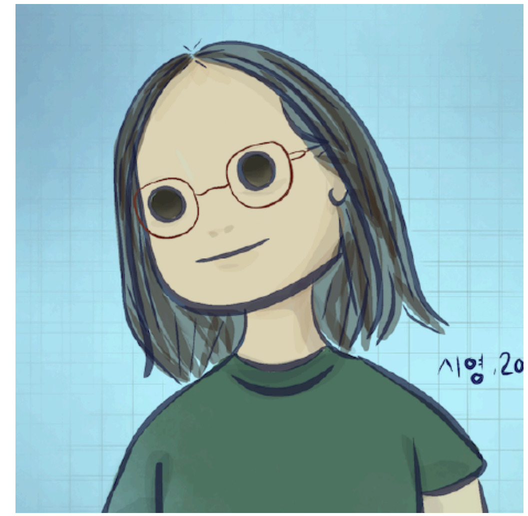
1D Viscous Fluid Flow Data Analysis Using Burgers' Equation

Siyoung B

About me

I am Siyoung *[she-young]*

- A developer, loves Clojure and loves to fidget all day
  - cat, hiking, cycling, walking, knitting & sewing
- Likes to look up to see the night sky
- Studied astrophysics in undergrad
  - participated research of simulating binary stars' collision



Fluid Dynamics & CFD

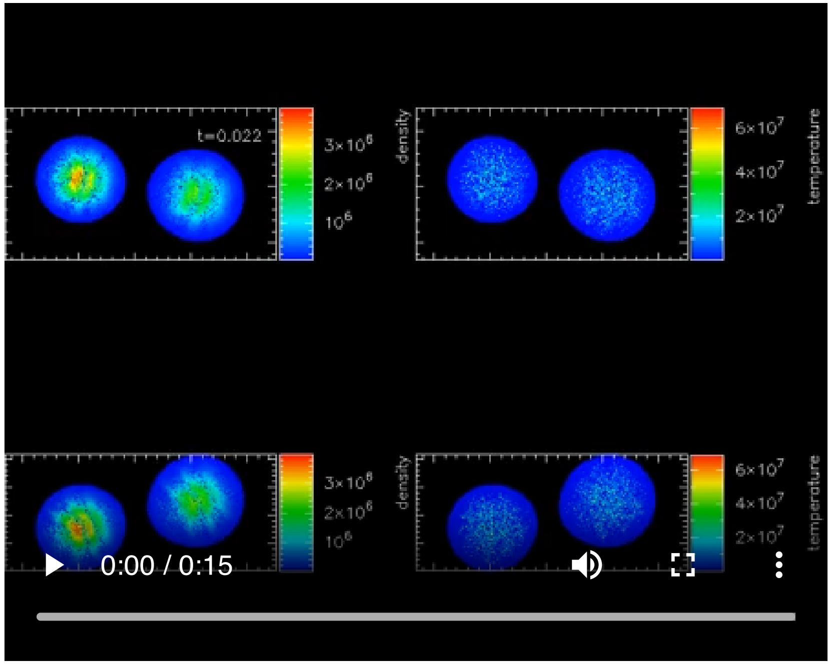
Fluid Dynamics

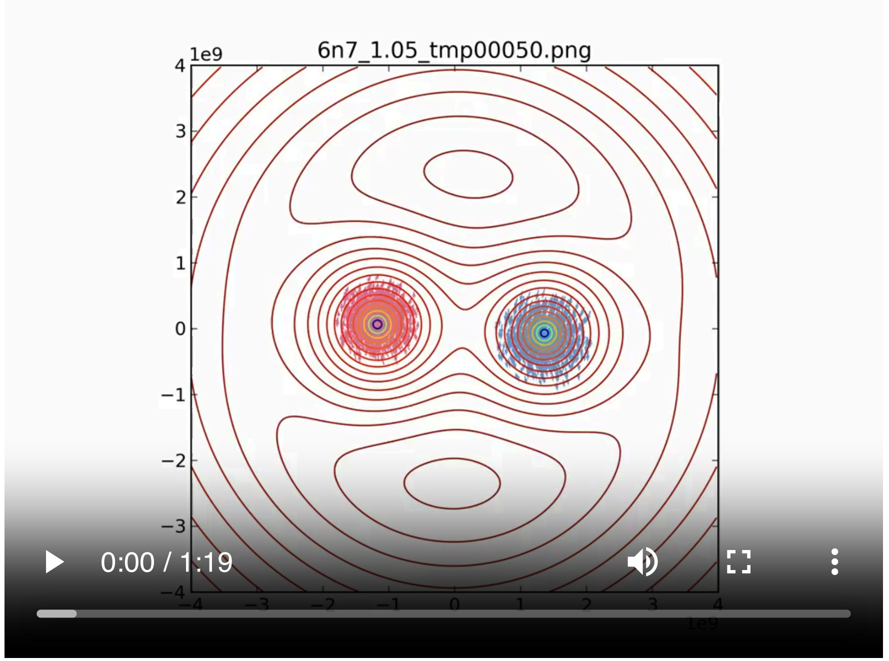
- The physics of how liquids and gases move and flow
- **Many Variables**: Involves factors like friction, pressure, heat, and momentum etc.
- **Complex Phenomena**: Includes complicated movements like turbulence
- **Crucial Understanding**: Important for fields like airplane design, healthcare, and plumbing and even more!

Computational Fluid Dynamics (CFD)

- Uses computers and math to simulate and predict fluid movement
- ***Clojure Gap***: No CFD tools using Clojure yet(?)
- ***My Initiative***: This project aims to create CFD tools using Clojure

Past Research





Where to start

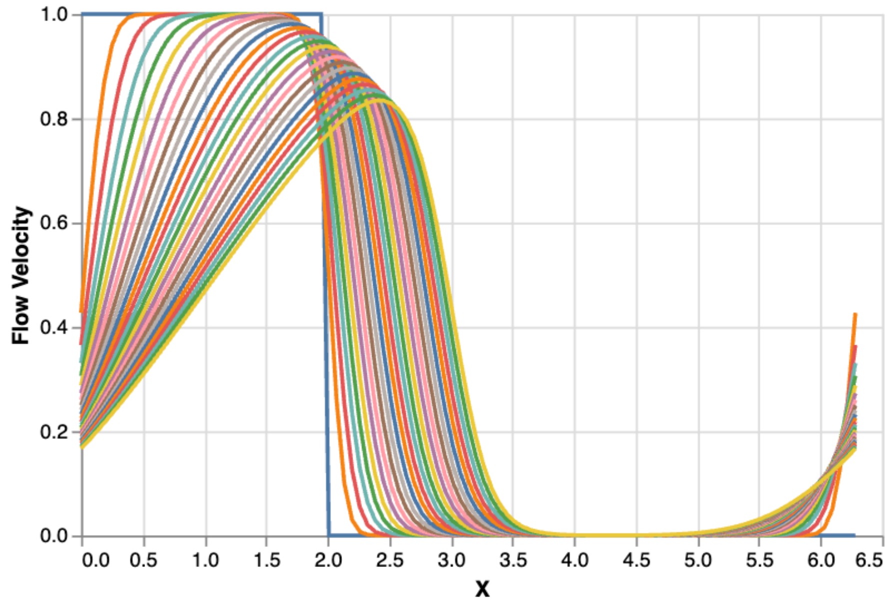
- **Past CFD Experience:** As an end user, preparing, running then analyzing out of existing tools in astrophysics research
- **Knowledge Refresh**: Not much of formal CFD knowledge, and dated since university

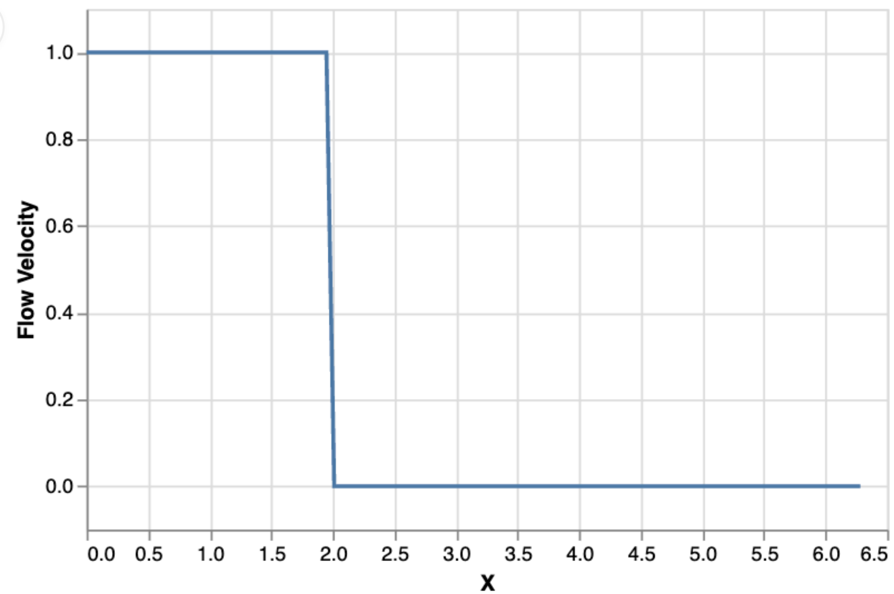
Learning Resource: CFD Python

- Utilizing Prof. Lorena Barba's "CFD Python" materials for relearning foundational CFD
- Developed at Boston University with Python code examples for teaching
- Broken down into 12 steps of learning materials to start with a simplified concept
- CFD Python in Clojure
  - Using "CFD Python" as a guide to implement CFD in Clojure
  - Currently in progress

1D Shock Interaction & Evolution

initial condition - Step function $u(x,0) = \begin{cases} 1, \text{if } x < 2.0 \\ 0, \text{otherwise} \end{cases}$



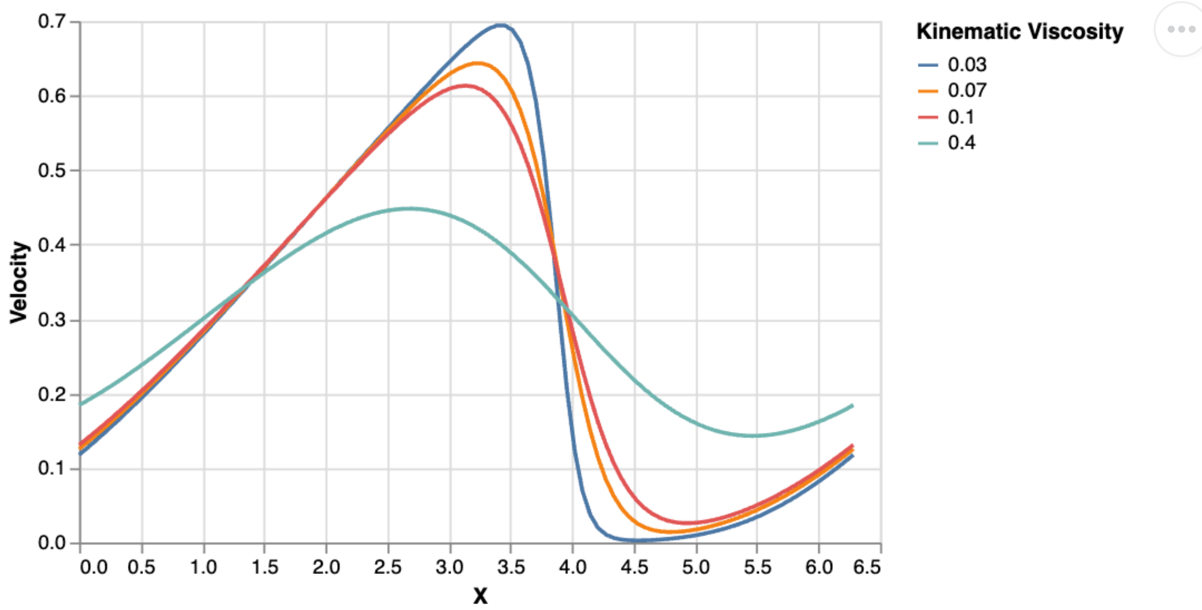


Time interval idx: 0

- **Non-linear convection**: Faster fluid tends to “bunch up” with slower fluid in a complex way, steepening changes
  - **Non-linear**: Effects not always simple nor directly proportional
  - **Convection**: Movement affected by some changes i.e. heat, flow etc.
- **Viscosity**: The “stickiness” of the fluid that resists flow and smooths out speed differences

Effect of Viscosity on Step Structure

- **Viscosity** = *"Stickiness"*
- $\nu(= viscosity) = [0.03, 0.07, 0.1, 0.4]$
- Lower viscosity → sharper shock features
- Higher viscosity → increased smoothing of shock features



Implementation in Clojure 1

Initial setup

```clojure
(defn create-initial-x [{:keys [x-start x-stop nx]}]
 (let [arr  (float-array nx)
       step (/ (- x-stop x-start) (dec nx))]
   (dotimes [i nx]
     (aset arr i (float (* i step))))
   arr))
```

```clojure
(defn create-initial-fluid-velocity
  [{:keys [array-x condition-fn] :as _params}]
  (let [nx      (alength array-x)
        array-u (float-array nx)]
    (dotimes [i nx]
      (let [x-val (aget array-x i)
            u-val (float (condition-fn x-val))]
        (aset array-u i u-val)))
    array-u))
```

- Chose Java primitive arrays(float-array) for performance
- Mutable, non-persistent approach for speed and memory efficiency for future large-scale simulation
- **Pros**: Low memory overhead, Fast access and updates, Better suited for large numerical grid
- **Cons**: Breaks Clojure’s idiomatic immutability, manual memory handling and index tracking, harder to debug and reason functionality

Implementation in Clojure 2

Python → Clojure

```python
u = numpy.ones(nx)
u[int(.5 / dx):int(1 / dx + 1)] = 2
un = numpy.ones(nx)

for n in range(nt):
    un = u.copy()
    for i in range(1, nx):
        u[i] = un[i] - c * dt / dx * (un[i] - un[i-1])
```

→

```clojure
(def arr-u (create-initial-fluid-velocity init-params))

(defn linear-convection-mode [idx arr-u un {:keys [c dt dx] :as _init-
  (aset arr-u (inc idx)
        (float (- (aget un (inc idx)) (* c (/ dt dx) (- (aget un (inc

(defn update-u [arr-u {:keys [c dt dx mode] :as params}]
  (let [un        (float-array arr-u)
        update-fn (case mode
                    :linear-convection linear-convection-mode
                    :burgers burgers-mode
                    other-modes)]
   (dotimes [idx (dec nx)]
     (update-fn idx arr-u un params))
   arr-u))

(defn simulate [arr-u {:keys [nt] :as init-params}]
  (loop [n 0]
    (if (= n nt)
      arr-u
      (do (update-u arr-u init-params) (recur (inc n))))))
```

Implementation in Clojure 3

Mathematical Equation → Computational Equation

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}$$

→ *discretization, forward difference for time, backward difference for space etc….* →

$$u_i^{n+1} = u_i^n - u_i^n\frac{\Delta t}{\Delta x}(u_i^n - u_{i-1}^n) + \nu\frac{\Delta t}{\Delta x^2}(u_{i+1}^n + u_{i-1}^n - 2u_i^n)$$

↓

```clojure
(defn get-burgers-arr-un [un-i+1 un-i un-i-1 {:keys [nu dx dt] :as params}]
  (float (+ un-i
            (- (* un-i dt (/ 1 dx) (- un-i un-i-1)))
            (* nu dt (/ 1 (* dx dx)) (+ un-i+1 (- (* 2 un-i)) un-i-1)))))
```

What's next

- Extend from 1D to 2D/3D
- Introduce pressure terms to meet Navier-Stokes equations
- Add boundary conditions and validation checks
- Explore further to implement w/ Clojure's functional and idiomatic way
- Scale up and test

Thank you :)

Questions?

---

source: notebooks/conferences/scinoj_light_1/siyoung_talk.clj