Differentiable Stable Fluid Solver using TensorFlow

PRIM Project

Sammy Rasamimanana

February 27, 2023

Table of Contents

1. Introduction
2. Stable fluid solver
3. Results
4. Differentiable physics in TensorFlow
5. Discussions and Future work

Introduction

Navier-Stokes equations

A fluid whose density ρ and temperature are nearly constant is described by a velocity field $u : \mathbb{R}^N \longrightarrow \mathbb{R}^N$ and a pressure field $p : \mathbb{R}^N \longrightarrow \mathbb{R}$.

Navier-Stokes equations

$$\nabla \cdot u = 0$$

$$\frac{\partial u}{\partial t} = -(u \cdot \nabla)u - \frac{1}{\rho}\nabla p + \nu \nabla^2 u + f$$

Resolution method

Too hard to analytcally solve...

Marching through each step

$$\nabla \cdot u_3 = 0$$

$$\frac{\partial u_4}{\partial t} = -(u_1 \cdot \nabla) u_1 - \frac{1}{\rho} \nabla p + \nu \nabla^2 u_2 + f$$

Problem: how to ensure $\nabla \cdot u_3 = 0$ ... ?

Projection operator

$$\nabla \cdot u_3 = 0 \text{ and } \frac{\partial u_4}{\partial t} = -(u_1 \cdot \nabla)u_1 - \frac{1}{\rho}\nabla p + \nu \nabla^2 u_2 + f$$

Problem: how to ensure $\nabla \cdot u_3 = 0$ ... ?

Helmholtz-Hodge Decomposition

Any vector field $w : \mathbb{R}^3 \longrightarrow \mathbb{R}^3$ can be decomposed into

$$w = u + \nabla q$$

where $u : \mathbb{R}^3 \longrightarrow \mathbb{R}^3, \nabla \cdot u = 0$ and $q : \mathbb{R}^3 \longrightarrow \mathbb{R}$

Projection operator

Any vector field w can be decomposed into a divergence-free field and a gradient

$$w = u + \nabla q$$

Define a projection operator and apply it to Navier-Stokes

$$\mathbb{P} : \mathbb{R}^3 \longrightarrow \mathbb{R}^3$$

$$w = u + \nabla q \longmapsto \mathbb{P}(w) = u = w - \nabla q$$

q can be recovered with a Poisson equation

$$\nabla \cdot w = \underbrace{\nabla \cdot u}_{0} + \nabla \cdot \nabla q \iff \nabla \cdot w = \nabla^2 q$$

Projection operator

By applying $\mathbb{P}$ to Navier-Stokes, we keep $\nabla \cdot u = 0$ implicit.

$$\mathbb{P}\left(\frac{\partial u}{\partial t}\right) = \mathbb{P}\left(-(u \cdot \nabla)u - \frac{1}{\rho}\nabla p + \nu \nabla^2 u + f\right)$$

$$\frac{\partial u}{\partial t} = \mathbb{P}\left(-(u \cdot \nabla)u\right) - \underbrace{\mathbb{P}\left(\frac{1}{\rho}\nabla p\right)}_{0} + \mathbb{P}\left(\nu \nabla^2 u\right) + \mathbb{P}\left(f\right)$$

$$\frac{\partial u}{\partial t} = \mathbb{P}\left(-(u \cdot \nabla)u + \nu \nabla^2 u + f\right)$$

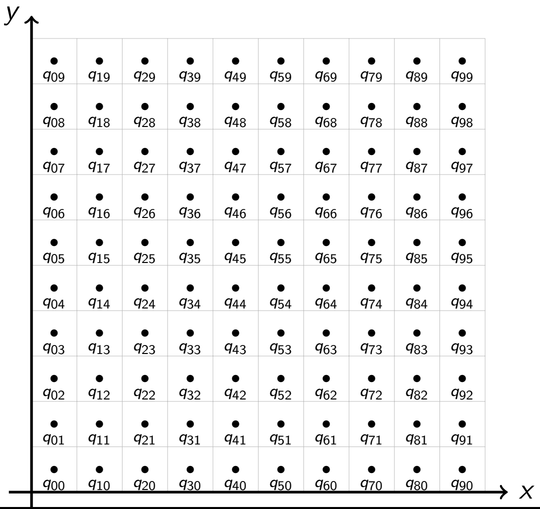
Roadmap

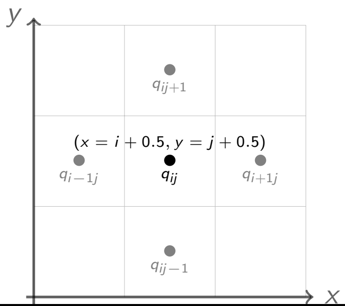
Goals of the project

- Implement a 2D stable fluid solver in Python
- Making it differentiable using TensorFlow library

Stable fluid solver

Stable fluid solver: Model





Stable fluid solver: Data structure

y

u_{09}	u_{19}	u_{29}	u_{39}	u_{49}	u_{59}	u_{69}	u_{79}	u_{89}	u_{99}
u_{08}	u_{18}	u_{28}	u_{38}	u_{48}	u_{58}	u_{68}	u_{78}	u_{88}	u_{98}
u_{07}	u_{17}	u_{27}	u_{37}	u_{47}	u_{57}	u_{67}	u_{77}	u_{87}	u_{97}
u_{06}	u_{16}	u_{26}	u_{36}	u_{46}	u_{56}	u_{66}	u_{76}	u_{86}	u_{96}
u_{05}	u_{15}	u_{25}	u_{35}	u_{45}	u_{55}	u_{65}	u_{75}	u_{85}	u_{95}
u_{04}	u_{14}	u_{24}	u_{34}	u_{44}	u_{54}	u_{64}	u_{74}	u_{84}	u_{94}
u_{03}	u_{13}	u_{23}	u_{33}	u_{43}	u_{53}	u_{63}	u_{73}	u_{83}	u_{93}
u_{02}	u_{12}	u_{22}	u_{32}	u_{42}	u_{52}	u_{62}	u_{72}	u_{82}	u_{92}
u_{01}	u_{11}	u_{21}	u_{31}	u_{41}	u_{51}	u_{61}	u_{71}	u_{81}	u_{91}
u_{00}	u_{10}	u_{20}	u_{30}	u_{40}	u_{50}	u_{60}	u_{70}	u_{80}	u_{90}

x

$$u = \begin{pmatrix} u_{00} & u_{10} & \cdots & u_{n0} \\ u_{01} & u_{11} & \cdots & u_{n1} \\ \vdots & \vdots & \ddots & \vdots \\ u_{0n} & u_{1n} & \cdots & u_{nn} \end{pmatrix}$$

Data structure 1

$$u_x = \begin{pmatrix} u_{x,00} \\ u_{x,10} \\ \vdots \\ u_{x,n0} \\ u_{x,n1} \\ \vdots \\ u_{x,nn} \end{pmatrix}, u_y = \begin{pmatrix} u_{y,00} \\ u_{y,10} \\ \vdots \\ u_{y,n0} \\ u_{y,n1} \\ \vdots \\ u_{y,nn} \end{pmatrix}$$

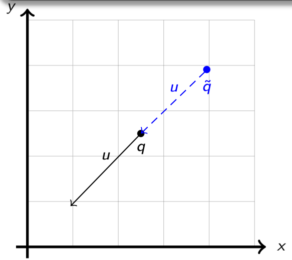
Data structure 2

Stable fluid solver: Advection

$$\frac{\partial u}{\partial t} = \mathbb{P}\left(-(u \cdot \nabla)u + \nu\nabla^2 u + f\right)$$

Backtracing - Method of characteristics

$$u(x, y, t + \Delta t) = u(x - \Delta t u_x(x, y, t), y - \Delta t u_y(x, y, t), t)$$



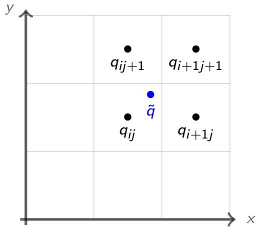

```python
def advectCentered(f,u,v,sizeX,sizeY,coords_x,coords_y,dt,offset,d):
    traced_x = np.clip(coords_x - dt*u, offset+0.5*d, offset+(sizeX-0.5)*d)
    traced_y = np.clip(coords_y - dt*v, offset+0.5*d, offset+(sizeY-0.5)*d)
    new_grid = []
    for it in range(len(traced_x)):
        new_grid.append(sampleAt(traced_x[it],traced_y[it],f,sizeX,sizeY,
                        offset,d))
    return np.array(new_grid)
```

Stable fluid solver: Advection

$$\frac{\partial u}{\partial t} = \mathbb{P}\left(-(u\cdot\nabla)u + \nu\nabla^2 u + f\right)$$

$\tilde{q}$ is not necessarely centered

Backtraced value is bilinearly interpolated using the stencil.




```python
def sampleAt(x,y,data,sizeX, sizeY, offset,d):
    _x = (x-offset)/d - 0.5
    _y = (y-offset)/d - 0.5
    i0 = np.clip(np.floor(_x),0,sizeX-1)
    j0 = np.clip(np.floor(_y),0,sizeY-1)
    i1 = np.clip(i0+1,0,sizeY-1)
    j1 = np.clip(j0+1,0,sizeY-1)

    p00 = data[ int(indexTo1D(i0,j0,sizeX))]
    p01 = data[ int(indexTo1D(i0,j1,sizeX))]
    p10 = data[ int(indexTo1D(i1,j0,sizeX))]
    p11 = data[ int(indexTo1D(i1,j1,sizeX))]

    t_i0 = (offset + (i1+0.5)*d -x)/d
    t_j0 = (offset + (j1+0.5)*d -y)/d
    t_i1 = (x - (offset + (i0+0.5)*d))/d
    t_j1 = (y - (offset + (j0+0.5)*d))/d

    return t_i0*t_j0*p00 + t_i0*t_j1*p01 + t_i1*t_j0*p10 + t_i1*t_j1*p11
```

Stable fluid solver: Poisson equation

Problem: how to solve $\nabla^2 q = w$?

Discretize using forward and backward derivatives

$$w_{ij} = \frac{q_{i+1j} + q_{i-1j} + q_{ij+1} + q_{ij-1} - 4q_{ij}}{h^2} \text{ by taking } dx = dy = h$$

Two possibilities: Gauss-Seidel updates or Laplacian matrix

```python
def gauss_seidel_solver(w,q,a,b,n_iter):
    dim_x, dim_y = u1.shape
    for it in range(n_iter):
        for i in range(1,dim_x-1):
            for j in range(1,dim_y-1):
                q[i,j] = (w[i,j] + a*(w[i+1, j] + q[i, j+1] + q[i
                        -1, j] + q[i,j-1]))/b
        u1 = set_solid_boundary(u1, 0)
    return u1
```

$$\begin{matrix}
-4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 1
\end{matrix}$$

Stable fluid solver: Diffusion

$$\frac{\partial u}{\partial t} = \mathbb{P}\left(-(u \cdot \nabla)u + \nu\nabla^2 u + f\right)$$

Implicit method: find $\tilde{u}$ such that $(I - \Delta t \nu \nabla^2)\tilde{u} = u$.

It is possible to use the Poisson solver

Apply the following transformations: $\frac{-4}{h^2} \longrightarrow \frac{1 + 4\nu\Delta t}{h^2}$ and $\frac{1}{h^2} \longrightarrow \frac{-\nu\Delta t}{h^2}$.

Stable fluid solver: Projection

$$\frac{\partial u}{\partial t} = \mathbb{P}\left(-(u \cdot \nabla)u + \nu\nabla^2 u + f\right)$$

1. Find q such that $\nabla^2 q = \nabla \cdot u$ (Poisson equation)
2. $u \longleftarrow u - \nabla q$

```python
def project(u,v,sizeX,sizeY,mat,h,boundary_func):
    _u,_v = set_boundary(u,v, sizeX, sizeY,boundary_func)
    p = solvePressure(_u,_v,sizeX,sizeY,h, mat)
    gradP_u = []
    gradP_v = []
    for j in range(sizeY):
        for i in range(sizeX):
            if (i>1) and (i < sizeX-1):
                gradP_u.append(p[indexTo1D(i+1,j,sizeX)] - p [indexTo1D(i
                    -1, j, sizeX)])
            else:
                gradP_u.append(0)
            if (j>1) and (j < sizeY-1):
                gradP_v.append(p[indexTo1D(i,j+1,sizeX)] - p[indexTo1D(i,j
                    -1,sizeX)])
            else:
                gradP_v.append(0)

    gradP_u = (0.5/h)*np.array(gradP_u)
    gradP_v = (0.5/h)*np.array(gradP_v)
    new_u = _u - gradP_u
    new_v = _v - gradP_v
    new_u, new_v = set_boundary(new_u, new_v, sizeX, sizeY, boundary_func)
    return new_u, new_v
```

Stable fluid solver: Moving a substance through the fluid

A substance $s : \mathbb{R}^N \longrightarrow \mathbb{R}$ moving through the field

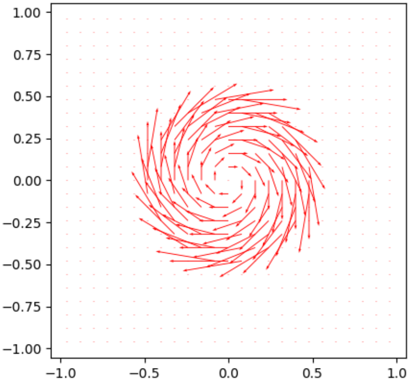
Navier-Stokes for a scalar field

$$\frac{\partial s}{\partial t} = -u \cdot \nabla s + \kappa \nabla^2 s - \alpha s + f_s$$

Similar to what we explained before ! $\Longrightarrow$ same resolution method

Results

Vortex

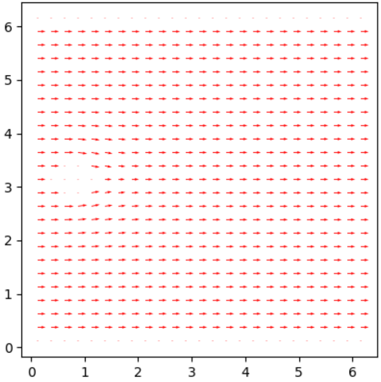


Initial velocity field (grid $[-1, 1]$ of size 25×25)

Initial density field (grid $[-1, 1]$ of size 99×99)

Vortex

Karman vortex



Initial velocity field (grid $[-1, 1]$ of size 25×25)

Initial vorticity field (grid $[-1, 1]$ of size 90×90)

About the two data structures

Using matrix data structure

Using tensor data structure

Roadmap

Goals of the project

- ✓ Implement a 2D stable fluid solver in Python
- Making it differentiable using TensorFlow library

Differentiable physics in TensorFlow

TensorFlow

TensorFlow

- open-source software library for ML and AI applications
- developed by Google Brain Team
- could be used to train a neural network to predict the behavior of a physical system based on a set of inputs
- uses backpropagation

$$\begin{tikzcd}
u \arrow[r, "\texttt{addForce()}"] & u \arrow[r, "\texttt{advect()}"] & u \arrow[r, "\texttt{diffuse()}"] & u \arrow[r, "\texttt{project()}"] & u \arrow[d, "\texttt{advect()}"] & & \\
 & & & s \arrow[r, "\texttt{addForce()}"'] & s \arrow[r, "\texttt{diffuse()}"'] & s \arrow[r, "\texttt{dissipate()}"'] & s
\end{tikzcd}$$

```python
import tensorflow as tf
u_init = ...#Initialize u_init
density_init = ...#Initialize density_init
u_init = tf.convert_to_tensor(u_init)
density_init = tf.convert_to_tensor(density_init)
with tf.GradientTape() as tape:
    velocity_field = tf.Variable(u_init)
    density_field = solveDens(density_init, solveVel(velocity_field))
grad = tape.gradient([density_field], [velocity_field])
```

Every operation must be differentiable $\implies$ is currently slow

An application to a matching shape problem

Starting density s_0

$\Longrightarrow$

Target density field t

An application to a matching shape problem

Goal: Minimize $\mathcal{L}(s_0, u) = ||\mathbb{F}_2(s_0, \mathbb{F}_1(u)) - t||^2$

Use the gradient to perform a gradient descent

$$u_{n+1} = u_n - \alpha_n \left. \frac{\partial \mathcal{L}(s_0, u_n)}{\partial u} \right|_{u_n}$$

Every operation must be differentiable $\Longrightarrow$ is currently slow

Different learning rate α_n possible

An application to a matching shape problem

Density obtained with the trained velocity

Target density field at frame 20

An application to a matching shape problem

Density obtained at frame 20 with the trained velocity

Target density field t at frame 20

Discussions and Future work

Discussions and Future work

- different grid model (velocity stored at the edges, ...)
- explore parallelization to reduce execution time
- extension in 3D

Conclusion

Goals of the project

- ✔ Implement a 2D stable fluid solver in Python
- ✔ Making it differentiable using TensorFlow library
- Testing on training problems
- Applying it to constraint-based physical problems

Thank You

Questions?