**import** numpy as np

**def** directed\_cycle\_A(n):

A = np.zeros((n,n), dtype=**float**)

**for** i **in** **range**(n):

A[i, (i+1)%n] = 1.0

**return** A

**def** undirected\_cycle\_A(n):

A = np.zeros((n,n), dtype=**float**)

**for** i **in** **range**(n):

A[i, (i+1)%n] = 1.0

A[(i+1)%n, i] = 1.0

**return** A

**def** add\_directed\_chord(A, eps=0.2, i=0, j=None):

n = A.shape[0]

**if** j **is** None:

j = n//2

B = A.copy()

B[i, j] += eps

**return** B

**def** D\_out(A):

d = A.**sum**(axis=1)

**if** np.**any**(d == 0):

**raise** ValueError("Found␣sink␣node␣(out-degree␣0).␣Fix␣by␣adding␣small␣outgoing␣weight.")

**return** np.diag(d)

**def** transition\_P(A):

D = D\_out(A)

**return** np.linalg.inv(D) @ A

**def** L\_rw(P):

n = P.shape[0]

**return** np.eye(n) - P

**def** asymmetry\_index(M):

den = np.linalg.norm(M, **ord**=’fro’)

**if** den == 0:

**return** 0.0

**return** np.linalg.norm(M - M.T, **ord**=’fro’) / den

**def** departure\_from\_normality(M):

Mf = np.linalg.norm(M, **ord**=’fro’)

**if** Mf == 0:

**return** 0.0

MMstar = M @ M.conj().T

MstarM = M.conj().T @ M

**return** np.linalg.norm(MMstar - MstarM, **ord**=’fro’) / (Mf\*\*2)

**def** bgft\_decomposition(P):

lam, V = np.linalg.eig(P)

Vinv = np.linalg.inv(V)

Ustar = Vinv

**return** lam, V, Ustar

**def** diffusion\_filter\_matrix(P, tau=2.0):

*# h(lambda) = exp(-tau\*(1-lambda))*

lam, V, Ustar = bgft\_decomposition(P)

h = np.exp(-tau\*(1.0 - lam))

H = V @ np.diag(h) @ Ustar

**return** H

**def** sample\_operator(n, M):

m = **len**(M)

Pm = np.zeros((m,n), dtype=**float**)

**for** r, idx **in** **enumerate**(M):

Pm[r, idx] = 1.0

**return** Pm

**def** reconstruct\_bandlimited(P, Omega, M, x, noise=0.0, seed=0):

np.random.seed(seed)

lam, V, \_ = bgft\_decomposition(P)

V\_O = V[:, Omega]

Pm = sample\_operator(P.shape[0], M)

y = Pm @ x

**if** noise > 0:

y = y + noise \* np.random.randn(\*y.shape)

B = Pm @ V\_O

c\_hat, \*\_ = np.linalg.lstsq(B, y, rcond=None)

x\_hat = V\_O @ c\_hat

relerr = np.linalg.norm(x\_hat - x) / np.linalg.norm(x)

condB = np.linalg.cond(B)

**return** x\_hat, relerr, condB

**if** \_\_name\_\_ == "\_\_main\_\_":

n = 64

A\_und = undirected\_cycle\_A(n)

A\_dir = directed\_cycle\_A(n)

A\_per = add\_directed\_chord(A\_dir, eps=20, i=0, j=n//2)

**for** name, A **in** [("undirected", A\_und), ("directed", A\_dir), ("perturbed", A\_per)]:

P = transition\_P(A)

lam, V, \_ = bgft\_decomposition(P)

**print**(name,

"alpha(P)=", asymmetry\_index(P),

"delta(P)=", departure\_from\_normality(P),

"cond(V)=", np.linalg.cond(V),

"rho(P)~=", np.**max**(np.**abs**(lam)))

*# Create a bandlimited signal on perturbed digraph*

P = transition\_P(A\_per)

lam, V, \_ = bgft\_decomposition(P)

K = 8

*# Choose "low diffusion-frequency": largest Re(lambda) (closest to 1)*

Omega = np.argsort(-np.real(lam))[:K]

c = np.random.randn(K) + 1j\*np.random.randn(K)

x = V[:, Omega] @ c

m = 20

M = np.sort(np.random.choice(n, size=m, replace=False))

x\_hat, relerr, condB = reconstruct\_bandlimited(P, Omega, M, x, noise=0.0)

**print**("RelErr=", relerr, "cond(P\_M␣V\_Omega)=", condB)

*# Diffusion smoothing example*

H = diffusion\_filter\_matrix(P, tau=2.0)

x\_smooth = H @ x

**print**("||x||2␣=", np.linalg.norm(x), "||Hx||2␣=", np.linalg.norm(x\_smooth))