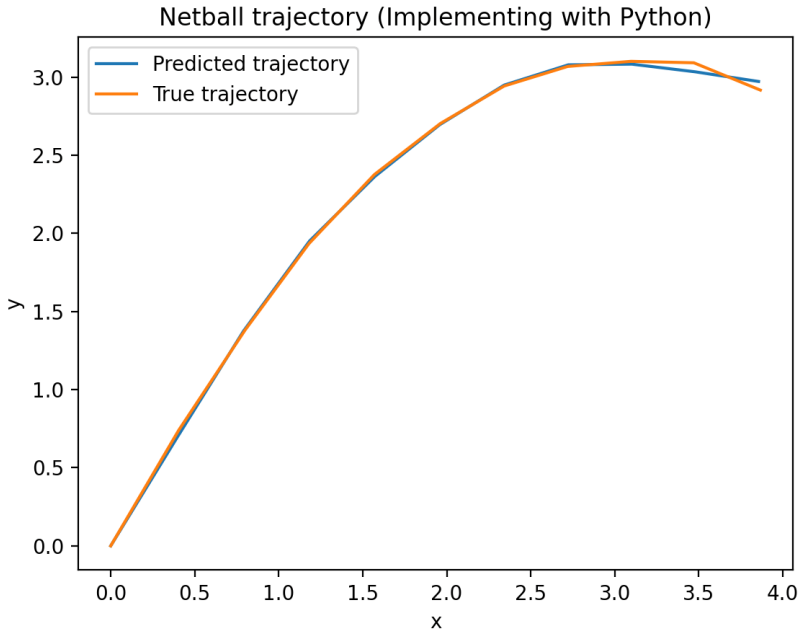
Take netball trajectory prediction case in https://www.chitre.net/EE5311/07/#/example-1-8 as the actual background for verifying the performance of the method.

Implement a Neural ODE in Python (torchdiffeq.odeint):
Code file: netball.py

```
pred_traj = odeint(model, u0[:2], torch.tensor(data['t'].values,
dtype=torch.float32))
```

```
Epoch 1000: Loss = 0.008537416346371174
Execution time for Python(torchdiffeq.odeint) implememt Neural ODEs:
 0.024893999099731445 seconds
1010 allocations: 2131.656 KiB
```

"1010 allocation" indicates that memory was allocated 1010 times during the execution of the Neural ODE function, and "2131.656 KiB" indicates that the total amount of memory allocated this time is 2131.656 KiB.

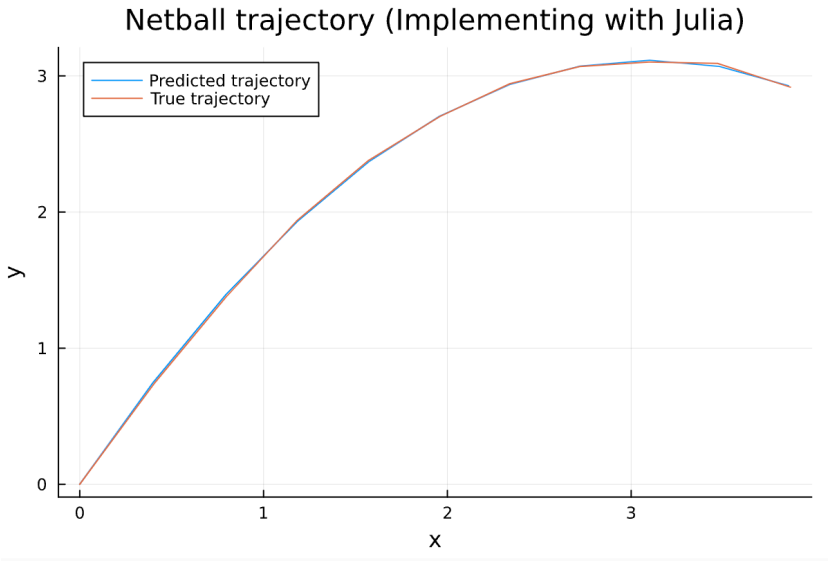


Implement a Neural ODE in Julia:
Code file: netball.jl

```
prob = NeuralODE(dudt_model, (data.t[1], data.t[end]), Tsit5();
saveat=data.t)
```

```
epoch 1000: Loss = 0.0015672721974196568
Execution time for Julia(DiffEqFlux.NeuralODE) implememt Neural ODEs:
  0.004972 seconds (2.55 k allocations: 184.273 KiB)
```

"2.55k allocation" indicates that memory was allocated 2550 times during the execution of the Neural ODE function, and "184.273 KiB" indicates that the total amount of memory allocated this time is 184.273 KiB.



Comparation

	Python(torchdiffeq.odeint)	Julia(DiffEqFlux.NeuralODE)
Execution time(s)	0.024894	0.004972
memory allocation number	1010	2550
size of memory allocation(KiB)	2131.656	184.273

When completing a neural ode function, Julia has a shorter execution time than Python. Julia is more flexible in memory management, although it allocates more times than Python, it allocates less total memory.