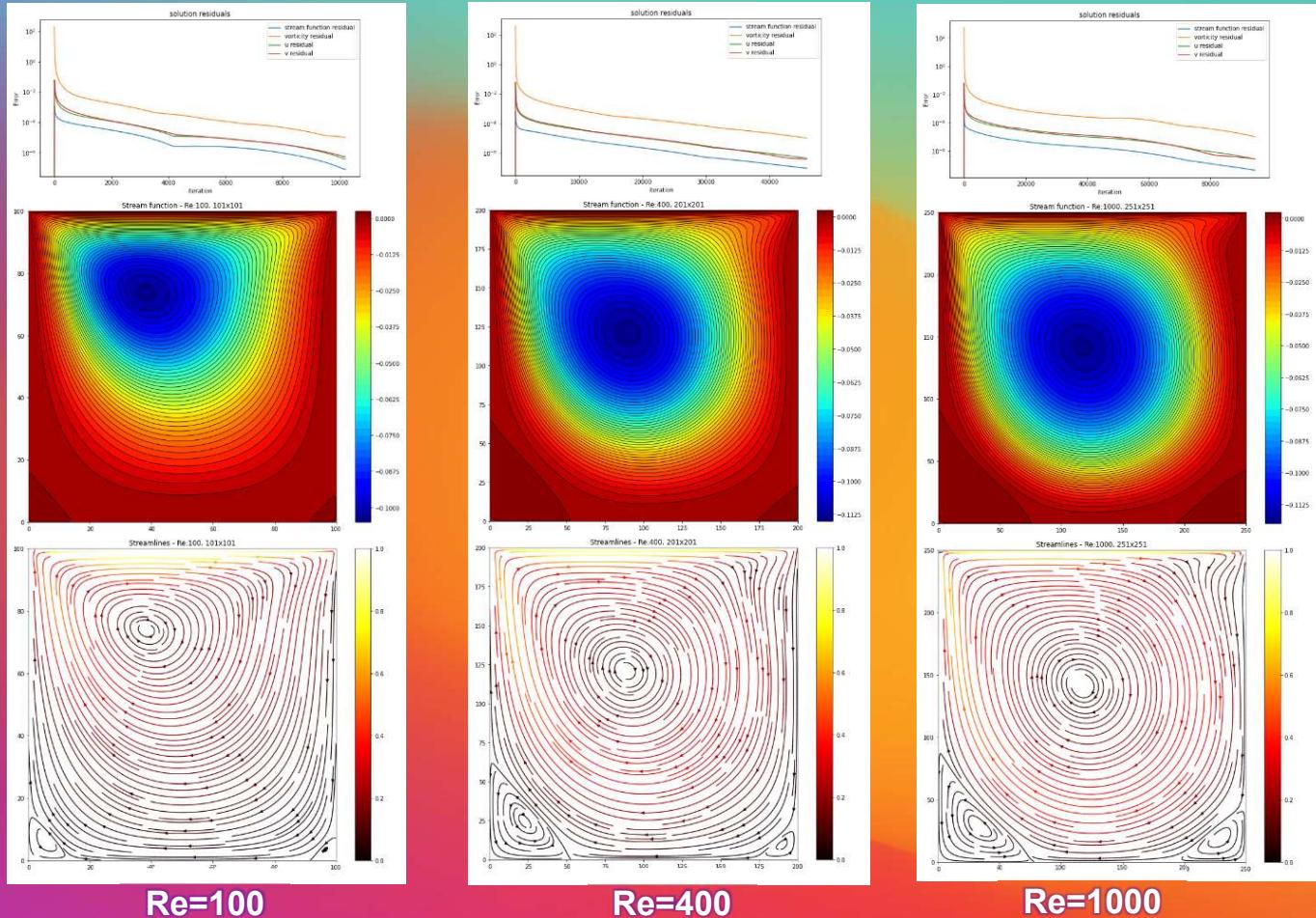
def plot_u(self):
    plt.figure(figsize=(12,10))
    plt.rcParams['contour.negative_linestyle'] = 'solid'
    plt.contour(self.u,100,colors='k',linewidths=0.8)
    plt.contourf(self.u,100,cmap='jet')
    plt.title("u velocity")

def plot_v(self):
    plt.figure(figsize=(12,10))
    plt.rcParams['contour.negative_linestyle'] = 'solid'
    plt.contour(self.v,100,colors='k',linewidths=0.8)
    plt.contourf(self.v,100,cmap='jet')
    plt.title("v velocity")

def plot_Velocity(self):
    plt.figure(figsize=(12,10))
    plt.rcParams['contour.negative_linestyle'] = 'solid'
    V_total = np.sqrt(self.u**2 + self.v**2)
    plt.contour(V_total, 100, colors='k', linewidths=0.8)
    plt.contourf(V_total, 100, cmap='jet')
    plt.title("Velocity Magnitude")

def plot_streamlines(self):
    plt.figure(figsize=(12,10))
    plt.rcParams['contour.negative_linestyle'] = 'solid'
```

نتایج بدست آمده برای رینولدز های مختلف

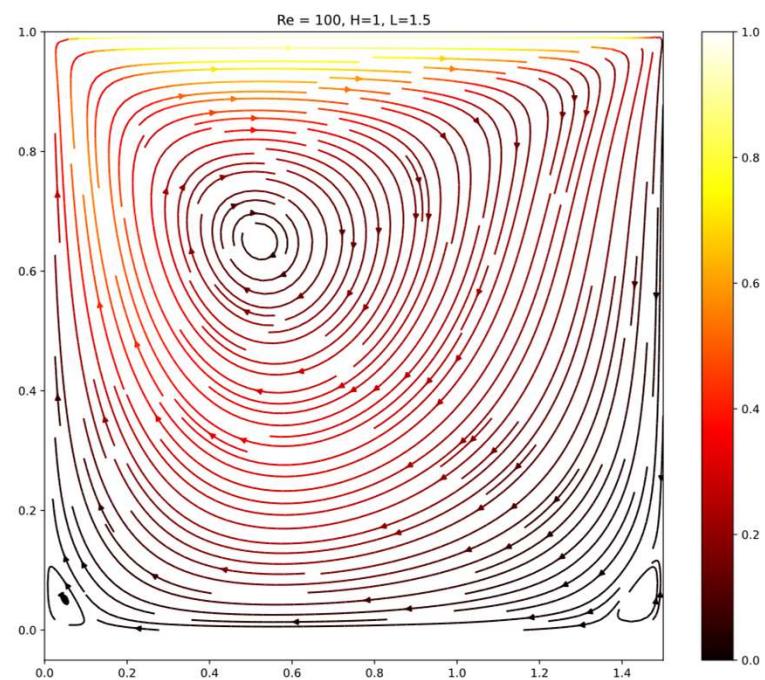
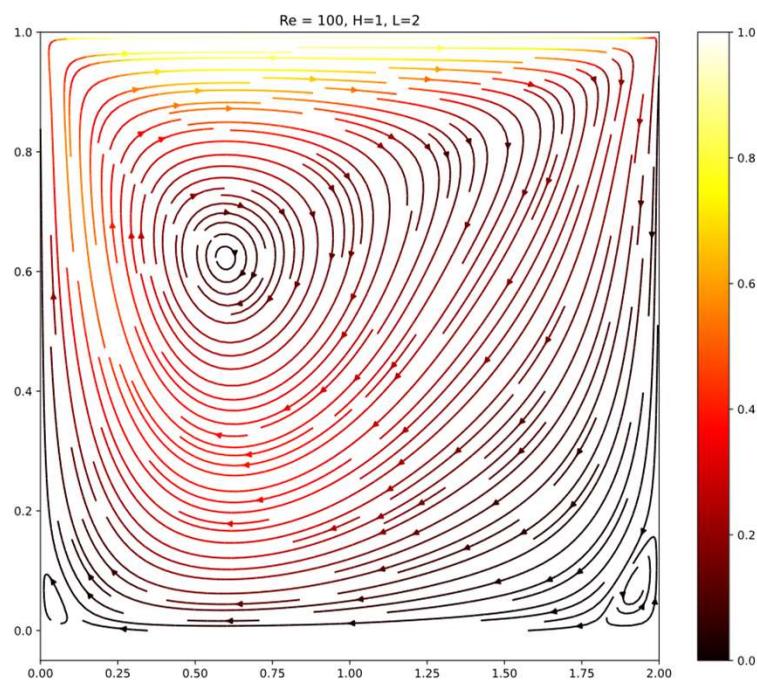


Re=100

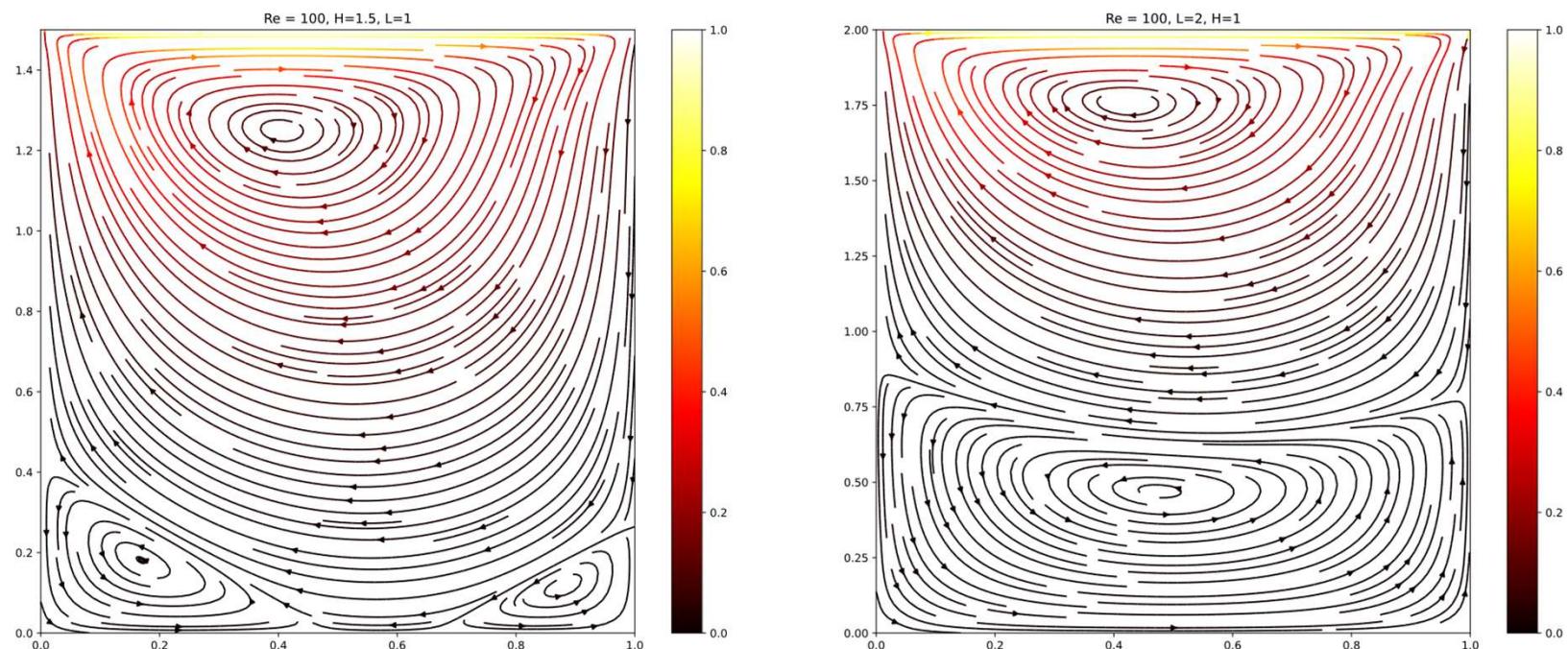
Re=400

Re=1000

مقایسه نسبت‌های مختلف H به L



مقایسه نسبت‌های مختلف L به H





با تشکر از توجه شما