```
"""
Synthetic 2D Suspended Sediment Concentration (SSC) Modeling using a Gaussian
Plume and sparse fixed sensors.

This script:
    - Generates a physically plausible SSC field using a Gaussian plume model
    - Simulates fixed surface sensor measurements with noise and missing values
    - Reconstructs a 2D SSC map from sparse sensors using RBF interpolation
    - Evaluates reconstruction accuracy against the true field

This is a synthetic, physically plausible plume model intended for data
analysis and spatial reconstruction demonstrations rather than a full
hydrodynamic sediment transport simulation.

Author: Dohyun Kim
Created: Dec 28, 2025
"""
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.interpolate import Rbf

#--------------------------------
# Reproducibility
#--------------------------------
RNG_SEED = 42
rng = np.random.default_rng(RNG_SEED)

#--------------------------------
# Spatial domain (construction site)
#--------------------------------
m,n = 80,80
x = np.arange(n)
y = np.arange(m)
X,Y = np.meshgrid(x,y)

#--------------------------------
# Temporal setup
#--------------------------------
T = 120 # number of time steps
dt = 1 # time step size
times = np.arange(T)*dt

#--------------------------------
# Source location
#--------------------------------
x0,y0 = 25,10

#--------------------------------
# Gaussian plume parameters
#--------------------------------
```

```python
A0 = 200 # initial amplitude
u,v = 0.15,0.05 # advection speed(grid units per time)
sigma0 = 2
D = 0.6
lam = 0.01 # decay rate

#-------------------------------
# Sensor noise and missingness
#-------------------------------
noise_std = 5
missing_rate = 0.03

#-------------------------------
# Gaussian plume model
#-------------------------------
def plume_concentration(X, Y, t, *, x0, y0, A0, u, v, sigma0, D, lam):
    """
    Compute a 2D Gaussian plume with advection, diffusion-like spreading, and
    exponential decay.
    Returns concentration C with same shape as X,Y

    Parameters
    ----------
    X, Y : 2D arrays
        Spatial grid coordinates.
    t : float
        Time step.
    x0, y0 : float
        Source location.
    A0 : float
        Initial concentration amplitude.
    u, v : float
        Advection velocities.
    sigma0 : float
        Initial plume width.
    D : float
        Diffusion coefficient.
    lam : float
        Exponential decay rate.

    Returns
    -------
    C : 2D array
        Suspended sediment concentration field.
    """

    mux = x0 + u*t
    muy = y0 + v*t
    sigma2 = sigma0**2 + 2*D*t
    sigma2 = max(sigma2, 1e-6)
```

```python
    # Gaussian
    expo = -((X - mux)**2 + (Y - muy)**2) / (2*sigma2)
    C = A0*np.exp(-lam*t)*np.exp(expo)
    return C


#--------------------------------
# Sensor placement
#--------------------------------
def generate_sensors(K, *, x0, y0, m, n, u, v, rng):
    """
    Generate fixed sensor locations within the domain.

    Sensors are placed:
        - Near the source
        - Along the downstream direction
        - Near the site boundary

    This configuration mimics realistic monitoring strategies at coastal
    construction sites.
    """

    K1 = int(K*0.4) # near source
    K2 = int(K*0.4) # downstream band
    K3 = K - K1 - K2 # boundary

    sensors = []

    # near source cluster
    for _ in range(K1):
        sx = x0 + rng.normal(0,6)
        sy = y0 + rng.normal(0,6)
        sensors.append((sx,sy))

    # downstream band: place sensors along the flow direction
    # by step along time and put sensors around the plume center line
    for _ in range(K2):
        t = rng.uniform(10,90) # downstream distance around encoded as time
        cx = x0 + u*t
        cy = y0 + v*t
        # small perpendicular jitter
        sensors.append((cx + rng.normal(0,4), cy + rng.normal(0,4)))

    # Boundary sensors: mostly downstream edge + some sides
    for _ in range(K3):
        edge_type = rng.choice(["downstream", "side"])
        if edge_type == "downstream":
            # put near far edge in flow direction
            sx = rng.uniform(n*0.75,n-1)
            sy = rng.uniform(0,m-1)
        else:
```

```python
            sx = rng.choice([0,float(n-1)])
            sy = rng.uniform(0,m-1)
        sensors.append((sx,sy))
    # Clip to domain and return
    sensors = np.array(sensors, dtype=float)
    sensors[:,0] = np.clip(sensors[:,0], 0, n-1)
    sensors[:,1] = np.clip(sensors[:,1], 0, m-1)
    return sensors[:,0], sensors[:,1]


K = 30
sx, sy = generate_sensors(K, x0=x0, y0=y0, m=m, n=n, u=u, v=v, rng=rng)

#-------------------------------
# SSC field visualization
#-------------------------------
def plot_field_with_sensors(C, sx, sy, title="SSC Field"):
    """
    Plot the 2D SSC field and overlay sensor locations.
    """

    plt.figure()
    plt.imshow(C, origin="lower")
    plt.colorbar(label="SSC (a.u.)")
    plt.scatter(sx, sy, s=25, marker="o", edgecolors="k", linewidths=0.5)
    plt.title(title)
    plt.xlabel("x")
    plt.ylabel("y")
    plt.tight_layout()
    plt.show()


t_test = 100
c_test = plume_concentration(X, Y, t_test, x0=x0, y0=y0, A0=A0, u=u, v=v,
                             sigma0=sigma0, D=D, lam=lam)
plot_field_with_sensors(c_test, sx, sy, title=f"SSC Field at t={t_test}")

#-------------------------------
# Synthetic sensor measurements (noise + missing values)
#-------------------------------
records = []

for t in times:
    c = plume_concentration(X, Y, t, x0=x0, y0=y0, A0=A0, u=u, v=v,
                            sigma0=sigma0, D=D, lam=lam)

    # sample at sensor positions (nearest grid point sampling)
    ix = np.rint(sx).astype(int)
    iy = np.rint(sy).astype(int)
    values = c[iy,ix]
```

```
    # add noise
    values = values + rng.normal(0, noise_std, size=K)

    # add missingness
    mask_missing = rng.uniform(0, 1, size=K) < missing_rate
    values = values.astype(float)
    values[mask_missing] = np.nan

    for k in range(K):
        records.append({
            "time": t,
            "sensor_id": k,
            "x": sx[k],
            "y": sy[k],
            "ssc": values[k],
            })

df = pd.DataFrame(records)
#df.head()

#--------------------------------
# Example sensor time series
#--------------------------------
plt.figure()
for sid in [0, 5, 12]:
    sub = df[df["sensor_id"]==sid].sort_values("time")
    plt.plot(sub["time"], sub["ssc"], label=f"sensor {sid}")
plt.legend()
plt.xlabel("time")
plt.ylabel("SSC(a.u.)")
plt.title("Sensor Time Series")
plt.tight_layout()
plt.show()

#--------------------------------
# Extract sensor measurements at a specific time
#--------------------------------
t0 = 60
sub = df[df["time"] == t0].copy()
sub = sub.dropna(subset=["ssc"])

sx0 = sub["x"].to_numpy()
sy0 = sub["y"].to_numpy()
sv0 = sub["ssc"].to_numpy()

#--------------------------------
# Spatial reconstruction using RBF interpolation
#--------------------------------

# Reconstruct the full 2D SSC field using only sparse sensor measurements
rbf = Rbf(sx0, sy0, sv0, function="multiquadric", smooth=5)
```

```
c_rbf = rbf(X,Y)

plt.figure()
plt.imshow(c_rbf, origin="lower")
plt.colorbar(label="SSC (RBF, a.u.)")
plt.scatter(sx0, sy0, s=25, edgecolors="k", linewidths=0.5)
plt.title(f"RBF estimated SSC from sensors (t={t0})")
plt.xlabel("x")
plt.ylabel("y")
plt.tight_layout()
plt.show()

#-------------------------------
# Evaluation: True field vs reconstructed field
#-------------------------------
c_true = plume_concentration(X, Y, t0, x0=x0, y0=y0, A0=A0, u=u, v=v,
                             sigma0=sigma0, D=D, lam=lam)

# RBF error
err = c_rbf - c_true
rmse = np.sqrt(np.nanmean(err**2))
mae = np.nanmean(np.abs(err))

print(f"RBF reconstruction RMSE: {rmse:.2f}")
print(f"RBF reconstruction MAE: {mae:.2f}")

plt.figure()
plt.imshow(err, origin="lower")
plt.colorbar(label="RBF Estimated - True (a.u.)")
plt.scatter(sx0, sy0, s=25, edgecolors="k", linewidths=0.5)
plt.title(f"RBF Error Map (t={t0})")
plt.xlabel("x")
plt.ylabel("y")
plt.tight_layout()
plt.show()
```