



## treeIDW

**treeIDW** is a Python library designed for performing Inverse Distance Weighting (IDW) interpolation with an efficient KD-tree approach. It offers a user-friendly interface for newcomers while providing advanced features and optimizations that will appeal to experts in numerical methods and spatial data analysis.

## Introduction

Imagine you have data known at certain points (boundary nodes) and you need to estimate values at other points (internal nodes). **treeIDW** simplifies this task by automatically selecting the most relevant boundary nodes using a KD-tree, ensuring that only nodes with significant contributions are considered. This not only makes the interpolation more accurate but also speeds up the computation considerably. Whether you're a researcher, engineer, or someone exploring spatial data, **treeIDW** offers a robust solution that adapts to your level of expertise.

For non-experts, the library hides complex mathematical operations behind a simple interface, while experts will appreciate the fine-tuned control over parameters such as weight thresholds, bisection tolerances, and parallel processing options.

## Installation

Since **treeIDW** is not yet available on PyPI, you can install it locally as follows:

```
git clone https://github.com/Dorian210/treeIDW
cd treeIDW
pip install -e .
```

## Dependencies

Ensure that you have the following dependencies installed:

- `numpy`
- `numba`
- `scipy`

## Main Modules

- **`treeIDW.treeIDW`**  
Implements the core IDW interpolation method using a KD-tree to select the most relevant boundary nodes. This module efficiently ignores nodes with negligible impact, improving both accuracy and performance.
- **`treeIDW.helper_functions`**  
Contains optimized functions (leveraging `numba`) for computing IDW weights. These include both vectorized and parallelized implementations, which are ideal for handling large datasets.
- **`treeIDW.weight_function`**  
Defines the specific IDW weight calculation function. By default, this function computes the weight as the inverse of the squared distance, but it can be adapted to suit specific needs.

## Examples

Several example scripts demonstrating the usage of **treeIDW** are available in the `examples/` directory. These examples cover topics such as:

- Basic IDW interpolation
- Utilizing parallel processing for enhanced performance
- Fine-tuning weight thresholds to ignore less influential nodes

## Documentation

Complete API documentation is available in the [docs/](#) directory of the project or through the [online documentation portal](#).

## Contributions

Currently, I am not actively reviewing contributions. However, if you encounter any issues or have suggestions, please feel free to open an issue on the repository.

## License

This project is licensed under the [CeCILL License](#).

- [View Source](#)



- [View Source](#)

```
@nb.njit(cache=True)
def inv_dist_weight(
    boundary_nodes: numpy.ndarray,
    boundary_field: numpy.ndarray,
    internal_nodes: numpy.ndarray,
    relevant_nodes_inds_flat: numpy.ndarray,
    relevant_nodes_inds_sizes: numpy.ndarray
) -> numpy.ndarray[typing.Any, numpy.dtype[numpy.float64]]:
```

- [View Source](#)

Performs the Inverse Distance Weighting interpolation method using only provided boundary nodes indices in the weighted sum. This function is designed to be used after selecting the relevant nodes through a KD-tree method.

#### Parameters

- **boundary\_nodes** (npt.NDArray[np.float64]): Boundary nodes where the data is known. Must be of shape ( `n_interp` , `space_dim` ).
- **boundary\_field** (npt.NDArray[np.float64]): Known data. Must be of shape ( `n_interp` , `field_dim` ).
- **internal\_nodes** (npt.NDArray[np.float64]): Internal nodes where the interpolator is evaluated. Must be of shape ( `n_eval` , `space_dim` ).
- **relevant\_nodes\_inds\_flat** (npt.NDArray[np.int32]): Boundary nodes indices that weight in the IDW interpolator for each internal node. Must contain `n_eval` concatenated lists of indices between 0 and `n_interp` excluded.
- **relevant\_nodes\_inds\_sizes** (npt.NDArray[np.int32]): Sizes of each of the `n_eval` lists concatenated in `relevant_nodes_inds_flat` .

#### Returns

- **internal\_field** (npt.NDArray[np.float64]): Interpolated data. Should be of shape ( `n_eval` , `field_dim` ).

```
@nb.njit(parallel=True, cache=True)
def inv_dist_weight_parallel(
    boundary_nodes: numpy.ndarray,
    boundary_field: numpy.ndarray,
    internal_nodes: numpy.ndarray,
    relevant_nodes_inds_flat: numpy.ndarray,
    relevant_nodes_inds_sizes: numpy.ndarray
) -> numpy.ndarray[typing.Any, numpy.dtype[numpy.float64]]:
```

- [View Source](#)

Performs the Inverse Distance Weighting interpolation method using only provided boundary nodes indices in the weighted sum. This function is designed to be used after selecting the relevant nodes through a KD-tree method.

#### Parameters

- **boundary\_nodes** (npt.NDArray[np.float64]): Boundary nodes where the data is known. Must be of shape ( `n_interp` , `space_dim` ).
- **boundary\_field** (npt.NDArray[np.float64]): Known data. Must be of shape ( `n_interp` , `field_dim` ).
- **internal\_nodes** (npt.NDArray[np.float64]): Internal nodes where the interpolator is evaluated. Must be of shape ( `n_eval` , `space_dim` ).
- **relevant\_nodes\_inds\_flat** (npt.NDArray[np.int32]): Boundary nodes indices that weight in the IDW interpolator for each internal node. Must contain `n_eval` concatenated lists of indices between 0 and `n_interp` excluded.
- **relevant\_nodes\_inds\_sizes** (npt.NDArray[np.int32]): Sizes of each of the `n_eval` lists concatenated in `relevant_nodes_inds_flat` .

#### Returns

- **internal\_field** (npt.NDArray[np.float64]): Interpolated data. Should be of shape ( `n_eval` , `field_dim` ).

```
@nb.njit(cache=True)
def bisect_weight_elem(
    dist_squared_a: float,
    dist_squared_b: float,
    weight_treshold: float,
    rtol: float
) -> float:
```

• [View Source](#)

Compute the squared distance that correspond to a specific weight threshold using a bisection approach.

Parameters

- **dist\_squared\_a** (float): Left boundary of the interval containing the looked for weight treshold preimage.
- **dist\_squared\_b** (float): Right boundary of the interval containing the looked for weight treshold preimage.
- **weight\_treshold** (float): Image of the looked for squared distance by the weight function of the IDW algorithm.
- **rtol** (float, optional): The relative tolerance is used to compute the necessary number of bisection iterations, by default 1e-3

Returns

- **dist\_squared\_c** (float): Weight treshold preimage found through bisection.

**bisect\_weight** = <numba.\_GUFunc 'bisect\_weight'>

Vectorized version of bisect\_weight\_elem that finds squared distances corresponding to weight thresholds.

Parameters

- **dist\_squared\_a** (float): Left boundary of interval containing weight threshold preimage
- **dist\_squared\_b** (float): Right boundary of interval containing weight threshold preimage
- **weight\_treshold** (float): Target weight threshold value
- **rtol** (float): Relative tolerance for bisection convergence
- **out** (ndarray): Output array for storing computed squared distance

**bisect\_weight\_parallel** = <ufunc 'bisect\_weight\_parallel'>

Vectorized version of bisect\_weight\_elem that finds squared distances corresponding to weight thresholds.

Parallelized version of the function.

Parameters

- **dist\_squared\_a** (float): Left boundary of interval containing weight threshold preimage
- **dist\_squared\_b** (float): Right boundary of interval containing weight threshold preimage
- **weight\_treshold** (float): Target weight threshold value
- **rtol** (float): Relative tolerance for bisection convergence
- **out** (ndarray): Output array for storing computed squared distance

**compute\_weight\_vectorized** = <numba.\_GUFunc 'compute\_weight\_vectorized'>

Vectorized version of compute\_weight function for IDW calculations.

Parameters

- **dist\_squared** (float): Squared distance between interior and boundary nodes.
- **out** (ndarray): Output array containing the computed IDW weights.

**compute\_weight\_vectorized\_parallel** = <ufunc 'compute\_weight\_vectorized\_parallel'>

Vectorized version of compute\_weight function for IDW calculations. Parallelized version of the function.

Parameters

- **dist\_squared** (float): Squared distance between interior and boundary nodes.
- **out** (ndarray): Output array containing the computed IDW weights.



- [View Source](#)

```
def treeIDW(  
    boundary_nodes: numpy.ndarray,  
    boundary_field: numpy.ndarray,  
    internal_nodes: numpy.ndarray,  
    neglectible_treshold: float = 0.2,  
    bisect_rtol: float = 0.001,  
    parallel: bool = False  
) -> numpy.ndarray[typing.Any, numpy.dtype[numpy.float64]]:
```

- [View Source](#)

Performs IDW interpolation using a KD-tree to select relevant boundary nodes for each internal node. Only boundary nodes with significant weights are included in the interpolation.

#### Parameters

- **boundary\_nodes** (npt.NDArray[np.float64]): Boundary nodes where the data is known. Must be of shape ( `n_interp` , `space_dim` ).
- **boundary\_field** (npt.NDArray[np.float64]): Known data. Must be of shape ( `n_interp` , `field_dim` ).
- **internal\_nodes** (npt.NDArray[np.float64]): Internal nodes where the interpolator is evaluated. Must be of shape ( `n_eval` , `space_dim` ).
- **neglectible\_treshold** (float, optional): Relative weight threshold below which boundary nodes are ignored, by default 0.2
- **bisect\_rtol** (float, optional): Relative tolerance for bisection convergence, by default 1e-3
- **parallel** (bool, optional): Whether to use parallel implementation, by default False

#### Returns

- **internal\_field** (npt.NDArray[np.float64]): Interpolated data. Should be of shape ( `n_eval` , `field_dim` ).

## treeIDW.weight\_function



- [View Source](#)

```
@nb.njit(cache=True)
```

```
def compute_weight(dist_squared: float) -> float:
```

- [View Source](#)

Compute the IDW weight. Can be changed but is usually  $1/\text{dist\_squared}$ .

### Parameters

- **dist\_squared** (float): Squared distance between the interior node and the boundary node.

### Returns

- **weight** (float): IDW weight.