PHYS 451: Computational Mechanics

Final Project Report:

Numerical Simulation of the Navier-Stokes Equations for Incompressible Viscous Fluid Flow in Two Dimensions

Yingsi Qin, Dejan Maksimovski

Background

Almost all macroscopic physical systems are "submerged" in a fluid. This means that any complete consideration of their mechanics involves solving for the effects of the motion of this surrounding fluid. In some cases, understanding the motion of the (surrounding) fluid is the actual goal of solving the mechanics of a system – eg. the flow of air around an aircraft wing, or analysis of a cars aerodynamic properties. By a relatively quick survey of fields in engineering, medicine and environmental studies, we can think of many more instances where the mechanical behaviour of a fluid is a major focus: atmospheric flows (models for weather prediction), models for global ocean currents, the circulation of blood, breathing, etc.

In all of the aforementioned examples, one needs to be able to simulate the equations theoretically describing the motion of the fluid. Because of this, we decided to embark on making a numerical model with which one can simulate the motion of a fluid, with the possibility of adding stationary obstructions in the fluid's path. Due to the character of most of the fluids we confront in our everyday life, it was decided that the model would correspond to incompressible, but viscous, fluids.

The System Modeled

The field of physics that deals with the dynamics of fluids, both classical and relativistic, is fluid dynamics. An analysis of the dynamics of incompressible viscous fluids done in this field, gives the following equations describing the fluids' motion:

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

$$\nabla \cdot \mathbf{v} = 0, \tag{2}$$

$$\mathbf{v}(\mathbf{r}, t = 0) = \mathbf{v}_0(\mathbf{r}), \tag{3}$$

where $\mathbf{v}(\mathbf{r}, t)$ is the velocity of the fluid, $p(\mathbf{r}, t)$ is the absolute pressure in the fluid, $\mathbf{f}(\mathbf{r}, t)$ is the force per unit mass of the liquid due to external forces, ρ is the fluid's density, and ν is the kinematic viscosity of the fluid. The first equation is also known as the Navier-Stokes Equation, and is basically Newton's second law for extremely small (practically point size), but still macroscopic segments of the fluid. The second, third and fourth terms on the right-hand-side correspond to the force per unit mass experienced by the small fluid segment due to viscosity, the gradient of the pressure in the fluid, and the external forces, respectively. The first term on the right-hand-side is the negative spatial part of the substantial derivative of the small segment's velocity. In other words this term gives the amount by which the segment's velocity changes per unit time due to it moving in the spatially varying vector field $\mathbf{v}$. With this term placed in the right-hand-side, the left-hand-side is simply the partial derivative of $\mathbf{v}$ with respect to time at some point in space determined by $\mathbf{r}$. The reason for why we have isolated this partial derivative is because in the numerical model we will be evaluating $\mathbf{v}$ for successive moment in time, only at a certain discrete collection of fixed

points in space. The second equation is the incompressibility condition, asserting that there is no net flux of fluid mass into any closed region in the fluid. The third equation is simply the initial condition for the fluid's velocity.

Numerical Methods and MATLAB Functions

For all points in our field, the Navier-Stokes equation (1) is solved from t to $t + \Delta t$ using Euler's method

$$\mathbf{v}(\mathbf{r}, t + \Delta t) \approx \mathbf{v}(\mathbf{r}, t) + \frac{\partial \mathbf{v}}{\partial t}\Big|_{\mathbf{r},t} \Delta t, \tag{4}$$

with the right-hand side evaluated at t. The evaluation of the right-hand side at each timestep involves **1)** calculating the the pressure $p(\mathbf{r}, t)$ on a two-dimensional spatial grid, and **2)** calculating the right-hand-side of (1) by using the knowledge of $\mathbf{v}(\mathbf{r}, t)$ and the "newly" found $p(\mathbf{r}, t)$.

In order to find the pressure $p(\mathbf{r}, t)$, we use the incompressibility condition (2). An expansion of the incompressibility condition imposed on $\mathbf{v}(\mathbf{r}, t + \Delta t)$, after using the Euler's method (4) from the previous time step, along with the assumption that $\mathbf{v}(\mathbf{r}, t)$ satisfies (2), gives

$$\nabla \cdot \frac{\partial \mathbf{v}}{\partial t}\Big|_{\mathbf{r},t} = \nabla \cdot \left(-(\mathbf{v} \cdot \nabla)\mathbf{v} + \nu \nabla^2 \mathbf{v} - \frac{1}{\rho}\nabla p + \mathbf{f} \right) \Big|_{\mathbf{r},t} = 0, \tag{5}$$

which is further reduced to

$$\nabla^2 p = \nabla \cdot \left(-(\mathbf{v} \cdot \nabla)\mathbf{v} + \nu \nabla^2 \mathbf{v} + \mathbf{f} \right) \Big|_{\mathbf{r},t}. \tag{6}$$

We can see that (6) is of the form of Possion's equation.

Thus, we calculate pressure $p(\mathbf{r}, t)$ at each time step using the method of relaxation by feeding

$\mathbf{v}(\mathbf{r}, t)$ from the previous timestep to our MATLAB function `p(ax, ay, bp, dL, deltap)`. It takes as inputs `ax` and `ay`, which are two-dimensional arrays for the x- and y-components of the divergence's argument on the right-hand-side of (6); `bp` is a two-dimensional matrix encoding the position of internal and boundary points with values of 0 and 1, respectively; `dL` is the calculation step size for spatial derivatives; and `deltap` is the tolerance value for pressure. The function returns a two-dimensional matrix giving the pressure at each point on the two-dimensional spatial domain, at the designated time step.

For each point in the field, all first and second order spatial partial derivatives are calculated by calling our MATLAB functions `d1(F, dL, bp, i, j, C)` and `d2(F, dL, bp, i, j, C)`, respectively. `F` is the differentiated function determined for all points in the spatial domain; `dL` is the calculation step size; `bp` is the aforementioned two-dimensional matrix for boundary points; `i` is the row index for the current point, and j is the column index. If `C` equals 1, we take the partial derivatives with respect to y, if it equals 2, we take the partial derivatives with respect to x.

The Navier-Stokes equation is solved by calling `NavierStokes2d(fx, fy, vx0, vy0, dL, dt, NL, Nt, bp, rho, nu, deltap)`. The function outputs three things: `vx` – a three-dimensional matrix for the x-component of velocity of the fluid with the first matrix dimension corresponding to the y-coordinate, the second one to the x coordinate, and the third dimension to t. `vy` is similarly a three-dimensional matrix for the y-component of the fluid's velocity. `P` is a similar three-dimensional matrix for the pressure. `fx` and `fy` are similar matrixes giving the x- and y-components of the external force per unit mass. `vx0` and `vy0` are two-dimensional matrices for the x- and y-components of the initial velocity of the fluid,

respectively. The rest of the inputs contain the aforementioned `dL` (the spatial grid spacing); `dt` – the time step; `NL` – the number of points in the field; `Nt` – the number of time steps; the aforementioned `bp`; `rho` – the density of the fluid; `nu` – the kinematic viscosity of the fluid; and `deltap` – the tolerance value when integrating pressure p. All variables are in SI units.

Initial Conditions

We have $\mathbf{v}(\mathbf{r}, t = 0) = \mathbf{v}_0(\mathbf{r})$ and $p(\mathbf{r}, 0) = p_0(\mathbf{v}_0(\mathbf{r}))$, and the aforementioned `bp`, as the initial conditions to solve for the Navier-Stokes equation. The initial conditions thus vary case by case. To place round-shape obstacles in the field, we call `circle(bp,r,N,dL,xa,ya)`, which adds a disk of boundary points (set to 1) to the input matrix `bp`. The function takes as inputs `bp` – the original boundary point matrix; `r` – the radius of the circle; `N` – the number of points in each direction of the (square) boundary point matrix; `dL` – the step size between points; and `xa` and `ya` – the horizontal and vertical translational distances of the center of the circle, respectively.

Results and Analysis

Simulation 1

In the first simulation, we set the top, bottom, and left boundaries at a fixed pressure of 500,000 Pa, and the right boundary at a fixed pressure of 900,000 Pa. For the initial velocity of the fluid, we had the center half of a thin band of fluid on the left moving in the x-direction (see Figure 2), with the speed ($v = v_x$) being modulated as the starting quarter wavelength of a sine function. In Figure 1, we have the initial pressure as function of space, which was

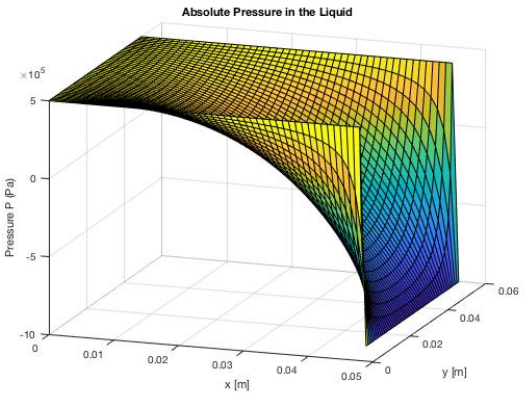
calculated using method of relaxation. From this initial pressure, we expect the fluid to accelerate and thus flow in the $+x$-direction.

Figure 3 shows the velocity of the fluid at the third timestep. We indeed see that the fluid has acquired a speed in the $+x$-direction, as dictated by the gradient of the initial pressure. We also see that the fluid has acquired speed in the y-direction near the top and bottom boundaries. This makes sense considering the sharp decrease in the pressure near their junction with the right boundary, pushing the fluid inward.

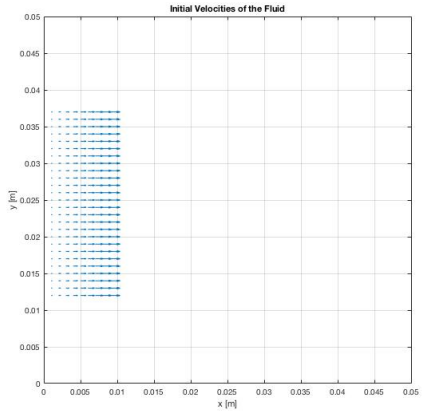
Simulation 2

In the second simulation, initial pressures were the same as in the first simulation except that we have a round-shape obstacle in the middle of the field where the pressure was fixed at 500,000 Pa, shown in Figure 4. For the initial velocity of the fluid, we had a thin band of fluid on the left moving in the x-direction (see Figure 5), with the speed ($v = v_x$) being modulated as the starting quarter wavelength of a sine function. From this initial pressure, we expect the fluid to flow in the $+x$-direction but around the obstacle with higher speed near areas of sharper change in pressure.

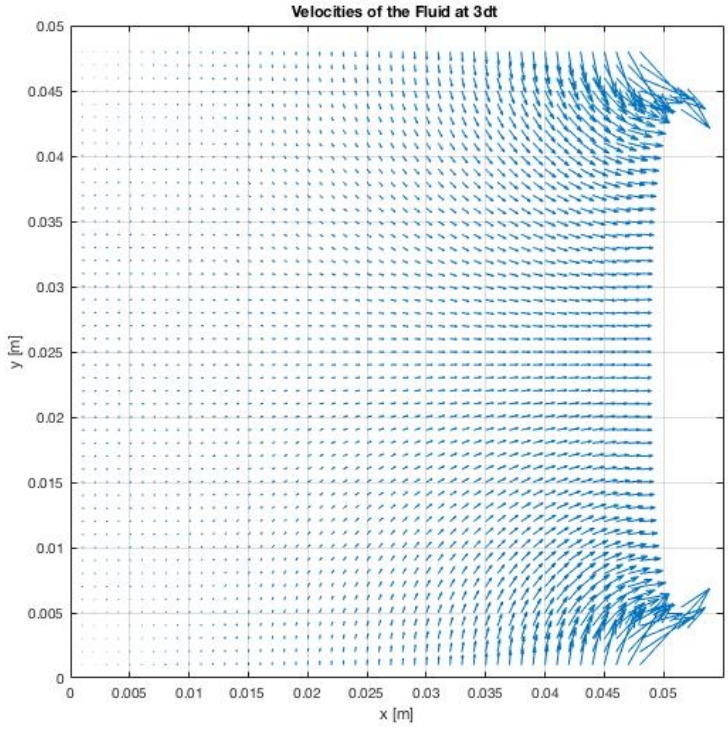
Figure 6 shows the velocity of the fluid at the third timestep. We indeed see that the fluid has acquired a speed everywhere in the field except where the obstacle is. As dictated by the gradient of the initial pressure, we also see that the magnitude of the velocity is higher near the area of sharper change in pressure. The fluid flows around the obstacle and continues to gain more speed in the $+x$ direction.



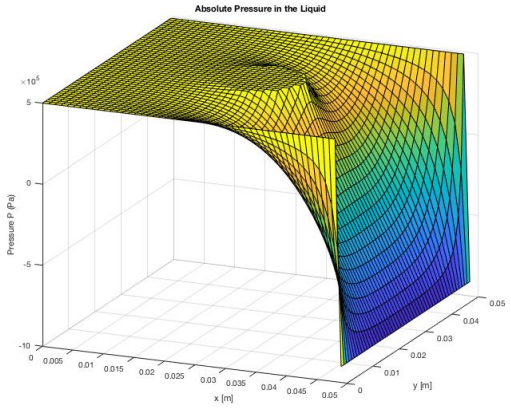
(a) Figure 1



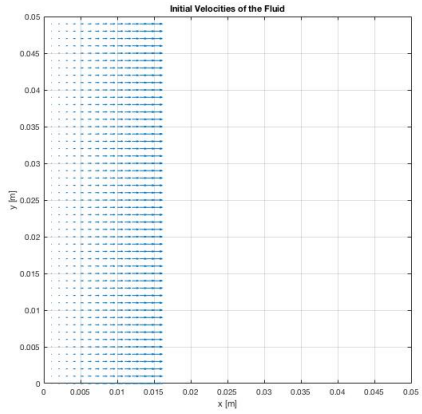
(b) Figure 2



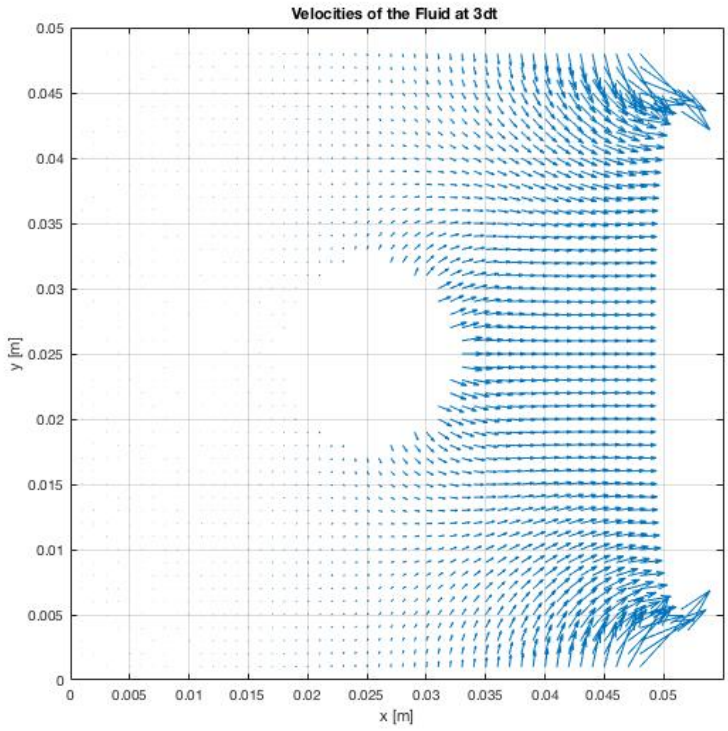
(c) Figure 3



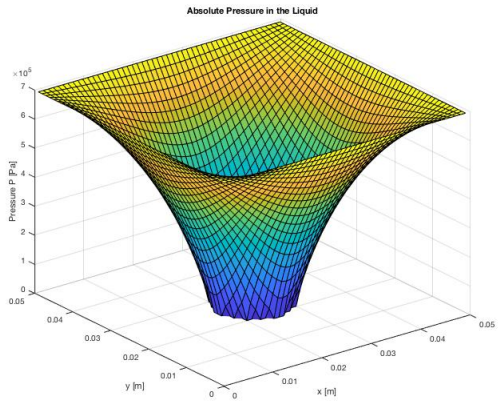
(a) Figure 4



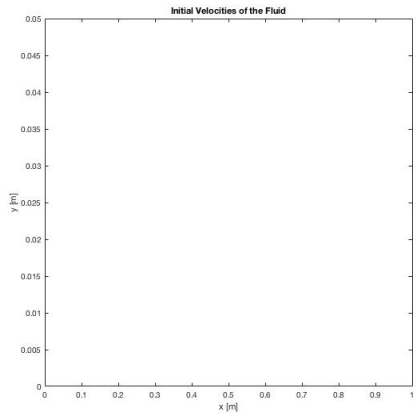
(b) Figure 5



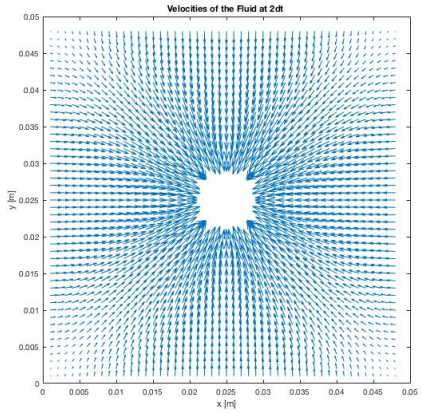
(c) Figure 6



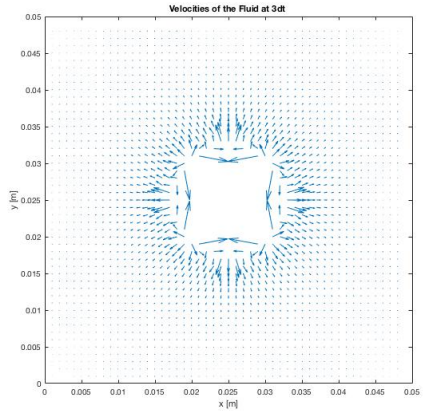
(a) Figure 7



(b) Figure 8



(c) Figure 9



(d) Figure 10

Simulation 3

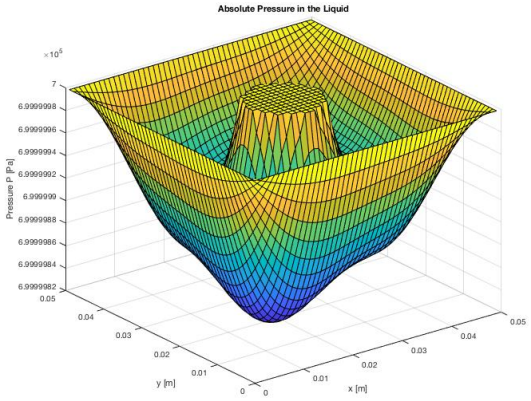
In the third simulation, we set all of the four exterior boundaries at a fixed pressure of 700,000 Pa, and a small circular region in the middle of the domain to a fixed pressure of 0 Pa (see Figure 7). In this case all the fluid was initially at rest (see Figure 8). In Figure 7, we have the initial pressure as function of space, calculated using method of relaxation. From this initial pressure, we expect the fluid to start flowing towards the middle circular region, which would be acting as a sink.

Figure 9 shows the velocity of the fluid at the second timestep. We indeed see that the fluid has at all places started to flow towards the middle circular region, as dictated by the gradient of the initial pressure. At the third time step however (see Figure 10), we see that the flow towards the middle region has stopped, and there is even flow directed away from this region. This makes sense, since the initial flow towards the middle would result in increased pressure in the vicinity of the circular region, causing the pressure there to increase and thus accelerate the fluid away from the middle. Naturally this would happen since the middle circular region is not a perfect sink, but just a region at a lower pressure.

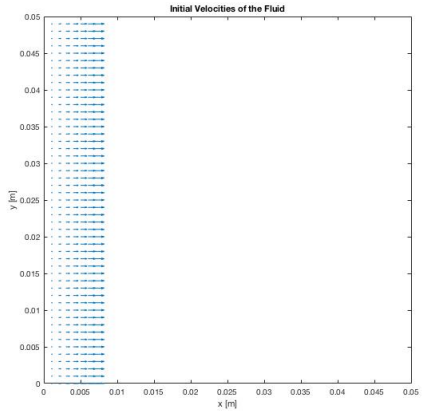
Simulation 4

In the fourth simulation, we set all of the four exterior boundaries, along with a circular region in the middle of the domain (as in Simulation 3), at a fixed pressure of 700,000 Pa (see Figure 11). In this case we used the same initial velocity for the fluid as in Simulation 2 (see Figure 12). In Figure 11, we have the initial pressure as function of space, calculated using method of relaxation. From this initial pressure, we expect the fluid to start flowing towards region between the boundaries and the middle disk, and interact also somehow with the initially moving segments of the fluid (acting like a superposed wave pulse).

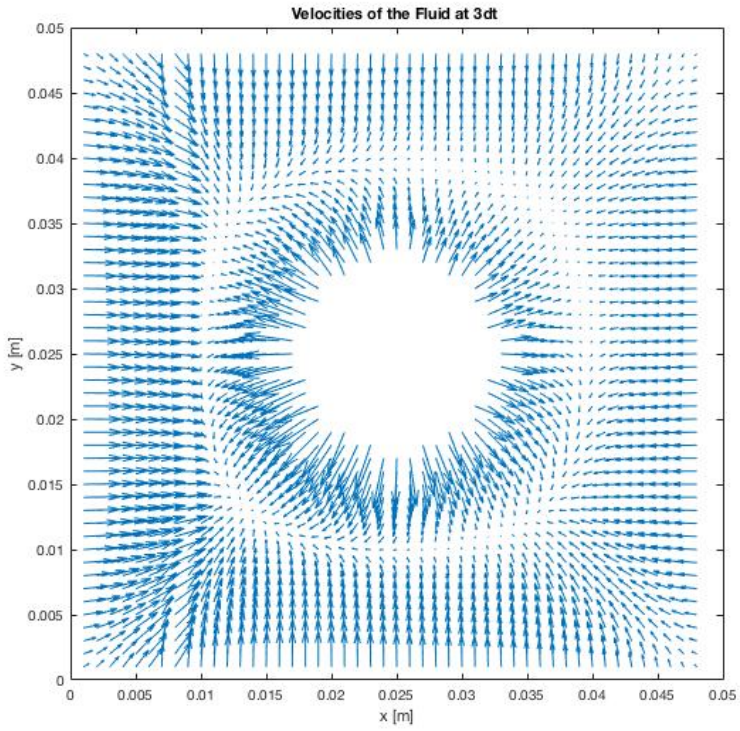
Figure 13 shows the velocity of the fluid at the third timestep. We indeed see that the fluid has at all places started to flow towards the region between the boundaries and the middle disk, as dictated by the gradient of the initial pressure. We also see that the initially moving region does resemble a superposed wave pulse traveling to the right, but we still require more computational time in order to be able to see how it proceeds to the right later in time.



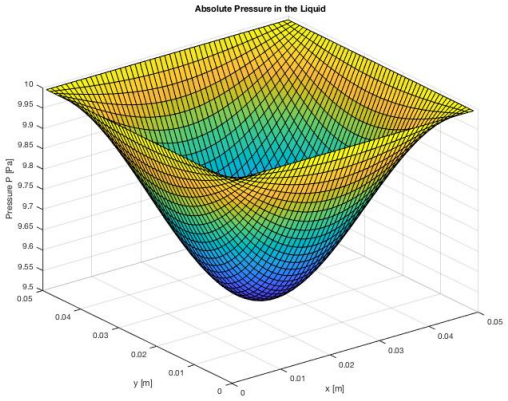
(a) Figure 11



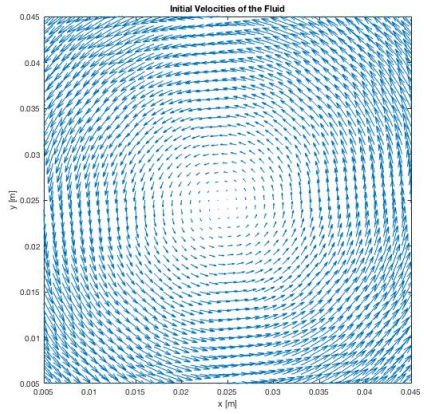
(b) Figure 12



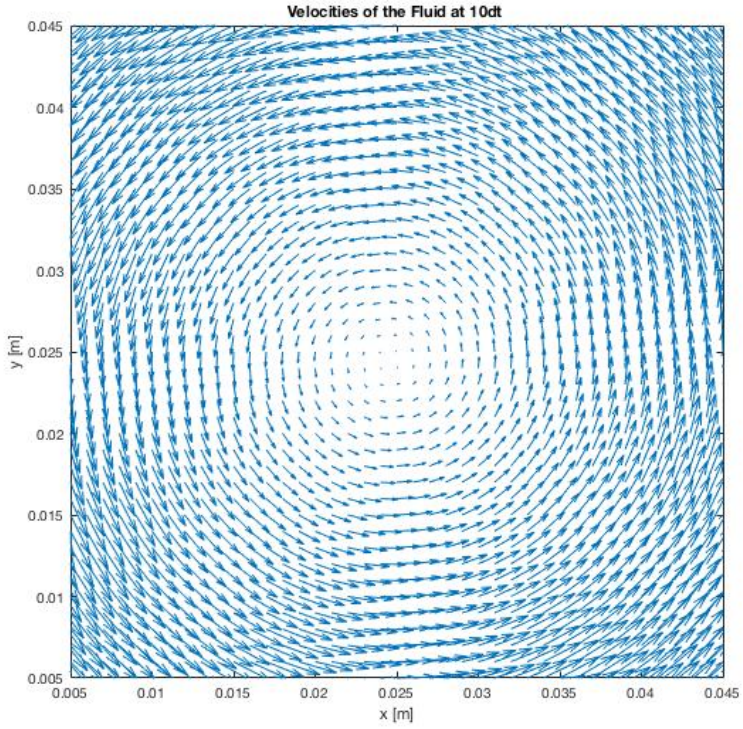
(c) Figure 13



(a) Figure 14



(b) Figure 15



(c) Figure 16

Simulation 5

In the fifth simulation, we set all of the four exterior boundaries at a fixed pressure of 10 Pa (see Figure 14). In this case the fluid was given an initial velocity (see Figure 8) of

$$\mathbf{v}_0(x, y) = \left(-\left(y - \frac{L}{2} \right), x - \frac{L}{2} \right), \tag{7}$$

where L is the length of an edge of the (square) spatial domain. In Figure 14, we have the initial pressure as function of space, calculated using method of relaxation. Considering this initial pressure, we expect that its gradient will be able to supply the needed centripetal acceleration in order to 'sustain' revolving motion of the fluid.

Figure 16 shows the velocity of the fluid at the tenth timestep. We indeed see that the fluid has practically maintained its initial velocity. That means that the pressure gradient was indeed able to supply the necessary centripetal acceleration needed to maintain the revolving motion. This simulation was also valuable since it showed (short-period) stability of a vortex – a trait of some solutions to the Navier-Stokes equation.

Conclusion

Considering some of the qualitative similarities between the previous simulations and commonly observed fluid flows, we can see that the method implemented is a valid start in attempting to simulate fluid flow. Even though the simulations are able to convey aspects of the mechanical behaviour of fluid, due to uncertain accuracy of the numerical method in some particular setups, we can't rely on this method to give us an understanding of the fluid's motion for instances when we don't have clear qualitative predictions and expectations.

This is primarily due to the fact that the executed simulations – due to practical time limitations – were done at a lower accuracy. Since we are using Euler's Method, along with the Method of Relaxation, with all spatial derivatives being $\mathcal{O}((\Delta L)^2)$, the error in the algorithm can be generally represented by $\max \mathcal{O}(\Delta t), \mathcal{O}((\Delta L)^2), \mathcal{O}(\Delta p)$. Thus, in order to increase the accuracy of the method, one should minimize the intervals Δt and ΔL, as well as the tolerance Δp used in the Method of Relaxation. By modifying these parameters sufficiently, we expect to the able to get very close to the actual solutions for the fluid's motion, since the two aforementioned numerical methods approach the actual solutions as limits, as accuracy is increased. Nevertheless, the lack of sufficient reliability of this method with relatively low accuracy, testifies to large error magnifications associated with this problem and our approach.

Another systematic limitation to our simulations is the fact that in our approach, we were unable to make the edges of the two dimensional domain be perfectly rigid walls. This is due to the fact that, when they were assigned to a large fixed pressure value – in an attempt to create a large discontinuity in the pressure, – after the method of relaxation it was always smoothly decreasing with increasing distance from the spatial domain's edges. Because of this, in the simulations one could observe that the fluid near the boundaries was acquiring a velocity component normal to them. The fact that the fluid could not be isolated by the boundaries, also lead to a non zero energy flux through these boundaries. Addressing the option of having rigid walls is definitely a priority in any continuation of this study.

Examples of other modifications that can be made in further work are: the option of having periodic boundary conditions (allowing to simulate the flow in a toroidal region); simulations

done for a three-dimensional spatial domain; and the possibility of drawing the trajectories of small segments of the fluid during the simulation.

References

[1] Li, Ming. Numerical Solutions for the Incompressible Navier Stokes Equations. PhD Thesis. 1999.

[2] U. Ghia, K.N Ghia, C.T Shin. High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method. Journal of Computational Physics. PP 387-411. 1982.

[3] Landau, L. D., Lifschitz, E. M., A Course in Theoretical Physics, Volume 6, Fluid Dynamics.

[4] Fefferman, C. L., Existence and Smoothness of the Navier-Stokes Equation, Official Millenium Problem description.

[5] Majda, A. J., Bertozzi, A. L., Vorticity and Incompressible Flow, Cambridge Texts in Applied Mathematics, 2002.

Appendix: MATLAB Code

```
1     function [vx, vy, P] = NavierStokes2d(fx, fy, vx0, vy0, dL, dt, NL, Nt, bp, rho, nu, deltap)
2  % NavierStokes2d(fx, fy, vx0, vy0, dL, dt, NL, Nt, bp, rho, nu, deltap):
3  % solves the two-dimensional Navier-Stokes equation
4  % inputs:
5  %     fx: a 3d matrix for the x-component of external forces
6  %     fy: a 3d matrix for the y-component of external forces
```

```
%     vx0: a 3d matrix for the x-component of initial velocity of the fluid
%     vy0: a 3d matrix for the y-component of the initial velocity
%     dL: the integration step size
%     dt: the time step
%     NL: the number of points in the field
%     Nt: the number of time steps
%     bp: the two-dimensional boundary point matrix
%     rho: the density of the fluid
%     nu: the viscosity of the fluid
%     deltap: the tolerance value when integrating pressure p
% output:
%     vx: a 3d matrix for the x-component of velocity of the fluid with the
%     first dimension for y, second dimension for different x, and the
%     third dimension for time t
%     vy: a 3d matrix for the y-component of velocity of the fluid
%     P: a 3d matrix for pressure
% errors: none

vx = zeros(NL, NL, Nt);
vy = zeros(NL, NL, Nt);
P = zeros(NL, NL, (Nt-1));
vx(:, :, 1) = vx0;
vy(:, :, 1) = vy0;
ax = zeros(NL, NL);
ay = zeros(NL, NL);

for k = 1:(Nt-1)

    %calculating 1st ax and ay
    for i = 1:NL
        for j = 1:NL
           if (bp(i, j) == 0)
               % calculate x components of the right-hand side of equation 7 inside the del operator
               ax(i, j) = -vx(i, j, k)*d1(vx(:, :, k), dL, bp, i, j, 1) - vy(i, j, k)*d1(vx(:, :, k), dL, bp, i, j, 2) +
                     nu*d2(vx(:, :, k), dL, bp, i, j, 1) + nu*d2(vx(:, :, k), dL, bp, i, j, 2) + fx((i-1)*dL, (j-1)*dL, (k
                     -1)*dt);
               % calculate y components
               ay(i, j) = -vx(i, j, k)*d1(vy(:, :, k), dL, bp, i, j, 1) - vy(i, j, k)*d1(vy(:, :, k), dL, bp, i, j, 2) +
                     nu*d2(vy(:, :, k), dL, bp, i, j, 1) + nu*d2(vy(:, :, k), dL, bp, i, j, 2) + fy((i-1)*dL, (j-1)*dL, (k
                     -1)*dt);
           end
        end
    end

    %using 1st ax and ay to get p at time step
    P(:, :, k) = p(ax, ay, bp, dL, deltap);
```

```
49
50      %calculating 2nd ax and ay, and then vx and vy for time step k+1
51      for i = 1:NL
52          for j = 1:NL
53              if (bp(i, j) == 0)
54                  ax(i, j) = ax(i, j) - d1(P(:, :, k), dL, bp, i, j, 1)/rho;
55                  ay(i, j) = ay(i, j) - d1(P(:, :, k), dL, bp, i, j, 2)/rho;
56                  vx(i, j, k+1) = vx(i, j, k) + dt*ax(i, j);
57                  vy(i, j, k+1) = vy(i, j, k) + dt*ay(i, j);
58              end
59          end
60      end
61  end
```

```
1     function p_out = p(ax, ay, bp, dL, deltap)
2   % p(p0, ax, ay, bp, deltap): solves the incompressibility
3   % equation for pressure p(x,y,t) at a time step using method of relaxation
4   % while taking the boundary and interior boundry points
5   % inputs:
6   %     ax: a 2d array for x components of the r.h.s of eq. 7 inside the del operator
7   %     ay: a 2d array for y components of the r.h.s of eq. 7 inside the del operator
8   %     bp: a 2-d array for boundry points; if bp(i, j) is 1, the
9   %         corresponding point is a boundary point which is held at a
10  %         fixed pressure and should not be changed during the iteration.
11  %     dL: the calculation step size for del
12  %     deltap: specifies the accuracy, the tolerance value for pressure
13  % output:
14  %     p_out: the final 2d matrix of pressure at a designated timestep
15  % errors: none
16
17  sizep = size(bp);
18  NL = sizep(1);
19
20  % the initial conditions for p(x,y) at all points
21  p0=zeros(NL,NL);
22  p0=bp*100000;
23
24  p_out=p0;
25  Maxdp=deltap+1; % initialize Maxdp to be an arbituary value
26  while Maxdp>=deltap
27      prevP=p_out;
28      for i=1:length(p0(:,1)) % loop for y
29          for j=1:length(p0(1,:)) % loop for x
30              if bp(i,j)~=1
31                  rhs=(1/4)*(dL^2)*(d1(ax,dL,bp,i,j,1)+d1(ay,dL,bp,i,j,2));
32                  p_out(i,j) = (1/4)*(prevP(i+1,j)+prevP(i-1,j)+prevP(i,j+1)+prevP(i,j-1))-rhs;
```

```
            end
        end
    end
    Maxdp=max(max(abs(p_out-prevP)));
end
```

```
   function d1_out = d1(F, dL, bp, i, j, C)
% d1(F, dL, bp, i, j, C): calculates the del of F at the point of y-index i
% and x-index j
% inputs:
%     F: a direction component of a two-dimensional vector for the current point in the field
%     dL: the calculation step size
%     bp: the two-dimensional matrix for boundary points
%     i: the row index for the current point
%     j: the column index for the current point
%     C: if it equals 1, we take the partial derivatives with respect to y;
%        if it equals 2, we take the partial derivatives with respect to x
% output:
%     d1_out: the del scalar value for the current point
% errors: none

if (C == 2)
    if ((bp(i+1, j) == 0) && (bp(i-1, j) == 0))
        d1_out = (F(i+1, j) - F(i-1, j))/(2*dL);
    elseif ((bp(i+1, j) == 0) && (bp(i-1, j) == 1))
        d1_out = ((-3/2)*F(i, j) + 2*F(i+1, j) + (-1/2)*F(i+2, j))/dL;
    else
        d1_out = ((3/2)*F(i, j) + (-2)*F(i-1, j) + (1/2)*F(i-2, j))/dL;
    end
else
    if ((bp(i, j+1) == 0) && (bp(i, j-1) == 0))
        d1_out = (F(i, j+1) - F(i, j-1))/(2*dL);
    elseif ((bp(i, j+1) == 0) && (bp(i, j-1) == 1))
        d1_out = ((-3/2)*F(i, j) + 2*F(i, j+1) + (-1/2)*F(i, j+2))/dL;
    else
        d1_out = ((3/2)*F(i, j) + (-2)*F(i, j-1) + (1/2)*F(i, j-2))/dL;
    end
end
```

```
   function d2_out = d2(F, dL, bp, i, j, C)
% d2(F, dL, bp, i, j, C): calculates the laplacian of F at the point of y-index i
% and x-index j
% inputs:
%     F: a direction component of a two-dimensional vector for the current point in the field
%     dL: the calculation step size
%     bp: the two-dimensional matrix for boundary points
```

```
%     i: the row index for the current point
%     j: the column index for the current point
%     C: if it equals 1, we take the partial derivatives with respect to y;
%        if it equals 2, we take the partial derivatives with respect to x
% output:
%     d2_out: the laplacian scalar value for the current point
% errors: none

if (C == 2)
    if ((bp(i+1, j) == 0) && (bp(i-1, j) == 0))
        d2_out = (F(i+1, j) - 2*F(i, j) + F(i-1, j))/(dL^2);
    elseif ((bp(i+1, j) == 0) && (bp(i-1, j) == 1))
        d2_out = (2*F(i, j) + (-5)*F(i+1, j) + 4*F(i+2, j) + (-1)*F(i+3, j))/(dL^2);
    else
        d2_out = (2*F(i, j) + (-5)*F(i-1, j) + 4*F(i-2, j) + (-1)*F(i-3, j))/(dL^2);
    end
else
    if ((bp(i, j+1) == 0) && (bp(i, j-1) == 0))
        d2_out = (F(i, j+1) - 2*F(i, j) + F(i, j-1))/(dL^2);
    elseif ((bp(i, j+1) == 0) && (bp(i, j-1) == 1))
        d2_out = (2*F(i, j) + (-5)*F(i, j+1) + 4*F(i, j+2) + (-1)*F(i, j+3))/(dL^2);
    else
        d2_out = (2*F(i, j) + (-5)*F(i, j-1) + 4*F(i, j-2) + (-1)*F(i, j-3))/(dL^2);
    end
end
```

```
   function bp = circle(bp,r,N,dL,xa,ya)
% circle(bp,r,N,dL,xa,ya): returns a boundary point matrix that has a
% round-shape obstacle; points covered in a circle have values of 1 and all
% other points have values of 0
% inputs:
%     bp: the original boundary point matrix
%     r: radius of the round-shape obstacle
%     N: number of points in the boundary point matrix
%     dL: step size between points
%     xa: the horizontal translational distance of the center of the circle
%     ya: the vertical translational distance of the center of the circle
% output:
%     bp: the final 2d boundary matrix
% errors: none

i=1;
j=1;
x=0:dL:(N-dL)*dL;
y=0:dL:(N-dL)*dL;
for j=1:N
```

```
    for i=1:N
        check=((x(j)-xa)^2)+((y(i)-ya)^2);
        if check<=(r^2)
            bp(i,j)=1;
        end
        i=i+1;
    end
    j=j+1;
end
```

In `run.m`:

```
   % This is the script for calling the functions and plotting the graphs
clear all;
dL = 0.001;
dt = 0.01;
NL = 50;
Nt = 60;
rho = 1000;
nu = 1e-4;
deltap = 0.001;
x = 0:dL:(NL*dL-dL);
y = 0:dL:(NL*dL-dL);
t = 0:dt:(Nt*dt-dt);

% ------------ initial conditions ------------
vx0 = zeros(NL, NL);
vy0 = zeros(NL, NL);
for i=1:NL
    for j=1:NL
        vx0(i,j) = -(y(i)-(NL-1)*dL/2);
        vy0(i,j) = x(j)-(NL-1)*dL/2;
    end
end
bp = zeros(NL, NL);
bp(:, 1) = 1;
bp(:, NL) = 1;
bp(1, :) = 1;
bp(NL, :) = 1;
% bp = circle(bp,NL*dL/6,NL,dL,NL*dL/2,NL*dL/2);

% ------------ calling the functions ------------
[vx, vy, P] = NavierStokes2d(@(x, y, t) 0, @(x, y, t) 0, vx0, vy0, dL, dt, NL, Nt, bp, rho, nu, deltap);
[xgrid, ygrid] = meshgrid(x, y);

% ------------ plotting the graphs ------------
```

```
for k = 1:(length(t)-1)
    fig = figure;
%     surf(x, y, P(:, :, k));
    quiver(xgrid, ygrid, vx(:, :, k), vy(:, :, k), 3);
    pause(2*dt);
    M(k)=getframe(fig);
end
movie(M)
```