Package: nat.fastcore (via r-universe)

July 11, 2026

Title Re-implementation of nat core functions in Rust

Version 0.5.0

Description A re-implementation of core 'nat' (Neuroanatomy Toolbox) functions in Rust for speed. Provides fast skeleton/tree (DAG) operations (geodesic distances, Strahler order, segment generation, twig pruning, synapse flow centrality, node classification, skeleton healing), mesh connected components, and NBLAST/synNBLAST neuron similarity scoring, exposed to R via extendr.

License GPL-3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Config/rextendr/version 0.4.2

SystemRequirements Cargo (Rust's package manager), rustc >= 1.65.0, xz

Config/pak/sysreqs xz-utils libclang-dev

Repository https://schlegelp.r-universe.dev

Date/Publication 2026-07-11 07:51:48 UTC

RemoteUrl https://github.com/schlegelp/fastcore-rs

RemoteRef HEAD

RemoteSha 56dbffdea75bc4c47ec43ebc03cce6f12eb764a3

RemoteSubdir R/nat.fastcore

Contents

`all_dists_to_root` *Compute all distances to root.*

Description

Compute all distances to root.

Usage

```
all_dists_to_root(parents, sources, weights)
```

Arguments

`parents`	Integer vector of 0-based parent indices (roots are `< 0`).
`sources`	Optional integer vector of node indices to measure from; `NULL` uses every node.
`weights`	Optional numeric vector of child-to-parent edge weights; `NULL` counts edges (hop distance).

Value

Numeric vector of distances to the root for each requested node.

`break_segments` *Break the tree into its linear segments (one integer vector per segment).*

Description

Break the tree into its linear segments (one integer vector per segment).

Usage

```
break_segments(parents)
```

Arguments

`parents`	Integer vector of 0-based parent indices (roots are `< 0`).

Value

List of integer vectors, one per linear segment.

`child_to_parent_dists` *Calculate child -> parent distances.*

Description

Calculate child -> parent distances.

Usage

```
child_to_parent_dists(parents, x, y, z)
```

Arguments

`parents`	Integer vector of 0-based parent indices (roots are `< 0`).
`x, y, z`	Numeric vectors of node coordinates, one entry per node.

Value

Numeric vector of Euclidean child-to-parent distances (`0` for roots).

`classify_nodes` *Classify nodes into roots (0), leaves (1), branch points (2) and slabs (3).*

Description

Classify nodes into roots (0), leaves (1), branch points (2) and slabs (3).

Usage

```
classify_nodes(parents)
```

Arguments

`parents`	Integer vector of 0-based parent indices (roots are `< 0`).

Value

Integer vector: `0` root, `1` leaf, `2` branch point, `3` slab.

`connected_components` *Connected components.*

Description

Connected components.

Usage

```
connected_components(parents)
```

Arguments

`parents`	Integer vector of 0-based parent indices (roots are `< 0`).

Value

Integer vector assigning each node a component id.

dist_to_root *Return path length from a single node to the root.*

Description

Return path length from a single node to the root.

Usage

```
dist_to_root(parents, node)
```

Arguments

parents	Integer vector of 0-based parent indices (roots are < 0).
node	Integer index of the node to measure from.

Value

Numeric path length (edge count) from node to its root.

generate_segments *Generate linear segments while maximising segment lengths.*

Description

Returns a list with segments (a list of integer vectors, one per segment) and lengths (per-segment lengths, or NULL if no weights were supplied).

Usage

```
generate_segments(parents, weights)
```

Arguments

parents	Integer vector of 0-based parent indices (roots are < 0).
weights	Optional numeric vector of edge weights; NULL returns no lengths.

Value

List with segments (list of integer node-index vectors) and lengths (numeric per-segment lengths, or NULL).

geodesic_distances *Geodesic distances between nodes.*

Description

Geodesic distances between nodes.

Usage

```
geodesic_distances(parents, sources, targets, weights, directed)
```

Arguments

parents	Integer vector of 0-based parent indices (roots are < 0).
sources	Optional integer vector of source node indices; NULL uses every node.
targets	Optional integer vector of target node indices; NULL uses every node.
weights	Optional numeric vector of edge weights; NULL counts edges.
directed	Logical; if TRUE only traverse edges child-to-parent.

Value

Numeric matrix of geodesic distances (sources in rows, targets in columns).

geodesic_nearest *Distance to the nearest target for each source.*

Description

Memory-efficient companion to geodesic_distances that never materialises the full distance matrix. Returns a list with distances (distance to the nearest target) and nearest (index of that target); sources without a reachable target get -1.

Usage

```
geodesic_nearest(parents, sources, targets, weights, directed)
```

Arguments

parents	Integer vector of 0-based parent indices (roots are < 0).
sources	Optional integer vector of source node indices; NULL uses every node.
targets	Optional integer vector of target node indices; NULL uses every node.
weights	Optional numeric vector of edge weights; NULL counts edges.
directed	Logical; if TRUE only traverse edges child-to-parent.

Value

List with `distances` (numeric, distance to the nearest target) and `nearest` (integer target index, `-1` when unreachable).

`geodesic_pairs` *Geodesic distances for explicit pairs of nodes.*

Description

`sources` and `targets` are parallel arrays of node indices; the returned vector holds the distance between each `(source, target)` pair.

Usage

```
geodesic_pairs(parents, sources, targets, weights, directed)
```

Arguments

`parents`	Integer vector of 0-based parent indices (roots are `< 0`).
`sources`	Integer vector of source node indices.
`targets`	Integer vector of target node indices (same length as `sources`).
`weights`	Optional numeric vector of edge weights; `NULL` counts edges.
`directed`	Logical; if `TRUE` only traverse edges child-to-parent.

Value

Numeric vector with the distance of each `(source, target)` pair.

`has_cycles` *Check whether the tree contains cycles.*

Description

Check whether the tree contains cycles.

Usage

```
has_cycles(parents)
```

Arguments

`parents`	Integer vector of 0-based parent indices (roots are `< 0`).

Value

Logical; `TRUE` if the parent structure contains a cycle.

heal_skeleton *Heal a fragmented skeleton by reconnecting its fragments.*

Description

Convenience wrapper that finds the minimal-length set of new edges between the skeleton's connected components (see `stitch_fragments()`) and regenerates a single rooted tree from them (see `reroot_rewire()`).

Usage

```
heal_skeleton(
  parents,
  x,
  y,
  z,
  method,
  max_dist,
  min_size,
  mask,
  radius,
  use_radius
)
```

Arguments

`parents`	Integer vector of 0-based parent indices (roots are `< 0`), e.g. from `node_indices()`.
`x, y, z`	Numeric vectors of node coordinates, one entry per node.
`method`	Character; `"ALL"` lets any node form a new edge, `"LEAFS"` restricts new edges to leaf and root nodes (faster, occasionally suboptimal attachment points).
`max_dist`	Optional numeric maximum length for any single new edge; gaps larger than this are left unhealed, so the result may stay fragmented. `NULL` means no limit.
`min_size`	Optional integer; fragments with fewer than this many nodes are excluded from healing and stay disconnected. `NULL` heals every fragment.
`mask`	Optional logical vector restricting which nodes may be used as endpoints for a new edge; combined with `method`. `NULL` allows every node.
`radius`	Optional numeric vector of node radii, one entry per node. Only required when `use_radius` is set.
`use_radius`	`TRUE`/`FALSE`, a number, or `NULL`. If set, node radii are taken into account when measuring distances, which prioritises connecting fragments of similar calibre. A number weights the effect: higher values give radius more influence (`TRUE` means 1). To keep this robust we use the mean radius of the segment a node belongs to, not the node's own radius. Note that `max_dist` is then measured in this augmented space too.

Value

Integer vector of new 0-based parent indices (roots are `-1`). If the skeleton could be fully healed this is a single tree with one root.

---

mesh_connected_components

Find connected components of a triangle mesh.

Description

`faces` is an (N, 3) matrix of vertex indices. Returns an integer vector of length `n_vertices` assigning each vertex the root-vertex index of its component.

Usage

```
mesh_connected_components(faces, n_vertices)
```

Arguments

`faces`	Integer or numeric `(N, 3)` matrix of triangle vertex indices.
`n_vertices`	Integer; total number of vertices in the mesh.

Value

Integer vector of length `n_vertices` giving each vertex the root-vertex index of its component.

---

nblast

Forward NBLAST of every query neuron against every target neuron.

Description

Returns an (n_query, n_target) score matrix.

Usage

```
nblast(
  q_points,
  q_vects,
  t_points,
  t_vects,
  q_alphas,
  t_alphas,
  smat_values,
  dist_edges,
  dot_edges,
```

```
  normalize,
  limit_dist,
  n_cores,
  precision,
  progress
)
```

Arguments

q_points	List of (N, 3) numeric matrices of query point coordinates.
q_vects	List of (N, 3) numeric matrices of query tangent vectors.
t_points	List of (N, 3) numeric matrices of target point coordinates.
t_vects	List of (N, 3) numeric matrices of target tangent vectors.
q_alphas	Optional list of per-point alpha vectors for the queries; NULL disables alpha weighting.
t_alphas	Optional list of per-point alpha vectors for the targets.
smat_values	Numeric scoring matrix, or NULL for the built-in FCWB matrix.
dist_edges	Numeric vector of distance bin edges for smat_values.
dot_edges	Numeric vector of dot-product bin edges for smat_values.
normalize	Logical; normalise each score by the query self-match score.
limit_dist	Optional numeric distance cut-off; NULL disables it.
n_cores	Optional integer thread count; NULL or <= 0 uses all cores.
precision	Integer; compute in 32- or 64-bit floats.
progress	Logical; display a progress bar.

Value

Numeric (n_query, n_target) score matrix.

---

nblast_allbyall *All-by-all forward NBLAST.*

Description

points/vects are lists of (N, 3) matrices (one per neuron). Returns an (n, n) score matrix; cell (i, j) is query i against target j.

Usage

```
nblast_allbyall(
  points,
  vects,
  alphas,
  smat_values,
  dist_edges,
  dot_edges,
  normalize,
  limit_dist,
  n_cores,
  precision,
  progress
)
```

Arguments

points	List of (N, 3) numeric matrices of neuron point coordinates.
vects	List of (N, 3) numeric matrices of unit tangent vectors, one per neuron and aligned with points.
alphas	Optional list of per-point alpha (anisotropy) vectors; NULL disables alpha weighting.
smat_values	Numeric scoring matrix, or NULL for the built-in FCWB matrix.
dist_edges	Numeric vector of distance bin edges for smat_values.
dot_edges	Numeric vector of dot-product bin edges for smat_values.
normalize	Logical; normalise each score by the query self-match score.
limit_dist	Optional numeric distance cut-off; NULL disables it.
n_cores	Optional integer thread count; NULL or <= 0 uses all cores.
precision	Integer; compute in 32- or 64-bit floats.
progress	Logical; display a progress bar.

Value

Numeric (n, n) score matrix; cell (i, j) is query i vs target j.

---

nblast_pairs *Forward NBLAST for a set of* (query, target) *index pairs.*

Description

q_idx/t_idx are 0-based indices into the query/target lists; element k of the result is query q_idx[k] against target t_idx[k].

Usage

```
nblast_pairs(
  q_points,
  q_vects,
  t_points,
  t_vects,
  q_idx,
  t_idx,
  q_alphas,
  t_alphas,
  smat_values,
  dist_edges,
  dot_edges,
  normalize,
  limit_dist,
  n_cores,
  precision,
  progress
)
```

Arguments

q_points	List of (N, 3) numeric matrices of query point coordinates.
q_vects	List of (N, 3) numeric matrices of query tangent vectors.
t_points	List of (N, 3) numeric matrices of target point coordinates.
t_vects	List of (N, 3) numeric matrices of target tangent vectors.
q_idx	Integer vector of 0-based query indices, one per pair.
t_idx	Integer vector of 0-based target indices (same length as q_idx).
q_alphas	Optional list of per-point alpha vectors for the queries; NULL disables alpha weighting.
t_alphas	Optional list of per-point alpha vectors for the targets.
smat_values	Numeric scoring matrix, or NULL for the built-in FCWB matrix.
dist_edges	Numeric vector of distance bin edges for smat_values.
dot_edges	Numeric vector of dot-product bin edges for smat_values.
normalize	Logical; normalise each score by the query self-match score.
limit_dist	Optional numeric distance cut-off; NULL disables it.
n_cores	Optional integer thread count; NULL or <= 0 uses all cores.
precision	Integer; compute in 32- or 64-bit floats.
progress	Logical; display a progress bar.

Value

Numeric vector of scores, one per (query, target) pair.

node_indices *For each node ID in* parents *find its index in* nodes.

Description

Importantly this is 0-indexed to match indexing in Rust. Roots will have parent index -1.

Usage

```
node_indices(nodes, parents)
```

Arguments

nodes	Integer vector of node IDs.
parents	Integer vector of parent IDs, one per node; roots use their own ID or a negative value.

Value

Integer vector of 0-based parent indices (-1 for roots).

prune_twigs *Prune twigs below given threshold.*

Description

Returns indices of nodes to keep.

Usage

```
prune_twigs(parents, threshold, weights)
```

Arguments

parents	Integer vector of 0-based parent indices (roots are < 0).
threshold	Numeric length threshold; twigs shorter than this are pruned.
weights	Optional numeric vector of edge weights; NULL counts edges.

Value

Integer vector of node indices to keep.

reroot_rewire *Regenerate a parent vector after adding a set of undirected edges.*

Description

Turns an edited edge set back into a valid rooted tree: the undirected adjacency is built from the original child -> parent edges plus the new `from`/`to` edges, then oriented away from `root` by breadth-first search.

Usage

```
reroot_rewire(parents, from, to, root)
```

Arguments

`parents`	Integer vector of 0-based parent indices (roots are `< 0`).
`from, to`	Integer vectors of 0-based node indices giving the undirected edges to add, e.g. the `from`/`to` returned by `stitch_fragments()`.
`root`	Integer 0-based index of the preferred root; its whole component is rooted there. Use a negative value to auto-pick (lowest index per component).

Value

Integer vector of new 0-based parent indices (roots are `-1`). Any component not reachable from `root` is rooted at its lowest-index node, so the result is valid even when the skeleton could not be fully healed.

smat_auto_limit *The* `limit_dist="auto"` *value for a scoring matrix.*

Description

The `limit_dist="auto"` value for a scoring matrix.

Usage

```
smat_auto_limit(smat_values, dist_edges, dot_edges, use_alpha)
```

Arguments

`smat_values`	Numeric scoring matrix, or `NULL` for the built-in FCWB matrix.
`dist_edges`	Numeric vector of distance bin edges for `smat_values`.
`dot_edges`	Numeric vector of dot-product bin edges for `smat_values`.
`use_alpha`	Logical; when falling back to the built-in matrix, use the alpha-weighted variant.

Value

Numeric `limit_dist` value implied by the scoring matrix.

`stitch_fragments` *Find the minimal-length edges that reconnect the fragments of a skeleton.*

Description

Given a per-node component label and node coordinates, this returns the set of new edges that would join the fragments into a single tree while minimising the total added length (a minimum spanning tree over the fragments). It does *not* modify the skeleton — see `heal_skeleton` for the one-shot version.

Usage

```
stitch_fragments(components, x, y, z, w, mask, max_dist)
```

Arguments

`components`	Integer vector giving each node's connected component, e.g. the output of `connected_components()`. Only equality of labels matters.
`x`, `y`, `z`	Numeric vectors of node coordinates, one entry per node.
`w`	Optional numeric vector giving a 4th coordinate, one entry per node. The search then happens in 4D, so nodes with similar `w` look closer together; pass a (scaled) radius here to prefer bridging fragments of similar calibre. Note that `max_dist` is then measured in 4D too. `NULL` searches in plain 3D.
`mask`	Optional logical vector marking the nodes that may be used as endpoints for a new edge; `NULL` allows every node. A fragment without a single eligible node cannot be connected.
`max_dist`	Optional numeric upper bound on the length of any single new edge; `NULL` means no limit. Fragments whose closest eligible nodes are farther apart than this are left disconnected.

Value

List with `from` and `to` (integer vectors of 0-based node indices, one pair per new edge) and `dist` (numeric edge lengths). At most `(#fragments - 1)` edges.

strahler_index *Calculate Strahler Index.*

Description

Calculate Strahler Index.

Usage

```
strahler_index(parents, greedy, to_ignore, min_twig_size)
```

Arguments

parents	Integer vector of 0-based parent indices (roots are < 0).
greedy	Logical; use the greedy variant of the algorithm.
to_ignore	Optional integer vector of node indices to skip.
min_twig_size	Optional integer; ignore twigs shorter than this.

Value

Integer vector with the Strahler index of each node.

synapse_flow_centrality *Synapse flow centrality for each node.*

Description

presynapses/postsynapses give the number of pre-/post-synapses at each node. mode is one of "centrifugal", "centripetal" or "sum".

Usage

```
synapse_flow_centrality(parents, presynapses, postsynapses, mode)
```

Arguments

parents	Integer vector of 0-based parent indices (roots are < 0).
presynapses	Integer vector: number of presynapses at each node.
postsynapses	Integer vector: number of postsynapses at each node.
mode	Character; one of "centrifugal", "centripetal" or "sum".

Value

Integer vector with the synapse flow centrality of each node.

`synblast` *Forward syNBLAST of every query neuron against every target neuron.*

Description

Returns an (n_query, n_target) score matrix.

Usage

```
synblast(
  q_points,
  q_types,
  t_points,
  t_types,
  smat_values,
  dist_edges,
  dot_edges,
  normalize,
  n_cores,
  precision,
  progress
)
```

Arguments

`q_points`	List of `(N, 3)` numeric matrices of query connector coordinates.
`q_types`	List of integer vectors of query per-connector type ids.
`t_points`	List of `(N, 3)` numeric matrices of target connector coordinates.
`t_types`	List of integer vectors of target per-connector type ids.
`smat_values`	Numeric scoring matrix, or `NULL` for the built-in FCWB matrix.
`dist_edges`	Numeric vector of distance bin edges for `smat_values`.
`dot_edges`	Numeric vector of dot-product bin edges for `smat_values`.
`normalize`	Logical; normalise each score by the query self-match score.
`n_cores`	Optional integer thread count; `NULL` or `<= 0` uses all cores.
`precision`	Integer; compute in 32- or 64-bit floats.
`progress`	Logical; display a progress bar.

Value

Numeric `(n_query, n_target)` score matrix.

`synblast_allbyall` *All-by-all forward syNBLAST over synapse clouds.*

Description

`points` are lists of (N, 3) connector coordinate matrices and `types` the matching per-connector integer type ids. Returns an (n, n) score matrix.

Usage

```
synblast_allbyall(
  points,
  types,
  smat_values,
  dist_edges,
  dot_edges,
  normalize,
  n_cores,
  precision,
  progress
)
```

Arguments

`points`	List of `(N, 3)` numeric matrices of connector coordinates, one per neuron.
`types`	List of integer vectors of per-connector type ids, aligned with `points`.
`smat_values`	Numeric scoring matrix, or `NULL` for the built-in FCWB matrix.
`dist_edges`	Numeric vector of distance bin edges for `smat_values`.
`dot_edges`	Numeric vector of dot-product bin edges for `smat_values`.
`normalize`	Logical; normalise each score by the query self-match score.
`n_cores`	Optional integer thread count; `NULL` or `<= 0` uses all cores.
`precision`	Integer; compute in 32- or 64-bit floats.
`progress`	Logical; display a progress bar.

Value

Numeric `(n, n)` score matrix; cell `(i, j)` is query `i` vs target `j`.

Index