CFD Solver for Two-Dimensional Cavity Flow Problem

Titaporn Fongwattanagoon

March 13, 2023

This work aims to solve the Navier-Stokes (N-S) equations using Finite Difference Method (FDM) for a Two-Dimensional Cavity Fluid Problem.

The CFD Solver is written in C while the obtained velocity profiles are visualized with Python.

1 Mathematics

1.1 The Governing Equation

The equation of motion can be written as:

$$\rho \frac{D\vec{v}}{Dt} = -\nabla p - \nabla \cdot \vec{\tau} + \rho \vec{g}$$

For an incompressible fluid with constant ρ and μ, the equation becomes:

$$\rho \frac{\partial \vec{v}}{\partial t} + \rho(\vec{v} \cdot \nabla)\vec{v} = -\nabla p - \mu \nabla^2 \vec{v} + \rho \vec{g}$$

The **Navier-Stokes equations** in two-dimensions are:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial x} + \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial y} + \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right)$$

1.2 Pressure Poisson Equation

The pressure Poisson equation can be written as:

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = -\rho\left(\frac{\partial u}{\partial x}\frac{\partial u}{\partial x} + 2\frac{\partial u}{\partial y}\frac{\partial v}{\partial x} + \frac{\partial v}{\partial y}\frac{\partial v}{\partial y}\right) + \rho\frac{\partial}{\partial t}\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right)$$

1.3 Discretization

The 2 momentum equations and a pressure equation are discretized using central difference method.

For velocity component **u**:

$$u_{i,j}^{n+1} = u_{i,j}^n - u_{i,j}^n \Delta t \frac{(u_{i+1,j}^n - u_{i-1,j}^n)}{2\Delta x} - v_{i,j}^n \Delta t \frac{(u_{i,j+1}^n - u_{i,j-1}^n)}{2\Delta y} - \frac{\Delta t}{\rho} \frac{(p_{i+1,j}^n - p_{i-1,j}^n)}{2\Delta x} + \nu \Delta t \left(\frac{u_{i+1,j}^n - 2u_{i,j}^n + u_{i-1,j}^n}{\Delta x^2} + \frac{u_{i,j+1}^n - 2u_{i,j}^n + u_{i,j-1}^n}{\Delta y^2} \right)$$

For velocity component **v**:

$$v_{i,j}^{n+1} = v_{i,j}^n - u_{i,j}^n \Delta t \frac{(v_{i+1,j}^n - v_{i-1,j}^n)}{2\Delta x} - v_{i,j}^n \Delta t \frac{(v_{i,j+1}^n - v_{i,j-1}^n)}{2\Delta y} - \frac{\Delta t}{\rho} \frac{(p_{i,j+1}^n - p_{i,j-1}^n)}{2\Delta y} + \nu \Delta t \left(\frac{v_{i+1,j}^n - 2v_{i,j}^n + v_{i-1,j}^n}{\Delta x^2} + \frac{v_{i,j+1}^n - 2v_{i,j}^n + v_{i,j-1}^n}{\Delta y^2} \right)$$

For pressure component **p**:

$$\text{Let } b = [\text{R.H.S.}]$$

$$\frac{p_{i+1,j}^n - 2p_{i,j}^n + p_{i-1,j}^n}{\Delta x^2} + \frac{p_{i,j+1}^n - 2p_{i,j}^n + p_{i,j-1}^n}{\Delta y^2} = b$$

$$p_{i,j}^n = \frac{\Delta y^2 (p_{i+1,j}^n + p_{i-1,j}^n) + \Delta x^2 (p_{i,j+1}^n + p_{i,j-1}^n) - b \Delta y^2 \Delta x^2}{2(\Delta x^2 + \Delta y^2)}$$

1.4 Initial Conditions (I.C.)

At the initial, let $u, v, p = 0$

1.5 Boundary Conditions (B.C.)

Within the domain $x_i \in [0, 2]$ and $y_i \in [0, 2]$

$u = \begin{cases} 1 \text{ for } & y_i = 2 \\ 0 \text{ for } & x_i = 0, 2, \ y_i = 0 \end{cases}$

$v = 0$ for $x_i = 0, 2, \ y_i = 0, 2$

$p = 0 \ at \ y = 2$

$\frac{\partial p}{\partial x} = 0 \ at \ x = 0, 2$

2 Codes

2.1 Process

The **CFD solver (C)** does the following steps:

1. Set up parameters (mesh, time, and fluid properties)
2. Initialize velocity components and pressure arrays
3. For each timestep:
   (a) Solve N-S and pressure equations via central difference method
   (b) Apply boundary conditions
4. Write output arrays of velocity components and pressure to output files

The **CFD visualizer (Python)** does the following steps:

1. Set up mesh
2. Initialize velocity components and pressure arrays
3. Fill the arrays with values from input files
4. Plot cavity flow

2.2 Directory Structure

```
/
├── vectorutils.c
├── vectorutils.h
├── finitedifference.c
├── finitedifference.h
├── fluiddynamics.c
├── fluiddynamics.h
├── main.c
├── Makefile
├── results/
│   ├── u.txt
│   ├── v.txt
│   ├── p.txt
│   └── cavityFlow.png
└── plotProfiles.py
```

2.3 Programs

The C program consists of the following sections:

- **finitedifference** - compute FDM approximations
- **fluiddynamics** - solve N-S and Poisson equations
- **vectorutils** - utilities
- **main** - setup, solve, and write outputs

Program, finitedifference.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "vectorutils.h"

float** central_ddx(float** f, float dx, int m, int n){
  float** diff = init_zero_vector(m,n);

  for (int i=1; i<m-1; ++i){
    for (int j=1; j<n-1; ++j){
      diff[i][j] = (f[i][j+1] - f[i][j-1])/(2.0*dx);
    }
  }
  return diff;
}


float** central_ddy(float** f, float dy, int m, int n){
  float** diff = init_zero_vector(m,n);

  for (int i=1; i<m-1; ++i){
    for (int j=1; j<n-1; ++j){
      diff[i][j] = (f[i+1][j] - f[i-1][j])/(2.0*dy);
    }
  }
  return diff;
}


float** laplace(float** f, float dx, float dy, int m, int n){
  float** diff = init_zero_vector(m,n);

  for (int i=1; i<m-1; ++i){
    for (int j=1; j<n-1; ++j){
      diff[i][j] = (f[i][j+1] - 2*f[i][j] + f[i][j-1])/pow(dx, 2) +
    (f[i+1][j] - 2*f[i][j] + f[i-1][j])/pow(dy, 2);
    }
  }
  return diff;
}
```

Program, fluiddynamics.c

```c
#include <stdio.h>
```

```c
#include <stdlib.h>
#include <math.h>
#include "vectorutils.h"
#include "finitedifference.h"

float** compute_b(float dx,float dy,float rho,float dt, float** u, float** v, int m, int n){
  float** b = init_zero_vector(m,n);
  float** dudx = central_ddx(u, dx, m, n);
  float** dvdy = central_ddy(v, dy, m, n);

  float** dudy = central_ddy(u, dy, m, n);
  float** dvdx = central_ddx(v, dx, m, n);

  for (int i=1; i<m-1; ++i){
      for (int j=1; j<n-1; ++j){
        b[i][j] = -rho*(pow(dudx[i][j],2) + 2*dudy[i][j]*dvdx[i][j] + pow(dvdy[i][j],2)) + rho/dt*(dudx[i][j] + dvdy[i][j]);
      }
    }

  free(dudx);
  free(dvdy);
  free(dudy);
  free(dvdx);

  return b;
}


float** pressure_poisson(float** p, float dx, float dy, float rho, float dt, float** u, float** v, int m, int n){
  float** pn = init_zero_vector(m,n);
  int maxIteration = 50;

  float** b = compute_b(dx,dy,rho,dt,u,v,m,n);

  for (int it=0; it<maxIteration; ++it){
    copy_vector(pn, p, m, n);

    for (int i=1; i<m-1; ++i){
      for (int j=1; j<n-1; ++j){
        p[i][j] = (pow(dy,2)*(pn[i][j+1] + pn[i][j-1]) + pow(dx,2)*(pn[i+1][j] + pn[i-1][j]) - b[i][j]*pow(dy,2)*pow(dx,2))/(2*pow(dx,2) + 2*pow(dy,2));

      }
    }
    // B.C.
    for (int i=0; i<m; ++i){
      p[i][n-1] = p[i][n-2]; // dp/dx = 0 at x = 2 (right)
      p[i][0] = p[i][1]; // dp/dx = 0 at x = 0
    }

    for (int j=0; j<n; ++j){
      p[0][j] = p[1][j]; // dp/dy = 0 at y = 0
      p[m-1][j] = 0.0; // p = 0   at y = 2 (top)
```

```
    }
  }
  free(pn);

  return p;
}


int cavity_flow(int nt, float** u, float** v, float dt, float dx, float dy, float** p, float rho, float nu, float ut, int m, int n){
  float** un = init_zero_vector(m,n);
  float** vn = init_zero_vector(m,n);

  for (int it=0; it<nt; ++it){
    copy_vector(un, u, m, n);
    copy_vector(vn, v, m, n);

    p = pressure_poisson(p, dx, dy, rho, dt, un, vn, m, n);

    float** dudx = central_ddx(un, dx, m, n);
    float** dudy = central_ddy(un, dy, m, n);
    float** dpdx = central_ddx(p, dx, m, n);
    float** laplaceU = laplace(un, dx, dy, m, n);

    float** dvdx = central_ddx(vn, dx, m, n);
    float** dvdy = central_ddy(vn, dy, m, n);
    float** dpdy = central_ddy(p, dy, m, n);
    float** laplaceV = laplace(vn, dx, dy, m, n);
    for (int i=1; i<m-1; ++i){
        for (int j=1; j<n-1; ++j){
          u[i][j] = un[i][j] + dt*(-un[i][j]*dudx[i][j]-vn[i][j]*dudy[i][j]-1/rho*dpdx[i][j]+nu*laplaceU[i][j]);
          v[i][j] = vn[i][j] + dt*(-un[i][j]*dvdx[i][j]-vn[i][j]*dvdy[i][j]-1/rho*dpdy[i][j]+nu*laplaceV[i][j]);
        }
    }

    free(dudx);
    free(dudy);
    free(dpdx);
    free(laplaceU);
    free(dvdx);
    free(dvdy);
    free(dpdy);
    free(laplaceV);

    // B.C.
      for (int j=0; j<n; ++j){
      u[0][j] = 0.0;    // bottom
      v[0][j] = 0.0;
      u[m-1][j] = ut;   // top
      v[m-1][j] = 0.0;
    }
    for (int i=0; i<m; ++i){
      u[i][0] = 0.0;    // left
      v[i][0] = 0.0;
```

```
      u[i][n-1] = 0.0; // right
      v[i][n-1] = 0.0;
    }
  }

  free(un);
  free(vn);

  return 0;
}
```

Program, vectorutils.c

```
#include <stdio.h>
#include <stdlib.h>

float** init_zero_vector(int m, int n){
  float* values = calloc(m*n, sizeof(float));
  float** vec = malloc(m*sizeof(float*));

  for (int i=0; i<m; ++i){
    vec[i] = values + i*n;
  }
  return vec;
}


void copy_vector(float** newV, float** previousV, int m, int n){
  size_t size = sizeof(float)*n;
  for (int i=0; i<m; ++i){
    for (int j=0; j<n; ++j){
      newV[i][j] = previousV[i][j];
    }
  }
}


void print_vector(float** vec, int m, int n){
  for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++)
            printf("%f,", vec[i][j]);
        printf("\n");
    }
}


int write_result(char* fileName, float** vec, int m, int n){
  FILE *outFile;
  outFile=fopen(fileName,"w+");
  if (outFile==NULL){
        printf("[Error] Failed to open output file %s", fileName);
        return -1;
    }

  for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++)
            fprintf(outFile, "%f,", vec[i][j]);
        fprintf(outFile, "\n");
```

```c
    }
  fclose(outFile);

  return 0;
}
```

Program, main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "vectorutils.h"
#include "fluiddynamics.h"

int main(void) {
    printf("Solving Cavity Fluid Flow using Finite-Difference Approach\n");

    // Mesh and parameter set-up
    const int N_GRID_X = 41;
    const int N_GRID_Y = 41;
    const float LENGTH_X = 2.0;
    const float LENGTH_Y = 2.0;

    const float dx = LENGTH_X / (N_GRID_X - 1);
    const float dy = LENGTH_Y / (N_GRID_Y - 1);

    const int N_TIMESTEP = 100;
    const float dt = 0.001;

    const float DENSITY = 1.0;
    const float KINEMATIC_VISCOSITY = 0.1;
    const float U_TOP = 1.0;

    // I.C.
    float** u = init_zero_vector(N_GRID_X, N_GRID_Y);
    float** v = init_zero_vector(N_GRID_X, N_GRID_Y);
    float** p = init_zero_vector(N_GRID_X, N_GRID_Y);

    // Loop through timesteps
    cavity_flow(
        N_TIMESTEP, u, v, dt, dx, dy, p, DENSITY,
        KINEMATIC_VISCOSITY, U_TOP, N_GRID_X, N_GRID_Y);

    int u_txt = write_result("./results/u.txt", u, N_GRID_X, N_GRID_Y);
    int v_txt = write_result("./results/v.txt", v, N_GRID_X, N_GRID_Y);
    int p_txt = write_result("./results/p.txt", p, N_GRID_X, N_GRID_Y);

    if (u_txt!=0 || u_txt!=0 || u_txt!=0) {
        printf("[ERROR] Failed to write output files\n");
    } else {
        printf("[SUCCESS] Wrote output files\n");
        return 0;
    }

}
```

The Python program consists of the following section:

- **plotProfiles -** plot velocity profiles

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


def fill_profile(filename, vec):
    with open(filename) as f:
        for i, line in enumerate(f):
            data = line.split(",")[:-1]
            for j, value in enumerate(data):
                vec[i][j] = float(value)


def plot_flow(lx,ly,nx,ny,u,v,p):
    x = np.linspace(0, lx, nx)
    y = np.linspace(0, ly, ny)
    X, Y = np.meshgrid(x, y)

    fig = plt.figure(figsize=(12,8), dpi=100)
    plt.contourf(X, Y, p, alpha=0.3)
    plt.streamplot(X, Y, u, v)
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.title('2D-Cavity Flow')
    return fig

def main():
    N_GRID_X = 41
    N_GRID_Y = 41
    LENGTH_X = 2.0
    LENGTH_Y = 2.0

    u = np.zeros((N_GRID_X, N_GRID_Y))
    v = np.zeros((N_GRID_X, N_GRID_Y))
    p = np.zeros((N_GRID_X, N_GRID_Y))

    fill_profile("./results/u.txt", u)
    fill_profile("./results/v.txt", v)
    fill_profile("./results/p.txt", p)

    cavity_flow_plot = plot_flow(
        lx=LENGTH_X,
        ly=LENGTH_Y,
        nx=N_GRID_X,
        ny=N_GRID_Y,
        u=u,
        v=v,
        p=p)

    cavity_flow_plot.savefig("./results/cavityFlow.png", dpi=300)

main()
```

2.4 Output

After compile and execute the C program to obtain velocities and pressure profiles, run the python program to obtain the cavity flow plot as shown:

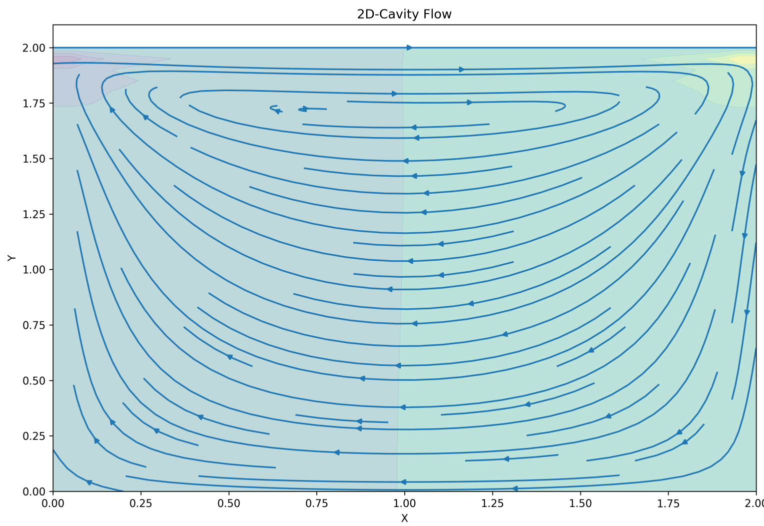

Figure 1: 2D Cavity Flow Plot

3 Appendix

3.1 Additional Code Sections of CFD Solver

C header files and Makefile

Program, finitedifference.h

```
float** central_ddx(float** f, float dx, int m, int n);
float** central_ddy(float** f, float dy, int m, int n);
float** laplace(float** f, float dx, float dy, int m, int n);
```

Program,fluiddynamics.h

```
float** compute_b(float dx,float dy,float rho,float dt, float** u,
    float** v, int m, int n);
float** pressure_poisson(float** p, float dx, float dy, float rho,
    float dt, float** u, float** v, int m, int n);
int cavity_flow(int nt, float** u, float** v, float dt, float dx,
    float dy, float** p, float rho, float nu, float ut, int m, int
    n);
```

Program, vectorutils.h

```
float** init_zero_vector(int m, int n);
void copy_vector(float** newV, float** previousV, int m, int n);
void print_vector(float** vec, int m, int n);
int write_result(char* fileName, float** vec, int m, int n);
```

Program,Makefile

```
CC    = gcc

all: vectorutils.o finitedifference.o fluiddynamics.o main.o
	$(CC) vectorutils.o finitedifference.o fluiddynamics.o main.o -o
    cavityflow

vectorutils.o: vectorutils.c
	$(CC) -c vectorutils.c -o vectorutils.o

finitedifference.o: finitedifference.c
	$(CC) -c finitedifference.c -o finitedifference.o

fluiddynamics.o: fluiddynamics.c
	$(CC) -c fluiddynamics.c -o fluiddynamics.o

main.o: main.c
	$(CC) -c main.c -o main.o

clean:
	rm *.o cavityflow
```