statOT

Release 0.1

Stephen Zhang

Apr 12, 2021

CONTENTS:

CHAPTER ONE

STATOT PACKAGE

1.1 Core implementation

`statot.inference.`**`compute_conditional_mfpt`**`(`*P*, *j*, *sink_idx*`)`

Compute conditional mean first passage time MFPT(x_i -> x_j). Based on implementation in PBA.

Parameters

- **`P`** – transition matrix
- **`j`** – index j of cell x_j
- **`sink_idx`** – boolean array of length *N*, set to *True* for sinks and *False* otherwise.

Returns vector t_i containing MFPT(x_i -> **x_**)

`statot.inference.`**`compute_fate_probs`**`(`*P*, *sink_idx*`)`

Compute fate probabilities by individual sink cell

Parameters

- **`P`** – transition matrix
- **`sink_idx`** – boolean array of length *N*, set to *True* for sinks and *False* otherwise.

Returns matrix with dimensions *(N, S)* where *S* is the number of sink cells present.

`statot.inference.`**`compute_fate_probs_lineages`**`(`*P*, *sink_idx*, *labels*`)`

Compute fate probabilities by lineage

Parameters

- **`P`** – transition matrix
- **`sink_idx`** – boolean array of length *N*, set to *True* for sinks and *False* otherwise.
- **`labels`** – string array of length *N* containing lineage names. Only those entries corresponding to sinks will be used.

Returns matrix with dimensions *(N, L)* where *L* is the number of lineages with sinks.

`statot.inference.`**`gaussian_tr`**`(`*C*, *h*`)`

Form Gaussian (discrete heat flow) transition matrix of bandwidth h

Parameters

- **`C`** – pairwise square distances
- **`h`** – bandwidth

`statot.inference.`**`row_normalise`**`(`*gamma*, *sink_idx=None*`)`

Enforce sink condition and row normalise coupling to produce transition matrix

Parameters

- **gamma** – coupling produced by *statot()*;
- **sink_idx** – boolean array of length *N*, set to *True* for sinks and *False* otherwise. If provided, sets the transition distributions for all sinks to be the identity.

Returns transition matrix obtained by row-normalising the input *gamma*.

`statot.inference.`**`statot`**(*x*, *C=None*, *eps=None*, *method='ent'*, *g=None*, *dt=None*, *maxiter=5000*, *tol=1e-09*, *verbose=False*)

Fit statOT model

Parameters

- **x** – input data – *N* points of *M* dimensions in the form of a matrix with dimensions *(N, M)*
- **C** – cost matrix for optimal transport problem
- **eps** – regularisation parameter
- **method** – choice of regularisation – either "ent" (entropy) or "quad" (L2). "unbal" for unbalanced transport is not yet implemented. if "marginals", return just *mu* and *nu*.
- **g** – numeric array of length *N*, containing the relative growth rates for cells.
- **flow_rate** – used only in the growth-free case (flow only)
- **dt** – choice of the time step over which to fit the model
- **maxiter** – max number of iterations for OT solver
- **tol** – relative tolerance for OT solver convergence
- **verbose** – detailed output on convergence of OT solver.

Returns gamma (optimal transport coupling), mu (source measure), nu (target measure)

`statot.inference.`**`velocity_from_transition_matrix`**(*P*, *x*, *deltat*)

Estimate velocity field from transition matrix (i.e. compute expected displacements)

Parameters

- **P** – transition matrix
- **x** – input data – *N* points of *M* dimensions in the form of a matrix with dimensions *(N, M)*
- **deltat** – timestep for which *P* was calculated.

1.2 CellRank wrapper

class `statot.cr.`**`OTKernel`**(*adata*, *g*, *compute_cond_num=False*)

Bases: `cellrank.tl.kernels._base_kernel.Kernel`

Kernel class allowing statOT method to be used from CellRank. Call first *set_terminal_states* to specify which cells to use as sinks.

Parameters

- **adata** – *AnnData* object containing *N* cells. We can use any embedding for statOT, selected when calling *OTKernel.compute_transition_matrix()*.
- **g** – string specifying the key in *adata.obs* to a numeric array of length *N*, containing the relative growth rates for cells, or the array itself.

- **compute_cond_num** – set to *True* to compute the condition number of the transition matrix.

compute_transition_matrix(*eps*, *dt*, *expr_key='X_pca'*, *cost_norm_method=None*, *method='ent'*, *tol=1e-09*, *thresh=0*, *maxiter=5000*, *C=None*, *verbose=False*)

Compute transition matrix using StationaryOT.

Parameters

- **eps** – regularisation parameter
- **dt** – choice of the time step over which to fit the model
- **expr_key** – key to embedding to use in *adata.obsm*.
- **cost_norm_method** – cost normalisation method to use. use "mean" to ensure *mean(C) = 1*, or refer to *ot.utils.cost_normalization* in Python OT.
- **thresh** – threshold for output transition probabilities (no thresholding by default)
- **maxiter** – max number of iterations for OT solver
- **C** – cost matrix for optimal transport problem
- **verbose** – detailed output on convergence of OT solver.

copy() → statot.cr.OTKernel

Return a copy of itself. Note that the underlying **:paramref:`adata`** object is not copied.

statot.cr.**set_terminal_states**(*adata*, *sink_idx*, *labels*, *terminal_colors*)

Set user-specified terminal states for CellRank API functions and OTKernel.

Parameters

- **adata** – *AnnData* object containing *N* cells.
- **sink_idx** – string specifying the key in *adata.uns* to a boolean array of length *N*, set to *True* for sinks and *False* otherwise, or the array itself.
- **labels** – string array of length *N* containing lineage names. Only those entries corresponding to sinks will be used.
- **terminal_colors** – colors corresponding to terminal state labels.

1.3 pyKeOps-numpy implementation

statot.keops.**compute_fate_probs**(*Q*, *R*)

Compute fate probabilities from Q (LazyTensor) and R (np.ndarray)

Parameters

- **Q** – transient part of transition matrix from *get_QR_submat*, as *LazyTensor*
- **R** – absorbing part of transition matrix. Should aggregate the columns across fates, since the solver cannot solve multiple RHS at once.

statot.keops.**form_cost**(*mu_spt*, *nu_spt*, *norm_factor=None*, *keops=True*)

Form cost matrix (matrix of squared Euclidean distances)

Parameters

- **mu_spt** – support of source measure

- **nu_spt** – support of target measure
- **norm_factor** – normalisation factor as a *float*, *None* or "mean"
- **keops** – whether to return a *LazyTensor* or *np.array*

statot.keops.**get_QR_submat_ent**(*u*, *K*, *v*, *X*, *sink_idx*, *eps*, *cost_norm_factor*)

Compute Q (as LazyTensor) and R (as np.ndarray) matrices for entropy-regularised OT dual potentials (u, v)

Parameters

- **u** – dual potential for source distribution
- **K** – Gibbs kernel as *LazyTensor*
- **v** – dual potential for target distribution
- **X** – coordinates as np.ndarray
- **sink_idx** – boolean array of length *N*, set to *True* for sinks and *False* otherwise.
- **eps** – value of *eps* used for solving with *sinkhorn*
- **cost_norm_factor** – normalisation factor used in *form_cost*

statot.keops.**get_QR_submat_quad**(*u*, *C*, *v*, *X*, *sink_idx*, *eps*, *cost_norm_factor*)

Compute Q (as LazyTensor) and R (as np.ndarray) matrices for quadratically regularised OT dual potentials (u, v)

Parameters

- **u** – dual potential for source distribution
- **C** – cost matrix as *LazyTensor*
- **v** – dual potential for target distribution
- **X** – coordinates as np.ndarray
- **sink_idx** – boolean array of length *N*, set to *True* for sinks and *False* otherwise.
- **eps** – value of *eps* used for solving with *quad_ot_semismooth_newton*
- **cost_norm_factor** – normalisation factor used in *form_cost*

statot.keops.**quad_ot_semismooth_newton**(*mu*, *nu*, *C*, *eps*, *max_iter=50*, *theta=0.1*, *kappa=0.5*, *tol=0.001*, *eta=1e-05*, *cg_max_iter=500*, *verbose=False*)

Semismooth Newton algorithm for solving quadratically regularised optimal transport compatible with KeOps LazyTensor framework. Uses the method from Algorithm 2 of *Lorenz, D.A., Manns, P. and Meyer, C., 2019. Quadratically regularized optimal transport. Applied Mathematics & Optimization, pp.1-31.*

Parameters

- **mu** – source distribution
- **nu** – target distribution
- **C** – cost matrix as *LazyTensor*
- **max_iter** – maximum number of Newton steps
- **theta** – Armijo control parameter (choose in $(0, 1)$)

- **kappa** – Armijo step scaling parameter (choose in $(0, 1)$)
- **tol** – tolerance (inf-norm on marginals)
- **eta** – conjugate gradient regularisation parameter
- **cg_max_iter** – maximum number of conjugate gradient iterations
- **verbose** – flag for verbose output

statot.keops.**set_dtype**(*d*)

Set dtype to use in statot.keops

statot.keops.**sinkhorn**(*mu*, *nu*, *K*, *max_iter=5000*, *err_check=10*, *tol=1e-09*, *verbose=False*)

Sinkhorn algorithm for solving entropy-regularised optimal transport compatible with KeOps LazyTensor framework.

Parameters

- **mu** – source distribution
- **nu** – target distribution
- **K** – Gibbs kernel as *LazyTensor*
- **max_iter** – maximum number of iterations
- **err_check** – interval for checking marginal error
- **tol** – tolerance (inf-norm on marginals)
- **verbose** – flag for verbose output

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

INDEX