A Python Implementation of the Lattice Boltzmann Method

Manxi (Maggie) Shi

Table of Contents

Introduction

Introduction

The Lattice Boltzmann Method

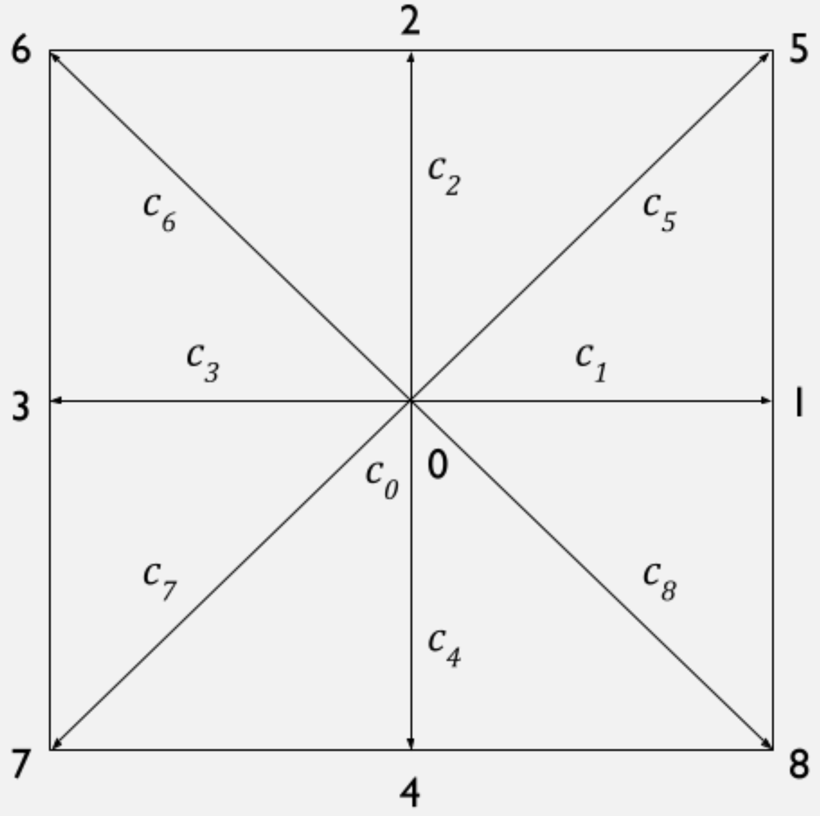
- based on a discretized Boltzmann equation
- mesoscopic approach: models fluid as the movement and collisions of particle distributions on a lattice

Advantages over Navier-Stokes Finite Element/Difference/Volume Methods

- can be parallelized more easily: all interactions in LBM are strictly local
- handles complex boundaries and multiphase scenarios well

Introduction

D2Q9 is used – 2D grid, 9 velocity unit vectors c_i



Motivation

Motivation

OpenLB and Palabos are great open source software for LBM, but offers less flexibility to modify

Easier to update or extend to different scenarios with my own code

Motivation

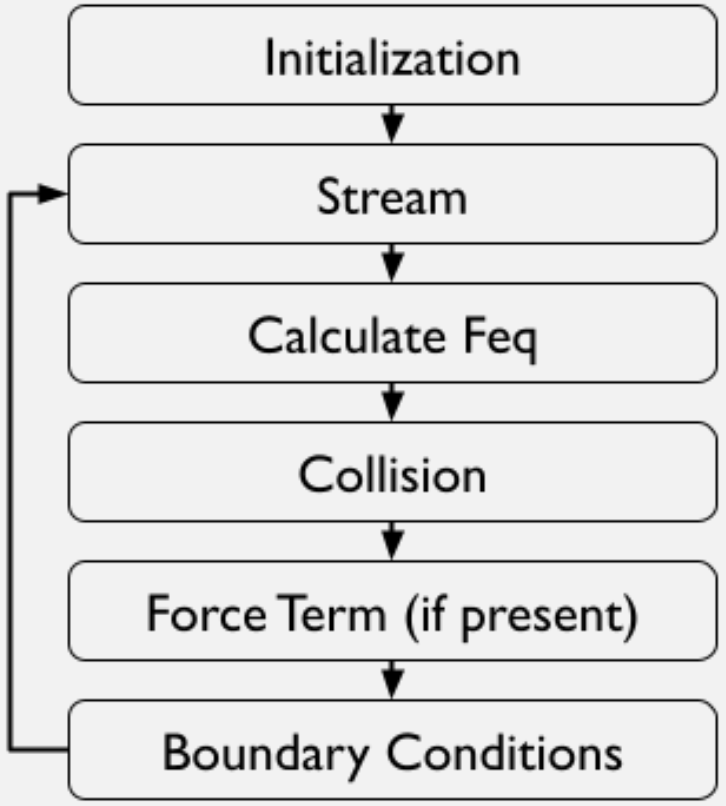
Popular Applications of LBM

- turbulent flows
- porous media
- suspension flows
- multiphase and multicomponent flows
- heat propagation

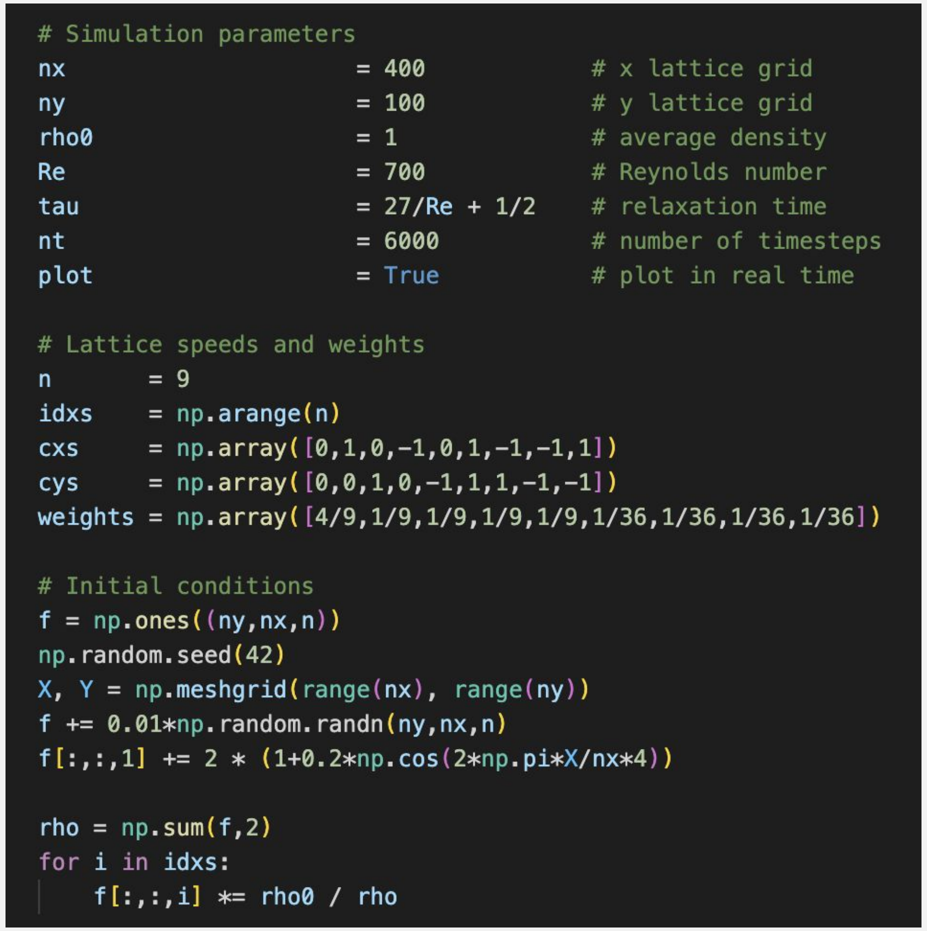
Python Implementation

Link to GitHub: https://github.com/maggies1105/LBM

Algorithm


Initialization
Stream
Calculate Feq
Collision
Force Term (if present)
Boundary Conditions

Initialization



```python
# Simulation parameters
nx                     = 400            # x lattice grid
ny                     = 100            # y lattice grid
rho0                   = 1              # average density
Re                     = 700            # Reynolds number
tau                    = 27/Re + 1/2    # relaxation time
nt                     = 6000           # number of timesteps
plot                   = True           # plot in real time

# Lattice speeds and weights
n       = 9
idxs    = np.arange(n)
cxs     = np.array([0,1,0,-1,0,1,-1,-1,1])
cys     = np.array([0,0,1,0,-1,1,1,-1,-1])
weights = np.array([4/9,1/9,1/9,1/9,1/9,1/36,1/36,1/36,1/36])

# Initial conditions
f = np.ones((ny,nx,n))
np.random.seed(42)
X, Y = np.meshgrid(range(nx), range(ny))
f += 0.01*np.random.randn(ny,nx,n)
f[:,:,1] += 2 * (1+0.2*np.cos(2*np.pi*X/nx*4))

rho = np.sum(f,2)
for i in idxs:
    f[:,:,i] *= rho0 / rho
```

Lattice grid: 400x100

9 directions of velocity unit vectors

particle density distribution function f

macroscopic density ρ

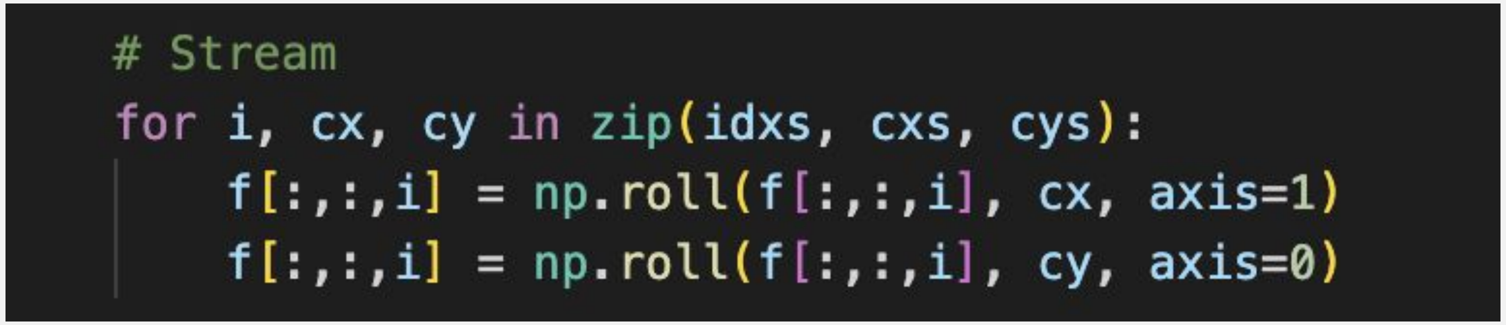
Stream

$$f_i(\mathbf{x} + \mathbf{c_i}\Delta t, t + \Delta t) = f_i(\mathbf{x}, t)$$

Streaming step takes particle distributions to the next lattice node along all 9 directions in one time step

Lattice spacing and time steps are chosen so that particles travelling with velocity $\mathbf{c_i}$ move one lattice space per time step

Python's NumPy library's *roll* function greatly speeds up and simplifies this Streaming step



```python
# Stream
for i, cx, cy in zip(idxs, cxs, cys):
    f[:,:,i] = np.roll(f[:,:,i], cx, axis=1)
    f[:,:,i] = np.roll(f[:,:,i], cy, axis=0)
```

Calculate Feq

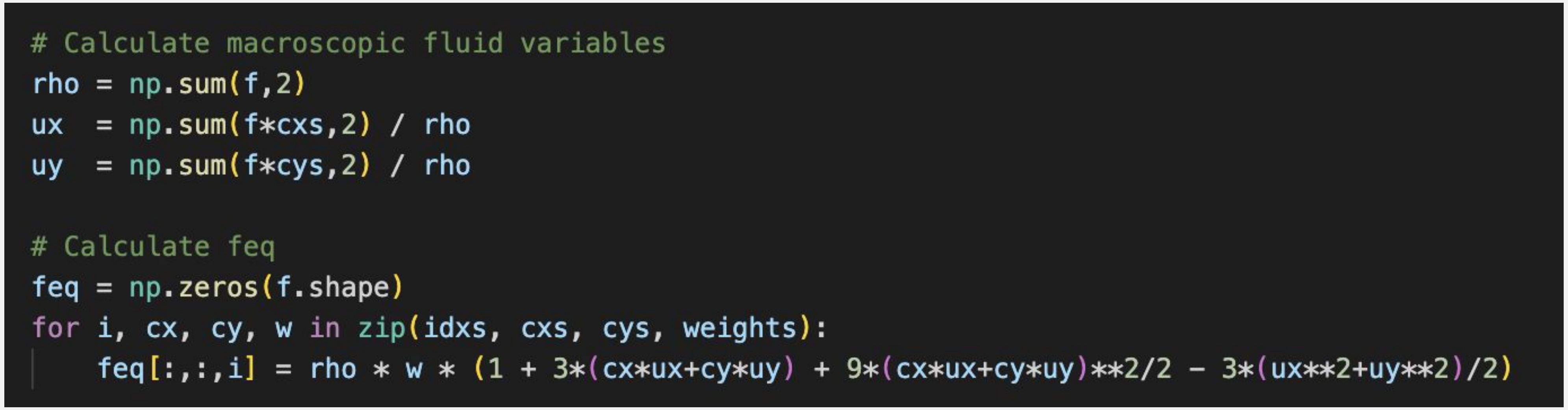
Calculate fluid variables

$$\rho = \sum_i f_i$$ macroscopic density

$$\mathbf{u} = \frac{1}{\rho}\sum_i f_i c_i$$ macroscopic velocity

Calculate equilibrium distribution f_i^{eq}

$$f_i^{eq} = \rho w_i \left[1 + 3\frac{\mathbf{e_i}\cdot\mathbf{u}}{\mathbf{c^2}} + 9\frac{(\mathbf{e_i}\cdot\mathbf{u})^2}{2\mathbf{c^4}} - 3\frac{\mathbf{u}^2}{2\mathbf{c^2}}\right]$$



```
# Calculate macroscopic fluid variables
rho = np.sum(f,2)
ux  = np.sum(f*cxs,2) / rho
uy  = np.sum(f*cys,2) / rho

# Calculate feq
feq = np.zeros(f.shape)
for i, cx, cy, w in zip(idxs, cxs, cys, weights):
    feq[:,:,i] = rho * w * (1 + 3*(cx*ux+cy*uy) + 9*(cx*ux+cy*uy)**2/2 - 3*(ux**2+uy**2)/2)
```

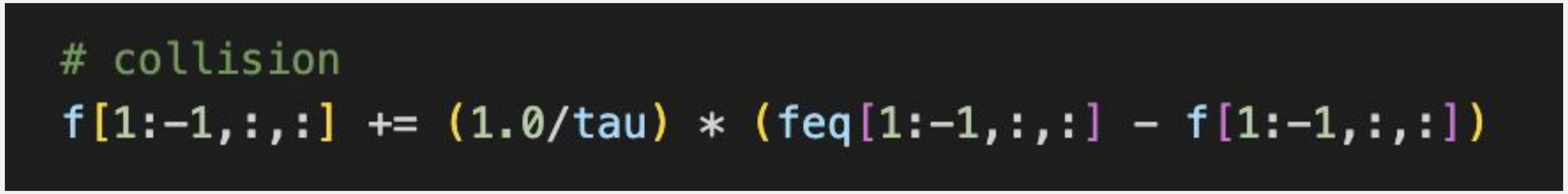
Collision

$$\Omega_i(f) = -\frac{1}{\tau}\left(f_i(\mathbf{x},t) - f_i^{eq}(\mathbf{x},t)\right)$$

Collision operator given by Bhatnagar-Gross-Kook Approximation

$$f_i(\mathbf{x}, t+\Delta t) = f_i(\mathbf{x},t) + \Omega_i[f(\mathbf{x},t)]$$

Collision step

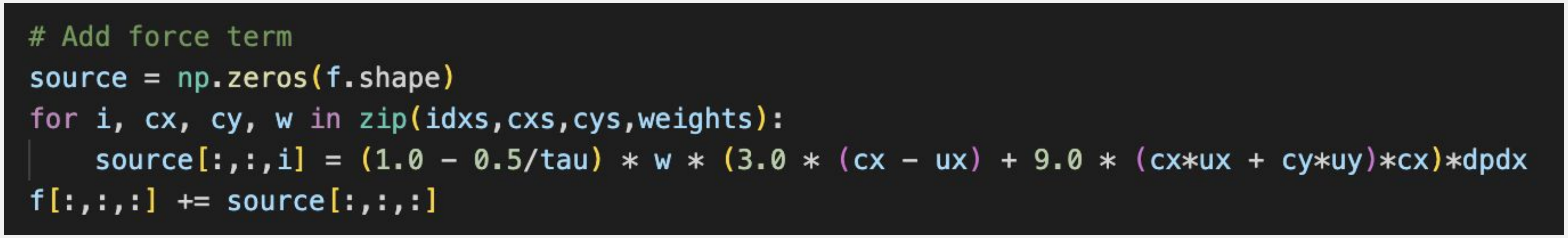


```
# collision
f[1:-1,:,:] += (1.0/tau) * (feq[1:-1,:,:] - f[1:-1,:,:])
```

Force Term (if present)

A constant presssure difference between the inlet and outlet drives Poiseuille flow in a channel

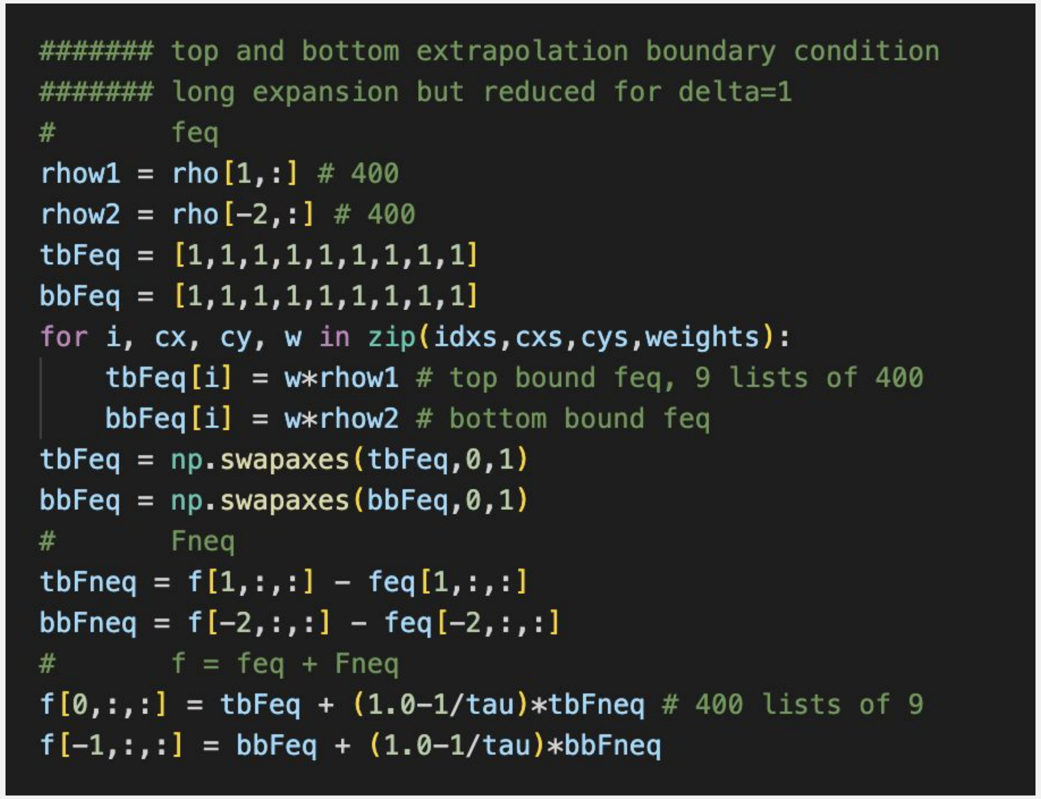
A force term, dependent on the pressure difference and added to the distribution function f, accounts for this



```
# Add force term
source = np.zeros(f.shape)
for i, cx, cy, w in zip(idxs,cxs,cys,weights):
    source[:,:,i] = (1.0 - 0.5/tau) * w * (3.0 * (cx - ux) + 9.0 * (cx*ux + cy*uy)*cx)*dpdx
f[:,:,:] += source[:,:,:]
```

Solid Wall Boundary Conditions

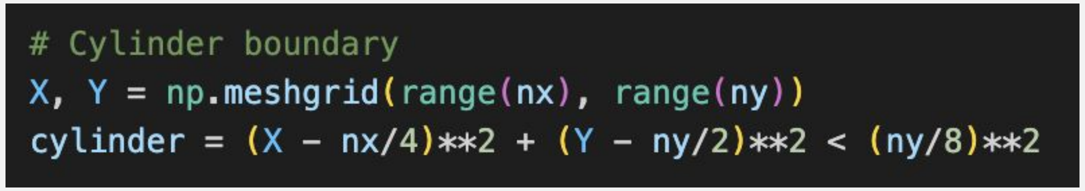
Extrapolation Scheme for Top/Bottom Walls



```
####### top and bottom extrapolation boundary condition
####### long expansion but reduced for delta=1
#       feq
rhow1 = rho[1,:] # 400
rhow2 = rho[-2,:] # 400
tbFeq = [1,1,1,1,1,1,1,1,1]
bbFeq = [1,1,1,1,1,1,1,1,1]
for i, cx, cy, w in zip(idxs,cxs,cys,weights):
    tbFeq[i] = w*rhow1 # top bound feq, 9 lists of 400
    bbFeq[i] = w*rhow2 # bottom bound feq
tbFeq = np.swapaxes(tbFeq,0,1)
bbFeq = np.swapaxes(bbFeq,0,1)
#       Fneq
tbFneq = f[1,:,:] - feq[1,:,:]
bbFneq = f[-2,:,:] - feq[-2,:,:]
#       f = feq + Fneq
f[0,:,:] = tbFeq + (1.0-1/tau)*tbFneq # 400 lists of 9
f[-1,:,:] = bbFeq + (1.0-1/tau)*bbFneq
```

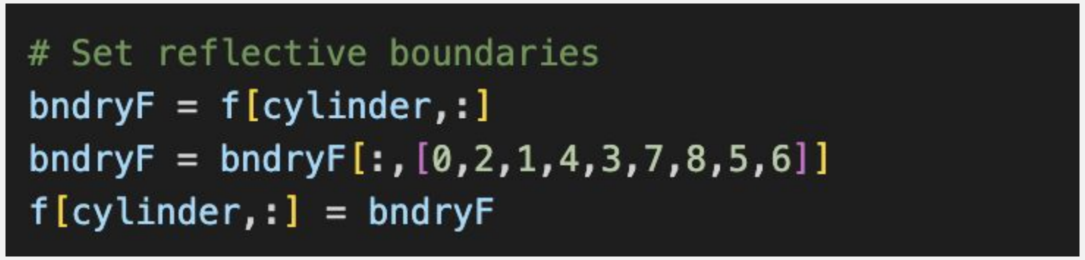
Bounce-Back Approach for Cylinder Walls

Initialize cylinder (an array of boolean values that are True inside the cylinder and FAlse everywhere else on the lattice grid



```
# Cylinder boundary
X, Y = np.meshgrid(range(nx), range(ny))
cylinder = (X - nx/4)**2 + (Y - ny/2)**2 < (ny/8)**2
```

Inside for loop: reverse all velocities at cylinder edge

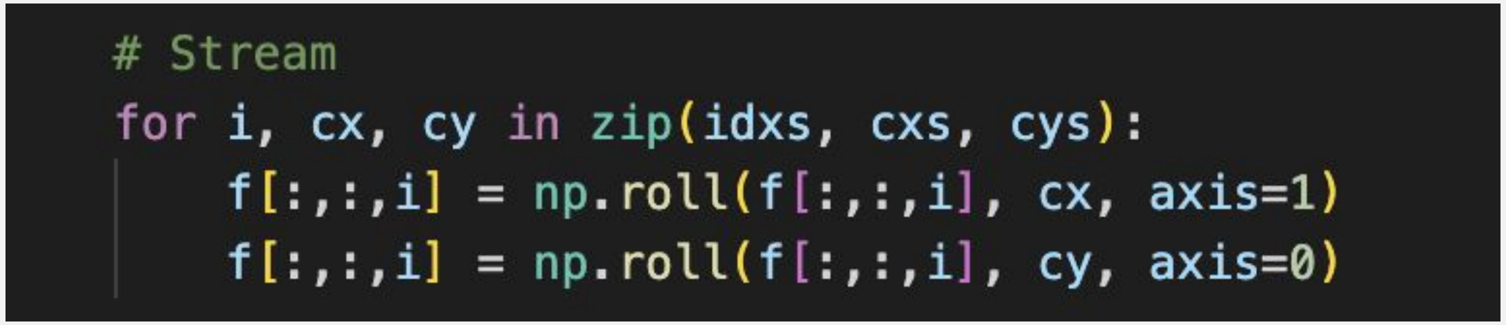


```
# Set reflective boundaries
bndryF = f[cylinder,:]
bndryF = bndryF[:,[0,2,1,4,3,7,8,5,6]]
f[cylinder,:] = bndryF
```

Inlet and Outlet Boundary Conditions: Periodic

Periodic boundary conditions for Cylinder simulation: left and right boundaries essentially connected, fluid flows out from left into right and vice versa

Automatically taken care of with NumPy *roll* in streaming step



```
# Stream
for i, cx, cy in zip(idxs, cxs, cys):
    f[:,:,i] = np.roll(f[:,:,i], cx, axis=1)
    f[:,:,i] = np.roll(f[:,:,i], cy, axis=0)
```

Inlet and Outlet Boundary Conditions: Velocity

Velocity boundary conditions at the inlet and outlet are another possibility

Initialize variables:

define a constant pressure difference between the inlet and outlet

define densities at the inlet and outlet accordingly (and indirectly fluid velocity)

```
dpdx = 8*nu*u_max/Ny**2
rho_outlet = 1
rho_inlet = 3*(Nx-1)*dpdx+rho_outlet
```

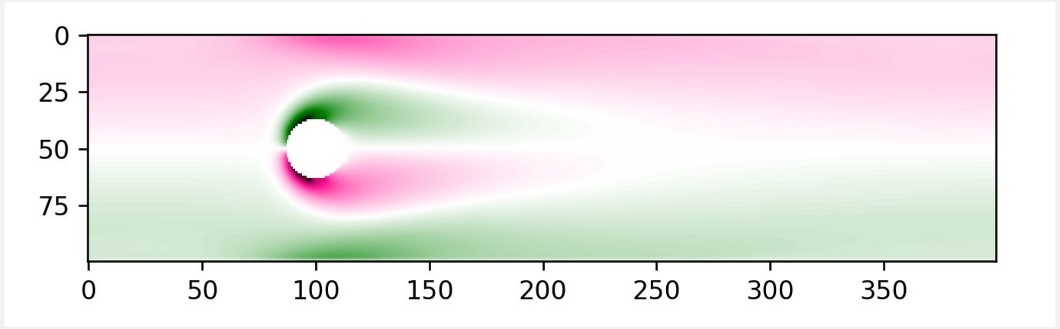
Inside for loop: apply inlet/outlet boundary conditions

```
for i, cx, cy, w in zip(idxs, cxs, cys, weights):
    f[0,:,i] = w*(rho_inlet+3*(cx*ux[Nx-1,:] + cy*uy[Nx-1,:])) + (f[Nx-1,:,i] - feq[Nx-1,:,i])
    f[Nx-1,:,i] = w*(rho_outlet + 3*(cx*uy[1,:] + cy*uy[1,:])) + (f[1,:,i] - feq[1,:,i])
```

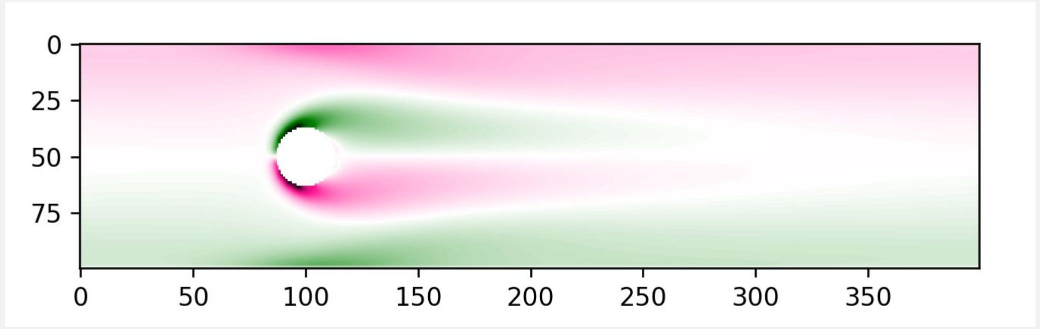
Results: 2 Benchmark Tests

Flow around a Cylinder in a Walled Channel

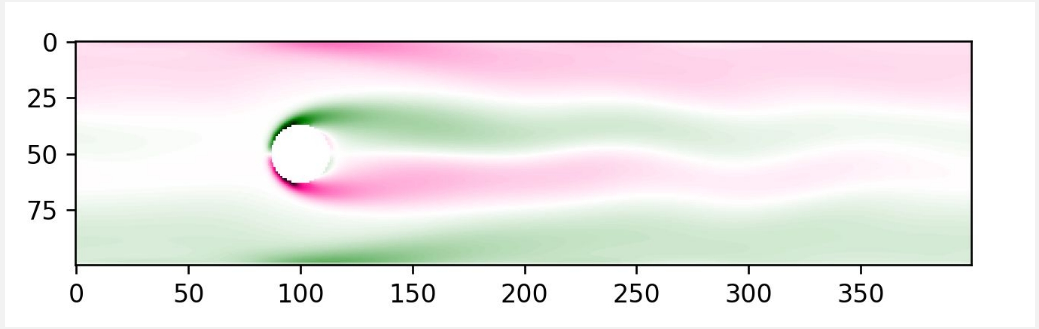
Re 60



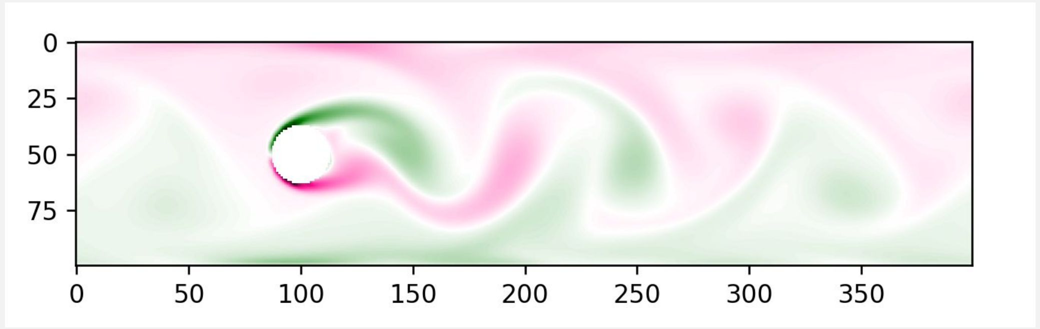
Re 120

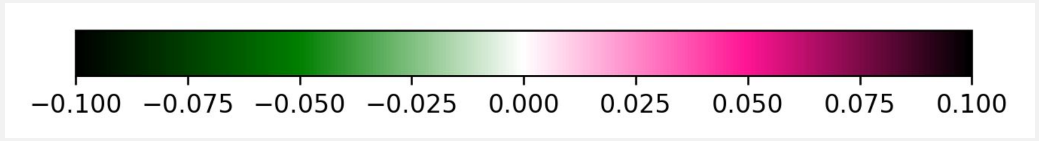


Re 180



Re 290

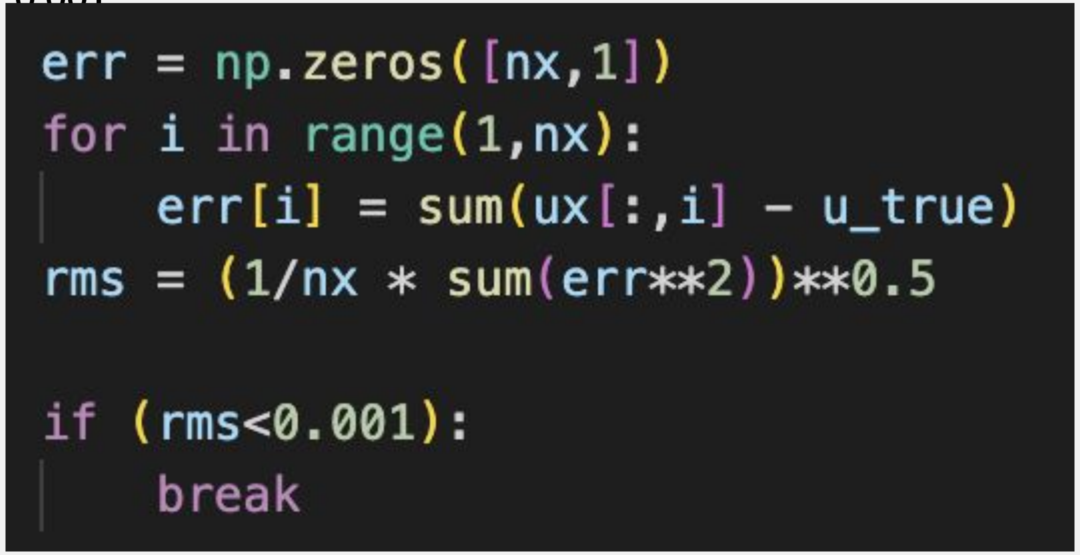




Poiseuille Flow

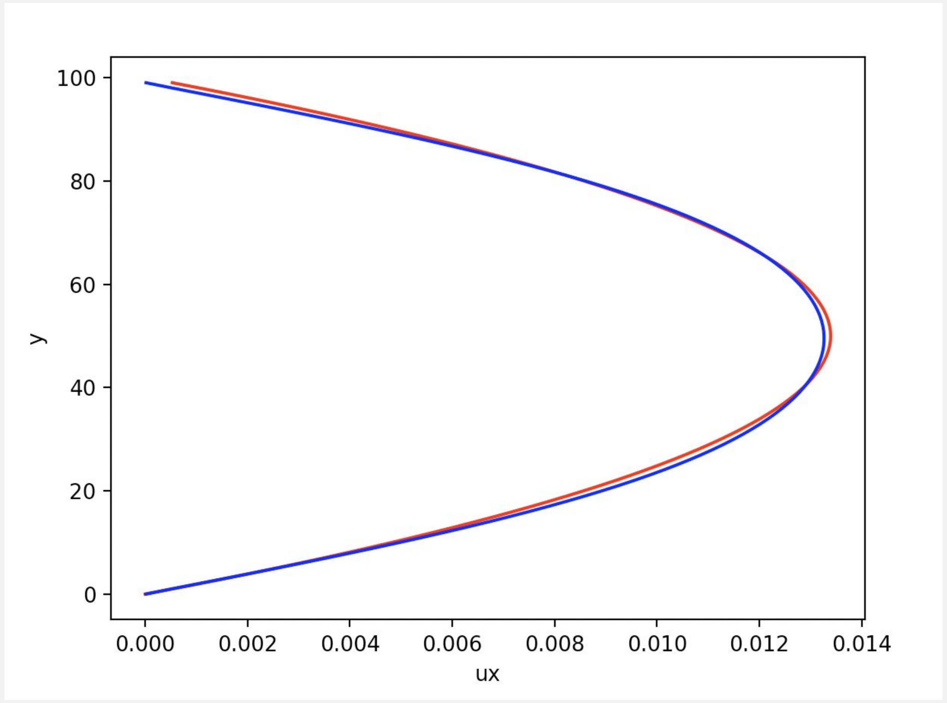
Poiseuille flow is a laminar, pressure-induced flow that naturally develops in a long pipe

Simulation stops when the root mean squared error between computational and analytical solution is within 0.001



```
err = np.zeros([nx,1])
for i in range(1,nx):
    err[i] = sum(ux[:,i] - u_true)
rms = (1/nx * sum(err**2))**0.5

if (rms<0.001):
    break
```



Red: analytical

Blue: simulation

Conclusion and Future Work

Conclusion

Results from both benchmark tests produced expected results

- Flow around cylinder: Karman vortex street at high Re, no flow separation at low Re
- Poiseuille flow: parabolic velocity profile, close to analytical solution

Future Work

Extend 2D model to 3D

Parallelize code for faster computation on a GPU or use MPI with Python

Incorporate Immersed Boundary Model with LBM to model deformable boundaries

Application: modelling movement and deformation of red blood cells in capillaries

Works Cited

Chen, S., Martínez, D., & Mei, R. (1996). On boundary conditions in lattice Boltzmann methods. Physics of Fluids, 8(9), 2527–2536.

Boltzmann, L. Lectures on Gas Theory; Courier Corporation: Chelmsford, MA, USA, 2012.

Shan X., Chen H. Lattice Boltzmann model for simulating flows with multiple phases and components. Phys. Rev. E. 1993;47:1815–1819. doi: 10.1103/PhysRevE.47.1815

Mohamad, A. A. Applied lattice Boltzmann method for transport phenomena, momentum, heat and mass transfer. Canadian Journal of Chemical Engineering, vol. 85, no. 6, Dec. 2007, p. 946. Gale Academic OneFile,

Frisch U., Hasslacher B., Pomeau Y. Lattice Gas Automata for the Navier-Stokes Equation. Physical Review Letters, vol. 56, no. 14, April 1986, p. 1505.

Dalcin L.D., Paz R.R., Kler P.A., & Cosimo A., 2011. Parallel distributed computing using python, Adv. Water Resour., vol. 34, no. 9, p. 1124–1139.

Schornbaum F. & Rüde U., 2016. Massively parallel algorithms for the lattice Boltzmann method on nonuniform grids, SIAM J. Scient. Comput., vol. 28, no. 2, p. 96–C126.

Higuera F.J. & Jimenez J., 1989. Boltzmann approach to lattice gas simulations, EPL (Europhys. Lett.), vol. 9, no. 7, p. 663 doi:10.1209/0295-5075/9/7/009.

Versteeg H.K., Malalasekera W., 2007. An Introduction to Computational Fluid Dynamics: The Finite Volume Method; Pearson Education: London, UK.

Girault, V., Raviart P.A., Finite Element Methods for Navier-Stokes Equations, Springer Berlin, Heidelberg, 1986

Chen, S., Wang, Z., Shan, X., Doolean, G. Lattice Boltzmann computational fluid dynamics in three dimensions. J Stat Phys 68, 379–400 (1992).

Boubakran M. & Mirzaee I., 2018. Numerical simulation of external flow around cylinder using improved lattice Boltzmann method. The Journal of Engineering, vol. 2018, no. 4, p. 248-253.

T. Krüger, H. Kusumaatmaja, A. Kuzmin, O. Shardt, G. Silva, E. M. Viggen. (2016). The Lattice Boltzmann Method - Principles and Practice. 10.1007/978-3-319-44649-3.

Guo, Zhaoli & Zheng, Chuguang & Shi, Baochang. (2002). An extrapolation method for boundary conditions in lattice Boltzmann method. Physics of Fluids. 14. 10.1063/1.1471914.

Cheng-Hsien Lee, Zhenhua Huang & Yee-Meng Chiew. (2015). An extrapolation-based boundary treatment for using the lattice Boltzmann method to simulate fluid- particle interaction near a wall, Engineering Applications of Computational Fluid Mechanics, 9:1, 370-381, DOI: 10.1080/19942060.2015.1061554

M. Shankar, S. Sundar. (2009). Asymptotic analysis of extrapolation boundary conditions for LBM. Computers & Mathematics with Applications. Vol. 57, Issue 8, Pg. 1313-1323.

Thank you!