Supplementary Material

Detectability of landscape-genetic signal in fragmented landscapes: habitat area, configuration, dispersal, and the dual role of corridors

Samuel A. Cushman

This Supplementary Material accompanies the manuscript and its companion paper (Cushman, in review). It is organised in four parts: (S1) the design and CDPOP parameterisation, restated so the paper can be evaluated on its own; (S2) a note reconciling the archived pipeline scripts with the parameters actually used for the 1,500 analysed simulations; (S3) verified summary statistics; and (S4) the full analysis pipeline, comprising the landscape-generation, sampling, cost-distance and CDPOP-input scripts shared with the companion paper, followed by the landscape-genetic (Mantel + MLPE) and synthesis (ANOVA + figure) scripts specific to the present paper.

S1. Design and CDPOP parameterisation

The factorial design crosses four factors: habitat area (5 levels: 10, 20, 30, 40, 50% of the 100 × 100 landscape); habitat clumping (5 levels of the Gaussian smoothing standard deviation σ nearest to 3.01, 4.00, 5.00, 6.00, 7.00); connectivity treatment (2 levels: base vs. area-conserved linked); and dispersal threshold (3 levels: 50, 100, 150 cost units). Each of the 5 × 5 × 2 × 3 = 150 combinations was replicated 10 times with independent landscape and CDPOP random seeds, for 1,500 simulations in total.

CDPOP parameterisation (identical across all treatments unless noted): 100 individuals initialised at the 100 within-habitat sample locations; 200 non-overlapping generations; density-dependent exponential dynamics with fixed carrying capacity K = 100; random mating with equal sex ratio; an inverse-square dispersal-and-mating kernel with the mate-finding cost-distance matrix set equal to the dispersal cost-distance matrix; Poisson fecundity (mean 3); and genotypes of 10 unlinked loci with 5 starting alleles and no mutation. Only two inputs vary across the 150 combinations: the cost-distance matrix (which encodes area, clumping and linkage) and the dispersal threshold.

Table S1. Factors and levels of the analysed factorial design.

Factor	Levels	No.
Habitat area	`10, 20, 30, 40, 50 %`	5
Clumping (σ)	`3.01, 4.00, 5.00, 6.00, 7.00`	5
Connectivity	`base, linked (area-conserved)`	2
Dispersal threshold	`50, 100, 150 cost units`	3
Replicates	`independent seeds`	10
Total simulations	**5 × 5 × 2 × 3 × 10**	**1,500**

S2. Reproducibility note: archived template vs. parameters as run

The pipeline scripts in S4 are the scripts used to generate the landscapes, sampling, cost distances and CDPOP inputs (shared with the companion paper) and to perform the landscape-genetic analysis (this paper). Two points of provenance are stated explicitly, in the interest of exact reproducibility.

(1) Parameters as run. The archived CDPOP-input template swept five dispersal thresholds (50/75/100/125/150 cost units) and carried placeholder values for the number of generations and for fecundity. The 1,500 simulations analysed here used the three dispersal thresholds 50/100/150 cost units, 200 non-overlapping generations, and Poisson fecundity with mean 3, as stated in Methods §2.3 and encoded in scenario_lookup.csv. The version of make_cdpop_inputs.py reproduced in S4 has been set to these as-run values so that it regenerates the analysed 5

× 5 × 2 × 3 design; the remaining CDPOP parameters (K = 100, inverse-square kernels, 10 loci × 5 alleles, no mutation) are as in §S1.

(2) Random seeds. run_landsim.py as archived draws landscapes with np.random.rand() without setting an explicit seed, so the script alone does not reproduce a specific base-landscape realisation. Reproduction of the exact analysed landscapes therefore requires the deposited raster set (to be archived with the companion paper on acceptance); the script reproduces the generative procedure and the statistical population of landscapes rather than the specific seeds.

S3. Verified summary statistics

All statistics below were recomputed directly from the 1,500 per-run output records (mlpe_results.csv, mantel_results.csv).

Table S2. Overall landscape-genetic outcomes across the 1,500 simulations.

Quantity	Value
Simulations analysed	1500
MLPE detection power (LR p < 0.05)	96.7%
Mantel detection power (p < 0.05)	90.4%
MLPE β positive	100%
MLPE β (mean, SD, range)	0.0139 (SD 0.0055; 0.0006-0.0408)
MLPE marginal R^2 (mean, SD, range)	0.020 (SD 0.015; 0.000-0.128)
Mantel r_cost (mean, SD, range)	0.093 (SD 0.042; -0.036-0.238)
Runs with Δr > 0	52.5%
Surviving individuals per run (mean, range)	97.1 (75-100)

Table S3. Landscape-genetic outcomes by dispersal threshold (means over the 500 runs per block).

Dispersal (cu)	Power (MLPE)	β positive	Mean β	Mean R^2	Mean Δr
50	0.99	100%	0.0155	0.024	-0.004
100	0.97	100%	0.0140	0.020	+0.004
150	0.94	100%	0.0123	0.016	+0.004

Detection power and signal magnitude decline monotonically as the dispersal threshold rises, consistent with progressive (but never complete) homogenisation of gene flow. β is positive in every one of the 1,500 fits.

S4. Analysis pipeline (scripts)

Scripts are listed in execution order. The first four (Python) are the landscape-generation and simulation pipeline shared with the companion paper; the last three (R) are the landscape-genetic analysis and synthesis pipeline specific to the present paper.

S4.1 Landscape generation and simulation (companion pipeline)

`run_landsim.py`

Generates the 100 × 100 neutral-landscape factorial (base + area-conserved linked); Methods §2.2–2.3.

```
# ============================================================================
# run_landsim.py  (companion paper: Cushman, in review)
# ----------------------------------------------------------------------------
# Generates the 100 x 100 pixel neutral-landscape factorial (base and
# linked/connected landscapes) described in Methods sections 2.2-2.3 of both
# the companion paper and the present paper. For each (area, sigma) combination
# it writes a "base" GeoTIFF and an area-conserved "linked" GeoTIFF in which
# nearest-neighbour patches are joined by 3-pixel Bresenham corridors and the
# original habitat area is restored exactly by stochastic edge erosion.
#
# NOTE ON PARAMETERS AS ARCHIVED vs. AS ANALYSED (see Supplementary Note S2):
#   This archived generator sweeps a 9 x 9 (area x sigma) grid (162 GeoTIFFs).
#   The landscape-genetic analysis in the present paper uses the 5 x 5 subset
#   selected by select_xy.py: area in {10,20,30,40,50}% and sigma nearest to
#   {3.01, 4.00, 5.00, 6.00, 7.00}.
# ============================================================================

import os
import numpy as np
from scipy.ndimage import (gaussian_filter, label, binary_erosion,
                           binary_dilation, generate_binary_structure,
                           center_of_mass)
from scipy.spatial.distance import cdist
from scipy.spatial import KDTree
from skimage.draw import line
import rasterio
from rasterio.transform import from_origin


def create_geotiff(filename, data):
    """Exports a 2D numpy array as a GeoTIFF."""
    # Dummy geotransform (origin at 0,0 with 1x1 pixel size)
    transform = from_origin(0, 0, 1, 1)
    with rasterio.open(
        filename, 'w', driver='GTiff',
        height=data.shape[0], width=data.shape[1],
        count=1, dtype=data.dtype,
        crs='+proj=latlong', transform=transform,
    ) as dst:
        dst.write(data, 1)


def connect_and_balance(landscape):
    """Link patches to nearest neighbours, thicken to 3 px, and conserve area."""
    # 1. Identify distinct habitat patches (8-connectivity)
    struct = generate_binary_structure(2, 2)
    labeled, num_features = label(landscape, structure=struct)

    # If the landscape is entirely continuous or empty, return as is
    if num_features < 2:
        return landscape.copy()

    # 2. Extract inner boundaries and centroids for distance calculation
    boundaries = []
    centroids = []
    for i in range(1, num_features + 1):
        patch = (labeled == i)
        eroded = binary_erosion(patch)
        boundary = patch ^ eroded  # inner edge
```

```
        coords = np.argwhere(boundary)
        if len(coords) == 0:            # fallback for single-pixel patches
            coords = np.argwhere(patch)
        boundaries.append(coords)
        centroids.append(center_of_mass(patch))

    corridor_mask = np.zeros_like(landscape, dtype=bool)

    # KDTree to find the nearest neighbouring patch by centroid
    tree = KDTree(centroids)
    # k=2 because the closest patch to patch A is patch A itself (index 0)
    _, nearest_indices = tree.query(centroids, k=2)

    # 3. Draw shortest boundary-to-boundary paths to the nearest neighbour
    for i in range(num_features):
        nearest_idx = nearest_indices[i][1]
        coords1 = boundaries[i]
        coords2 = boundaries[nearest_idx]
        dists = cdist(coords1, coords2)
        idx1, idx2 = np.unravel_index(np.argmin(dists), dists.shape)
        p1 = coords1[idx1]
        p2 = coords2[idx2]
        rr, cc = line(p1[0], p1[1], p2[0], p2[1])  # Bresenham line
        corridor_mask[rr, cc] = True

    # 4. Thicken corridors to 3 pixels (dilate 1-px line by a 3x3 structure)
    thick_corridors = binary_dilation(
        corridor_mask, structure=generate_binary_structure(2, 2))

    # Isolate only the newly added corridor pixels (excluding existing habitat)
    new_corridors = thick_corridors & ~landscape.astype(bool)
    excess_area = new_corridors.sum()

    # 5. Restore original habitat area exactly by eroding original-patch edges
    combined = landscape.astype(bool) | new_corridors
    balanced_landscape = combined.copy()
    removable = landscape.astype(bool).copy()

    while excess_area > 0 and removable.sum() > 0:
        eroded = binary_erosion(removable)
        edges = removable ^ eroded
        edge_coords = np.argwhere(edges)
        if len(edge_coords) == 0:
            edge_coords = np.argwhere(removable)
        np.random.shuffle(edge_coords)          # remove random edge pixels
        to_remove_count = min(excess_area, len(edge_coords))
        for k in range(to_remove_count):
            r, c = edge_coords[k]
            balanced_landscape[r, c] = False
            removable[r, c] = False
        excess_area -= to_remove_count

    return balanced_landscape.astype(np.uint8)



def generate_landscape_factorial():
    grid_size = 100
    levels = 9
    areas = np.linspace(0.1, 0.9, levels)
    sigmas = np.linspace(0.01, 8.0, levels)

    output_dir = "landscape_geotiffs"
    os.makedirs(output_dir, exist_ok=True)
    print("Generating and exporting 162 GeoTIFFs (81 Base + 81 Linked)...")

    file_count = 0
    for i, area in enumerate(areas):
        for j, sigma in enumerate(sigmas):
            # --- Base landscape ---
            noise = np.random.rand(grid_size, grid_size)
            smoothed_noise = gaussian_filter(noise, sigma=sigma)
            percentile_threshold = (1 - area) * 100
            threshold_value = np.percentile(smoothed_noise, percentile_threshold)
            base_landscape = (smoothed_noise >= threshold_value).astype(np.uint8)
```

```
            # --- Linked & area-conserved landscape ---
            linked_landscape = connect_and_balance(base_landscape)

            # --- Naming & export ---
            area_pct = int(area * 100)
            base_filename = os.path.join(
                output_dir, f"base_area_{area_pct}_clump_{sigma:.2f}.tif")
            link_filename = os.path.join(
                output_dir, f"linked_area_{area_pct}_clump_{sigma:.2f}.tif")
            create_geotiff(base_filename, base_landscape)
            create_geotiff(link_filename, linked_landscape)
            file_count += 2
            if file_count % 20 == 0:
                print(f"Exported {file_count}/162 files...")

    print("Process complete. All 162 GeoTIFFs have been saved.")


if __name__ == "__main__":
    generate_landscape_factorial()
```

select_xy.py

Draws the fixed 100 within-habitat sample points per landscape; Methods §2.3–2.4.

```
# ============================================================================
# select_xy.py  (companion paper: Cushman, in review)
# ----------------------------------------------------------------------------
# Draws the fixed set of 100 within-habitat sample points per landscape used
# for CDPOP initialisation and for the pairwise distance analyses (Methods 2.3-
# 2.4). Sampling is restricted to the 5 x 5 (area x sigma) subset analysed here:
# area in {10,20,30,40,50}% and sigma nearest {3.01, 4.00, 5.00, 6.00, 7.00}.
# Point identifiers are preserved so that base and linked versions of a given
# landscape share the same 100 sample locations.
# ============================================================================

import os
import glob
import pandas as pd
import numpy as np
import rasterio



def sample_habitat_to_individual_csvs():
    input_dir = "landscape_geotiffs"
    output_dir = "habitat_samples"
    os.makedirs(output_dir, exist_ok=True)

    target_areas = [10, 20, 30, 40, 50]
    target_clumps = ["3.01", "4.00", "5.00", "6.00", "7.00"]
    samples_per_file = 100

    print(f"Sampling from matching TIFFs in '{input_dir}'...")
    base_files = glob.glob(os.path.join(input_dir, "base_area_*.tif"))
    matched_count = 0

    for file_path in base_files:
        filename = os.path.basename(file_path)
        try:
            parts = filename.split('_')
            area_val = int(parts[2])
            clump_val = parts[4].replace(".tif", "")
            if area_val in target_areas and clump_val in target_clumps:
                with rasterio.open(file_path) as src:
                    data = src.read(1)
                    transform = src.transform

                habitat_indices = np.argwhere(data == 1)
                if len(habitat_indices) == 0:
                    print(f"Skipping {filename}: no habitat found.")
                    continue

                n_to_sample = min(samples_per_file, len(habitat_indices))
                random_indices = np.random.choice(
                    len(habitat_indices), size=n_to_sample, replace=False)
                selected_coords = habitat_indices[random_indices]

                file_points = []
                for i, (row, col) in enumerate(selected_coords, start=1):
                    x, y = transform * (col, row)
                    file_points.append({'ID': i, 'X': x, 'Y': y})

                df = pd.DataFrame(file_points)
                csv_name = filename.replace(".tif", "_samples.csv")
                df.to_csv(os.path.join(output_dir, csv_name), index=False)
                matched_count += 1
                print(f"Generated: {csv_name}")
        except Exception as e:
            print(f"Error processing {filename}: {e}")
            continue

    print(f"\nDone. Created {matched_count} sample CSV files in '{output_dir}'.")
```

```
if __name__ == "__main__":
    sample_habitat_to_individual_csvs()
```

make_costdistance.py

Computes aligned base/linked least-cost-distance matrices (skimage MCP_Geometric); Methods §2.3.

```
# ============================================================================
# make_costdistance.py  (companion paper: Cushman, in review)
# ----------------------------------------------------------------------------
# Computes the pairwise least-cost-distance matrices (base and linked) supplied
# to CDPOP as both the dispersal and mate-finding cost-distance matrices
# (Methods 2.3). Habitat pixels are assigned a resistance of 1 and non-habitat
# a resistance of 10; least-cost paths use the geometric minimum-cost-path
# algorithm (skimage.graph.MCP_Geometric). Base and linked matrices for the same
# landscape share row/column order (the CSV sample-point IDs), so downstream
# survivor sub-setting stays aligned.
# ============================================================================

import os
import glob
import numpy as np
import pandas as pd
import rasterio
from skimage.graph import MCP_Geometric



def compute_synchronized_cost_matrices():
    sample_dir = "habitat_samples"
    tif_dir = "landscape_geotiffs"
    output_dir = "cost_matrices"
    os.makedirs(output_dir, exist_ok=True)

    sample_files = glob.glob(os.path.join(sample_dir, "*_samples.csv"))
    if not sample_files:
        print(f"No sample CSVs found in '{sample_dir}'.")
        return

    print(f"Generating matrices for {len(sample_files)} sample sets...")
    for csv_path in sample_files:
        points_df = pd.read_csv(csv_path)
        point_ids = points_df['ID'].tolist()
        csv_name = os.path.basename(csv_path)

        base_tif_name = csv_name.replace("_samples.csv", ".tif")
        linked_tif_name = base_tif_name.replace("base_", "linked_")

        for tif_path, label in [
                (os.path.join(tif_dir, base_tif_name), "base"),
                (os.path.join(tif_dir, linked_tif_name), "linked")]:
            if not os.path.exists(tif_path):
                continue
            with rasterio.open(tif_path) as src:
                data = src.read(1)
                # Cost: habitat (1) = 1, non-habitat (0) = 10
                cost_surface = np.where(data == 1, 1.0, 10.0).astype(np.float32)
                # Map X,Y to pixel indices in the EXACT order of the CSV IDs
                pixel_coords = [src.index(x, y)
                                for x, y in zip(points_df['X'], points_df['Y'])]

            n_points = len(pixel_coords)
            cost_matrix = np.zeros((n_points, n_points))
            mcp = MCP_Geometric(cost_surface)
            for i in range(n_points):
                cum_costs, _ = mcp.find_costs(starts=[pixel_coords[i]])
                for j in range(n_points):
                    r_target, c_target = pixel_coords[j]
                    cost_matrix[i, j] = cum_costs[r_target, c_target]

            matrix_df = pd.DataFrame(cost_matrix, index=point_ids,
                                     columns=point_ids)
            matrix_filename = csv_name.replace(
                "_samples.csv", f"_{label}_cost_matrix.csv")
            matrix_df.to_csv(os.path.join(output_dir, matrix_filename))
            print(f"Exported: {matrix_filename}")

    print("\nAll matrices matched and exported.")
```

```
if __name__ == "__main__":
    compute_synchronized_cost_matrices()
```

make_cdpop_inputs.py

Builds CDPOP individual-XY and inputvariables files per dispersal threshold; Methods §2.3 (parameters as run, see S2).

```
# ============================================================================
# make_cdpop_inputs.py  (companion paper: Cushman, in review)
# ----------------------------------------------------------------------------
# Builds the CDPOP individual-XY and inputvariables files for each dispersal-
# threshold scenario (Methods 2.3). Each landscape/linkage combination is paired
# with its aligned cost-distance matrix and run at each dispersal threshold.
#
# PARAMETERS AS RUN (see Supplementary Note S2). The values below reproduce the
# parameterisation of the 1,500 analysed simulations reported in the present
# paper and the companion paper:
#     * dispersal thresholds : 50, 100, 150 cost units   (3 levels)
#     * generations          : 200 non-overlapping
#     * Monte-Carlo reps      : 10 per combination
#     * mating / movement    : inverse-square kernel, mate matrix = move matrix
#     * fecundity            : Poisson, mean 3
#     * genetics             : 10 unlinked loci, 5 starting alleles, no mutation
#     * carrying capacity K  : 100 (density-dependent exponential model)
# The archived template from which this script derives swept five dispersal
# thresholds (50/75/100/125/150) and carried placeholder values for generations
# and fecundity; those placeholders are corrected here to the values actually
# used for the analysed runs, so that this script regenerates the analysed
# 5 x 5 x 2 x 3 (area x clump x linkage x dispersal) design.
# ============================================================================

import os
import glob
import pandas as pd
import numpy as np



def generate_cdpop_inputs():
    sample_dir = "habitat_samples"
    matrix_dir = "cost_matrices"
    out_dir = "CDPOP_Inputs"
    os.makedirs(out_dir, exist_ok=True)

    distances = [50, 100, 150]   # dispersal thresholds actually analysed

    matrices = glob.glob(os.path.join(matrix_dir, "*_cost_matrix.csv"))
    for m_path in matrices:
        m_name = os.path.basename(m_path)
        # Match back to the XY sample file, e.g.
        # base_area_10_clump_3.01_base_cost_matrix.csv -> base_area_10_clump_3.01_samples.csv
        prefix = (m_name.split("_cost_matrix")[0]
                  .replace("_base", "").replace("_linked", ""))
        xy_file = os.path.join(sample_dir, f"{prefix}_samples.csv")
        if not os.path.exists(xy_file):
            continue

        # 1. Individual XY file (genetics initialised inside CDPOP)
        df = pd.read_csv(xy_file)
        df_cdp = pd.DataFrame({
            'ID': df['ID'], 'X': df['X'], 'Y': df['Y'],
            'Status': 1, 'Age': 0,
            'Sex': np.random.randint(0, 2, len(df)),
            'Subpop': 1, 'L0A0': 1, 'L0A1': 2,   # placeholder genetics
        })
        run_name = m_name.replace("_cost_matrix.csv", "")
        run_path = os.path.join(out_dir, run_name)
        os.makedirs(run_path, exist_ok=True)
        df_cdp.to_csv(os.path.join(run_path, "ind_xy.csv"), index=False)

        # 2. inputvariables.csv for each dispersal threshold
        for d in distances:
            params = [
                ['mcruns', '10'],
                ['n_generations', '200'],
                ['fecundity', '1,3'],        # 1 = Poisson; mean = 3
                ['nonoverlapping', '1'],
                ['mate_func', '2'],          # 2 = inverse square
                ['mate_dist', str(d)],
                ['move_func', '2'],          # 2 = inverse square
```

```
                ['move_dist', str(d)],
                ['xyfilename', 'ind_xy.csv'],
                ['matrix_filename', m_name],
            ]
            pd.DataFrame(params).to_csv(
                os.path.join(run_path, f"inputvars_d{d}.csv"),
                index=False, header=False)

    print(f"CDPOP project structure ready in {out_dir}")


if __name__ == "__main__":
    generate_cdpop_inputs()
```

S4.2 Landscape-genetic analysis and synthesis (this paper)

cdpop_distances.R

Builds aligned per-run Euclidean and Bray–Curtis matrices on extant individuals; Methods §2.4.

```
# ============================================================================
#  CDPOP simulation analysis: aligned Euclidean and Bray-Curtis matrices
# ============================================================================
#
#  Layout
#  ------
#  data/
#    xyfiles/
#       formatted_base_area_<A>_clump_<C>_<base|linked>_ind_xy.csv      # 50 files
#    output_100ind/
#       batchrun<B>mcrun<M>/grid1.csv                                   # ~1500 dirs
#                                                                       # (150 batchruns x 10 mcruns)
#    inputvars_area_frag_corr.csv                                       # row N -> batchrunN scenario
#
#  Why per-mcrun pairing
#  ---------------------
#  Each grid1.csv has a different number of OPEN cells (varies by mcrun).
#  Bray-Curtis must therefore be computed only on EXTANT individuals, and
#  the corresponding Euclidean matrix must be subset to those same individuals
#  IN THE SAME ROW ORDER so distance pairs line up element-by-element
#  (essential for Mantel tests, partial Mantel, regression, etc.).
#
#  Joining xy to grid1
#  -------------------
#  IDs differ between files: xy has "initial0..initial99", grid1 has offspring
#  IDs like "T0M72F81Pop1". But (XCOORD, YCOORD) is unique in each xy file
#  (verified: 100 unique pairs out of 100 rows), and grid1 reports the
#  occupied cell coordinates, so xy is joined to extant grid1 rows by
#  (XCOORD, YCOORD).
#
#  Output (per batchrun#mcrun#)
#  ----------------------------
#  distance_results/
#      euclid/euclid_batchrun<B>_mcrun<M>.csv     # n_extant x n_extant
#      bray/bray_batchrun<B>_mcrun<M>.csv         # n_extant x n_extant, same order
#      scenario_lookup.csv                        # batchrun -> scenario crosswalk
#      coverage.csv                               # n_extant per mcrun, parsing flags
# ============================================================================



# ---- 0. Packages -------------------------------------------------------------
# install.packages("vegan")
suppressPackageStartupMessages(library(vegan))



# ---- 1. Paths ----------------------------------------------------------------
base_dir        <- "C:/papers/new_area_frag_corridor_sim/CDPOP-master/data"
xy_dir          <- file.path(base_dir, "xyfiles")
output_root     <- file.path(base_dir, "output_100ind")
inputvars_path  <- file.path(base_dir, "inputvars_area_frag_corr.csv")

results_dir     <- "C:/papers/new_area_frag_corridor_sim/working2"
euclid_out_dir  <- file.path(results_dir, "euclid")
bray_out_dir    <- file.path(results_dir, "bray")
dir.create(euclid_out_dir, showWarnings = FALSE, recursive = TRUE)
dir.create(bray_out_dir,   showWarnings = FALSE, recursive = TRUE)



# ---- 2. Build batchrun -> scenario lookup from inputvars ---------------------
# inputvars row 1 of data (line 2) -> batchrun0, row 2 -> batchrun1, etc.

iv <- read.csv(inputvars_path, stringsAsFactors = FALSE)

xy_basename <- sub("^xyfiles/", "", iv$xyfilename)
xy_basename <- sub("\\.csv$",   "", xy_basename)

scen_regex <- "formatted_base_area_(\\d+)_clump_([0-9.]+)_(base|linked)_ind_xy"
```

```
mlist <- regmatches(xy_basename, regexec(scen_regex, xy_basename))
m_mat <- do.call(rbind, lapply(mlist, function(x) if (length(x) == 4) x[2:4] else c(NA,NA,NA)))

scenario_lookup <- data.frame(
  batchrun     = seq_len(nrow(iv)) - 1L,               # 0-indexed
  xy_file_stem = xy_basename,
  area         = m_mat[, 1],
  clump        = m_mat[, 2],
  type         = m_mat[, 3],
  scenario_key = sprintf("area%s_clump%s_%s", m_mat[,1], m_mat[,2], m_mat[,3]),
  stringsAsFactors = FALSE
)
write.csv(scenario_lookup,
          file.path(results_dir, "scenario_lookup.csv"),
          row.names = FALSE)
cat("Scenario lookup built for", nrow(scenario_lookup), "batchruns.\n")



# ---- 3. Cache xy data to avoid re-reading the same xy file 10x per mcrun ---
xy_cache <- new.env(parent = emptyenv())
load_xy <- function(stem) {
  if (exists(stem, envir = xy_cache)) return(get(stem, envir = xy_cache))
  path <- file.path(xy_dir, paste0(stem, ".csv"))
  if (!file.exists(path)) stop("xy file not found: ", path)
  d <- read.csv(path, stringsAsFactors = FALSE)
  # Sanity: each (XCOORD, YCOORD) must be unique to allow safe coord-based join
  key <- paste(d$XCOORD, d$YCOORD, sep = "_")
  if (anyDuplicated(key))
    stop("Duplicate (XCOORD,YCOORD) in ", basename(path),
         " -- coord-based join is unsafe.")
  assign(stem, d, envir = xy_cache)
  d
}



# ---- 4. Helpers -----------------------------------------------------------
read_grid1 <- function(grid_path) {
  # CDPOP grid files have a trailing comma on every data row but NOT on the
  # header. That gives header = 59 fields, data = 60 fields. Two read.csv
  # behaviours that arise without pre-processing:
  #   - default: tries to use first column as row names -> fails on
  #     duplicate Subpopulation values
  #   - row.names = NULL: keeps all 60 cols but inserts a phantom first
  #     column called "row.names" and shifts every other column over by 1
  # Fix: read the file as raw text, strip trailing commas, then parse.
  txt <- readLines(grid_path)
  txt <- sub(",\\s*$", "", txt)
  df <- read.csv(textConnection(txt), stringsAsFactors = FALSE,
                 na.strings = c("NA", "OPEN", ""),
                 check.names = FALSE)
  # Belt-and-braces: drop any all-NA columns just in case
  all_na <- vapply(df, function(x) all(is.na(x)), logical(1))
  if (any(all_na)) df <- df[, !all_na, drop = FALSE]
  df
}



# ---- 5. Main loop: per batchrun#mcrun# folder, build aligned matrices -------
run_dirs <- list.dirs(output_root, recursive = FALSE, full.names = TRUE)
run_dirs <- run_dirs[grepl("batchrun\\d+mcrun\\d+$", basename(run_dirs))]
cat("\nbatchrun#mcrun# folders found:", length(run_dirs), "\n")

run_regex <- "batchrun(\\d+)mcrun(\\d+)"

coverage <- data.frame(
  batchrun  = integer(0),
  mcrun     = integer(0),
  scenario  = character(0),
  n_grid    = integer(0),
  n_extant  = integer(0),
  n_matched = integer(0),
  status    = character(0),
  stringsAsFactors = FALSE
)

n_done <- 0L; n_skip <- 0L
```

```
for (rd in run_dirs) {
  m <- regmatches(basename(rd), regexec(run_regex, basename(rd)))[[1]]
  batchrun <- as.integer(m[2]); mcrun <- as.integer(m[3])

  scen_row <- scenario_lookup[scenario_lookup$batchrun == batchrun, ]
  if (nrow(scen_row) == 0) {
    warning("No inputvars row for batchrun", batchrun); n_skip <- n_skip + 1L; next
  }
  scen_key <- scen_row$scenario_key

  grid_path <- file.path(rd, "grid1.csv")
  if (!file.exists(grid_path)) {
    warning("No grid1.csv in ", rd); n_skip <- n_skip + 1L; next
  }

  # ---- read grid1, drop OPEN ----
  grid <- tryCatch(read_grid1(grid_path),
                   error = function(e) { warning(rd, ": ", e$message); NULL })
  if (is.null(grid)) { n_skip <- n_skip + 1L; next }

  n_grid <- nrow(grid)
  grid_alive <- grid[!is.na(grid$ID), , drop = FALSE]
  n_extant <- nrow(grid_alive)

  if (n_extant < 2) {
    coverage <- rbind(coverage,
      data.frame(batchrun, mcrun, scenario = scen_key,
                 n_grid, n_extant, n_matched = 0L,
                 status = "too_few_extant", stringsAsFactors = FALSE))
    n_skip <- n_skip + 1L; next
  }

  # ---- load matching xy for this scenario ----
  xy <- tryCatch(load_xy(scen_row$xy_file_stem),
                 error = function(e) { warning(e$message); NULL })
  if (is.null(xy)) { n_skip <- n_skip + 1L; next }

  # ---- coord-based join: for each extant individual, look up its xy row ----
  xy_key   <- paste(xy$XCOORD,         xy$YCOORD,         sep = "_")
  grid_key <- paste(grid_alive$XCOORD, grid_alive$YCOORD, sep = "_")
  idx <- match(grid_key, xy_key)

  unmatched <- sum(is.na(idx))
  if (unmatched > 0) {
    warning(sprintf("%s: %d/%d extant individuals could not be matched to xy by coord",
                    basename(rd), unmatched, n_extant))
    keep <- !is.na(idx)
    grid_alive <- grid_alive[keep, , drop = FALSE]
    idx        <- idx[keep]
  }
  n_matched <- nrow(grid_alive)
  if (n_matched < 2) {
    coverage <- rbind(coverage,
      data.frame(batchrun, mcrun, scenario = scen_key,
                 n_grid, n_extant, n_matched,
                 status = "too_few_matched", stringsAsFactors = FALSE))
    n_skip <- n_skip + 1L; next
  }

  # ---- unique row labels ------------------------------------------------
  # In CDPOP, IDs at gen 1 encode parentage (e.g. "T0M72F81Pop1"), so siblings
  # share an ID. Make make.unique-style suffixes so dist/vegdist row.names work.
  row_ids <- make.unique(as.character(grid_alive$ID), sep = "_")

  # ---- build aligned coordinate matrix (xy rows in same order as grid_alive)
  coords <- xy[idx, c("XCOORD", "YCOORD"), drop = FALSE]
  rownames(coords) <- row_ids

  # ---- build allele matrix from grid_alive (rows already in target order) --
  allele_cols <- grep("^L\\d+A\\d+$", names(grid_alive), value = TRUE)
  if (!length(allele_cols)) {
    warning(rd, ": no L#A# allele cols"); n_skip <- n_skip + 1L; next
  }
  alleles <- as.matrix(grid_alive[, allele_cols, drop = FALSE])
```

```
  storage.mode(alleles) <- "numeric"
  rownames(alleles) <- row_ids

  # Sanity: 10 diploid loci -> rowSums == 20. Warn (non-fatal) if off.
  rs <- rowSums(alleles, na.rm = TRUE)
  if (length(unique(rs)) > 1)
    warning(rd, ": uneven allele row sums (range ",
            min(rs), "-", max(rs), ") -- check parsing")

  # ---- distances ----
  d_eu <- dist(coords, method = "euclidean")
  d_bc <- vegdist(alleles, method = "bray")

  tag      <- sprintf("batchrun%d_mcrun%d", batchrun, mcrun)
  write.csv(as.matrix(d_eu),
            file.path(euclid_out_dir, paste0("euclid_", tag, ".csv")))
  write.csv(as.matrix(d_bc),
            file.path(bray_out_dir,   paste0("bray_",   tag, ".csv")))

  coverage <- rbind(coverage,
    data.frame(batchrun, mcrun, scenario = scen_key,
               n_grid, n_extant, n_matched,
               status = "ok", stringsAsFactors = FALSE))
  n_done <- n_done + 1L
}



# ---- 6. Write coverage report and finish ---------------------------------
write.csv(coverage,
          file.path(results_dir, "coverage.csv"),
          row.names = FALSE)

cat("\nDone.\n",
    "  ok:    ", n_done, "\n",
    "  skip:  ", n_skip, "\n",
    "  euclid:", euclid_out_dir, "\n",
    "  bray:  ", bray_out_dir, "\n",
    "  cover: ", file.path(results_dir, "coverage.csv"), "\n\n",
    "Each batchrun<B>_mcrun<M> now has matched-order pairs:\n",
    "  euclid_batchrun<B>_mcrun<M>.csv  &  bray_batchrun<B>_mcrun<M>.csv\n",
    "Both are n_matched x n_matched with rownames/colnames = grid1 IDs in\n",
    "the same order, so they're ready for mantel(d_eu, d_bc) etc.\n")
```

cdpop_stage1ResistGA_v3_4factor.r

Stage 1: per-run Mantel and MLPE fits over the four-factor design; Methods §2.5–2.6.

```
# ============================================================================
# CDPOP Stage 1: Mantel + MLPE  (v3, four-factor design)
# ============================================================================
# Changes vs. v2:
#   * The factorial design is 5 (area) x 5 (clump) x 2 (linkage) x 3 (dispersal),
#     replicated 10 times -> 1,500 simulations.
#   * batchrun 0-149 maps deterministically to (area, clump, linkage, dispersal):
#       - dispersal block: 0-49 -> 50 cu;  50-99 -> 100 cu;  100-149 -> 150 cu
#       - within a block of 50: linkage cycles fastest, then clump, then area
#   * scenario_lookup now includes dispersal, area, clump, linkage so that the
#     downstream analysis script never has to recompute it.
#   * scenario_lookup is also written to disk for use by stage 2.
#   * MLPE/Mantel computation itself is unchanged from v2.
# ============================================================================

suppressPackageStartupMessages({
  library(vegan)           # mantel()
  library(ResistanceGA)    # MLPE.lmm()
  library(MuMIn)           # r.squaredGLMM()
  library(lme4)            # update() / anova() on lmerMod under the hood
})

# ---- 1. Configuration -------------------------------------------------------

bray_in_dir    <- "C:/papers/new_area_frag_corridor_sim/working2/bray"
euclid_in_dir  <- "C:/papers/new_area_frag_corridor_sim/working2/euclid"
cdmats_dir     <- "C:/papers/new_area_frag_corridor_sim/CDPOP-master/data/cdmats"
xy_dir         <- "C:/papers/new_area_frag_corridor_sim/CDPOP-master/data/xyfiles"
output_root    <- "C:/papers/new_area_frag_corridor_sim/CDPOP-master/data/output_100ind"
inputvars_path <- "C:/papers/new_area_frag_corridor_sim/CDPOP-master/data/inputvars_area_frag_corr.csv"


results_dir    <- "C:/papers/new_area_frag_corridor_sim/working2"
dir.create(results_dir, showWarnings = FALSE, recursive = TRUE)


# ---- 2. Scenario lookup (4-factor) ------------------------------------------
# Two ways to derive the scenario for each batchrun:
#   (A) read inputvars and parse xyfilename + the dispersal-block index, OR
#   (B) reconstruct from the known design.
# Option (A) is used so the script remains correct if inputvars is regenerated;
# (B) is cross-checked for safety and the script aborts on mismatch.

if (!file.exists(inputvars_path)) stop("Cannot find inputvars file: ", inputvars_path)


inputvars   <- read.csv(inputvars_path, stringsAsFactors = FALSE)
N_BATCH     <- nrow(inputvars)
if (N_BATCH != 150) {
  warning(sprintf("inputvars has %d rows; expected 150 for the 5x5x2x3 design.", N_BATCH))
}

# Strip the xyfiles/ prefix and .csv extension to get the true stem.
xy_basename <- sub("^xyfiles/", "", inputvars$xyfilename)
xy_basename <- sub("\\.csv$",   "", xy_basename)

# Parse area, clump and linkage from the xy filename
# Expected pattern: formatted_base_area_<A>_clump_<C>_(base|linked)_ind_xy
parsed <- regmatches(xy_basename,
                     regexec("area_(\\d+)_clump_([0-9.]+)_(base|linked)", xy_basename))
parse_ok <- vapply(parsed, function(m) length(m) == 4, logical(1))
if (!all(parse_ok)) {
  bad <- which(!parse_ok)
  stop(sprintf("Could not parse area/clump/linkage from %d xyfilenames; e.g. row %d: %s",
               length(bad), bad[1], xy_basename[bad[1]]))
}
area_vec  <- as.integer (vapply(parsed, function(m) m[2], character(1)))
clump_vec <- as.numeric (vapply(parsed, function(m) m[3], character(1)))
type_vec  <-             vapply(parsed, function(m) m[4], character(1))

# Dispersal: companion design states first 50 inputvars rows = dispersal 50,
# next 50 = dispersal 100, last 50 = dispersal 150.
disp_vec  <- rep(c(50, 100, 150), each = 50)[seq_len(N_BATCH)]
```

```
scenario_lookup <- data.frame(
  batchrun     = seq_len(N_BATCH) - 1L,
  xy_file_stem = xy_basename,
  area         = area_vec,
  clump        = clump_vec,
  type         = type_vec,
  dispersal    = disp_vec,
  stringsAsFactors = FALSE
)

# Sanity check (B): the design should be a complete 5x5x2x3 factorial.
chk <- table(scenario_lookup$area, scenario_lookup$clump,
             scenario_lookup$type, scenario_lookup$dispersal)
if (!all(chk == 1)) {
  stop("Design is not a complete 5x5x2x3 factorial. Check inputvars ordering.")
}
cat("Design verified: 5 (area) x 5 (clump) x 2 (linkage) x 3 (dispersal) factorial.\n")

write.csv(scenario_lookup,
          file.path(results_dir, "scenario_lookup.csv"),
          row.names = FALSE)
cat("Wrote scenario_lookup.csv with", nrow(scenario_lookup), "rows.\n")

# ---- 3. Helpers (unchanged from v2) ----------------------------------------

make_safe_matrix <- function(path) {
  df  <- read.csv(path, header = TRUE, row.names = 1, check.names = FALSE,
                  stringsAsFactors = FALSE)
  mat <- as.matrix(df)
  storage.mode(mat) <- "numeric"
  dimnames(mat) <- NULL
  mat
}

read_grid1 <- function(grid_path) {
  txt <- readLines(grid_path)
  txt <- sub(",\\s*$", "", txt)
  df  <- read.csv(textConnection(txt), stringsAsFactors = FALSE,
                  na.strings = c("NA", "OPEN", ""), check.names = FALSE)
  all_na <- vapply(df, function(x) all(is.na(x)), logical(1))
  if (any(all_na)) df <- df[, !all_na, drop = FALSE]
  df
}

iv_xystem_to_costfull <- function(xy_stem) {
  cost_stem <- sub("^formatted_", "", xy_stem)
  cost_stem <- sub("_ind_xy$",    "", cost_stem)
  paste0(cost_stem, "_cost_matrix")
}

find_master_indices <- function(survivor_x, survivor_y, master_x, master_y) {
  vapply(seq_along(survivor_x), function(i) {
    d   <- sqrt((master_x - survivor_x[i])^2 + (master_y - survivor_y[i])^2)
    idx <- which.min(d)
    if (length(idx) == 0L || d[idx] > 0.5) NA_integer_ else as.integer(idx)
  }, integer(1))
}

mlpe_r2 <- function(mlpe_model) {
  ml_fit <- tryCatch(update(mlpe_model, REML = FALSE),
                     error = function(e) mlpe_model)
  r2 <- tryCatch(suppressWarnings(MuMIn::r.squaredGLMM(ml_fit)),
                 error = function(e) NULL)
  if (is.null(r2)) return(c(R2m = NA_real_, R2c = NA_real_))
  r2v <- drop(r2)
  c(R2m = unname(r2v["R2m"]), R2c = unname(r2v["R2c"]))
}

# ---- 4. Analyse one batchrun#mcrun# folder (unchanged from v2) --------------

analyze_one <- function(rd) {

  folder_name <- basename(rd)
  m <- regmatches(folder_name, regexec("batchrun(\\d+)mcrun(\\d+)", folder_name))[[1]]
```

```
if (length(m) < 3) {
  message(folder_name, ": cannot parse batchrun/mcrun"); return(NULL)
}
batchrun <- as.integer(m[2]); mcrun <- as.integer(m[3])

scen_row <- scenario_lookup[scenario_lookup$batchrun == batchrun, ]
if (nrow(scen_row) == 0) {
  message(folder_name, ": no inputvars row for batchrun ", batchrun); return(NULL)
}

euclid_p <- file.path(euclid_in_dir, sprintf("euclid_batchrun%d_mcrun%d.csv", batchrun, mcrun))
bray_p   <- file.path(bray_in_dir,   sprintf("bray_batchrun%d_mcrun%d.csv",   batchrun, mcrun))
if (!file.exists(euclid_p)) { message(folder_name, ": missing ", euclid_p); return(NULL) }
if (!file.exists(bray_p))   { message(folder_name, ": missing ", bray_p);   return(NULL) }

d_eu <- make_safe_matrix(euclid_p)
d_bc <- make_safe_matrix(bray_p)
if (!identical(dim(d_eu), dim(d_bc))) {
  message(folder_name, ": eu/bc dim mismatch ",
          paste(dim(d_eu), collapse = "x"), " vs ",
          paste(dim(d_bc), collapse = "x")); return(NULL)
}
n_survivors <- nrow(d_bc)
if (n_survivors < 3) {
  message(folder_name, ": only ", n_survivors, " survivors -- skipping"); return(NULL)
}

grid_path <- file.path(rd, "grid1.csv")
if (!file.exists(grid_path)) { message(folder_name, ": no grid1.csv"); return(NULL) }
grid <- tryCatch(read_grid1(grid_path),
                 error = function(e) { message(folder_name, ": grid read failed -- ", e$message); NULL })
if (is.null(grid)) return(NULL)

grid_alive <- grid[!is.na(grid$ID) & grid$ID != "-9999", , drop = FALSE]
if (nrow(grid_alive) != n_survivors) {
  message(folder_name, ": grid_alive (", nrow(grid_alive),
          ") != n_survivors (", n_survivors, ") -- ordering mismatch"); return(NULL)
}
gx <- as.numeric(grid_alive$XCOORD)
gy <- as.numeric(grid_alive$YCOORD)

xy_master_path <- file.path(xy_dir, paste0(scen_row$xy_file_stem, ".csv"))
if (!file.exists(xy_master_path)) {
  message(folder_name, ": missing master xy ", xy_master_path); return(NULL)
}
xy_master <- read.csv(xy_master_path, stringsAsFactors = FALSE)
idx <- find_master_indices(gx, gy, as.numeric(xy_master$XCOORD), as.numeric(xy_master$YCOORD))
if (any(is.na(idx))) {
  message(folder_name, ": ", sum(is.na(idx)), " survivors had no master xy match"); return(NULL)
}

cost_stem <- iv_xystem_to_costfull(scen_row$xy_file_stem)
cost_path <- file.path(cdmats_dir, paste0(cost_stem, ".csv"))
if (!file.exists(cost_path))
  cost_path <- file.path(cdmats_dir, paste0("stripped_", cost_stem, ".csv"))
if (!file.exists(cost_path)) {
  message(folder_name, ": no cost matrix at ", cost_path); return(NULL)
}
cost_full  <- make_safe_matrix(cost_path)
if (max(idx) > nrow(cost_full)) {
  message(folder_name, ": idx (max ", max(idx),
          ") exceeds cost matrix (", nrow(cost_full), " rows)"); return(NULL)
}
d_cost <- cost_full[idx, idx]

m_cost <- vegan::mantel(as.dist(d_bc), as.dist(d_cost), permutations = 999)
m_eucl <- vegan::mantel(as.dist(d_bc), as.dist(d_eu),   permutations = 999)

bc_vec <- as.numeric(as.dist(d_bc))
ct_vec <- as.numeric(as.dist(d_cost))

mlpe_model <- tryCatch(
  ResistanceGA::MLPE.lmm(resistance       = ct_vec,
                         pairwise.genetic = bc_vec,
                         REML             = TRUE),
  error = function(e) {
```

```
      message(folder_name, ": MLPE failed -- ", e$message); NULL
    }
  )
  if (is.null(mlpe_model)) return(NULL)

  r2_vals        <- mlpe_r2(mlpe_model)
  r2_marginal    <- r2_vals[["R2m"]]
  r2_conditional <- r2_vals[["R2c"]]

  lr_p <- tryCatch({
    full_ml <- update(mlpe_model, REML = FALSE)
    null_ml <- update(full_ml, . ~ 1)
    a <- suppressWarnings(anova(null_ml, full_ml))
    pcol <- intersect(c("Pr(>Chisq)", "Pr(>F)"), colnames(a))
    if (length(pcol)) a[2, pcol[1]] else NA_real_
  }, error = function(e) {
    message(folder_name, ": LR test failed -- ", e$message); NA_real_
  })

  beta_cost <- tryCatch(summary(mlpe_model)$coefficients["resistance", "Estimate"],
                        error = function(e) NA_real_)
  se_cost   <- tryCatch(summary(mlpe_model)$coefficients["resistance", "Std. Error"],
                        error = function(e) NA_real_)
  t_cost    <- tryCatch(summary(mlpe_model)$coefficients["resistance", "t value"],
                        error = function(e) NA_real_)

  list(
    mantel = data.frame(
      batchrun       = batchrun,
      mcrun          = mcrun,
      mantel_r_cost  = m_cost$statistic,
      mantel_p_cost  = m_cost$signif,
      mantel_r_eucl  = m_eucl$statistic,
      mantel_p_eucl  = m_eucl$signif,
      n_survivors    = n_survivors,
      stringsAsFactors = FALSE
    ),
    mlpe_summary = data.frame(
      batchrun        = batchrun,
      mcrun           = mcrun,
      n_pairs         = length(bc_vec),
      AIC             = AIC(mlpe_model),
      Beta_Cost       = beta_cost,
      SE_Cost         = se_cost,
      t_Cost          = t_cost,
      R2_marginal     = r2_marginal,
      R2_conditional  = r2_conditional,
      LR_p            = lr_p,
      stringsAsFactors = FALSE
    )
  )
}

# ---- 5. Batch execution ------------------------------------------------------

run_dirs <- list.dirs(output_root, recursive = FALSE, full.names = TRUE)
run_dirs <- run_dirs[grepl("batchrun\\d+mcrun\\d+$", basename(run_dirs))]

cat(sprintf("Analysing %d simulation folders...\n", length(run_dirs)))

mantel_list <- list()
mlpe_list   <- list()

for (i in seq_along(run_dirs)) {
  res <- tryCatch(analyze_one(run_dirs[i]),
                  error = function(e) {
                    message("HARD ERROR in ", basename(run_dirs[i]), ": ", e$message); NULL
                  })
  if (!is.null(res)) {
    mantel_list[[length(mantel_list) + 1L]] <- res$mantel
    mlpe_list  [[length(mlpe_list)   + 1L]] <- res$mlpe_summary
  }
  if (i %% 20 == 0) cat(sprintf("Progress: %d/%d  (kept: %d)\n",
                                i, length(run_dirs), length(mantel_list)))
}
```

```
# ---- 6. Bind, attach metadata, write ---------------------------------------

if (length(mantel_list) == 0L) {
  stop("No folders produced results -- check the messages above for skip reasons.")
}

mantel_df <- do.call(rbind, mantel_list)
mlpe_df   <- do.call(rbind, mlpe_list)

# Attach (area, clump, type, dispersal) for the 4-factor analyses downstream
mantel_df <- merge(mantel_df, scenario_lookup[,c("batchrun","area","clump","type","dispersal")],
                   by = "batchrun", all.x = TRUE)
mlpe_df   <- merge(mlpe_df,   scenario_lookup[,c("batchrun","area","clump","type","dispersal")],
                   by = "batchrun", all.x = TRUE)

write.csv(mantel_df, file.path(results_dir, "mantel_results.csv"), row.names = FALSE)
write.csv(mlpe_df,   file.path(results_dir, "mlpe_results.csv"),   row.names = FALSE)

cat(sprintf("\nDone. %d Mantel rows, %d MLPE rows written to:\n  %s\n  %s\n",
            nrow(mantel_df), nrow(mlpe_df),
            file.path(results_dir, "mantel_results.csv"),
            file.path(results_dir, "mlpe_results.csv")))

cat("\nMLPE summary (first rows):\n")
print(utils::head(mlpe_df))
cat(sprintf("\nMedian marginal R2 across runs:    %.4f\n", median(mlpe_df$R2_marginal,    na.rm = TRUE)))
cat(sprintf("Median conditional R2 across runs: %.4f\n",   median(mlpe_df$R2_conditional, na.rm = TRUE)))
```

cdpop_stage2_4factor_analysis.r

Stage 2: four-way factorial ANOVA, effect sizes, detection-power summaries and figures; Methods §2.6.

```
# ============================================================================
# CDPOP Stage 2 (v3, four-factor): synthesis, ANOVA, figures
# ============================================================================
# Companion paper design: 5 (area) x 5 (clump) x 2 (linkage) x 3 (dispersal),
# replicated 10 times -> 1,500 simulations.
#
# Inputs (from Stage 1 v3):
#   - mantel_results.csv  (1500 rows; area, clump, type, dispersal attached)
#   - mlpe_results.csv    (1500 rows; area, clump, type, dispersal attached)
#   - scenario_lookup.csv
#   - compiled_1500_runs_Ho.csv (200 generations x 1500 simulations,
#                                from the companion CDPOP run)
#
# Outputs:
#   - anova_table_4factor_mantel.csv
#   - anova_table_4factor_mlpe.csv
#   - anova_table_4factor_R2.csv
#   - cell_summary_4factor.csv              (mean/SE per cell, 5x5x2x3 = 150 cells)
#   - manuscript_summary_stats.csv          (collapsed across dispersal for tables)
#   - logreg_detection_4factor.csv
#   - figure_R2_by_dispersal.png            (Fig 3: R2 across dispersal blocks)
#   - figure_beta_by_dispersal.png          (Fig 2)
#   - figure_mantel_delta_by_dispersal.png  (Fig 1)
#   - figure_power_heatmaps_4panel.png      (Fig 4: 3-disp x 2-link power grid)
#   - figure_dR2_vs_dHo.png                 (Fig 5: rescue vs signal-erosion)
# ============================================================================

suppressPackageStartupMessages({
  library(dplyr)
  library(tidyr)
  library(ggplot2)
  library(patchwork)
  library(scales)
})

# ---- 1. Configuration -------------------------------------------------------

working_dir <- "C:/papers/new_area_frag_corridor_sim/working2"

mantel_path  <- file.path(working_dir, "mantel_results.csv")
mlpe_path    <- file.path(working_dir, "mlpe_results.csv")
scen_path    <- file.path(working_dir, "scenario_lookup.csv")
ho_path      <- file.path(working_dir, "compiled_1500_runs_Ho.csv")
out_dir      <- working_dir
fig_dir      <- working_dir

# ---- 2. Load & merge --------------------------------------------------------

mantel_results <- read.csv(mantel_path, stringsAsFactors = FALSE)
mlpe_results   <- read.csv(mlpe_path,   stringsAsFactors = FALSE)
scenarios      <- read.csv(scen_path,   stringsAsFactors = FALSE)

# Defensive: if the input files were produced by an older Stage 1 that did NOT
# attach metadata, it is re-attached here from scenarios.
need_meta <- function(df) !all(c("area","clump","type","dispersal") %in% names(df))
if (need_meta(mantel_results)) {
  mantel_results <- merge(mantel_results,
                          scenarios[,c("batchrun","area","clump","type","dispersal")],
                          by = "batchrun", all.x = TRUE)
}
if (need_meta(mlpe_results)) {
  mlpe_results <- merge(mlpe_results,
                        scenarios[,c("batchrun","area","clump","type","dispersal")],
                        by = "batchrun", all.x = TRUE)
}

df <- mlpe_results %>%
  inner_join(mantel_results %>%
               select(batchrun, mcrun,
                      mantel_r_cost, mantel_p_cost,
                      mantel_r_eucl, mantel_p_eucl,
```

```
                    n_survivors),
            by = c("batchrun","mcrun")) %>%
  mutate(
    delta_r        = mantel_r_cost - mantel_r_eucl,
    success_mlpe   = as.integer(LR_p          < 0.05),
    success_mantel = as.integer(mantel_p_cost < 0.05),
    # Order factors so plots and contrasts are readable
    area_f      = factor(area,       levels = sort(unique(area))),
    clump_f     = factor(clump,      levels = sort(unique(clump))),
    type_f      = factor(type,       levels = c("base","linked")),
    dispersal_f = factor(dispersal,  levels = sort(unique(dispersal)))
  )

cat(sprintf("Loaded %d simulations.\n", nrow(df)))
cat("Design check (must be all 10):\n")
print(table(df$area, df$clump, df$type, df$dispersal))


# ---- 3. Four-way factorial ANOVAs -------------------------------------------
# Type-I (sequential) ANOVAs are fit on three landscape-genetic response
# variables: Mantel delta_r, MLPE Beta_Cost, and MLPE marginal R2.
# Effects are reported with partial eta^2.


partial_eta_sq <- function(aov_obj) {
  ss <- summary(aov_obj)[[1]][, "Sum Sq"]
  resid_ss <- tail(ss, 1)
  petasq <- ss / (ss + resid_ss)
  petasq[length(petasq)] <- NA   # residual row gets NA
  petasq
}

fit_4way <- function(formula, data, label) {
  m <- aov(formula, data = data)
  s <- summary(m)[[1]]
  eta <- partial_eta_sq(m)
  out <- data.frame(
    response  = label,
    term      = trimws(rownames(s)),
    df        = s[, "Df"],
    SumSq     = round(s[, "Sum Sq"],  4),
    MeanSq    = round(s[, "Mean Sq"], 4),
    F_value   = round(s[, "F value"], 3),
    p_value   = signif(s[, "Pr(>F)"], 4),
    partialEta2 = round(eta, 4),
    stringsAsFactors = FALSE
  )
  rownames(out) <- NULL
  out
}

cat("\n--- 4-way ANOVA: Mantel delta_r ---\n")
av_dr <- fit_4way(delta_r        ~ area_f*clump_f*type_f*dispersal_f, df, "delta_r")
print(av_dr, row.names = FALSE)

cat("\n--- 4-way ANOVA: MLPE Beta_Cost ---\n")
av_b  <- fit_4way(Beta_Cost      ~ area_f*clump_f*type_f*dispersal_f, df, "Beta_Cost")
print(av_b,  row.names = FALSE)

cat("\n--- 4-way ANOVA: MLPE marginal R2 ---\n")
av_r2 <- fit_4way(R2_marginal    ~ area_f*clump_f*type_f*dispersal_f, df, "R2_marginal")
print(av_r2, row.names = FALSE)

write.csv(av_dr, file.path(out_dir, "anova_table_4factor_mantel.csv"), row.names = FALSE)
write.csv(av_b,  file.path(out_dir, "anova_table_4factor_beta.csv"),   row.names = FALSE)
write.csv(av_r2, file.path(out_dir, "anova_table_4factor_R2.csv"),     row.names = FALSE)

# Combined ANOVA table for the manuscript
av_all <- bind_rows(av_dr, av_b, av_r2)
write.csv(av_all, file.path(out_dir, "anova_table_4factor_combined.csv"), row.names = FALSE)

# ---- 4. Logistic regression: drivers of detection ---------------------------

logmod <- glm(success_mlpe ~ area_f + clump_f + type_f + dispersal_f,
              data = df, family = binomial)
cat("\n--- Logistic regression (MLPE detection) ---\n")
print(summary(logmod))
log_coef <- as.data.frame(summary(logmod)$coefficients)
```

```
log_coef$term <- rownames(log_coef); rownames(log_coef) <- NULL
log_coef <- log_coef[, c("term", setdiff(names(log_coef), "term"))]
write.csv(log_coef, file.path(out_dir, "logreg_detection_4factor.csv"), row.names = FALSE)


# ---- 5. Cell summary table (5x5x2x3 = 150 cells) ----------------------------


cell_sum <- df %>%
  group_by(area, clump, type, dispersal) %>%
  summarise(
    n_reps        = n(),
    power_mlpe    = mean(success_mlpe,   na.rm = TRUE),
    power_mantel  = mean(success_mantel, na.rm = TRUE),
    mean_R2_marg  = mean(R2_marginal,    na.rm = TRUE),
    se_R2_marg    = sd(R2_marginal,      na.rm = TRUE) / sqrt(n()),
    mean_beta     = mean(Beta_Cost,      na.rm = TRUE),
    se_beta       = sd(Beta_Cost,        na.rm = TRUE) / sqrt(n()),
    mean_delta_r  = mean(delta_r,        na.rm = TRUE),
    se_delta_r    = sd(delta_r,          na.rm = TRUE) / sqrt(n()),
    .groups = "drop"
  )
write.csv(cell_sum, file.path(out_dir, "cell_summary_4factor.csv"), row.names = FALSE)


# Manuscript-table version: collapse by dispersal too (50 cells, comparable to v2)
manuscript_sum <- df %>%
  group_by(area, clump, type) %>%
  summarise(
    n_reps        = n(),
    power_mlpe    = mean(success_mlpe,   na.rm = TRUE),
    mean_R2_marg  = mean(R2_marginal,    na.rm = TRUE),
    se_R2_marg    = sd(R2_marginal,      na.rm = TRUE) / sqrt(n()),
    .groups = "drop"
  )
write.csv(manuscript_sum, file.path(out_dir, "manuscript_summary_stats.csv"), row.names = FALSE)


# ---- 6. Figures -------------------------------------------------------------


base_theme <- theme_light(base_size = 11) +
  theme(legend.position = "bottom",
        strip.background = element_rect(fill = "grey90", colour = NA),
        strip.text = element_text(colour = "grey15"))


disp_lab <- function(x) paste0("dispersal = ", x, " cu")


# Fig 1 - Mantel delta_r boxplots, faceted by clump (rows) x dispersal (cols)
fig1 <- ggplot(df, aes(x = factor(area), y = delta_r, fill = type_f)) +
  geom_hline(yintercept = 0, linetype = "dashed", colour = "grey50", linewidth = 0.4) +
  geom_boxplot(outlier.size = 0.6, linewidth = 0.3) +
  facet_grid(clump ~ dispersal, labeller = labeller(
    clump     = function(x) paste("clump =", x),
    dispersal = disp_lab)) +
  scale_fill_manual(values = c(base = "#E07B7B", linked = "#3FA9A9"), name = "Linkage") +
  labs(title = expression(paste("Mantel ", Delta, "r = ", r[cost], " - ", r[Euclidean],
                                " by area, clumping, linkage and dispersal")),
       subtitle = "Values > 0 indicate that cost-distance outperforms straight-line distance",
       x = "Habitat area (%)", y = expression(Delta~"r")) +
  base_theme


ggsave(file.path(fig_dir, "figure_mantel_delta_by_dispersal.png"),
       fig1, width = 11, height = 8, dpi = 300)


# Fig 2 - MLPE Beta lines, faceted by clump (rows) x dispersal (cols)
fig2 <- df %>%
  group_by(area, clump, type_f, dispersal) %>%
  summarise(mean_b = mean(Beta_Cost, na.rm = TRUE),
            se_b   = sd(Beta_Cost,   na.rm = TRUE) / sqrt(n()),
            .groups = "drop") %>%
  ggplot(aes(x = area, y = mean_b, colour = type_f, group = type_f)) +
  geom_errorbar(aes(ymin = mean_b - se_b, ymax = mean_b + se_b), width = 1.5) +
  geom_line(linewidth = 0.7) +
  geom_point(size = 1.6) +
  facet_grid(clump ~ dispersal, labeller = labeller(
    clump     = function(x) paste("clump =", x),
    dispersal = disp_lab)) +
  scale_colour_manual(values = c(base = "#C0392B", linked = "#3FA9A9"), name = "Linkage") +
  labs(title = "MLPE cost coefficient (Beta) across the design",
       subtitle = "Mean +/- SE over 10 replicates per cell",
       x = "Habitat area (%)", y = expression(beta[cost])) +
```

```
  base_theme

ggsave(file.path(fig_dir, "figure_beta_by_dispersal.png"),
       fig2, width = 11, height = 8, dpi = 300)

# Fig 3 - Marginal R2 lines, faceted by clump (rows) x dispersal (cols)
fig3 <- cell_sum %>%
  ggplot(aes(x = area, y = mean_R2_marg, colour = type, group = type)) +
  geom_errorbar(aes(ymin = mean_R2_marg - se_R2_marg, ymax = mean_R2_marg + se_R2_marg), width = 1.5) +
  geom_line(linewidth = 0.7) +
  geom_point(size = 1.6) +
  facet_grid(clump ~ dispersal, labeller = labeller(
    clump     = function(x) paste("clump =", x),
    dispersal = disp_lab)) +
  scale_colour_manual(values = c(base = "#C0392B", linked = "#3FA9A9"), name = "Linkage") +
  labs(title = expression(paste("MLPE marginal R"^2, " across the design")),
       subtitle = "Mean +/- SE over 10 replicates per cell",
       x = "Habitat area (%)", y = expression("Mean marginal R"^2)) +
  base_theme

ggsave(file.path(fig_dir, "figure_R2_by_dispersal.png"),
       fig3, width = 11, height = 8, dpi = 300)

# Fig 4 - Detection power heatmaps: rows = dispersal, cols = linkage
fig4 <- cell_sum %>%
  ggplot(aes(x = factor(area), y = factor(clump), fill = power_mlpe)) +
  geom_tile(colour = "white", linewidth = 0.2) +
  geom_text(aes(label = sprintf("%.2f", power_mlpe)),
            colour = "white", size = 2.6) +
  facet_grid(dispersal ~ type, labeller = labeller(
    dispersal = disp_lab,
    type      = c(base = "Base", linked = "Linked"))) +
  scale_fill_viridis_c(option = "magma", limits = c(0, 1), name = "Power\n(MLPE LR)") +
  labs(title = "MLPE detection power across area, clumping, linkage and dispersal",
       x = "Habitat area (%)", y = "Clumping (sigma)") +
  base_theme

ggsave(file.path(fig_dir, "figure_power_heatmaps_4panel.png"),
       fig4, width = 9, height = 9, dpi = 300)

# ---- 7. Rescue vs. signal-erosion duality (Fig 5) ----------------------------
# Pull final-generation Ho per simulation from compiled_1500_runs_Ho.csv,
# correlate cell-level dHo (linked - base) with cell-level dR2 (linked - base),
# stratified by dispersal block.

if (file.exists(ho_path)) {
  ho_raw <- read.csv(ho_path, stringsAsFactors = FALSE, check.names = FALSE)

  # Drop the index column if present
  if (names(ho_raw)[1] == "" || tolower(names(ho_raw)[1]) == "unnamed: 0" ||
      tolower(names(ho_raw)[1]) == "x") {
    ho_raw <- ho_raw[, -1, drop = FALSE]
  }

  if (ncol(ho_raw) != 1500) {
    warning(sprintf("Ho file has %d columns; expected 1500.", ncol(ho_raw)))
  }
  # Final generation = last row (gen 199)
  ho_last <- ho_raw[nrow(ho_raw), , drop = TRUE]

  # Each cell value is a string like "0.815|0.815|" - take the mean of the
  # parsed numbers per cell.
  parse_cell <- function(s) {
    if (is.na(s) || s == "") return(NA_real_)
    parts <- strsplit(as.character(s), "\\|")[[1]]
    parts <- parts[parts != ""]
    if (length(parts) == 0) return(NA_real_)
    mean(suppressWarnings(as.numeric(parts)), na.rm = TRUE)
  }
  ho_vals <- vapply(ho_last, parse_cell, numeric(1))

  # The 1500 columns are ordered: dispersal block (50/100/150) -> 500 cols each.
  # Within each 500-col block: area outer, clump middle, type middle-inner, mc inner.
  # Reconstruct the metadata from the column order.
  col_idx <- seq_len(1500)
  disp_block_idx <- (col_idx - 1) %/% 500     # 0,1,2
```

```
  in_block       <- (col_idx - 1) %% 500
  # Within block: 5 area * 5 clump * 2 type * 10 mc = 500
  area_idx_inb   <- in_block %/% 100          # 0..4 (5 areas)
  rest1          <- in_block %% 100
  clump_idx_inb  <- rest1 %/% 20              # 0..4 (5 clumps)
  rest2          <- rest1 %% 20
  type_idx_inb   <- rest2 %/% 10              # 0,1
  mc_idx         <- rest2 %% 10               # 0..9

  ho_df <- data.frame(
    col_idx   = col_idx,
    dispersal = c(50, 100, 150)[disp_block_idx + 1],
    area      = c(10, 20, 30, 40, 50)[area_idx_inb  + 1],
    clump     = c(3.01, 4, 5, 6, 7) [clump_idx_inb + 1],
    type      = c("base","linked")  [type_idx_inb  + 1],
    mc        = mc_idx,
    Ho        = ho_vals,
    stringsAsFactors = FALSE
  )

  # Per-cell mean Ho
  ho_cell <- ho_df %>%
    group_by(area, clump, type, dispersal) %>%
    summarise(mean_Ho = mean(Ho, na.rm = TRUE), .groups = "drop")

  # Build cell-level deltas: linked - base, per (area, clump, dispersal)
  delta_cell <- cell_sum %>%
    select(area, clump, type, dispersal, mean_R2_marg, mean_beta, mean_delta_r,
           power_mlpe) %>%
    inner_join(ho_cell, by = c("area","clump","type","dispersal")) %>%
    pivot_wider(names_from = type,
                values_from = c(mean_R2_marg, mean_beta, mean_delta_r,
                                power_mlpe, mean_Ho)) %>%
    mutate(
      dR2     = mean_R2_marg_linked  - mean_R2_marg_base,
      dBeta   = mean_beta_linked     - mean_beta_base,
      ddR     = mean_delta_r_linked  - mean_delta_r_base,
      dPower  = power_mlpe_linked    - power_mlpe_base,
      dHo     = mean_Ho_linked       - mean_Ho_base
    )
  write.csv(delta_cell, file.path(out_dir, "cell_deltas_linked_minus_base.csv"),
            row.names = FALSE)

  # Pearson correlations of dR2, dBeta, ddR with dHo, by dispersal
  cor_summary <- delta_cell %>%
    group_by(dispersal) %>%
    summarise(
      r_dR2_dHo   = cor(dR2,   dHo, use = "complete.obs"),
      r_dBeta_dHo = cor(dBeta, dHo, use = "complete.obs"),
      r_ddR_dHo   = cor(ddR,   dHo, use = "complete.obs"),
      n_cells     = sum(!is.na(dHo) & !is.na(dR2)),
      .groups = "drop"
    )
  cat("\n--- Rescue vs. signal-erosion correlations (cell-level) ---\n")
  print(cor_summary)
  write.csv(cor_summary, file.path(out_dir, "rescue_signal_correlations.csv"),
            row.names = FALSE)

  # Fig 5 - dR2 vs dHo by dispersal
  fig5 <- ggplot(delta_cell,
                 aes(x = dHo, y = dR2,
                     colour = factor(clump),
                     shape  = factor(area))) +
    geom_hline(yintercept = 0, linetype = "dashed", colour = "grey60", linewidth = 0.4) +
    geom_vline(xintercept = 0, linetype = "dashed", colour = "grey60", linewidth = 0.4) +
    geom_smooth(method = "lm", se = TRUE, colour = "black", linewidth = 0.5,
                inherit.aes = FALSE,
                aes(x = dHo, y = dR2)) +
    geom_point(size = 2.2, alpha = 0.9) +
    facet_wrap(~ dispersal, labeller = labeller(dispersal = disp_lab), nrow = 1) +
    scale_colour_brewer(palette = "Dark2", name = "Clumping (sigma)") +
    scale_shape_manual(values = c(15, 16, 17, 18, 8), name = "Area (%)") +
    labs(title = "Rescue vs. signal-erosion duality at the cell level",
         subtitle = expression(paste(Delta, "R"^2,
                                     " (linked - base) vs. ", Delta, "Ho (linked - base); one point per (area, clump)
cell")),
         x = expression(paste(Delta, " observed heterozygosity (Ho)")),
         y = expression(paste(Delta, " marginal R"^2))) +
```

```
    base_theme

  ggsave(file.path(fig_dir, "figure_dR2_vs_dHo.png"),
         fig5, width = 11, height = 4.6, dpi = 300)

} else {
  message("compiled_1500_runs_Ho.csv not found at ", ho_path,
          " - skipping Fig 5 (dR2 vs dHo).")
}

cat("\nAll figures and tables written to:\n  ", out_dir, "\n")
```