BSc Computational and Data Science
cds-304 Strömungssimulation
Dozent: Demian Lauper

Lid Driven Cavity

Daniel Schafhäutle[1]

[1]*Fachhochschule Graubünden*
[*]*E-Mail Adressen: daniel.schafhäutle@stud.fhgr.ch*

1. Januar 2025

1 Projektbeschreibung

Lid Driven Cavity ist ein weit verbreitetes Benchmarkproblem in der Strömungssimulation. Ein Quadratischen Container mit drei statischen Wänden und einem Deckel, der sich mit einer konstanten Geschwindigkeit in x Richtung bewegt, enthält eine Flüssigkeit.
Es soll simuliert werden, wie sich diese Flüssigkeit verhält. Dafür wird angenommen, dass die Flüssigkeit Inkompressibel und nicht turbulent ist.

Die Simulation wird in Python durchgeführt. Die Domäne wird dabei in einem Gitter diskretisiert und jeweils nur an diesen Punkten ausgewertet.

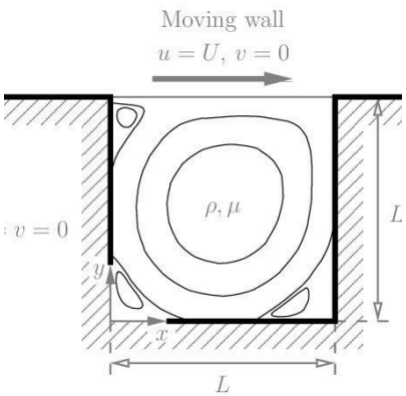

Abbildung 1: Lid Driven Cavity

Die Navier Stokes Equations werden an diesen Punkte mittels Finiter Differenzen Methode numerisch gelöst.
Für die Lösung werden Finite Differenzen mit Chorins Projektion verwendet.

Ich habe zuerst versucht, das Projekt ohne Hilfe umzusetzten, bin aber beim Lösen der Poisson Druckgleichung nicht mehr weitergekommen. Da habe ich ein Tutorial auf YouTube gefunden, dass mir weiterhelfen konnte (Koehler, o. J.). Der Code ist teilweise inspiriert von dem Video. Habe es hauptsächlich benutzt, um meine Formeln zu überprüfen.

1.1 Umsetzung

1.1.1 Impulsgleichung ohne Berücksichtigung des Druckgradienten für die vorläufige Geschwindigkeit lösen

$$\frac{\partial u}{\partial t} = \nu \nabla^2 u - (u * \nabla) u \tag{1}$$

Das resultiert in einer Gleichung je Dimension:
Für die x-Komponente:

$$\frac{\partial u_x}{\partial t} = \nu \left(\frac{\partial^2 u_x}{\partial x^2} + \frac{\partial^2 u_x}{\partial y^2} \right) - \left(u_x \frac{\partial u_x}{\partial x} + u_y \frac{\partial u_x}{\partial y} \right) \tag{2}$$

Für die y-Komponente:

$$\frac{\partial u_y}{\partial t} = \nu \left(\frac{\partial^2 u_y}{\partial x^2} + \frac{\partial^2 u_y}{\partial y^2} \right) - \left(u_x \frac{\partial u_y}{\partial x} + u_y \frac{\partial u_y}{\partial y} \right) \tag{3}$$

1.1.2 Mit der Vorläufigen Geschwindigkeit kann nun die rechte Seite der Poisson Druckgleichung gelöst werden

Lösen der Rechten Seite

$$\nabla^2 p = \frac{\rho}{\Delta t} * \nabla u \tag{4}$$

$$\nabla^2 p = \frac{\rho}{\Delta t} * \left(\frac{\partial u_x}{x} + \frac{\partial u_y}{\partial y} \right) \tag{5}$$

Lösen der Gleichung Damit kann nun mit Hilfe eines Iterativen Solvers wie zb. Jacobi, die Poisson Druckgleichung gelöst werden.
Ein Schritt des Jacobi sieht dafür folgendermassen aus:

$$p_{i,j} = \frac{1}{4} (p_{i-1,j} + p_{i+1,j} + p_{i,j-1} + p_{i,j+1} - h^2 * RHS\_PPE) \tag{6}$$

Danach einfach die Randbedingungen erzwingen und wiederholen, bis die Lösung genau genug ist.

1.1.3 Vorläufige Geschwindigkeit mit der Druckveränderung anpassen

$$u_{t+1} \leftarrow u_t - \frac{\Delta t}{\rho} \nabla p \tag{7}$$

$$u_{x,t+1} = u_{x,t} - \frac{\Delta t}{\rho} \frac{\partial p}{\partial x} \tag{8}$$

$$u_{y,t+1} = u_{y,t} - \frac{\Delta t}{\rho} \frac{\partial p}{\partial y} \tag{9}$$

Jetzt haben wir die Geschwindigkeit und den Druck für diesen Zeitschritt berechnet und können mit dem Nächsten weitermachen.

2 Erklärung Code

Der Code wurde in Python, in einem JupyterNotebook geschrieben und verwendet hauptsächlich Numpy und Matplotlib.
Als erstes definiere ich die zu simulierende Domäne: Grösse, Anzahl Gitterpunkte, Gitterweite, Zeitschrittweite, Dichte, Kinematische Viskosität der Flüssigkeit und die Anzahl Iterationen für die Gerechnet werden soll.

Ich definiere einige Enums für die Positionen im Grid und die Fluid Property (Velocity, Pressure). Jeweils für die Zellen im Grid ganz oben, unten, links und rechts, sowie für die Reihen eines weiter innen.

Anschliessend definiere ich eine Klasse für das Grid, welche die Arrays für Geschwindigkeit und Druck enthält. Ich habe es aber vorallem eine Klasse gemacht, dass ich einfach durch den Aufruf einer Methode die Boundary Conditions erzwingen kann und einfach die Ergebnisse plotten kann. Auf dem Grid kann ich jetzt sehr einfach, viele Boundary Conditions erzwingen. Die Neumann Randbedingung habe ich nur für unseren spezifischen Fall implementiert, wo der Gradient am Rand null sein muss, das heisst ich kann die werte einfach gleich setzen, ohne noch etwas berechnen zu müssen.

Als nächstes definiere ich die Funktionen für die Ableitung in x, y Richtung und den Laplace Operator.
Damit definiere ich eine weitere Funktion, welche die Veränderung der Geschwindigkeit berechnet, ohne die Veränderung des Drucks zu berücksichtigen. Damit haben ich alle Tools, die ich brauche, um die Berechnung zu starten.

Jetzt kann das vorgeschlagene Schema in Abbildung 2 einfach implementiert werden.

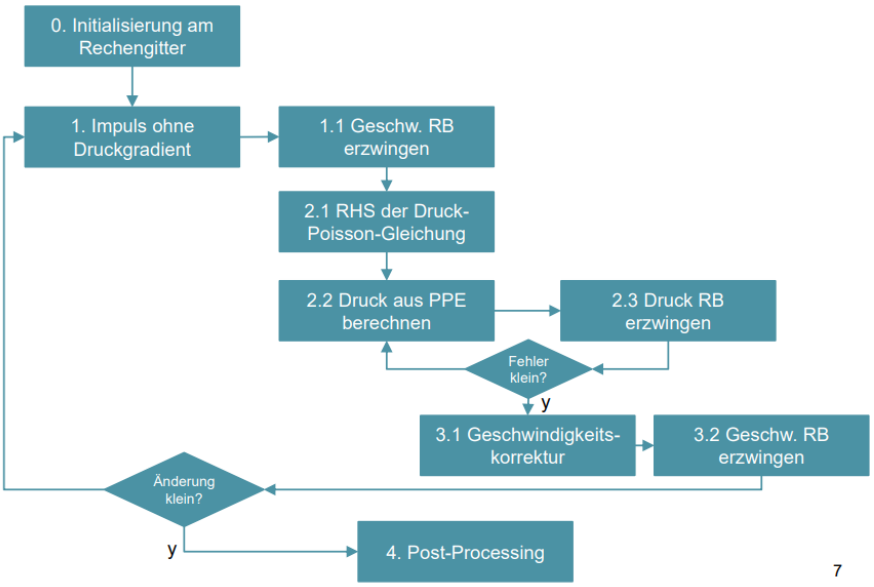

Abbildung 2: Lösungsschema

3 Instruktionen zum Code

Der Code der Simulation ist frei auf Github verfügbar.

```
git clone https://github.com/danielschafi/Lid-Driven-Cavity.git
```

Es müssen noch einige Packages installiert werden:

```
pip install -r requirements.txt
```

Der gesamte Code ist im Jupyter Notebook lid_driven_cavity.ipynb. Einfach alle Zellen im Notebook ausführen. Die Plots sind ganz unten, oder im reports Ordner zu finden. Die Parameter können einfach am Anfang des Notebooks verstellt werden.

4 Resultate

Die Ergebnisse werden in Abbildung 3 mit Geschwindigkeit und Druck dargestellt. In der oberen rechten Ecke ist der Druck höher, da die Flüssigkeit dort in die Ecke gedrückt wird. Links oben ist der Druck tiefer, da die Flüssigkeit, die da von dem Deckel mitgerissen wird einen Unterdruck hinterlässt, das von der nachströmenden Flüssigkeit wieder äufgefüllt"werden muss. Für mich sehen die Ergebnisse plausibel aus. Ich hätte jedoch in den unteren Ecken Verwirblungen erwartet.

Im zweiten Experiment in Abbildung 4 habe ich die Viskosität auf 0.001 heruntergesetzt. Die Strömung scheint in den oberen schichten recht verwirbelt zu sein. Meinem Verständnis nach macht es Sinn, dass sich die Verwirblung mit abnehmender Viskosität auf einen kleineren Bereich beschränken, da mit einer kleineren Viskosität die Bewegung eines Teilchens kleinere

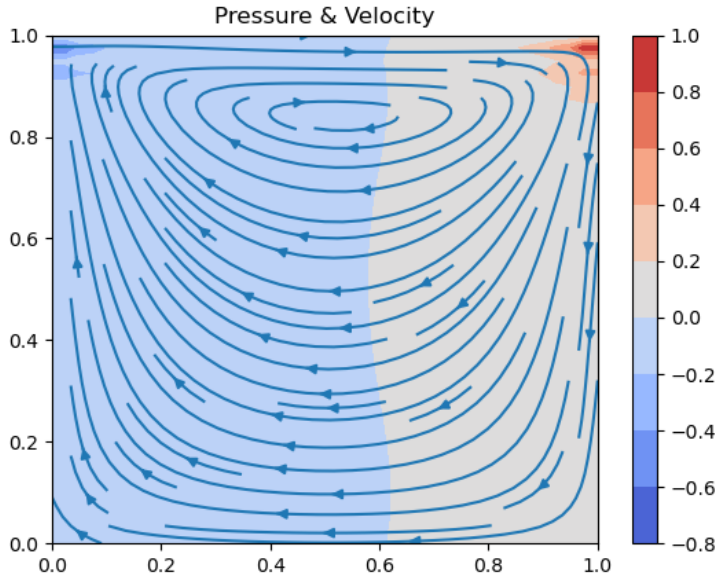

Abbildung 3: Pressure & Velocity Experiment 1

$$\rho = 1.0\frac{kg}{m^3}, \nu = 0.01\frac{m}{s^2}, \Delta t = 0.001s$$

Auswirkungen auf die benachbarten Teilchen hat.

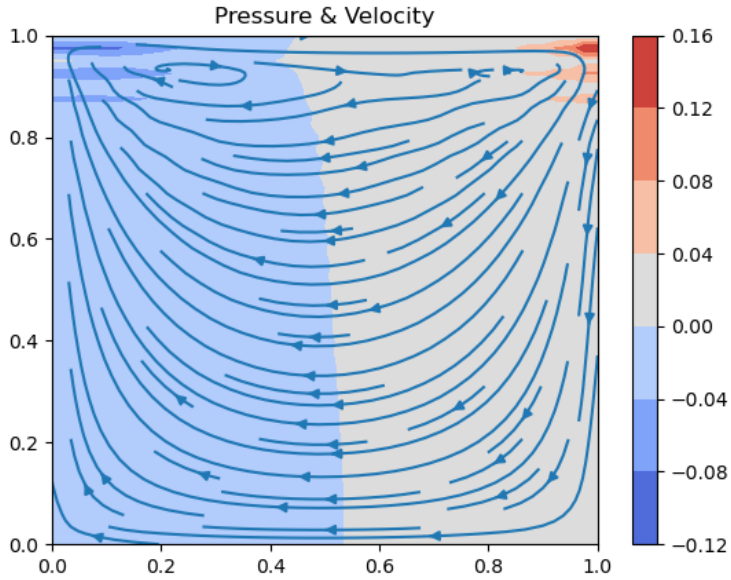

Abbildung 4: Pressure & Velocity Experiment 2

$$\rho = 1.0\frac{kg}{m^3}, \nu = 0.001\frac{m}{s^2}, \Delta t = 0.001s$$

Im dritten Experiment in Abbildung 5 habe ich die Viskosität auf 0.1 erhöht. Die Spirale ist wie erwartet grösser geworden und mehr in Richtung Mitte der Domäne gerutscht. Jetzt gibt

es auch in den unteren Ecken die Verwirblungen, die ich zuvor erwartet habe.

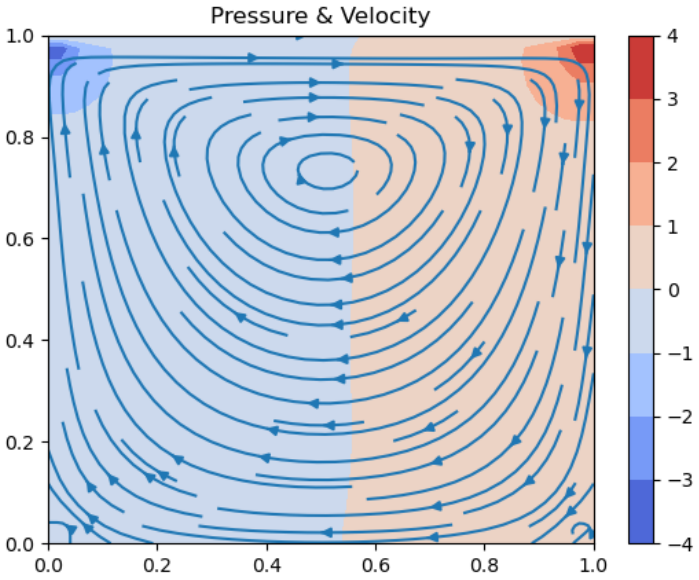

Abbildung 5: Pressure & Velocity Experiment 3

$$\rho = 1.0\frac{kg}{m^3}, \nu = 0.1\frac{m}{s^2}, \Delta t = 0.001s$$

Im vierten Experiment in Abbildung 6 habe ich die Viskosität wieder auf 0.1 gelassen, dafür aber die Dichte ρ auf 10 erhöht, um zu schauen, was für Auswirkungen das hat. Ich war etwas überrascht zu sehen, dass die Strömung exakt gleich aussieht, wie in Abbildung 5. Ich denke jetzt, dass das so ist, weil wir hier von einem inkompressiblen Fluid ausgehen und die Dichte nur im Kompressiblen Fall wirklich eine Auswirkung hat. Das macht auch Sinn, wenn wir die verwendeten Formeln betrachten. Es werden nur die Druckgradienten verwendet.

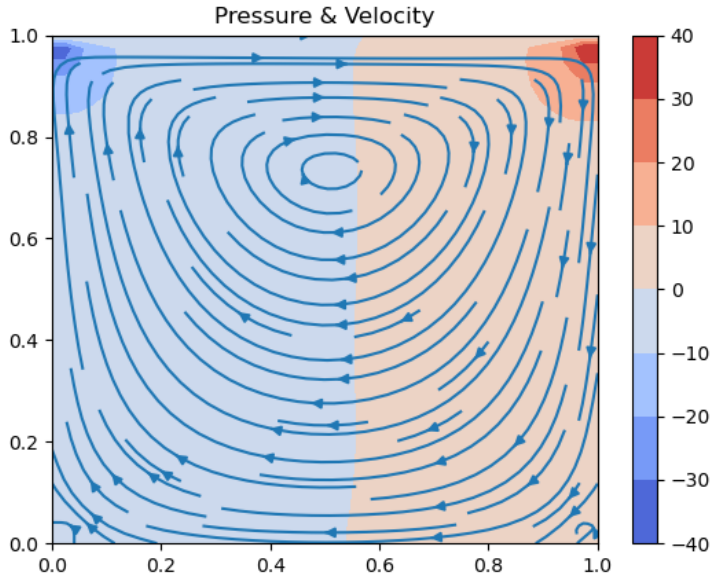

Abbildung 6: Pressure & Velocity Experiment 4

$$\rho = 10.0\frac{kg}{m^3}, \nu = 0.1\frac{m}{s^2}, \Delta t = 0.001s$$

5 Reflexion

Für mich war es ein lehrreiches Projekt. Ich konnte viel davon mitnehmen. Am Anfang war es eine kleine Challenge, die Mathematischen Formeln in Python Code umzusetzen, einfach weil wir das nicht viel gemacht haben bisher. Jetzt fühle ich mich damit wesentlich sicherer und weiss wie das zu programmieren und umzusetzten ist.

Auch für das Thema Strömungssimulation konnte ich einiges lernen, da ich mich für das Projekt intensiv mit der Materie auseinandersetzen musste. Auch das variieren der Parameter und schauen, wie sich die Strömung verändert war sehr lehrreich.

Im nachhinein denke ich, dass der Code noch etwas besser gemacht hätte werden können. Die ganzen Enums und Klassen waren etwas overkill. Es sieht gut aus und ist wahrscheinlich verständlicherer Code, als viele andere, aber eben, da es anderst als andere ist, macht es das auch etwas schwieriger die Codes zu vergleichen.

Literatur

Koehler, F. (o. J.). *Solving the navier-stokes equations in python | cfd in python | lid-driven cavity.* YouTube. Zugriff auf `https://www.youtube.com/watch?v=BQLvNLgMTQE`