Package: CellOracleR (via r-universe)

May 25, 2026

Type Package

Title In Silico Gene Perturbation Analysis with Single-Cell Data

Version 0.1.0

Description An R implementation of the CellOracle framework for in silico gene perturbation analysis and gene regulatory network (GRN) inference from single-cell RNA-seq data. Predicts cell state transitions in response to transcription factor perturbations by combining GRN models with single-cell expression data. Key features include motif analysis for base GRN construction, cluster-specific GRN inference using regularized regression, perturbation simulation with signal propagation, and comprehensive visualization of predicted cell fate changes. Based on the methodology described in Kamimoto et al. (2023) <doi:10.15252/msb.202211547>.

URL https://github.com/Zaoqu-Liu/CellOracleR

BugReports https://github.com/Zaoqu-Liu/CellOracleR/issues

Encoding UTF-8

Roxygen list(markdown = TRUE)

License Apache License (>= 2) | file LICENSE

RoxygenNote 7.3.3

Depends R (>= 4.0.0)

Imports R6, Rcpp (>= 1.0.0), Matrix, methods, stats, utils, grDevices, Seurat (>= 4.0.0), SeuratObject, glmnet, igraph, ggplot2, scales, future, future.apply, progressr, data.table, FNN, rlang

Suggests qs, testthat (>= 3.0.0), knitr, rmarkdown, BiocManager, TFBSTools, motifmatchr, BSgenome, GenomicRanges, Biostrings, GenomeInfoDb, IRanges, S4Vectors, SingleCellExperiment, SummarizedExperiment, clusterProfiler, org.Hs.eg.db, org.Mm.eg.db, JASPAR2020, viridis, patchwork, cowplot, ggraph, irlba, arrow, reshape2, e1071, rstanarm, networkD3, htmlwidgets

LinkingTo Rcpp, RcppArmadillo

Config/testthat/edition 3

VignetteBuilder knitr

SystemRequirements C++17

Config/pak/sysreqs cmake libglpk-dev make libicu-dev libpng-dev libuv1-dev libxml2-dev libssl-dev python3 zlib1g-dev

Repository https://zaoqu-liu.r-universe.dev

Date/Publication 2026-01-24 20:11:44 UTC

RemoteUrl https://github.com/Zaoqu-Liu/CellOracleR

RemoteRef main

RemoteSha e60a1b9d84a1b89c2e8f36efc8ea16065f673b57

Contents

`annotate_peaks` *Annotate peaks with nearby genes*

Description

Associates peaks with nearby genes based on TSS proximity.

Usage

```
annotate_peaks(peaks_df, ref_genome, upstream = 2000, downstream = 2000)
```

Arguments

`peaks_df`	Data frame with chr, start, end columns
`ref_genome`	Reference genome
`upstream`	Distance upstream of TSS to consider
`downstream`	Distance downstream of TSS to consider

Value

Data frame with peak-gene associations

`build_coef_matrix_cpp` *Build coefficient matrix from gene-wise results*

Description

Efficiently assembles the full coefficient matrix from individual gene regression results stored in a list.

Usage

```
build_coef_matrix_cpp(coef_list, gene_names, target_genes)
```

Arguments

`coef_list`	List of coefficient vectors (one per target gene)
`gene_names`	Names of all genes
`target_genes`	Names of genes with fitted models

Value

Full coefficient matrix (genes x genes)

`build_knn_graph_cpp` *Build sparse connectivity matrix from KNN indices*

Description

Creates a sparse adjacency matrix from KNN indices where entry (i,j) = 1 if j is a neighbor of i.

Usage

```
build_knn_graph_cpp(knn_idx, n_cells, mutual = FALSE)
```

Arguments

`knn_idx`	KNN index matrix (cells x k), 0-indexed
`n_cells`	Total number of cells
`mutual`	Whether to require mutual neighbors

Value

Sparse connectivity matrix (cells x cells)

`calculate_embedding_shift_cpp`
Calculate embedding shift from transition probability

Description

Computes the expected embedding shift (velocity) based on transition probabilities and neighbor positions.

Usage

```
calculate_embedding_shift_cpp(embedding, trans_prob, knn_adj)
```

Arguments

`embedding`	Embedding coordinates (cells x 2)
`trans_prob`	Transition probability matrix (cells x cells)
`knn_adj`	KNN adjacency matrix

Value

Delta embedding matrix (cells x 2)

`calculate_grid_arrows_cpp` *Grid arrow computation*

Description

Computes flow vectors on a regular grid using weighted averaging from nearby cells with Gaussian kernel.

Usage

```
calculate_grid_arrows_cpp(
  embedding,
  delta_embedding,
  grid_points,
  knn_idx,
  knn_dist,
  smooth = 0.5
)
```

Arguments

`embedding`	Cell embedding coordinates (cells x 2)
`delta_embedding`	Cell velocity vectors (cells x 2)
`grid_points`	Grid coordinates (grid_points x 2)
`knn_idx`	KNN indices for each grid point (grid_points x k)
`knn_dist`	KNN distances (grid_points x k)
`smooth`	Smoothing parameter (multiplied by grid step)

Value

Grid flow vectors (grid_points x 2)

`calculate_relative_ratio_cpp` *Calculate relative ratio for OOD detection*

Description

Calculates the relative position of simulated values within the original expression range for out-of-distribution detection.

Usage

```
calculate_relative_ratio_cpp(simulated, reference)
```

Arguments

simulated	Simulated expression matrix
reference	Reference expression matrix

Value

Matrix of relative ratios (0-1 = within range)

---

center_cols_cpp *Center matrix columns*

Description

Subtracts column means from each column.

Usage

```
center_cols_cpp(X)
```

Arguments

X	Input matrix

Value

Centered matrix

---

clean_transition_prob_cpp *Handle NA and zero-sum rows in transition matrix*

Description

Cleans transition probability matrix by replacing NaN values with 0 and assigning self-transition probability of 1 for rows that sum to 0.

Usage

```
clean_transition_prob_cpp(trans_prob)
```

Arguments

trans_prob	Transition probability matrix

Value

Cleaned transition matrix

---

clip_to_range_cpp *Clip delta_X to distribution range*

Description

Clips simulated gene expression values to stay within the original expression distribution range to avoid out-of-distribution predictions.

Usage

```
clip_to_range_cpp(simulated, original)
```

Arguments

simulated	Simulated expression matrix (cells x genes)
original	Original expression matrix (cells x genes)

Value

Clipped simulated matrix

---

col_delta_cor_full_cpp *Compute full delta correlation matrix*

Description

Computes correlation between expression shifts for all cell pairs. More expensive than partial version but useful for validation.

Usage

```
col_delta_cor_full_cpp(X, delta_X)
```

Arguments

X	Expression matrix (cells x genes)
delta_X	Simulated expression shift (cells x genes)

Value

Full correlation matrix (cells x cells)

`col_delta_cor_partial_cpp` *Compute column-wise delta correlation (partial neighbors)*

Description

Computes correlation between expression (X) and simulated shift (delta_X) for each cell with its neighbors. This is the core computation for transition probability estimation.

Usage

```
col_delta_cor_partial_cpp(X, delta_X, neigh_idx)
```

Arguments

`X`	Expression matrix (cells x genes)
`delta_X`	Simulated expression shift (cells x genes)
`neigh_idx`	Neighbor index matrix (cells x k), 0-indexed

Value

Correlation coefficient matrix (cells x cells)

`colSds_cpp` *Column-wise standard deviation*

Description

Computes standard deviation for each column of a matrix.

Usage

```
colSds_cpp(X)
```

Arguments

`X`	Input matrix

Value

Vector of column standard deviations

compare_networks *Compare networks between clusters*

Description

Computes overlap and differences between cluster GRNs.

Usage

```
compare_networks(links, clusters = NULL)
```

Arguments

links	Links object
clusters	Clusters to compare (default: all)

Value

List with comparison statistics

compute_embedding_knn_subset_cpp *Compute embedding KNN with subset selection*

Description

Computes KNN in embedding space, optionally restricted to a subset of cells. This is useful for Markov simulation on subsampled cells.

Usage

```
compute_embedding_knn_subset_cpp(embedding, cell_idx, k)
```

Arguments

embedding	Full embedding matrix (all cells x dims)
cell_idx	Indices of cells to use (0-indexed)
k	Number of neighbors

Value

List with knn_idx and distances

`compute_fate_probability` *Compute fate probability to terminal states*

Description

Computes the probability of reaching each terminal state from each starting state.

Usage

```
compute_fate_probability(trans_prob, terminal_states, n_steps = 500)
```

Arguments

`trans_prob`	Transition probability matrix
`terminal_states`	Indices of terminal states
`n_steps`	Number of simulation steps

Value

Matrix of fate probabilities (cells x terminal states)

`compute_pseudotime` *Compute pseudotime based on transition probability*

Description

Estimates pseudotime ordering based on the perturbation-induced transition probability structure.

Usage

```
compute_pseudotime(
  oracle,
  start_cells,
  method = c("markov", "diffusion"),
  n_steps = 500,
  n_duplicates = 100
)
```

Arguments

`oracle`	Oracle object with transition probabilities
`start_cells`	Starting cell indices (1-indexed)
`method`	Method: "markov" (simulation-based) or "diffusion"
`n_steps`	Number of steps for Markov simulation
`n_duplicates`	Duplicates per cell for Markov simulation

Value

Named vector of pseudotime values

`correlation_to_transition_prob_cpp` *Calculate transition probability from correlation*

Description

Converts correlation coefficients to transition probabilities using exponential kernel, restricted to KNN neighbors.

Usage

```
correlation_to_transition_prob_cpp(corrcoef, knn_adj, sigma_corr = 0.05)
```

Arguments

`corrcoef`	Correlation coefficient matrix
`knn_adj`	KNN adjacency matrix (sparse, 1 for neighbors)
`sigma_corr`	Correlation kernel bandwidth

Value

Transition probability matrix (rows sum to 1)

`create_base_grn` *Create base GRN from scATAC-seq data*

Description

Complete workflow to create base GRN from scATAC-seq peaks.

Usage

```
create_base_grn(
  peaks_df,
  ref_genome,
  motifs = NULL,
  upstream = 2000,
  downstream = 2000,
  fpr = 0.02,
  min_peaks = 10,
  n_cores = 1
)
```

Arguments

peaks_df	Data frame with chr, start, end columns
ref_genome	Reference genome
motifs	Motif database (NULL for JASPAR)
upstream	TSS upstream distance
downstream	TSS downstream distance
fpr	Motif scanning FPR
min_peaks	Minimum peaks for TF filtering
n_cores	Number of parallel cores

Value

TFdict suitable for Oracle$import_TF_data()

---

create_oracle *Create Oracle object from Seurat*

Description

Create Oracle object from Seurat

Usage

```
create_oracle(seurat, cluster_column, embedding_name, verbose = TRUE)
```

Arguments

seurat	Seurat object
cluster_column	Cluster column name
embedding_name	Embedding name
verbose	Print messages

Value

Oracle object

create_perturb_condition *Create perturbation condition*

Description

Helper function to create perturbation condition dictionary.

Usage

```
create_perturb_condition(seurat, genes, values = "knockout")
```

Arguments

seurat	Seurat object
genes	Gene(s) to perturb
values	Value(s) for perturbation. Can be numeric or character (method name like "knockout", "max")

Value

Named list suitable for simulate_shift

create_tfinfo *Create TFinfo object from peak data*

Description

Creates a TFinfo object from peak data frame for motif analysis.

Usage

```
create_tfinfo(peak_df, ref_genome)
```

Arguments

peak_df	Data frame with columns: chr, start, end, gene_short_name
ref_genome	Reference genome (e.g., "hg38", "mm10")

Value

TFinfo object

do_simulation_cpp *Simulate signal propagation in GRN (C++ implementation)*

Description

Performs iterative signal propagation through the gene regulatory network. For each propagation step, expression changes are computed based on the GRN coefficient matrix, while perturbed gene values are maintained.

Usage

```
do_simulation_cpp(coef_matrix, simulation_input, gem, n_propagation)
```

Arguments

coef_matrix	Coefficient matrix (genes x genes) representing GRN weights
simulation_input	Initial expression state with perturbation applied (cells x genes)
gem	Original gene expression matrix (cells x genes)
n_propagation	Number of propagation steps

Value

Simulated gene expression matrix (cells x genes)

export_network *Export network to various formats*

Description

Export network to various formats

Usage

```
export_network(
  links,
  format = c("graphml", "gml", "edge_list", "pajek"),
  file_path,
  cluster = NULL
)
```

Arguments

links	Links object
format	Output format: "graphml", "gml", "edge_list", "pajek"
file_path	Output file path
cluster	Specific cluster (NULL = all)

---

extract_expression *Extract expression matrix from Seurat object*

Description

Extract expression data from a Seurat object, compatible with both V4 and V5.

Usage

```
extract_expression(
  seurat,
  layer = c("data", "counts", "scale.data"),
  assay = "RNA",
  as_dense = FALSE
)
```

Arguments

seurat	Seurat object
layer	Layer/slot to extract: "counts", "data" (normalized), or "scale.data"
assay	Assay name (default: "RNA")
as_dense	Convert to dense matrix (default: FALSE)

Value

Expression matrix (genes x cells)

Examples

```
## Not run:
# Get normalized expression
expr <- extract_expression(seurat, layer = "data")

# Get raw counts as dense matrix
counts <- extract_expression(seurat, layer = "counts", as_dense = TRUE)

## End(Not run)
```

extract_peaks_from_signac *Process ATAC-seq peaks from Signac*

Description

Extracts peak information from a Signac/Seurat object.

Usage

```
extract_peaks_from_signac(seurat, assay = "peaks", min_cells = 10)
```

Arguments

seurat	Seurat object with ATAC assay
assay	Name of ATAC assay
min_cells	Minimum cells for peak filtering

Value

Data frame with peak coordinates

filter_links *Filter links by threshold*

Description

Filter GRN links based on coefficient thresholds.

Usage

```
filter_links(links, p = 0.001, threshold_number = 10000, min_coef = 0)
```

Arguments

links	Links object
p	Percentile threshold
threshold_number	Maximum number of links
min_coef	Minimum absolute coefficient

Value

Modified Links object

find_network_motifs *Find network motifs*

Description

Identifies common regulatory motifs (feedforward loops, feedback loops, etc.)

Usage

```
find_network_motifs(links, cluster = NULL, motif_size = 3)
```

Arguments

links	Links object
cluster	Specific cluster
motif_size	Size of motifs to find (3 or 4)

Value

Data frame with motif counts

fit_grn *Fit GRN for perturbation simulation*

Description

Higher-level function that fits GRN and stores in Oracle object.

Usage

```
fit_grn(oracle, GRN_unit = c("cluster", "whole"), alpha = 1, verbose = TRUE)
```

Arguments

oracle	Oracle object
GRN_unit	"cluster" or "whole"
alpha	Regularization strength
verbose	Print progress

Value

Modified Oracle object

fit_grn_bagging *Fit GRN with bagging (batch version)*

Description

Fit GRN using Ridge regression with bootstrap aggregation (bagging) for all target genes. This matches Python's behavior exactly.

Usage

```
fit_grn_bagging(
  gem,
  TFdict,
  alpha = 10,
  bagging_number = 20,
  verbose = TRUE,
  n_jobs = -1
)
```

Arguments

gem	Gene expression matrix (cells x genes)
TFdict	TF-target dictionary
alpha	Regularization strength
bagging_number	Number of bootstrap iterations
verbose	Print progress
n_jobs	Number of parallel jobs (-1 for all cores)

Value

List with median coefficients and all bootstrap coefficients

fit_grn_coef_matrix *Fit GRN coefficient matrix (single Ridge regression)*

Description

Fit a gene regulatory network using Ridge regression. Returns a coefficient matrix where entry (i,j) represents the regulatory effect of gene i on gene j. This matches Python's _getCoefMatrix function.

Usage

```
fit_grn_coef_matrix(gem, TFdict, alpha = 1, verbose = TRUE)
```

Arguments

gem	Gene expression matrix (cells x genes)
TFdict	Named list mapping target genes to regulator TFs
alpha	Regularization strength for Ridge regression
verbose	Print progress

Value

Coefficient matrix (genes x genes)

---

gaussian_kernel_cpp *Gaussian kernel on distance matrix*

Description

Applies Gaussian kernel to convert distances to similarities.

Usage

```
gaussian_kernel_cpp(dist, sigma)
```

Arguments

dist	Distance matrix
sigma	Kernel bandwidth

Value

Similarity matrix

---

get_bagging_ridge_coefs *Get Bagging Ridge coefficients for a single target gene*

Description

Fit Ridge regression with bootstrap aggregation (bagging) for a single target gene. EXACT port from Python's get_bagging_ridge_coefs - uses:

- bootstrap=TRUE: sample rows with replacement
- max_features=0.8: randomly select 80% of features for each estimator

Usage

```
get_bagging_ridge_coefs(
  target_gene,
  gem,
  gem_scaled,
  TFdict,
  cellstate = NULL,
  bagging_number = 1000,
  scaling = TRUE,
  alpha = 1,
  n_jobs = -1
)
```

Arguments

target_gene	Target gene name
gem	Gene expression matrix (cells x genes data.frame)
gem_scaled	Scaled gene expression matrix
TFdict	TF-target dictionary
cellstate	Optional cell state data frame
bagging_number	Number of bootstrap iterations (default 1000 like Python)
scaling	Whether to use scaled data
alpha	Ridge regularization strength
n_jobs	Number of parallel jobs (not used in single-gene function)

Value

Data frame with coefficient values for each bootstrap iteration, or 0 if not applicable

---

get_base_grn_from_cicero

Get base GRN from Cicero/ATAC data

Description

Processes Cicero coaccessibility results to create a base GRN. This connects peaks to genes based on proximity and coaccessibility.

Usage

```
get_base_grn_from_cicero(
  cicero_connections,
  peak_annotation,
  coaccess_threshold = 0.05,
  max_distance = 1e+05
)
```

Arguments

`cicero_connections`	Cicero connection data frame
`peak_annotation`	Peak-to-gene annotation data frame
`coaccess_threshold`	Coaccessibility score threshold
`max_distance`	Maximum distance for peak-gene association

Value

Data frame suitable for TFinfo import

`get_bayesian_ridge_coefs` *Get Bayesian Ridge coefficients for a single target gene*

Description

Fit Bayesian Ridge regression for a target gene and return coefficients with uncertainty estimates. Exact port from Python's get_bayesian_ridge_coefs.

Usage

```
get_bayesian_ridge_coefs(
  target_gene,
  gem,
  gem_scaled,
  TFdict,
  cellstate = NULL,
  scaling = TRUE
)
```

Arguments

`target_gene`	Target gene name
`gem`	Gene expression matrix (cells x genes data.frame)
`gem_scaled`	Scaled gene expression matrix
`TFdict`	TF-target dictionary
`cellstate`	Optional cell state data frame
`scaling`	Whether to use scaled data

Value

List with coef_mean, coef_variance, coef_names (or 0,0,0 if not applicable)

get_embedding *Get embedding from Seurat object*

Description

Extract dimensional reduction embedding (e.g., UMAP, tSNE, PCA).

Usage

```
get_embedding(seurat, embedding_name = "umap", dims = 1:2)
```

Arguments

seurat	Seurat object
embedding_name	Name of the reduction (e.g., "umap", "tsne", "pca")
dims	Dimensions to extract (default: first 2)

Value

Matrix (cells x dimensions)

Examples

```
## Not run:
# Get UMAP coordinates
umap <- get_embedding(seurat, "umap")

# Get first 50 PCs
pcs <- get_embedding(seurat, "pca", dims = 1:50)

## End(Not run)
```

get_links *Get Links object for network analysis*

Description

Fits GRNs per cluster and returns a Links object containing regulatory networks for each cluster.

Usage

```
get_links(
  oracle,
  cluster_name_for_GRN_unit = NULL,
  alpha = 10,
  bagging_number = 20,
  verbose_level = 1,
  n_jobs = -1
)
```

Arguments

oracle	Oracle object
cluster_name_for_GRN_unit	Column name for clustering
alpha	Regularization strength
bagging_number	Number of bagging iterations
verbose_level	Verbosity (0=silent, 1=progress, 2=detailed)
n_jobs	Number of parallel jobs

Value

Links object

---

get_network_entropy *Calculate network entropy*

Description

Computes the entropy of degree distribution as a measure of network complexity.

Usage

```
get_network_entropy(links, cluster = NULL)
```

Arguments

links	Links object
cluster	Specific cluster (NULL for all)

Value

Named vector of entropy values

get_network_score *Calculate network scores*

Description

Compute network centrality metrics for genes.

Usage

```
get_network_score(links)
```

Arguments

links	Links object

Value

Modified Links object with scores

get_perturb_value *Get perturbation values from expression data*

Description

Helper function to get perturbation values based on expression statistics.

Usage

```
get_perturb_value(seurat, gene, method = "knockout")
```

Arguments

seurat	Seurat object
gene	Gene name
method	Method for value: "knockout" (0), "min", "max", "median", "mean", "percentile_X" (e.g., "percentile_90")

Value

Perturbation value

identify_hubs *Identify hub genes*

Description

Finds highly connected genes (hubs) in the network.

Usage

```
identify_hubs(
  links,
  cluster = NULL,
  top_n = 20,
  method = c("degree", "betweenness", "eigenvector")
)
```

Arguments

links	Links object
cluster	Specific cluster
top_n	Number of top hubs
method	Hub definition: "degree", "betweenness", "eigenvector"

Value

Data frame with hub genes

knn_distances_to_weights_cpp *Compute KNN connectivity weights from distances*

Description

Converts KNN distance matrix to connectivity weights using Gaussian kernel.

Usage

```
knn_distances_to_weights_cpp(distances, sigma = 0)
```

Arguments

distances	Distance matrix (cells x k)
sigma	Gaussian kernel bandwidth (if 0, auto-estimate)

Value

Weight matrix (cells x k)

knn_impute_cpp *KNN imputation using precomputed neighbors*

Description

Performs KNN-based smoothing/imputation of expression data using precomputed nearest neighbor indices and weights.

Usage

```
knn_impute_cpp(X, knn_idx, weights, diag_weight = 1)
```

Arguments

X	Expression matrix (cells x genes)
knn_idx	KNN index matrix (cells x k), 0-indexed
weights	Weight vector for neighbors (length k)
diag_weight	Weight for self (diagonal)

Value

Smoothed expression matrix

Links *Links Class for Network Analysis*

Description

Links class stores and analyzes gene regulatory networks. It contains GRN links (edges) for each cluster and provides methods for filtering, merging, and computing network metrics.

Public fields

`links_dict` Named list of link data frames (one per cluster)

`filtered_links` Filtered links after applying thresholds

`merged_score` Merged network scores across clusters

`TFdict` TF-target dictionary

`cluster_column` Cluster column name

`raw_count` Raw link counts per cluster

Methods

Public methods:

- Links$new()
- Links$filter_links()
- Links$get_network_score()
- Links$get_links_df()
- Links$get_degree_distribution()
- Links$compare_gene()
- Links$get_top_genes()
- Links$get_regulators()
- Links$get_targets()
- Links$to_igraph()
- Links$save()
- Links$print()
- Links$clone()

Method `new()`**:** Create a new Links object

Usage:

```
Links$new(links_dict = NULL, TFdict = NULL, cluster_column = NULL)
```

Arguments:

`links_dict` Named list of link data frames

`TFdict` TF dictionary

`cluster_column` Cluster column name

Method `filter_links()`**:** Filter links by statistical significance and coefficient threshold

Usage:

```
Links$filter_links(p = 0.001, threshold_number = 10000, min_coef = 0)
```

Arguments:

`p` Percentile threshold for filtering (0-1)

`threshold_number` Maximum number of links to keep per cluster

`min_coef` Minimum absolute coefficient

Method `get_network_score()`**:** Get network scores for each gene

Computes degree, betweenness, eigenvector centrality for genes.

Usage:

```
Links$get_network_score()
```

Method `get_links_df()`**:** Get links as combined data frame

Usage:

```
Links$get_links_df(filtered = TRUE)
```

Arguments:

`filtered` Use filtered links

Returns: Data frame with all links

Method `get_degree_distribution()`**:** Get degree distribution statistics

Usage:

```
Links$get_degree_distribution(cluster = NULL, mode = "all")
```

Arguments:

`cluster` Specific cluster (NULL for all)
`mode` Degree mode: "all", "in", "out"

Returns: Data frame with degree statistics

Method `compare_gene()`**:** Compare network metrics between clusters

Usage:

```
Links$compare_gene(gene)
```

Arguments:

`gene` Gene to compare

Returns: Data frame with per-cluster metrics

Method `get_top_genes()`**:** Get top genes by network metric

Usage:

```
Links$get_top_genes(metric = "degree_all", cluster = NULL, n = 20)
```

Arguments:

`metric` Metric name (degree_all, betweenness, eigenvector)
`cluster` Specific cluster (NULL for all)
`n` Number of top genes

Returns: Data frame with top genes

Method `get_regulators()`**:** Get regulators of a target gene

Usage:

```
Links$get_regulators(target, cluster = NULL)
```

Arguments:

`target` Target gene name
`cluster` Specific cluster (NULL for all)

Returns: Data frame with regulator information

Method `get_targets()`**:** Get targets of a regulator

Usage:

```
Links$get_targets(source, cluster = NULL)
```

Arguments:

`source` Regulator gene name
`cluster` Specific cluster (NULL for all)

Returns: Data frame with target information

Method `to_igraph()`**:** Export to igraph object

Usage:

```
Links$to_igraph(cluster = NULL)
```

Arguments:

`cluster` Specific cluster (NULL = merge all)

Returns: igraph object

Method `save()`**:** Save Links object

Usage:

```
Links$save(file_path)
```

Arguments:

`file_path` Path to save file

Method `print()`**:** Print Links summary

Usage:

```
Links$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Links$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`load_base_grn` *Load pre-built base GRN*

Description

Loads pre-built base GRN data for common reference genomes.

Usage

```
load_base_grn(ref_genome, lineage = NULL)
```

Arguments

`ref_genome`	Reference genome name
`lineage`	Tissue/lineage type (if available)

Value

TFdict (named list)

load_oracle *Load Oracle object from file*

Description

Load Oracle object from file

Usage

```
load_oracle(path)
```

Arguments

path	File path to saved Oracle object

Value

Oracle object

load_tfinfo *Load TFinfo object from file*

Description

Load TFinfo object from file

Usage

```
load_tfinfo(file_path)
```

Arguments

file_path	Path to saved TFinfo object

Value

TFinfo object

markov_simulate *Markov simulation of cell trajectories*

Description

Simulates cell state transitions using Markov chain based on transition probability matrix.

Usage

```
markov_simulate(
  trans_prob,
  start_cells,
  n_steps = 100,
  n_duplicates = 10,
  seed = 123
)
```

Arguments

trans_prob	Transition probability matrix
start_cells	Starting cell indices (1-indexed)
n_steps	Number of simulation steps
n_duplicates	Number of trajectories per start cell
seed	Random seed

Value

Matrix of trajectories (n_trajectories x n_steps+1)

markov_walk_batch_cpp *Batch Markov simulation with duplication*

Description

Runs multiple Markov simulations per starting cell for robust estimation.

Usage

```
markov_walk_batch_cpp(
  start_cells,
  trans_prob,
  n_steps,
  n_duplicates,
  seed = 123L
)
```

Arguments

start_cells	Starting cell indices (0-indexed)
trans_prob	Transition probability matrix
n_steps	Number of steps
n_duplicates	Number of simulations per start cell
seed	Random seed

Value

Matrix of trajectories (n_start * n_duplicates x n_steps+1)

---

markov_walk_cpp *Markov chain random walk simulation*

Description

Performs Markov chain simulation based on transition probability matrix. This simulates cell state transitions over multiple time steps.

Usage

```
markov_walk_cpp(start_cells, trans_prob, n_steps, seed = 123L)
```

Arguments

start_cells	Starting cell indices (0-indexed)
trans_prob	Transition probability matrix (cells x cells)
n_steps	Number of simulation steps
seed	Random seed for reproducibility

Value

Matrix of trajectory indices (n_start x n_steps+1)

Net *Net Class for GRN Inference*

Description

Net class represents a gene regulatory network model for a specific gene subset. It handles the regression fitting and coefficient estimation.

Public fields

`target_gene` Target gene for this model

`regulators` Vector of regulator gene names

`all_genes` All genes in the expression matrix

`coef_matrix` Full coefficient matrix

`fitted` Whether model has been fitted

Methods

Public methods:

- `Net$new()`
- `Net$fit()`
- `Net$get_coef()`
- `Net$get_active_regulators()`
- `Net$print()`
- `Net$clone()`

Method `new()`**:** Create a new Net object

Usage:

```
Net$new(target_gene = NULL, regulators = NULL, all_genes = NULL)
```

Arguments:

`target_gene` Target gene

`regulators` Regulator genes

`all_genes` All genes

Method `fit()`**:** Fit Ridge regression model

Usage:

```
Net$fit(gem, alpha = 10, bagging_number = 20, sample_frac = 0.8)
```

Arguments:

`gem` Gene expression matrix (cells x genes)

`alpha` Regularization strength

`bagging_number` Number of bagging iterations

`sample_frac` Sample fraction for bagging

Returns: Self (modified)

Method `get_coef()`**:** Get coefficient for a specific regulator

Usage:

```
Net$get_coef(regulator)
```

Arguments:

`regulator` Regulator gene name

Returns: Coefficient value

Method `get_active_regulators()`**:** Get all non-zero regulators

Usage:

```
Net$get_active_regulators()
```

Returns: Character vector of regulator names

Method `print()`**:** Print Net summary

Usage:

```
Net$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Net$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`network_summary` *Calculate network statistics summary*

Description

Calculate network statistics summary

Usage

```
network_summary(links, cluster = NULL)
```

Arguments

`links`	Links object
`cluster`	Specific cluster (NULL for each cluster separately)

Value

Data frame with network statistics

normalize_flow_cpp *Normalize flow vectors*

Description

Normalizes flow vectors to a reference percentile magnitude.

Usage

```
normalize_flow_cpp(flow, percentile = 99.5)
```

Arguments

flow	Flow matrix (grid_points x 2)
percentile	Percentile for normalization (default 99.5)

Value

Normalized flow and magnitude

Oracle *Oracle Class for In Silico Gene Perturbation Analysis*

Description

Oracle is the main class in CellOracleR. It imports scRNA-seq data (Seurat object) and TF information to infer cluster-specific GRNs. It can predict future gene expression patterns and cell state transitions in response to TF perturbations.

Details

The Oracle class stores:

- Seurat object with scRNA-seq data
- TF-target gene regulatory information (TFdict)
- GRN coefficients for simulation
- Perturbation simulation results

Public fields

`seurat` Seurat object containing scRNA-seq data

`cluster_column` Column name in metadata containing cluster information

`embedding_name` Name of dimensional reduction to use (e.g., "umap")

`TFdict` Named list: key = target gene, value = vector of regulator TFs

`all_target_genes` All target genes in TFdict

`all_regulatory_genes` All regulatory genes in TFdict

`active_regulatory_genes` Regulatory genes with active connections in GRN

`high_var_genes` High variability genes

`pcs` PCA results

`pca` PCA model

`k_knn_imputation` K used for KNN imputation

`knn` KNN graph

`knn_smoothing_w` KNN smoothing weights

`GRN_unit` GRN calculation unit: "cluster" or "whole"

`alpha_for_simulation` Regularization strength used for GRN

`coef_matrix` Coefficient matrix (for GRN_unit="whole")

`coef_matrix_per_cluster` List of coefficient matrices per cluster

`perturb_condition` Current perturbation condition

`embedding` Embedding coordinates

`delta_embedding` Simulated embedding shifts

`delta_embedding_random` Randomized embedding shifts (control)

`transition_prob` Transition probability matrix

`transition_prob_random` Random transition probability (control)

`corrcoef` Correlation coefficients

`corrcoef_random` Random correlation coefficients (control)

`flow_grid` Grid coordinates for flow visualization

`flow` Flow vectors on grid

`flow_norm` Normalized flow vectors

`total_p_mass` Probability mass at each grid point

`mass_filter` Mass filter for visualization

`colorandum` Cell colors based on cluster

Methods

Public methods:

- Oracle$new()
- Oracle$import_TF_data()
- Oracle$perform_PCA()
- Oracle$knn_imputation()
- Oracle$fit_GRN_for_simulation()
- Oracle$simulate_shift()
- Oracle$estimate_transition_prob()
- Oracle$calculate_embedding_shift()
- Oracle$calculate_grid_arrows()
- Oracle$calculate_mass_filter()
- Oracle$get_links()
- Oracle$save()
- Oracle$copy()
- Oracle$change_cluster_unit()
- Oracle$update_cluster_colors()
- Oracle$get_cluster_colors()
- Oracle$print()
- Oracle$clone()

Method `new()`**:** Create a new Oracle object

Usage:

```
Oracle$new(
  seurat = NULL,
  cluster_column = NULL,
  embedding_name = NULL,
  verbose = TRUE
)
```

Arguments:

`seurat` Seurat object with scRNA-seq data (normalized, not scaled)

`cluster_column` Column name containing cluster assignments

`embedding_name` Name of dimensional reduction (e.g., "umap", "tsne")

`verbose` Print messages

Returns: A new Oracle object

Method `import_TF_data()`**:** Import TF-target regulatory data

Usage:

```
Oracle$import_TF_data(
  TFinfo_df = NULL,
  TFinfo_path = NULL,
  TFdict = NULL,
  verbose = TRUE
)
```

Arguments:

`TFinfo_df` Data frame with columns: peak_id, gene_short_name, and TF columns

`TFinfo_path` Path to parquet file with TF info

`TFdict` Named list mapping target genes to regulator TFs

`verbose` Print messages

Method `perform_PCA()`**:** Perform PCA on expression data

Usage:

```
Oracle$perform_PCA(n_components = 50, use_seurat_pca = TRUE)
```

Arguments:

`n_components` Number of PCs to compute

`use_seurat_pca` Use existing PCA from Seurat object

Method `knn_imputation()`**:** Perform KNN imputation of expression data

Usage:

```
Oracle$knn_imputation(
  k = NULL,
  n_pca_dims = NULL,
  balanced = FALSE,
  diag_weight = 1
)
```

Arguments:

`k` Number of neighbors (default: 2.5% of cells)

`n_pca_dims` Number of PCA dimensions to use for KNN

`balanced` Use balanced KNN

`diag_weight` Weight for self in smoothing

Method `fit_GRN_for_simulation()`**:** Fit GRN for perturbation simulation

Usage:

```
Oracle$fit_GRN_for_simulation(
  GRN_unit = c("cluster", "whole"),
  alpha = 1,
  verbose_level = 1
)
```

Arguments:

`GRN_unit` "cluster" for cluster-specific GRNs or "whole" for one GRN

`alpha` Regularization strength for Ridge regression

`verbose_level` Verbosity: 0=silent, 1=progress, 2=detailed

Method `simulate_shift()`**:** Simulate gene perturbation effects

Usage:

```
Oracle$simulate_shift(
  perturb_condition,
  n_propagation = 3,
  GRN_unit = NULL,
  clip_delta_X = FALSE,
  ignore_warning = FALSE
)
```

Arguments:

`perturb_condition` Named list: gene name -> expression value

`n_propagation` Number of signal propagation steps (1-5)

`GRN_unit` Override GRN unit for simulation

`clip_delta_X` Clip to original expression range

`ignore_warning` Ignore validation warnings

Method `estimate_transition_prob()`: Estimate transition probability

Usage:

```
Oracle$estimate_transition_prob(
  n_neighbors = NULL,
  sigma_corr = 0.05,
  sampled_fraction = 0.3,
  calculate_randomized = TRUE,
  n_jobs = 1,
  random_seed = 123
)
```

Arguments:

`n_neighbors` Number of neighbors for KNN

`sigma_corr` Correlation kernel bandwidth

`sampled_fraction` Fraction of neighbors to sample

`calculate_randomized` Calculate randomized control

`n_jobs` Number of parallel jobs

`random_seed` Random seed

Method `calculate_embedding_shift()`: Calculate embedding shifts from transition probability

Usage:

```
Oracle$calculate_embedding_shift(sigma_corr = 0.05)
```

Arguments:

`sigma_corr` Kernel bandwidth (not used, kept for API compatibility)

Method `calculate_grid_arrows()`: Calculate grid arrows for visualization

Usage:

```
Oracle$calculate_grid_arrows(n_grid = 40, n_neighbors = 100, smooth = 0.5)
```

Arguments:

`n_grid` Number of grid points per dimension

`n_neighbors` Number of neighbors for smoothing

`smooth` Smoothing parameter

Method `calculate_mass_filter()`: Calculate mass filter for visualization

Usage:

```
Oracle$calculate_mass_filter(min_mass = 0.01)
```

Arguments:

`min_mass` Minimum mass threshold

Method `get_links()`: Get Links object for network analysis

Usage:

```
Oracle$get_links(
  cluster_name_for_GRN_unit = NULL,
  alpha = 10,
  bagging_number = 20,
  verbose_level = 1,
  n_jobs = -1
)
```

Arguments:

`cluster_name_for_GRN_unit` Cluster column for GRN unit

`alpha` Regularization strength

`bagging_number` Number of bagging iterations

`verbose_level` Verbosity level

`n_jobs` Number of parallel jobs

Returns: Links object

Method `save()`: Save Oracle object to file

Usage:

```
Oracle$save(file_path)
```

Arguments:

`file_path` Path to save file (should end with .celloracle.oracle)

Method `copy()`: Deep copy the Oracle object

Usage:

```
Oracle$copy()
```

Returns: A new Oracle object

Method `change_cluster_unit()`: Change the cluster unit used for analysis

Usage:

```
Oracle$change_cluster_unit(new_cluster_column)
```

Arguments:

`new_cluster_column` New cluster column name

Method `update_cluster_colors()`: Update cluster colors

Usage:

```
Oracle$update_cluster_colors(palette)
```

Arguments:

`palette` Named vector of colors for each cluster

Method `get_cluster_colors()`: Get cluster colors as a named vector

Usage:

```
Oracle$get_cluster_colors()
```

Returns: Named vector of colors

Method `print()`: Print Oracle object summary Process TFdict metadata Get imputed expression as data frame Get delta_X from simulation results Store simulation results in Seurat Validate perturbation condition Extract active regulatory genes from coefficient matrix

Usage:

```
Oracle$print()
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
Oracle$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`pairwise_dist_cpp` *Compute pairwise Euclidean distances*

Description

Computes Euclidean distance between all pairs of rows.

Usage

```
pairwise_dist_cpp(X)
```

Arguments

`X`	Input matrix (observations x features)

Value

Distance matrix (symmetric)

permute_rows_nsign_cpp *Permute rows with sign flip for randomization*

Description

Randomly permutes elements within each row and flips signs. Used for generating randomized control in transition probability.

Usage

```
permute_rows_nsign_cpp(X, seed = 123L)
```

Arguments

X	Input matrix (will be modified in place)
seed	Random seed

Value

Permuted matrix (also modifies X in place)

plot_cluster *Plot cells by cluster*

Description

Creates a scatter plot of cells colored by cluster on dimensional reduction.

Usage

```
plot_cluster(
  oracle,
  cluster_column = NULL,
  embedding_name = NULL,
  point_size = 0.5,
  alpha = 0.8,
  show_legend = TRUE,
  title = NULL
)
```

Arguments

oracle	Oracle object
cluster_column	Column for coloring (default: oracle$cluster_column)
embedding_name	Embedding to use (default: oracle$embedding_name)
point_size	Size of points
alpha	Point transparency
show_legend	Show legend
title	Plot title

Value

ggplot object

---

plot_degree_distribution

Plot degree distribution

Description

Plot degree distribution

Usage

```
plot_degree_distribution(
  links,
  cluster = NULL,
  mode = "all",
  log_scale = TRUE,
  title = NULL
)
```

Arguments

links	Links object
cluster	Specific cluster (NULL for all)
mode	Degree mode: "all", "in", "out"
log_scale	Use log scale
title	Plot title

Value

ggplot object

plot_gene_expression *Plot gene expression on embedding*

Description

Plot gene expression on embedding

Usage

```
plot_gene_expression(
  oracle,
  gene,
  layer = "data",
  point_size = 0.5,
  title = NULL
)
```

Arguments

oracle	Oracle object
gene	Gene name
layer	Expression layer: "data", "simulated", "delta_X"
point_size	Point size
title	Plot title

Value

ggplot object

plot_network_graph *Plot GRN as network graph*

Description

Plot GRN as network graph

Usage

```
plot_network_graph(
  links,
  cluster,
  top_n = 50,
  layout = "fr",
  node_size_by = "degree",
  title = NULL
)
```

Arguments

links	Links object
cluster	Cluster to plot
top_n	Number of top edges to show
layout	Layout algorithm: "fr", "kk", "circle", "star"
node_size_by	Size nodes by: "degree", "betweenness", "fixed"
title	Plot title

Value

ggplot object (requires ggraph)

---

plot_pseudotime *Plot pseudotime on embedding*

Description

Plot pseudotime on embedding

Usage

```
plot_pseudotime(oracle, pseudotime, point_size = 0.5, title = NULL)
```

Arguments

oracle	Oracle object
pseudotime	Named vector of pseudotime values
point_size	Point size
title	Plot title

Value

ggplot object

plot_quiver *Plot quiver (cell-level velocity arrows)*

Description

Plot quiver (cell-level velocity arrows)

Usage

```
plot_quiver(
  oracle,
  scale = 1,
  sample_frac = 0.3,
  arrow_color = "black",
  title = NULL
)
```

Arguments

oracle	Oracle object
scale	Arrow scale
sample_frac	Fraction of cells to show
arrow_color	Arrow color
title	Plot title

Value

ggplot object

plot_score_comparison *Compare network scores between clusters*

Description

Compare network scores between clusters

Usage

```
plot_score_comparison(links, gene, metric = "degree_all", title = NULL)
```

Arguments

links	Links object
gene	Gene to compare
metric	Metric to plot
title	Plot title

Value

ggplot object

`plot_scores_as_rank` *Plot network scores as ranked bar plot*

Description

Plot network scores as ranked bar plot

Usage

```
plot_scores_as_rank(
  links,
  cluster,
  metric = "degree_all",
  top_n = 20,
  fill_color = "#3288BD",
  title = NULL
)
```

Arguments

`links`	Links object
`cluster`	Cluster to plot
`metric`	Metric: "degree_all", "degree_in", "degree_out", "betweenness", "eigenvector"
`top_n`	Number of top genes
`fill_color`	Bar fill color
`title`	Plot title

Value

ggplot object

plot_simulation_combined *Create combined simulation plot*

Description

Create combined simulation plot

Usage

```
plot_simulation_combined(oracle, ncol = 2, genes = NULL)
```

Arguments

oracle	Oracle object with simulation results
ncol	Number of columns
genes	Genes to show expression for

Value

Combined plot (requires patchwork)

plot_simulation_flow *Plot perturbation simulation vector field*

Description

Visualizes the predicted cell state changes as a quiver/arrow plot.

Usage

```
plot_simulation_flow(
  oracle,
  scale = 1,
  min_mass = 0.01,
  arrow_color = "black",
  arrow_alpha = 0.8,
  point_size = 0.3,
  point_alpha = 0.3,
  title = NULL
)
```

Arguments

oracle	Oracle object with simulation results
scale	Arrow scaling factor
min_mass	Minimum probability mass threshold
arrow_color	Arrow color (NULL for cluster colors)
arrow_alpha	Arrow transparency
point_size	Background point size
point_alpha	Background point transparency
title	Plot title

Value

ggplot object

---

plot_transition_sankey

Create Sankey diagram from Oracle transition results

Description

Creates a Sankey diagram visualizing cell state transitions from perturbation simulation results.

Usage

```
plot_transition_sankey(
  oracle,
  cluster_column = NULL,
  before_column = "cluster_original",
  after_column = "cluster_predicted",
  color_dict = NULL,
  ...
)
```

Arguments

oracle	Oracle object with simulation results
cluster_column	Column containing cluster assignments
before_column	Column containing original cluster assignments
after_column	Column containing predicted cluster assignments
color_dict	Optional named vector of colors
...	Additional arguments passed to sankey()

Value

Sankey diagram

print.Links *Print method for Links*

Description

Print method for Links

Usage

```
## S3 method for class 'Links'
print(x, ...)
```

Arguments

x	Links object
...	Additional arguments (unused)

Value

Invisibly returns x

print.Net *Print method for Net*

Description

Print method for Net

Usage

```
## S3 method for class 'Net'
print(x, ...)
```

Arguments

x	Net object
...	Additional arguments (unused)

Value

Invisibly returns x

print.Oracle *Print method for Oracle*

Description

Print method for Oracle

Usage

```
## S3 method for class 'Oracle'
print(x, ...)
```

Arguments

x	Oracle object
...	Additional arguments (unused)

Value

Invisibly returns x

print.TFinfo *Print method for TFinfo*

Description

Print method for TFinfo

Usage

```
## S3 method for class 'TFinfo'
print(x, ...)
```

Arguments

x	TFinfo object
...	Additional arguments (unused)

Value

Invisibly returns x

row_norms_cpp *Row-wise L2 norm*

Description

Computes L2 norm (Euclidean length) for each row.

Usage

```
row_norms_cpp(X)
```

Arguments

X	Input matrix

Value

Vector of row norms

rowSds_cpp *Row-wise standard deviation*

Description

Computes standard deviation for each row of a matrix.

Usage

```
rowSds_cpp(X)
```

Arguments

X	Input matrix

Value

Vector of row standard deviations

save_oracle *Save Oracle object*

Description

Save Oracle object

Usage

```
save_oracle(oracle, path)
```

Arguments

oracle	Oracle object
path	File path to save

save_tfinfo *Save TFinfo object*

Description

Save TFinfo object

Usage

```
save_tfinfo(tfinfo, file_path)
```

Arguments

tfinfo	TFinfo object
file_path	Path to save

scale_cols_cpp *Scale matrix columns*

Description

Divides each column by its standard deviation.

Usage

```
scale_cols_cpp(X)
```

Arguments

X	Input matrix

Value

Scaled matrix

scan_motifs *Scan peaks for TF motifs*

Description

Convenience function for motif scanning.

Usage

```
scan_motifs(tfinfo, motifs = NULL, fpr = 0.02, n_cores = 1)
```

Arguments

tfinfo	TFinfo object
motifs	PWMatrixList or path to motif database (NULL for JASPAR)
fpr	False positive rate threshold
n_cores	Number of parallel cores

Value

Modified TFinfo object

score_cv_vs_mean *Score CV vs Mean for variable gene selection*

Description

Uses Support Vector Regression (SVR) to fit a nonparametric relationship between CV and mean expression, exactly like Python CellOracle.

Usage

```
score_cv_vs_mean(
  expr_matrix,
  n_top = 1000,
  min_expr_cells = 2,
  max_expr_avg = 20,
  min_expr_avg = 0,
  svr_gamma = NULL,
  winsorize = FALSE,
  winsor_perc = c(1, 99.5),
  sort_inverse = FALSE,
  plot = FALSE
)
```

Arguments

expr_matrix	Expression matrix (genes x cells)
n_top	Number of top variable genes to select
min_expr_cells	Minimum cells expressing gene
max_expr_avg	Maximum average expression
min_expr_avg	Minimum average expression
svr_gamma	SVR gamma parameter (default: 150/n_genes)
winsorize	Whether to winsorize data
winsor_perc	Winsorization percentiles
sort_inverse	If TRUE, sort from less to more noisy
plot	Whether to plot results

Value

List with scores and selected genes

`setup_parallel` *Setup parallel backend*

Description

Configure the future parallel backend for CellOracleR computations.

Usage

```
setup_parallel(
  workers = NULL,
  plan = c("multisession", "multicore", "sequential"),
  verbose = TRUE
)
```

Arguments

`workers`	Number of workers (cores) to use. Default uses all available.
`plan`	Parallel plan: "multisession" (default, cross-platform), "multicore" (Unix only, faster), or "sequential" (no parallelization).
`verbose`	Whether to print information

Details

This function sets up the parallel backend using the future framework.

- "multisession": Works on all platforms (Windows, Mac, Linux)
- "multicore": Faster but Unix/Mac only (not Windows)
- "sequential": No parallelization, useful for debugging

Value

Invisibly returns the previous plan

Examples

```
## Not run:
# Use 4 cores
setup_parallel(workers = 4)

# Use all available cores
setup_parallel()

# Disable parallelization
setup_parallel(plan = "sequential")

## End(Not run)
```

shortest_paths *Calculate shortest paths between genes*

Description

Calculate shortest paths between genes

Usage

```
shortest_paths(links, from, to, cluster = NULL)
```

Arguments

links	Links object
from	Source gene
to	Target gene(s)
cluster	Specific cluster

Value

List with path information

shuffle_coef_matrix_cpp *Shuffle coefficient matrix for randomization control*

Description

Creates a randomized version of the coefficient matrix by shuffling target gene assignments while preserving the overall structure.

Usage

```
shuffle_coef_matrix_cpp(coef_matrix, seed = 123L)
```

Arguments

coef_matrix	Original coefficient matrix
seed	Random seed for reproducibility

Value

Shuffled coefficient matrix

simulate_shift *Simulate gene perturbation shift*

Description

Simulate the effect of gene perturbation on the gene regulatory network. This function propagates the perturbation signal through the GRN to predict changes in gene expression.

Usage

```
simulate_shift(
  oracle,
  perturb_condition,
  n_propagation = 3,
  GRN_unit = NULL,
  clip_delta_X = FALSE,
  ignore_warning = FALSE
)
```

Arguments

`oracle`	Oracle object with fitted GRN
`perturb_condition`	Named list: gene -> expression value
`n_propagation`	Number of signal propagation steps (1-5)
`GRN_unit`	Which GRN to use: "cluster" or "whole"
`clip_delta_X`	Clip simulated values to original range
`ignore_warning`	Ignore validation warnings

Value

Modified Oracle object with simulation results

simulation_summary *Summary of simulation results*

Description

Print summary statistics of simulation results.

Usage

```
simulation_summary(oracle)
```

Arguments

oracle	Oracle object after simulation

Value

Invisible data frame with summary stats

---

summarize_trajectories *Summarize Markov simulation trajectories*

Description

Summarize Markov simulation trajectories

Usage

```
summarize_trajectories(trajectory, cluster_labels, n_steps_for_summary = NULL)
```

Arguments

trajectory	Trajectory matrix from markov_simulate
cluster_labels	Cluster labels for cells
n_steps_for_summary	Steps to consider for summary

Value

Summary statistics

TFinfo *TFinfo Class for Motif Analysis*

Description

TFinfo class handles transcription factor binding site (TFBS) analysis. It processes peak data, scans for motif matches, and generates TF-target gene dictionaries for GRN construction.

Public fields

peak_df	Data frame with peak information
ref_genome	Reference genome name
bsgenome	BSgenome object name
peak_ranges	GRanges object for peaks
scanned_df	Data frame with motif scanning results
TF_onehot	One-hot encoded TF binding matrix
all_TF_list	All TF names

Methods

Public methods:

- TFinfo$new()
- TFinfo$scan()
- TFinfo$filter()
- TFinfo$to_dataframe()
- TFinfo$to_dictionary()
- TFinfo$save()
- TFinfo$print()
- TFinfo$clone()

Method `new()`**:** Create a new TFinfo object

Usage:

```
TFinfo$new(peak_df = NULL, ref_genome = NULL, genomes_dir = NULL)
```

Arguments:

`peak_df` Data frame with columns: chr, start, end, peak_id, gene_short_name

`ref_genome` Reference genome (e.g., "hg38", "mm10")

`genomes_dir` Directory for genome data (optional)

Method `scan()`**:** Scan peaks for TF motif matches

Usage:

```
TFinfo$scan(motifs = NULL, fpr = 0.02, n_cores = 1, verbose = TRUE)
```

Arguments:

`motifs` PWMatrixList or path to motif database

`fpr` False positive rate threshold

`n_cores` Number of cores for parallel processing

`verbose` Print progress

Method `filter()`**:** Filter TF results by various criteria

Usage:

```
TFinfo$filter(min_peaks = 10, tfs_to_keep = NULL, tfs_to_remove = NULL)
```

Arguments:

`min_peaks` Minimum peaks with TF binding

`tfs_to_keep` TFs to include (NULL for all)

`tfs_to_remove` TFs to exclude

Method `to_dataframe()`**:** Convert to data frame format

Usage:

```
TFinfo$to_dataframe()
```

Returns: Data frame with peak_id, gene_short_name, and TF columns

Method `to_dictionary()`**:** Convert to TF dictionary format

Usage:

```
TFinfo$to_dictionary()
```

Returns: Named list mapping target genes to regulator TFs

Method `save()`**:** Save TFinfo object

Usage:

```
TFinfo$save(file_path)
```

Arguments:

`file_path` Path to save file

Method `print()`**:** Print TFinfo summary Get BSgenome package name from reference genome Load genome from BSgenome Get default motifs from JASPAR Load motifs from file

Usage:

```
TFinfo$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
TFinfo$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`visualization` *Visualization Functions*

Description

ggplot2-based visualization functions for CellOracleR

Index