Flujo en una cavidad

Josué Juárez Morales

La ecuacion de continuidad y la ecuacion de Navier-Stokes se componen de un sistema de ecuaciones para la presion y la densidad. adicionalmente, una variable más para cada coordenada de la velocidad considerada. Para un sisema de ecuaciones para la presion **P**, la densidad ρ y las velocidades en x y en y se tienen cuatro variables y tres ecuaciones diferenciales. Es ahí que debemos añadir constricciones a nuestro sistema de ecuaciones.

Consideremos un fluido inscomprensible, es decir, para un $\mathbf{u} = (u, v)$ se cumple la siguiente relación.

$$\nabla \cdot \mathbf{u} = 0, \tag{1}$$

o bien

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0. \tag{2}$$

Para un fluído de estas características, la ecuación de Navier-Stokes se reduce a

$$\frac{\partial \vec{v}}{\partial t} + (\vec{v} \cdot \nabla)\vec{v} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \vec{v}, \tag{3}$$

la cual es la ecuación de conservación de momentos.

Escribimos la ecuación de Navier-Stokes en notación tensorial, usando la convención de Einstein

$$\partial_i u_i + u^k \partial_k u_i = -\frac{1}{\rho}\partial_i P + v\partial^k \partial_k u_i, \tag{4}$$

al aplica el operador divergencia a ésta ecuación.

$$\partial^i \partial_i u_i + \partial^i u^k \partial_k u_i = -\frac{1}{\rho}\partial^i \partial_i P + v\partial^i \partial^k \partial_k u_i, \tag{5}$$

dado que la coordenadas son independientes, los operadores diferenciales conmutan. Reescribimos como

$$\partial_i(\partial^i u_i) + (\partial^i u^k)(\partial_k u_i) + u^k \partial_k(\partial^i u_i) = -\frac{1}{\rho}\partial^i \partial_i P + v\partial^k \partial_k(\partial^i u_i), \tag{6}$$

como estamos analizando un fluido incompresible $\partial^i u_i = 0$ obtenemos

$$(\partial^i u^k)(\partial_k u_i) = -\frac{1}{\rho}\partial^i \partial_i P, \tag{7}$$

en dos dimensiones escribimos como

$$-\frac{1}{\rho}\left(\frac{\partial^2 P}{\partial x^2}+\frac{\partial^2 P}{\partial y^2}\right)=\left(\frac{\partial u}{\partial x}\right)^2+2\left(\frac{\partial u}{\partial y}\right)\left(\frac{\partial v}{\partial x}\right)+\left(\frac{\partial v}{\partial x}\right)^2, \tag{8}$$

en adición con la ecuación de Navier - Stokes en dos dimensiones

$$\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}=-\frac{1}{\rho}\frac{\partial p}{\partial x}+\nu\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right), \tag{9}$$

$$\frac{\partial v}{\partial t}+u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}=-\frac{1}{\rho}\frac{\partial p}{\partial y}+\nu\left(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}\right), \tag{10}$$

y con este par de ecuaciones completamos un sistema de tres ecuaciones con tres variables. Lo que basta ahora es resolver numéricamente bajo los mismos métodos que hemos estudiado.

Podemos discretizar estas dos ecuaciones como lo hemos hecho en las anteriores pasos

$$\begin{aligned}\frac{u_{i,j}^{n+1}-u_{i,j}^n}{\Delta t}+u_{i,j}^n\frac{u_{i,j}^n-u_{i-1,j}^n}{\Delta x}+v_{i,j}^n\frac{u_{i,j}^n-u_{i,j-1}^n}{\Delta y}&=\\-\frac{1}{\rho}\frac{p_{i+1,j}^n-p_{i-1,j}^n}{2\Delta x}+\nu\left(\frac{u_{i+1,j}^n-2u_{i,j}^n+u_{i-1,j}^n}{\Delta x^2}+\frac{u_{i,j+1}^n-2u_{i,j}^n+u_{i,j-1}^n}{\Delta y^2}\right)&,\end{aligned} \tag{11}$$

$$\begin{aligned}\frac{v_{i,j}^{n+1}-v_{i,j}^n}{\Delta t}+u_{i,j}^n\frac{v_{i,j}^n-v_{i-1,j}^n}{\Delta x}+v_{i,j}^n\frac{v_{i,j}^n-v_{i,j-1}^n}{\Delta y}&=\\-\frac{1}{\rho}\frac{p_{i,j+1}^n-p_{i,j-1}^n}{2\Delta y}+\nu\left(\frac{v_{i+1,j}^n-2v_{i,j}^n+v_{i-1,j}^n}{\Delta x^2}+\frac{v_{i,j+1}^n-2v_{i,j}^n+v_{i,j-1}^n}{\Delta y^2}\right)&.\end{aligned} \tag{12}$$

En cuanto a la ecuación de Poisson, la podemos escribir como

$$\frac{p_{i+1,j}^n-2p_{i,j}^n+p_{i-1,j}^n}{\Delta x^2}+\frac{p_{i,j+1}^n-2p_{i,j}^n+p_{i,j-1}^n}{\Delta y^2}=\rho f_{i,j}^n, \tag{13}$$

definimos el termino

$$\begin{aligned}f_{i,j}^n=&\frac{1}{\Delta t}\left(\frac{u_{i+1,j}-u_{i-1,j}}{2\Delta x}+\frac{v_{i,j+1}-v_{i,j-1}}{2\Delta y}\right)-\frac{u_{i+1,j}-u_{i-1,j}}{2\Delta x}\frac{u_{i+1,j}-u_{i-1,j}}{2\Delta x}\\&-2\frac{u_{i,j+1}-u_{i,j-1}}{2\Delta y}\frac{v_{i+1,j}-v_{i-1,j}}{2\Delta x}\\&-\frac{v_{i,j+1}-v_{i,j-1}}{2\Delta y}\frac{v_{i,j+1}-v_{i,j-1}}{2\Delta y},\end{aligned} \tag{14}$$

y entonces la iteración de la para $p_{i,j}^n$ queda como

$$p_{i,j}^n=\frac{\left(p_{i+1,j}^n+p_{i-1,j}^n\right)\Delta y^2+\left(p_{i,j+1}^n+p_{i,j-1}^n\right)\Delta x^2}{2\left(\Delta x^2+\Delta y^2\right)}-\frac{\rho\Delta x^2\Delta y^2}{2\left(\Delta x^2+\Delta y^2\right)}f_{i,j}^n \tag{15}$$

Como ejemplo usamos las condiciones iniciales $u, v, p = 0$ en todas partes, y las condiciones de frontera son:

$u = 1$ en $y = 2$ en el borde
$u, v = 0$ en las ontras fronteras;
$\frac{\partial p}{\partial y} = 0$ en $y = 0$;
$p = 0$ en $y = 2$
$\frac{\partial p}{\partial x} = 0$ en $x = 0, 2$

[0]:
```
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
```

[0]:
```
nx = 41
ny = 41
dt = 0.001
paro = 100
Lx = 2
Ly = 2
dx = Lx/(nx-1)
dy = Ly/(ny-1)
x = np.linspace(0, Lx, nx)
y = np.linspace(0, Ly, ny)
X, Y = np.meshgrid(x,y)

rho = 1
nu = 0.1

def plot3D(x, y, phi):
  fig = plt.figure(figsize=(11,7), dpi=100)
  ax = fig.gca(projection='3d')
  X, Y = np.meshgrid(x,y)
  surf = ax.plot_surface(X, Y, phi.transpose(), rstride=1, cstride=1, cmap=cm.
 ↪viridis, linewidth=0, antialiased=False)
  ax.set_xlim(0, max(x))
  ax.set_ylim(0, max(y))
  ax.set_xlabel("$x$")
  ax.set_ylabel("$y$")
```

El termino $f_{i,j}^n$ se calcula en una función aparte. Esto con el proposito de poder reutilizar el codigo que ya hicimos para la ecuación de Poisson.

El término $f_{i,j}^n$ se interpreta como una función funete de la ecuacion de Poisson.

[0]:
```
def Fnij(f, u, v):
  for i in range(1, nx-1):
    for j in range(1, ny-1):
      f[i,j] = ((u[1+i, j] - u[i-1, j])/(2*dx) + (v[i, j+1]- v[i, j-1])/(2*dy))/
 ↪dt - (u[i+1, j] - u[i-1, j])*(u[i+1, j] - u[i-1, j])/(4*dx*dx) - 2*(u[i+1, j]␣
 ↪- u[i-1, j])*(v[i, j+1] - v[i,j-1])/(4*dx*dy)
  return f
```

[0]:
```python
def poisson_2d(p, x, y, f, precision, n):
  norma = 1
  pn = np.empty_like(p)

  pasos = 0
  for k in range(paro):
    pn = p.copy()
    for i in range(1, len(x)-1):
      for j in range(1, len(y)-1):
        p[i, j] = (dy*dy*(pn[i+1, j] + pn[i-1, j]) + dx*dx*(pn[i, j+1] + pn[i,␣
↪j-1]) - dx*dx*dy*dy*rho*f[i,j])/(2.0*(dx*dx + dy*dy)) #se añade el termino f

    p[:,0] = p[:,1]
    p[:,-1] = 0
    p[0,:] = p[1,:]
    p[-1,:] = p[-2,:]

    if np.sum(np.abs(pn[:])) == 0.0:
      pasos = pasos + 1
      continue

    norma = np.sum(np.abs(p[:])) - np.abs(pn[:])/np.sum(np.abs(pn[:]))

    pasos = pasos + 1

  #print("convergge a los", pasos, "pasos con una precision de", norma)
  return p
```

[0]:
```python
def cavidad_2D(nt, u, v, p, rho, nu):
    un = np.empty_like(u)
    vn = np.empty_like(v)
    f = np.zeros((nx,ny))

    for n in range(nt):
        #print('Paso temporal nt = ',n)
        un = u.copy()
        vn = v.copy()

        f = Fnij(f, u , v)
        p = poisson_2d(p,x,y,f,1e-4,n)


        for i in range(1,nx-1):
            for j in range(1,ny-1):
```

```
                u[i,j] = un[i,j] - un[i,j]*dt*(un[i,j] - un[i-1,j])/dx -␣
↪vn[i,j]*dt*(un[i,j] - un[i,j-1])/dy - dt*(p[i+1,j] - p[i-1,j])/(2.0*rho*dx) +␣
↪nu*dt*(un[i+1,j] - 2.0*un[i,j] + un[i-1,j])/(dx*dx)+ dt*(un[i,j+1] - 2.0*un[i,␣
↪j] + un[i,j-1])/(dy*dy)
                v[i,j] = vn[i,j] - vn[i,j]*dt*(vn[i,j] - vn[i-1,j])/dx -␣
↪vn[i,j]*dt*(vn[i,j] - vn[i,j-1])/dy - dt*(p[i,j+1] - p[i,j-1])/(2.0*rho*dy) +␣
↪nu*dt*(vn[i+1,j] - 2.0*vn[i,j] + vn[i-1,j])/(dx*dx) + dt*(vn[i,j+1] - 2.
↪0*vn[i, j] + vn[i,j-1])/(dy*dy)




        u[0,:] = 0.0
        u[-1,:] = 0.0
        u[:, 0] = 0.0
        u[:, -1] = 1.0

        v[0,:] = 0.0
        v[-1,:] = 0.0
        v[:,0] = 0.0
        v[:,-1] = 0.0

    return u, v, p
```

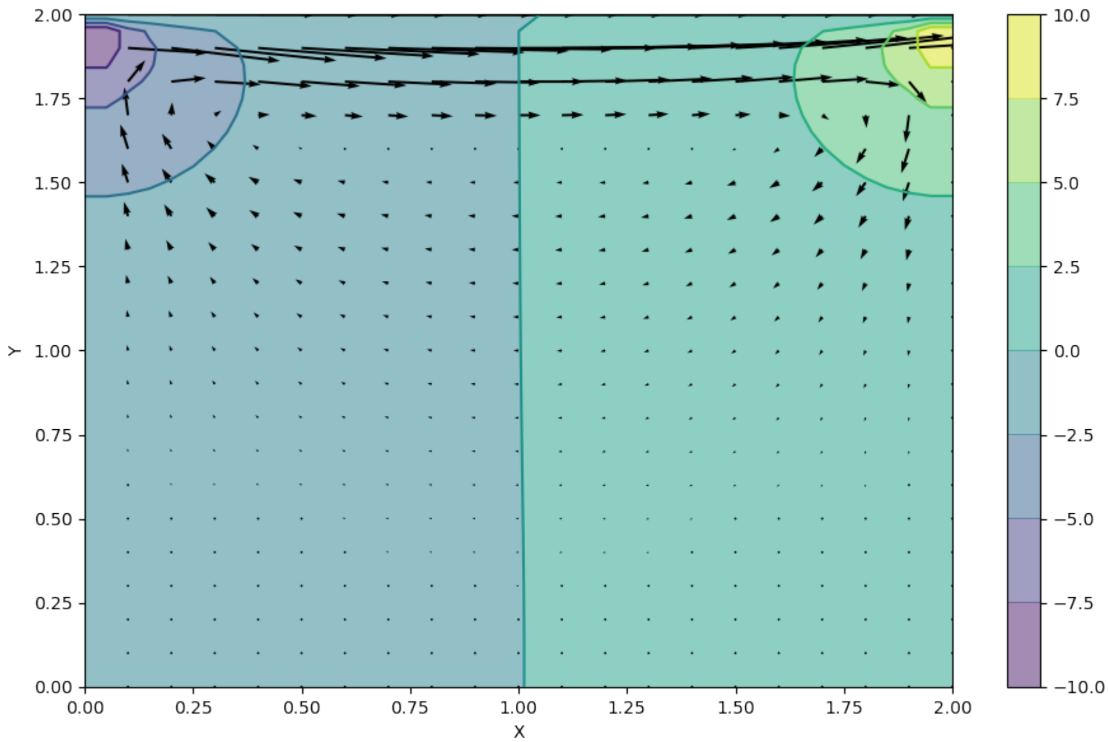
Hacemos evolucionar el sistema para $nt = 20$ pasos.

[0]:
```
u = np.zeros((nx,ny))
v = np.zeros((nx,ny))
p = np.zeros((nx,ny))

u, v, p = cavidad_2D(20, u, v, p, rho, nu)
```

[0]:
```
fig = plt.figure(figsize=(11,7), dpi=100)
# plotting the pressure field as a contour
plt.contourf(X, Y, p.transpose(), alpha=0.5, cmap=cm.viridis)
plt.colorbar()
# plotting the pressure field outlines
plt.contour(X, Y, p.transpose(), cmap=cm.viridis)
# plotting velocity field
plt.quiver(X[::2, ::2], Y[::2, ::2], u[::2, ::2].transpose(), v[::2, ::2].
↪transpose())
plt.xlabel('X')
plt.ylabel('Y')
```

[0]:
```
Text(0, 0.5, 'Y')
```



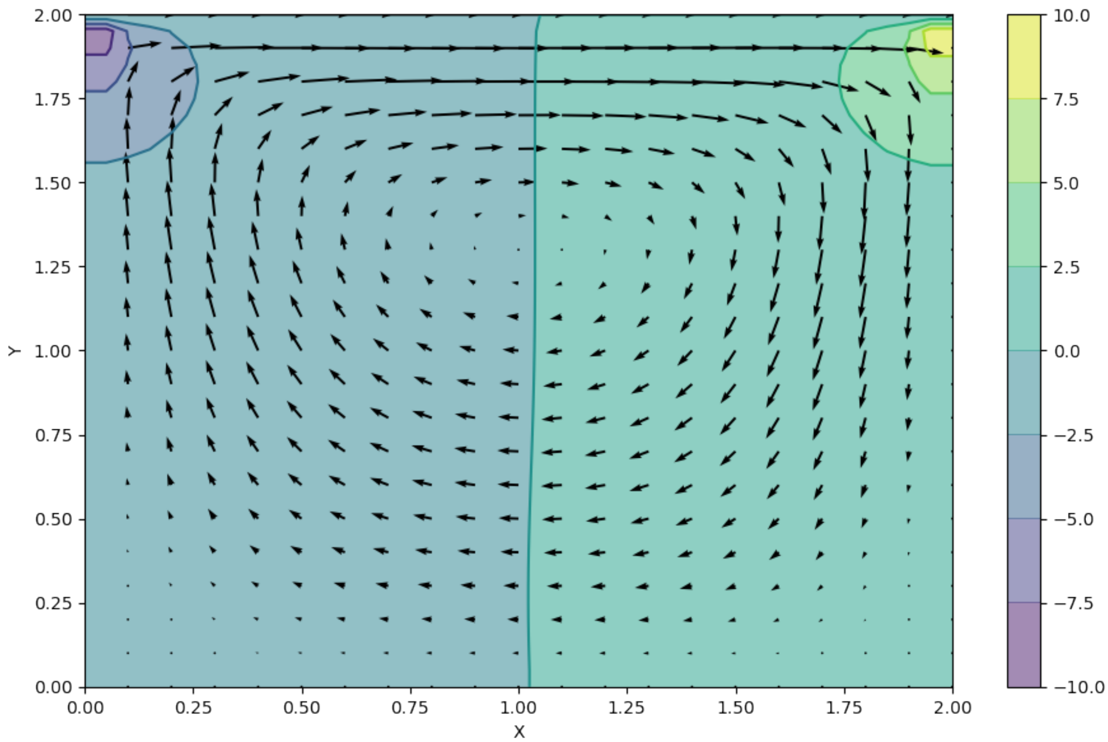
Se observa como los dos puntos de presion se empiezan a formar, asi como el patron en forma de espiral esperado.

Aumentando el valor *nt* podemos ver como el sistema llega mas estable.

[0]:
```
u = np.zeros((ny, nx))
v = np.zeros((ny, nx))
p = np.zeros((ny, nx))
b = np.zeros((ny, nx))
u, v, p = cavidad_2D(700, u, v, p, rho, nu)
```

[0]:
```
fig = plt.figure(figsize=(11,7), dpi=100)
# plotting the pressure field as a contour
plt.contourf(X, Y, p.transpose(), alpha=0.5, cmap=cm.viridis)
plt.colorbar()
# plotting the pressure field outlines
plt.contour(X, Y, p.transpose(), cmap=cm.viridis)
# plotting velocity field
plt.quiver(X[::2, ::2], Y[::2, ::2], u[::2, ::2].transpose(), v[::2, ::2].
 ↪transpose())
plt.xlabel('X')
plt.ylabel('Y')
```

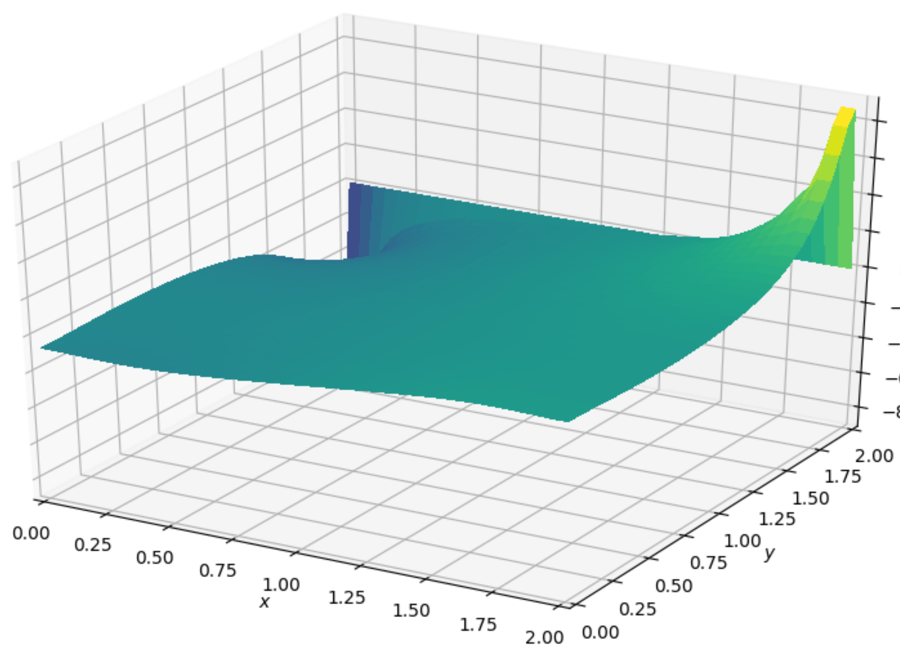
[0]:
```
Text(0, 0.5, 'Y')
```



2.00
1.75
1.50
1.25
Y 1.00
0.75
0.50
0.25
0.00
0.00 0.25 0.50 0.75 1.00 1.25 1.50 1.75 2.00
X
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0

[0]:

```
plot3D(x,y,p)
```



8
6
4
2
0
−2
−4
−6
−8
0.00 0.25 0.50 0.75 1.00 1.25 1.50 1.75 2.00
x
0.00 0.25 0.50 0.75 1.00 1.25 1.50 1.75 2.00
y