13_Step_10

November 27, 2025

1 12 steps to Navier–Stokes

For a moment, recall the Navier–Stokes equations for an incompressible fluid, where $\vec{v}$ represents the velocity field:

$$\nabla \cdot \vec{v} = 0$$
$$\frac{\partial \vec{v}}{\partial t} + (\vec{v} \cdot \nabla)\vec{v} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \vec{v}$$

The first equation represents mass conservation at constant density. The second equation is the conservation of momentum. But a problem appears: the continuity equation for incompressble flow does not have a dominant variable and there is no obvious way to couple the velocity and the pressure. In the case of compressible flow, in contrast, mass continuity would provide an evolution equation for the density ρ, which is coupled with an equation of state relating ρ and p.

In incompressible flow, the continuity equation $\nabla \cdot \vec{v} = 0$ provides a *kinematic constraint* that requires the pressure field to evolve so that the rate of expansion $\nabla \cdot \vec{v}$ should vanish everywhere. A way out of this difficulty is to *construct* a pressure field that guarantees continuity is satisfied; such a relation can be obtained by taking the divergence of the momentum equation. In that process, a Poisson equation for the pressure shows up!

1.1 Step 10: 2D Poisson Equation

Poisson's equation is obtained from adding a source term to the right-hand-side of Laplace's equation:

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = b$$

So, unlinke the Laplace equation, there is some finite value inside the field that affects the solution. Poisson's equation acts to "relax" the initial sources in the field.

In discretized form, this looks almost the same as Step 9, except for the source term:

$$\frac{p^n_{i+1,j} - 2p^n_{i,j} + p^n_{i-1,j}}{\Delta x^2} + \frac{p^n_{i,j+1} - 2p^n_{i,j} + p^n_{i,j-1}}{\Delta y^2} = b^n_{i,j}$$

As before, we rearrange this so that we obtain an equation for p at point i, j. Thus, we obtain:

$$p^n_{i,j} = \frac{(p^n_{i+1,j} + p^n_{i-1,j})\Delta y^2 + (p^n_{i,j+1} + p^n_{i,j-1})\Delta x^2 - b^n_{i,j}\Delta x^2 \Delta y^2}{2(\Delta x^2 + \Delta y^2)}$$

We will solve this equation by assuming an initial state of $p = 0$ everywhere, and applying boundary conditions as follows:

$p = 0$ at $x = 0,\ 2$ and $y = 0,\ 1$

and the source term consists of two initial spikes inside the domain, as follows:

$b_{i,j} = 100$ at $i = \frac{1}{4}nx, j = \frac{1}{4}ny$

$b_{i,j} = -100$ at $i = \frac{3}{4}nx, j = \frac{3}{4}ny$

$b_{i,j} = 0$ everywhere else.

The iterations will advance in pseudo-time to relax the initial spikes. The relaxation under Poisson's equation gets slower and slower as they progress. *Why?*

Let's look at one possible way to write the code for Poisson's equation. As always, we load our favorite Python libraries. We also want to make some lovely plots in 3D. Let's get our parameters defined and the initialization out of the way. What do you notice of the approach below?

[51]:
```
import numpy
from matplotlib import pyplot, cm
from mpl_toolkits.mplot3d import Axes3D
%matplotlib inline
```

[52]:
```
# Parameters
nx = 50
ny = 50
nt  = 100
xmin = 0
xmax = 2
ymin = 0
ymax = 1

dx = (xmax - xmin) / (nx - 1)
dy = (ymax - ymin) / (ny - 1)

# Initialization
p  = numpy.zeros((ny, nx))
pd = numpy.zeros((ny, nx))
b  = numpy.zeros((ny, nx))
```

```
x  = numpy.linspace(xmin, xmax, nx)
y  = numpy.linspace(xmin, xmax, ny)

# Source
b[int(ny / 4), int(nx / 4)]  = 100
b[int(3 * ny / 4), int(3 * nx / 4)] = -100
```

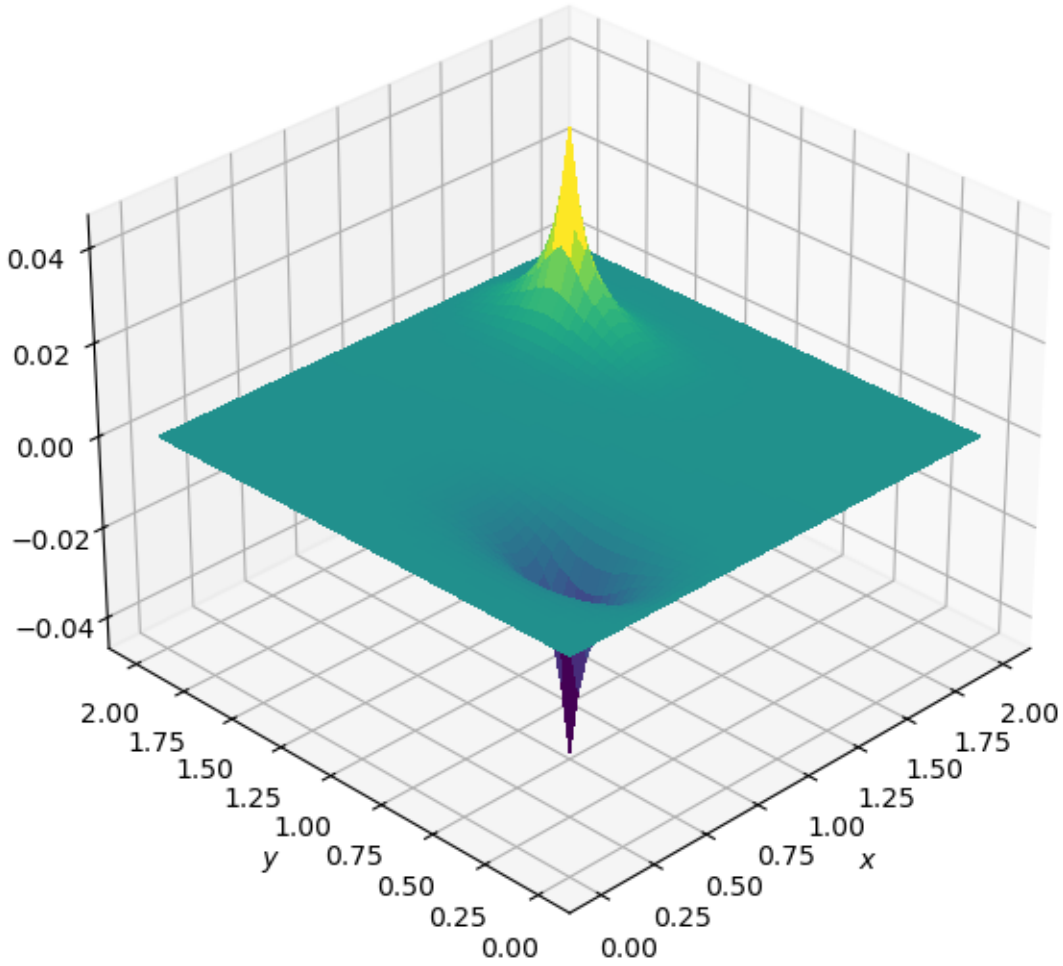
With that, we are ready to advance the initial guess in pseudo-time. How is the code below different from the function used in Step 9 to solve Laplace's equation?

[53]:
```
for it in range(nt):

    pd = p.copy()

    p[1:-1,1:-1] = (((pd[1:-1, 2:] + pd[1:-1, :-2]) * dy**2 +
                     (pd[2:, 1:-1] + pd[:-2, 1:-1]) * dx**2 -
                     b[1:-1, 1:-1] * dx**2 * dy**2) /
                     (2 * (dx**2 + dy**2)))

    p[0, :] = 0
    p[ny-1, :] = 0
    p[:, 0] = 0
    p[:, nx-1] = 0
```

Maybe we could reuse our plotting function from Step 9, don't you think?

[54]:
```
def plot2D(x, y, p):
    fig = pyplot.figure(figsize=(11, 7), dpi=100)
    ax = fig.add_subplot(projection='3d')
    X, Y = numpy.meshgrid(x, y)
    surf = ax.plot_surface(X, Y, p[:], rstride=1, cstride=1, cmap=cm.viridis,
            linewidth=0, antialiased=False)
    ax.view_init(30, 225)
    ax.set_xlabel('$x$')
    ax.set_ylabel('$y$')
```

[55]:
```
plot2D(x, y, p)
pyplot.show()
```



Ah! The wonders of code reuse! Now, you probably think: "Well, if I've written this neat little function that does something so useful, I want to use it over and over again. How can I do this without copying and pasting it each time? —If you are very curious about this, you'll have to learn about *packaging*. But this goes beyond the scope of our CFD lessons. You'll just have to Google it if you really want to know.

1.2 Learn More

To learn more about the role of the Poisson equation in CFD, watch **Video Lesson 11** on You Tube:

[56]:
```
# from IPython.display import YouTubeVideo
# YouTubeVideo('ZjfxA3qq2Lg')
```

[57]:
```
# from IPython.core.display import HTML
# def css_styling():
#     styles = open("./custom.css", "r").read()
#     return HTML(styles)
# css_styling()
```

(The cell above executes the style for this notebook.)