creek

unknown

Mar 25, 2022

CONTENTS

CHAPTER

ONE

A DATAASSIMILATION ROUTINE

This package implements dataassimilation [1]_ for `numpy.ndarray` objects, along with hand-written matrix multiplication.

See :*tools* folder for more information.

CHAPTER

TWO

LOADDATA.PY

`tools.loaddata.`**`load_all_data`**(*path_train*, *path_test*, *path_back*, *path_obs*)

Given paths to folders that contain only .npy files, this function loads 4 datasets (from the 4 different folders) and returns 4 numpy arrays which combine the .npy files - one numpy array associated with each folder.

Parameters

- **path_train** (*string*) –
- **path_test** (*string*) –
- **path_back** (*string*) –
- **path_obs** (*string*) –

Return type one numpy array associated with each folder.

Examples

```
>>> path1 = "data/train/"
>>> path2 = "data/test/"
>>> path3 = "data/background/"
>>> path4 = "data/satellite/"
>>> train, test, back, obs = load_all_data(path1, path2, path3, path4)
>>> np.shape(train)
(1200, 871, 913)
```

`tools.loaddata.`**`load_data`**(*path*)

Given a path, to the folder that contains only .npy files This function returns a numpy array that merges all these files together

Parameters **path** (*string*) –

Returns

- *numpy array that merges all these files*
- *together*

Examples

```
>>> a = 'data/background/'
>>> model_data = load_data(a)
>>> np.shape(model_data)
(5, 871, 913)
>>> b = 'data/satellite/'
>>> satellite_data = load_data(b)
>>> np.shape(satellite_data)
(5, 871, 913)
```

`tools.loaddata.`**`make_sequential`**(*data*)

Takes as input the raw wildfire data and returns the data in a sequential format, in the form of two arrays. The first is an array of "previous days", the second is an array of "next days". This is used in the prediction part of the project to train our models

The function returns two arrays, so in the case of test data, the following is the usage:

train_X, train_y = make_sequential(train_data)

where train_data is the raw wildfire data as loaded by the loaddata function

*** Important to look at required data shape in data parameter below ***

Parameters **`data`** (*`np.array`*) – A numpy array fo the wildfire data. In our case this would need to be either train or test data. *** –Required shape– * (simulations, days, x, y) ***

Returns

An array of all previous days. The shape of this array will be (n_sims * (days-1))

data_y (np.array): An array of all next days. The shape of this array will be (n_sims * (days-1))

Return type data_X (np.array)

`tools.loaddata.`**`reshape`**(*arr*)

Given a three dimensional numpy array, this function reshapes it into a two dimensional numpy array.

Parameters **`arr`** (*`array-like/numpy array`*) –

Return type two dimensional numpy array.

Examples

```
>>> a = np.array([[[1, 2], [3, 4], [5, 6]], [[1, 2], [3, 4], [5, 6]]])
>>> a_new = reshape(a)
>>> np.shape(a_new)
(2, 6)
```

```
>>> b = np.array([[[1, 0], [0, 1]], [[1, 0], [0, 1]]])
>>> b_new = reshape(b)
>>> np.shape(b_new)
(2, 4)
```

```
>>> c = np.array([[[-4, 0.2], [0.5, 2.3]], [[5.6, 0], [0, 1.3]]])
>>> c_new = reshape(c)
```

(continues on next page)

(continued from previous page)

```
>>> np.shape(c_new)
(2, 4)
```

`tools.loaddata.`**`reshape_all_datasets`**(*a*, *b*, *c*, *d*)

Reshapes four 3D numpy arrays into four 2D numpy arrays.

Parameters

- **a** (*array-like/numpy arrays*) –
- **b** (*array-like/numpy arrays*) –
- **c** (*array-like/numpy arrays*) –
- **d** (*array-like/numpy arrays*) –

Return type four 2D numpy arrays.

Examples

```
>>> a = np.array([[[1, 2], [3, 4], [5, 6]], [[1, 2], [3, 4], [5, 6]]])
>>> b = np.array([[[1, 0], [0, 1]], [[1, 0], [0, 1]]])
>>> c = np.array([[[-4, 0.2], [0.5, 2.3]], [[5.6, 0], [0, 1.3]]])
>>> d = np.array([[[-2, 4], [1, 0]], [[0, 0], [0, 0]]])
>>> a_new, b_new, c_new, d_new = reshape_all_datasets(a, b, c, d)
>>> np.shape(b_new)
(2, 4)
```

CHAPTER

THREE

DATAASSIMILATION.PY

`tools.dataassimilation.`**`KalmanGain`**(*B*, *H*, *R*)

Given B, H and R which are symmetric n x n matrix where n is the number of latent space, return kalman gain n x n matrix

Parameters

- **B** (*np.array or list of lists*) – 'n x n' array
- **H** (*np.array or list of lists*) – 'n x n' array
- **R** (*np.array or list of lists*) – 'n x n' array

Return type np.array 'n x n' (matrix)

Examples

```
>>> nNodes = 3
>>> B = np.identity(nNodes)
>>> H = np.identity(nNodes)
>>> R = np.identity(nNodes)
>>> k1 = KalmanGain(B, H, R)
>>> k1
array([[0.5, 0. , 0. ],
       [0. , 0.5, 0. ],
       [0. , 0. , 0.5]])
>>> nNodes = 99
>>> B = np.identity(nNodes)
>>> H = np.identity(nNodes)
>>> R = np.identity(nNodes)
>>> k2 = KalmanGain(B, H, R)
>>> np.shape(k2)
(99, 99)
```

`tools.dataassimilation.`**`assimilate`**(*B*, *H*, *R*, *mod_comp*, *sat_comp*)

Given an n x n matrix of B, H, R and model/satellite compressed data as a numpy array of size n return the updated prediction as an array of size n

Parameters

- **mod_comp** (*np.array*) – 'n' array
- **sat_comp** (*np.array*) – 'n' array
- **B** (*np.array or list of lists*) – 'n x n' array

- **H** (*np.array or list of lists*) – 'n x n' array
- **R** (*np.array or list of lists*) – 'n x n' array

Return type np.array of size n

Examples

```
>>> B = np.identity(1)
>>> H = np.identity(1)
>>> R = np.identity(1)
>>> mod_comp = np.array(list(range(1)))
>>> sat_comp = np.array(list(range(1)))
>>> assim = assimilate(B, H, R, mod_comp, sat_comp)
>>> assim
array([[[0.]]])
```

tools.dataassimilation.**covariance_diagonal_only**(*matrix*, *latent_space*)

Given an n x n matrix and a value for latent_space return the diagonal covariance matrix with entries ofdiagonals as 0

Parameters

- **latent_space** (*integer*) –
- **matrix** (*np.array or list of lists*) – 'n x n' array

Return type np.array 'n x n' (matrix)

Examples

```
>>> mat = np.identity(4)
>>> lat = 4
>>> cov_d = covariance_diagonal_only(mat, lat)
>>> cov_d
array([[0.25, 0.  , 0.  , 0.  ],
       [0.  , 0.25, 0.  , 0.  ],
       [0.  , 0.  , 0.25, 0.  ],
       [0.  , 0.  , 0.  , 0.25]])
```

tools.dataassimilation.**mse**(*y_obs*, *y_pred*)

Given y_obs, y_pred a numpy array of size n return the mse between y_obs and y_pred

Parameters

- **y_obs** (*np.array*) – 'n' array
- **y_pred** (*np.array*) – 'n' array

Return type mse value if the dimension of input matches

Examples

```
>>> y_obs = np.array([1,2,3,4,5])
>>> y_pred = np.array([1,2,3,9,5])
>>> mse_val = mse(y_obs, y_pred)
>>> mse_val
5.0
```

`tools.dataassimilation.`**`update_prediction`**(*x*, *K*, *H*, *y*)

Given K and H an n x n matrix and x, y a numpy array of size n and returns an array of size n with the updated prediction

Parameters

- **x** (*np.array*) – 'n' array
- **y** (*np.array*) – 'n' array
- **K** (*np.array or list of lists*) – 'n x n' array
- **H** (*np.array or list of lists*) – 'n x n' array

Return type np.array of size n

Examples

```
>>> x = np.array([1, 2, 3, 4, 5])
>>> y = np.array([1, 2, 3, 9, 5])
>>> K = np.identity(5)
>>> H = np.identity(5)
>>> updated_data = update_prediction(x, K, H, y)
>>> updated_data
array([1., 2., 3., 9., 5.])
```

CHAPTER

FOUR

VISUALISATION.PY

`tools.visualisation.create_slider`(*data*, *simulation=0*, *cmap='viridis'*)

Returns a slider holoview object which allows the user to view the progression of the wildfire spread.

Usage: Simply calling the function will return the slider plot.

*** Important to look at required data shape in data parameter below ***

Parameters

- **data** (*np.array*) – A numpy array for the wildfire data. In our case this would need to be either train, test, observation or background data. *** –Required shape– * For train and testdata: (sims, days, x, y) * For obs or background data: (days, x, y) ***
- **simulation** (*int*) – Only needs to be provided if using train or test data. It selects which simulation to run (of which there are 300 in the train data, and 75 in the test data). Defaulted to 0.
- **cmap** (*str*) – Used to specify the colour used in the wildfire plot images. This has to be a string that matches one of the values listed here: https://matplotlib.org/3.5.1/tutorials/colors/colormaps.html Default value is 'viridis' - the same is the matplotlib imshow default.

Returns

A Holomap object of the slider. Needs to be assigned to a variable and run separatelky in order to be displayed in Jupyter Notebooks - see the above usage.

Return type hmap (HoloMap object)

`tools.visualisation.plot_data`(*data_arr*)

This function produces a figure of 4 subplots, one for each day of a wildfire progression.

Usage: Simply calling the function will return the slider plot. E.g:

plot_data(data_arr)

Parameters **data_arr** (*np.array*) – A numpy array fo the wildfire data. This will need to be either the background or satellite data.

`tools.visualisation.plot_pca_variance`(*pca*, *n_components*)

Function that plots explained variance vs number of components for PCA analysis, with annotation of explaned variance for a provided number of components.

Usage: Simply calling the function with the correct args will return the PCA variance plot. E.g.:

plot_pca_variance(pca=pca, n_components=60)

Parameters

- **pca** (*PCA object*) – An sklearn PCA object. The result of using sklearn.decomposition.PCA on a fitting dataset.
- **n_components** (*int*) – The number of components for the annotation on the plot to show the explained variance for. This can be a maximum of the total number of components available in the PCA. This max can be found using **PCA.components_**.shape[0].

Returns No return variable, just the displaying of the plot.

`tools.visualisation.plot_simulation_data`(*data*, *simulation=0*, *title=''*, *sequential=False*)

This function produces a figure of 4 subplots, one for each day of a wildfire simulation. The user can specify what data set to use (train or test), and which simulation from these datasets to show.

Usage: Simply calling the function will return the slider plot. E.g:

plot_simulation_data(train_data, simulation=42)

*** Important to look at required data shape in data parameter below ***

Parameters

- **data** (*np.array*) – A numpy array fo the wildfire data. In our case this would need to be either train or test. *** –Required shape– * For train and testdata: (sims, days, x, y) ***
- **simulation** (*int*) – Selects which simulation to run (of which there are 300 in the train data, and 75 in the test data). Defaulted to 0.
- **title** (*str*) – Option to add a string to augment the title of the resultant figure
- **sequential** (*bool*) – Let the function know whether you are plotting from a dataset that has a reduced amout of days. I.e. sequential datasets that have been reshaped for model training, or have been predicted using our model.

References

PYTHON MODULE INDEX