Noisy Circle: End-to-End gflow Workflow

Overview

This vignette demonstrates a full `gflow` workflow on a noisy circle:

1. Build and inspect ikNN graphs across a `k` range.
2. Select a graph scale using graph diagnostics.
3. Denoise coordinates with `data.smoother()`.
4. Smooth a noisy response `y` (mixture of two circular Gaussian peaks).
5. Smooth a feature matrix `Z`.
6. Find refined basins/extrema from `y.hat`.
7. Compare Pearson/Spearman and local correlation (`lcor`) analyses.
8. Add permutation-based inference for `lcor` (implemented here in the vignette).
9. Build a feature module graph from local-correlation profiles.

Helper Functions

```r
wrap_angle <- function(theta, mu) {
  atan2(sin(theta - mu), cos(theta - mu))
}

`%||%` <- function(x, y) {
  if (!is.null(x)) x else y
}

as_lcor_matrix <- function(x) {
  if (is.list(x) && !is.null(x$column.coefficients)) {
    return(as.matrix(x$column.coefficients))
  }
  as.matrix(x)
}

assign_from_vertices_list <- function(vertices.list, n) {
  out <- rep(NA_character_, n)
  if (is.null(vertices.list) || length(vertices.list) == 0) {
    return(out)
  }
  for (nm in names(vertices.list)) {
    out[as.integer(vertices.list[[nm]])] <- nm
  }
  out
}

compute_basins_safe <- function(adj.list, edge.length.list, fitted.values, verbose = TRUE) {
  has_both_extrema <- function(obj) {
    if (is.null(obj$summary) || nrow(obj$summary) == 0L || is.null(obj$summary$type)) {
      return(FALSE)
    }
```

```r
    tab <- table(obj$summary$type)
    ("max" %in% names(tab)) && ("min" %in% names(tab)) &&
      (tab[["max"]] > 0L) && (tab[["min"]] > 0L)
  }

  strict <- try(
    compute.refined.basins(
      adj.list = adj.list,
      edge.length.list = edge.length.list,
      fitted.values = fitted.values,
      edge.length.quantile.thld = 0.75,
      min.rel.value.max = 1.1,
      max.rel.value.min = 0.9,
      max.overlap.threshold = 0.15,
      min.overlap.threshold = 0.15,
      p.mean.nbrs.dist.threshold = 0.9,
      p.mean.hopk.dist.threshold = 0.9,
      p.deg.threshold = 0.9,
      min.basin.size = 10,
      expand.basins = TRUE,
      with.trajectories = TRUE,
      verbose = verbose
    ),
    silent = TRUE
  )

  if (!inherits(strict, "try-error") && has_both_extrema(strict)) {
    return(strict)
  }

  message("Strict basin settings failed (or removed all maxima/minima); retrying with relaxed settings.
  compute.refined.basins(
    adj.list = adj.list,
    edge.length.list = edge.length.list,
    fitted.values = fitted.values,
    edge.length.quantile.thld = 0.9,
    min.rel.value.max = 1.02,
    max.rel.value.min = 0.98,
    max.overlap.threshold = 0.25,
    min.overlap.threshold = 0.25,
    p.mean.nbrs.dist.threshold = 0.98,
    p.mean.hopk.dist.threshold = 0.98,
    p.deg.threshold = 0.98,
    min.basin.size = 5,
    apply.geometric.filter = FALSE,
    expand.basins = TRUE,
    with.trajectories = TRUE,
    verbose = verbose
  )
}

perm_test_lcor <- function(adj.list,
                           edge.length.list,
```

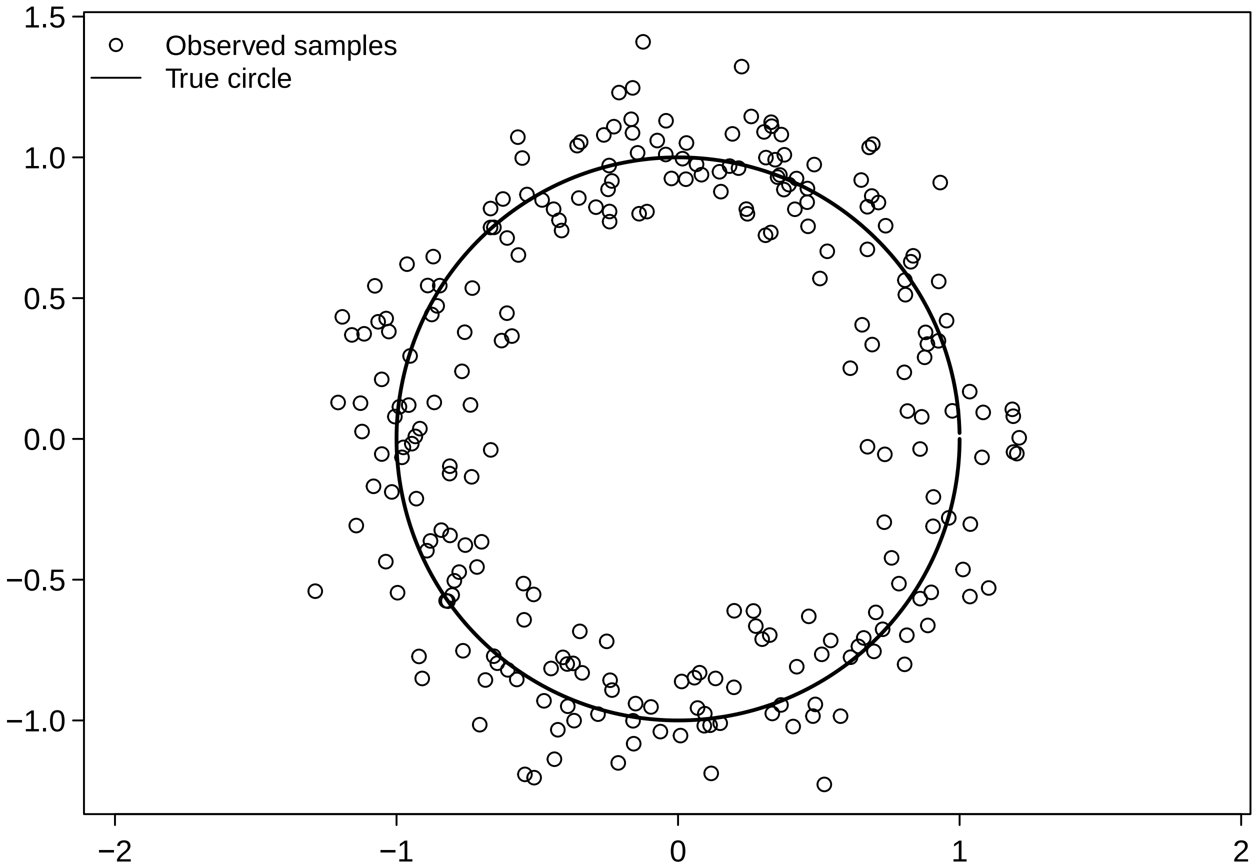
```
                         y,
                         Z,
                         type = "derivative",
                         y.diff.type = "difference",
                         z.diff.type = "difference",
                         n.perm = 200L,
                         seed = 1L) {
set.seed(seed)
Z <- as.matrix(Z)
y <- as.numeric(y)

lcor.obs <- as_lcor_matrix(
  lcor(
    adj.list = adj.list,
    weight.list = edge.length.list,
    y = y,
    z = Z,
    type = type,
    y.diff.type = y.diff.type,
    z.diff.type = z.diff.type
  )
)
stat.obs <- colMeans(abs(lcor.obs))

stat.perm <- matrix(NA_real_, nrow = n.perm, ncol = ncol(Z))
colnames(stat.perm) <- colnames(Z)

for (b in seq_len(n.perm)) {
  Zb <- Z[sample.int(nrow(Z)), , drop = FALSE]
  lcor.b <- as_lcor_matrix(
    lcor(
      adj.list = adj.list,
      weight.list = edge.length.list,
      y = y,
      z = Zb,
      type = type,
      y.diff.type = y.diff.type,
      z.diff.type = z.diff.type
    )
  )
  stat.perm[b, ] <- colMeans(abs(lcor.b))
}

p.value <- (1 + colSums(t(t(stat.perm) >= stat.obs))) / (n.perm + 1)
q.value <- stats::p.adjust(p.value, method = "BH")

list(
  stat.obs = stat.obs,
  stat.perm = stat.perm,
  p.value = p.value,
  q.value = q.value,
  table = data.frame(
    feature = colnames(Z),
    stat.obs = stat.obs,
```

```
      p.value = p.value,
      q.value = q.value,
      row.names = NULL
    )
  )
}
```

Simulate Noisy Circle and Response

```
n <- 250L
radius <- 1

X.df <- generate.circle.data(
  n = n,
  radius = radius,
  noise = 0.15,
  type = "random",
  noise.type = "normal",
  seed = 11
)
X <- as.matrix(X.df[, c("x", "y")])

if ("angles" %in% colnames(X.df)) {
  theta.obs <- as.numeric(X.df$angles)
} else {
  theta.obs <- atan2(X[, 2], X[, 1])
  theta.obs <- ifelse(theta.obs < 0, theta.obs + 2 * pi, theta.obs)
}

## Two wrapped Gaussian peaks over the circle: one larger, one smaller
mu1 <- 0.80
mu2 <- 3.95
sd1 <- 0.30
sd2 <- 0.48
amp1 <- 1.40
amp2 <- 0.75

comp1 <- amp1 * exp(-0.5 * (wrap_angle(theta.obs, mu1) / sd1)^2)
comp2 <- amp2 * exp(-0.5 * (wrap_angle(theta.obs, mu2) / sd2)^2)
y.true <- comp1 + comp2
y <- y.true + rnorm(n, sd = 0.20)

circle.true.df <- generate.circle.data(
  n = 300,
  radius = radius,
  noise = 0,
  type = "uniform",
  noise.type = "normal",
  seed = 1
)
circle.true <- as.matrix(circle.true.df[, c("x", "y")])
```

```r
op <- par(mar = c(2.5, 2.5, 0.5, 0.5), mgp = c(2.5, 0.5, 0), tcl = -0.3)
plot(X, asp = 1, las = 1, xlab = "", ylab = "")
lines(circle.true, lwd = 2)
legend("topleft",
       legend = c("Observed samples", "True circle"),
       pch = c(1, NA),
       lty = c(NA, 1),
       col = c("black", "black"),
       bty = "n", cex = 0.9)
```




```r
par(op)
```

Build ikNN Graphs Across k

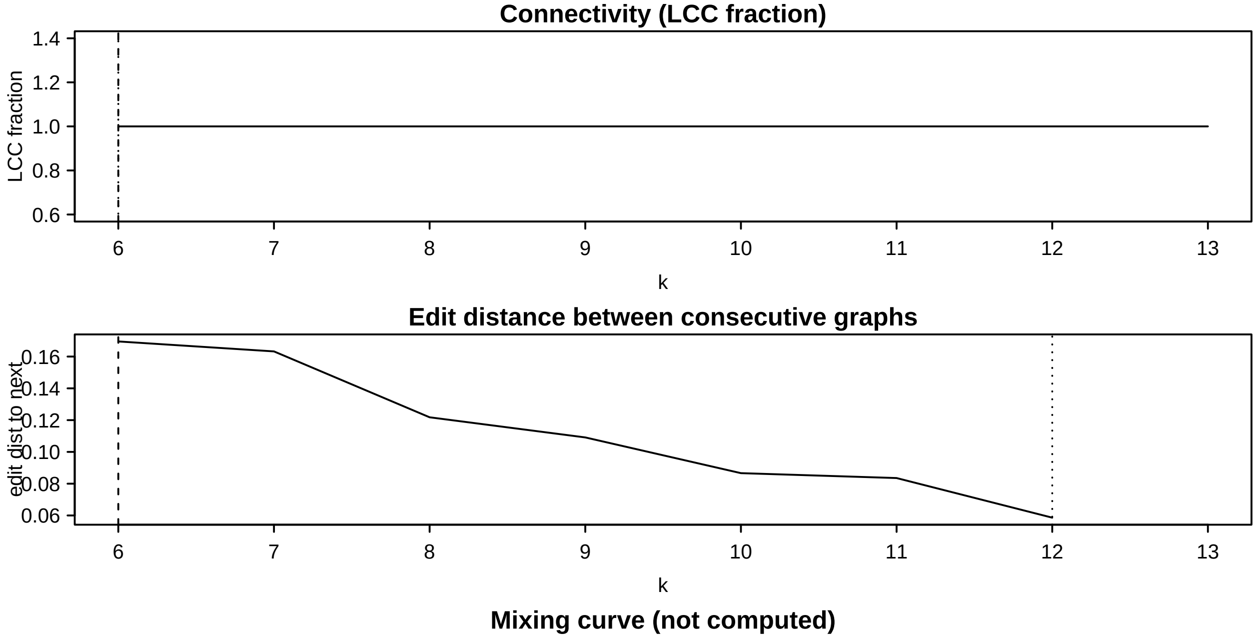
```r
k.min <- 5L
k.max <- 12L

X.graphs <- create.iknn.graphs(
  X,
  kmin = k.min,
  kmax = k.max,
  max.path.edge.ratio.deviation.thld = 0.1,
  n.cores = 1L,
  verbose = TRUE
)
#>  Using 1 OpenMP thread
```

```r
#> Processing k values from 5 to 12 for 250 vertices
#> Requested parallel.mode: auto
#> Parallel mode: serial
#> Starting graph processing
#> [compute_knn] running...[compute_knn] 0.001s
#> [graph_build/geometric_prune/isize_prune] running...[graph_build/geometric_prune/isize_prune] 0.361s
#> Graph processing completed (0.361)
#> [compute_knn] 0.001s
#> [graph_build] 0.013s
#> [geometric_prune] 0.348s
#> [isize_prune] skipped (with.isize.pruning=FALSE)
#> Creating return list objects ... DONE (0.000)
#> Total elapsed time (0.362)
summary(X.graphs)
#> Summary of iknn_graphs object
#> ----------------------------
#> Number of vertices: 250
#> k range: 5 to 12
#> Max path-edge ratio deviation threshold: 0.1
#> Path-edge ratio percentile: 0.5
#> Graph type: geometrically pruned
#>
#>   idx  k n_ccomp edges mean_degree min_degree max_degree sparsity
#>     1  5       1   877        7.02          2         15  0.97182
#>     2  6       1  1056        8.45          3         17  0.96607
#>     3  7       1  1262       10.10          3         22  0.95945
#>     4  8       1  1437       11.50          3         23  0.95383
#>     5  9       1  1613       12.90          3         29  0.94818
#>     6 10       1  1766       14.13          3         29  0.94326
#>     7 11       1  1927       15.42          3         32  0.93809
#>     8 12       1  2047       16.38          3         33  0.93423
```

Alternative k Selection Utility

```r
X.graphs2 <- build.iknn.graphs.and.selectk(
  X,
  kmin = k.min,
  kmax = k.max,
  method = "edit",
  n.cores = 1L,
  verbose = TRUE
)
#>  Using 1 OpenMP thread
#> Processing k values from 5 to 12 for 250 vertices
#> Requested parallel.mode: auto
#> Parallel mode: serial
#> Starting graph processing
#> [compute_knn] running...[compute_knn] 0.001s
#> [graph_build/geometric_prune/isize_prune] running...[graph_build/geometric_prune/isize_prune] 0.371s
#> Graph processing completed (0.371)
#> [compute_knn] 0.001s
#> [graph_build] 0.013s
#> [geometric_prune] 0.357s
```

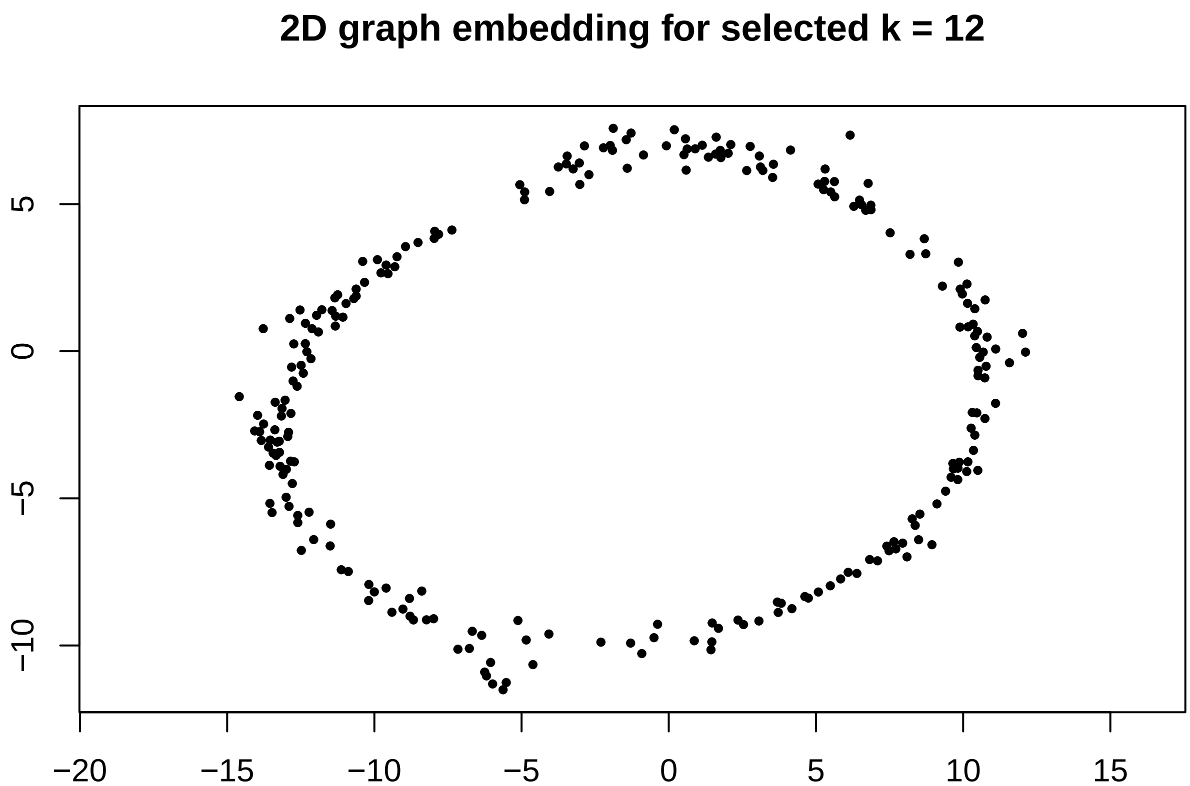
```
#> [isize_prune] skipped (with.isize.pruning=FALSE)
#> Creating return list objects ... DONE (0.000)
#> Total elapsed time (0.371)

X.graphs2$k.opt.edit
#> [1] 12
plot(X.graphs2)
```



Visualize One Graph Layout

```
k.values <- if (!is.null(colnames(X.graphs$k_statistics)) &&
                "k" %in% colnames(X.graphs$k_statistics)) {
  as.integer(X.graphs$k_statistics[, "k"])
} else {
  seq_len(length(X.graphs$geom_pruned_graphs))
}

sel.k <- as.integer(X.graphs2$k.opt.edit %||% k.values[1])
k.idx <- which(k.values == sel.k)[1]
if (is.na(k.idx)) k.idx <- 1L

g.sel <- X.graphs$geom_pruned_graphs[[k.idx]]
layout.2d <- graph.embedding(
  adj.list = g.sel$adj_list,
  weights.list = g.sel$weight_list,
  invert.weights = TRUE,
  dim = 2,
  method = "fr"
)

plot(layout.2d, pch = 19, cex = 0.5, asp = 1,
     main = sprintf("2D graph embedding for selected k = %d", sel.k),
```

```
    xlab = "", ylab = "")
```



Denoise X with `data.smoother()`

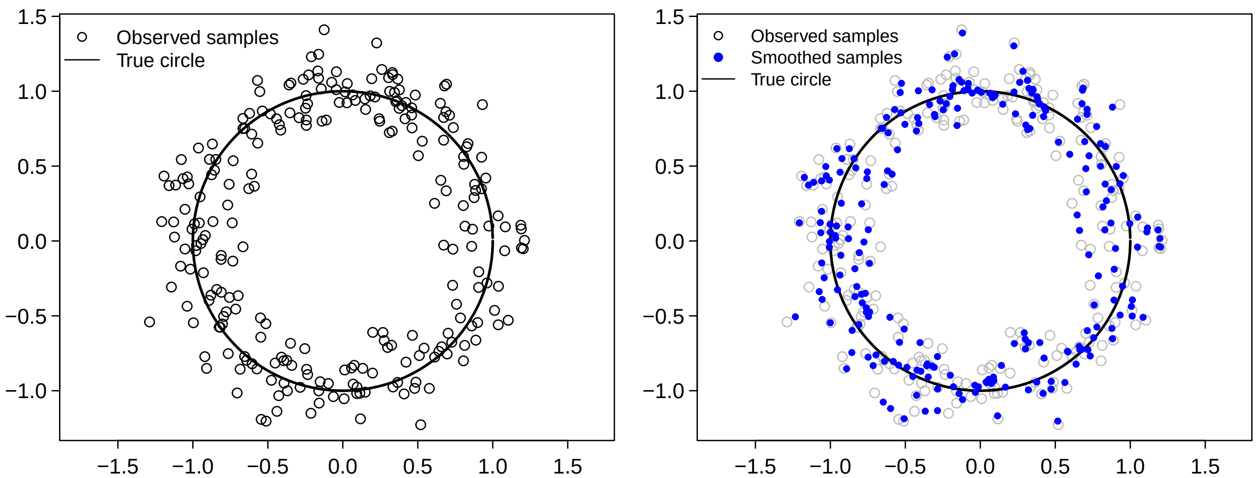
```
X.denoise.res <- data.smoother(
  X,
  kmin = k.min,
  kmax = k.max,
  n.cores = 1L,
  proxy.response = "pc1",
  max.iterations = 8L,
  n.eigenpairs = 100L,
  filter.type = "heat_kernel",
  t.scale.factor = 0.5,
  beta.coef.factor = 0.1,
  verbose = TRUE
)
#> Building ikNN graphs (geom pruned) for k in [5, 12]
#>  Using 1 OpenMP thread
#> Processing k values from 5 to 12 for 250 vertices
#> Requested parallel.mode: auto
#> Parallel mode: serial
#> Starting graph processing
#> [compute_knn] running...[compute_knn] 0.001s
#> [graph_build/geometric_prune/isize_prune] running...[graph_build/geometric_prune/isize_prune] 0.368s
#> Graph processing completed (0.368)
#> [compute_knn] 0.001s
#> [graph_build] 0.013s
```

```
#> [geometric_prune] 0.355s
#> [isize_prune] skipped (with.isize.pruning=FALSE)
#> Creating return list objects ... DONE (0.000)
#> Total elapsed time (0.369)
#> Summary of iknn_graphs object
#> ----------------------------
#> Number of vertices: 250
#> k range: 5 to 12
#> Max path-edge ratio deviation threshold: 0.1
#> Path-edge ratio percentile: 0.5
#> Graph type: geometrically pruned
#>
#>   idx  k n_ccomp edges mean_degree min_degree max_degree sparsity
#>     1  5       1   877        7.02          2         15  0.97182
#>     2  6       1  1056        8.45          3         17  0.96607
#>     3  7       1  1262       10.10          3         22  0.95945
#>     4  8       1  1437       11.50          3         23  0.95383
#>     5  9       1  1613       12.90          3         29  0.94818
#>     6 10       1  1766       14.13          3         29  0.94326
#>     7 11       1  1927       15.42          3         32  0.93809
#>     8 12       1  2047       16.38          3         33  0.93423
#> Proxy response: PC1 score (fraction variance explained = 0.5565)
#> Fitting fit.rdgraph.regression() across k values and extracting GCV
#> Selected k = 5 (min GCV = 1.212e-05)

X.smoothed <- X.denoise.res$X.smoothed
X.denoise.res$k.best
#> [1] 5
```

```
if (isTRUE(X.denoise.res$trimmed)) {
  kept.rows <- X.denoise.res$kept.rows
  X.use <- X[kept.rows, , drop = FALSE]
  theta.use <- theta.obs[kept.rows]
  y.use <- y[kept.rows]
  y.true.use <- y.true[kept.rows]
  comp1.use <- comp1[kept.rows]
  comp2.use <- comp2[kept.rows]
} else {
  X.use <- X
  theta.use <- theta.obs
  y.use <- y
  y.true.use <- y.true
  comp1.use <- comp1
  comp2.use <- comp2
}
```

```
op <- par(mfrow = c(1, 2),
          mar = c(2.75, 2.75, 0.5, 0.5),
          mgp = c(2.5, 0.5, 0),
          tcl = -0.3)

plot(X.use, las = 1, asp = 1, xlab = "", ylab = "")
lines(circle.true, lwd = 2)
legend("topleft",
```

```
       legend = c("Observed samples", "True circle"),
       pch = c(1, NA),
       lty = c(NA, 1),
       col = c("black", "black"),
       bty = "n", cex = 0.9)

plot(X.use, las = 1, asp = 1, col = "gray", xlab = "", ylab = "")
lines(circle.true, lwd = 2)
points(X.smoothed, col = "blue", pch = 19, cex = 0.6)
legend("topleft",
       legend = c("Observed samples", "Smoothed samples", "True circle"),
       pch = c(1, 19, NA),
       lty = c(NA, NA, 1),
       col = c("black", "blue", "black"),
       bty = "n", cex = 0.85)
```




```
par(op)
```

```
rad.obs <- sqrt(rowSums(X.use^2))
rad.sm <- sqrt(rowSums(X.smoothed^2))

rmse.obs <- sqrt(mean((rad.obs - radius)^2))
rmse.sm <- sqrt(mean((rad.sm - radius)^2))

data.frame(
  metric = c("RMSE radius (observed)", "RMSE radius (smoothed)"),
  value = c(rmse.obs, rmse.sm)
)
#>                    metric     value
#> 1 RMSE radius (observed) 0.1472329
#> 2 RMSE radius (smoothed) 0.1321032
```

Smooth y on the Selected Graph

We reuse the graph/spectral structure selected by `data.smoother()` and apply it to y via `refit.rdgraph.regression()`.

```
fit.seed <- X.denoise.res$fit.best

y.fit <- refit.rdgraph.regression(
```

```r
  fitted.model = fit.seed,
  y.new = as.double(y.use),
  per.column.gcv = FALSE,
  n.cores = 1L,
  verbose = FALSE
)

y.hat <- as.double(y.fit$fitted.values)

data.frame(
  metric = c("cor(y.hat, y.true)", "RMSE(y.hat, y.true)"),
  value = c(cor(y.hat, y.true.use),
            sqrt(mean((y.hat - y.true.use)^2)))
)
#>                metric    value
#> 1  cor(y.hat, y.true) 0.941873
#> 2 RMSE(y.hat, y.true) 0.128056
```

Optional 3D Layout and Surface Coloring

```r
X.denoise.graph.layout.3d <- graph.embedding(
  adj.list = fit.seed$graph$adj.list,
  weights.list = fit.seed$graph$edge.length.list,
  invert.weights = TRUE,
  dim = 3,
  method = "fr"
)
```

```r
if (requireNamespace("rgl", quietly = TRUE)) {
  plot3D.cont(
    X.denoise.graph.layout.3d,
    y.hat,
    radius = 0.03,
    legend.title = "y.hat"
  )
}
```

```r
if (requireNamespace("rgl", quietly = TRUE) &&
    requireNamespace("htmlwidgets", quietly = TRUE)) {
  plot3D.cont.html(
    X.denoise.graph.layout.3d,
    y.hat,
    legend.title = "y.hat",
    output.file = NULL
  )
}
```

Build Feature Matrix Z and Smooth It

Features are built so some correlate with selected arcs of the response, while others are pure noise.

```r
set.seed(1002)

window.1 <- as.numeric(abs(wrap_angle(theta.use, mu1)) <= 1.1)
```

```r
window.2 <- as.numeric(abs(wrap_angle(theta.use, mu2)) <= 1.1)

z.global <- y.true.use + rnorm(length(theta.use), sd = 0.08)
z.peak1.arc <- window.1 * y.true.use + rnorm(length(theta.use), sd = 0.08)
z.peak2.arc <- window.2 * y.true.use + rnorm(length(theta.use), sd = 0.08)
z.peak.contrast <- (comp1.use - comp2.use) + rnorm(length(theta.use), sd = 0.08)

z.noise1 <- rnorm(length(theta.use))
z.noise2 <- rnorm(length(theta.use))
z.noise3 <- rnorm(length(theta.use))

Z <- cbind(
  z_global = z.global,
  z_peak1_arc = z.peak1.arc,
  z_peak2_arc = z.peak2.arc,
  z_peak_contrast = z.peak.contrast,
  z_noise1 = z.noise1,
  z_noise2 = z.noise2,
  z_noise3 = z.noise3
)
Z <- scale(Z)
```

```r
Z.fit <- refit.rdgraph.regression(
  fitted.model = fit.seed,
  y.new = Z,
  per.column.gcv = FALSE,
  n.cores = 1L,
  verbose = FALSE
)
Z.sm <- as.matrix(Z.fit$fitted.values)
summary(Z.fit)
#>
#> Summary: Refitted Riemannian Graph Regression
#> ==============================================
#>
#> Observations: 250
#> Responses: 7
#>
#> Fit quality summary across 7 responses:
#>   R-sq: min=0.3491, Q1=0.3768, med=0.9406, Q3=0.9621, max=0.9775
#>   RMSE: min=0.1499, Q1=0.1941, med=0.2433, Q3=0.7879, max=0.8051
```

Compute Refined Basins from y.hat

```r
y.basins <- compute_basins_safe(
  adj.list = fit.seed$graph$adj.list,
  edge.length.list = fit.seed$graph$edge.length.list,
  fitted.values = y.hat,
  verbose = TRUE
)
#> Step 1: Computing initial basins of attraction...
#>   Found 17 maxima and 22 minima
#> initial.summary:
```

```
#>     label vertex       value   rel.value type hop.idx basin.size
#> 1      m1    245 -0.26630000 -0.81112583  min       2          4
#> 2      m2    120 -0.26183263 -0.79751862  min       2          7
#> 3      m3     92 -0.19902809 -0.60622164  min       2         10
#> 4      m4    230 -0.18650066 -0.56806424  min       3         13
#> 5      m5     78 -0.18240414 -0.55558661  min       4         14
#> 6      m6    208 -0.15682252 -0.47766730  min       1          3
#> 7      m7     93 -0.12477777 -0.38006185  min       3          6
#> 8      m8    250 -0.12346690 -0.37606906  min       1          6
#> 9      m9    101 -0.10990627 -0.33476461  min       1          6
#> 10    m10      6 -0.06511288 -0.19832799  min       6         15
#> 11    m11    203  0.02142402  0.06525565  min       8         25
#> 12    m12    211  0.02177421  0.06632230  min       2          6
#> 13    m13     64  0.04779295  0.14557303  min       5         30
#> 14    m14     72  0.05842157  0.17794685  min       1          4
#> 15    m15    132  0.12752894  0.38844168  min       3         15
#> 16    m16    146  0.18143515  0.55263513  min       1          6
#> 17    m17     55  0.26844958  0.81767328  min       3         14
#> 18    m18    180  0.37570701  1.14436974  min       4          6
#> 19    m19    162  0.56169281  1.71086575  min       1          3
#> 20    m20    164  0.62733373  1.91080209  min       2          7
#> 21    m21     33  0.97835737  2.97998852  min       2          5
#> 22    m22     23  1.01710762  3.09801830  min       0          1
#> 23     M1     30  1.46221755  4.45378309  max       8         39
#> 24     M2     28  1.32545340  4.03721181  max       2          7
#> 25     M3    148  0.96668007  2.94442051  max       0          1
#> 26     M4    163  0.90997849  2.77171260  max       2          3
#> 27     M5    153  0.85694374  2.61017354  max       2          8
#> 28     M6    190  0.54820939  1.66979647  max       4         12
#> 29     M7    217  0.35249067  1.07365485  max       1          6
#> 30     M8     60  0.34274227  1.04396212  max       0          1
#> 31     M9    103  0.33509074  1.02065624  max       2          7
#> 32    M10      8  0.29976846  0.91306776  max       3          5
#> 33    M11     77  0.29952927  0.91233921  max       4         18
#> 34    M12    129  0.27586066  0.84024674  max       1          3
#> 35    M13    216  0.25715592  0.78327380  max       1          4
#> 36    M14    247  0.24114239  0.73449802  max       3          8
#> 37    M15     71  0.22591942  0.68813024  max       0          1
#> 38    M16    233  0.16328801  0.49736059  max       2         11
#> 39    M17    102  0.12545658  0.38212947  max       2          7
#>    p.mean.nbrs.dist p.mean.hopk.dist deg p.deg
#> 1             0.788            0.912   4 0.956
#> 2             0.100            0.512   5 0.896
#> 3             0.168            0.092   6 0.684
#> 4             0.696            0.364   7 0.504
#> 5             0.404            0.648   6 0.684
#> 6             0.780            0.964   5 0.896
#> 7             0.812            0.720   5 0.896
#> 8             0.016            0.380   5 0.896
#> 9             0.012            0.632   5 0.896
#> 10            0.496            0.224   5 0.896
#> 11            0.928            0.916   4 0.956
#> 12            0.732            0.572   6 0.684
```

```
#> 13            0.164            0.080   9 0.252
#> 14            0.464            0.592   5 0.896
#> 15            0.748            0.480   6 0.684
#> 16            0.172            0.412   6 0.684
#> 17            0.348            0.176   7 0.504
#> 18            0.664            0.976   3 0.980
#> 19            0.860            0.848   5 0.896
#> 20            0.764            0.440   5 0.896
#> 21            0.624            0.952   5 0.896
#> 22            0.996            0.996   4 0.956
#> 23            0.328            0.208   3 0.980
#> 24            0.096            0.616   6 0.684
#> 25            1.000            1.000   3 0.980
#> 26            0.628            0.288   2 1.000
#> 27            0.580            0.524   7 0.504
#> 28            0.920            0.764   4 0.956
#> 29            0.304            0.388   6 0.684
#> 30            0.972            0.676   2 1.000
#> 31            0.548            0.472   5 0.896
#> 32            0.568            0.048   3 0.980
#> 33            0.408            0.232   5 0.896
#> 34            0.244            0.008   2 1.000
#> 35            0.444            0.904   6 0.684
#> 36            0.032            0.504   5 0.896
#> 37            0.988            0.992   2 1.000
#> 38            0.240            0.096   5 0.896
#> 39            0.020            0.756   5 0.896
#>
#> Step 2: Filtering by relative values (max >= 1.10, min <= 0.90)...
#>   Retained 6 maxima and 17 minima
#> relvalue.summary:
#>     label vertex       value   rel.value type hop.idx basin.size
#> 1      m1    245 -0.26630000 -0.81112583  min       2          4
#> 2      m2    120 -0.26183263 -0.79751862  min       2          7
#> 3      m3     92 -0.19902809 -0.60622164  min       2         10
#> 4      m4    230 -0.18650066 -0.56806424  min       3         13
#> 5      m5     78 -0.18240414 -0.55558661  min       4         14
#> 6      m6    208 -0.15682252 -0.47766730  min       1          3
#> 7      m7     93 -0.12477777 -0.38006185  min       3          6
#> 8      m8    250 -0.12346690 -0.37606906  min       1          6
#> 9      m9    101 -0.10990627 -0.33476461  min       1          6
#> 10    m10      6 -0.06511288 -0.19832799  min       6         15
#> 11    m11    203  0.02142402  0.06525565  min       8         25
#> 12    m12    211  0.02177421  0.06632230  min       2          6
#> 13    m13     64  0.04779295  0.14557303  min       5         30
#> 14    m14     72  0.05842157  0.17794685  min       1          4
#> 15    m15    132  0.12752894  0.38844168  min       3         15
#> 16    m16    146  0.18143515  0.55263513  min       1          6
#> 17    m17     55  0.26844958  0.81767328  min       3         14
#> 18     M1     30  1.46221755  4.45378309  max       8         39
#> 19     M2     28  1.32545340  4.03721181  max       2          7
#> 20     M3    148  0.96668007  2.94442051  max       0          1
#> 21     M4    163  0.90997849  2.77171260  max       2          3
```

```
#> 22    M5    153  0.85694374  2.61017354  max       2          8
#> 23    M6    190  0.54820939  1.66979647  max       4         12
#>    p.mean.nbrs.dist p.mean.hopk.dist deg p.deg
#> 1             0.788            0.912   4 0.956
#> 2             0.100            0.512   5 0.896
#> 3             0.168            0.092   6 0.684
#> 4             0.696            0.364   7 0.504
#> 5             0.404            0.648   6 0.684
#> 6             0.780            0.964   5 0.896
#> 7             0.812            0.720   5 0.896
#> 8             0.016            0.380   5 0.896
#> 9             0.012            0.632   5 0.896
#> 10            0.496            0.224   5 0.896
#> 11            0.928            0.916   4 0.956
#> 12            0.732            0.572   6 0.684
#> 13            0.164            0.080   9 0.252
#> 14            0.464            0.592   5 0.896
#> 15            0.748            0.480   6 0.684
#> 16            0.172            0.412   6 0.684
#> 17            0.348            0.176   7 0.504
#> 18            0.328            0.208   3 0.980
#> 19            0.096            0.616   6 0.684
#> 20            1.000            1.000   3 0.980
#> 21            0.628            0.288   2 1.000
#> 22            0.580            0.524   7 0.504
#> 23            0.920            0.764   4 0.956
#>
#> Step 3: Clustering maxima (overlap threshold = 0.15)...
#>   Found 6 clusters (6 singletons, 0 to be merged)
#>   Merging clustered maxima...
#>   Result: 6 maxima after merging
#> merged.max.summary:
#>    label vertex       value   rel.value type hop.idx basin.size
#> 1     m1    245 -0.26630000 -0.81112583  min       2          4
#> 2     m2    120 -0.26183263 -0.79751862  min       2          7
#> 3     m3     92 -0.19902809 -0.60622164  min       2         10
#> 4     m4    230 -0.18650066 -0.56806424  min       3         13
#> 5     m5     78 -0.18240414 -0.55558661  min       4         14
#> 6     m6    208 -0.15682252 -0.47766730  min       1          3
#> 7     m7     93 -0.12477777 -0.38006185  min       3          6
#> 8     m8    250 -0.12346690 -0.37606906  min       1          6
#> 9     m9    101 -0.10990627 -0.33476461  min       1          6
#> 10   m10      6 -0.06511288 -0.19832799  min       6         15
#> 11   m11    203  0.02142402  0.06525565  min       8         25
#> 12   m12    211  0.02177421  0.06632230  min       2          6
#> 13   m13     64  0.04779295  0.14557303  min       5         30
#> 14   m14     72  0.05842157  0.17794685  min       1          4
#> 15   m15    132  0.12752894  0.38844168  min       3         15
#> 16   m16    146  0.18143515  0.55263513  min       1          6
#> 17   m17     55  0.26844958  0.81767328  min       3         14
#> 18    M1     30  1.46221755  4.45378309  max       8         39
#> 19    M2     28  1.32545340  4.03721181  max       2          7
#> 20    M3    148  0.96668007  2.94442051  max       0          1
```

```
#> 21    M4    163  0.90997849  2.77171260  max       2          3
#> 22    M5    153  0.85694374  2.61017354  max       2          8
#> 23    M6    190  0.54820939  1.66979647  max       4         12
#>    p.mean.nbrs.dist p.mean.hopk.dist deg p.deg
#> 1             0.788            0.912   4 0.956
#> 2             0.100            0.512   5 0.896
#> 3             0.168            0.092   6 0.684
#> 4             0.696            0.364   7 0.504
#> 5             0.404            0.648   6 0.684
#> 6             0.780            0.964   5 0.896
#> 7             0.812            0.720   5 0.896
#> 8             0.016            0.380   5 0.896
#> 9             0.012            0.632   5 0.896
#> 10            0.496            0.224   5 0.896
#> 11            0.928            0.916   4 0.956
#> 12            0.732            0.572   6 0.684
#> 13            0.164            0.080   9 0.252
#> 14            0.464            0.592   5 0.896
#> 15            0.748            0.480   6 0.684
#> 16            0.172            0.412   6 0.684
#> 17            0.348            0.176   7 0.504
#> 18            0.328            0.208   3 0.980
#> 19            0.096            0.616   6 0.684
#> 20            1.000            1.000   3 0.980
#> 21            0.628            0.288   2 1.000
#> 22            0.580            0.524   7 0.504
#> 23            0.920            0.764   4 0.956
#>
#> Step 4: Clustering minima (overlap threshold = 0.15)...
#>   Found 16 clusters (15 singletons, 1 to be merged)
#>   Merging clustered minima...
#>   Result: 16 minima after merging
#> merged.min.summary:
#>    label vertex       value   rel.value type hop.idx basin.size
#> 1     m1    245 -0.26630000 -0.81112583  min       2          4
#> 2     m2    120 -0.26183263 -0.79751862  min       2          7
#> 3     m3     92 -0.19902809 -0.60622164  min       2         10
#> 4     m4    230 -0.18650066 -0.56806424  min       3         13
#> 5     m5     78 -0.18240414 -0.55558661  min       4         14
#> 6     m6    208 -0.15682252 -0.47766730  min       1          3
#> 7     m7     93 -0.12477777 -0.38006185  min       3          6
#> 8     m8    250 -0.12346690 -0.37606906  min       1          6
#> 9     m9    101 -0.10990627 -0.33476461  min       1          6
#> 10   m10      6 -0.06511288 -0.19832799  min       6         15
#> 11   m11    203  0.02142402  0.06525565  min       8         25
#> 12   m12    211  0.02177421  0.06632230  min       2          6
#> 13   m13     64  0.04779295  0.14557303  min       5         31
#> 14   m14     72  0.05842157  0.17794685  min       1          4
#> 15   m15    132  0.12752894  0.38844168  min       3         15
#> 16   m16    146  0.18143515  0.55263513  min       1          6
#> 17    M1     30  1.46221755  4.45378309  max       8         39
#> 18    M2     28  1.32545340  4.03721181  max       2          7
#> 19    M3    148  0.96668007  2.94442051  max       0          1
```

```
#> 20    M4    163  0.90997849  2.77171260  max       2          3
#> 21    M5    153  0.85694374  2.61017354  max       2          8
#> 22    M6    190  0.54820939  1.66979647  max       4         12
#>    p.mean.nbrs.dist p.mean.hopk.dist deg p.deg
#> 1             0.788            0.912   4 0.956
#> 2             0.100            0.512   5 0.896
#> 3             0.168            0.092   6 0.684
#> 4             0.696            0.364   7 0.504
#> 5             0.404            0.648   6 0.684
#> 6             0.780            0.964   5 0.896
#> 7             0.812            0.720   5 0.896
#> 8             0.016            0.380   5 0.896
#> 9             0.012            0.632   5 0.896
#> 10            0.496            0.224   5 0.896
#> 11            0.928            0.916   4 0.956
#> 12            0.732            0.572   6 0.684
#> 13            0.164            0.080   9 0.252
#> 14            0.464            0.592   5 0.896
#> 15            0.748            0.480   6 0.684
#> 16            0.172            0.412   6 0.684
#> 17            0.328            0.208   3 0.980
#> 18            0.096            0.616   6 0.684
#> 19            1.000            1.000   3 0.980
#> 20            0.628            0.288   2 1.000
#> 21            0.580            0.524   7 0.504
#> 22            0.920            0.764   4 0.956
#>
#> Step 5: Filtering by geometric characteristics and basin size:
#>   p.mean.nbrs.dist < 0.90, p.mean.hopk.dist < 0.90, p.deg < 0.90, basin.size >= 10 ...
#>   Maxima: 6 -> 0 (removed 6)
#> geom.summary:
#>    label vertex       value   rel.value type hop.idx basin.size
#> 1     m1    245 -0.26630000 -0.81112583  min       2          4
#> 2     m2    120 -0.26183263 -0.79751862  min       2          7
#> 3     m3     92 -0.19902809 -0.60622164  min       2         10
#> 4     m4    230 -0.18650066 -0.56806424  min       3         13
#> 5     m5     78 -0.18240414 -0.55558661  min       4         14
#> 6     m6    208 -0.15682252 -0.47766730  min       1          3
#> 7     m7     93 -0.12477777 -0.38006185  min       3          6
#> 8     m8    250 -0.12346690 -0.37606906  min       1          6
#> 9     m9    101 -0.10990627 -0.33476461  min       1          6
#> 10   m10      6 -0.06511288 -0.19832799  min       6         15
#> 11   m11    203  0.02142402  0.06525565  min       8         25
#> 12   m12    211  0.02177421  0.06632230  min       2          6
#> 13   m13     64  0.04779295  0.14557303  min       5         31
#> 14   m14     72  0.05842157  0.17794685  min       1          4
#> 15   m15    132  0.12752894  0.38844168  min       3         15
#> 16   m16    146  0.18143515  0.55263513  min       1          6
#> 17    M1     30  1.46221755  4.45378309  max       8         39
#> 18    M2     28  1.32545340  4.03721181  max       2          7
#> 19    M3    148  0.96668007  2.94442051  max       0          1
#> 20    M4    163  0.90997849  2.77171260  max       2          3
#> 21    M5    153  0.85694374  2.61017354  max       2          8
```

```
#> 22    M6    190  0.54820939  1.66979647  max       4         12
#>    p.mean.nbrs.dist p.mean.hopk.dist deg p.deg
#> 1             0.788            0.912   4 0.956
#> 2             0.100            0.512   5 0.896
#> 3             0.168            0.092   6 0.684
#> 4             0.696            0.364   7 0.504
#> 5             0.404            0.648   6 0.684
#> 6             0.780            0.964   5 0.896
#> 7             0.812            0.720   5 0.896
#> 8             0.016            0.380   5 0.896
#> 9             0.012            0.632   5 0.896
#> 10            0.496            0.224   5 0.896
#> 11            0.928            0.916   4 0.956
#> 12            0.732            0.572   6 0.684
#> 13            0.164            0.080   9 0.252
#> 14            0.464            0.592   5 0.896
#> 15            0.748            0.480   6 0.684
#> 16            0.172            0.412   6 0.684
#> 17            0.328            0.208   3 0.980
#> 18            0.096            0.616   6 0.684
#> 19            1.000            1.000   3 0.980
#> 20            0.628            0.288   2 1.000
#> 21            0.580            0.524   7 0.504
#> 22            0.920            0.764   4 0.956
#>   Minima: 16 -> 6 (removed 10)
#>
#> Generating final summary (hop.k = 2)...
#> Populating extrema labels in basins object...
#> final.summary:
#>   label vertex       value  rel.value type hop.idx basin.size p.mean.nbrs.dist
#> 1    m1     92 -0.19902809 -0.6062216  min       2         10            0.168
#> 2    m2    230 -0.18650066 -0.5680642  min       3         13            0.696
#> 3    m3     78 -0.18240414 -0.5555866  min       4         14            0.404
#> 4    m4      6 -0.06511288 -0.1983280  min       6         15            0.496
#> 5    m5     64  0.04779295  0.1455730  min       5         31            0.164
#> 6    m6    132  0.12752894  0.3884417  min       3         15            0.748
#>   p.mean.hopk.dist deg p.deg
#> 1            0.092   6 0.684
#> 2            0.364   7 0.504
#> 3            0.648   6 0.684
#> 4            0.224   5 0.896
#> 5            0.080   9 0.252
#> 6            0.480   6 0.684
#>   Assigned 0 maximum labels
#>   Assigned 6 minimum labels
#> Storing graph structure in basins object...
#> Constructing basin vertices lists...
#> Computing final overlap distance matrices...
#>
#> Step 6: Expanding basins to cover all vertices...
#>   Minima basins: 90 -> 250 vertices covered
#>
#> Refinement complete!
```

```
#> Final structure: 0 maxima and 6 minima
#> Step 1: Computing initial basins of attraction...
#>   Found 17 maxima and 22 minima
#> initial.summary:
#>    label vertex       value   rel.value type hop.idx basin.size
#> 1     m1    245 -0.26630000 -0.81112583  min       6         16
#> 2     m2    120 -0.26183263 -0.79751862  min       4         16
#> 3     m3     92 -0.19902809 -0.60622164  min       9         57
#> 4     m4    230 -0.18650066 -0.56806424  min       4         17
#> 5     m5     78 -0.18240414 -0.55558661  min       8         45
#> 6     m6    208 -0.15682252 -0.47766730  min       2          6
#> 7     m7     93 -0.12477777 -0.38006185  min      11         51
#> 8     m8    250 -0.12346690 -0.37606906  min       2          7
#> 9     m9    101 -0.10990627 -0.33476461  min       1          6
#> 10   m10      6 -0.06511288 -0.19832799  min       5         22
#> 11   m11    203  0.02142402  0.06525565  min       8         38
#> 12   m12    211  0.02177421  0.06632230  min       3         10
#> 13   m13     64  0.04779295  0.14557303  min       5         39
#> 14   m14     72  0.05842157  0.17794685  min       1          6
#> 15   m15    132  0.12752894  0.38844168  min       4         22
#> 16   m16    146  0.18143515  0.55263513  min       4         15
#> 17   m17     55  0.26844958  0.81767328  min       4         22
#> 18   m18    180  0.37570701  1.14436974  min       6         11
#> 19   m19    162  0.56169281  1.71086575  min       2          5
#> 20   m20    164  0.62733373  1.91080209  min       2          7
#> 21   m21     33  0.97835737  2.97998852  min       4          9
#> 22   m22     23  1.01710762  3.09801830  min       0          1
#> 23    M1     30  1.46221755  4.45378309  max       9         63
#> 24    M2     28  1.32545340  4.03721181  max       2          8
#> 25    M3    148  0.96668007  2.94442051  max       0          1
#> 26    M4    163  0.90997849  2.77171260  max       2          3
#> 27    M5    153  0.85694374  2.61017354  max       6         25
#> 28    M6    190  0.54820939  1.66979647  max       4         16
#> 29    M7    217  0.35249067  1.07365485  max       3          9
#> 30    M8     60  0.34274227  1.04396212  max       1          2
#> 31    M9    103  0.33509074  1.02065624  max       4         16
#> 32   M10      8  0.29976846  0.91306776  max       2          7
#> 33   M11     77  0.29952927  0.91233921  max       4         26
#> 34   M12    129  0.27586066  0.84024674  max       1          3
#> 35   M13    216  0.25715592  0.78327380  max       5         20
#> 36   M14    247  0.24114239  0.73449802  max       3          8
#> 37   M15     71  0.22591942  0.68813024  max       1          2
#> 38   M16    233  0.16328801  0.49736059  max       2         13
#> 39   M17    102  0.12545658  0.38212947  max       3         11
#>    p.mean.nbrs.dist p.mean.hopk.dist deg p.deg
#> 1             0.788            0.912   4 0.956
#> 2             0.100            0.512   5 0.896
#> 3             0.168            0.092   6 0.684
#> 4             0.696            0.364   7 0.504
#> 5             0.404            0.648   6 0.684
#> 6             0.780            0.964   5 0.896
#> 7             0.812            0.720   5 0.896
#> 8             0.016            0.380   5 0.896
```

```
#> 9             0.012           0.632   5 0.896
#> 10            0.496           0.224   5 0.896
#> 11            0.928           0.916   4 0.956
#> 12            0.732           0.572   6 0.684
#> 13            0.164           0.080   9 0.252
#> 14            0.464           0.592   5 0.896
#> 15            0.748           0.480   6 0.684
#> 16            0.172           0.412   6 0.684
#> 17            0.348           0.176   7 0.504
#> 18            0.664           0.976   3 0.980
#> 19            0.860           0.848   5 0.896
#> 20            0.764           0.440   5 0.896
#> 21            0.624           0.952   5 0.896
#> 22            0.996           0.996   4 0.956
#> 23            0.328           0.208   3 0.980
#> 24            0.096           0.616   6 0.684
#> 25            1.000           1.000   3 0.980
#> 26            0.628           0.288   2 1.000
#> 27            0.580           0.524   7 0.504
#> 28            0.920           0.764   4 0.956
#> 29            0.304           0.388   6 0.684
#> 30            0.972           0.676   2 1.000
#> 31            0.548           0.472   5 0.896
#> 32            0.568           0.048   3 0.980
#> 33            0.408           0.232   5 0.896
#> 34            0.244           0.008   2 1.000
#> 35            0.444           0.904   6 0.684
#> 36            0.032           0.504   5 0.896
#> 37            0.988           0.992   2 1.000
#> 38            0.240           0.096   5 0.896
#> 39            0.020           0.756   5 0.896
#>
#> Step 2: Filtering by relative values (max >= 1.02, min <= 0.98)...
#>   Retained 9 maxima and 17 minima
#> relvalue.summary:
#>     label vertex       value   rel.value type hop.idx basin.size
#> 1      m1    245 -0.26630000 -0.81112583  min       6         16
#> 2      m2    120 -0.26183263 -0.79751862  min       4         16
#> 3      m3     92 -0.19902809 -0.60622164  min       9         57
#> 4      m4    230 -0.18650066 -0.56806424  min       4         17
#> 5      m5     78 -0.18240414 -0.55558661  min       8         45
#> 6      m6    208 -0.15682252 -0.47766730  min       2          6
#> 7      m7     93 -0.12477777 -0.38006185  min      11         51
#> 8      m8    250 -0.12346690 -0.37606906  min       2          7
#> 9      m9    101 -0.10990627 -0.33476461  min       1          6
#> 10    m10      6 -0.06511288 -0.19832799  min       5         22
#> 11    m11    203  0.02142402  0.06525565  min       8         38
#> 12    m12    211  0.02177421  0.06632230  min       3         10
#> 13    m13     64  0.04779295  0.14557303  min       5         39
#> 14    m14     72  0.05842157  0.17794685  min       1          6
#> 15    m15    132  0.12752894  0.38844168  min       4         22
#> 16    m16    146  0.18143515  0.55263513  min       4         15
#> 17    m17     55  0.26844958  0.81767328  min       4         22
```

```
#> 18    M1     30  1.46221755  4.45378309  max       9         63
#> 19    M2     28  1.32545340  4.03721181  max       2          8
#> 20    M3    148  0.96668007  2.94442051  max       0          1
#> 21    M4    163  0.90997849  2.77171260  max       2          3
#> 22    M5    153  0.85694374  2.61017354  max       6         25
#> 23    M6    190  0.54820939  1.66979647  max       4         16
#> 24    M7    217  0.35249067  1.07365485  max       3          9
#> 25    M8     60  0.34274227  1.04396212  max       1          2
#> 26    M9    103  0.33509074  1.02065624  max       4         16
#>    p.mean.nbrs.dist p.mean.hopk.dist deg p.deg
#> 1             0.788            0.912   4 0.956
#> 2             0.100            0.512   5 0.896
#> 3             0.168            0.092   6 0.684
#> 4             0.696            0.364   7 0.504
#> 5             0.404            0.648   6 0.684
#> 6             0.780            0.964   5 0.896
#> 7             0.812            0.720   5 0.896
#> 8             0.016            0.380   5 0.896
#> 9             0.012            0.632   5 0.896
#> 10            0.496            0.224   5 0.896
#> 11            0.928            0.916   4 0.956
#> 12            0.732            0.572   6 0.684
#> 13            0.164            0.080   9 0.252
#> 14            0.464            0.592   5 0.896
#> 15            0.748            0.480   6 0.684
#> 16            0.172            0.412   6 0.684
#> 17            0.348            0.176   7 0.504
#> 18            0.328            0.208   3 0.980
#> 19            0.096            0.616   6 0.684
#> 20            1.000            1.000   3 0.980
#> 21            0.628            0.288   2 1.000
#> 22            0.580            0.524   7 0.504
#> 23            0.920            0.764   4 0.956
#> 24            0.304            0.388   6 0.684
#> 25            0.972            0.676   2 1.000
#> 26            0.548            0.472   5 0.896
#>
#> Step 3: Clustering maxima (overlap threshold = 0.25)...
#>   Found 9 clusters (9 singletons, 0 to be merged)
#>   Merging clustered maxima...
#>   Result: 9 maxima after merging
#> merged.max.summary:
#>    label vertex       value   rel.value type hop.idx basin.size
#> 1     m1    245 -0.26630000 -0.81112583  min       6         16
#> 2     m2    120 -0.26183263 -0.79751862  min       4         16
#> 3     m3     92 -0.19902809 -0.60622164  min       9         57
#> 4     m4    230 -0.18650066 -0.56806424  min       4         17
#> 5     m5     78 -0.18240414 -0.55558661  min       8         45
#> 6     m6    208 -0.15682252 -0.47766730  min       2          6
#> 7     m7     93 -0.12477777 -0.38006185  min      11         51
#> 8     m8    250 -0.12346690 -0.37606906  min       2          7
#> 9     m9    101 -0.10990627 -0.33476461  min       1          6
#> 10   m10      6 -0.06511288 -0.19832799  min       5         22
```

```
#> 11   m11    203  0.02142402  0.06525565  min       8         38
#> 12   m12    211  0.02177421  0.06632230  min       3         10
#> 13   m13     64  0.04779295  0.14557303  min       5         39
#> 14   m14     72  0.05842157  0.17794685  min       1          6
#> 15   m15    132  0.12752894  0.38844168  min       4         22
#> 16   m16    146  0.18143515  0.55263513  min       4         15
#> 17   m17     55  0.26844958  0.81767328  min       4         22
#> 18    M1     30  1.46221755  4.45378309  max       9         63
#> 19    M2     28  1.32545340  4.03721181  max       2          8
#> 20    M3    148  0.96668007  2.94442051  max       0          1
#> 21    M4    163  0.90997849  2.77171260  max       2          3
#> 22    M5    153  0.85694374  2.61017354  max       6         25
#> 23    M6    190  0.54820939  1.66979647  max       4         16
#> 24    M7    217  0.35249067  1.07365485  max       3          9
#> 25    M8     60  0.34274227  1.04396212  max       1          2
#> 26    M9    103  0.33509074  1.02065624  max       4         16
#>    p.mean.nbrs.dist p.mean.hopk.dist deg p.deg
#> 1             0.788            0.912   4 0.956
#> 2             0.100            0.512   5 0.896
#> 3             0.168            0.092   6 0.684
#> 4             0.696            0.364   7 0.504
#> 5             0.404            0.648   6 0.684
#> 6             0.780            0.964   5 0.896
#> 7             0.812            0.720   5 0.896
#> 8             0.016            0.380   5 0.896
#> 9             0.012            0.632   5 0.896
#> 10            0.496            0.224   5 0.896
#> 11            0.928            0.916   4 0.956
#> 12            0.732            0.572   6 0.684
#> 13            0.164            0.080   9 0.252
#> 14            0.464            0.592   5 0.896
#> 15            0.748            0.480   6 0.684
#> 16            0.172            0.412   6 0.684
#> 17            0.348            0.176   7 0.504
#> 18            0.328            0.208   3 0.980
#> 19            0.096            0.616   6 0.684
#> 20            1.000            1.000   3 0.980
#> 21            0.628            0.288   2 1.000
#> 22            0.580            0.524   7 0.504
#> 23            0.920            0.764   4 0.956
#> 24            0.304            0.388   6 0.684
#> 25            0.972            0.676   2 1.000
#> 26            0.548            0.472   5 0.896
#>
#> Step 4: Clustering minima (overlap threshold = 0.25)...
#>   Found 11 clusters (8 singletons, 3 to be merged)
#>   Merging clustered minima...
#>   Result: 11 minima after merging
#> merged.min.summary:
#>     label vertex       value   rel.value type hop.idx basin.size
#> 1      m1    245 -0.26630000 -0.81112583  min       6         25
#> 2      m2    120 -0.26183263 -0.79751862  min       4         16
#> 3      m3     92 -0.19902809 -0.60622164  min       9         64
```

```
#> 4      m4    230 -0.18650066 -0.56806424  min       4         17
#> 5      m5    208 -0.15682252 -0.47766730  min       2          6
#> 6      m6    250 -0.12346690 -0.37606906  min       2          7
#> 7      m7    101 -0.10990627 -0.33476461  min       1          6
#> 8      m8    203  0.02142402  0.06525565  min       8         40
#> 9      m9     72  0.05842157  0.17794685  min       1          6
#> 10    m10    132  0.12752894  0.38844168  min       4         22
#> 11    m11    146  0.18143515  0.55263513  min       4         15
#> 12     M1     30  1.46221755  4.45378309  max       9         63
#> 13     M2     28  1.32545340  4.03721181  max       2          8
#> 14     M3    148  0.96668007  2.94442051  max       0          1
#> 15     M4    163  0.90997849  2.77171260  max       2          3
#> 16     M5    153  0.85694374  2.61017354  max       6         25
#> 17     M6    190  0.54820939  1.66979647  max       4         16
#> 18     M7    217  0.35249067  1.07365485  max       3          9
#> 19     M8     60  0.34274227  1.04396212  max       1          2
#> 20     M9    103  0.33509074  1.02065624  max       4         16
#>    p.mean.nbrs.dist p.mean.hopk.dist deg p.deg
#> 1             0.788            0.912   4 0.956
#> 2             0.100            0.512   5 0.896
#> 3             0.168            0.092   6 0.684
#> 4             0.696            0.364   7 0.504
#> 5             0.780            0.964   5 0.896
#> 6             0.016            0.380   5 0.896
#> 7             0.012            0.632   5 0.896
#> 8             0.928            0.916   4 0.956
#> 9             0.464            0.592   5 0.896
#> 10            0.748            0.480   6 0.684
#> 11            0.172            0.412   6 0.684
#> 12            0.328            0.208   3 0.980
#> 13            0.096            0.616   6 0.684
#> 14            1.000            1.000   3 0.980
#> 15            0.628            0.288   2 1.000
#> 16            0.580            0.524   7 0.504
#> 17            0.920            0.764   4 0.956
#> 18            0.304            0.388   6 0.684
#> 19            0.972            0.676   2 1.000
#> 20            0.548            0.472   5 0.896
#>
#> Step 5: Skipping geometric filtering
#>
#> Generating final summary (hop.k = 2)...
#> Populating extrema labels in basins object...
#> final.summary:
#>    label vertex       value   rel.value type hop.idx basin.size
#> 1      m1    245 -0.26630000 -0.81112583  min       6         25
#> 2      m2    120 -0.26183263 -0.79751862  min       4         16
#> 3      m3     92 -0.19902809 -0.60622164  min       9         64
#> 4      m4    230 -0.18650066 -0.56806424  min       4         17
#> 5      m5    208 -0.15682252 -0.47766730  min       2          6
#> 6      m6    250 -0.12346690 -0.37606906  min       2          7
#> 7      m7    101 -0.10990627 -0.33476461  min       1          6
#> 8      m8    203  0.02142402  0.06525565  min       8         40
```

```
#> 9     m9     72  0.05842157  0.17794685  min       1          6
#> 10   m10    132  0.12752894  0.38844168  min       4         22
#> 11   m11    146  0.18143515  0.55263513  min       4         15
#> 12    M1     30  1.46221755  4.45378309  max       9         63
#> 13    M2     28  1.32545340  4.03721181  max       2          8
#> 14    M3    148  0.96668007  2.94442051  max       0          1
#> 15    M4    163  0.90997849  2.77171260  max       2          3
#> 16    M5    153  0.85694374  2.61017354  max       6         25
#> 17    M6    190  0.54820939  1.66979647  max       4         16
#> 18    M7    217  0.35249067  1.07365485  max       3          9
#> 19    M8     60  0.34274227  1.04396212  max       1          2
#> 20    M9    103  0.33509074  1.02065624  max       4         16
#>    p.mean.nbrs.dist p.mean.hopk.dist deg p.deg
#> 1             0.788            0.912   4 0.956
#> 2             0.100            0.512   5 0.896
#> 3             0.168            0.092   6 0.684
#> 4             0.696            0.364   7 0.504
#> 5             0.780            0.964   5 0.896
#> 6             0.016            0.380   5 0.896
#> 7             0.012            0.632   5 0.896
#> 8             0.928            0.916   4 0.956
#> 9             0.464            0.592   5 0.896
#> 10            0.748            0.480   6 0.684
#> 11            0.172            0.412   6 0.684
#> 12            0.328            0.208   3 0.980
#> 13            0.096            0.616   6 0.684
#> 14            1.000            1.000   3 0.980
#> 15            0.628            0.288   2 1.000
#> 16            0.580            0.524   7 0.504
#> 17            0.920            0.764   4 0.956
#> 18            0.304            0.388   6 0.684
#> 19            0.972            0.676   2 1.000
#> 20            0.548            0.472   5 0.896
#>   Assigned 9 maximum labels
#>   Assigned 11 minimum labels
#> Storing graph structure in basins object...
#> Constructing basin vertices lists...
#> Computing final overlap distance matrices...
#>
#> Step 6: Expanding basins to cover all vertices...
#>   Maxima basins: 134 -> 250 vertices covered
#>   Minima basins: 201 -> 250 vertices covered
#>
#> Refinement complete!
#> Final structure: 9 maxima and 11 minima

y.basins$summary
#>    label vertex       value   rel.value type hop.idx basin.size
#> 1     m1    245 -0.26630000 -0.81112583  min       6         25
#> 2     m2    120 -0.26183263 -0.79751862  min       4         16
#> 3     m3     92 -0.19902809 -0.60622164  min       9         64
#> 4     m4    230 -0.18650066 -0.56806424  min       4         17
#> 5     m5    208 -0.15682252 -0.47766730  min       2          6
```

```
#> 6      m6    250 -0.12346690 -0.37606906  min        2          7
#> 7      m7    101 -0.10990627 -0.33476461  min        1          6
#> 8      m8    203  0.02142402  0.06525565  min        8         40
#> 9      m9     72  0.05842157  0.17794685  min        1          6
#> 10    m10    132  0.12752894  0.38844168  min        4         22
#> 11    m11    146  0.18143515  0.55263513  min        4         15
#> 12     M1     30  1.46221755  4.45378309  max        9         63
#> 13     M2     28  1.32545340  4.03721181  max        2          8
#> 14     M3    148  0.96668007  2.94442051  max        0          1
#> 15     M4    163  0.90997849  2.77171260  max        2          3
#> 16     M5    153  0.85694374  2.61017354  max        6         25
#> 17     M6    190  0.54820939  1.66979647  max        4         16
#> 18     M7    217  0.35249067  1.07365485  max        3          9
#> 19     M8     60  0.34274227  1.04396212  max        1          2
#> 20     M9    103  0.33509074  1.02065624  max        4         16
#>    p.mean.nbrs.dist p.mean.hopk.dist deg p.deg
#> 1             0.788            0.912   4 0.956
#> 2             0.100            0.512   5 0.896
#> 3             0.168            0.092   6 0.684
#> 4             0.696            0.364   7 0.504
#> 5             0.780            0.964   5 0.896
#> 6             0.016            0.380   5 0.896
#> 7             0.012            0.632   5 0.896
#> 8             0.928            0.916   4 0.956
#> 9             0.464            0.592   5 0.896
#> 10            0.748            0.480   6 0.684
#> 11            0.172            0.412   6 0.684
#> 12            0.328            0.208   3 0.980
#> 13            0.096            0.616   6 0.684
#> 14            1.000            1.000   3 0.980
#> 15            0.628            0.288   2 1.000
#> 16            0.580            0.524   7 0.504
#> 17            0.920            0.764   4 0.956
#> 18            0.304            0.388   6 0.684
#> 19            0.972            0.676   2 1.000
#> 20            0.548            0.472   5 0.896
```

```
true.max.label <- ifelse(comp1.use >= comp2.use, "M1.true", "M2.true")

est.max.label <- assign_from_vertices_list(
  y.basins$expanded.max.vertices.list %||% y.basins$max.vertices.list,
  n = length(y.hat)
)

idx.ok <- !is.na(est.max.label)
table(True = true.max.label[idx.ok], Estimated = est.max.label[idx.ok])
#>          Estimated
#> True      M1 M2 M3 M4 M5 M6 M7 M8 M9
#>   M1.true 90  9  0  0  0  0  0  2  0
#>   M2.true 20  0  1 11 41 29 27  0 20
```

Basin-Wise Pearson and Spearman Analyses

```r
max.assign <- assign_from_vertices_list(
  y.basins$expanded.max.vertices.list %||% y.basins$max.vertices.list,
  n = length(y.hat)
)
min.assign <- assign_from_vertices_list(
  y.basins$expanded.min.vertices.list %||% y.basins$min.vertices.list,
  n = length(y.hat)
)
cell.assign <- ifelse(is.na(max.assign) | is.na(min.assign),
                      NA_character_,
                      paste(max.assign, min.assign, sep = "|"))

cor_by_group <- function(yv, zv, g, feature, min.size = 20L) {
  ii <- split(seq_along(g), g)
  ii <- ii[!is.na(names(ii))]
  ii <- ii[!(names(ii) %in% "NA")]
  if (length(ii) == 0L) return(data.frame())
  out <- lapply(names(ii), function(gr) {
    idx <- ii[[gr]]
    if (length(idx) < min.size) return(NULL)
    data.frame(
      feature = feature,
      group = gr,
      n = length(idx),
      pearson = suppressWarnings(cor(yv[idx], zv[idx], method = "pearson")),
      spearman = suppressWarnings(cor(yv[idx], zv[idx], method = "spearman"))
    )
  })
  out <- do.call(rbind, out)
  if (is.null(out)) data.frame() else out
}

basin.cor <- do.call(
  rbind,
  lapply(seq_len(ncol(Z.sm)), function(j) {
    cor_by_group(y.hat, Z.sm[, j], max.assign, colnames(Z.sm)[j], min.size = 20L)
  })
)

if (nrow(basin.cor) > 0) {
  head(basin.cor[order(-abs(basin.cor$spearman)), ], 12)
} else {
  basin.cor
}
#>            feature group   n    pearson   spearman
#> 13     z_peak2_arc    M6  29  0.9171324  0.8926108
#> 18 z_peak_contrast    M6  29 -0.8740734 -0.8472906
#> 3         z_global    M6  29  0.8860806  0.8394089
#> 6      z_peak1_arc    M1 110  0.9632611  0.8263869
#> 1         z_global    M1 110  0.9692105  0.8241688
#> 16 z_peak_contrast    M1 110  0.9658861  0.8131776
#> 2         z_global    M5  41  0.8799458  0.8090592
```

```
#> 17 z_peak_contrast    M5  41 -0.8541682 -0.8083624
#> 12     z_peak2_arc    M5  41  0.8385678  0.8024390
#> 23        z_noise1    M6  29  0.8270852  0.7753695
#> 35        z_noise3    M9  20  0.5967139  0.7127820
#> 33        z_noise3    M6  29 -0.4186189 -0.6842365
```

Local Correlation with `lcor()`

```
adj.list <- fit.seed$graph$adj.list
edge.length.list <- fit.seed$graph$edge.length.list

lcor.derivative <- as_lcor_matrix(
  lcor(adj.list, edge.length.list, y.hat, Z, type = "derivative")
)
lcor.unit <- as_lcor_matrix(
  lcor(adj.list, edge.length.list, y.hat, Z, type = "unit")
)
lcor.sign <- as_lcor_matrix(
  lcor(adj.list, edge.length.list, y.hat, Z, type = "sign")
)

summarize_lcor <- function(mat, type) {
  data.frame(
    feature = colnames(mat),
    type = type,
    mean.lcor = colMeans(mat),
    mean.abs.lcor = colMeans(abs(mat)),
    row.names = NULL
  )
}

lcor.summary <- rbind(
  summarize_lcor(lcor.derivative, "derivative"),
  summarize_lcor(lcor.unit, "unit"),
  summarize_lcor(lcor.sign, "sign")
)

lcor.summary[order(lcor.summary$type, -lcor.summary$mean.abs.lcor), ]
#>            feature       type    mean.lcor mean.abs.lcor
#> 1         z_global derivative  0.223553319     0.5408227
#> 4  z_peak_contrast derivative -0.018374965     0.4996884
#> 2      z_peak1_arc derivative  0.172994951     0.4936340
#> 3      z_peak2_arc derivative  0.136151960     0.4776319
#> 7         z_noise3 derivative -0.005235465     0.4560861
#> 6         z_noise2 derivative  0.001632273     0.4532052
#> 5         z_noise1 derivative  0.014150734     0.4502928
#> 15        z_global       sign  0.247438268     0.4879175
#> 18 z_peak_contrast       sign  0.017436406     0.4683590
#> 16     z_peak1_arc       sign  0.183656190     0.4448450
#> 17     z_peak2_arc       sign  0.111610027     0.3974714
#> 19        z_noise1       sign  0.046876824     0.3869158
#> 21        z_noise3       sign -0.008896981     0.3846748
#> 20        z_noise2       sign -0.012852809     0.3717648
```

```
#> 8          z_global       unit  0.247438268      0.4879175
#> 11 z_peak_contrast       unit  0.017436406      0.4683590
#> 9       z_peak1_arc       unit  0.183656190      0.4448450
#> 10      z_peak2_arc       unit  0.111610027      0.3974714
#> 12         z_noise1       unit  0.046876824      0.3869158
#> 14         z_noise3       unit -0.008896981      0.3846748
#> 13         z_noise2       unit -0.012852809      0.3717648
```

Permutation Tests for `lcor`

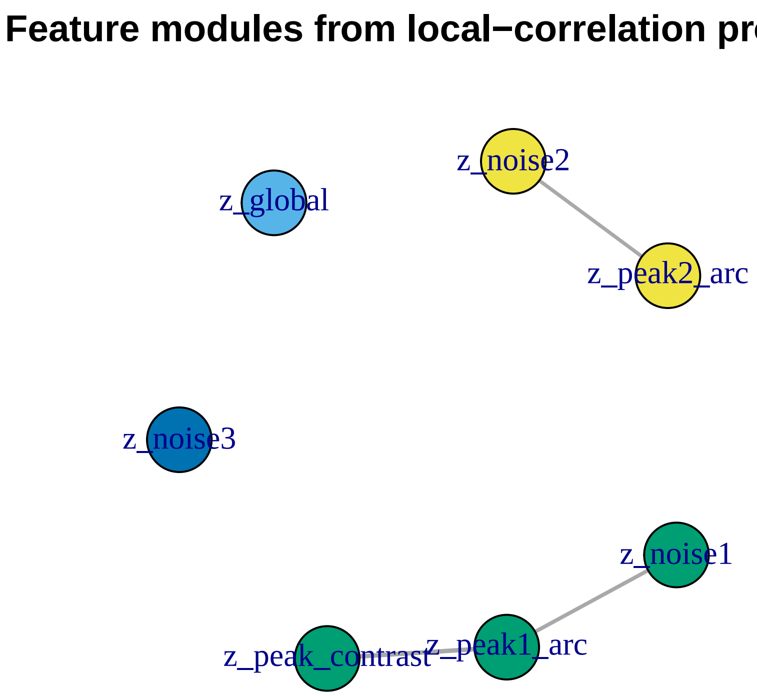
`lcor()` computes local-correlation coefficients but does not directly return permutation p-values. We add a permutation layer by shuffling vertex labels of `Z` and recomputing the chosen `lcor` summary statistic.

```
perm.lcor <- perm_test_lcor(
  adj.list = adj.list,
  edge.length.list = edge.length.list,
  y = y.hat,
  Z = Z,
  type = "derivative",
  n.perm = 200L,
  seed = 2026L
)

perm.lcor$table[order(perm.lcor$table$q.value), ]
#>           feature  stat.obs     p.value    q.value
#> 1        z_global 0.5408227 0.004975124 0.03482587
#> 2     z_peak1_arc 0.4936340 0.009950249 0.03482587
#> 4 z_peak_contrast 0.4996884 0.019900498 0.04643449
#> 3     z_peak2_arc 0.4776319 0.124378109 0.21766169
#> 5        z_noise1 0.4502928 0.835820896 0.83582090
#> 6        z_noise2 0.4532052 0.830845771 0.83582090
#> 7        z_noise3 0.4560861 0.771144279 0.83582090
```

Build an lcor-Based Feature Module Graph

```
sim <- stats::cor(lcor.derivative, use = "pairwise.complete.obs")
adj.sim <- abs(sim)
diag(adj.sim) <- 0

threshold <- 0.20
adj.sim[adj.sim < threshold] <- 0

if (sum(adj.sim > 0) == 0) {
  message("No edges above threshold. Lower `threshold` to make the module graph denser.")
} else {
  g.mod <- igraph::graph_from_adjacency_matrix(
    adj.sim,
    mode = "undirected",
    weighted = TRUE,
    diag = FALSE
  )

  modules <- igraph::cluster_louvain(g.mod, weights = igraph::E(g.mod)$weight)
```

```
  igraph::plot.igraph(
    g.mod,
    vertex.label = igraph::V(g.mod)$name,
    vertex.size = 26,
    vertex.color = as.integer(igraph::membership(modules)) + 1,
    edge.width = 1 + 4 * igraph::E(g.mod)$weight,
    main = "Feature modules from local-correlation profiles"
  )

  data.frame(
    feature = igraph::V(g.mod)$name,
    module = as.integer(igraph::membership(modules)),
    row.names = NULL
  )
}
```




```
#>           feature module
#> 1        z_global      1
#> 2     z_peak1_arc      2
#> 3     z_peak2_arc      3
#> 4 z_peak_contrast      2
#> 5        z_noise1      2
#> 6        z_noise2      3
#> 7        z_noise3      4
```

Optional grip 3D Layout

```
if (requireNamespace("grip", quietly = TRUE) &&
    requireNamespace("rgl", quietly = TRUE)) {
  g <- X.graphs$geom_pruned_graphs[[k.idx]]
```

```r
  X.graphs.3d <- grip::grip.layout(
    adj_list = g$adj_list,
    weight_list = g$weight_list,
    dim = 3,
    rounds = 50,
    final_rounds = 50,
    num_init = 36,
    num_nbrs = 8,
    seed = 6
  )

  plot3D.plain(X.graphs.3d, radius = 0.03)
}
```

Notes

- For publication-quality inference, increase permutation count (`n.perm`), report confidence intervals, and retain `q.value` (FDR-controlled) summaries.
- For larger datasets, use `n.cores > 1` and consider chunk-level caching.
- If basin refinement is unstable for a specific dataset, start with relaxed filtering and tighten thresholds gradually.