NLMpy Reference Card

for version 0.1.1

NEUTRAL LANDSCAPE MODEL FUNCTIONS

---

random(nRow, nCol, mask=None)

Create a spatially **random** neutral landscape model with values ranging 0-1.

nRow : int
    The number of rows in the array.
nCol : int
    The number of columns in the array.
mask : array, optional
    2D array used as a binary mask to limit the elements with values.

out : array
    2D array.

---

planarGradient(nRow, nCol, direction=None, mask=None)

Create a **planar gradient** neutral landscape model with values ranging 0-1.

nRow : int
    The number of rows in the array.
nCol : int
    The number of columns in the array.
direction: int, optional
    The direction of the gradient as a bearing from north, if unspecified the direction is randomly determined.
mask : array, optional
    2D array used as a binary mask to limit the elements with values.

out : array
    2D array.

---

edgeGradient(nRow, nCol, direction=None, mask=None)

Create an **edge gradient** neutral landscape model with values ranging 0-1.

nRow : int
    The number of rows in the array.
nCol : int
    The number of columns in the array.
direction: int, optional
    The direction of the gradient as a bearing from north, if unspecified the direction is randomly determined.
mask : array, optional
    2D array used as a binary mask to limit the elements with values.

out : array
    2D array.

---

distanceGradient(source, mask=None)

Create a **distance gradient** neutral landscape model with values ranging 0-1.

source : array
    2D array binary array that defines the source elements from which distance will be measured. The dimensions of source also specify the output dimensions of the distance gradient.
mask : array, optional
    2D array used as a binary mask to limit the elements with values.

out : array
    2D array.

---

mpd(nRow, nCol, h, mask=None)

Create a **midpoint displacement** neutral landscape model with values ranging 0-1.

nRow : int
    The number of rows in the array.
nCol : int
    The number of columns in the array.
h: float
    The h value controls the level of spatial autocorrelation in element values.
mask : array, optional
    2D array used as a binary mask to limit the elements with values.

out : array
    2D array.

---

randomRectangularCluster(nRow, nCol, minL, maxL, mask=None)

Create a **random rectangular cluster** neutral landscape model with values ranging 0-1.

nRow : int
    The number of rows in the array.
nCol : int
    The number of columns in the array.
minL: int
    The minimum possible length of width and height for each random rectangular cluster.
maxL: int
    The maximum possible length of width and height for each random rectangular cluster.
mask : array, optional
    2D array used as a binary mask to limit the elements with values.

out : array
    2D array.

---

randomElementNN(nRow, nCol, n, mask=None)

Create a **random element nearest-neighbour** neutral landscape model with values ranging 0-1.

nRow : int
    The number of rows in the array.
nCol : int
    The number of columns in the array.
n: int
    The number of elements randomly selected to form the basis of nearest-neighbour clusters.
mask : array, optional
    2D array used as a binary mask to limit the elements with values.

out : array
    2D array.

---

randomClusterNN(nRow, nCol, p, n='4-neighbourhood', mask=None)

Create a **random cluster nearest-neighbour** neutral landscape model with values ranging 0-1.

nRow : int
    The number of rows in the array.
nCol : int
    The number of columns in the array.
p: float
    The p value controls the proportion of elements randomly selected to form clusters.
n: string, optional
    Clusters are defined using a set of neighbourhood structures that include:

```
                          [0,1,0]
    '4-neighbourhood' = [1,1,1] (the default)
                          [0,1,0]

                          [1,1,1]
    '8-neighbourhood' = [1,1,1]
                          [1,1,1]

                    [0,1,1]
    'diagonal' = [1,1,1]
                    [1,1,0]
```

mask : array, optional
    2D array used as a binary mask to limit the elements with values.

out : array
    2D array.

ADDITIONAL FUNCTIONS

---

linearRescale01(array):

A rescale in which the values in the array are linearly rescaled to range between 0 and 1.

array : array
    2D array of data values.

out : array
    2D array with rescaled values.

---

maskArray(array, maskArray)

Return the array with nan values inserted where present in the mask array. It is assumed that both the arrays have the same dimensions.

array : array
    2D array of data values.
maskArray : array
    2D array used as a binary mask.

out : array
    2D array with masked values.

---

randomUniform01(nRow, nCol, mask=None)

Create an array with random values ranging 0-1.

nRow : int
    The number of rows in the array.
nCol : int
    The number of columns in the array.
mask : array, optional
    2D array used as a binary mask to limit the elements with values.

out : array
    2D array of data values.

---

nnInterpolate(array, missing)

Two-dimensional array nearest-neighbour interpolation in which the elements in the positions indicated by the array "missing" are replaced by the nearest value from the "array" of data values.

array : array
    2D array of data values.
missing: boolean array
    Values of True receive interpolated values.

out : array
    2D array with interpolated values.

---

w2cp(weights)

Convert a list of category weights into a 1D NumPy array of cumulative proportions.

weights : list
    A list of numeric values

out : array
    1D array of class cumulative proportions.

---

calcBoundaries(array, cumulativeProportions, classifyMask=None)

Determine upper class boundaries for classification of an array with values ranging 0-1 based upon an array of cumulative proportions.

array : array
    2D array of data values.
cumulativeProportions : array
    1D array of class cumulative proportions.
classifyMask : array, optional
    2D array used as a binary mask to limit the elements used to determine the upper boundary values for each class.

out : array
    1D float array.

---

classifyArray(array, weights, classifyMask=None)

Classify an array with values ranging 0-1 into proportions based upon a list of class weights.

array : array
    2D array of data values.
weights : list
    A list of numeric values
classifyMask : array, optional
    2D array used as a binary mask to limit the elements used to determine the upper boundary values for each class.

out : array
    2D array.

---

blendArray(primaryArray, arrays, scalingFactors=None)

Blend a primary array with other arrays weighted by scaling factors.

primaryArray : array
    2D array of data values.
arrays : list
    List of 2D arrays of data values.
scalingFactors : list
    List of scaling factors used to weight the arrays in the blend.

out : array
    2D array.

---

blendClusterArray(primaryArray, arrays, scalingFactors=None)

Blend a primary cluster NLM with other arrays in which the mean value per cluster is weighted by scaling factors.

primaryArray : array
    2D array of data values in which values are clustered.
arrays : list
    List of 2D arrays of data values.
scalingFactors : list
    List of scaling factors used to weight the arrays in the blend.

out : array
    2D array.

---

meanOfCluster(clusterArray, array)

For each cluster of elements in an array, calculate the mean value for the cluster based on a second array.

clutserArray : array
    2D array of data values in which values are clustered.
array : array
    2D array of data values.

out : array
    2D array.

---

exportASCIIGrid(outFile, nlm, xll=0, yll=0, cellSize=1)

Export a NLM array as a ASCII grid raster file.

outFile : string
    The path and name of the output raster file.
nlm : 2D array
    The NLM to be exported.
xll : number
    Raster lower left corner x coordinate
yll : number
    Raster lower left corner y coordinate
cellSize : number
    The size of the cells in the output raster.