Exercise_1

May 5, 2025

1 Introduction to ΦFlow

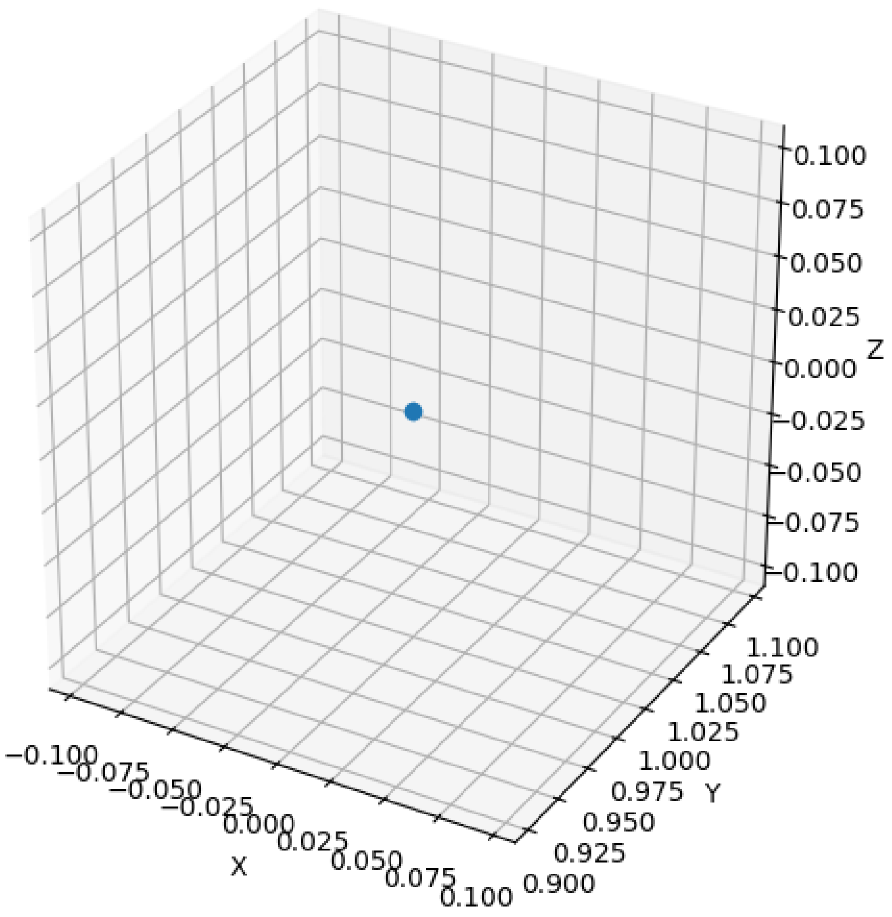
1.1 part 1: Tensors representing physical data

[21]:
```
# Install PhyFlow and Import libraries
%%capture
!pip install --quiet phiflow;
!pip install nbconvert;
!apt-get install texlive texlive-xetex texlive-latex-extra pandoc;
from google.colab import drive
drive.mount("/content/drive");
from phi.torch.flow import *
```

1.1.1 Create and plot the following tensors:

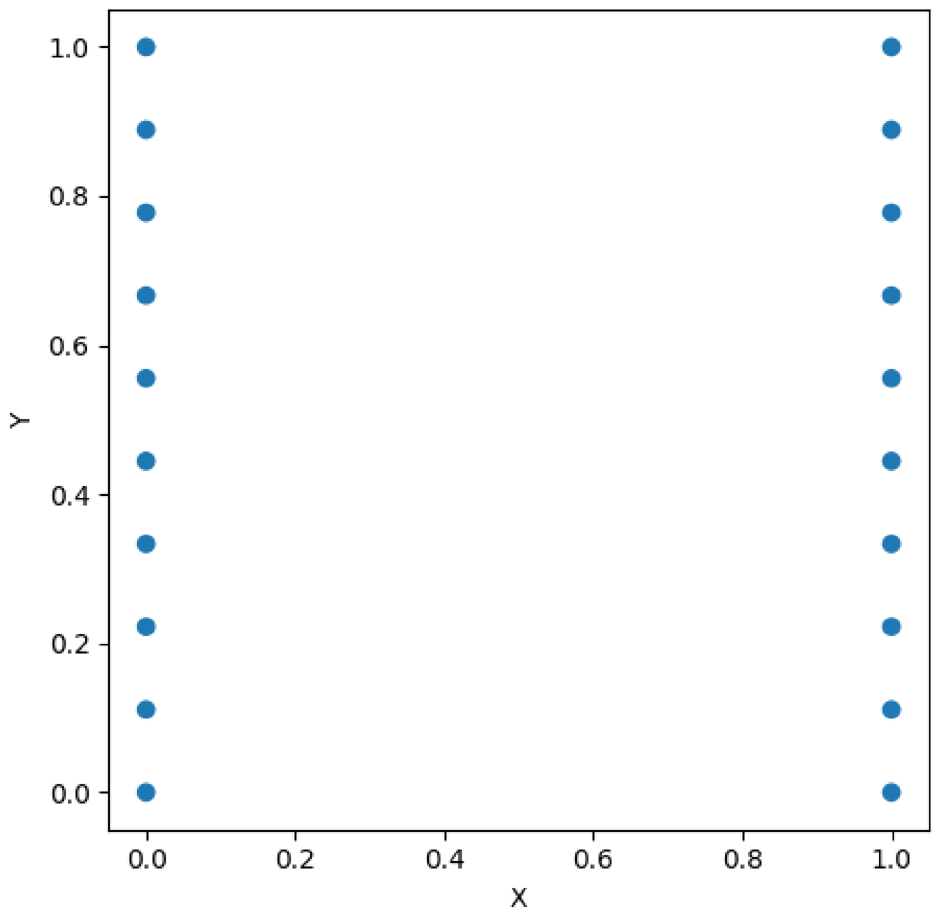
[1] A single vector representing the point x = 0, y = 1, z = 0

[22]:
```
t = tensor([0,1,0], channel(vector='x,y,z'))
#plot(t)
show(t, lib='matplotlib')
```



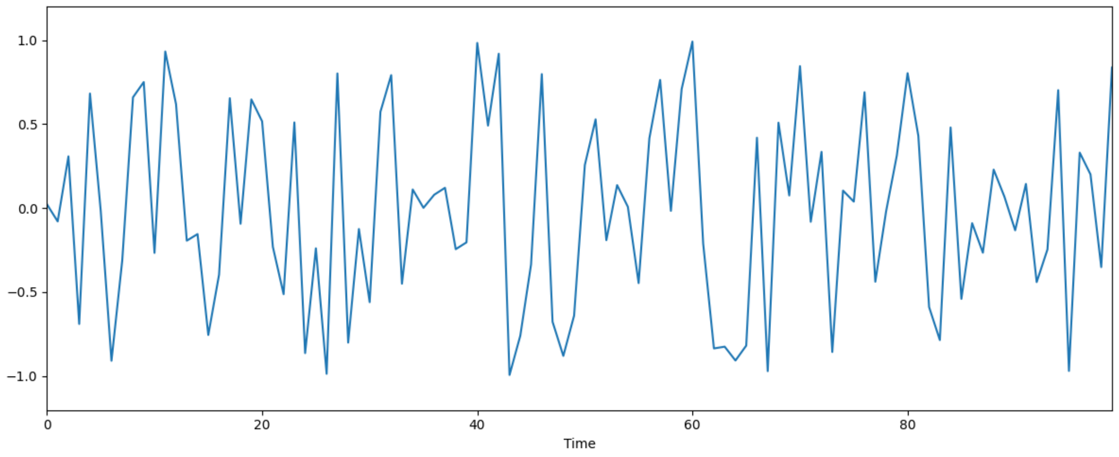
[2] Two columns of points, at x = 0 and x = 1, 10 points per column linearly spaced between 0 and 1.

```
[23]: points = concat([
          stack({'x': math.zeros(instance(i=10)), 'y': linspace(0, 1,␣
        ↪instance(i=10))}, channel('vector')),
          stack({'x': math.ones(instance(i=10)), 'y': linspace(0, 1,␣
        ↪instance(i=10))}, channel('vector'))
      ], dim='i')

      #plot(points)
      show(points)
```



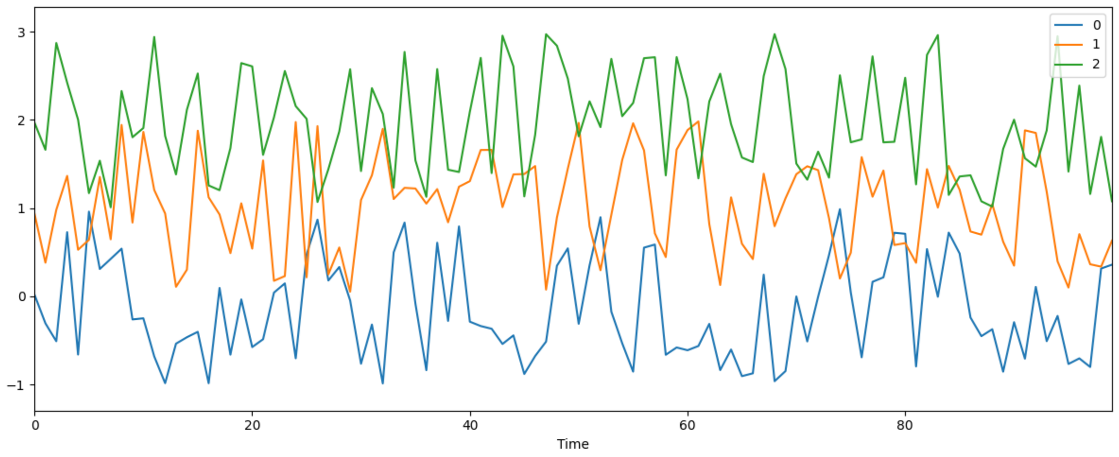
[3] A time-dependent waveform or signal consisting of 100 samples. Each sample is uniformly distributed between-1 and 1.

[24]:
```
t = math.random_uniform(spatial(time=100), low=-1, high=1)
show(t)
```



[4] Three such curves in one plot, centered around 0, 1 and 2, respectively

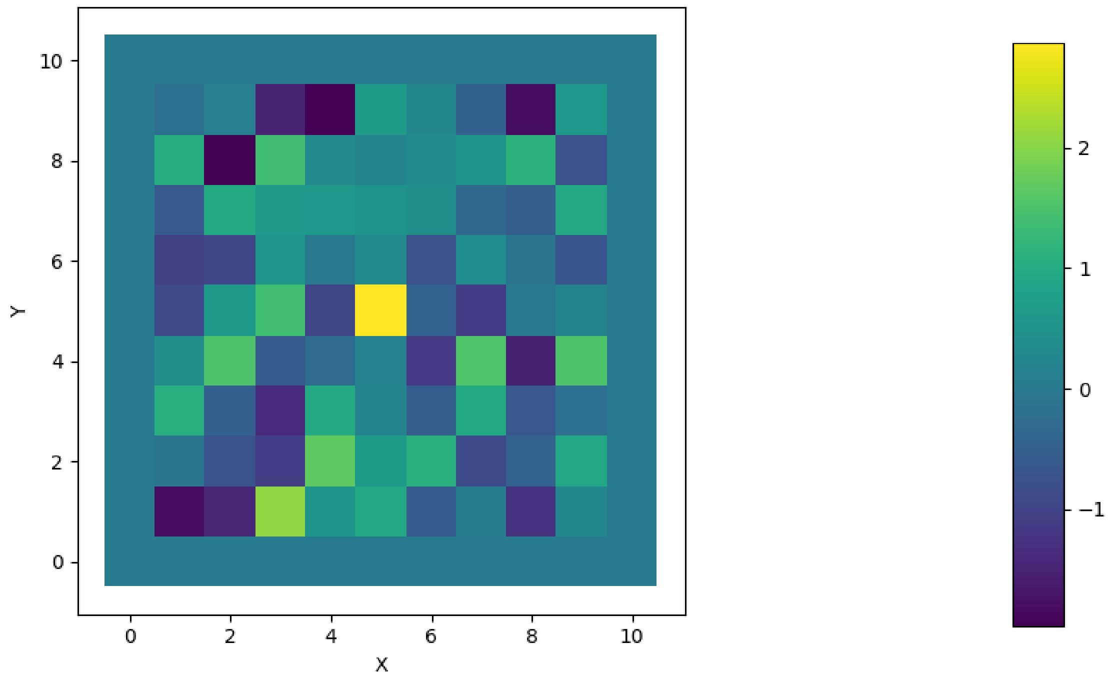
```
[25]: t = stack([math.random_uniform(spatial(time=100), low=-1, high=1), math.
        ↪random_uniform(spatial(time=100), low=0, high=2), math.
        ↪random_uniform(spatial(time=100), low=1, high=3)], channel('wave'))
      show(t)
```



[5] A two-dimensional scalar 10x10 grid. Each value is sampled from a normal distribution but all edge values are zero.

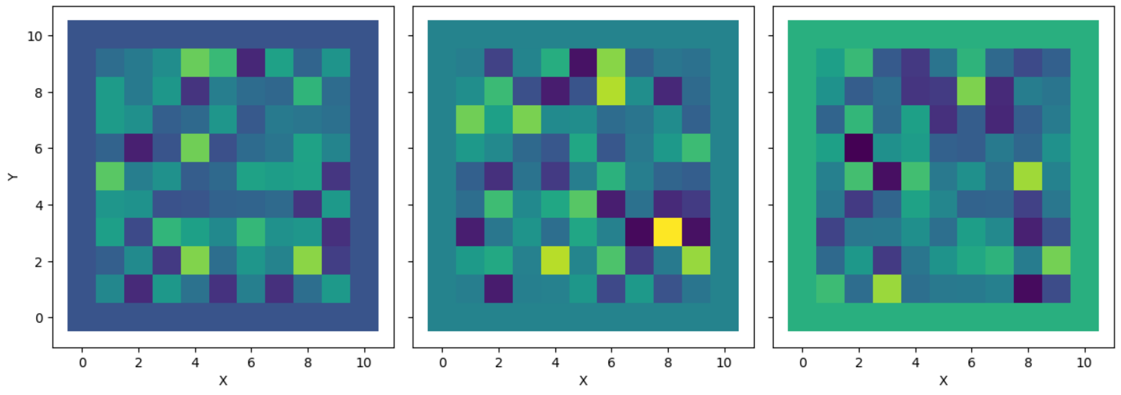
```
[26]: t = math.pad(math.random_normal(spatial(x=9,y=9)), {'x': (1, 1), 'y': (1, 1)},␣
        ↪mode=0)
      show(t)
```

```
/usr/local/lib/python3.11/dist-
packages/phi/vis/_matplotlib/_matplotlib_plots.py:167: UserWarning:

This figure includes Axes that are not compatible with tight_layout, so results
might be incorrect.
```



[6] Three such plots next to each other, with edge values -1, 0, and 1, respectively.

```
[27]: t = [math.pad(math.random_normal(spatial(x=9,y=9)), {'x': (1, 1), 'y': (1, 1)},␣
      ↪mode=m) for m in [-1,0,1] ]
      show(t, show_color_bar=False)
```



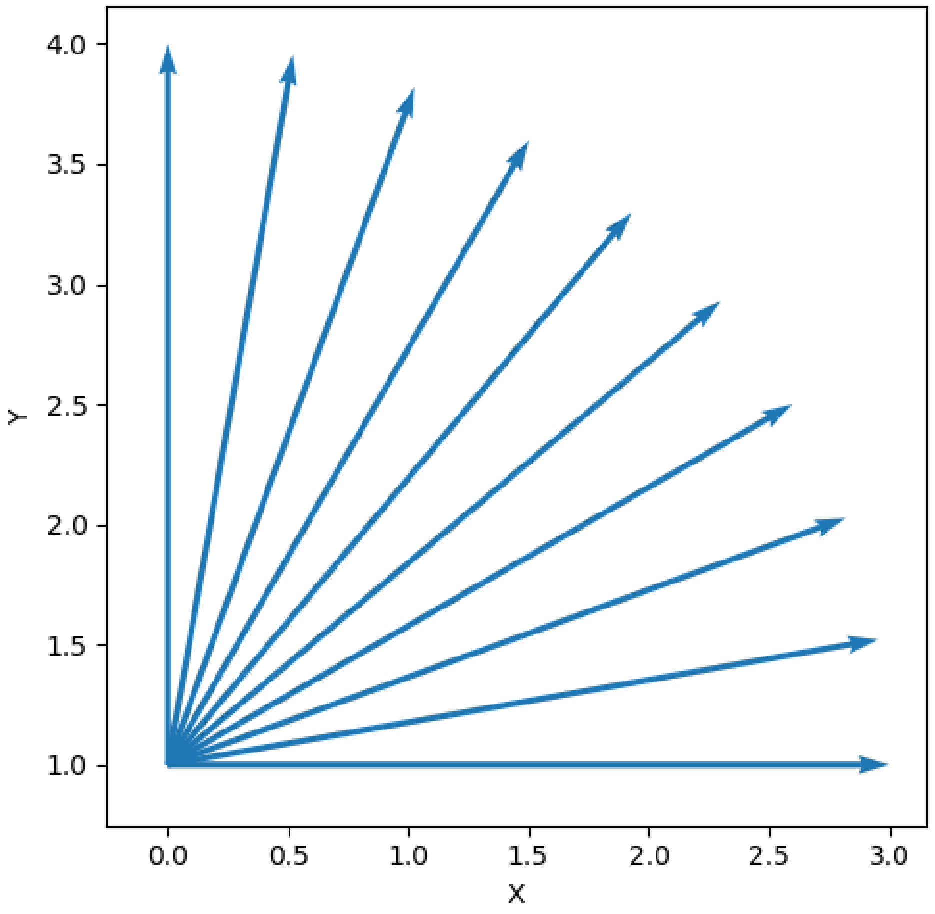
1.2 part 2: Bouncing balls simulation

We want to simulate perfectly round balls with radius 0.1 bouncing up and down in two dimensions due to gravity and an elastic interaction with the floor. Initially, there are 10 balls, all located at x = 0, y = 1 with initial speed 3. Their angles of attack should be linearly spaced between horizontal (positive x) and vertical (positive y). Write a simulation function that takes in the current state and a time increment t. It should compute the linear movement, air friction, gravitational force and elastic collision with the floor at y = 0 and return the next state. For the air friction, you can simply multiply the speed by 0.7 t each step. Run your simulation for about 10 seconds with t 0.1 and plot the final state. Also create a video of the simulation.

Setup of the simulation

[28]:
```
# Define the inital positions and velocities
x0 = stack({'x': zeros(instance(coord=10)), 'y': ones(instance(coord=10))},␣
 ↪channel('vector'))
v0 = stack({'x':3 * cos(linspace(0,math.pi/2,instance(coord=10))), 'y':3 *␣
 ↪sin(linspace(0,math.pi/2,instance(coord=10)))}, channel('vector'))
balls = PointCloud(elements = Sphere(center = x0,radius = .1),values = v0)
print(balls)
show(balls)
```

```
Field[(coord =10, vector =x,y)]
```



Add constraints and define the simulation function

[29]:
```
# Create constraints
floor = Box(x=(-1, 10), y=(-0.1, 0))  # wide and flat floor under y=0
bounds = Box(x=50, y=10)

# Simulate function
def simulate(balls: PointCloud ,dt: float,elasticity: float):
    # Gravitational Acceleration is a constant vector
    g = stack({'x': zeros(instance(coord=10)), 'y': -9.8 *␣
  ↪ones(instance(coord=10))}, channel('vector'))
    # air friction
    v = balls.values*0.7**0.1
    x = balls.geometry.center + v*dt
    v = v + g*dt
    # Floor interaction
    dist, _, normal, *_ = floor.approximate_closest_surface(balls.points)
    bounce = (dist < balls.geometry.bounding_radius()) & (v.vector @ normal < 0)
    impact = -(1+elasticity) * (v.vector @ normal.vector) * normal
    v = math.where(bounce, v + impact, v)
    x = math.clip(balls.points + dt * v, bounds.lower, bounds.upper)
    return balls.shifted_to(x).with_values(v)
```

Create a video iterating the simulation process

[30]:
```
# Run the simulation for 100 time steps
balls = phi.math.iterate(simulate, batch(time=100), balls, f_kwargs={'dt': .
  ↪1,'elasticity' : 1})
# Plot the trajectory as spheres
show([balls.geometry, floor], animate='time', overlay='list', color=[0,1])
```

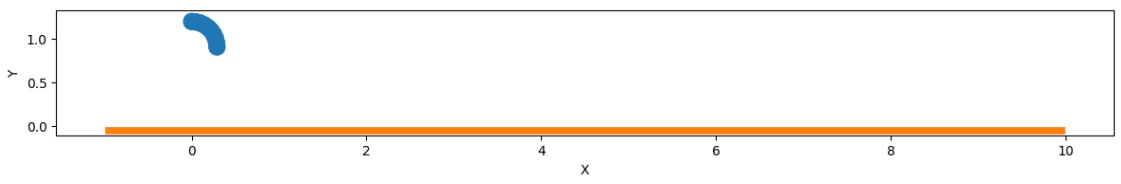
```
/usr/local/lib/python3.11/dist-packages/phiml/backend/_backend.py:1687:
RuntimeWarning:

divide by zero encountered in divide
```

[30]:
```
<IPython.core.display.HTML object>

<Figure size 640x480 with 0 Axes>
```

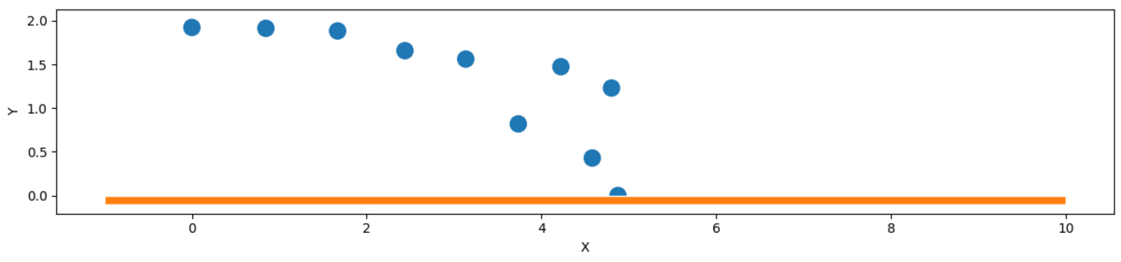
Visualize some frames

[31]:
```
show([balls.geometry.time[1],floor], overlay='list',color=[0,1])
```

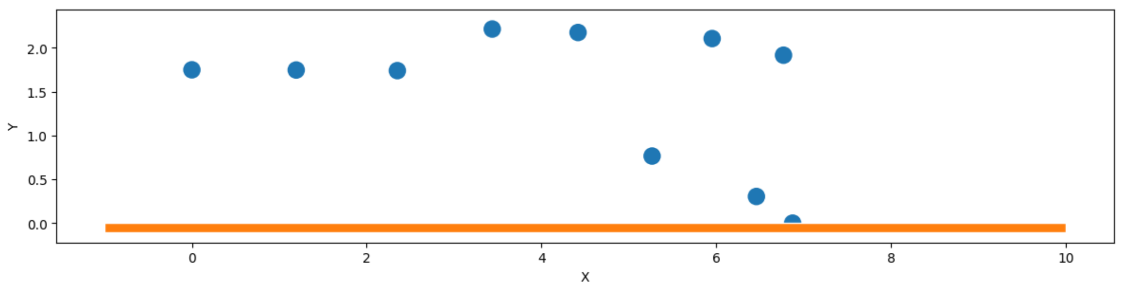
```
/usr/local/lib/python3.11/dist-packages/phiml/backend/_backend.py:1687:
RuntimeWarning:

divide by zero encountered in divide
```



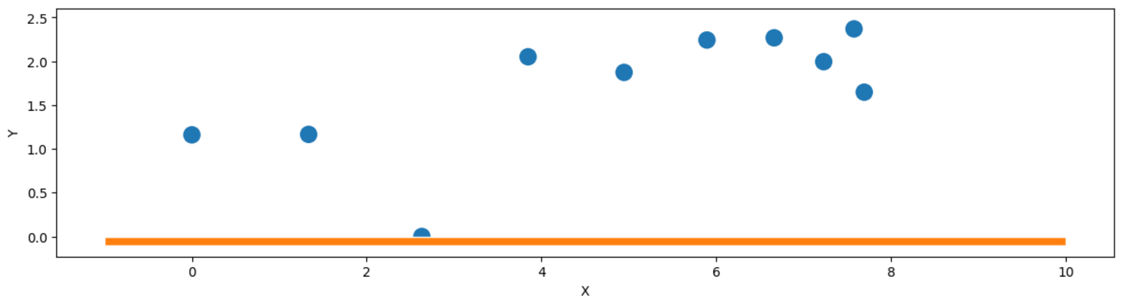
[32]:
```
show([balls.geometry.time[25],floor], overlay='list',color=[0,1])
```



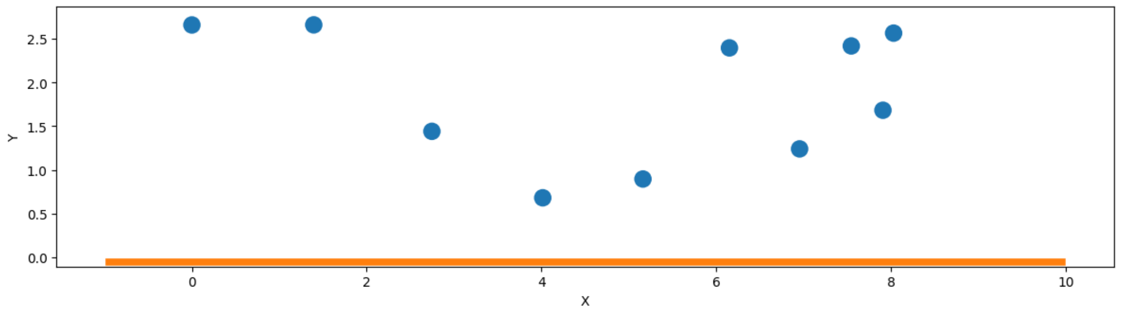
[33]:
```
show([balls.geometry.time[50],floor], overlay='list',color=[0,1])
```



[34]:
```
show([balls.geometry.time[75],floor], overlay='list',color=[0,1])
```



[35]:
```
show([balls.geometry.time[100],floor], overlay='list',color=[0,1])
```



[36]:
```
%%capture
!jupyter nbconvert --to pdf --output /content/drive/MyDrive/Fisica/ADL4P/
 ↪Exercise_1.pdf /content/drive/MyDrive/Fisica/ADL4P/Exercise_1.ipynb
```