Final Assignment
DNS course 2015

Compressible 1D flow, Sod shock tube
Compressible 2D flow, Lid driven cavity
Finite Volume Method

Antti Mikkonen
antti.mikkonen at tut.fi

June 9, 2015

Sod Shock tube

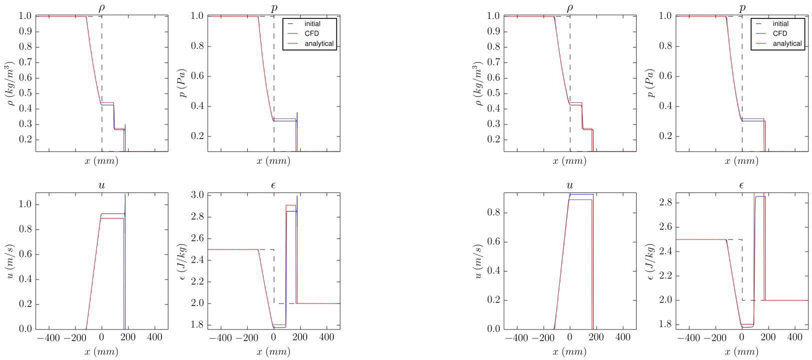


Figure : N=10000, $t = 0.1s$, left $\nu_H = 0.1$,right$\nu_H = 0.6$

Lid driven cavity

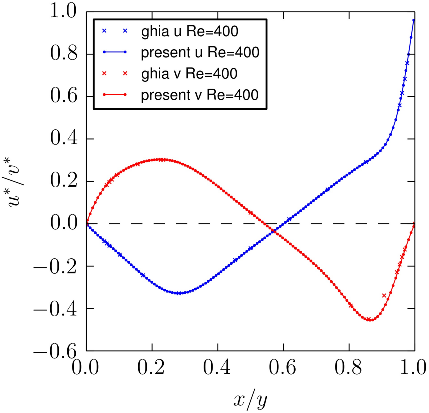
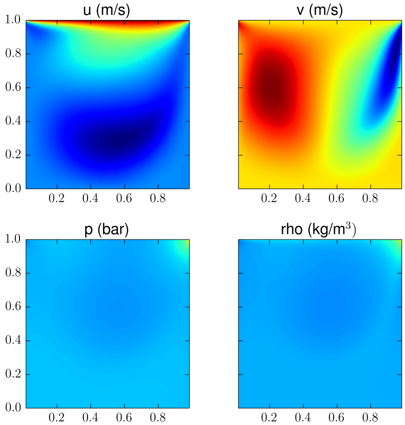


Figure : Test data

Results




```
Co        0.01152
clf diff  0.0036864
clf soni  0.0384

u lid     90.0
Re        400
nu        0.225
mu        0.271125

max u     0.963856162576
min u     -0.337051956293
max v     0.320720990696
min v     -0.637082135204
max rho   1.40450644535
min rho   1.12269824544
max p     116642.464675
min p     92493.3107295
```

Figure : Overview

Algorithm

- Initialize with smoothed fields
- Integrate time steps (Euler or Runge-Kutta 4)
  1. Solve continuity equation $\rightarrow d\rho$
  2. Solve momentum equation $\rightarrow d\rho U$
  3. Solve energy equation $\rightarrow de$
  4. Update primitives $\rightarrow du, dp$
  5. Filter ρ, e, p after every 100th time step with Laplace filter and $0.07\left(\frac{\Delta x}{\pi}\right)^2$ as the cutter
- Post-process

Governing equations

$$
\begin{aligned}
\frac{\partial \rho}{\partial t} &= -\nabla \cdot (\rho \overline{u}) \\
\frac{\partial (\rho \overline{u})}{\partial t} &= -\nabla \cdot (\rho \overline{u}\overline{u}) - \frac{\partial p}{\partial x} + \mu \Delta \overline{u} \\
\frac{\partial e}{\partial t} &= -\nabla \cdot (\overline{u}(\rho e + p)) \\
e &= \frac{p}{(\gamma - 1)} + \tfrac{1}{2}\rho |\overline{u}|^2
\end{aligned}
\tag{1}
$$

Python implementation snippet, N-S

```
drho  = self.dt * ( - self.dx * rhoU
                    - self.dy * rhoV
                    )

drhoU = self.dt * ( - self.dx * (rhoU * u)
                    - self.dy * (rhoU * v)
                    - self.dxZeroGrad * p
                    + self.mu * self.laplacian * u
                    + self.bRhoU # Source term
                    )

drhoV = self.dt * ( - self.dx * (rhoV * u)
                    - self.dy * (rhoV * v)
                    - self.dyZeroGrad * p
                    + self.mu * self.laplacian * v
                    )

de    = self.dt * ( - self.dx * ( u * (e + p))
                    - self.dy * ( v * (e + p))
                  )
```

Figure : Governing equation in Python

Discretization, continuity

$$
\begin{aligned}
\frac{\partial \rho}{\partial t} &= -\nabla \cdot (\rho \overline{u}) \\
&= -\frac{\partial(\rho u)}{\partial x} - \frac{\partial(\rho v)}{\partial y} \\
\Rightarrow \left(\frac{\partial \rho}{\partial t}\right)_{i,j} &\approx -\frac{1}{2\Delta x}\Big(\underbrace{[(\rho u)_{i+1} + (\rho u)_i}_{right} \underbrace{-[(\rho u)_{i-1} + (\rho u)_i]}_{left} \\
&\quad + \underbrace{[(\rho u)_{j+1} + (\rho u)_j}_{up} \underbrace{-[(\rho u)_{j-1} + (\rho u)_j]}_{down} \Big)
\end{aligned} \tag{2}
$$

At boundaries the corresponding term is set to zero. Operators dx and dy, no sources

Python implementation snippet, $\frac{\partial f}{\partial x}$

```
C = (2 * self.h)**-1
for row in range(self.nx):
    for col in range(self.nx):
        matrix_index = row*self.nx + col

        # left
        if col > 0:
            # middle
            xrows.append(matrix_index)
            xcols.append(matrix_index)
            xvals.append(-C)
            # left
            xrows.append(matrix_index)
            xcols.append(matrix_index-1)
            xvals.append(-C)
        # right
        if col < self.nx -1 :
            # middle
            xrows.append(matrix_index)
            xcols.append(matrix_index)
            xvals.append(C)
            # right
            xrows.append(matrix_index)
            xcols.append(matrix_index+1)
            xvals.append(C)

# No boundary effects
self.dx = sparse.csr_matrix(sparse.coo_matrix((xvals, (xrows, xcols))))
```

Figure : $\frac{\partial f}{\partial x}$ operator