average-minimum-distance

Release 1.1.6

Daniel Widdowson

Nov 11, 2021

Table of Contents

amd.calculate.**AMD** (*periodic_set: Union[amd.PeriodicSet.PeriodicSet, Tuple[numpy.ndarray, numpy.ndarray]]*, *k: int*) → numpy.ndarray

Computes an AMD vector up to k from a periodic set.

Parameters

- **periodic_set** (*amd.PeriodicSet or tuple of ndarrays (motif, cell)*) – Representation of the periodic set, in Cartesian coordinates. amd.CifReader yields PeriodicSets which can be given here. Otherwise pass a tuple of arrays (motif, cell) in Cartesian form.
- **k** (*int*) – A m_b by k PDD matrix (with weights in the first column).

Returns AMD of periodic_set up to k.

Return type ndarray

amd.calculate.**AMD_estimate** (*periodic_set: Union[amd.PeriodicSet.PeriodicSet, Tuple[numpy.ndarray, numpy.ndarray]]*, *k: int*) → numpy.ndarray

Calculates an estimate of AMD based on the PPC.

amd.calculate.**PDD** (*periodic_set: Union[amd.PeriodicSet.PeriodicSet, Tuple[numpy.ndarray, numpy.ndarray]]*, *k: int*, *order: bool = True*, *collapse: bool = True*, *collapse_tol: float = 0.0001*) → numpy.ndarray

Computes a PDD up to k from a periodic set.

Parameters

- **periodic_set** (*amd.PeriodicSet or tuple of ndarrays (motif, cell)*) – Representation of the periodic set, in Cartesian coordinates. amd.CifReader yields PeriodicSets which can be given here. Otherwise pass a tuple of arrays (motif, cell) in Cartesian form.
- **k** (*int*) – A m_b by k PDD matrix (with weights in the first column).
- **order** (*bool, optional*) – Whether or not to lexicographically order the rows. Default True.
- **collapse** (*bool, optional*) – Whether or not to collapse identical rows (within a tolerance). Default True.
- **collapse_tol** (*float*) – If two rows have all entries closer than collapse_tol, they get collapsed. Default 1e-4.

Returns PDD of periodic_set up to k.

Return type ndarray

amd.calculate.**PDD_to_AMD** (*pdd: numpy.ndarray*) → numpy.ndarray

Calculates AMD from a PDD.

amd.calculate.**PPC** (*periodic_set: Union[amd.PeriodicSet.PeriodicSet, Tuple[numpy.ndarray, numpy.ndarray]]*)

Calculate the point packing coefficient (ppc) of periodic_set.

The ppc is a constant of any periodic set determining the asymptotic behaviour of its AMD/PDD as k -> infinity.

As k -> infinity, the ratio AMD_k / (n-th root of k) approaches the ppc (as does any row of a PDD).

For a unit cell U and m motif points in n dimensions, `ppc = \sqrt[n]{Vol[U] / (m * V_n)}`, where V_n is the volume of a unit sphere in n dimensions.

amd.compare.**AMD_cdist** (*amds: Union[int, float, complex, str, bytes, numpy.generic, Sequence[Union[int, float, complex, str, bytes, numpy.generic]], Sequence[Sequence[Any]], numpy.typing._array_like._SupportsArray], amds_: Union[int, float, complex, str, bytes, numpy.generic, Sequence[Union[int, float, complex, str, bytes, numpy.generic]], Sequence[Sequence[Any]], numpy.typing._array_like._SupportsArray], k: Optional[int] = None, low_memory: bool = False, metric: str = 'chebyshev', **kwargs*) → numpy.ndarray

Compare two sets of AMDs with each other. Returns a distance matrix.

Parameters

- **amds** (*ndarray or list of ndarrays*) – An m_a by n array (of m_a vectors/AMDs).
- **amds** – An m_b by n array (of m_b vectors/AMDs).
- **k** (*int, optional*) – If None, compare whole AMDs (largest k). Set k to an int to compare for a specific k (less than the maximum).
- **low_memory** (*bool, optional*) – Optionally use a slightly slower but more memory efficient method for large collections of AMDs (Chebyshev/l-inf distance only).
- **metric** (*str or callable, optional*) – Usually AMDs are compared with the Chebyshev/l-infinity distance. Can take any metric + kwargs accepted by scipy.spatial.distance.cdist.

Returns Returns a m_a by m_b distance matrix. The ij th entry is the distance between amds[i] and amds_[j] given by the metric.

Return type ndarray

amd.compare.**AMD_mst** (*amds: Union[int, float, complex, str, bytes, numpy.generic, Sequence[Union[int, float, complex, str, bytes, numpy.generic]], Sequence[Sequence[Any]], numpy.typing._array_like._SupportsArray], k: Optional[int] = None, low_memory: bool = False, metric: str = 'chebyshev', **kwargs*) → List[Tuple[int, int, float]]

Return list of edges in a minimum spanning tree based on AMDs.

Parameters

- **amds** (*ndarray or list of ndarrays*) – An m_a by n array (of m_a vectors/AMDs).
- **k** (*int, optional*) – If None, compare whole PDDs (largest k). Set k to an int to compare for a specific k (less than the maximum).
- **metric** (*str or callable, optional*) – Usually PDD rows are compared with the Chebyshev/l-infinity distance. Can take any metric + kwargs accepted by scipy.spatial.distance.cdist.

Returns Each tuple (i,j,w) is an edge in the mimimum spanning tree, where i and j are the indices of nodes and w is the weight on the edge connecting them.

Return type list of tuples

amd.compare.**AMD_pdist** (*amds: Union[int, float, complex, str, bytes, numpy.generic, Sequence[Union[int, float, complex, str, bytes, numpy.generic]], Sequence[Sequence[Any]], numpy.typing._array_like._SupportsArray], k: Optional[int] = None, low_memory: bool = False, metric: str = 'chebyshev', **kwargs*) → numpy.ndarray

Do a pairwise comparison on one set of AMDs.

Parameters

- **amds** (*ndarray or list of ndarrays*) – An m_a by n array (of m_a vectors/AMDs).
- **k** (*int, optional*) – If None, compare whole AMDs (largest k). Set k to an int to compare for a specific k (less than the maximum).
- **low_memory** (*bool, optional*) – Optionally use a slightly slower but more memory efficient method for large collections of AMDs (Chebyshev/l-inf distance only).
- **metric** (*str or callable, optional*) – Usually AMDs are compared with the Chebyshev/l-infinity distance. Can take any metric + kwargs accepted by scipy.spatial.distance.cdist.

Returns Returns a condensed distance matrix. Collapses a square distance matrix into a vector just keeping the upper traingle. scipy's squareform will convert to a square distance matrix.

Return type ndarray

amd.compare.**PDD_cdist** (*pdds: List[numpy.ndarray], pdds_: List[numpy.ndarray], k: Optional[int] = None, metric: str = 'chebyshev', verbose: bool = False, **kwargs*) → numpy.ndarray

Compare two sets of PDDs with each other. Returns a distance matrix.

Parameters

- **pdds** (*ndarray or list of ndarrays*) – A list of PDDs (ndarrays whose last/second dimension agree). If a 2D array, is interpreted as one PDD.
- **pdds** – A list of PDDs (ndarrays whose last/second dimension agree). If a 2D array, is interpreted as one PDD.
- **k** (*int, optional*) – If None, compare whole PDDs (largest k). Set k to an int to compare for a specific k (less than the maximum).
- **metric** (*str or callable, optional*) – Usually PDD rows are compared with the Chebyshev/l-infinity distance. Can take any metric + kwargs accepted by scipy.spatial.distance.cdist.
- **verbose** (*bool, optional*) – Optionally print progress as for large sets this can be long.

Returns Returns a m_a by m_b distance matrix. The ij th entry is the distance between pdds[i] and pdds_[j] given by earth mover's distance.

Return type ndarray

amd.compare.**PDD_mst** (*pdds: List[numpy.ndarray], amd_filter_cutoff: Optional[int] = None, k: Optional[int] = None, metric: str = 'chebyshev', verbose: bool = False, **kwargs*) → List[Tuple[int, int, float]]

Return list of edges in a minimum spanning tree based on PDDs.

Parameters

- **pdds** (*ndarray or list of ndarrays*) – A list of PDDs (ndarrays whose last/second dimension agree). If a 2D array, is interpreted as one PDD.
- **amd_filter_cutoff** (*int, optional*) – If specified, apply the AMD filter behaviour of the filter() function. The int specified is the n passed to filter(), the number of neighbours to connect in the neighbourhood graph.
- **k** (*int, optional*) – If None, compare whole PDDs (largest k). Set k to an int to compare for a specific k (less than the maximum).
- **metric** (*str or callable, optional*) – Usually PDD rows are compared

with the Chebyshev/l-infinity distance. Can take any metric + kwargs accepted by scipy.spatial.distance.cdist.

- **verbose** (*bool, optional*) – Optionally print progress as for large sets this can be long.

Returns Each tuple (i,j,w) is an edge in the mimimum spanning tree, where i and j are the indices of nodes and w is the weight on the edge connecting them.

Return type list of tuples

amd.compare.**PDD_pdist** (*pdds: List[numpy.ndarray], k: Optional[int] = None, metric: str = 'chebyshev', verbose: bool = False, **kwargs*) → numpy.ndarray

Do a pairwise comparison on one set of PDDs.

Parameters

- **pdds** (*list of ndarrays*) – A list of PDDs (ndarrays whose last/second dimension agree). If a 2D array, is interpreted as one PDD.
- **k** (*int, optional*) – If None, compare whole PDDs (largest k). Set k to an int to compare for a specific k (less than the maximum).
- **metric** (*str or callable, optional*) – Usually PDD rows are compared with the Chebyshev/l-infinity distance. Can take any metric + kwargs accepted by scipy.spatial.distance.cdist.
- **verbose** (*bool, optional*) – Optionally print progress as for large sets this can be long.

Returns Returns a condensed distance matrix. Collapses a square distance matrix into a vector just keeping the upper traingle. scipy's squareform will convert to a square distance matrix.

Return type ndarray

amd.compare.**emd** (*pdd, pdd_, metric='chebyshev', **kwargs*)

Earth mover's distance between two PDDs.

Parameters

- **pdd** (*ndarray*) – A m_a by k PDD matrix (with weights in the first column).
- **pdd** – A m_b by k PDD matrix (with weights in the first column).
- **metric** (*str or callable, optional*) – Usually rows are compared with the Chebyshev/l-infinity distance. Can take any metric + kwargs accepted by scipy.spatial.distance.cdist.

Returns Earth mover's distance between PDDs, where rows of the PDDs are compared with metric.

Return type float

Raises **ValueError** – Thrown if reference and comparison do not have the same number of columns.

amd.compare.**filter** (*n: int, pdds: List[numpy.ndarray], pdds_: Optional[List[numpy.ndarray]] = None, k: Optional[int] = None, low_memory: bool = False, metric: str = 'chebyshev', verbose: bool = False, **kwargs*) → Tuple[numpy.ndarray, numpy.ndarray]

For each item in pdds, get the n nearest items in pdds_ by AMD, then compare references to these nearest items with PDDs. Tries to comprimise between the speed of AMDs and the accuracy of PDDs.

If pdds_ is None, essentially sets pdds_ = pdds, i.e. do an 'AMD neighbourhood graph' for one set whose weights are PDD distances.

If n is smaller than the comparison set (pdds_ if it is not None and pdds if pdds_ is None), the AMD filter doesn't happen, this is essentially the same behaviour as pdd_cdist or pdd_pdist.

Parameters

- **n** (*int*) – Number of nearest neighbours to find.

- **pdds** (*ndarray or list of ndarrays*) – A list of PDDs (ndarrays whose last/second dimension agree). If a 2D array, is interpreted as one PDD.
- **pdds_** (*ndarray or list of ndarrays*) – A list of PDDs (ndarrays whose last/second dimension agree). If a 2D array, is interpreted as one PDD.
- **k** (*int, optional*) – If None, compare whole PDDs (largest k). Set k to an int to compare for a specific k (less than the maximum).
- **low_memory** (*bool, optional*) – Optionally use a slightly slower but more memory efficient method for large collections of AMDs (Chebyshev/l-inf distance only).
- **metric** (*str or callable, optional*) – Usually PDD rows are compared with the Chebyshev/l-infinity distance. Can take any metric + kwargs accepted by scipy.spatial.distance.cdist.
- **verbose** (*bool, optional*) – Optionally print progress as for large sets this can be long.

Returns For the i-th item in reference and some j < n, distance_matrix[i][j] is the distance from reference i to its j-th nearest neighbour in comparison (after the AMD filter). indices[i][j] is the index of said neighbour in comparison.

Return type tuple of ndarrays (distance_matrix, indices)

amd.compare.**linf**(*v*, *v_*)

l-infinity distance between vectors (AMDs).

amd.compare.**neighbours_from_distance_matrix**(*n: int*, *dm: numpy.ndarray*) → Tuple[numpy.ndarray, numpy.ndarray]

Given a distance matrix, find the n nearest neighbours of each item.

- genindex
- modindex
- search

a

A

E

F

L

M

N

P