FOURIER-SPECTRAL METHODS FOR NAVIER STOKES EQUATIONS IN 2D

MASHBAT SUZUKI

Abstract. We implement Fourier-Spectral method for Navier-Stokes Equations on two dimensional flat torus with Crank-Nicolson method for time stepping. We use the vorticity stream formulation for implementation and get back velocity and pressure from the stream function. We use our implementation to better understand the dependency on initial condition by Navier-Stokes Equations adding small random perturbation and see the difference in evolution as well as evolution of uniform random initial data.

1. Introduction

The incompressible Navier-Stokes equation in the traditional form solving for velocity is following

$$\partial_t u + u \cdot \nabla u + \nabla p = \mu \Delta u \tag{1.1}$$

$$\nabla \cdot u = 0 \tag{1.2}$$

where μ viscosity. We derive vorticity stream function formulation of Navier-Stokes equation in two and three dimensions by applying curl to the Navier-Stokes equation. The following is a common way of deriving vorticity equation. First note that vorticity is defined as $w = \nabla \times u$, observe the following identity

$$\frac{1}{2}\nabla(u \cdot u) = (u \cdot \nabla)u + u \times (\nabla \times u)$$

Date: December 17, 2014.
Key words and phrases. Fourier Spectral Method, Navier-Stokes Equation .

also note that for scalar function ϕ, that $\nabla \times \nabla\phi = 0$. Observe

$$\nabla \times (u \times w) = (w \cdot \nabla)u - (u \cdot \nabla)w + u\nabla \cdot w - w\nabla \cdot u$$

where the last two terms vanish since $\nabla \cdot w = 0$ and $\nabla \cdot u = 0$. Using above identities we can transform 1.1 to

$$\partial_t u + \frac{1}{2}\nabla(u \cdot u) - u \times (\nabla \times u) + \nabla p = \mu\Delta u$$

We can take the curl on both sides of the equation. Using the identities previously mentioned we get the **Vorticity Equation**

$$\partial_t w + (u \cdot \nabla)w - (w \cdot \nabla)u = \mu\Delta w$$

It is often denoted $\frac{Dw}{Dt} = \partial_t + (u \cdot \nabla)w$. The Vorticity equation shows that the rate of change of the vorticity is controlled by the term reffered as vorticity stretching term $(w \cdot \nabla)u$ and by diffusion term $\mu\Delta w$. Note that in two dimensions $u = v_1 e_x + v_2 e_y$ and $w = w(x, y)e_z$ and thus $(w \cdot \nabla)u = 0$. This gives us **Two dimensional Vorticity Equation**

$$\partial_t w = \mu\Delta w - (u \cdot \nabla)w$$

Above is the main equation we want to consider. This equation is a nonlinear advection diffusion equation. Once we can successfully solve for vorticity we solve for stream function ψ defined as

$$w = -\Delta\psi$$

and recover the velocity $u = v_1 e_x + v_2 e_y$ from the stream function as $v_1 = \partial_y \psi$ and$v_2 = -\partial_x \psi$.

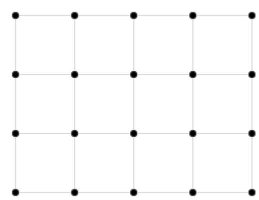
Scaling arguments show that in the limit of very high viscosity or zero Reynolds number the streamfunction essentially reduces to biharmonic equation of the following form

$$\Delta^2\psi = 0$$

In this paper we will focus mainly on two dimensional vorticity equation on $\mathbb{T}^2$. The vorticity streamfunction formulation is easier to implement than more primitive variable formulation velocity.

2. Discretization and Implementation

We discretize both space and time. For space discretization of $\mathbb{T}^2$ we use equidistant square grid identifying both top and bottom and right and left sides.



We use fourier-spectral method for differentiation. So we take fourier transform of the vorticity equation which gives

$$\partial_t \hat{w} = -\mu(\xi_x^2 + \xi_y^2)\hat{w} - \widehat{u \cdot \nabla w} \tag{2.1}$$

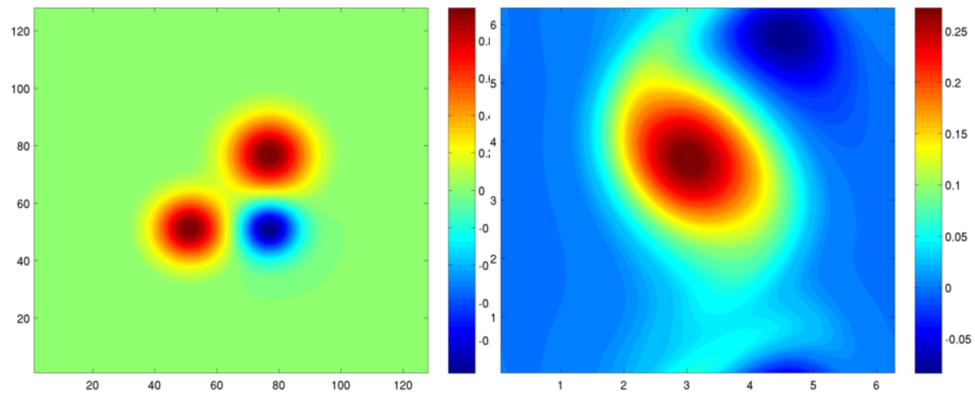
$$\partial_t \hat{w} = -\mu(\xi_x^2 + \xi_y^2)\hat{w} - \hat{v}_1 * \xi_x \hat{w} - \hat{v}_2 * \xi_y \hat{w} \tag{2.2}$$

The right hand side of the equation (2.2) can be solved using discrite fourier transform on the grid points. We use FFT algorithm to solve the right hand side.

For time stepping we use the Crank-Nicolson method. For linear evolution PDE's this method unconditionally stable hence also thought to be good method for some non-linear PDE's. Crank-Nicolson method is an average of Forward Euler and Backward Euler methods after long algebra one can write the method in the explicit form

$$\hat{w}_{i,j}^{n+1} = \frac{1}{\frac{1}{\Delta t} - \frac{1}{2}\mu(\xi_x^2 + \xi_y^2)} \left(\left(\frac{1}{\Delta t} + \frac{1}{2}\mu(\xi_x^2 + \xi_y^2) \right) \hat{w}_{i,j}^n - \widehat{u_{i,j}^n \cdot \nabla w_{i,j}^n} \right)$$

We also have to take care of the aliasing problem by trowing out the frequencies that are higher than 2/3 times the grid size in the convolution. We



(A) Initial Vorticity with $\mu = 0.005$ (B) Final Vorticity with $\mu = 0.005$

FIGURE 1. Evolution of vorticity with smooth initial data

were able to successfully implement above method and obtain a physically feasible answer.

3. RESULTS AND OBSERVATIONS

We try our implementations with different initial conditions smooth, uniform random noise and combination of the two.

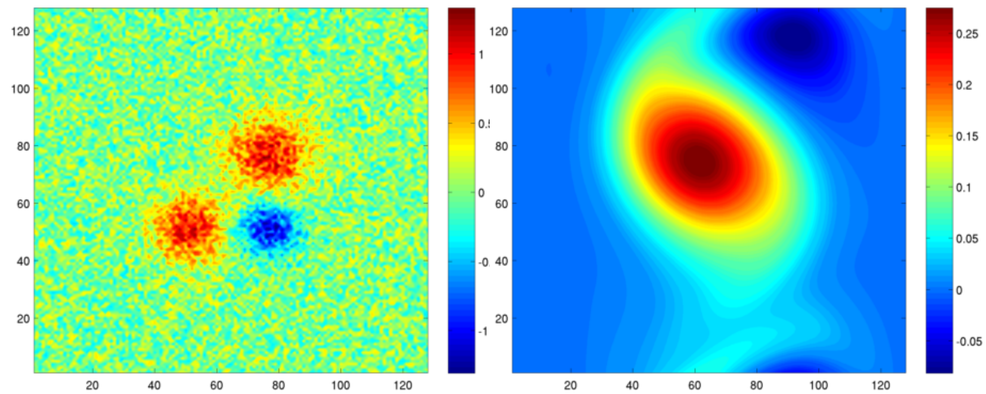
First we evolve smooth initial vorticity

$$\begin{aligned}\tilde{w}|_{t=0} =& \exp(-\frac{(x-\pi+\pi/5)^2+(y-\pi+\pi/5)^2}{0.3})- \\ &-\exp(-\frac{(x-\pi-\pi/5)^2+(y-\pi+\pi/5)^2}{0.2})+ \\ &+\exp(-\frac{(x-\pi-\pi/5)^2+(y-\pi-\pi/5)^2}{0.4})\end{aligned}$$

The solution is of the following initial condition is given by Figure 1. The figure shows evolution of vorticity field with parameters $\mu = 0.005$, $T = 50$ with $\Delta t = 0.1$.

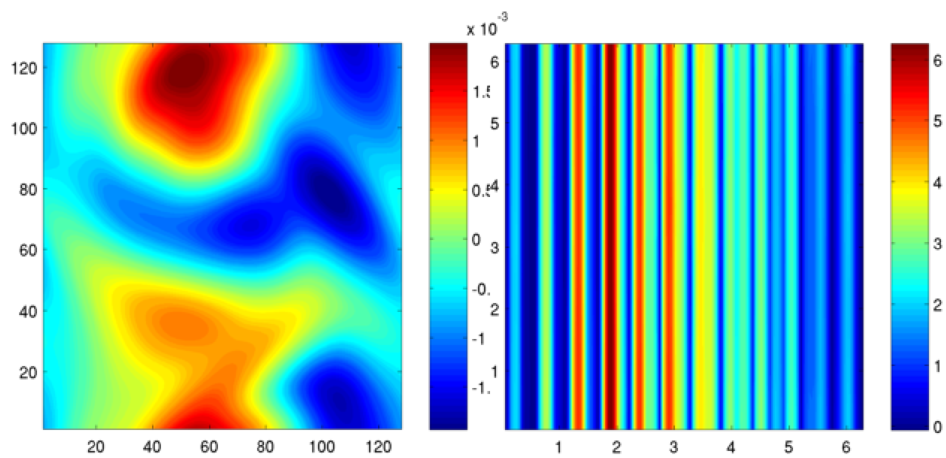
Next we add uniform random noise $N \sim Unif(-1, 1)$ to the smooth initial data $\tilde{w}|_{t=0}$ and see the evolution of vorticity field

$$w|_{t=0} = \tilde{w} + \epsilon N$$



(A) Initial Vorticity with $\mu = 0.005$ (B) Final Vorticity with $\mu = 0.005$

FIGURE 2. Evolution of vorticity with non-smooth initial data w with $\epsilon = 0.1$

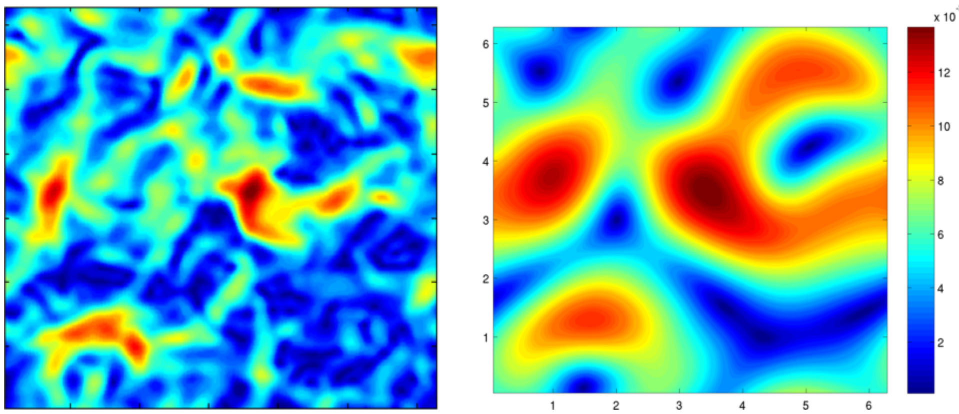


(A) $T = 50$ difference of vorticity in $w, \tilde{w}$ (B) $T = 50$ difference of speed in $w, \tilde{w}$

FIGURE 3. Evolution of vorticity with non-smooth initial data w

From figure 2 we see that it seems that the final evolution of w is not very different from that of $\tilde{w}$. Indeed if we plot the difference the two vorticities the value was bounded by $2 \cdot 10^{-3}$ as seen in figure3a at time $t = 50$ with viscosity $\mu = 0.005$.

An unexpected result had emerged when we plotted the difference between absolute value of velocity fields of initial vorticity field $w, \tilde{w}$ as shown in figure3b.It seems that although we added random perturbation somehow the difference in velocity field is structured along a line. We believe that

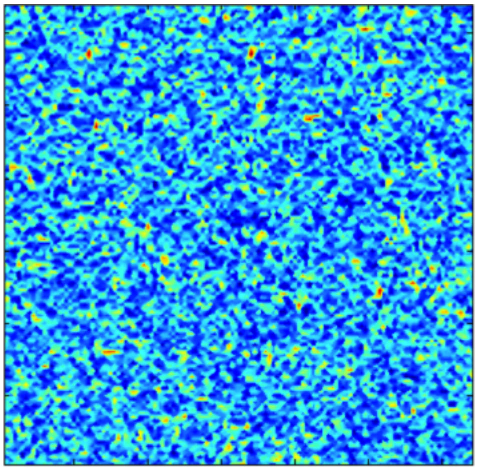


(A) Initial velocity distribution (B) Final velocity distribution

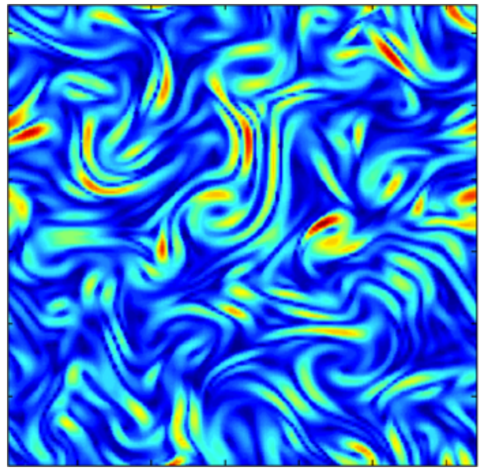
FIGURE 4. Evolution of velocity with random initial data N with $\mu = 0.005$

this is not physical rather result of our implementation. The main reason is that the absolute value of velocity field should not have a preferred direction since the perturbations are random however the figure3b has a preferred direction.

For completeness we also evolve random velocity field which is given by figure4. Also to check the limit our implementation we have evolve random uniform random velocity field with very low viscosity $\mu = 0.0001$. Our result is shown in figure5 which is what we expect qualitatively.



(A) Initial velocity distribution



(B) Final velocity distribution

FIGURE 5. Evolution of velocity with random initial data N with $\mu = 0.0001$

4. Appendix: MATLAB Codes

Algorithm1: Solve for vorticities at each time step, saves a frame and stores it so that it can be played back as a movie.

```
clear all
%Simulation Property Setting

GridSize=128;

Visc=0.005;

% Space Setting

h=2*pi/GridSize;

```

```
axis=h*[1:1:GridSize];

[x,y]=meshgrid(axis,axis);

% Time Setting

FinTime=50;

dt=0.1;

t=0;

% Movie File Data Allocation Set Up
FrameRate=10;
Mov(10)=struct('cdata',[],'colormap',[]);

k=0;
j=1;

% Defining Initial Vorticity Distribution

H=exp(-((x-pi+pi/5).^2+(y-pi+pi/5).^2)/(0.3))-exp(-((x-pi-pi/5).^2+(y-pi+pi/5).^2)/(0.2))+exp(-((x-pi-pi/5).^2+(y-pi-pi/5).^2)/(0.4));

% Adding Random Noise to Initial Vorticity
epsilon=0.3;
Noise=random('unif',-1,1,GridSize,GridSize);
```

```

% Note that for Low Viscosities Adding Noise to Non-Trivial Vorticity
% Distribution results in blow up, so either do pure noise or smooth data

w=H+epsilon*Noise;

w_hat=fft2(w);


%%%%%%%%     Method Begins Here     %%%%%%%%%

kx=1i*ones(1,GridSize)'*(mod((1:GridSize)-ceil(GridSize/2+1),GridSize)-floor(GridSize/2));
ky=1i*(mod((1:GridSize)'-ceil(GridSize/2+1),GridSize)-floor(GridSize/2))*ones(1,GridSize);
AliasCor=kx<2/3*GridSize&ky<2/3*GridSize;


Lap_hat=kx.^2+ky.^2;

ksqr=Lap_hat; ksqr(1,1)=1;


while t<FinTime

    psi_hat = -w_hat./ksqr;
```

```
    u   =real(ifft2( ky.*psi_hat));

    v   =real(ifft2(-kx.*psi_hat));

    w_x=real(ifft2( kx.*w_hat   ));

    w_y=real(ifft2( ky.*w_hat   ));

    VgradW      = u.*w_x + v.*w_y;
    VgradW_hat  = fft2(VgradW);

    VgradW_hat  = AliasCor.*VgradW_hat;

    %Crank-Nicholson Update Method

    w_hat_update= 1./(1/dt - 0.5*Visc*Lap_hat).*((1/dt
        +0.5*Visc*Lap_hat).*w_hat-VgradW_hat);

    if (k==FrameRate)

        w=real(ifft2(w_hat_update));

        %Vel=sqrt(u.^2+v.^2);    %This is for plotting
            velocity

        contourf(x,y,w,80);
        colorbar;
```

```
          shading flat; colormap('jet');
          drawnow
          Mov(j)=getframe;
          k=0;
          j=j+1
     end
     w_hat=w_hat_update;
     t=t+dt;
     k=k+1;
end
```

Algorithm2: Solves two equations at once one with different initial data and computers their solutions difference at each time step,

---

```
clear all
%Simulation Property Setting

GridSize=128;

Visc=0.005;

% Space Setting

h=2*pi/GridSize;

axis=h*[1:1:GridSize];

[x,y]=meshgrid(axis,axis);
```

```

% Time Setting

FinTime=80;

dt=0.1;

t=0;

% Movie File Data Allocation Set Up
FrameRate=10;
Mov(10)=struct('cdata',[],'colormap',[]);

k=0;
j=1;

% Defining Initial Vorticity Distribution
%[i,j]=meshgrid(1:GridSize,1:GridSize);

w=exp(-((x-pi+pi/5).^2+(y-pi+pi/5).^2)/(0.3))-exp(-((x
    -pi-pi/5).^2+(y-pi+pi/5).^2)/(0.2))+exp(-((x-pi-pi
    /5).^2+(y-pi-pi/5).^2)/(0.4));

% Adding Random Noise to Initial Vorticity
epsilon=0.1;

Noise=random('unif',-1,1,GridSize,GridSize);

```

```
sw=w+epsilon*Noise;


w_hat=fft2(w);
sw_hat=fft2(sw);

%%%%%%%%        Method  Begins  Here        %%%%%%%%%

kx=1i*ones(1,GridSize)'*(mod((1:GridSize)-ceil(GridSize/2+1),GridSize)-floor(GridSize/2));
ky=1i*(mod((1:GridSize)'-ceil(GridSize/2+1),GridSize)-floor(GridSize/2))*ones(1,GridSize);
AliasCor=kx<2/3*GridSize&ky<2/3*GridSize;


Lap_hat=kx.^2+ky.^2;

ksqr=Lap_hat;  ksqr(1,1)=1;


while t<FinTime

    psi_hat = -w_hat./ksqr;

    u   =real(ifft2( ky.*psi_hat));

    v   =real(ifft2(-kx.*psi_hat));

```

```
    w_x=real(ifft2( kx.*w_hat  ));

    w_y=real(ifft2( ky.*w_hat  ));

    VgradW     = u.*w_x + v.*w_y;
    VgradW_hat = fft2(VgradW);

    VgradW_hat = AliasCor.*VgradW_hat;



    spsi_hat = -sw_hat./ksqr;

    su  =real(ifft2( ky.*spsi_hat));

    sv  =real(ifft2(-kx.*spsi_hat));

    sw_x=real(ifft2( kx.*sw_hat  ));

    sw_y=real(ifft2( ky.*sw_hat  ));

    sVgradW     = su.*sw_x + sv.*sw_y;
    sVgradW_hat = fft2(sVgradW);

    sVgradW_hat = AliasCor.*sVgradW_hat;


    %Crank-Nicholson Update Method
```

```
    w_hat_update= 1./(1/dt - 0.5*Visc*Lap_hat).*((1/dt
        +0.5*Visc*Lap_hat).*w_hat-VgradW_hat);

    sw_hat_update= 1./(1/dt - 0.5*Visc*Lap_hat).*((1/
        dt +0.5*Visc*Lap_hat).*sw_hat-sVgradW_hat);

    if (k==FrameRate)

        w=real(ifft2(w_hat_update));

        sw=real(ifft2(sw_hat_update));

        %w=sqrt(u.^2+v.^2);     %This is for plotting
            velocity

        contourf(w-sw,80);

        colorbar;

        shading flat; colormap('jet');

        drawnow

        Mov(j)=getframe;
        k=0;
        j=j+1
    end
```

```
    w_hat=w_hat_update;

    sw_hat=sw_hat_update;

    t=t+dt;
    k=k+1;
end
```

References

Department of Mathematics, McGill University, Montreal, Canada

E-mail address: `mashbat.suzuki@mail.mcgill.ca`