Navier-stokes forward simulation using phiflow

- incompressible form, Boussinesq approximation: approximates changes in density for incompressible solvers without explicitly calculating rho
- additional pressure field p + constraint for conservation of mass
- marker field d - regions of higher temperature, exerts a force via a buoyancy factor zhi
- We assume a gravity force that acts along the y direction

We'll solve this PDE on a closed domain with Dirichlet boundary conditions for the velocity u=0, and Neumann boundaries del rho/del x = 0 for pressure, on a domain omega with a physical size of 100*80 units

```
!pip install --upgrade --quiet phiflow==2.2

from phi.flow import *  # The Dash GUI is not supported on Google colab, ignore the warning
import pylab
```

40*32 cells to discretize our domain, introduce a slight viscosity nu

Creating a first CenteredGrid here, which is initialized by a Sphere geometry object. This will represent the inflow region INFLOW where hot smoke is generated.

```
#caps for constants
DT = 1.5 #times step
NU = 0.01 #kinematic viscosity

INFLOW = CenteredGrid(Sphere(center=tensor([30,15], channel(vector='x,y')), radius=10), extrapolation.BOUNDARY, x=32, y=40, bounds=Box(x=(0,80),y=(0,100))) * 0.2
```

The **inflow** will be used to inject smoke into a second centered grid smoke that represents the marker field d from above.

Box size: *100x80*. This is the physical scale in terms of spatial units in our simulation, i.e., a velocity of magnitude 1 will move the smoke density by 1 unit per 1 time unit, which may be larger or smaller than a cell in the discretized grid depending on the settings for x,y.

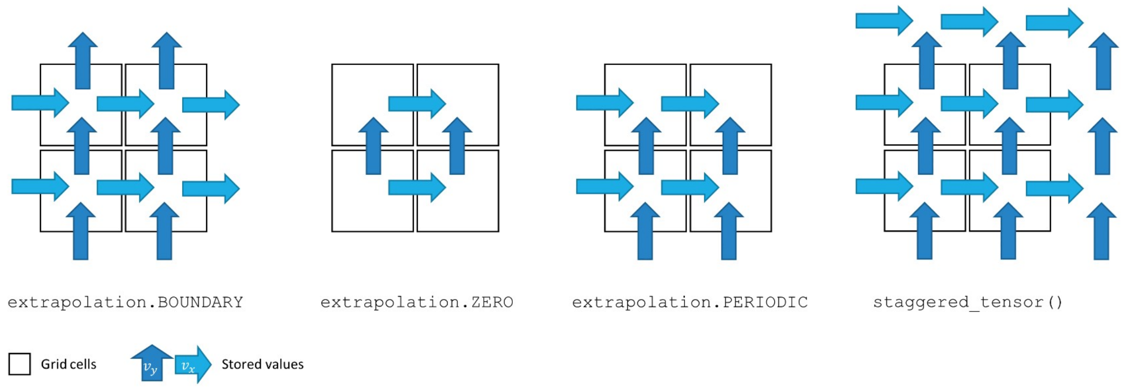
The inflow sphere above is already using the "world" coordinates: it is located at x=30 along the first axis, and y=15 (within the 100x80 domain box).

Next, we create grids for the quantities we want to simulate. For this example, we require a velocity field and a smoke density field.

```
smoke = CenteredGrid(0, extrapolation.BOUNDARY, x=32, y=40,
bounds=Box(x=(0,80),y=(0,100)))  # sampled at cell centers
velocity = StaggeredGrid(0, extrapolation.ZERO, x=32, y=40,
bounds=Box(x=(0,80),y=(0,100)))  # sampled in staggered form at face
centers
```

We sample the smoke field at the cell centers and the velocity in staggered form. The staggered grid internally contains 2 centered grids with different dimensions, and can be converted into centered grids (or simply numpy arrays) via the unstack function.

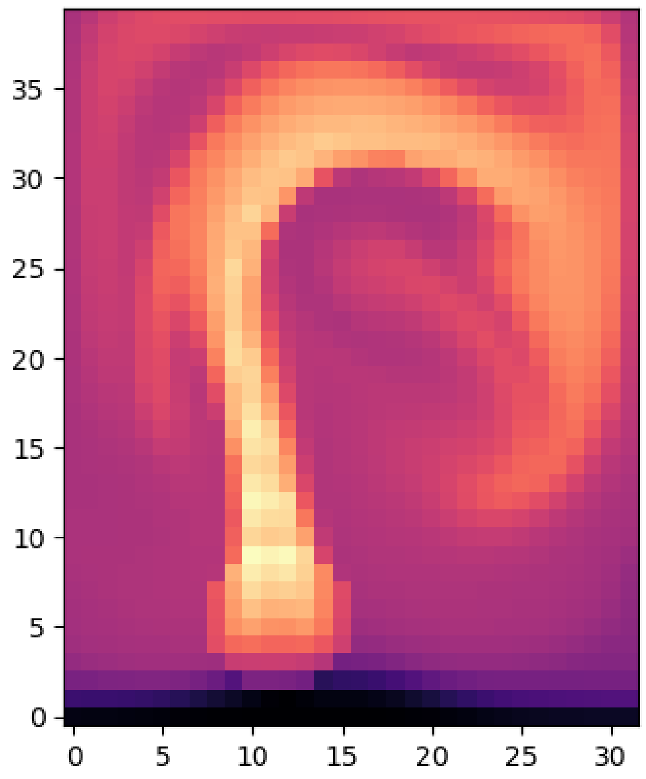
extrapolation.ZERO gives: each component of the values tensor has one less sample point in the direction it is facing. (here centered grid - face center) **extrapolation.boundary** gives: each component of the values tensor has one more sample point in the direction it is facing. (here centered grid - face center)



Next we define the update step of the simulation, which calls the necessary functions to advance the state of our fluid system by **dt**. The next cell computes one such step, and plots the marker density after one simulation frame.

```
def step(velocity, smoke, pressure, dt=1.0, zhi =1.0): #zhi=buoyancy
factor
    smoke = advect.semi_lagrangian(smoke, velocity, dt) + INFLOW
    buoyant_force = (smoke * (0, zhi)).at(velocity)  # resamples smoke
to velocity sample points
    velocity = advect.semi_lagrangian(velocity, velocity, dt) + dt *
buoyant_force
    velocity = diffuse.explicit(velocity, NU, dt)
    velocity, pressure = fluid.make_incompressible(velocity)
    return velocity, smoke, pressure

velocity, smoke, pressure = step(velocity, smoke, None, dt=DT)

print("Max. velocity and mean marker density: " +
format([math.max(velocity.values),math.mean(smoke.values)]))

pylab.imshow(np.asarray(smoke.values.numpy('y,x')), origin='lower',
cmap='magma')
```

```
Max. velocity and mean marker density: [7.5043564, 0.939316]

<matplotlib.image.AxesImage at 0x20285a25700>
```



A lot has happened in this step() call: We've advected the ***smoke field***, added an upwards force via a Boussinesq model, advected the ***velocity field***, and finally made it ***divergence free via a pressure solve***.

The Boussinesq model uses a multiplication by a tuple (0, zhi) to turn the smoke field into a staggered, 2 component force field, sampled at the locations of the velocity components via the at() function. This function makes sure the individual force components are correctly interpolated for the velocity components of the staggered velocity. This also directly ensure the boundary conditions of the original grid are kept. It internally also does StaggeredGrid(..., extrapolation.ZERO,...) for the resulting force grid.

The pressure projection step in make_incompressible is typically the computationally most expensive step in the sequence above. It solves a Poisson equation for the boundary conditions of the domain, and updates the velocity field with the gradient of the computed pressure.

Just for testing, we've also printed the mean value of the velocities, and the max density after the update. In the resulting image, we have a first round region of smoke, with a slight upwards motion (which does not show here yet).

The variables we created for the fields of the simulation here are instances of the class Grid. Like tensors, grids also have the shape attribute which lists all batch, spatial and channel dimensions. Shapes in phiflow store not only the sizes of the dimensions but also their names and types.

```
print(f"Smoke: {smoke.shape}")
print(f"Velocity: {velocity.shape}")
print(f"Inflow: {INFLOW.shape}, spatial only: {INFLOW.shape.spatial}")
```

```
Smoke: (xˢ=32, yˢ=40)
Velocity: (xˢ=32, yˢ=40, vectorᶜ=x,y)
Inflow: (xˢ=32, yˢ=40), spatial only: (xˢ=32, yˢ=40)
```

The phiflow output here indicates the type of a dimension, e.g., ***S*** for a spatial, ***V*** and for a vector dimension.

The actual content of a shape object can be obtained via **.sizes**, or alternatively we can query the size of a specific dimension dim via **.get_size('dim')**.

Example:

```
print(f"Shape content: {velocity.shape.sizes}")
print(f"Vector dimension: {velocity.shape.get_size('vector')}")
```

```
Shape content: (32, 40, 2)
Vector dimension: 2
```

The grid values can be accessed using the values property. This is an important difference to a phiflow tensor object, which does not have values:

```
print("Statistics of the different simulation grids:")
print(smoke.values)
print(velocity.values)

# in contrast to a simple tensor:
test_tensor = math.tensor(numpy.zeros([3, 5, 2]), spatial('x,y'),
channel(vector="x,y"))
print("Reordered test tensor shape: " +
format(test_tensor.numpy('vector,y,x').shape) ) # reorder to
vector,y,x
#print(test_tensor.values.numpy('y,x')) gives error as tensors don't
return their content via ".values"
```

```
Statistics of the different simulation grids:
(xˢ=32, yˢ=40) 0.008 ± 0.039 (0e+00...2e-01)
(xˢ=(x=31, y=32), yˢ=(x=40, y=39), vectorᶜ=x,y) -7.23e-09 ± 5.3e-03 (-
1e-01...2e-01)
Reordered test tensor shape: (2, 5, 3)
```

The staggered grid has a non-uniform shape because the number of faces is not equal to the number of cells. x and y components have different no. of cells. The INFLOW grid naturally has the same dimensions as the smoke grid.

Time evolution

With this setup, we can advance the simulation forward in time a bit more by repeatedly calling the step function.

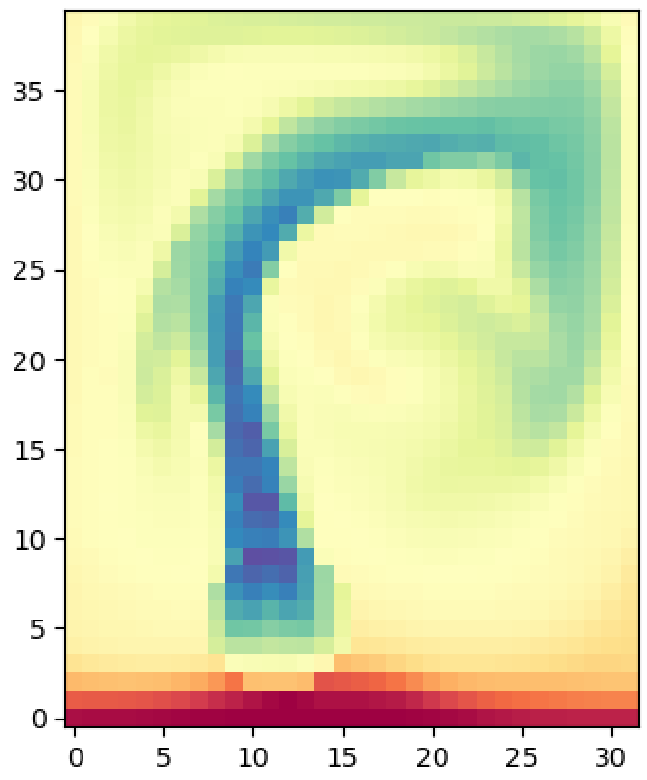
```
for time_step in range(10):
    velocity, smoke, pressure = step(velocity, smoke, pressure, dt=DT)
    print('Computed frame {}, max velocity {}'.format(time_step ,
np.asarray(math.max(velocity.values))))
```

```
Computed frame 0, max velocity 0.463001
Computed frame 1, max velocity 0.896646
Computed frame 2, max velocity 1.4098884
Computed frame 3, max velocity 2.0411277
Computed frame 4, max velocity 2.927958
Computed frame 5, max velocity 3.8394766
Computed frame 6, max velocity 4.526946
Computed frame 7, max velocity 4.867982
Computed frame 8, max velocity 5.13108
Computed frame 9, max velocity 5.4838786
```

The hot plume is starting to rise:

```
pylab.imshow(smoke.values.numpy('y,x'), origin='lower',
cmap='Spectral')
```

```
<matplotlib.image.AxesImage at 0x20283b5ff10>
```

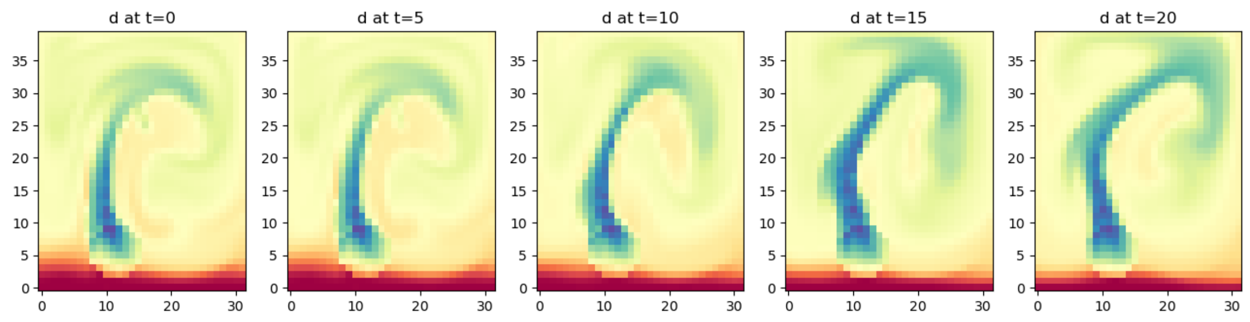


A few more steps of the simulation:

Because of the inflow being located off-center to the left (with x position 30), the plume will curve towards the right when it hits the top wall of the domain.

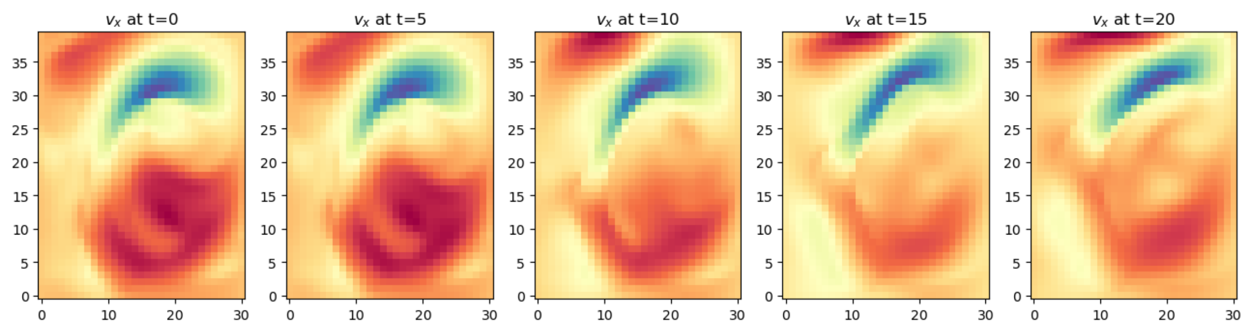
```
steps = [[smoke.values, velocity.values.vector[0],
velocity.values.vector[1]]]
for time_step in range(20):
  if time_step<3 or time_step%10==0:
    print('Computing time step %d' % time_step)
  velocity, smoke, pressure = step(velocity, smoke, pressure, dt=DT)
  if time_step%5==0:
    steps.append([smoke.values, velocity.values.vector[0],
velocity.values.vector[1]])

fig, axes = pylab.subplots(1, len(steps), figsize=(16, 5))
for i in range(len(steps)):
    axes[i].imshow(steps[i][0].numpy('y,x'), origin='lower',
cmap='Spectral')
    axes[i].set_title(f"d at t={i*5}")
```

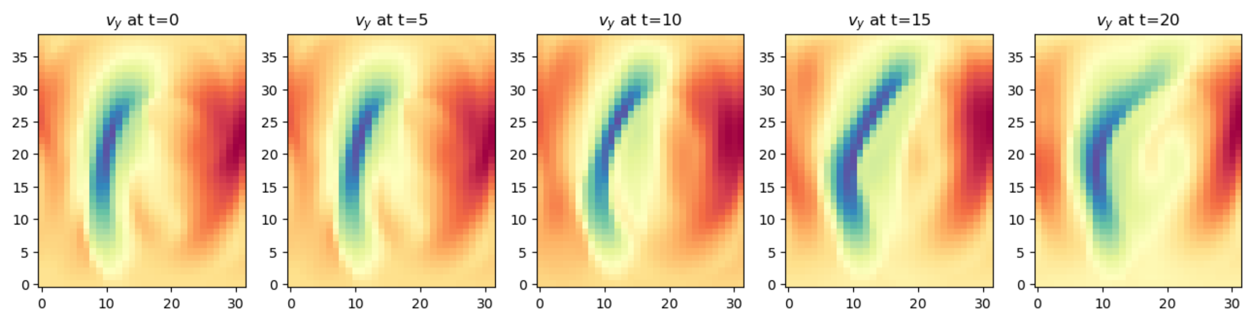
```
Computing time step 0
Computing time step 1
Computing time step 2
Computing time step 10
```



The steps list above already stores vector[0] and vector[1] components of the velocities as numpy arrays:

```
fig, axes = pylab.subplots(1, len(steps), figsize=(16, 5))
for i in range(len(steps)):
    axes[i].imshow(steps[i][1].numpy('y,x'), origin='lower',
cmap='Spectral')
    axes[i].set_title(f"$v_x$ at t={i*5}") #x component of velocity

fig, axes = pylab.subplots(1, len(steps), figsize=(16, 5))
for i in range(len(steps)):
    axes[i].imshow(steps[i][2].numpy('y,x'), origin='lower',
cmap='Spectral')
    axes[i].set_title(f"$v_y$ at t={i*5}") #y component of velocity
```


v_x at t=0
v_x at t=5
v_x at t=10
v_x at t=15
v_x at t=20


v_y at t=0
v_y at t=5
v_y at t=10
v_y at t=15
v_y at t=20