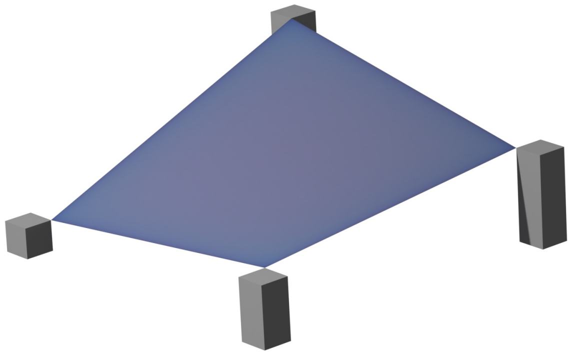
# interpylate



**interpylate** is a Python library for N-linear regular grid interpolation. It provides a flexible and efficient method to interpolate N-dimensional arrays using a N-linear approach, making it particularly useful for image interpolation and its higher-dimensional equivalents.

## Installation

```
pip install interpylate
```

## Dependencies

- `numpy` : Core dependency for array operations

## Architecture

The library is structured with a unified interface and specialized implementations:

- `NLinearRegularGridInterpolator` : A unified interface that automatically selects the appropriate dimension-specific implementation based on the input dimension
    - For dimensions > 3, it utilizes `NLinearRegularGridInterpolatorLarge`
- `LinearRegularGridInterpolator` : Optimized for 1D arrays
- `BiLinearRegularGridInterpolator` : Optimized for 2D arrays (images)
- `TriLinearRegularGridInterpolator` : Optimized for 3D arrays (volumes)
- `NLinearRegularGridInterpolatorLarge` : Less optimized general implementation for any dimensions

This design provides both optimization for common cases and flexibility for higher dimensions.

## Usage

```python
from interpylate import NLinearRegularGridInterpolator

# Create an instance of the interpolator
# This will automatically select the appropriate implementation based on dimension
interpolator = NLinearRegularGridInterpolator(dim=3)

# Evaluate the interpolation at specified coordinates
interpolated_values = interpolator.evaluate(NDarray, continuous_inds)

# Compute the gradient of the interpolated array
gradient = interpolator.grad(NDarray, continuous_inds, evaluate_too=False)

# Compute the hessian of the interpolated array
hessian = interpolator.hess(NDarray, continuous_inds, grad_too=False, evaluate_too=False)
```

## Main Features

- **Dimension-agnostic API**: Work with arrays of any dimension using a consistent interface
- **Performance-optimized implementations**: Specialized algorithms for 1D, 2D, and 3D cases
- **Gradient computation**: Calculate first-order derivatives of interpolated arrays
- **Hessian computation**: Compute second-order derivatives for advanced analysis
- **Regular grid support**: Designed specifically for regular grid structures

## Examples

The interpylate GitHub repository includes several example scripts demonstrating the library's capabilities:

- **1D Interpolation**: Basic interpolation for one-dimensional arrays
- **2D Interpolation**: Image interpolation techniques
- **3D Interpolation**: Volume data interpolation methods
- **Speed Contest**: Performance comparison between interpylate's TriLinearRegularGridInterpolator and scipy.interpolate.RegularGridInterpolator

## Documentation

The full API documentation is available in the docs/ directory of the project or via theonline documentation portal.

## Contributing

At the moment, I am not actively reviewing contributions. However, if you encounter issues or have suggestions, feel free to open an issue.

## License



- View Source

# interpylate.NLinearRegularGridInterpolatorLarge

- View Source

**class NLinearRegularGridInterpolatorLarge:**

- View Source

N-linear grid interpolator : interpolate a ND array between the indices using a N-linear method. For N<4, consider using `TriLinearRegularGridInterpolator` , `BiLinearRegularGridInterpolator` , or `LinearRegularGridInterpolator` . For exemple, for a 2D array, the interpolator finds a, b, c and d on each square of the grid such that F(x, y) = a + bx + cy + dxy

**NLinearRegularGridInterpolatorLarge**(dim)

- View Source

Create and initialize the interpolator.

Parameters

- **dim** (int): The dimension of the array to be interpolated.

**dim**

**nb_coefs**

**masks**

**mat**

**def evaluate**(self, NDarray, continuous_inds):

- View Source

Evaluate the interpolation at the coordinates given as input.

Parameters

- **NDarray** (numpy.ndarray): The array to interpolate.
- **continuous_inds** (numpy.ndarray of float): Coordinates where the interpolation is computed. It's shape should be (dim, n) if dim is the number of dimensions of the array interpolated.

Returns

- **evaluated** (numpy.ndarray of float): Interpolated values at the coordinates given. If `continuous_inds` is of shape (dim, n), the output will be of shape (n,).

**def grad**(self, NDarray, continuous_inds, evaluate_too=**False**):

- View Source

Compute the gradient of the interpolated array.

Parameters

- **NDarray** (numpy.ndarray): The array to interpolate.
- **continuous_inds** (numpy.ndarray of float): Coordinates where the gradient is computed. It's shape should be (dim, n) if dim is the number of dimensions of the array interpolated.
- **evaluate_too** (bool, optional): Set to True to evaluate on these points too, by default False

Returns

- **grad** (list of numpy.ndarray of float): The derivative of the interpolated array in each axis's direction. If `continuous_inds` is of shape (dim, n), each of the output will be of shape (n,).

**def hess**(self, NDarray, continuous_inds, grad_too=**False**, evaluate_too=**False**):

- View Source

Compute the hessian of the interpolated array.

Parameters

- **NDarray** (numpy.ndarray): The array to interpolate.
- **continuous_inds** (numpy.ndarray of float): Coordinates where the hessian is computed. It's shape should be (dim, n) if dim is the number of dimensions of the array interpolated.
- **grad_too** (bool, optional): Set to True to compute the gradient on these points too, by default False
- **evaluate_too** (bool, optional): Set to True to evaluate on these points too, by default False

Returns

- **hess** (list of numpy.ndarray of float): The second order derivatives of the interpolated array in each axis couple's direction. If `continuous_inds` is of shape (dim, n), each of the output will be of shape (n,).

# interpylate.NLinearRegularGridInterpolator

- View Source

**class NLinearRegularGridInterpolator**: 
- View Source

N-linear grid interpolator : interpolate a ND array between the indices using a N-linear method. For exemple, for a 2D array, the interpolator finds a, b, c and d on each square of the grid such that F(x, y) = a + bx + cy + dxy

**NLinearRegularGridInterpolator**(dim)
- View Source

Create and initialize the interpolator.

Parameters

- **dim** (int): The dimension of the array to be interpolated.

**dim**

**def evaluate**(self, NDarray, continuous_inds):
- View Source

Evaluate the interpolation at the coordinates given as input.

Parameters

- **NDarray** (numpy.ndarray): The array to interpolate.
- **continuous_inds** (numpy.ndarray of float): Coordinates where the interpolation is computed. It's shape should be (dim, n) if dim is the number of dimensions of the array interpolated.

Returns

- **numpy.ndarray of float**: Interpolated values at the coordinates given. If `continuous_inds` is of shape (dim, n), the output will be of shape (n,).

**def grad**(self, NDarray, continuous_inds, evaluate_too=**False**):
- View Source

Compute the gradient of the interpolated array.

Parameters

- **NDarray** (numpy.ndarray): The array to interpolate.
- **continuous_inds** (numpy.ndarray of float): Coordinates where the gradient is computed. It's shape should be (dim, n) if dim is the number of dimensions of the array interpolated.
- **evaluate_too** (bool, optional): Set to True to evaluate on these points too, by default False

Returns

- **list of numpy.ndarray of float**: The derivative of the interpolated array in each axis's direction. If `continuous_inds` is of shape (dim, n), each of the output will be of shape (n,).

**def hess**(self, NDarray, continuous_inds, grad_too=**False**, evaluate_too=**False**):
- View Source

Compute the hessian of the interpolated array.

Parameters

- **NDarray** (numpy.ndarray): The array to interpolate.
- **continuous_inds** (numpy.ndarray of float): Coordinates where the hessian is computed. It's shape should be (dim, n) if dim is the number of dimensions of the array interpolated.
- **grad_too** (bool, optional): Set to True to compute the gradient on these points too, by default False
- **evaluate_too** (bool, optional): Set to True to evaluate on these points too, by default False

Returns

- **list of numpy.ndarray of float**: The second order derivatives of the interpolated array in each axis couple's direction. If `continuous_inds` is of shape (dim, n), each of the output will be of shape (n,).

# interpylate.LinearRegularGridInterpolator

- View Source

**class LinearRegularGridInterpolator**: 
- View Source

Linear grid interpolator : interpolate a 1D array between the indices using a linear (afine) method : F(x) = a + bx

**LinearRegularGridInterpolator**()
- View Source

Create the interpolator.

**def evaluate**(self, vector, continuous_inds):
- View Source

Evaluate the interpolation at the coordinates given as input.

Parameters

- **vector** (numpy.ndarray): The array to interpolate.
- **continuous_inds** (numpy.ndarray of float): Coordinates where the interpolation is computed. It's shape should be (n,).

Returns

- **evaluated** (numpy.ndarray of float): Interpolated values at the coordinates given. If `continuous_inds` is of shape (n,), the output will be of shape (n,).

**def grad**(self, vector, continuous_inds, evaluate_too=**False**):
- View Source

Compute the gradient of the interpolated array.

Parameters

- **vector** (numpy.ndarray): The array to interpolate.
- **continuous_inds** (numpy.ndarray of float): Coordinates where the gradient is computed. It's shape should be (n,).
- **evaluate_too** (bool, optional): Set to True to evaluate on these points too, by default False

Returns

- **[grad_x]** (list of numpy.ndarray of float): The derivative of the interpolated array in each axis's direction. If `continuous_inds` is of shape (n,), each of the output will be of shape (n,).

**def hess**(self, vector, continuous_inds, grad_too=**False**, evaluate_too=**False**):
- View Source

Compute the hessian of the interpolated array.

Parameters

- **vector** (numpy.ndarray): The array to interpolate.
- **continuous_inds** (numpy.ndarray of float): Coordinates where the hessian is computed. It's shape should be (n,).
- **grad_too** (bool, optional): Set to True to compute the gradient on these points too, by default False
- **evaluate_too** (bool, optional): Set to True to evaluate on these points too, by default False

Returns

- **[]** (list of numpy.ndarray of float): The second order derivatives of the interpolated array in each axis couple's direction. If `continuous_inds` is of shape (n,), each of the output will be of shape (n,).

# interpylate.TriLinearRegularGridInterpolator

- View Source

**class TriLinearRegularGridInterpolator**:

- View Source

Tri-linear grid interpolator : interpolate a 3D array between the indices using a tri-linear method : F(x, y, z) = a + bx + cy + dz + exy + fxz + gyz + hxyz

**TriLinearRegularGridInterpolator**()

- View Source

Create the interpolator.

**def evaluate**(self, volume, continuous_inds):

- View Source

Evaluate the interpolation at the coordinates given as input.

Parameters

- **volume** (numpy.ndarray): The array to interpolate.
- **continuous_inds** (numpy.ndarray of float): Coordinates where the interpolation is computed. It's shape should be (3, n).

Returns

- **evaluated** (numpy.ndarray of float): Interpolated values at the coordinates given. If `continuous_inds` is of shape (3, n), the output will be of shape (n,).

**def grad**(self, volume, continuous_inds, evaluate_too=**False**):

- View Source

Compute the gradient of the interpolated array.

Parameters

- **volume** (numpy.ndarray): The array to interpolate.
- **continuous_inds** (numpy.ndarray of float): Coordinates where the gradient is computed. It's shape should be (3, n).
- **evaluate_too** (bool, optional): Set to True to evaluate on these points too, by default False

Returns

- **[grad_x, grad_y, grad_z]** (list of numpy.ndarray of float): The derivative of the interpolated array in each axis's direction. If `continuous_inds` is of shape (3, n), each of the output will be of shape (n,).

**def hess**(self, volume, continuous_inds, grad_too=**False**, evaluate_too=**False**):

- View Source

Compute the hessian of the interpolated array.

Parameters

- **volume** (numpy.ndarray): The array to interpolate.
- **continuous_inds** (numpy.ndarray of float): Coordinates where the hessian is computed. It's shape should be (3, n).
- **grad_too** (bool, optional): Set to True to compute the gradient on these points too, by default False
- **evaluate_too** (bool, optional): Set to True to evaluate on these points too, by default False

Returns

- **[hess_xy, hess_xz, hess_yz]** (list of numpy.ndarray of float): The second order derivatives of the interpolated array in each axis couple's direction. If `continuous_inds` is of shape (3, n), each of the output will be of shape (n,).

# interpylate.BiLinearRegularGridInterpolator

- View Source

**class BiLinearRegularGridInterpolator**:

- View Source

Bi-linear grid interpolator : interpolate a 2D array between the indices using a bi-linear method : F(x, y) = a + bx + cy + dxy

**BiLinearRegularGridInterpolator**()

- View Source

Create the interpolator.

**def evaluate**(self, image, continuous_inds):

- View Source

Evaluate the interpolation at the coordinates given as input.

Parameters

- **image** (numpy.ndarray): The array to interpolate.
- **continuous_inds** (numpy.ndarray of float): Coordinates where the interpolation is computed. It's shape should be (2, n).

Returns

- **evaluated** (numpy.ndarray of float): Interpolated values at the coordinates given. If `continuous_inds` is of shape (2, n), the output will be of shape (n,).

**def grad**(self, image, continuous_inds, evaluate_too=**False**):

- View Source

Compute the gradient of the interpolated array.

Parameters

- **image** (numpy.ndarray): The array to interpolate.
- **continuous_inds** (numpy.ndarray of float): Coordinates where the gradient is computed. It's shape should be (2, n).
- **evaluate_too** (bool, optional): Set to True to evaluate on these points too, by default False

Returns

- **[grad_x, grad_y]** (list of numpy.ndarray of float): The derivative of the interpolated array in each axis's direction. If `continuous_inds` is of shape (2, n), each of the output will be of shape (n,).

**def hess**(self, image, continuous_inds, grad_too=**False**, evaluate_too=**False**):

- View Source

Compute the hessian of the interpolated array.

Parameters

- **image** (numpy.ndarray): The array to interpolate.
- **continuous_inds** (numpy.ndarray of float): Coordinates where the hessian is computed. It's shape should be (2, n).
- **grad_too** (bool, optional): Set to True to compute the gradient on these points too, by default False
- **evaluate_too** (bool, optional): Set to True to evaluate on these points too, by default False

Returns

- **[hess_xy]** (list of numpy.ndarray of float): The second order derivatives of the interpolated array in each axis couple's direction. If `continuous_inds` is of shape (2, n), each of the output will be of shape (n,).