```python
# %% imports
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats

rng = np.random.default_rng()
# %% trajectory generation
# scenario parameters
x0 = np.array([np.pi / 2, -np.pi / 100])
Ts = 0.05
K = round(20 / Ts)

# constants
g = 9.81
l = 1
a = g / l
d = 0.5  # dampening
S = 5

# disturbance PDF
process_noise_sampler = lambda: rng.uniform(-S, S)

# dynamic function
def modulo2pi(x, idx=0):
    xmod = x
    xmod[idx] = (xmod[idx] + np.pi) % (2 * np.pi) - np.pi  # wrap to [-pi, pi]
    return xmod


def pendulum_dynamics(x, a, d=0):  # continuous dynamics
    xdot = np.array([x[1], -d * x[1] - a * np.sin(x[0])])
    return xdot


def pendulum_dynamics_discrete(xk, vk, Ts, a, d=0):
    xkp1 = modulo2pi(xk + Ts * pendulum_dynamics(xk, a, d))  # euler discretize
    xkp1[1] += Ts * vk  #  zero orde hold noise
    return xkp1


# sample a trajectory
x = np.zeros((K, 2))
x[0] = x0
for k in range(K - 1):
    v = process_noise_sampler()
    x[k + 1] = pendulum_dynamics_discrete(x[k], v, Ts, a, d)


# vizualize
fig1, axs1 = plt.subplots(2, sharex=True, num=1, clear=True)
axs1[0].plot(x[:, 0])
axs1[0].set_ylabel(r"$\theta$")
axs1[0].set_ylim((-np.pi, np.pi))

axs1[1].plot(x[:, 1])
axs1[1].set_xlabel("Time step")
axs1[1].set_ylabel(r"$\dot \theta$")
```

```
# %% measurement generation

# constants
Ld = 4
Ll = 5
r = 0.25

# noise pdf
measurement_noise_sampler = lambda: rng.triangular(-r, 0, r)

# measurement function
def h(x, Ld, l, Ll):  # measurement function
    lcth = l * np.cos(x[0])
    lsth = l * np.sin(x[0])
    z = np.sqrt((Ld - lcth) ** 2 + (lsth - Ll) ** 2)  # 2norm
    return z


Z = np.zeros(K)
for k, xk in enumerate(x):
    wk = measurement_noise_sampler()
    Z[k] = h(x[k], Ld, l, Ll) + wk


# vizualize
fig2, ax2 = plt.subplots(num=2, clear=True)
ax2.plot(Z)
ax2.set_xlabel("Time step")
ax2.set_ylabel("z")

# %% Task: Estimate using a particle filter

# number of particles to use
N = 1000 #Done


# initialize particles, pretend you do not know where the pendulum starts
px = np.array([
    rng.uniform(-np.pi, np.pi, N), # Done, this represents theta, an angle between -pi and pi. Can
    rng.normal(size=N)*np.pi/3 # Done this is theta dot, should probably start around zero so norm
    ]).T

# initial weights
w = np.full((N,1), 1/N) #weights must be in sum 1.

# allocate some space for resampling particles
pxn = np.zeros_like(px)

# PF transition PDF: SIR proposal, or something you would like to test
PF_dynamic_distribution = scipy.stats.uniform(loc=-S, scale=2 * S)
PF_measurement_distribution = scipy.stats.triang(c=0.5, loc=-r, scale=2 * r)

# initialize a figure for particle animation.
plt.ion()
fig4, ax4 = plt.subplots(num=4, clear=True)
plotpause = 0.01
```

```
sch_particles = ax4.scatter(np.nan, np.nan, marker=".", c="b", label=r"$\hat \theta^n$")
sch_true = ax4.scatter(np.nan, np.nan, c="r", marker="x", label=r"$\theta$")
ax4.set_ylim((-1.5 * l, 1.5 * l))
ax4.set_xlim((-1.5 * l, 1.5 * l))
ax4.set_xlabel("x")
ax4.set_ylabel("y")
th = ax4.set_title(f"theta mapped to x-y")
ax4.legend()

eps = np.finfo(float).eps
for k in range(K):
    print(f"k = {k}")
    # weight update
    for n in range(N):
        dz = Z[k] - h(px[n], Ld, l, Ll)
        w[n] = PF_measurement_distribution.pdf(dz)*w[n]

    sum_weights = np.sum(w)
    w = np.array([weight/sum_weights for weight in w])

    # resample
    # TODO: some pre calculations?
    cumweights = np.cumsum(w)
    indicesout = np.zeros((N, 1))
    noise = rng.random((1,1)) / N
    i = 0
    for n in range(N):
        # find a particle 'i' to pick
        uj = (n)/N +noise
        while  i != len(cumweights) and uj>cumweights[i]:
            i +=1
        # algorithm in the book, but there are other options as well
        pxn[n] = px[i]
    rng.shuffle(pxn, axis=0)  # shuffling because recommended

    #Recommended to replace weights with 1/N
    w.fill(1.0/N)

    # trajecory sample prediction
    for n in range(n):
        vkn = PF_dynamic_distribution.rvs() #hint: PF_dynamic_distribution.rvs
        px[n] = pendulum_dynamics_discrete(pxn[n], vkn, Ts, a) # TODO: particle prediction

    # plot
    sch_particles.set_offsets(np.c_[l * np.sin(pxn[:, 0]), -l * np.cos(pxn[:, 0])])
    sch_true.set_offsets(np.c_[l * np.sin(x[k, 0]), -l * np.cos(x[k, 0])])

    fig4.canvas.draw_idle()
    plt.show(block=False)
    plt.waitforbuttonpress(plotpause)

#plt.waitforbuttonpress()
# %%
```