Exercise 1

May 1, 2023

1 Advanced Deep Learning for Physics Exercise 1: Introduction to Φ_{Flow}

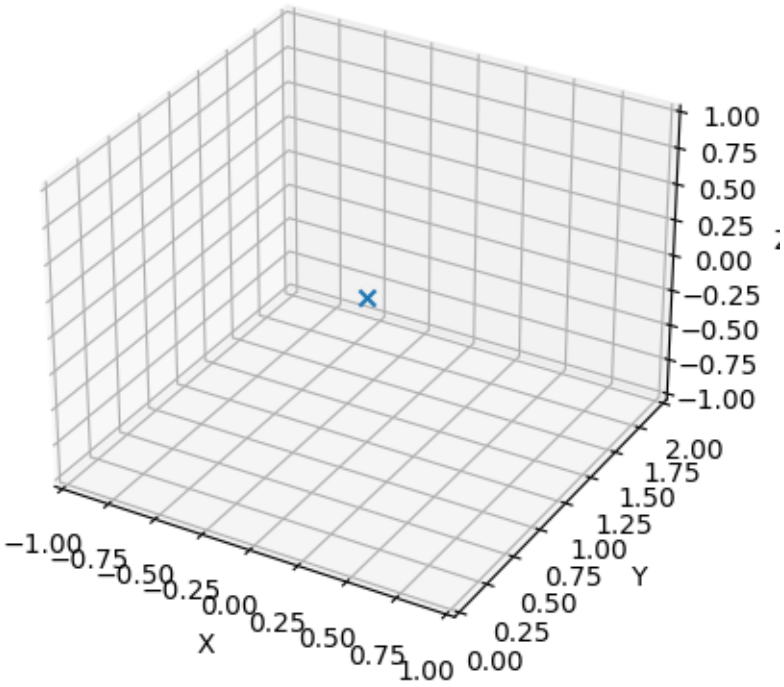
[ ]:
```
from phi.torch.flow import*
import matplotlib.pyplot as plt
```

1.1 Tensors Representing Physical Data

- Representing a Single Point

[ ]:
```
v1 = vec(x=0, y=1, z=0)
plot(v1, size=(4,4))
```

[ ]:
```
<Figure size 400x400 with 1 Axes>
```



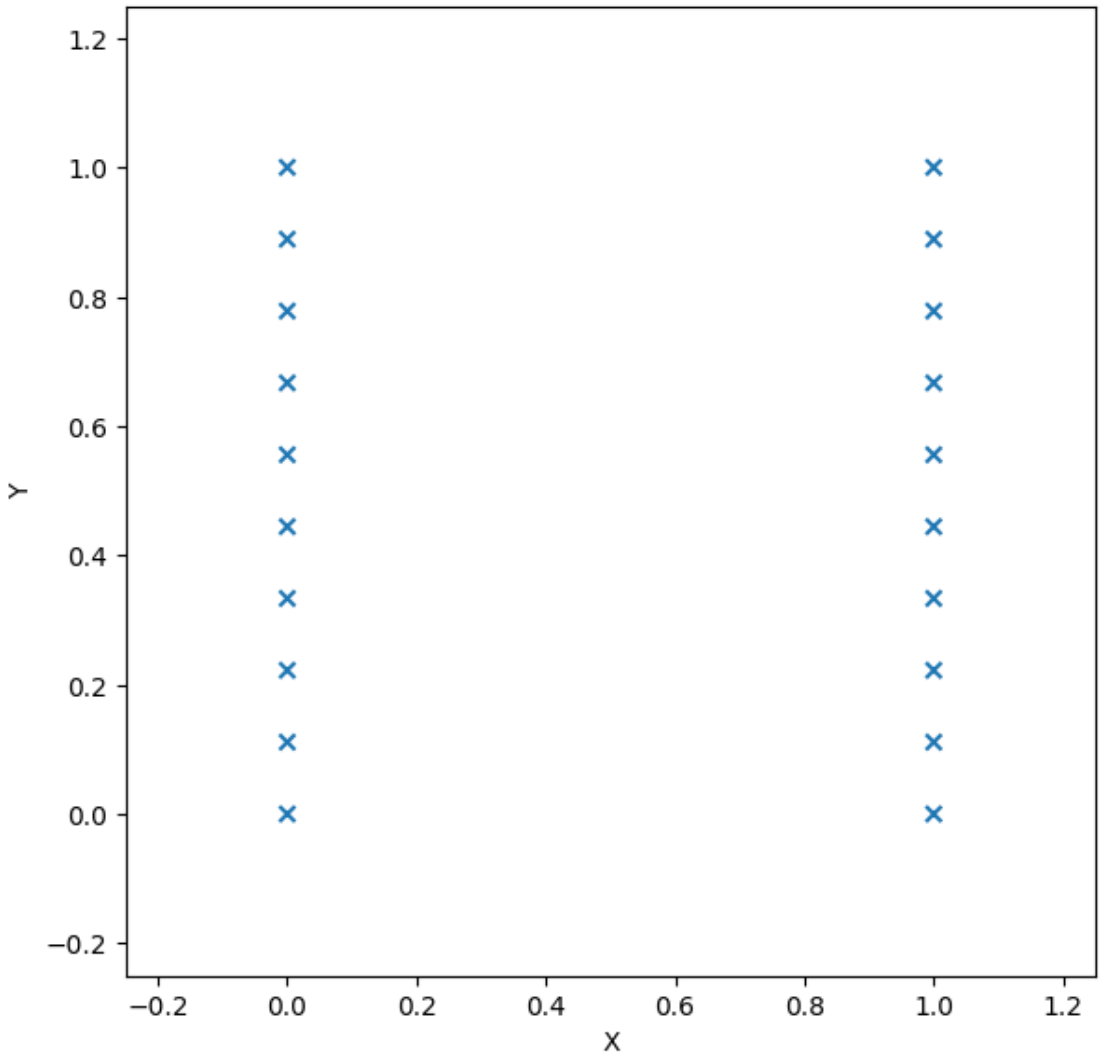
- Representing Linearly Spaced Points

[ ]:
```
points = concat([vec(x = 0, y = math.linspace(0, 1, instance(points = 10))),
            vec(x = 1, y = math.linspace(0, 1, instance(points = 10)))],␣
 ↪'points')
plot(points, size = (6,6))

plt.xlim(left = -.25, right = 1.25)

plt.ylim(bottom = -.25, top = 1.25)
```

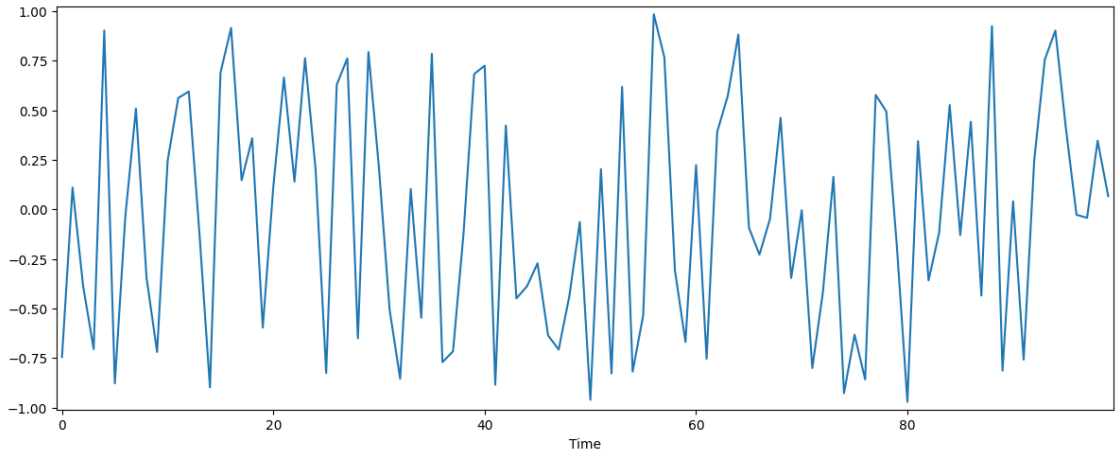
[ ]:
```
(-0.25, 1.25)
```



- Time Dependent Signal Consisting of 100 Samples

[ ]:
```
t1 = math.random_uniform(spatial(time=100), low = -1, high = 1)
plot(t1)
```

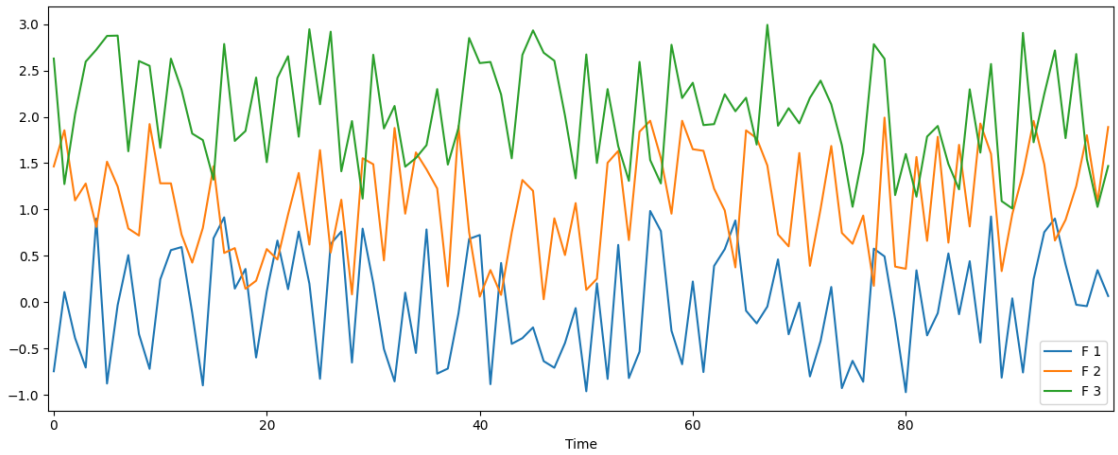
[ ]:
```
<Figure size 1200x500 with 1 Axes>
```



- Three Time Dependent Signals Consisting of 100 Samples

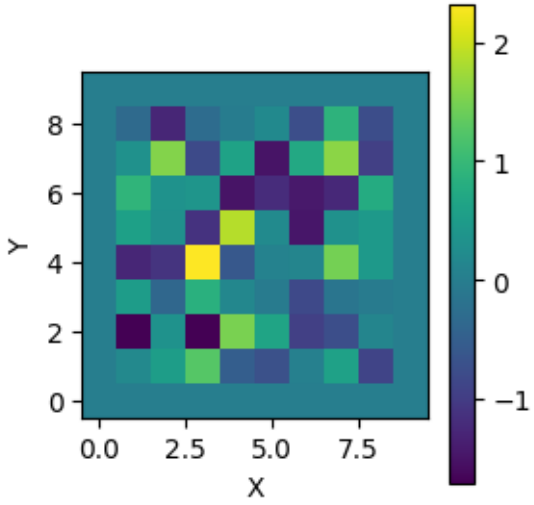
[ ]:
```
x = stack({
    'f_1': t1,
    'f_2': math.random_uniform(spatial(time=100), low = 0, high = 2),
    'f_3':  math.random_uniform(spatial(time=100), low = 1, high = 3)
}, channel('spatial'))
plot(x)
```

[ ]:
```
<Figure size 1200x500 with 1 Axes>
```



- A Two-Dimensional Scalar 10x10 Grid

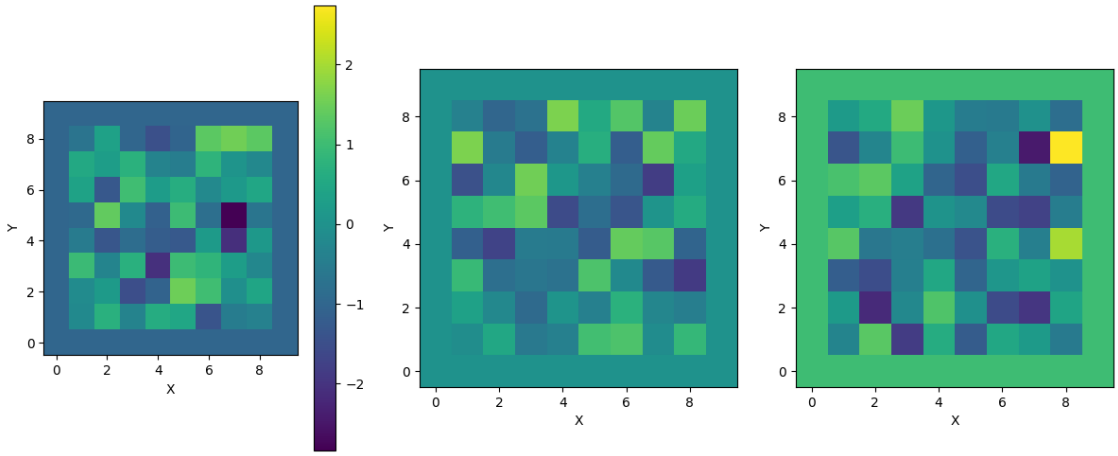
```
[ ]: p1 = plot(math.pad(math.random_normal(spatial(x=8, y=8)), {'x': (1, 1), 'y':␣
     ↪(1, 1)}, 0), size=(3,3))
```



- Three Two-Dimensional Scalar 10x10 Grids

```
[ ]: plot(math.pad(math.random_normal(spatial(x=8, y=8)), {'x': (1, 1), 'y': (1,␣
     ↪1)}, -1),
         math.pad(math.random_normal(spatial(x=8, y=8)), {'x': (1, 1), 'y': (1,␣
     ↪1)}, 0),
         math.pad(math.random_normal(spatial(x=8, y=8)), {'x': (1, 1), 'y': (1,␣
     ↪1)}, 1))
```

```
[ ]: <Figure size 1200x500 with 4 Axes>
```

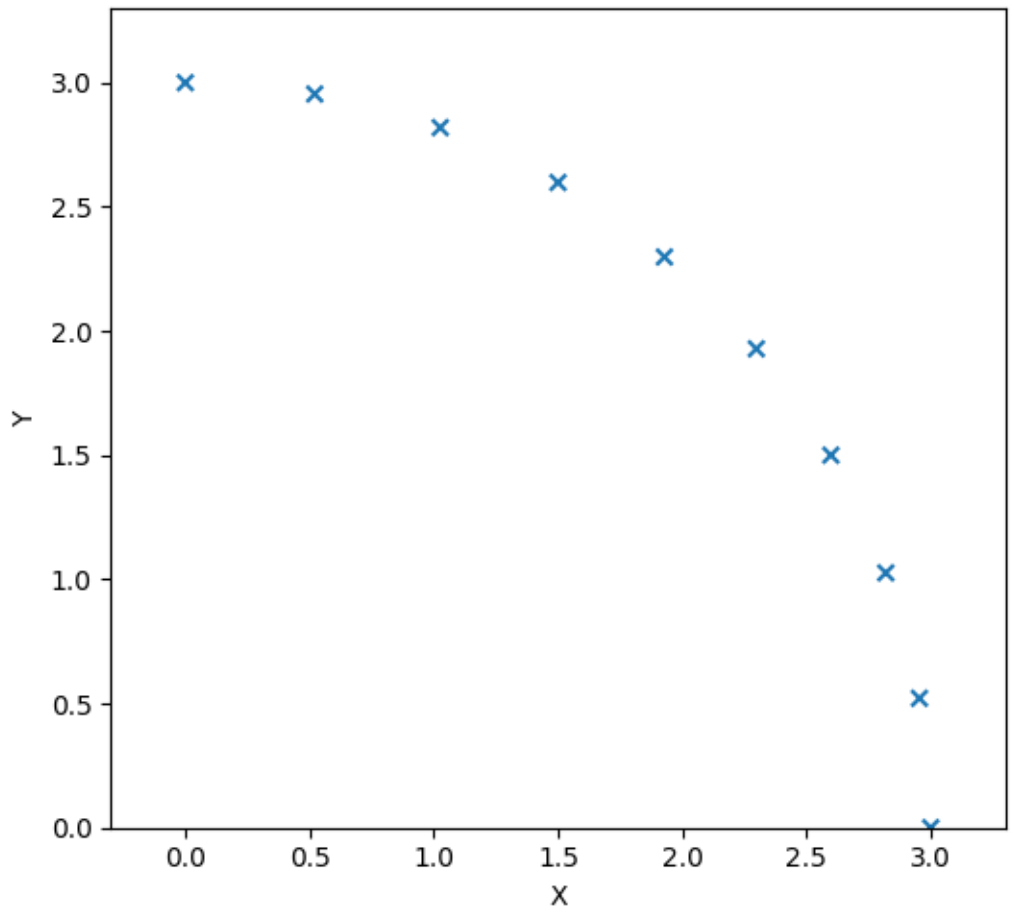


1.2 Bouncing Balls Simulation

[ ]:
```
# 10 Balls located at x = 0, y = 1
x0 = vec(x = math.zeros(instance(balss = 10)), y = math.ones(instance(balls =␣
↪10)))
```

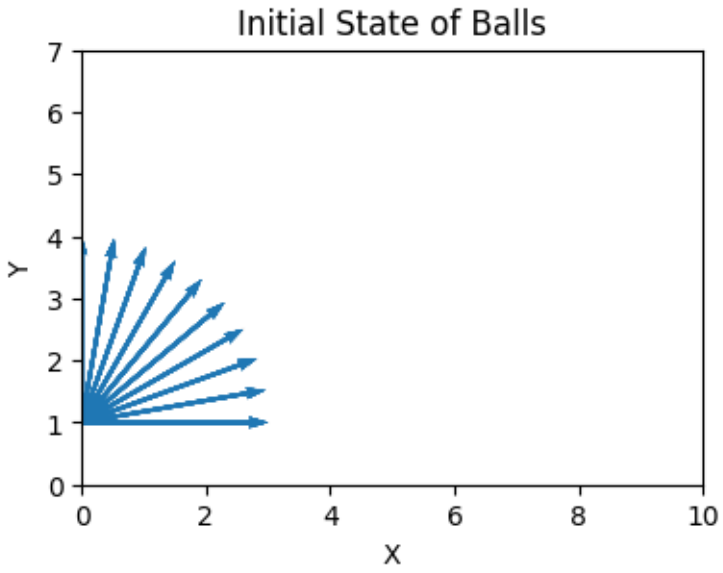
[ ]:
```
# All balls have initial speed of 3
# their angle of attacks linearly spaced between x & y
v0 = vec(
    x = 3 * math.cos(math.linspace(0, PI / 2, instance(balls = 10))),
    y = 3 * math.sin(math.linspace(0, PI / 2, instance(balls = 10))))
plot(v0)
```

[ ]:
```
<Figure size 1200x500 with 1 Axes>
```



[ ]:
```
# store location & velocity on the same object
balls = PointCloud(Sphere(x0, radius=.1), v0, bounds=Box( x = 10, y = 7))
plot(balls, title='Initial State of Balls', size=(4,4))
```

[ ]:
```
<Figure size 400x400 with 1 Axes>
```



[ ]:
```
# air_friction = 0.7
# gravity = 9.81
# dt = 0.1
air_friction = 0.7
gravity = 9.81

# simulation function
def step(balls, dt=.1):
  balls *= math.where(balls.points.vector['y'] < 0, (1, -1), 1) * air_friction␣
 ↪** dt
  return advect.points(balls, balls, dt) + (0, -gravity * dt)
```

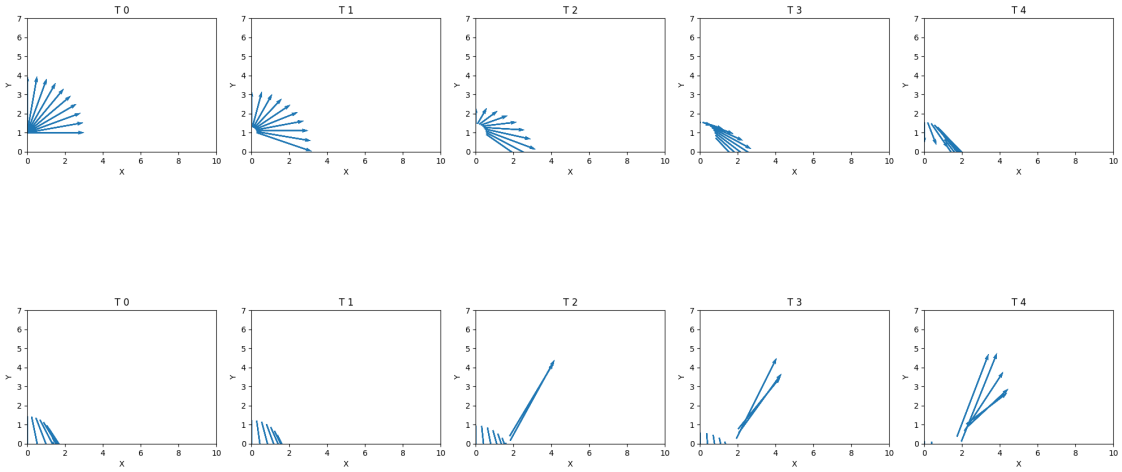
[ ]:
```
# iterate 10s
motion =  iterate(step, batch(t=100), balls)

# animation of result
#plot(field.mask(res), animate='t')
```

[ ]:
```
# plot some represantative results
t1 = motion.t[0:5]
t2 = motion.t[5:10]
```

```
t3 = motion.t[91:96]
t4 = motion.t[-5:]
plot(t1, size=(20,10))
plot(t2, size=(20,10))
```

[ ]: <Figure size 2000x1000 with 5 Axes>




```
plot(t3, size=(20,10))
plot(t4, size=(20,10))
```

[ ]: <Figure size 2000x1000 with 5 Axes>