Modeling 2D turbulence

Balachandra Suri, Brian D. Storey

(Dated: June 30, 2013)

I. INTRODUCTION

In this document we introduce, rather very briefly, the theory associated with the experiment demonstating fluid flows in shallow electrolytic layers. The idea is to use computation as a tool that mimics the experiment qualitatively, rather than delving too deep into the mathematical aspects of the problem. In these notes we briefly provide an introduction to the equations governing the evolution of a fluid flow - the Navier Stokes equations. Then, we show how in a shallow layer of electrolyte the flow could be described by a two-dimensional (2D) vorticity equation. We then describe the integration scheme associated with the 2D equation. Finally, we present a 'black-box' approach to using the MATLAB simulation.

II. NAVIER STOKES EQUATIONS

We will consider the problem of fluid flows where the flow is governed by the incompressible constant density Navier Stokes equations, which represent conservation of mass,

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

and momentum,

$$\frac{\partial \mathbf{U}}{\partial t} + \mathbf{U} \cdot \nabla \mathbf{U} = -\frac{1}{\rho}\nabla P + \nu \nabla^2 \mathbf{U} + \mathbf{F}. \tag{2}$$

Here $\mathbf{U} = [u, v, w]$ is the velocity vector, ρ is the mass density, P is the pressure, ν is the kinematic viscosity, and $\mathbf{F}$ is a forcing term which can be a function of space and time. The equations have four unknowns, the three components of the velocity and the pressure.

For flows in shallow electrolye layers, the component of the velocity normal to the layer (w) is negligible. In such cases the flow is described accurately enough by using only two components $\mathbf{u} = [u, v]$ of the velocity. However, the in-plane velocity, $\mathbf{u}(x, y, z)$, of the fluid varies within the electrolye layer, being maximum at the free surface ($z = h$) and zero at the bottom ($z = 0$). The effect of the solid bottom is incorporated by adding a linear friction term to the Navier-Stokes equations, thus eliminating the z dependence of the flow field.

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\frac{1}{\rho}\nabla P + -\alpha \mathbf{u} + \nu \nabla^2 \mathbf{u} + \mathbf{F}. \tag{3}$$

In 2D, the incompressibility is taken care of by the streamfunction (Ψ) formalism. The two components of velocity are related to a scalar function (Ψ) defined in the plane of the fluid as

$$u = \frac{\partial \Psi}{\partial y}; \quad v = -\frac{\partial \Psi}{\partial x}. \tag{4}$$

It is usually more convenient to use the vorticity-streamfunction form of the Navier-Stokes equations which is obtained by taking the curl of the momentum equation,

$$\frac{\partial \omega}{\partial t} + \mathbf{u} \cdot \nabla \omega = \nu \nabla^2 \omega - \alpha \omega + f, \tag{5}$$

where ω is the vorticity ($\omega = \nabla \times \mathbf{u}$) in the z direction and $f = \nabla \times \mathbf{F}$. Physically, vorticity is related to the local amount of solid body rotation of the fluid. The vorticity is related to the streamfunction, Ψ, through,

$$\omega = -\nabla^2 \Psi, \tag{6}$$

All of the above relationships can be found in more detail in any fluid dynamics textbook and we can discuss their origin in more detail as well. To completely define the evolution of a fluid flow, we have to specify the velocity field at the initial instant of time as well as boundary conditions that constrain the flow for all given times. In this article will consider the flow to be periodic in both directions.

A. Forcing

The forcing term on the R.H.S of the vorticity equation is the one which describes the specific problem at hand. In this article we discuss two most common forms of forcing that have been realised in experiments extensively. The first of these is the chessboard pattern which is experimentally reaslied - as in the demosntration - using small magnets arranged in the form of a chessboard like structure.

$$f = A \sin \kappa x \cos \kappa y \tag{7}$$

The other form of forcing is the sinusoidal pattern - which results in continuous bands of alternating forcing

$$f = A \sin \kappa y \tag{8}$$

For forcing with simple forms like the ones above, when the driving is very weak - this corresponds to a very small forcing amplitude - the flow has the same functional form as the forcing, often refered to as laminar flow.

III. NONLINEAR SIMULATION

In this section we will build up the 2D non-linear simulation of turbulent flow. Here we will use a technique based on spectral methods, where we will expand each variable in its Fourier series. There are, of course, numerous other ways to perform these simulations (finite elements, finite difference, finite volume) which we will not cover. Spectral methods are especially useful in simple geometries and are very easy to implement in problems with periodic boundary conditions. With periodic boundary conditions the use of a Fourier series means the boundary conditions are naturally satisfied and do not need to be enforced in some explicit way.

In physical space, each variable will be defined as a $N \times M$ matrix of values on equally spaced grid points in the $x - y$ plane. Our domain will be $0 \leq x < L_x$ and $0 \leq y < L_y$. We can take the 2D Fourier transform of each field such that,

$$\omega(x, y, t) = \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} \sum_{l=-\frac{N}{2}}^{\frac{N}{2}-1} \hat{\omega}_{k,l}(t) \mathrm{e}^{ikx} \mathrm{e}^{ily} \tag{9}$$

where k and l are the integer wave numbers and $\hat{\omega}_{k,l}$ are the Fourier coefficients. Here we assume that N, M are even. The Fourier coefficients comprise a 2D matrix with components $\hat{\omega}_{k,l}$. The Fourier coefficients can be found numerically with the use of the Fast Fourier Transform (FFT); in MATLAB we use the function `fft2` to compute the 2D FFT.

In Fourier space, it is trivial to connect our different flow variables together. If the instantaneous vorticity is known, the other variables can be found as,

$$\hat{\Psi}_{k,l} = \frac{\hat{\omega}_{k,l}}{k^2 + l^2}, \tag{10}$$

$$\hat{u}_{k,l} = il\hat{\Psi}_{k,l}, \tag{11}$$

$$\hat{v}_{k,l} = -ik\hat{\Psi}_{k,l}. \tag{12}$$

Notice that the streamfunction is undefined when $k = l = 0$ which represents the fact that the streamfunction is defined up to an arbitrary constant. Since our flow has no mean component, we can set $\hat{\Psi}_{0,0} = 0$. Also note that the vorticity gradient needed for the non-linear terms can be calculated following the way the velocity is computed from the streamfunction.

A. Implementation: solving the linear part

Let us start by implementing the Navier-Stokes equations, piece by piece to make sure each step is working. To begin, consider the simple linear part of the unforced equation,

$$\frac{\partial \omega}{\partial t} = \frac{1}{\mathrm{Re}}(\nabla^2 - \beta)\omega. \tag{13}$$

Substituting the Fourier transform into the linear equation yields,

$$\frac{d\hat{\omega}_{k,l}}{dt} = -\frac{1}{\text{Re}}\left(k^2 + l^2 + \beta\right)\hat{\omega}_{k,l}. \tag{14}$$

Since the equation is linear, the Fourier coefficients fully decouple from each other and the above equation represents an ordinary differential equation (ODE) for each Fourier mode. Since our ultimate goal is to solve the non-linear equation where we cannot solve much analytically, we will proceed directly to considering numerical integration in time.

Let's try integrating this equation using a few different methods. We will use the superscript n to denote the time step. The simplest method, forward Euler, would be written as,

$$\frac{\hat{\omega}^{n+1}_{k,l} - \hat{\omega}^{n}_{k,l}}{\Delta t} = -\frac{1}{\text{Re}}\left(k^2 + l^2 + \beta\right)\hat{\omega}^{n}_{k,l},$$

which can re-arranged as

$$\hat{\omega}^{n+1}_{k,l} = \left(1 - \frac{\Delta t}{\text{Re}}\left(k^2 + l^2 + \beta\right)\right)\hat{\omega}^{n}_{k,l}. \tag{15}$$

It is called the forward Euler method since the right hand side of the ODE is evaluated based on the known values of $\hat{\omega}_{k,l}$ at the current time; everything marches forward in time based on the current state. Equation 15 is an iteration equation where we simply update the new values of the Fourier coefficients. While a very simple scheme, problems with numerical stability can occur if the time step Δt is too large such that $\frac{\Delta t}{\text{Re}}\left(k^2 + l^2 + \beta\right) > 1$. If that condition is true, it is easy to see with each time step the Fourier coefficient changes sign, which is clearly un-physical for a diffusion problem. Further if, $\frac{\Delta t}{\text{Re}}\left(k^2 + l^2 + \beta\right) > 2$, the amplitude of the coefficients grow with time. Since k and l can be large, stability is a major problem with this method.

The backward Euler method would be written as,

$$\frac{\hat{\omega}^{n+1}_{k,l} - \hat{\omega}^{n}_{k,l}}{\Delta t} = -\frac{1}{\text{Re}}\left(k^2 + l^2 + \beta\right)\hat{\omega}^{n+1}_{k,l},$$

which can written as

$$\hat{\omega}^{n+1}_{k,l} = \frac{\hat{\omega}^{n}_{k,l}}{1 + \frac{\Delta t}{\text{Re}}\left(k^2 + l^2 + \beta\right)}. \tag{16}$$

It is called the backward Euler method since the right hand side of the ODE is evaluated at the future time, $n + 1$. This method is stable for all time step sizes and it is easy to see with each iteration the Fourier coefficient is reduced. Stability and accuracy are different issues. While the backward Euler scheme is stable for any time step size the accuracy depends linearly on the step size.

The Forward Euler method is known as an explicit method since the evaluation of the right hand side is explicitly evaluated with known information. The Backward Euler method is known as an implicit since the right hand side is implicitly known, or must be inverted.

Both the forward and backward Euler methods have an error proportional to the time step. Other methods are higher order accurate in time, such as the Crank-Nicholson method which evaluates the right hand side at the average of the present and future time step. The Crank-Nicholson method is written as,

$$\frac{\hat{\omega}^{n+1}_{k,l} - \hat{\omega}^{n}_{k,l}}{\Delta t} = -\frac{1}{\text{Re}}\left(k^2 + l^2 + \beta\right)\frac{\left(\hat{\omega}^{n+1}_{k,l} + \hat{\omega}^{n}_{k,l}\right)}{2},$$

which can written as

$$\hat{\omega}^{n+1}_{k,l} = \frac{1 - \frac{\Delta t}{2\text{Re}}\left(k^2 + l^2 + \beta\right)}{1 + \frac{\Delta t}{2\text{Re}}\left(k^2 + l^2 + \beta\right)}\hat{\omega}^{n}_{k,l}. \tag{17}$$

For our purposes, the Crank-Nicholson method will work well.

B. Implementation: add in the non-linear part

Now that we can solve the linear parts of the governing equation, let's add in the non-linear terms. Taking the Fourier transform of the governing equation yields,

$$\frac{d\hat{\omega}_{k,l}}{dt} = -(\widehat{\mathbf{u}\cdot\nabla\omega})_{k,l} - \frac{1}{\mathrm{Re}}\left(\left(k^2+l^2+\beta\right)\hat{\omega}_{k,l}\right) + \hat{f}_{k,l}, \tag{18}$$

where $\hat{f}$ is the Fourier transform of $f = \frac{1}{\mathrm{Re}}n(\beta+n^2)\sin(ny)$. Everything except the non-linear term was dealt with in the previous section.

To compute the Fourier coefficients of the nonlinear term we use the pseudospectral approximation. The "recipe" for calculating $(\widehat{\mathbf{u}\cdot\nabla\omega})_{k,l}$ is;

- Convert the physical space vorticity field (a 2D matrix with values of vorticity on an equally spaced grid) to it's Fourier coefficients using MATLAB's `fft2` command.
- Compute the streamfunction using the expression $\hat{\psi}_{k,l} = \hat{\omega}_{k,l}/(k^2+l^2)$.
- Compute the velocity using the expressions, $\hat{u}_{k,l} = il\hat{\psi}_{k,l}$ and $\hat{v}_{k,l} = -ik\hat{\psi}_{k,l}$
- Compute the voriticity gradient using the expressions, $\widehat{\frac{\partial\omega}{\partial x}}\Big|_{k,l} = ik\hat{\omega}_{k,l}$ and $\widehat{\frac{\partial\omega}{\partial y}}\Big|_{k,l} = il\hat{\omega}_{k,l}$
- Convert the velocity and vorticity gradients to physical space using MATLAB's `ifft2` command.
- Compute the nonlinear term, $u\frac{\partial\omega}{\partial x} + v\frac{\partial\omega}{\partial y}$ using MATLAB's element by element matrix multiply operator; i.e. `u.*wx + v.*wy`.
- Convert the entire non-linear term back to Fourier space using the `fft2` command.

The method involves taking all the derivatives in Fourier space, converting the results to physical space, taking the non-linear products in physical space, and converting the result back to Fourier space. The error in this approximation only occurs because when we multiply two things, we have a high frequency component which goes beyond our resolution. If the flow is well-resolved (enough points for the range of features you are trying to resolve), this error can be kept to a minimum.

There a number of time stepping schemes useful for integrating the equations in time. The simplest methods to implement that have good stability properties are explicit with respect to the non-linear term (i.e. the evaluation of the terms on the right hand side of the equation is based on the current state and no inversion must be used) and use implicit for the linear terms (i.e. the evaluation of the terms on the right hand side depends upon the future state).

An example method we will use is an Adams-Bashforth/Crank-Nicholson scheme.

$$\frac{\hat{\omega}^{n+1}_{k,l} - \hat{\omega}^{n}_{k,l}}{\Delta t} = -\frac{1}{2}\left(3(\widehat{\mathbf{u}\cdot\nabla\omega})^{n}_{k,l} - (\widehat{\mathbf{u}\cdot\nabla\omega})^{n-1}_{k,l}\right) - \frac{1}{\mathrm{Re}}\left(k^2+l^2+\beta\right)\frac{\hat{\omega}^{n+1}_{k,l}+\hat{\omega}^{n}_{k,l}}{2} + \frac{\hat{f}^{n+1}_{k,l}+\hat{f}^{n}_{k,l}}{2}.$$

Since we consider the case where the forcing is steady in time, we do not need to worry about the time step of evaluation for the forcing.

$$\hat{\omega}^{n+1}_{k,l} = \frac{-\frac{\Delta t}{2}\left(3(\widehat{\mathbf{u}\cdot\nabla\omega})^{n}_{k,l} - (\widehat{\mathbf{u}\cdot\nabla\omega})^{n-1}_{k,l}\right) + \left(1-\frac{\Delta t}{2\mathrm{Re}}\left(k^2+l^2+\beta\right)\right)\hat{\omega}^{n}_{k,l} + \Delta t\hat{f}_{k,l}}{1+\frac{\Delta t}{2\mathrm{Re}}\left(k^2+l^2+\beta\right)}. \tag{19}$$

Every term on the right side of the equation is known at each time step and thus this equation can be iterated forward in time.

IV. MATLAB-PACKAGE

The matlab package has three functions - each for very specfic tasks. The first of these functions is the **'define_parameters'** function. This functions defines the simulation parameters. The most primary of these are the size of the magnets (assumed to be squares) ($L0$), the viscosity of the fluid (ν) and the friction coefficient (α). Since for a given fluid layer depth we have a constant friction coefficient, we define it within the file rather than as an argument to the function. Instead, we need to define how many magnets we use in simulation - whether it is a 4 X 4 or a 5 X 5 array. Interestingly, one cannot use an odd number of magnets when using periodic boundary conditions, as it would essentially imply a magnet of twice the intended size at the ends. Also, one has to define the density of the grid - per inch. Hence, the arguments that go into the function are

```
define_parameters(2.54/100, 4, 4, 32);
```

The next of these functions - '**flow_evolution**' -is the one that evolves the flow starting 'randomised initial conditions'. Since we use a 'black-box' appraoch, we are concerned with only three parameters during the flow evolution. The first of these is the amplitude of forcing, the second is the time of integration and the last one is the time step of integration. Hence, the typical inputs for this function are

```
flow_evolution(0.1, 240, 0.02);
```

The last of the scripts - **plot_omega** - takes the file name containing the vorticity field as an input and plots the vorticity field every so often as indicated by the pause_for argument. So, a typical command-line input to run this function would be

```
plot_omega('filename.mat', 0.1);
```