Incompressible Flows About an Object In a Driven Vertically Repeating Rectangular Domain

Forrest Bullard[1]

[1]California State University, Chico

December 12, 2020

Introduction

In this report we will discuss how the Navier-Stokes momentum equation for incompressible flows can be adapted to a numerical solution so that analysis of fluid flow around a fixed object can be made. We will discuss a simplification of the Navier-Stokes equation by a change of variable from velocity to vorticity, the curl of velocity, and the stream function who's spatial derivatives will return the components of our velocity for each section of flow. As well we will examine what boundary conditions are necessary in order to create a flow through our domain and other conditions which will define the boundaries of our object. Finally we will discuss some of the limitations of this examination.

Math

In our derivation we began with the incompressible Navier-Stokes equation of the form,

$$\frac{d\mathbf{u}}{dt} = -\frac{1}{\rho}\nabla p + \mathbf{g} + \nu\nabla^2\mathbf{u} \tag{1}$$

we will then take the curl of this formula to change it into the vorticity form.

$$\nabla \times [\frac{d\mathbf{u}}{dt} = -\frac{1}{\rho}\nabla p + \mathbf{g} + \nu\nabla^2\mathbf{u}] \tag{2}$$

The curl of the first two terms on the right side of this equation will both be zero as the curl of the gradient of any scalar function is zero. As long as $\mathbf{g}$ is conservative it can be expressed as the gradient of a scalar. Here we must also note that the gradient of the curl of any vector field is zero. Expanding the left side of this equation gives,

$$\nabla \times [\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u}] = \frac{\partial \omega}{\partial t} + \nabla \times [(\mathbf{u}\cdot\nabla)\mathbf{u}] = \frac{\partial \omega}{\partial t} + \nabla \times [\nabla(\frac{\mathbf{u}\cdot\mathbf{u}}{2}) + \omega \times \mathbf{u}]$$

$$= \frac{\partial \omega}{\partial t} + \nabla \times (\omega \times \mathbf{u})$$

So our equation reduces to,

$$\frac{\partial \omega}{\partial t} + \nabla \times (\omega \times \mathbf{u}) = \nu \nabla^2 \omega$$

Using a few vector identities gives us the form of our equation before the use of the stream function. Namely,

$$\frac{d\omega}{dt} = (\omega \cdot \nabla)\mathbf{u} + \nu \nabla^2 \omega \tag{3}$$

We now have the field equations for the vorticity in a fluid with constant ρ. We can see that $\nu \nabla^2 u$ will represent the rate of change of the ω caused by diffusion of vorticity and $(\omega \cdot \nabla)\, u$ representing the rate of vorticity caused by the stretching and tilting of vortex lines. We can also see that the central body forces for gravity and pressure are not found in this equation as they cause no torques. To relate this equation to the equation we will use to numerically solve our system we will relate the vorticity and velocity functions by the stream function such that,

$$\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = -\omega \tag{4}$$

and

$$u = \frac{\partial \psi}{\partial y}; \quad v = -\frac{\partial \psi}{\partial x} \tag{5}$$

We may now write the advection-diffusion equation for the vorticity.

$$\frac{\partial \omega}{\partial t} = -\frac{\partial \psi}{\partial y}\frac{\partial \omega}{\partial x} + \frac{\partial \psi}{\partial x}\frac{\partial \omega}{\partial y} + \nu\left(\frac{\partial^2 \omega}{\partial x^2} + \frac{\partial^2 \omega}{\partial y^2}\right) \tag{6}$$

This coupled with equation 4, the Poission equation that relates the stream function to vorticity, gives us the self-contained system we will need to solve for the flow given the appropriate boundary and initial conditions. The boundary values for the stream function can be found along the wall by use of equation 5. This as well shows that for a stationary object inside our rectangular domain we will need a region where both the vorticity and stream function is zero. We will also create repeated boundaries at the top and bottom by solving for the stream function at the bottom as if the next increment down is the first zone from the top and then by resolving the stream function at the top layer with the solutions from the bottom.

Numerical Approximations

The process for our numerical approximation will be as stated in Fluid Mechanics section 6.3

1. Given $\omega_{i,j}$ at all interior points, solve for $\psi_{i,j}$.

2. Find the boundary vorticity, ω_{wall}.

3. Calculate the vorticity at the new time, $\omega_{i,j}^{n+1}$ for all interior points.

4. Set $t = t + \Delta t$ and go back to first step.

For step one we will use the equation,

$$\frac{\psi^n_{i+1,j} + \psi^n_{i-1,j} + \psi^n_{i,j+1} + \psi^n_{i,j-1} - 4\psi^n_{i,j}}{h^2} = -\omega^n_{i,j} \tag{7}$$

along with a successive over relaxation method so we will converge to a solution faster. The boundary conditions for the stream function will not change during the progression of the program but we will have to calculate for the vorticity function along the wall separately before we may calculate the next time step for the vorticity function inside. The formula for solving the vorticity function along the wall will involve a Taylor expansion at the boundaries and will be of the form,

$$\omega_{wall} = (\psi_{i,1} - \psi_{i,2})\frac{2}{h^2} + U_{wall} \cdot \frac{2}{h} + O(h) \tag{8}$$

Once we know the vorticity function on the wall we can use a forward difference method to approximate the next time step of the vorticity function inside using the following equation,

$$\frac{\omega^{n+1}_{i,j} - \omega^n_{i,j}}{\Delta t} = -\left(\frac{\psi^n_{i,j+1} - \psi^n_{i,j-1}}{2h}\right)\left(\frac{\omega^n_{i+1,j} - \omega^n_{i-1,j}}{2h}\right) + \left(\frac{\psi^n_{i+1,j} - \psi^n_{i-1,j}}{2h}\right)\left(\frac{\omega^n_{i,j+1} - \omega^n_{i,j-1}}{2h}\right)$$

$$+\nu\left(\frac{\psi^n_{i+1,j} + \psi^n_{i-1,j} + \psi^n_{i,j+1} + \psi^n_{i,j-1} - 4\psi^n_{i,j}}{h^2}\right) \tag{9}$$

Note that these equations have been simplified under the assumption that the spacing between the grid points will be the same in the x and y direction. It was also necessary to reapply the region of our object having zero vorticity and zero stream after every cycle.

Code

The following was the code, written in python, used to examine the flow of fluid with different viscosities about a circular object. It was necessary to find appropriate time steps as related to the size of the box and to the given viscosity so that convergence could be maintained as low viscosity does not allow for enough dissipation of the vorticity function.

“”” Created on Fri Jan 4 13:19:28 2019

Based on code from Fluid Dynamics sixth edition section 6.3 (code 3)

Solution for Unsteady Two-Dimensional Navier-Stokes equations in vorticity-stream function form, using an explicit forward in time, centered in space scheme

Wind tunnel simulation expanded by repeated boundary conditions on horizontal boundary and net zero flux on vertical boundaries

Succesive Over Relaxation (SOR) is used for generation of stream function, assuming uniform resolution

@author: fbullard “””

from scipy import zeros, linspace, meshgrid, sqrt from numpy import sum as sumpy import matplotlib.pyplot as plt

#parameters for SOR

scale=1 ; Ny=60 ; Nx=(Ny*scale)-scale+1 ; MaxStep=1000 ; Visc=.5 ; dt=0.00005 ; t = 0.0; MaxIt=600 ; Beta=1.5 ; MaxErr=0.001 ;

#functions and intitial conditions

sf=zeros([Ny, Nx]) ; vt=zeros([Ny, Nx]) ; vto=zeros([Ny,Nx]) ; Ui = .1; height=2.0

#for contour mapping. h is also used for boundary conditions on stream function

x=zeros([Ny,Nx]) ; y=zeros([Ny,Nx]) ; h=height/(Ny-1);

for creation of shapes

shape = zeros([Ny , Nx]) xi = linspace(-(height*scale)/2,(height*scale)/2, Nx , endpoint = True) yj = linspace(-height/2, height/2, Ny, endpoint = True) xx, yy = meshgrid(xi, yj, indexing = 'xy')

for a in range(Nx): for b in range(Ny): if sqrt(xx[b,a]**2 + yy[b,a]2) > 0.3: shape[b,a] = 1

#could be replaced by meshgrid

for i in range(Nx): for j in range(Ny): x[j,i] = h*i y[j,i] = h*j

#generate boundaries for stream function

for j in range(Ny): sf[j,0] = Ui*j*h sf[j,Nx-1] = Ui*j*h

for tstep in range(MaxStep): for SOR in range(MaxIt): vto = sf.copy() for i in range(1,Nx-1): for j in range(1,Ny-1): sf[j,i] = 0.25*Beta*(sf[j,i+1] + sf[j,i-1] + sf[j+1,i] + sf[j-1,i] + h*h*vt[j,i]) + (1.0-Beta)*sf[j,i]

```
        #for repeated boundary condition solve bottom with solution of third from top then set bottom tw

    for i in range(1,Nx-1):
        sf[Ny-1, i] = 0.25*Beta*(sf[Ny-1,i+1] + sf[Ny-1,i-1] + sf[2,i] + sf[Ny-2,i] + h*h*vt[j,i]) + (1
    sf[:2,1:Nx-1] = sf[Ny-2:Ny,1:Nx-1]
    sf *= shape

        #check for convergence of the stream function given some maximum error and steps

    error = sumpy(abs(sf - vto))
    if error <= MaxErr:
        break

    #generate vorticity function for boundaries

vt[1:Ny-1,0] = (2.0*(sf[1:Ny-1,0] - sf[1:Ny-1,1]))/(h*h)
vt[1:Ny-1,Nx-1] = (2.0*(sf[1:Ny-1,Nx-1] - sf[1:Ny-1,Nx-2]))/(h*h)
vt[0,1:Nx-1] = (2.0*(sf[0,1:Nx-1] - sf[1,1:Nx-1]))/(h*h)
vt[Ny-1,1:Nx-1] = (2.0*(sf[0,1:Nx-1] - sf[1,1:Nx-1]))/(h*h)
vto = vt.copy()

#print(tstep)

for i in range(1,Nx-1):
    for j in range(1,Ny-1):
```

```
        vt[j,i] = vt[j,i]+dt*(-0.25*((sf[j+1,i]-sf[j-1,i])*(vto[j,i+1]-vto[j,i-1]) - (sf[j,i+1]-sf[j,i-
        +Visc*(vto[j,i+1]+vto[j,i-1]+vto[j+1,i]+vto[j-1,i]-4.0*vto[j,i])/(h*h) )
vt *= shape

t = t + dt
```

plt.subplot(121) plt.contour(x,y,vt,40) plt.subplot(122) plt.contour(x,y,sf) plt.show()

Results

This code is meant to eventually reach a steady state but the requirements for this to happen will be held in the Reynolds number. That is, the flow into the box must not be too high as compared to the viscosity of the fluid inside the box. In the case that this is not true we will not get enough dissipation in the vorticity function and our solution will eventually diverge. To solve this one must only turn up the viscosity. However we must also note that the time scale at which we advance to the next solution is relative to the size of the box. If we are going to increase the scale at which we are looking to determine a solution we must also decrease the time increment, failure to do so will also result in divergence. This code will determine the stream function and the vorticity function but the often more useful result of the velocity vectors can be easily generated from equations 5. We can see from the next figure an appropriate choice of time step, size, viscosity, and inflow that allows the program to come close to its steady state. These figures show flow lines of constant velocity.

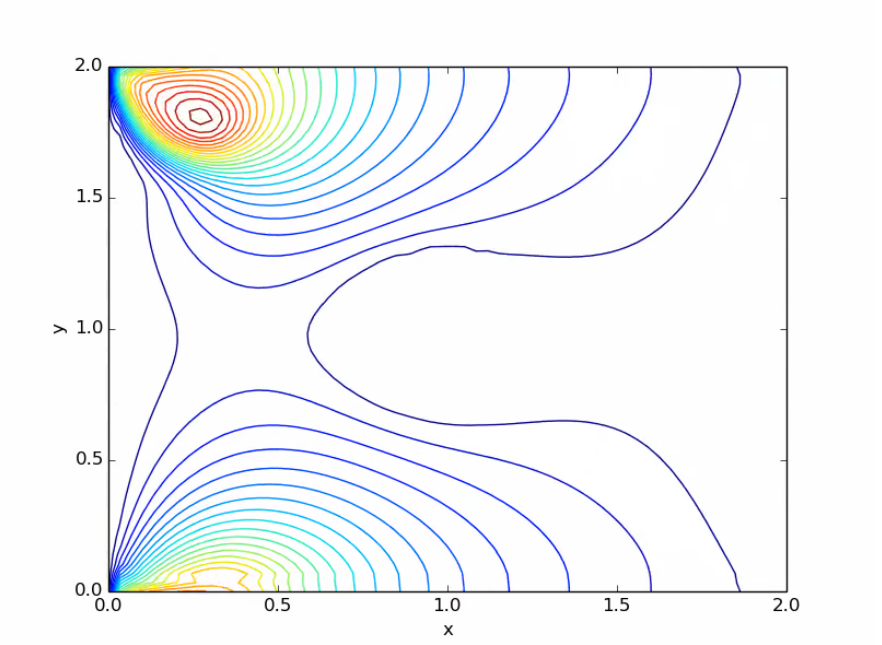

Figure 1: Viscosity = .1, Inflow=.1, grid: (20x20), time step = .0005

The following shows what happens just before instability given a poor choice of parameters.

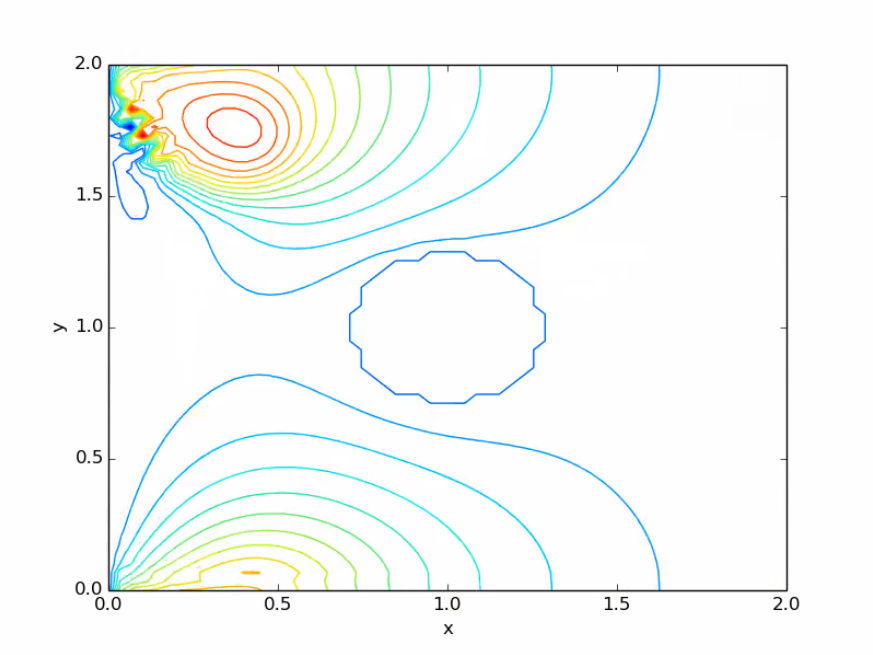

Figure 2: Viscosity=.1, Inflow=.1, grid: (60x60), time step =.00005

One issue with this specific numerical method is the false creation of curl at the corners do to the inability to generate our stream functions in these locations. Part of the reason for the instability as seen in the figure above. This also would affect our ability to determine the true nature of the flow around the object in question but this can be resolved some amount by pushing the boundaries far away as compared to the size of the object in question.