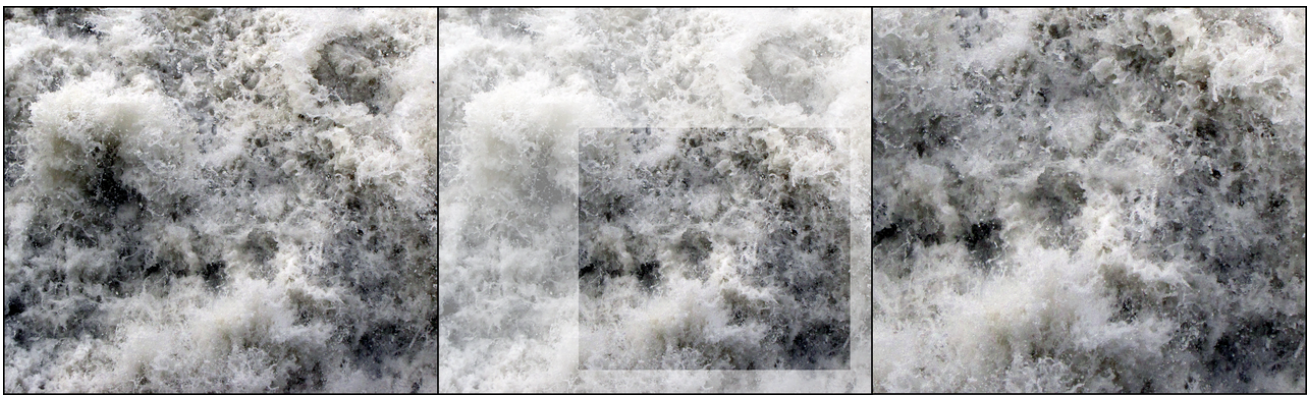
Coding Example: Fluid Dynamics

Now we will discuss a case study related to computation fluids and see which approach out of Eulerian and Lagrangian is easier to implement in NumPy.

WE'LL COVER THE FOLLOWING

Problem Description

Now let's look at another interesting example!



Hydrodynamic flow at two different zoom levels, Neckar river, Heidelberg, Germany. Image by Steven Mathey, 2012.

Lagrangian vs Eulerian method

In classical field theory, the Lagrangian specification of the field is a way of looking at fluid motion where the observer follows an individual fluid parcel

as it moves through space and time. Plotting the position of an individual

parcel through time gives the pathline of the parcel. This can be visualized as sitting in a boat and drifting down a river.

The Eulerian specification of the flow field is a way of looking at fluid motion that focuses on specific locations in the space through which the fluid flows as time passes. This can be visualized by sitting on the bank of a river and watching the water pass the fixed location.

In other words, in the Eulerian case, you divide a portion of space into cells and each cell contains a velocity vector and other information, such as density and temperature. In the Lagrangian case, we need particle-based physics with dynamic interactions and generally, we need a high number of particles. Both methods have advantages and disadvantages and the choice between the two methods depends on the nature of your problem. Of course, you can also mix the two methods into a hybrid method.

However, the biggest problem for particle-based simulation is that particle interaction requires finding neighboring particles and this has a cost as we've seen in the boids case. If we target Python and NumPy only, it is probably better to choose the Eulerian method since vectorization will be almost trivial compared to the Lagrangian method.

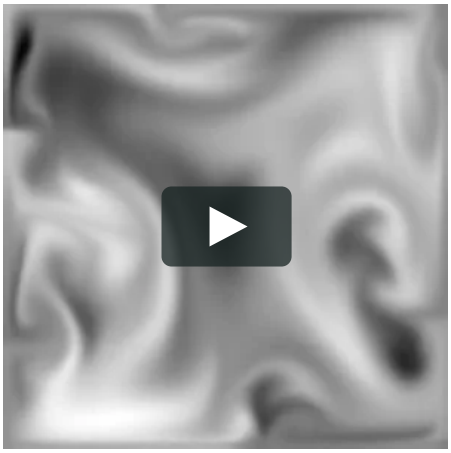
NumPy Implementation

I won't explain all the theory behind computational fluid dynamics because first, I cannot (I'm not an expert at all in this domain) and there are many resources online that explain this nicely (have a look at references below, especially tutorial by L. Barba). Why choose a computational fluid as an example then? Because results are (almost) always beautiful and fascinating. I couldn't resist (look at the video below).

We'll further simplify the problem by implementing a method from computer graphics where the goal is not correctness but convincing behavior. Jos Stam wrote a very nice article for SIGGRAPH 1999 describing a technique to have stable fluids over time (i.e. whose solution in the long term does not diverge). Alberto Santini wrote a Python replication a long time ago (using numarray!) such that I only had to adapt it to modern NumPy and accelerate it a bit using modern NumPy tricks.

modern NumPy tricks.

I won't comment the code since it would be too long, but you can read the original paper as well as the explanation by Philip Rideout on his blog. Below are some movies I've made using this technique.



Smoke Simulation Implementation (Image 1)

To generate the first image in the video, use the following NumPy implementation which is based on the Eulerian approach.

main.py

smoke_solver.py

```
# ----------------------------------------------------------------------
# From Numpy to Python
# Copyright (2017) Nicolas P. Rougier - BSD license
# More information at https://github.com/rougier/numpy-book
# ----------------------------------------------------------------------
import numpy as np
from smoke_solver import vel_step, dens_step
```

```
N = 128
size = N + 2
dt = 0.1

diff = 0.0
visc = 0.0
force = 5.0
source = 100.0

u = np.zeros((size, size), np.float32)  # velocity
u_prev = np.zeros((size, size), np.float32)

v = np.zeros((size, size), np.float32)  # velocity
v_prev = np.zeros((size, size), np.float32)

dens = np.zeros((size, size), np.float32)  # density
dens_prev = np.zeros((size, size), np.float32)


def initialization():
    global u, v, u_prev, v_prev, dens, dens_prev, size

    u[:, :] = 0.0
    v[:, :] = 0.0
    u_prev[:, :] = 0.0
    v_prev[:, :] = 0.0
    dens[:, :] = 0.0
    dens_prev[:, :] = 0.0

    def disc(shape=(size, size), center=(size/2, size/2), radius=10):
        def distance(x, y):
            return np.sqrt((x-center[0])**2+(y-center[1])**2)
        D = np.fromfunction(distance, shape)
        return np.where(D <= radius, True, False)

    D = disc(radius=32) ^ disc(radius=16)
    dens[...] = D*source/10
    u[:, :] = force * 0.1 * np.random.uniform(-1, 1, u.shape)
    v[:, :] = force * 0.1 * np.random.uniform(-1, 1, u.shape)


def update(*args):
    global im, dens, dens_prev, u, u_prev, v, v_prev, N, visc, dt, diff

    vel_step(N, u, v, u_prev, v_prev, visc, dt)
    dens_step(N, dens, dens_prev, u, v, diff, dt)
    im.set_data(dens)
    im.set_clim(vmin=dens.min(), vmax=dens.max())


if __name__ == '__main__':
    import matplotlib.pyplot as plt
    import matplotlib.animation as animation


    fig = plt.figure(figsize=(5, 5))
    ax = fig.add_axes([0, 0, 1, 1], frameon=False)
    ax.set_xlim(0, 1)
    ax.set_xticks([])
    ax.set_ylim(0, 1)
    ax.set_yticks([])

    initialization()
```

```
    im = ax.imshow(dens[1:-1, 1:-1],
                   interpolation='bicubic', extent=[0, 1, 0, 1],
                   cmap=plt.cm.magma, origin="lower", vmin=0, vmax=1)
    Writer = animation.writers['ffmpeg']
    writer = Writer(fps=15, metadata=dict(artist='Me'), bitrate=1800)

    anim = animation.FuncAnimation(fig, update, interval=10, frames=100)
    anim.save('output/output.mp4', writer=writer)
    plt.show()
```

Smoke Simulation Implementation (Image 2)

To generate the second image in the video, use the following NumPy implementation which is based on the Eulerian approach.

main.py

smoke_solver.py

```
# ------------------------------------------------------------------------
# From Numpy to Python
# Copyright (2017) Nicolas P. Rougier - BSD license
# More information at https://github.com/rougier/numpy-book
# ------------------------------------------------------------------------
import numpy as np
from smoke_solver import vel_step, dens_step

N = 128
size = N + 2
dt = 0.1
diff = 0.0
visc = 0.0
force = 5.0
source = 100.0
dvel = False

u = np.zeros((size, size), np.float32)  # velocity
u_prev = np.zeros((size, size), np.float32)

v = np.zeros((size, size), np.float32)  # velocity
v_prev = np.zeros((size, size), np.float32)

dens = np.zeros((size, size), np.float32)  # density
dens_prev = np.zeros((size, size), np.float32)


def initialization():
    global u, v, u_prev, v_prev, dens, dens_prev, size

    u[:, :] = 0.0
    v[:, :] = 0.0
    u_prev[:, :] = 0.0
```

```
    v_prev[:, :] = 0.0
    dens[:, :] = 0.0
    dens_prev[:, :] = 0.0

    def disc(shape=(size, size), center=(size/2, size/2), radius=10):
        def distance(x, y):
            return np.sqrt((x-center[0])**2+(y-center[1])**2)
        D = np.fromfunction(distance, shape)
        return np.where(D <= radius, True, False)

    D = disc(radius=10) ^ disc(radius=5)
    dens[...] += D*source/50

    D = disc(radius=20) ^ disc(radius=15)
    dens[...] += D*source/50

    ox, oy = size/2, size/2
    for j in range(1, N+1):
        for i in range(1, N+1):
            d = np.sqrt((i-ox)**2+(j-oy)**2)
            u[i, j] = (i-ox)/max(d, 1) * force * 0.25
            v[i, j] = (j-oy)/max(d, 1) * force * 0.25
    u[:, :] += force * 0.1 * np.random.uniform(-1, 1, u.shape)
    v[:, :] += force * 0.1 * np.random.uniform(-1, 1, u.shape)


def update(*args):
    global im, dens, dens_prev, u, u_prev, v, v_prev, N, visc, dt, diff

    vel_step(N, u, v, u_prev, v_prev, visc, dt)
    dens_step(N, dens, dens_prev, u, v, diff, dt)
    im.set_data(dens)
    im.set_clim(vmin=dens.min(), vmax=dens.max())

if __name__ == '__main__':
    import matplotlib.pyplot as plt
    import matplotlib.animation as animation

    fig = plt.figure(figsize=(5, 5))
    ax = fig.add_axes([0, 0, 1, 1], frameon=False)
    ax.set_xlim(0, 1)
    ax.set_xticks([])
    ax.set_ylim(0, 1)
    ax.set_yticks([])
    initialization()
    im = ax.imshow(dens[1:-1, 1:-1],
                   interpolation='bicubic', extent=[0, 1, 0, 1],
                   cmap=plt.cm.gray, origin="lower", vmin=0, vmax=1)

    Writer = animation.writers['ffmpeg']
    writer = Writer(fps=15, metadata=dict(artist='Me'), bitrate=1800)

    anim = animation.FuncAnimation(fig, update, interval=10, frames=100)
    anim.save('output/output.mp4', writer=writer)
    plt.show()
```

Further Readings

- 12 Steps to Navier-Stokes, Lorena Barba, 2013.
- Stable Fluids , Jos Stam, 1999.
- Simple Fluid Simulation, Philip Rideout, 2010
- Animating Sand as a Fluid, Yongning Zhu & Robert Bridson, 2005.