Noisy Circle: End-to-End gflow Workflow

Overview

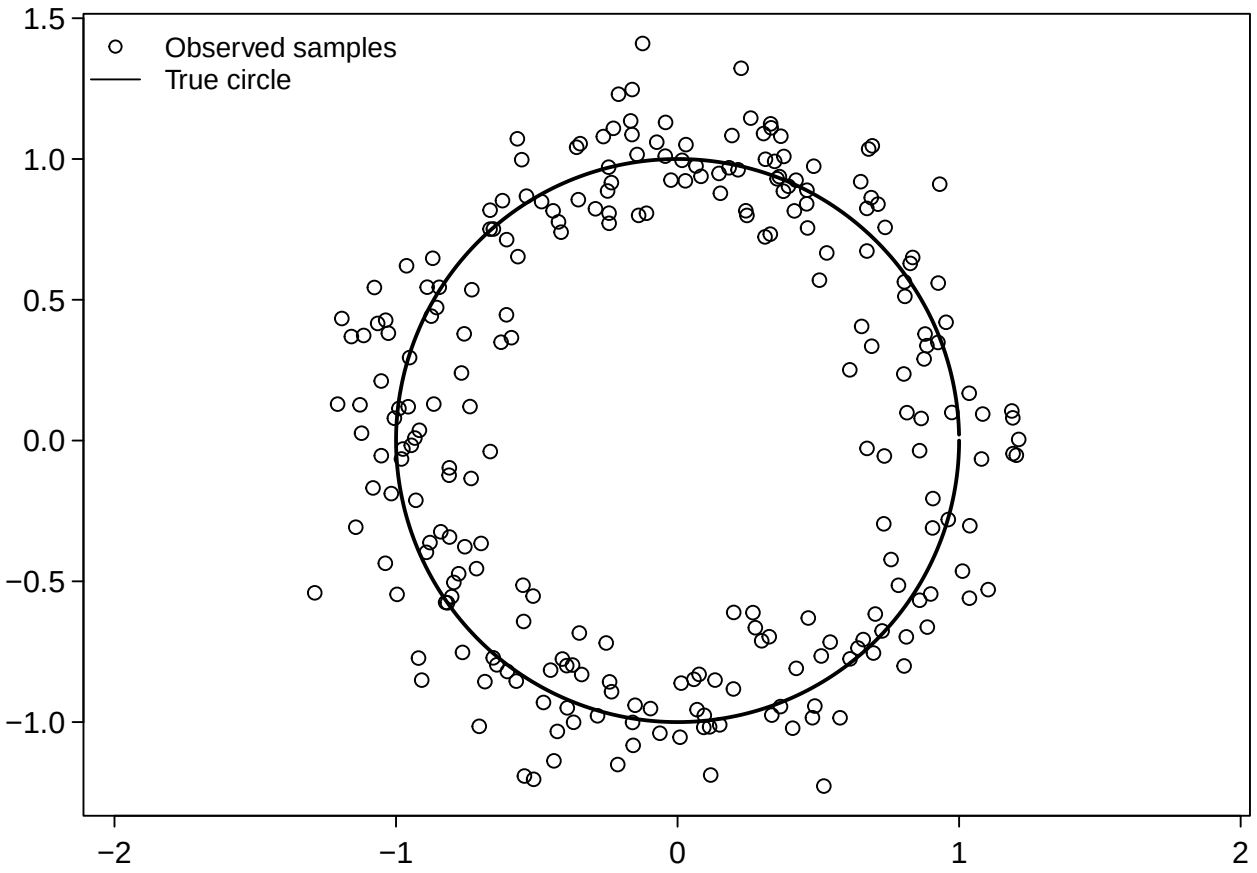
This vignette follows one complete analysis story. We start with noisy samples from a circle and a noisy response field over that circle. We then construct graph scales, denoise geometry, smooth responses and features, recover local extrema/basins, and finally test local associations with permutation-based inference.

1) Simulate Geometry 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)
}

# Response: a mixture of two circular Gaussian peaks (one larger, one smaller)
mu1 <- 0.80
mu2 <- 3.95
sd1 <- 0.30
sd2 <- 0.48
amp1 <- 1.40
amp2 <- 0.75

comp1 <- circular.synthetic.mixture.of.gaussians(
  x = theta.obs,
  x.knot = mu1,
  y.knot = amp1,
  sd.knot = sd1
)
comp2 <- circular.synthetic.mixture.of.gaussians(
  x = theta.obs,
  x.knot = mu2,
  y.knot = amp2,
```

```
  sd.knot = sd2
)
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")])
```

```
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)
```



```
par(op)
```

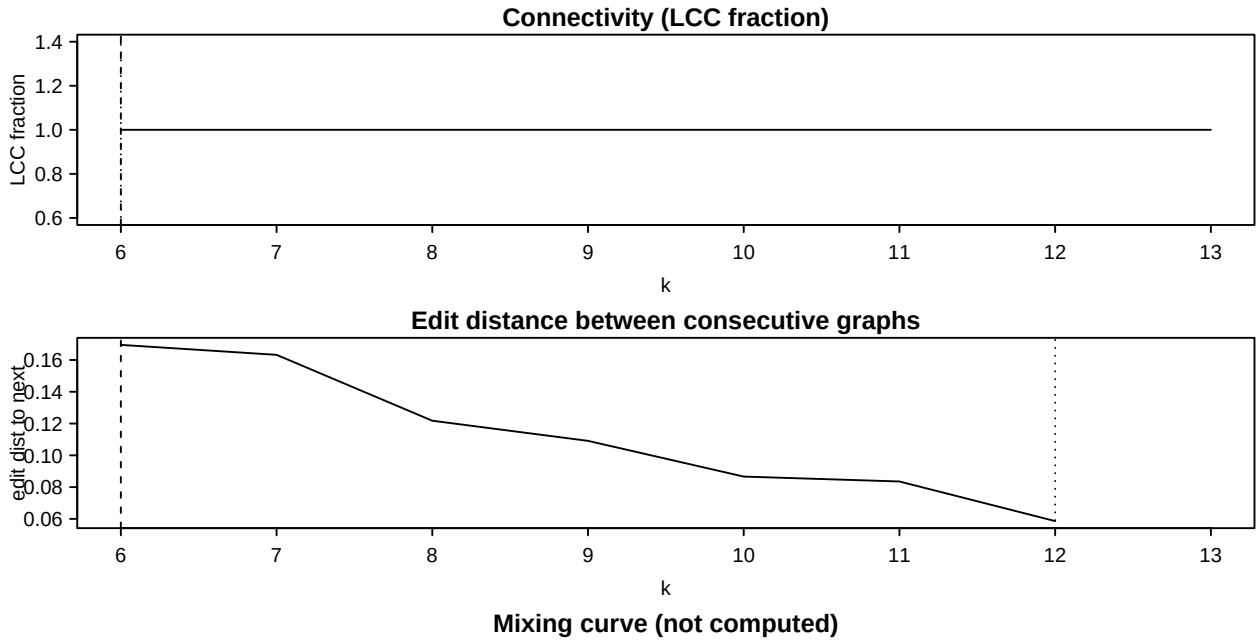
2) Build Graphs Across k

```
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
#> 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] 0.367s
#> Graph processing completed (0.367)
#> [compute_knn] 0.001s
#> [graph_build] 0.013s
#> [geometric_prune] 0.354s
#> [isize_prune] skipped (with.isize.pruning=FALSE)
#> Creating return list objects ... DONE (0.000)
#> Total elapsed time (0.368)
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
```

3) Select Graph Scale

```
X.graphs2 <- build.iknn.graphs.and.selectk(
  X,
```

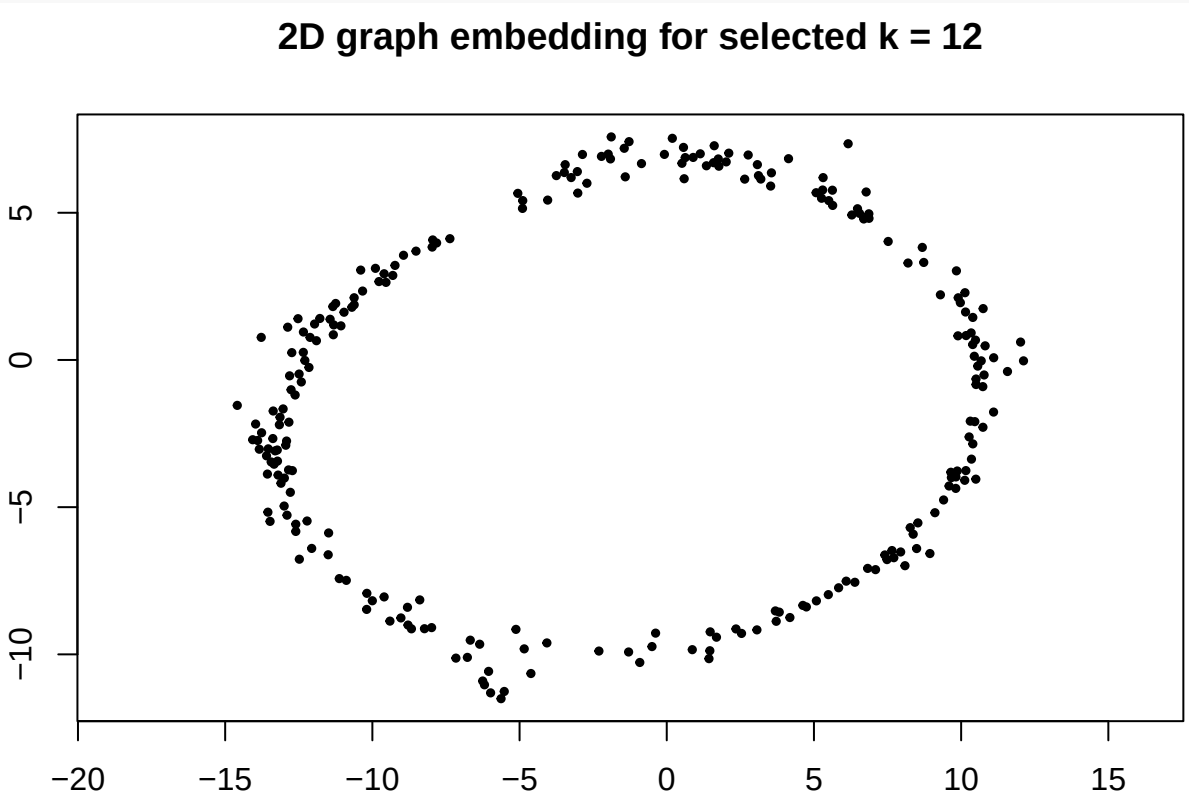
```
  kmin = k.min,
  kmax = k.max,
  method = "edit",
  n.cores = 1L,
  verbose = TRUE
)
#>  Using 1 OpenMP thread
#> Processing k values from 5 to 12 for 250 vertices
#> Requested parallel.mode: auto
#> Parallel mode: serial
#> Starting graph processing
#> [compute_knn] running...[compute_knn] 0.000s
#> [graph_build/geometric_prune/isize_prune] running...[graph_build/geometric_prune/isize_prune] 0.368s
#> Graph processing completed (0.368)
#> [compute_knn] 0.000s
#> [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 (0.368)

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



4) Visualize the Selected Graph

```
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 <- X.graphs2$k.opt.edit
if (is.null(sel.k) || is.na(sel.k)) {
  sel.k <- k.values[1]
}
sel.k <- as.integer(sel.k)
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 = "")
```



5) Denoise Geometry with `data.smoother()`

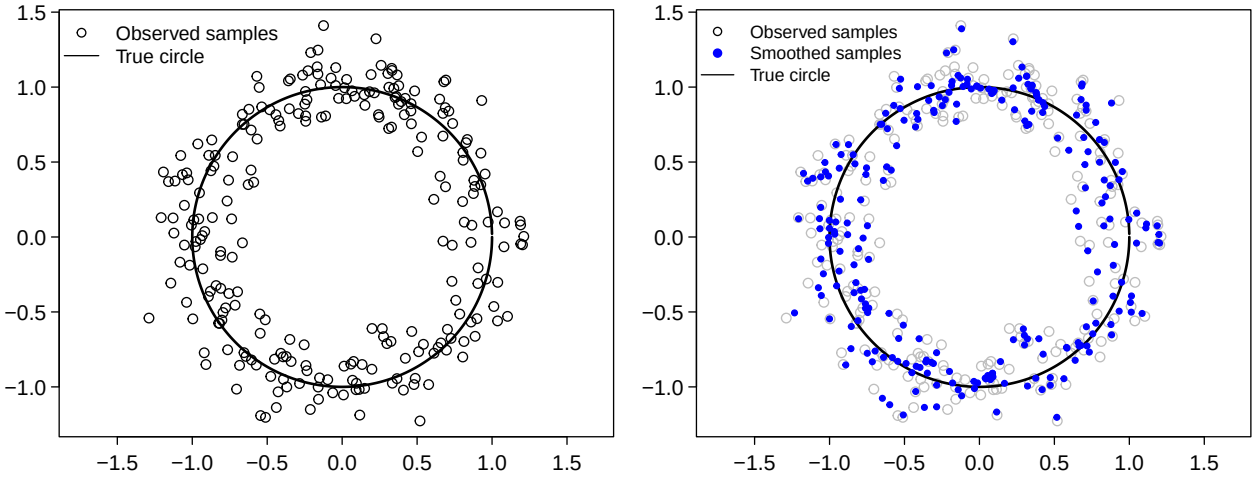
```
X.denoise.res <- data.smoother(
  X,
  kmin = k.min,
```

```
  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] 0.000s
#> [graph_build/geometric_prune/isize_prune] running...[graph_build/geometric_prune/isize_prune] 0.370s
#> Graph processing completed (0.370)
#> [compute_knn] 0.000s
#> [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 (0.370)
#> 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)
```

```r
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
```

6) Smooth the Response on the Selected Graph

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

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

y.fit <- refit.rdgraph.regression(
  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 Surface View

```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"
  )
}
```

```
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
  )
}
```

7) Build and Smooth Feature Functions

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

```
set.seed(1002)

window.1 <- as.numeric(comp1.use >= quantile(comp1.use, probs = 0.60))
window.2 <- as.numeric(comp2.use >= quantile(comp2.use, probs = 0.60))

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)
```

```
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=0.9401, Q3=0.9621, max=0.9775
#>   RMSE: min=0.1499, Q1=0.1941, med=0.2442, Q3=0.7879, max=0.8051
```

8) Recover Extrema and Basins

```
y.gfc <- compute.gfc(
  adj.list = fit.seed$graph$adj.list,
  edge.length.list = fit.seed$graph$edge.length.list,
  fitted.values = y.hat,
  edge.length.quantile.thld = 0.9,
  min.rel.value.max = 1.05,
  max.rel.value.min = 0.95,
  max.overlap.threshold = 0.20,
  min.overlap.threshold = 0.20,
  p.mean.nbrs.dist.threshold = 0.95,
  p.mean.hopk.dist.threshold = 0.95,
  p.deg.threshold = 0.95,
  min.basin.size = 5L,
  expand.basins = TRUE,
  apply.geometric.filter = FALSE,
  with.trajectories = TRUE,
  verbose = TRUE
)
#> GFC computation: 250 vertices, median = 0.2195
#> Step 1: Computing initial basins of attraction...
#>   Found 17 maxima and 22 minima
#> Step 2: Filtering by relative values (max >= 1.05, min <= 0.95)...
#>   Retained 14 maxima and 16 minima
#> Step 3: Clustering maxima (overlap threshold = 0.20)...
#>   Result: 13 maxima after merging
#> Step 4: Clustering minima (overlap threshold = 0.20)...
#>   Result: 11 minima after merging
#> Computing joined trajectories (max chain depth = 5)...
#>   Computing representative trajectories (one per cell)...
#>   Non-spurious: 13 maxima, 11 minima
#>   Processing max 1/13 (vertex 217): 3 terminal sets, 1 valid target minima
#>   Processing max 2/13 (vertex 163): 1 terminal sets, 0 valid target minima
#>     No valid target minima in basin, skipping
#>   Processing max 3/13 (vertex 153): 4 terminal sets, 2 valid target minima
#>   Processing max 4/13 (vertex 148): 0 terminal sets, 0 valid target minima
#>     No valid target minima in basin, skipping
#>   Processing max 5/13 (vertex 129): 1 terminal sets, 1 valid target minima
#>   Processing max 6/13 (vertex 103): 2 terminal sets, 1 valid target minima
#>   Processing max 7/13 (vertex 77): 5 terminal sets, 2 valid target minima
#>   Processing max 8/13 (vertex 30): 8 terminal sets, 2 valid target minima
#>     Warning: spurious min 33 could not reach any valid min via chain (depth limit = 5), trying geode
#>     Warning: spurious min 6 could not reach any valid min via chain (depth limit = 5), trying geodes
#>   Processing max 9/13 (vertex 28): 1 terminal sets, 0 valid target minima
```

```
#>     No valid target minima in basin, skipping
#>   Processing max 10/13 (vertex 216): 4 terminal sets, 3 valid target minima
#>   Processing max 11/13 (vertex 190): 1 terminal sets, 0 valid target minima
#>     No valid target minima in basin, skipping
#>   Processing max 12/13 (vertex 60): 1 terminal sets, 0 valid target minima
#>     No valid target minima in basin, skipping
#>   Processing max 13/13 (vertex 8): 2 terminal sets, 1 valid target minima
#>   Trajectory joining complete: 13 cells
#>     Direct: 13, Chain-extended: 3, Geodesic-extended: 2
#>     Geodesic-fallback: 1, Unreachable: 0
#>   Created 13 joined trajectories in 13 cells
#> Step 6: Expanding basins to cover all vertices...
#>   Max coverage: 250/250, Min coverage: 250/250
#> GFC computation complete: 13 maxima, 11 minima
```

```
y.gfc$summary
```

```
#>    label vertex       value   rel.value type hop.idx basin.size
#> 1     m1    245 -0.26630000 -1.21341097  min       7         25
#> 2     m2    120 -0.26183263 -1.19305514  min       4         16
#> 3     m3     92 -0.19902809 -0.90688271  min      12         62
#> 4     m4    230 -0.18650066 -0.84980080  min       4         17
#> 5     m5    208 -0.15682252 -0.71457068  min       2          6
#> 6     m6    250 -0.12346690 -0.56258388  min       2          7
#> 7     m7    101 -0.10990627 -0.50079412  min       1          6
#> 8     m8    203  0.02142402  0.09761977  min       9         40
#> 9     m9     72  0.05842157  0.26620119  min       1          6
#> 10   m10    132  0.12752894  0.58109282  min       4         22
#> 11   m11    146  0.18143515  0.82671949  min       4         15
#> 12    M1     30  1.46221755  6.66267679  max       9         63
#> 13    M2     28  1.32545340  6.03950324  max       2          8
#> 14    M3    148  0.96668007  4.40473229  max       0          1
#> 15    M4    163  0.90997849  4.14636834  max       2          3
#> 16    M5    153  0.85694374  3.90471254  max       6         25
#> 17    M6    190  0.54820939  2.49794701  max       4         16
#> 18    M7    217  0.35249067  1.60614362  max       3          9
#> 19    M8     60  0.34274227  1.56172451  max       1          2
#> 20    M9    103  0.33509074  1.52685986  max       4         16
#> 21   M10      8  0.29976846  1.36591191  max       4          9
#> 22   M11     77  0.29952927  1.36482204  max       4         26
#> 23   M12    129  0.27586066  1.25697466  max       1          3
#> 24   M13    216  0.25715592  1.17174547  max       5         20
#>    p.mean.nbrs.dist p.mean.hopk.dist deg p.deg
#> 1             0.784            0.908   4 0.044
#> 2             0.096            0.044   5 0.104
#> 3             0.164            0.344   6 0.316
#> 4             0.692            0.476   7 0.496
#> 5             0.776            0.912   5 0.104
#> 6             0.012            0.048   5 0.104
#> 7             0.008            0.424   5 0.104
#> 8             0.924            0.956   4 0.044
#> 9             0.460            0.700   5 0.104
#> 10            0.744            0.656   6 0.316
#> 11            0.168            0.524   6 0.316
```

```
#> 12            0.324            0.792   3 0.020
#> 13            0.092            0.296   6 0.316
#> 14            0.996            0.996   3 0.020
#> 15            0.624            0.784   2 0.000
#> 16            0.576            0.548   7 0.496
#> 17            0.916            0.896   4 0.044
#> 18            0.300            0.040   6 0.316
#> 19            0.968            0.920   2 0.000
#> 20            0.544            0.308   5 0.104
#> 21            0.564            0.392   3 0.020
#> 22            0.404            0.560   5 0.104
#> 23            0.240            0.264   2 0.000
#> 24            0.440            0.440   6 0.316
```

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

est.max.label <- ifelse(
  is.na(y.gfc$expanded.max.assignment),
  NA_character_,
  paste0("M", y.gfc$expanded.max.assignment)
)

idx.ok <- !is.na(est.max.label)
table(True = true.max.label[idx.ok], Estimated = est.max.label[idx.ok])
#>          Estimated
#> True      M1 M10 M11 M12 M13 M2 M3 M4 M5 M6 M7 M8 M9
#>   M1.true 66   9  16   0   0  9  0  0  0  0  0  1  0
#>   M2.true  5   0   9   3  28  0  1 11 38 31  3  0 20
```

9) Basin-Wise Pearson and Spearman Checks

```
max.assign <- ifelse(
  is.na(y.gfc$expanded.max.assignment),
  NA_character_,
  paste0("M", y.gfc$expanded.max.assignment)
)
min.assign <- ifelse(
  is.na(y.gfc$expanded.min.assignment),
  NA_character_,
  paste0("m", y.gfc$expanded.min.assignment)
)
cell.assign <- ifelse(is.na(max.assign) | is.na(min.assign),
                      NA_character_,
                      paste(max.assign, min.assign, sep = "|"))

group.levels <- setdiff(sort(unique(max.assign)), NA_character_)
basin.cor <- data.frame()

for (j in seq_len(ncol(Z.sm))) {
  for (gr in group.levels) {
    idx <- which(max.assign == gr)
    if (length(idx) < 20L) next
    basin.cor <- rbind(
```

```
      basin.cor,
      data.frame(
        feature = colnames(Z.sm)[j],
        group = gr,
        n = length(idx),
        pearson = suppressWarnings(cor(y.hat[idx], Z.sm[idx, j], method = "pearson")),
        spearman = suppressWarnings(cor(y.hat[idx], Z.sm[idx, j], method = "spearman"))
      )
    )
  }
}

if (nrow(basin.cor) > 0) {
  head(basin.cor[order(-abs(basin.cor$spearman)), ], 12)
} else {
  basin.cor
}
#>            feature group  n    pearson   spearman
#> 1         z_global    M1 71  0.9872421  0.9771630
#> 7      z_peak1_arc    M1 71  0.9793675  0.9627096
#> 19 z_peak_contrast    M1 71  0.9776120  0.9471160
#> 17     z_peak2_arc    M6 31  0.9283557  0.9116935
#> 23 z_peak_contrast    M6 31 -0.8868152 -0.8737903
#> 5         z_global    M6 31  0.8966431  0.8540323
#> 4         z_global    M5 38  0.8716238  0.8146406
#> 16     z_peak2_arc    M5 38  0.8293874  0.8131087
#> 22 z_peak_contrast    M5 38 -0.8418890 -0.8023854
#> 14     z_peak2_arc   M11 25 -0.8123586 -0.7876923
#> 42        z_noise3    M9 20  0.5967139  0.7127820
#> 24 z_peak_contrast    M9 20 -0.6391321 -0.6511278
```

10) Local Correlation with `lcor()`

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

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

lcor.summary <- rbind(
  data.frame(
    feature = colnames(lcor.derivative),
    type = "derivative",
    mean.lcor = colMeans(lcor.derivative),
    mean.abs.lcor = colMeans(abs(lcor.derivative)),
    row.names = NULL
  ),
  data.frame(
    feature = colnames(lcor.unit),
    type = "unit",
    mean.lcor = colMeans(lcor.unit),
    mean.abs.lcor = colMeans(abs(lcor.unit)),
```

```
    row.names = NULL
  ),
  data.frame(
    feature = colnames(lcor.sign),
    type = "sign",
    mean.lcor = colMeans(lcor.sign),
    mean.abs.lcor = colMeans(abs(lcor.sign)),
    row.names = NULL
  )
)

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  0.172828467     0.4935075
#> 3      z_peak2_arc derivative  0.138613912     0.4793121
#> 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  0.183499414     0.4447169
#> 17     z_peak2_arc       sign  0.110216441     0.3996195
#> 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  0.183499414     0.4447169
#> 10     z_peak2_arc       unit  0.110216441     0.3996195
#> 12        z_noise1       unit  0.046876824     0.3869158
#> 14        z_noise3       unit -0.008896981     0.3846748
#> 13        z_noise2       unit -0.012852809     0.3717648
```

11) Permutation Inference for `lcor`

We assess feature significance by permuting feature rows (`permute = "z"`), recomputing local correlations, and comparing observed statistics to this null.

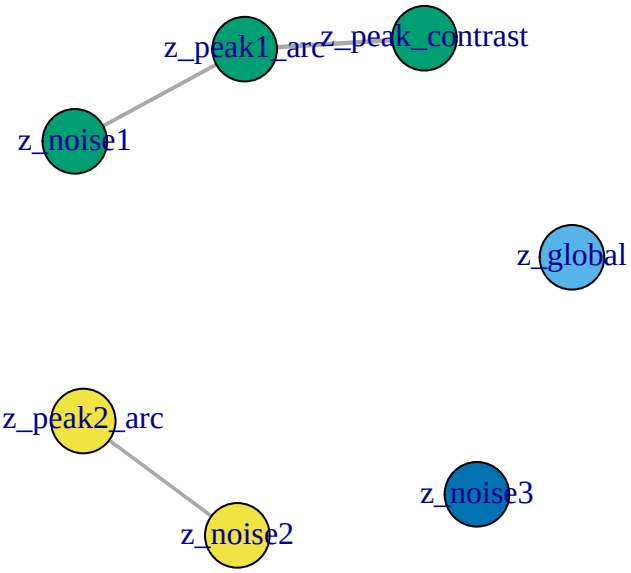
```
perm.lcor <- permutation.test.lcor(
  adj.list = adj.list,
  weight.list = edge.length.list,
  y = y.hat,
  z = Z,
  type = "derivative",
  statistic = "mean.abs",
  permute = "z",
  n.perm = 200L,
  seed = 2026L
)

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 0.4935075 0.009950249 0.03482587
#> 4 z_peak_contrast 0.4996884 0.019900498 0.04643449
#> 3     z_peak2_arc 0.4793121 0.119402985 0.20895522
#> 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
```

12) 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 modules from local−correlation profiles




```
#>           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]]

  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.