Supplementary Material S2

Analysis code

Butterflies (Lepidoptera: Papilionoidea) of the southern Bolkar Mountains and adjacent Çukurova plain (Mersin, Türkiye): richness and turnover along a 2–2200 m elevational gradient. Yusuf Hüseyinoğlu.

The script below reproduces every value reported in Sections 3.1, 3.3, 3.5, 3.6 and 3.7 and in Section 4.3 of the manuscript, from the occurrence data of Supplementary Material S1. It requires Python 3.12 with NumPy and SciPy only; each estimator is implemented directly from its primary source rather than through a package, so that the computation can be checked line by line against the published formulae. To run it, save the two tables of S1 as comma-separated files named S1_incidence_matrix.csv and S1_events.csv in the same directory, save the text below as a plain-text file with the extension .py, and execute it.

```
"""
Supplementary Material S2
=========================
Analysis code for:

    Butterflies (Lepidoptera: Papilionoidea) of the southern Bolkar Mountains
    and adjacent Cukurova plain (Mersin, Turkiye): richness and turnover
    along a 2-2200 m elevational gradient

All analyses reported in the manuscript are reproduced by this script from the
species-by-event incidence matrix (Supplementary Material S1). Only NumPy and
SciPy are required; every estimator is implemented directly from the primary
source cited in the manuscript, so the computation can be checked line by line
against the published formulae.

    Python 3.12, NumPy 2.x, SciPy 1.x

Input
-----
S1_incidence_matrix.csv
    Rows    : species (143)
    Columns : species, provisional, then one column per sampling event (71),
              named by the locality numbers of Table 1
    Values  : 1 = recorded, 0 = not recorded
    provisional : yes / no, as defined in Section 2.5 of the manuscript

S1_events.csv
    One row per sampling event, with columns:
    event, district, locality, elevation_m, belt, date_iso, month, protected

Author: Yusuf Huseyinoglu
"""


import math
import numpy as np
from scipy import stats
from scipy.optimize import minimize_scalar

# ---------------------------------------------------------------------
# 1. Richness estimation and inventory completeness
# ---------------------------------------------------------------------

def chao2(X):
    """Chao2 incidence-based richness estimator (Chao 1987).

    X : 2-D array, species x sampling units, 0/1
    Returns (S_obs, Chao2, Q1, Q2) where Q1 and Q2 are the numbers of species
    found in exactly one and exactly two sampling units.
    """
    T = X.shape[1]
    Y = X.sum(axis=1)
    S_obs = int((Y > 0).sum())
    Q1 = int((Y == 1).sum())
    Q2 = int((Y == 2).sum())
    if Q2 > 0:
        est = S_obs + ((T - 1) / T) * (Q1 ** 2) / (2 * Q2)
    else:                                   # bias-corrected form when Q2 = 0
        est = S_obs + ((T - 1) / T) * Q1 * (Q1 - 1) / 2
    return S_obs, est, Q1, Q2


def sample_coverage(X):
```

```
    """Incidence-based sample coverage (Chao & Jost 2012, eq. 4a).

    The proportion of the total incidence of the assemblage that is
    represented by the species detected.
    """
    T = X.shape[1]
    Y = X.sum(axis=1)
    U = int(Y.sum())                          # total incidences
    Q1 = int((Y == 1).sum())
    Q2 = int((Y == 2).sum())
    if Q1 == 0 or U == 0:
        return 1.0
    denom = (T - 1) * Q1 + 2 * Q2
    if denom == 0:
        return 1.0
    return 1 - (Q1 / U) * ((T - 1) * Q1 / denom)


def rarefy_incidence(X, t):
    """Expected species richness for t sampling units (Colwell et al. 2012).

    Sample-based rarefaction, computed exactly by the hypergeometric
    formulation rather than by resampling.
    """
    T = X.shape[1]
    Y = X.sum(axis=1)
    Y = Y[Y > 0]
    expected_absent = sum(
        math.comb(T - int(y), t) / math.comb(T, t)
        for y in Y if T - y >= t
    )
    return len(Y) - expected_absent


def rarefy_sd(X, t):
    """Standard deviation of the rarefied richness (Colwell et al. 2012)."""
    T = X.shape[1]
    Y = X.sum(axis=1)
    Y = Y[Y > 0]
    p = [math.comb(T - int(y), t) / math.comb(T, t) for y in Y if T - y >= t]
    return math.sqrt(max(sum(a * (1 - a) for a in p), 0.0))


# ---------------------------------------------------------------------
# 2. Richness along the gradient: negative-binomial GLM
# ---------------------------------------------------------------------

def negbin_glm(design, y, max_iter=200, tol=1e-10):
    """Negative-binomial GLM with log link.

    Coefficients are estimated by iteratively reweighted least squares;
    the dispersion parameter theta is re-estimated by maximum likelihood
    at each iteration. Used in preference to Poisson because the per-event
    richness data are overdispersed.

    Returns (beta, se, theta, loglik).
    """
    beta = np.zeros(design.shape[1])
    beta[0] = math.log(max(y.mean(), 0.1))
    theta = 5.0
    for _ in range(max_iter):
        eta = design @ beta
        mu = np.exp(np.clip(eta, -20, 20))
        w = mu / (1 + mu / theta)
        z = eta + (y - mu) / np.maximum(mu, 1e-8)
        W = np.diag(w)
        try:
            beta_new = np.linalg.solve(
                design.T @ W @ design + 1e-8 * np.eye(design.shape[1]),
                design.T @ W @ z)
        except np.linalg.LinAlgError:
            break
        converged = np.max(np.abs(beta_new - beta)) < tol
        beta = beta_new
        mu = np.exp(np.clip(design @ beta, -20, 20))
        theta = minimize_scalar(
            lambda t: -np.sum(stats.nbinom.logpmf(
```

```
                y.astype(int), max(t, 1e-3), max(t, 1e-3) / (max(t, 1e-3) + mu))),
            bounds=(0.05, 500), method='bounded').x
        if converged:
            break
    eta = design @ beta
    mu = np.exp(np.clip(eta, -20, 20))
    W = np.diag(mu / (1 + mu / theta))
    cov = np.linalg.inv(design.T @ W @ design + 1e-8 * np.eye(design.shape[1]))
    se = np.sqrt(np.diag(cov))
    loglik = np.sum(stats.nbinom.logpmf(
        y.astype(int), theta, theta / (theta + mu)))
    return beta, se, theta, loglik


# ---------------------------------------------------------------------
# 3. Beta diversity and its components
# ---------------------------------------------------------------------

def sorensen_components(a, b):
    """Partition pairwise Sorensen dissimilarity (Baselga 2010).

    Returns (beta_sor, beta_sim, beta_sne):
        beta_sor : total dissimilarity
        beta_sim : turnover component (Simpson dissimilarity)
        beta_sne : nestedness-resultant component
    """
    A = int(np.sum((a == 1) & (b == 1)))      # shared species
    B = int(np.sum((a == 1) & (b == 0)))      # unique to a
    C = int(np.sum((a == 0) & (b == 1)))      # unique to b
    if 2 * A + B + C == 0:
        return 0.0, 0.0, 0.0
    beta_sor = (B + C) / (2 * A + B + C)
    beta_sim = min(B, C) / (A + min(B, C)) if (A + min(B, C)) > 0 else 0.0
    return beta_sor, beta_sim, beta_sor - beta_sim


def dissimilarity_matrices(X):
    """Pairwise Sorensen, turnover and nestedness matrices for all events."""
    M = X.shape[1]
    D_sor = np.zeros((M, M))
    D_sim = np.zeros((M, M))
    D_sne = np.zeros((M, M))
    for i in range(M):
        for j in range(i + 1, M):
            s, t, n = sorensen_components(X[:, i], X[:, j])
            D_sor[i, j] = D_sor[j, i] = s
            D_sim[i, j] = D_sim[j, i] = t
            D_sne[i, j] = D_sne[j, i] = n
    return D_sor, D_sim, D_sne


# ---------------------------------------------------------------------
# 4. PERMANOVA
# ---------------------------------------------------------------------

def permanova(D, groups, n_perm=4999, seed=1):
    """Permutational multivariate analysis of variance (Anderson 2001).

    D      : symmetric distance matrix
    groups : 1-D array of group labels, one per row of D

    The pseudo-F statistic is computed from the Gower-centred matrix, and
    its null distribution obtained by permuting the group labels.
    Returns (pseudo_F, p_value).
    """
    rng = np.random.default_rng(seed)
    n = len(groups)
    A = -0.5 * D ** 2
    I = np.eye(n)
    ones = np.ones((n, n)) / n
    G = (I - ones) @ A @ (I - ones)           # Gower-centred

    def pseudo_F(g):
        SST = np.trace(G)
        SSW = 0.0
        for level in np.unique(g):
            idx = np.where(g == level)[0]
```

```
            Gi = G[np.ix_(idx, idx)]
            SSW += np.trace(Gi) - Gi.sum() / len(idx)
        a = len(np.unique(g))
        return ((SST - SSW) / (a - 1)) / (SSW / (n - a))

    F_obs = pseudo_F(groups)
    count = sum(pseudo_F(rng.permutation(groups)) >= F_obs
                for _ in range(n_perm))
    return F_obs, (count + 1) / (n_perm + 1)


# ---------------------------------------------------------------------
# 5. Indicator species analysis
# ---------------------------------------------------------------------

def indval(X, groups, target, n_perm=999, seed=7):
    """Indicator value analysis (Dufrene & Legendre 1997).

    IndVal = A x B, where
        A = specificity : the proportion of the occurrences of the species
            that fall within the target group
        B = fidelity    : the proportion of sampling units of the target
            group in which the species occurs

    Significance is assessed by permuting the group labels.
    Returns a list of (species_index, IndVal, p_value).
    """
    rng = np.random.default_rng(seed)
    perms = [rng.permutation(groups) for _ in range(n_perm)]
    in_group = groups == target
    n_group = in_group.sum()
    results = []
    for i in range(X.shape[0]):
        total = X[i].sum()
        if total == 0:
            continue
        A = X[i, in_group].sum() / total
        B = X[i, in_group].sum() / n_group
        iv = A * B
        if iv <= 0:
            continue
        count = 0
        for pg in perms:
            pin = pg == target
            npin = max(pin.sum(), 1)
            if (X[i, pin].sum() / total) * (X[i, pin].sum() / npin) >= iv:
                count += 1
        results.append((i, iv, (count + 1) / (n_perm + 1)))
    return results


# ---------------------------------------------------------------------
# 6. Reproducing the results reported in the manuscript
# ---------------------------------------------------------------------

def main(matrix_csv='S1_incidence_matrix.csv', events_csv='S1_events.csv'):
    import csv

    with open(matrix_csv, encoding='utf-8') as f:
        rows = list(csv.reader(f))
    header, body = rows[0], rows[1:]
    species = [r[0] for r in body]
    provisional = np.array([r[1] == 'yes' for r in body])
    X = np.array([[int(v) for v in r[2:]] for r in body])

    with open(events_csv, encoding='utf-8') as f:
        events = list(csv.DictReader(f))
    elevation = np.array([float(e['elevation_m']) for e in events])
    month = np.array([e['month'] for e in events])
    protected = np.array([e['protected'] == 'yes' for e in events])

    def belt(a):
        return 0 if a <= 400 else 1 if a <= 1000 else 2 if a <= 1500 else 3
    belts = np.array([belt(a) for a in elevation])
    BELT_NAMES = ['2-400 m', '401-1000 m', '1001-1500 m', '1501-2200 m']

    print(f'Matrix: {X.shape[0]} species x {X.shape[1]} sampling events, '
```

```
      f'{int(X.sum())} records')
print(f'        {int(provisional.sum())} species flagged as provisional, '
      f'{int(protected.sum())} events inside protected areas\n')

# --- Section 3.1 ---------------------------------------------------
S_obs, ch2, Q1, Q2 = chao2(X)
print('3.1  Inventory completeness')
print(f'     S_obs = {S_obs}, Q1 = {Q1}, Q2 = {Q2}')
print(f'     Chao2 = {ch2:.1f}   undetected = {ch2 - S_obs:.1f}')
print(f'     sample coverage = {sample_coverage(X) * 100:.1f} %')
print(f'     completeness    = {S_obs / ch2 * 100:.1f} %')
for t in (20, 40, X.shape[1]):
    print(f'     rarefied S({t}) = {rarefy_incidence(X, t):.1f}')

# --- Section 3.3 ---------------------------------------------------
richness = X.sum(axis=0).astype(float)
rho, p = stats.spearmanr(elevation, richness)
print('\n3.3  Richness along the gradient')
print(f'     per-event richness: mean {richness.mean():.1f}, '
      f'range {int(richness.min())}-{int(richness.max())}')
print(f'     Spearman rho = {rho:.3f}, p = {p:.4g}')

elev_z = (elevation - elevation.mean()) / elevation.std()
MONTHS = ['Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov']
month_dummies = np.column_stack(
    [(month == m).astype(float) for m in MONTHS[1:]])

b1, se1, th1, ll1 = negbin_glm(
    np.column_stack([np.ones(len(richness)), elev_z]), richness)
b2, se2, th2, ll2 = negbin_glm(
    np.column_stack([np.ones(len(richness)), elev_z, month_dummies]), richness)
z1, z2 = b1[1] / se1[1], b2[1] / se2[1]
print(f'     NB-GLM  ~ elevation          : beta = {b1[1]:+.3f} '
      f'(SE {se1[1]:.3f}), z = {z1:+.2f}, '
      f'p = {2 * (1 - stats.norm.cdf(abs(z1))):.3g}')
print(f'     NB-GLM  ~ elevation + month  : beta = {b2[1]:+.3f} '
      f'(SE {se2[1]:.3f}), z = {z2:+.2f}, '
      f'p = {2 * (1 - stats.norm.cdf(abs(z2))):.3g}')
lr = 2 * (ll2 - ll1)
print(f'     LR test for month: chi2 = {lr:.2f}, df = '
      f'{month_dummies.shape[1]}, '
      f'p = {1 - stats.chi2.cdf(lr, month_dummies.shape[1]):.3g}')

n_min = min(int((belts == b).sum()) for b in range(4))
print(f'\n     Effort-standardised richness (rarefied to {n_min} events)')
for b in range(4):
    Xb = X[:, belts == b]
    obs = int((Xb.sum(axis=1) > 0).sum())
    print(f'       {BELT_NAMES[b]:14} S_obs = {obs:>3}   '
          f'rarefied = {rarefy_incidence(Xb, n_min):.1f} '
          f'+/- {1.96 * rarefy_sd(Xb, n_min):.1f}   '
          f'coverage = {sample_coverage(Xb) * 100:.1f} %   '
          f'Chao2 = {chao2(Xb)[1]:.1f}')

# --- Section 3.5 ---------------------------------------------------
D_sor, D_sim, D_sne = dissimilarity_matrices(X)
iu = np.triu_indices(X.shape[1], 1)
print('\n3.5  Beta diversity')
print(f'     total       = {D_sor[iu].mean():.3f}')
print(f'     turnover    = {D_sim[iu].mean():.3f} '
      f'({D_sim[iu].mean() / D_sor[iu].mean() * 100:.1f} %)')
print(f'     nestedness  = {D_sne[iu].mean():.3f} '
      f'({D_sne[iu].mean() / D_sor[iu].mean() * 100:.1f} %)')
d_elev = np.abs(elevation[:, None] - elevation[None, :])[iu]
r, p = stats.spearmanr(d_elev, D_sor[iu])
print(f'     dissimilarity ~ elevational difference: '
      f'rho = {r:.3f}, p = {p:.3g}')
for label, g in (('belt', belts), ('month', month)):
    F, p = permanova(D_sor, g)
    print(f'     PERMANOVA ~ {label:6}: pseudo-F = {F:.3f}, p = {p:.4f}')

# --- Section 3.6 ---------------------------------------------------
print('\n3.6  Indicator species (IndVal > 0.30, p < 0.05; '
      'provisional records excluded, Section 2.5)')
keep = ~provisional
X_ind = X[keep]
species_ind = [s for s, k in zip(species, keep) if k]
for b in range(4):
```

```
        hits = [(species_ind[i], iv, p) for i, iv, p in indval(X_ind, belts, b)
                if iv > 0.30 and p < 0.05]
        hits.sort(key=lambda z: -z[1])
        if hits:
            print(f'     {BELT_NAMES[b]}')
            for name, iv, p in hits:
                print(f'       {name:32} IndVal = {iv:.3f}, p = {p:.3f}')

    # --- Section 3.7: protected areas ------------------------------
    print('\n3.7  Protected areas')
    high = elevation >= 1000
    n_pa = int(protected.sum())
    for label, sub in (('inside', X[:, protected]), ('outside', X[:, ~protected])):
        obs, est, _, _ = chao2(sub)
        print(f'     {label:8} S_obs = {obs:>3}   Chao2 = {est:.1f}   '
              f'coverage = {sample_coverage(sub) * 100:.1f} %   '
              f'completeness = {obs / est * 100:.1f} %')
    Xi = X[:, protected & high]
    Xo = X[:, (~protected) & high]
    n_min = min(Xi.shape[1], Xo.shape[1])
    print(f'     above 1000 m only, rarefied to {n_min} events:')
    for label, sub in (('inside', Xi), ('outside', Xo)):
        print(f'       {label:8} {rarefy_incidence(sub, n_min):.1f} '
              f'+/- {1.96 * rarefy_sd(sub, n_min):.1f}')
    a = X[:, protected & high].sum(axis=0)
    b = X[:, (~protected) & high].sum(axis=0)
    u, p = stats.mannwhitneyu(a, b, alternative='two-sided')
    print(f'       per-event richness {a.mean():.1f} inside vs {b.mean():.1f} outside, '
          f'U = {u:.0f}, p = {p:.3f}')
    only_in = set(np.where(X[:, protected].sum(axis=1) > 0)[0]) - \
              set(np.where(X[:, ~protected].sum(axis=1) > 0)[0])
    print(f'     species recorded only inside protected areas: {len(only_in)} '
          f'({sum(1 for i in only_in if provisional[i])} of them provisional)')

    # --- Discussion 4.3: robustness to the provisional records ------
    print('\n4.3  Robustness check: analyses re-run without provisional records')
    Xk = X[~provisional]
    for label, Xs in (('all species', X), ('excluding provisional', Xk)):
        D_sor, D_sim, _ = dissimilarity_matrices(Xs)
        iu = np.triu_indices(Xs.shape[1], 1)
        print(f'     {label:22} n = {Xs.shape[0]:>3}   '
              f'beta_sor = {D_sor[iu].mean():.3f}   '
              f'turnover = {D_sim[iu].mean() / D_sor[iu].mean() * 100:.1f} %')
    rho, p = stats.spearmanr(elevation, Xk.sum(axis=0))
    print(f'     richness ~ elevation, provisional excluded: '
          f'rho = {rho:.3f}, p = {p:.3g}')


if __name__ == '__main__':
    main()
```