MATHS IN ACTION B – ASSIGNMENT 1

LUKE APPLEBY

Abstract. An investigation of systems of vortices with different levels of fluid viscosity. The merging of two same sign vortices is the focus of the report where we look at the rate of energy decay and the rate at which the vortices merge. Higher viscosity levels were found to result in greater rate of energy decay and greater rate of merging.

1. Introduction

One common way to model flows is with the Euler equations which assume that the only forces between adjacent fluid particles are normal to their surface. The problem with this is it does not account for the viscosity of a fluid which is its resistance to flow relative to surrounding particles. When we include this in our model this gives rise to the Naiver-Stokes equations for viscous fluids. In this report we will investigate how the degree of viscosity affects the interaction of vortices specifically the rate at which same sign vortices merge as well as the energy decay of the system. We use computational methods to simulate the flows. This is not a flawless approach and therefore we will provide some critique of the efficacy of our code in this investigation. The results of this investigation are important to the field of meteorology since in the prediction of weather systems accurately modeling the effects of viscosity in systems of vortices is important, especially when considering the evolution of storms.

2. Viscosity and Naiver-stokes Equations

Viscosity is the degree to which fluids resist motion. It is a concept similar to friction and the restive forces act tangentially opposite to the direction of motion [1]. The equations we use to model flows with non-zero viscosity are the Naiver-Stokes equations [2]. That is

$$\rho D_t \boldsymbol{u} = -\nabla p + \mu \Delta \boldsymbol{u} + (\lambda + \mu)\nabla(\nabla \cdot \boldsymbol{u}) \tag{1}$$

along with the continuity equation

$$\partial_t \rho + \nabla \cdot (\rho \boldsymbol{u}) = D_t \rho + \rho \nabla \cdot \boldsymbol{u}. \tag{2}$$

Here ρ is the density, $\boldsymbol{u}$ is the velocity field, D_t is the material derivative given by

$$D_t f(\boldsymbol{x}, t) = \partial_t f(\phi(a, t), t)|_{a=\phi^{-1}(x,t)}$$

and ϕ is the flow map. The parameters λ, μ are called the first and second coefficients of viscosity and these completely determine the viscous properties of the fluid [2]. These equations are solved in our code in order to simulate the merging of two same sign vortices A.1.

3. Energy equations

One of the goals of this investigation is to examine energy decay with various viscosity levels. The formula for energy is

$$E = \frac{1}{2} \int u^2 + v^2 \, d\underline{x} \tag{3}$$

where u, v are the horizontal and vertical components of velocity respectively. This is computationally prohibitive especially as we will be working in the Fourier space so we need an alternative expression for kinetic energy. The stream-function ψ is such that $u = -\psi_y$ and $v = \psi_x$. Additionally we have that curl $\omega = \nabla^2\psi$. So evaluating the integral

$$-\frac{1}{2}\int \omega\psi \, d\underline{x} = -\frac{1}{2}\int \nabla^2\psi \cdot \psi \, d\underline{x} = \frac{1}{2}\iint \psi_{xx}\psi + \psi_{yy}\psi \, dxdy \tag{4}$$

we integrate each term separately with $\psi\psi_{xx}$ first in x then in y and likewise $\psi\psi_{yy}$ first in y then in x. Integrating in parts with $f = \psi$, $dg = \psi_{xx}$ gives $df = \psi_x$ and $g = \psi_x$ respectively. So $\int \psi\psi_{xx}\, dx = [\psi\psi_x]_0^{2\pi} - \int \psi_x^2 dx$ and since we assume the stream-function is periodic the first term vanishes and thus summing each gives us:

$$-\frac{1}{2}\iint \omega\psi \, d\underline{x} = \frac{1}{2}\int u^2 + v^2 \, d\underline{x} = E. \tag{5}$$

Therefore we are able to use the left hand side of this equation to calculate energy in our code A.4.

4. Distance

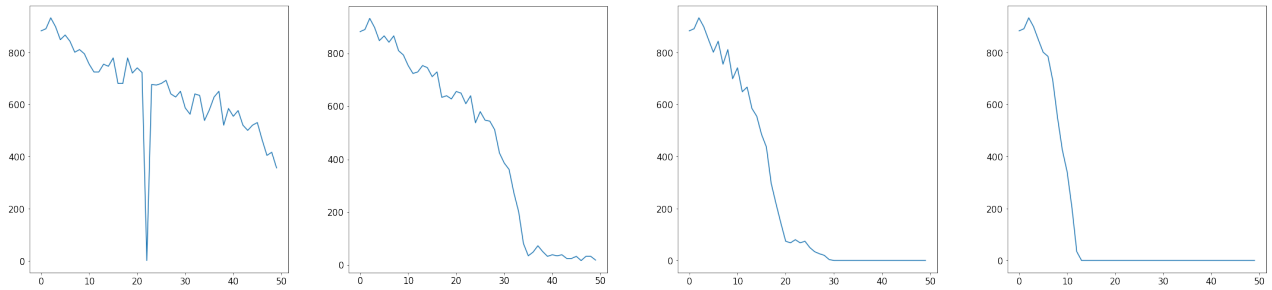
We are also interested in how rapidly two same sign vortices merge depending on the viscosity parameters. To investigate this we need a way of measuring the distance between the vortices. We achive this by first finding the pixels of maximum curl and then assuming this to be the centre of the vortices by using the numpy `isclose` function we select pixels that are within a small ϵ value of the maximum. Then we find the euclidean distance between these pixels A.3. In essence this is a measure of the distance between the centres of the vortices however, one could argue that it would be more appropriate to take into account the radii of the vortices. Despite this we are still able to explore how this distance changes over time for differing levels of viscosity.

5. Results

We observe the merging of two same sign vortices with various viscosity levels. Both the change in distance as well as the change in energy are of interest. In the code we used viscosity parameters $10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}$. As we can see in figure 1 greater levels of viscosity results in the vortices merging more rapidly than when viscosity is low. In plot 1a we can see that the vortices do not merge by the end of the simulated time-frame wheres in the 10^{-4} case the vortices have merged by the 35th timestamp. As seen in figure 1 for the 10^{-3} and 10^{-2} cases the vortices merge by the 25th and 15th timestamps respectively. We also studied the energy decay on these systems. In figure 2 we can see that for low viscosity levels the energy is periodic and as the viscosity levels increase the energy decreases more rapidly and exhibits damped periodic motion. In plots 2a and 2b the energy is periodic throughout with the amplitude decreasing greatly for 2b after the 35th timestamp. In plot 2c the energy curve is initially periodic before becoming linear with negative gradient after the timestamps 25 and in plot 2d the energy decays linearly from the beginning of the simulation.

6. Discussion and conclusions

In conclusion, our investigation into the interaction of vortices with different viscosity levels has shown that greater viscosity results in a greater rate of energy decay and greater rate of merging of same sign vortices. We observe a point at which the energy would go from acting periodically to becoming negative linear. This point approximately lines up with when the vortices merge however further simulations should be carried out before we can say this conclusively. We were able to use computational methods to simulate the flows however this is not a perfect



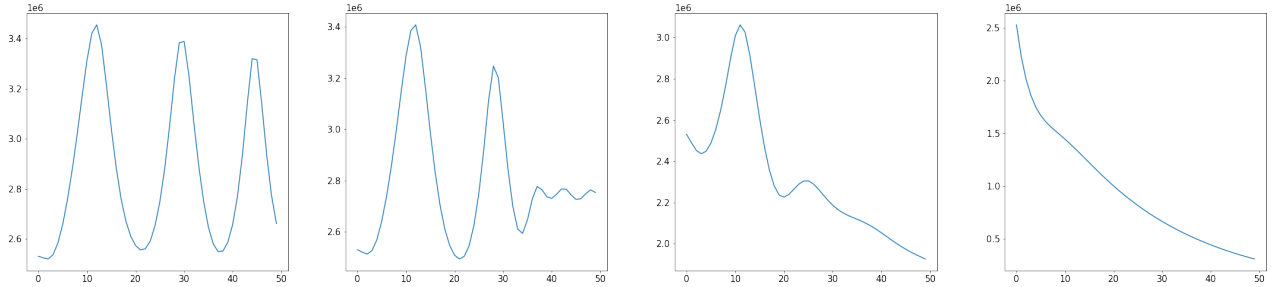
(a) Distance between vortices with viscosity 0.00001

(b) Distance between vortices with viscosity 0.0001

(c) Distance between vortices with viscosity 0.001

(d) Distance between vortices with viscosity 0.01

Figure 1



(a) Energy over time with viscosity 0.00001

(b) Energy over time with viscosity 0.0001

(c) Energy over time with viscosity 0.001

(d) Energy over time with viscosity 0.01

Figure 2

approach. First of all we must assume that our mathematical models are correct and there are no significant factors unaccounted for. Additionally our simulations were discrete both spatially and in the time dimension, this means that the simulation is not entirely smooth which may introduce artifacts that are not found in the real world flow. To account for these factors we could use a smaller timestamp as well as using a larger grid of points however this would be computationally prohibitive. Additionally as discussed there are two coefficients for viscosity however we only varied one, later study could investigate the two dimensional space of possible viscosity values. This study contributes to our understanding of fluid dynamics and has potential implications for a range of fields, from meteorology to engineering.

References

[1] Keith R. Symon. *Mechanics - Keith R Symon - 3rd Ed — PDF*. Scribd, 1971. URL: `https://www.scribd.com/doc/138003274/Mechanics-Keith-r-Symon-3rd-Ed` (visited on 03/17/2023).

[2] Jacques Vanneste. *Maths in Action B notes*. 2023.

Appendix A. Codes

A.1. **Code for solving Naiver-Stokes equations.**

```
import numpy as np
import matplotlib.pyplot as plt
```

```
from matplotlib.animation import FuncAnimation
from scipy.fft import rfft2, irfft2

def NavSto2D_solve(z_init, nu, Nx, Tend, dt, Nout):
    '''
    This function solves the incompressible viscous 2D Navier Stokes equations
    using pseudo-spectral method on doubly period domain
    === INPUTS ===
    z_init : initial condition for vorticity field
    nu : viscosit coefficient
    Nx : number of grid points in x (and y) direction(s). Total number of points Nx*Nx
    Tend : the length of time for which the simulation is performed
    dt : time step (if the solution blows up make this smaller)
    Nout : number of times the vorticity field is returned as an output
            These outputs are equally spaced in time from time 0 to Tend

    === OUTPUTS ===
    zeta(Nx,Nx,Nout): vorticity field at different time derived from
    solving the PDE (Navier Stokes)
    '''
    Nt = int(np.floor(Tend/dt))
    # number of steps in time
    Nevery = int(np.ceil(Nt/Nout))
    # every Nevery step the vorticity field is save for output
    Nout = int(Nt/Nevery)
    # ++++++++++ forming spectral coordinates ++++++++++
    Kx = np. zeros (Nx)
    Kx [: int(Nx /2)] = np. arange ( int(Nx /2))
    Kx[ int(Nx /2) :] = np. arange (- int(Nx /2) ,0)
    Ky = np. zeros (Nx)
    Ky [: int(Nx /2)] = np. arange ( int(Nx /2))
    Ky[ int(Nx /2) :] = np. arange (- int(Nx /2) ,0)
    Kxx , Kyy = np. meshgrid (Kx ,Ky)
    Kxx = Kxx [: ,: Nx //2+1]
    Kyy = Kyy [: ,: Nx //2+1]
    # ++++++++++ calculating operators used in time loop ++++++++++
    # this helps the code to be faster
    # and avoid unncessary computation inside the time loop
    # >>> calculating spectral operators <<<
    k2poisson = Kxx**2+Kyy**2
    k2poisson[0, 0] = 1
    ikx_k2 = 1j*Kxx / k2poisson
    iky_k2 = 1j*Kyy / k2poisson
    # de-aliasing mask: forces the nonlinear terms for kx,ky>2/3 to be zero
    # depending on the problem and the type of dissipation can be relaxed ...
    L = np.ones(np.shape(k2poisson))
    for i in range(np.shape(k2poisson)[1]):
```

```
        for j in range(np.shape(k2poisson)[0]):
            if (abs(Kxx[j, i]) > max(Kx)*8./9.):
                L[j, i] = 0
            elif (abs(Kyy[j, i]) > max(Ky)*8./9.):
                L[j, i] = 0

    zk = rfft2(z_init)
    c_visc = np.exp(-nu*dt*(Kxx**2+Kyy**2))
    def RHS(zk):
        '''
        calculate the RHS of momentum eq written for vorticity
        (RHS: everything but time derivative)
        '''
        ur = irfft2(iky_k2*zk)
        z_xr = irfft2(1j*Kxx*zk)
        vr = np.real(irfft2(-ikx_k2*zk))
        z_yr = irfft2(1j*Kyy*zk)
        nlr = ur*z_xr+vr*z_yr
        return -L*rfft2(nlr)

    iTsave = 0
    zeta = np.zeros((Nx,Nx,Nout+1))
    times = np.zeros(Nout+1)
    energy = np.zeros(Nout+1)
    zk[0,0]=0
    for iTime in range(0, Nt):
        if (iTime%Nevery==0):
            zeta[:,:,iTsave] = irfft2(zk)
            times[iTsave] = iTime * dt
            energy[iTsave] = np.sum(np.abs(zk)**2/k2poisson)
            iTsave += 1
        t = iTime * dt
        KK1z = RHS(zk)
        KK2z = RHS(zk + KK1z*(dt/2))
        KK3z = RHS(zk + KK2z*(dt/2))
        KK4z = RHS(zk + KK3z*dt)
        zk = c_visc*(zk + (dt/6)*(KK1z + 2*KK2z + 2*KK3z + KK4z))
        zk[0,0]=0
    return zeta, times, energy
```

A.2. Code for simulating system of two same sign vortices.

```
# ++++++++++ Setting up the flow and simulation parameters ++++++++++
nu = 0.1             # Viscosity
Nx = 128 # 96                # number of nodes in one dimension
Tend = 100              # end time (integration length)
dt = 0.1                # time step
Nout = 50               # the number of times the output is returned
```

```
# ++++++++++ forming physical (spatial) coordinates ++++++++++
dx = 2*np.pi/Nx                      # spatial grid spacing
dy = 2*np.pi/Nx                      #
x_vec = (np.arange(0, Nx)+1)*dx      # vector of spatial grid points in x
y_vec = (np.arange(0, Nx)+1)*dy      #
xx, yy = np.meshgrid(x_vec, y_vec)  # meshgrid of spatial grid points



# ++++++++++ initial condition (vorticity) ++++++++++
z_init = 2*(np.exp(-((xx-np.pi)**2+(yy-np.pi+np.pi/3)**2)*5) +
        +np.exp(-((xx-np.pi)**2+(yy-np.pi-np.pi/3)**2)*5))



# ++++++++++ Solving the Navier Stokes Equations ++++++++++
zeta, times, energy = NavSto2D_solve(z_init, nu, Nx, Tend, dt, Nout)



# ++++++++++ plotting the results ++++++++++
plt.rcParams["figure.figsize"] = [7, 7]
plt.rcParams["figure.autolayout"] = True


fig4, ax4 = plt.subplots()

cax = ax4.pcolormesh(zeta[:-1, :-1, 0], vmin=-.8, vmax=.8, cmap='RdBu_r')
ax4.set_aspect('equal', 'box')
fig4.colorbar(cax)
plt.close(fig4)

def animate(i):
   cax.set_array(zeta[:-1, :-1, i].flatten())

anim = FuncAnimation(fig4, animate, frames=np.shape(zeta)[2],
interval=120, repeat = False)

from IPython.display import HTML
HTML(anim.to_jshtml())
```

A.3. **Code for plotting distance between vortices.**

```
n = 50
dists = []
for i in range(n):
    arr = np.isclose(zeta[:,:,i], np.amax(zeta[:,:,i]), rtol=1e-05,
    atol=1e-08, equal_nan=False)
```

```
    if arr.sum() > 1:
        v0, v1 = arr.nonzero()
        dists.append(vort_dist(v0[0], v1[0], v0[1], v1[1]))
    else:
        dists.append(0)


x = np.arange(0,n)

plt.plot(x, dists)
plt.show()
```

A.4. **Code for plotting energy decay.**

```
x = np.arange(50)
plt.plot(x, energy[:-1])
```