ANTsPy

Release dev (latest)

ANTs Contributors

Jul 26, 2026

PACKAGE REFERENCE

ANTsPy is an optimized and validated medical imaging library for Python.

The "Package Reference" highlights commonly used functionality, organized thematically.

See the "API Reference" for the full list of available modules and functions.

More resources:

- Installation instructions
- ANTsX examples
- ANTsPy GitHub for support, bug reports, and contributions.

CHAPTER

ONE

CORE

1.1 Images

1.1.1 ANTsImage

class ANTsImage(*pointer*)

Bases: `object`

abp_n4(*intensity_truncation=(0.025, 0.975, 256)*, *mask=None*, *usen3=False*)

Truncate outlier intensities and bias correct with the N4 algorithm.

ANTsR function: *abpN4*

Parameters

- **image** (`ants.core.ANTsImage`) – image to correct and truncate
- **intensity_truncation** (`typing.Tuple`) – quantiles for intensity truncation
- **mask** (`ANTsImage (optional)`) – mask for bias correction
- **usen3** (`bool`) – if True, use N3 bias correction instead of N4

Return type
`ants.core.ANTsImage`

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> image2 = ants.abp_n4(image)
```

abs(*axis=None*)

Return absolute value of image

add_noise_to_image(*noise_model*, *noise_parameters*)

Add noise to an image using additive Gaussian, salt-and-pepper, shot, or speckle noise.

Parameters

- **image** (`ants.core.ANTsImage`) – scalar image.
- **noise_model** (`str`) – 'additivegaussian', 'saltandpepper', 'shot', or 'speckle'.
- **noise_parameters** (`tuple` or `numpy.ndarray` or `float`) – 'additivegaussian': (mean, standardDeviation) 'saltandpepper': (probability, saltValue, pepperValue) 'shot': scale 'speckle': standardDeviation

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> noise_image = ants.add_noise_to_image(image, 'additivegaussian', (0.0, 1.0))
>>> noise_image = ants.add_noise_to_image(image, 'saltandpepper', (0.1, 0.0,␣
↪100.0))
>>> noise_image = ants.add_noise_to_image(image, 'shot', 1.0)
>>> noise_image = ants.add_noise_to_image(image, 'speckle', 1.0)
```

allclose(*image2*)

Check if two images have the same array values

anti_alias()

Apply Anti-Alias filter to a binary image

ANTsR function: N/A

Parameters
image (ants.core.ANTsImage) – binary image to which anti-aliasing will be applied

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> img = ants.image_read(ants.get_data('r16'))
>>> mask = ants.get_mask(img)
>>> mask_aa = ants.anti_alias(mask)
>>> ants.plot(mask)
>>> ants.plot(mask_aa)
```

apply(*fn*)

Apply an arbitrary function to ANTsImage.

Parameters
fn (python function or lambda) – function to apply to ENTIRE image at once

Returns
image with function applied to it

Return type
ants.core.ANTsImage

argmax(*axis=None*)

Return argmax along specified axis

argmin(*axis=None*)

Return argmin along specified axis

argrange(*axis=None*)

Return argrange along specified axis

astype(*dtype*)

Cast & clone an ANTsImage to a given numpy datatype.

Map:
uint8 : unsigned char uint32 : unsigned int float32 : float float64 : double

clone(*pixeltype=None*)

Create a copy of the given ANTsImage with the same data and info, possibly with a different data type for the image data. Only supports casting to uint8 (unsigned char), uint32 (unsigned int), float32 (float), and float64 (double)

Parameters
pixeltype (string (optional)) – if None, the pixeltype will be the same as the cloned ANTsImage. Otherwise, the data will be cast to this type. This can be a numpy type or an ITK type. Options:

'unsigned char' or 'uint8', 'unsigned int' or 'uint32', 'float' or 'float32', 'double' or 'float64'

Return type
ants.core.ANTsImage

property components

crop_image(*label_image=None*, *label=1*)

Use a label image to crop a smaller ANTsImage from within a larger ANTsImage

ANTsR function: *cropImage*

Parameters

- **image** (ants.core.ANTsImage) – image to crop
- **label_image** (ants.core.ANTsImage) – image with label values. If not supplied, estimated from data.
- **label** (int) – the label value to use

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> fi = ants.image_read( ants.get_ants_data('r16') )
>>> cropped = ants.crop_image(fi)
>>> fi2 = ants.merge_channels([fi,fi])
>>> cropped2 = ants.crop_image(fi2)
>>> cropped = ants.crop_image(fi, fi, 100 )
```

crop_indices(*lowerind*, *upperind*)

Create a proper ANTsImage sub-image by indexing the image with indices. This is similar to but different from array sub-setting in that the resulting sub-image can be decropped back into its place without having to store its original index locations explicitly.

ANTsR function: *cropIndices*

Parameters

- **image** (ants.core.ANTsImage) – image to crop

- **lowerind** (list/tuple of integers) – vector of lower index, should be length image dimensionality
- **upperind** (list/tuple of integers) – vector of upper index, should be length image dimensionality

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> fi = ants.image_read( ants.get_ants_data("r16"))
>>> cropped = ants.crop_indices( fi, (10,10), (100,100) )
>>> cropped = ants.smooth_image( cropped, 5 )
>>> decropped = ants.decrop_image( cropped, fi )
```

decrop_image(*full_image*)

The inverse function for *ants.crop_image*

ANTsR function: *decropImage*

Parameters

- **cropped_image** (ants.core.ANTsImage) – cropped image
- **full_image** (ants.core.ANTsImage) – image in which the cropped image will be put back

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> fi = ants.image_read(ants.get_ants_data('r16'))
>>> mask = ants.get_mask(fi)
>>> cropped = ants.crop_image(fi, mask, 1)
>>> cropped = ants.smooth_image(cropped, 1)
>>> decropped = ants.decrop_image(cropped, fi)
```

denoise_image(*mask=None*, *shrink_factor=1*, *p=1*, *r=2*, *noise_model='Rician'*, *v=0*)

Denoise an image using a spatially adaptive filter originally described in J. V. Manjon, P. Coupe, Luis Marti-Bonmati, D. L. Collins, and M. Robles. Adaptive Non-Local Means Denoising of MR Images With Spatially Varying Noise Levels, Journal of Magnetic Resonance Imaging, 31:192-203, June 2010.

ANTsR function: *denoiseImage*

Parameters

- **image** (ants.core.ANTsImage) – scalar image to denoise.
- **mask** (ants.core.ANTsImage) – to limit the denoise region.
- **shrink_factor** (scalar) – downsampling level performed within the algorithm.
- **p** (int or character of format '2x2' where the x separates vector entries) – patch radius for local sample.
- **r** (int or character of format '2x2' where the x separates vector entries) – search radius from which to choose extra local samples.

- **noise_model** (str) – 'Rician' or 'Gaussian'

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> import numpy as np
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> # add fairly large salt and pepper noise
>>> imagenoise = image + np.random.randn(*image.shape).astype('float32')*5
>>> imagedenoise = ants.denoise_image(imagenoise, ants.get_mask(image))
```

property dimension

property direction

Get image direction

Return type
tuple

property dtype

flatten()

Flatten image data

get_center_of_mass()

Compute an image center of mass in physical space which is defined as the mean of the intensity weighted voxel coordinate system.

ANTsR function: *getCenterOfMass*

Parameters
image (ants.core.ANTsImage) – image from which center of mass will be computed

Return type
scalar

Example

```
>>> fi = ants.image_read( ants.get_ants_data("r16"))
>>> com1 = ants.get_center_of_mass( fi )
>>> fi = ants.image_read( ants.get_ants_data("r64"))
>>> com2 = ants.get_center_of_mass( fi )
```

get_centroids(*clustparam=0*)

Reduces a variate/statistical/network image to a set of centroids describing the center of each stand-alone non-zero component in the image

ANTsR function: *getCentroids*

Parameters

- **image** (ants.core.ANTsImage) – image from which centroids will be calculated
- **clustparam** (int) – look at regions greater than or equal to this size

Return type
numpy.ndarray

Example

```
>>> import ants
>>> image = ants.image_read( ants.get_ants_data( "r16" ) )
>>> image = ants.threshold_image( image, 90, 120 )
>>> image = ants.label_clusters( image, 10 )
>>> cents = ants.get_centroids( image )
```

get_mask(*low_thresh=None*, *high_thresh=None*, *cleanup=2*)

Get a binary mask image from the given image after thresholding

ANTsR function: *getMask*

Parameters

- **image** (ants.core.ANTsImage) – image from which mask will be computed. Can be an antsImage of 2, 3 or 4 dimensions.
- **low_thresh** (scalar (optional)) – An inclusive lower threshold for voxels to be included in the mask. If not given, defaults to image mean.
- **high_thresh** (scalar (optional)) – An inclusive upper threshold for voxels to be included in the mask. If not given, defaults to image max
- **cleanup** (int) – If > 0, morphological operations will be applied to clean up the mask by eroding away small or weakly-connected areas, and closing holes. If cleanup is >0, the following steps are applied
  1. Erosion with radius 2 voxels
  2. Retain largest component
  3. Dilation with radius 1 voxel
  4. Morphological closing

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read( ants.get_ants_data('r16') )
>>> mask = ants.get_mask(image)
```

get_neighborhood_at_voxel(*center*, *kernel*, *physical_coordinates=False*)

Get a hypercube neighborhood at a voxel. Get the values in a local neighborhood of an image.

ANTsR function: *getNeighborhoodAtVoxel*

Parameters

- **image** (ants.core.ANTsImage) – image to get values from.
- **center** (tuple/list) – indices for neighborhood center
- **kernel** (tuple/list) – either a collection of values for neighborhood radius (in voxels) or a binary collection of the same dimension as the image, specifying the shape of the neighborhood to extract
- **physical_coordinates** (bool) – whether voxel indices and offsets should be in voxel or physical coordinates

Returns

values
[ndarray] array of neighborhood values at the voxel

indices
[ndarray] matrix providing the coordinates for each value

Return type
`dictionary w/ following key-value pairs`

Example

```
>>> import ants
>>> img = ants.image_read(ants.get_ants_data('r16'))
>>> center = (2,2)
>>> radius = (3,3)
>>> retval = ants.get_neighborhood_at_voxel(img, center, radius)
```

get_neighborhood_in_mask(*mask*, *radius*, *physical_coordinates=False*, *boundary_condition=None*, *spatial_info=False*, *get_gradient=False*)

Get neighborhoods for voxels within mask.

This converts a scalar image to a matrix with rows that contain neighbors around a center voxel

ANTsR function: *getNeighborhoodInMask*

Parameters

- **image** (`ants.core.ANTsImage`) – image to get values from
- **mask** (`ants.core.ANTsImage`) – image indicating which voxels to examine. Each voxel > 0 will be used as the center of a neighborhood
- **radius** (`tuple/list`) – array of values for neighborhood radius (in voxels)
- **physical_coordinates** (`bool`) – whether voxel indices and offsets should be in voxel or physical coordinates
- **boundary_condition** (`string (optional)`) –

  how to handle voxels in a neighborhood, but not in the mask.
  None : fill values with *NaN* *image* : use image value, even if not in mask *mean* : use mean of all non-NaN values for that neighborhood
- **spatial_info** (`bool`) – whether voxel locations and neighborhood offsets should be returned along with pixel values.
- **get_gradient** (`bool`) – whether a matrix of gradients (at the center voxel) should be returned in addition to the value matrix (WIP)

Returns

- `if spatial_info is False` –

  if get_gradient is False:

  ndarray
  an array of pixel values where the number of rows is the size of the neighborhood and there is a column for each voxel

  else if get_gradient is True:

  dictionary w/ following key-value pairs:

values
[ndarray] array of pixel values where the number of rows is the size of the neighborhood and there is a column for each voxel.

gradients
[ndarray] array providing the gradients at the center voxel of each neighborhood

- `else if spatial_info is True` –

dictionary w/ following key-value pairs:

values
[ndarray] array of pixel values where the number of rows is the size of the neighborhood and there is a column for each voxel.

indices
[ndarray] array provinding the center coordinates for each neighborhood

offsets
[ndarray] array providing the offsets from center for each voxel in a neighborhood

Example

```
>>> import ants
>>> r16 = ants.image_read(ants.get_ants_data('r16'))
>>> mask = ants.get_mask(r16)
>>> mat = ants.get_neighborhood_in_mask(r16, mask, radius=(2,2))
```

get_orientation()

property has_components

hausdorff_distance(*image2*)

Get Hausdorff distance between non-zero pixels in two images

ANTsR function: *hausdorffDistance*

Parameters

- **image** (*source*) – Source image
- **target_image** (`ants.core.ANTsImage`) – Target image

Return type
data frame with `"Distance"` and `"AverageDistance"`

Example

```
>>> import ants
>>> r16 = ants.image_read( ants.get_ants_data('r16') )
>>> r64 = ants.image_read( ants.get_ants_data('r64') )
>>> s16 = ants.kmeans_segmentation( r16, 3 )['segmentation']
>>> s64 = ants.kmeans_segmentation( r64, 3 )['segmentation']
>>> stats = ants.hausdorff_distance(s16, s64)
```

hessian_objectness(*object_dimension=1*, *is_bright_object=True*, *sigma_min=0.1*, *sigma_max=10*, *number_of_sigma_steps=10*, *use_sigma_logarithmic_spacing=True*, *alpha=0.5*, *beta=0.5*, *gamma=5.0*, *set_scale_objectness_measure=True*)

Interface to ITK filter. Based on the paper by Westin et al., "Geometrical Diffusion Measures for MRI from Tensor Basis Analysis" and Luca Antiga's Insight Journal paper http://hdl.handle.net/1926/576.

Parameters

- **image** (ants.core.ANTsImage) – scalar image.
- **object_dimension** (unsigned int) – 0: 'sphere', 1: 'line', or 2: 'plane'.
- **is_bright_object** (bool) – Set 'true' for enhancing bright objects and 'false' for dark objects.
- **sigma_min** (float) – Define scale domain for feature extraction.
- **sigma_max** (float) – Define scale domain for feature extraction.
- **number_of_sigma_steps** (unsigned int) – Define number of samples for scale space.
- **use_sigma_logarithmic_spacing** (bool) – Define sample spacing the for scale space.
- **alpha** (float) – Hessian filter parameter.
- **beta** (float) – Hessian filter parameter.
- **gamma** (float) – Hessian filter parameter.
- **set_scale_objectness_measure** (bool) – ...

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> hessian_object_image = ants.hessian_objectness(image)
```

histogram_equalize_image(*number_of_histogram_bins=256*)

Histogram equalize image

from http://www.janeriksolem.net/histogram-equalization-with-python-and.html

Parameters

- **image** (ants.core.ANTsImage) – source image
- **number_of_histogram_bins** (int) – number of bins for cumulative histogram

Example

```
>>> import ants
>>> src_img = ants.image_read(ants.get_data('r16'))
>>> src_img_eq = ants.histogram_equalize_image(src_img)
```

histogram_match_image(*reference_image*, *number_of_histogram_bins=255*, *number_of_match_points=64*, *use_threshold_at_mean_intensity=False*)

Histogram match source image to reference image.

Parameters

- **source_image** (ants.core.ANTsImage) – source image
- **reference_image** (ants.core.ANTsImage) – reference image

- **number_of_histogram_bins** (int) – number of bins for source and reference histograms
- **number_of_match_points** (int) – number of points for histogram matching
- **use_threshold_at_mean_intensity** (bool) – see ITK description.

Example

```
>>> import ants
>>> src_img = ants.image_read(ants.get_data('r16'))
>>> ref_img = ants.image_read(ants.get_data('r64'))
>>> src_ref = ants.histogram_match_image(src_img, ref_img)
```

histogram_match_image2(*reference_image*, *source_mask=None*, *reference_mask=None*, *match_points=64*, *transform_domain_size=255*)

Transform image intensities based on histogram mapping.

Apply B-spline 1-D maps to an input image for intensity warping.

Parameters

- **source_image** (ants.core.ANTsImage) – source image
- **reference_image** (ants.core.ANTsImage) – reference image
- **source_mask** (ants.core.ANTsImage) – source mask
- **reference_mask** (ants.core.ANTsImage) – reference mask
- **match_points** (int or tuple) – Parametric points at which the intensity transform displacements are specified between [0, 1], i.e. quantiles. Alternatively, a single number can be given and the sequence is linearly spaced in [0, 1].
- **transform_domain_size** (int) – Defines the sampling resolution of the B-spline warping.

Return type
ANTs image

Example

```
>>> import ants
>>> src_img = ants.image_read(ants.get_data('r16'))
>>> ref_img = ants.image_read(ants.get_data('r64'))
>>> src_ref = ants.histogram_match_image2(src_img, ref_img)
```

iMath(*operation*, **args*)

Perform various (often mathematical) operations on the input image/s. Additional parameters should be specific for each operation. See the the full iMath in ANTs, on which this function is based.

ANTsR function: *iMath*

Parameters

- **image** (ants.core.ANTsImage) – input object, usually antsImage
- **operation** – a string e.g. "GetLargestComponent" ... the special case of "GetOperations" or "GetOperationsFull" will return a list of operations and brief description. Some operations may not be valid (WIP), but most are.

- ***args** (non-keyword arguments) – additional parameters specific to the operation

Example

```
>>> import ants
>>> img = ants.image_read(ants.get_ants_data('r16'))
>>> img2 = ants.iMath(img, 'Canny', 1, 5, 12)
```

iMath_GC(*radius=1*)

iMath_GD(*radius=1*)

iMath_GE(*radius=1*)

iMath_GO(*radius=1*)

iMath_MC(*radius=1*, *value=1*, *shape=1*, *parametric=False*, *lines=3*, *thickness=1*, *include_center=False*)

iMath_MD(*radius=1*, *value=1*, *shape=1*, *parametric=False*, *lines=3*, *thickness=1*, *include_center=False*)

iMath_ME(*radius=1*, *value=1*, *shape=1*, *parametric=False*, *lines=3*, *thickness=1*, *include_center=False*)

iMath_MO(*radius=1*, *value=1*, *shape=1*, *parametric=False*, *lines=3*, *thickness=1*, *include_center=False*)

iMath_canny(*sigma*, *lower*, *upper*)

iMath_fill_holes(*hole_type=2*)

iMath_get_largest_component(*min_size=50*)

iMath_grad(*sigma=0.5*, *normalize=False*)

iMath_histogram_equalization(*alpha*, *beta*)

iMath_laplacian(*sigma=0.5*, *normalize=False*)

iMath_maurer_distance(*foreground=1*)

iMath_normalize()

iMath_pad(*padding*)

iMath_perona_malik(*conductance=0.25*, *n_iterations=1*)

iMath_propagate_labels_through_mask(*labels*, *stopping_value=100*, *propagation_method=0*)

```
>>> import ants
>>> wms = ants.image_read('~/desktop/wms.nii.gz')
>>> thal = ants.image_read('~/desktop/thal.nii.gz')
>>> img2 = ants.iMath_propagate_labels_through_mask(wms, thal, 500, 0)
```

iMath_sharpen()

iMath_truncate_intensity(*lower_q*, *upper_q*, *n_bins=64*)

```
>>> import ants
>>> img = ants.image_read(ants.get_ants_data('r16'))
>>> ants.iMath_truncate_intensity( img, 0.2, 0.8 )
```

image_mutual_information(*image2*)

Compute mutual information between two ANTsImage types

ANTsR function: *antsImageMutualInformation*

Parameters

- **image1** (ants.core.ANTsImage) – image 1
- **image2** (ants.core.ANTsImage) – image 2

Return type
scalar

Example

```
>>> import ants
>>> fi = ants.image_read( ants.get_ants_data('r16') ).clone('float')
>>> mi = ants.image_read( ants.get_ants_data('r64') ).clone('float')
>>> mival = ants.image_mutual_information(fi, mi) # -0.1796141
```

image_physical_space_consistency(*image2*, *tolerance=0.01*, *datatype=False*)

Check if two or more ANTsImage objects occupy the same physical space

ANTsR function: *antsImagePhysicalSpaceConsistency*

Parameters

- ***images** (ANTsImages) – images to compare
- **tolerance** (float) – tolerance when checking origin and spacing
- **data_type** (bool) – If true, also check that the image data types are the same

Returns
true if images share same physical space, false otherwise

Return type
bool

image_similarity(*moving_image*, *metric_type='MeanSquares'*, *fixed_mask=None*, *moving_mask=None*, *sampling_strategy='regular'*, *sampling_percentage=1.0*)

Measure similarity between two images. NOTE: Similarity is actually returned as distance (i.e. dissimilarity) per ITK/ANTs convention. E.g. using Correlation metric, the similarity of an image with itself returns -1.

ANTsR function: *imageSimilarity*

Parameters

- **fixed** (ants.core.ANTsImage) – the fixed image
- **moving** (ants.core.ANTsImage) – the moving image
- **metric_type** (str) –

  image metric to calculate
  MeanSquares Correlation ANTSNeighborhoodCorrelation MattesMutualInformation JointHistogramMutualInformation Demons

- **fixed_mask** (ANTsImage (optional)) – mask for the fixed image
- **moving_mask** (ANTsImage (optional)) – mask for the moving image

- **sampling_strategy** (string (optional)) –

  sampling strategy, default is full sampling
  None (Full sampling) random regular

- **sampling_percentage** (scalar) – percentage of data to sample when calculating metric Must be between 0 and 1

Return type
scalar

Example

```
>>> import ants
>>> x = ants.image_read(ants.get_ants_data('r16'))
>>> y = ants.image_read(ants.get_ants_data('r30'))
>>> metric = ants.image_similarity(x,y,metric_type='MeanSquares')
```

image_to_cluster_images(*min_cluster_size=50*, *min_thresh=1e-06*, *max_thresh=1*)

Converts an image to several independent images.

Produces a unique image for each connected component 1 through N of size > min_cluster_size

ANTsR function: *image2ClusterImages*

Parameters

- **image** (ants.core.ANTsImage) – input image
- **min_cluster_size** (int) – throw away clusters smaller than this value
- **min_thresh** (scalar) – threshold to a statistical map
- **max_thresh** (scalar) – threshold to a statistical map

Return type
list of ANTsImage types

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> image = ants.threshold_image(image, 1, 1e15)
>>> image_cluster_list = ants.image_to_cluster_images(image)
```

image_write(*filename*, *ri=False*)

Write an ANTsImage to file

ANTsR function: *antsImageWrite*

Parameters

- **image** (ants.core.ANTsImage) – image to save to file
- **filename** (str) – name of file to which image will be saved
- **ri** (bool) –

  if True, return image. This allows for using this function in a pipeline:

```
>>> img2 = img.smooth_image(2.).image_write(file1,␣
↪ri=True).threshold_image(0,20).image_write(file2,␣
↪ri=True)
```

if False, do not return image

property is_rgb

label_clusters(*min_cluster_size=50*, *min_thresh=1e-06*, *max_thresh=1*, *fully_connected=False*)

This will give a unique ID to each connected component 1 through N of size > min_cluster_size

ANTsR function: *labelClusters*

Parameters

- **image** (ants.core.ANTsImage) – input image e.g. a statistical map
- **min_cluster_size** (int) – throw away clusters smaller than this value
- **min_thresh** (scalar) – threshold to a statistical map
- **max_thresh** (scalar) – threshold to a statistical map
- **fully_connected** (bool) – boolean sets neighborhood connectivity pattern

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read( ants.get_ants_data('r16') )
>>> timageFully = ants.label_clusters( image, 10, 128, 150, True )
>>> timageFace = ants.label_clusters( image, 10, 128, 150, False )
```

label_geometry_measures(*intensity_image=None*)

Wrapper for the ANTs funtion LabelGeometryMeasures

ANTsR function: *labelGeometryMeasures*

Parameters

- **label_image** (ants.core.ANTsImage) – image on which to compute geometry. Labels must be representable as uint32.
- **intensity_image** (ANTsImage (optional)) – image with intensity values

Return type
pandas.DataFrame

Example

```
>>> import ants
>>> fi = ants.image_read( ants.get_ants_data('r16') )
>>> seg = ants.kmeans_segmentation( fi, 3 )['segmentation']
>>> geom = ants.label_geometry_measures(seg,fi)
```

label_image_centroids(*physical=False*, *convex=True*, *verbose=False*)

Converts a label image to coordinates summarizing their positions

ANTsR function: *labelImageCentroids*

Parameters

- **image** (ants.core.ANTsImage) – image of integer labels
- **physical** (bool) – whether you want physical space coordinates or not
- **convex** (bool) – if True, return centroid if False return point with min average distance to other points with same label

Returns

labels
[1D-ndarray] array of label values

vertices
[pd.DataFrame] coordinates of label centroids

Return type
dictionary w/ following key-value pairs

Example

```
>>> import ants
>>> import numpy as np
>>> image = ants.from_numpy(np.asarray([[[0,2],[1,3]],[[4,6],[5,7]]]).astype(
↪'float32'))
>>> labels = ants.label_image_centroids(image)
```

label_overlap_measures(*target_image*)

Get overlap measures from two label images (e.g., Dice)

ANTsR function: *labelOverlapMeasures*

Parameters

- **image** (*source*) – Source image
- **target_image** (ants.core.ANTsImage) – Target image

Return type
data frame with measures for each label and all labels combined

Example

```
>>> import ants
>>> r16 = ants.image_read( ants.get_ants_data('r16') )
>>> r64 = ants.image_read( ants.get_ants_data('r64') )
>>> s16 = ants.kmeans_segmentation( r16, 3 )['segmentation']
>>> s64 = ants.kmeans_segmentation( r64, 3 )['segmentation']
>>> stats = ants.label_overlap_measures(s16, s64)
```

label_stats(*label_image*)

Get label statistics from an image. The labels must be representable as uint32.

ANTsR function: *labelStats*

Parameters

- **image** (ants.core.ANTsImage) – Image from which statistics will be calculated
- **label_image** (ants.core.ANTsImage) – Label image

Return type
pandas.DataFrame

Example

```
>>> import ants
>>> image = ants.image_read( ants.get_ants_data('r16') , 2 )
>>> image = ants.resample_image( image, (64,64), 1, 0 )
>>> mask = ants.get_mask(image)
>>> segs1 = ants.kmeans_segmentation( image, 3 )
>>> stats = ants.label_stats(image, segs1['segmentation'])
```

labels_to_matrix(*mask*, *target_labels=None*, *missing_val=nan*)

Convert a labeled image to an n x m binary matrix where n = number of voxels and m = number of labels. Only includes values inside the provided mask while including background (image == 0) for consistency with timeseries2matrix and other image to matrix operations.

ANTsR function: *labels2matrix*

Parameters

- **image** (ants.core.ANTsImage) – input label image
- **mask** (ants.core.ANTsImage) – defines domain of interest
- **target_labels** (list/tuple) – defines target regions to be returned. if the target label does not exist in the input label image, then the matrix will contain a constant value of missing_val (default None) in that row.
- **missing_val** (scalar) – value to use for missing label values

Return type
numpy.ndarray

Example

```
>>> import ants
>>> fi = ants.image_read(ants.get_ants_data('r16')).resample_image((60,60),1,0)
>>> mask = ants.get_mask(fi)
>>> labs = ants.kmeans_segmentation(fi,3)['segmentation']
>>> labmat = ants.labels_to_matrix(labs, mask)
```

list_to_ndimage(*image_list*)

Merge list of multiple scalar ANTsImage types of dimension into one ANTsImage of dimension plus one

ANTsR function: *mergeListToNDImage*

Parameters

- **image** (target image space) –
- **image_list** (list/tuple of ANTsImage types) – scalar images to merge into target image space

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> image2 = ants.image_read(ants.get_ants_data('r16'))
>>> imageTar = ants.make_image( ( *image2.shape, 2 ) )
>>> image3 = ants.list_to_ndimage( imageTar, [image,image2])
>>> image3.dimension == 3
```

mask_image(*mask*, *level=1*, *binarize=False*)

Mask an input image by a mask image. If the mask image has multiple labels, it is possible to specify which label(s) to mask at.

ANTsR function: *maskImage*

Parameters

- **image** (ants.core.ANTsImage) – Input image.
- **mask** (ants.core.ANTsImage) – Mask or label image.
- **level** (scalar or tuple of scalars) – Level(s) at which to threshold the mask. Can be a single value or an iterable of values.
- **binarize** (bool) – whether to binarize the output image. This computes an intersection between (image != 0) and (mask == i), for all i in level.

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> myimage = ants.image_read(ants.get_ants_data('r16'))
>>> mask = ants.get_mask(myimage)
>>> myimage_mask = ants.mask_image(myimage, mask, 3)
>>> seg = ants.kmeans_segmentation(myimage, 3)
>>> myimage_mask = ants.mask_image(myimage, seg['segmentation'], (1,3))
```

matrix_to_timeseries(*matrix*, *mask=None*)

converts a matrix to a ND image.

ANTsR function: *matrix2timeseries*

Parameters

- **image** (reference ND image) –
- **matrix** (matrix to convert to image) –
- **mask** (mask image defining voxels of interest) –

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> img = ants.make_image( (10,10,10,5 ) )
```

(continues on next page)

(continued from previous page)

```
>>> mask = ants.ndimage_to_list( img )[0] * 0
>>> mask[ 4:8, 4:8, 4:8 ] = 1
>>> mat = ants.timeseries_to_matrix( img, mask = mask )
>>> img2 = ants.matrix_to_timeseries( img,  mat, mask)
```

max(*axis=None*)

Return max along specified axis

mean(*axis=None*)

Return mean along specified axis

median(*axis=None*)

Return median along specified axis

min(*axis=None*)

Return min along specified axis

morphology(*operation*, *radius*, *mtype='binary'*, *value=1*, *shape='ball'*, *radius_is_parametric=False*, *thickness=1*, *lines=3*, *include_center=False*)

Apply morphological operations to an image

ANTsR function: *morphology*

Parameters

- **input** (ants.core.ANTsImage) – input image
- **operation** (str) –

  operation to apply
  ”close” Morpholgical closing “dilate” Morpholgical dilation “erode” Morphological erosion “open” Morpholgical opening
- **radius** (scalar) – radius of structuring element
- **mtype** (str) –

  type of morphology
  ”binary” Binary operation on a single value “grayscale” Grayscale operations
- **value** (scalar) – value to operation on (type=’binary’ only)
- **shape** (str) –

  shape of the structuring element (type=’binary’ only)
  ”ball” spherical structuring element “box” box shaped structuring element “cross” cross shaped structuring element “annulus” annulus shaped structuring element “polygon” polygon structuring element
- **radius_is_parametric** (bool) – used parametric radius boolean (shape=’ball’ and shape=’annulus’ only)
- **thickness** (scalar) – thickness (shape=’annulus’ only)
- **lines** (int) – number of lines in polygon (shape=’polygon’ only)
- **include_center** (bool) – include center of annulus boolean (shape=’annulus’ only)

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> fi = ants.image_read( ants.get_ants_data('r16') , 2 )
>>> mask = ants.get_mask( fi )
>>> dilated_ball = ants.morphology( mask, operation='dilate', radius=3, mtype=
↪'binary', shape='ball')
>>> eroded_box = ants.morphology( mask, operation='erode', radius=3, mtype=
↪'binary', shape='box')
>>> opened_annulus = ants.morphology( mask, operation='open', radius=5, mtype=
↪'binary', shape='annulus', thickness=2)
```

movie(*filename=None*, *writer=None*, *fps=30*)

Create and save a movie - mp4, gif, etc - of the various 2D slices of a 3D ants image

Try this:

conda install -c conda-forge ffmpeg

Example

```
>>> import ants
>>> mni = ants.image_read(ants.get_data('mni'))
>>> ants.movie(mni, filename='~/desktop/movie.mp4')
```

multi_label_morphology(*operation*, *radius*, *dilation_mask=None*, *label_list=None*, *force=False*)

Morphology on multi label images.

Wraps calls to iMath binary morphology. Additionally, dilation and closing operations preserve pre-existing labels. The choices of operation are:

Dilation: dilates all labels sequentially, but does not overwrite original labels. This reduces dependence on the intensity ordering of adjoining labels. Ordering dependence can still arise if two or more labels dilate into the same space - in this case, the label with the lowest intensity is retained. With a mask, dilated labels are multiplied by the mask and then added to the original label, thus restricting dilation to the mask region.

Erosion: Erodes labels independently, equivalent to calling iMath iteratively.

Closing: Close holes in each label sequentially, but does not overwrite original labels.

Opening: Opens each label independently, equivalent to calling iMath iteratively.

Parameters

- **image** (`ants.core.ANTsImage`) – Input image should contain only 0 for background and positive integers for labels.
- **operation** (`str`) – One of MD, ME, MC, MO, passed to iMath.
- **radius** (`int`) – radius of the morphological operation.
- **dilation_mask** (`ants.core.ANTsImage`) – Optional binary mask to constrain dilation only (eg dilate cortical label into WM).
- **label_list** (`list` or `tuple` or `numpy.ndarray`) – Optional list of labels, to perform operation upon. Defaults to all unique intensities in image.

Return type

`ants.core.ANTsImage`

Example

```
>>> import ants
>>> img = ants.image_read(ants.get_data('r16'))
>>> labels = ants.get_mask(img,1,150) + ants.get_mask(img,151,225) * 2
>>> labels_dilated = ants.multi_label_morphology(labels, 'MD', 2)
>>> # should see original label regions preserved in dilated version
>>> # label N should have mean N and 0 variance
>>> print(ants.label_stats(labels_dilated, labels))
```

multiply_images(*image2*)

n3_bias_field_correction(*downsample_factor=3*)

N3 Bias Field Correction

ANTsR function: *n3BiasFieldCorrection*

Parameters

- **image** (ants.core.ANTsImage) – image to be bias corrected
- **downsample_factor** (scalar) – how much to downsample image before performing bias correction

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read( ants.get_ants_data('r16') )
>>> image_n3 = ants.n3_bias_field_correction(image)
```

n3_bias_field_correction2(*mask=None*, *rescale_intensities=False*, *shrink_factor=4*, *convergence={'iters': 50, 'tol': 1e-07}*, *spline_param=None*, *number_of_fitting_levels=4*, *return_bias_field=False*, *verbose=False*, *weight_mask=None*)

N3 Bias Field Correction

ANTsR function: *n3BiasFieldCorrection2*

Parameters

- **image** (ants.core.ANTsImage) – image to bias correct
- **mask** (ants.core.ANTsImage) – Input mask. If not specified, the entire image is used.
- **rescale_intensities** (bool) – At each iteration, a new intensity mapping is calculated and applied but there is nothing which constrains the new intensity range to be within certain values. The result is that the range can "drift" from the original at each iteration. This option rescales to the [min,max] range of the original image intensities within the user-specified mask. A mask is required to perform rescaling. Default is False in ANTsR/ANTsPy but True in ANTs.
- **shrink_factor** (scalar) – Shrink factor for multi-resolution correction, typically integer less than 4

- **convergence** (dict w/ keys `iters` and *tol*) – iters : maximum number of iterations tol : the convergence tolerance. Default tolerance is 1e-7 in ANTsR/ANTsPy but 0.0 in ANTs.
- **spline_param** (float or vector Parameter controlling number of control) – points in spline. Either single value, indicating the spacing in each direction, or vector with one entry per dimension of image, indicating the mesh size. If None, defaults to mesh size of 1 in all dimensions.
- **number_of_fitting_levels** (int) – Number of fitting levels per iteration.
- **return_bias_field** (bool) – Return bias field instead of bias corrected image.
- **verbose** (bool) – enables verbose output.
- **weight_mask** (ANTsImage (optional)) – antsImage of weight mask

Return type
ants.core.ANTsImage

Example

```
>>> image = ants.image_read( ants.get_ants_data('r16') )
>>> image_n3 = ants.n3_bias_field_correction2(image)
```

n4_bias_field_correction(*mask=None*, *rescale_intensities=False*, *shrink_factor=4*, *convergence={'iters': [50, 50, 50, 50], 'tol': 1e-07}*, *spline_param=None*, *return_bias_field=False*, *verbose=False*, *weight_mask=None*)

N4 Bias Field Correction

ANTsR function: *n4BiasFieldCorrection*

Parameters

- **image** (ants.core.ANTsImage) – image to bias correct
- **mask** (ants.core.ANTsImage) – Input mask. If not specified, the entire image is used.
- **rescale_intensities** (bool) – At each iteration, a new intensity mapping is calculated and applied but there is nothing which constrains the new intensity range to be within certain values. The result is that the range can "drift" from the original at each iteration. This option rescales to the [min,max] range of the original image intensities within the user-specified mask. A mask is required to perform rescaling. Default is False in ANTsR/ANTsPy but True in ANTs.
- **shrink_factor** (scalar) – Shrink factor for multi-resolution correction, typically integer less than 4
- **convergence** (dict w/ keys `iters` and *tol*) – iters : vector of maximum number of iterations for each level tol : the convergence tolerance. Default tolerance is 1e-7 in ANTsR/ANTsPy but 0.0 in ANTs.
- **spline_param** (float or vector) – Parameter controlling number of control points in spline. Either single value, indicating the spacing in each direction, or vector with one entry per dimension of image, indicating the mesh size. If None, defaults to mesh size of 1 in all dimensions.
- **return_bias_field** (bool) – Return bias field instead of bias corrected image.
- **verbose** (bool) – enables verbose output.

- **weight_mask** (ANTsImage (optional)) – antsImage of weight mask

Return type
ants.core.ANTsImage

Example

```
>>> image = ants.image_read( ants.get_ants_data('r16') )
>>> image_n4 = ants.n4_bias_field_correction(image)
```

ndimage_to_list()

Split a n dimensional ANTsImage into a list of n-1 dimensional ANTsImages

Parameters
image (ants.core.ANTsImage) – n-dimensional image to split

Return type
list of ANTsImage types

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> image2 = ants.image_read(ants.get_ants_data('r16'))
>>> imageTar = ants.make_image( ( *image2.shape, 2 ) )
>>> image3 = ants.list_to_ndimage( imageTar, [image,image2])
>>> image3.dimension == 3
>>> images_unmerged = ants.ndimage_to_list( image3 )
>>> len(images_unmerged) == 2
>>> images_unmerged[0].dimension == 2
```

new_image_like(*data*)

Create a new ANTsImage with the same header information, but with a new image array.

Parameters
data (numpy.ndarray or py::capsule) – New array or pointer for the image. It must have the same shape as the current image data.

Return type
ants.core.ANTsImage

nonzero()

Return non-zero indices of image

numpy(*single_components=False*)

Get a numpy array copy representing the underlying image data. Altering this ndarray will have NO effect on the underlying image data.

Parameters
single_components (boolean (default is False)) – if True, keep the extra component dimension in returned array even if image only has one component (i.e. self.has_components == False)

Return type
numpy.ndarray

property orientation

property origin

Get image origin

Return type
tuple

pad_image(*shape=None*, *pad_width=None*, *value=0.0*, *return_padvals=False*)

Pad an image to have the given shape or to be isotropic.

Parameters

- **image** (ants.core.ANTsImage) – image to pad
- **shape** (tuple) –
  - if shape is given, the image will be padded in each dimension until it has this shape
  - if shape and pad_width are both None, the image will be padded along each dimension to match the largest existing dimension so that it has isotropic dimensions.
- **pad_width** (list of integers or list-of-list of integers) – How much to pad in each direction. If a single list is supplied (e.g., [4,4,4]), then the image will be padded by half that amount on both sides. If a list-of-list is supplied (e.g., [(0,4),(0,4),(0,4)]), then the image will be padded unevenly on the different sides
- **pad_value** (scalar) – value with which image will be padded

Example

```
>>> import ants
>>> img = ants.image_read(ants.get_data('r16'))
>>> img2 = ants.pad_image(img, shape=(300,300))
>>> mni = ants.image_read(ants.get_data('mni'))
>>> mni2 = ants.pad_image(mni)
>>> mni3 = ants.pad_image(mni, pad_width=[(0,4),(0,4),(0,4)])
>>> mni4 = ants.pad_image(mni, pad_width=(4,4,4))
```

property physical_shape

property pixeltype

plot(*overlay=None*, *blend=False*, *alpha=1*, *cmap='Greys_r'*, *overlay_cmap='turbo'*, *overlay_alpha=0.9*, *vminol=None*, *vmaxol=None*, *cbar=False*, *cbar_length=0.8*, *cbar_dx=0.0*, *cbar_vertical=True*, *axis=0*, *nslices=12*, *slices=None*, *ncol=None*, *slice_buffer=None*, *black_bg=True*, *bg_thresh_quant=0.01*, *bg_val_quant=0.99*, *domain_image_map=None*, *crop=False*, *scale=False*, *reverse=False*, *title=None*, *title_fontsize=20*, *title_dx=0.0*, *title_dy=0.0*, *filename=None*, *dpi=500*, *figsize=1.5*, *reorient=True*, *resample=True*)

Plot an ANTsImage.

Use mask_image and/or threshold_image to preprocess images to be be overlaid and display the overlays in a given range. See the wiki examples.

By default, images will be reoriented to 'LAI' orientation before plotting. So, if axis == 0, the images will be ordered from the left side of the brain to the right side of the brain. If axis == 1, the images will be ordered from the anterior (front) of the brain to the posterior (back) of the brain. And if axis == 2, the images will be ordered from the inferior (bottom) of the brain to the superior (top) of the brain.

ANTsR function: *plot.antsImage*

Parameters

- **image** (ants.core.ANTsImage) – image to plot
- **overlay** (ants.core.ANTsImage) – image to overlay on base image
- **cmap** (str) – colormap to use for base image. See matplotlib.
- **overlay_cmap** (str) – colormap to use for overlay images, if applicable. See matplotlib.
- **overlay_alpha** (float) – level of transparency for any overlays. Smaller value means the overlay is more transparent. See matplotlib.
- **axis** (int) – which axis to plot along if image is 3D
- **nslices** (int) – number of slices to plot if image is 3D
- **slices** (list or tuple of integers) – specific slice indices to plot if image is 3D. If given, this will override *nslices*. This can be absolute array indices (e.g. (80,100,120)), or this can be relative array indices (e.g. (0.4,0.5,0.6))
- **ncol** (int) – Number of columns to have on the plot if image is 3D.
- **slice_buffer** (int) – how many slices to buffer when finding the non-zero slices of a 3D images. So, if slice_buffer = 10, then the first slice in a 3D image will be the first non-zero slice index plus 10 more slices.
- **black_bg** (bool) – if True, the background of the image(s) will be black. if False, the background of the image(s) will be determined by the

  values *bg_thresh_quant* and *bg_val_quant*.
- **bg_thresh_quant** (float) – if white_bg=True, the background will be determined by thresholding the image at the *bg_thresh* quantile value and setting the background intensity to the *bg_val* quantile value. This value should be in [0, 1] - somewhere around 0.01 is recommended.
  - equal to 1 will threshold the entire image
  - equal to 0 will threshold none of the image
- **bg_val_quant** (float) – if white_bg=True, the background will be determined by thresholding the image at the *bg_thresh* quantile value and setting the background intensity to the *bg_val* quantile value. This value should be in [0, 1]
  - equal to 1 is pure white
  - equal to 0 is pure black
  - somewhere in between is gray
- **domain_image_map** (ants.core.ANTsImage) – this input ANTsImage or list of ANTsImage types contains a reference image *domain_image* and optional reference mapping named *domainMap*. If supplied, the image(s) to be plotted will be mapped to the domain image space before plotting - useful for non-standard image orientations.
- **crop** (bool) – if true, the image(s) will be cropped to their bounding boxes, resulting in a potentially smaller image size. if false, the image(s) will not be cropped
- **scale** (bool or typing.Tuple) – if true, nothing will happen to intensities of image(s) and overlay(s) if false, dynamic range will be maximized when visualizing overlays if 2-tuple, the image will be dynamically scaled between these quantiles

- **reverse** (bool) – if true, the order in which the slices are plotted will be reversed. This is useful if you want to plot from the front of the brain first to the back of the brain, or vice-versa
- **title** (str) – add a title to the plot
- **filename** (str) – if given, the resulting image will be saved to this file
- **dpi** (int) – determines resolution of image if saved to file. Higher values result in higher resolution images, but at a cost of having a larger file size
- **resample** (bool) – if true, resample image if spacing is very unbalanced.

Example

```
>>> import ants
>>> import numpy as np
>>> img = ants.image_read(ants.get_data('r16'))
>>> segs = img.kmeans_segmentation(k=3)['segmentation']
>>> ants.plot(img, segs*(segs==1), crop=True)
>>> ants.plot(img, segs*(segs==1), crop=False)
>>> mni = ants.image_read(ants.get_data('mni'))
>>> segs = mni.kmeans_segmentation(k=3)['segmentation']
>>> ants.plot(mni, segs*(segs==1), crop=False)
```

plot_hist(*threshold=0.0*, *fit_line=False*, *normfreq=True*, *title=None*, *grid=True*, *xlabel=None*, *ylabel=None*, *facecolor='green'*, *alpha=0.75*)

Plot a histogram from an ANTsImage

Parameters

image (ants.core.ANTsImage) – image from which histogram will be created

plot_ortho(*overlay=None*, *reorient=True*, *blend=False*, *xyz=None*, *xyz_lines=True*, *xyz_color='red'*, *xyz_alpha=0.6*, *xyz_linewidth=2*, *xyz_pad=5*, *orient_labels=True*, *alpha=1*, *cmap='Greys_r'*, *overlay_cmap='jet'*, *overlay_alpha=0.9*, *cbar=False*, *cbar_length=0.8*, *cbar_dx=0.0*, *cbar_vertical=True*, *black_bg=True*, *bg_thresh_quant=0.01*, *bg_val_quant=0.99*, *crop=False*, *scale=False*, *domain_image_map=None*, *title=None*, *titlefontsize=24*, *title_dx=0*, *title_dy=0*, *text=None*, *textfontsize=24*, *textfontcolor='white'*, *text_dx=0*, *text_dy=0*, *filename=None*, *dpi=500*, *figsize=1.0*, *flat=False*, *transparent=True*, *resample=False*, *allow_xyz_change=True*)

Plot an orthographic view of a 3D image

Use mask_image and/or threshold_image to preprocess images to be be overlaid and display the overlays in a given range. See the wiki examples.

ANTsR function: N/A

image
[ANTsImage] image to plot

overlay
[ANTsImage] image to overlay on base image

xyz
[list or tuple of 3 integers] selects index location on which to center display if given, solid lines will be drawn to converge at this coordinate. This is useful for pinpointing a specific location in the image.

flat
[boolean] if true, the ortho image will be plot in one row if false, the ortho image will be a 2x2 grid with the bottom

left corner blank

cmap
[string] colormap to use for base image. See matplotlib.

overlay_cmap
[string] colormap to use for overlay images, if applicable. See matplotlib.

overlay_alpha
[float] level of transparency for any overlays. Smaller value means the overlay is more transparent. See matplotlib.

cbar: boolean
if true, a colorbar will be added to the plot

cbar_length: float
length of the colorbar relative to the image

cbar_dx: float
horizontal shift of the colorbar relative to the image

cbar_vertical: boolean
if true, the colorbar will be vertical, if false, it will be horizontal underneath the image

axis
[integer] which axis to plot along if image is 3D

black_bg
[boolean] if True, the background of the image(s) will be black. if False, the background of the image(s) will be determined by the

values *bg_thresh_quant* and *bg_val_quant*.

bg_thresh_quant
[float] if white_bg=True, the background will be determined by thresholding the image at the *bg_thresh* quantile value and setting the background intensity to the *bg_val* quantile value. This value should be in [0, 1] - somewhere around 0.01 is recommended.

- equal to 1 will threshold the entire image
- equal to 0 will threshold none of the image

bg_val_quant
[float] if white_bg=True, the background will be determined by thresholding the image at the *bg_thresh* quantile value and setting the background intensity to the *bg_val* quantile value. This value should be in [0, 1]

- equal to 1 is pure white
- equal to 0 is pure black
- somewhere in between is gray

domain_image_map
[ANTsImage] this input ANTsImage or list of ANTsImage types contains a reference image *domain_image* and optional reference mapping named *domainMap*. If supplied, the image(s) to be plotted will be mapped to the domain image space before plotting - useful for non-standard image orientations.

crop
[boolean] if true, the image(s) will be cropped to their bounding boxes, resulting in a potentially smaller image size. if false, the image(s) will not be cropped

scale
[boolean or 2-tuple] if true, nothing will happen to intensities of image(s) and overlay(s) if false, dynamic range will be maximized when visualizing overlays if 2-tuple, the image will be dynamically scaled between these quantiles

title
[string] add a title to the plot

filename
[string] if given, the resulting image will be saved to this file

dpi
[integer] determines resolution of image if saved to file. Higher values result in higher resolution images, but at a cost of having a larger file size

resample : resample image in case of unbalanced spacing

allow_xyz_change : boolean will attempt to adjust xyz after padding

```
>>> import ants
>>> mni = ants.image_read(ants.get_data('mni'))
>>> ants.plot_ortho(mni, xyz=(100,100,100))
>>> mni2 = mni.threshold_image(7000, mni.max())
>>> ants.plot_ortho(mni, overlay=mni2)
>>> ants.plot_ortho(mni, overlay=mni2, flat=True)
>>> ants.plot_ortho(mni, overlay=mni2, xyz=(110,110,110), xyz_
↪lines=False,
                    text='Lines Turned Off', textfontsize=22)
>>> ants.plot_ortho(mni, mni2, xyz=(120,100,100),
                    text=' Example
```

Ortho Text', textfontsize=26,
title='Example Ortho Title', titlefontsize=26)

quantile(*q*, *nonzero=True*)

Get the quantile values from an ANTsImage

Examples

```
>>> img = ants.image_read(ants.get_data('r16'))
>>> ants.quantile(img, 0.5)
>>> ants.quantile(img, (0.5, 0.75))
```

range(*axis=None*)

Return range tuple along specified axis

reflect_image(*axis=None*, *tx=None*, *metric='mattes'*)

Reflect an image along an axis

ANTsR function: *reflectImage*

Parameters

- **image** (ants.core.ANTsImage) – image to reflect

- **axis** (integer (optional)) – which dimension to reflect across, numbered from 0 to imageDimension-1
- **tx** (string (optional)) – transformation type to estimate after reflection
- **metric** (str) – similarity metric for image registration. see antsRegistration.

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> fi = ants.image_read( ants.get_ants_data('r16'), 'float' )
>>> axis = 2
>>> asym = ants.reflect_image(fi, axis, 'Affine')['warpedmovout']
>>> asym = asym - fi
```

reorient_image2(*orientation='RAS'*)

Reorient an image. .. rubric:: Example

```
>>> import ants
>>> mni = ants.image_read(ants.get_data('mni'))
>>> mni2 = mni.reorient_image2()
```

resample_image(*resample_params*, *use_voxels=False*, *interp_type=1*)

Resample image by spacing or number of voxels with various interpolators. Works with multi-channel images.

ANTsR function: *resampleImage*

Parameters

- **image** (ants.core.ANTsImage) – input image
- **resample_params** (tuple/list) – vector of size dimension with numeric values
- **use_voxels** (bool) – True means interpret resample params as voxel counts
- **interp_type** (int) – one of 0 (linear), 1 (nearest neighbor), 2 (gaussian), 3 (windowed sinc), 4 (bspline)

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> fi = ants.image_read( ants.get_ants_data("r16"))
>>> finn = ants.resample_image(fi,(50,60),True,0)
>>> filin = ants.resample_image(fi,(1.5,1.5),False,1)
>>> img = ants.image_read( ants.get_ants_data("r16"))
>>> img = ants.merge_channels([img, img])
>>> outimg = ants.resample_image(img, (128,128), True)
```

resample_image_to_target(*target*, *interp_type='linear'*, *imagetype=0*, *verbose=False*, ***kwargs*)

Resample image by using another image as target reference. This function uses ants.apply_transform with an identity matrix to achieve proper resampling.

ANTsR function: *resampleImageToTarget*

Parameters

- **image** (ants.core.ANTsImage) – image to resample
- **target** (ants.core.ANTsImage) – image of reference, the output will be in this space and will have the same pixel type.
- **interp_type** (str) –

  Choice of interpolator. Supports partial matching.
  linear nearestNeighbor multiLabel for label images but genericlabel is preferred gaussian bSpline cosineWindowedSinc welchWindowedSinc hammingWindowedSinc lanczosWindowedSinc genericLabel use this for label images
- **imagetype** (int) – choose 0/1/2/3 mapping to scalar/vector/tensor/time-series
- **verbose** (bool) – print command and run verbose application of transform.
- **kwargs** (keyword arguments) – additional arugment passed to antsApplyTransforms C code

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> fi = ants.image_read(ants.get_ants_data('r16'))
>>> fi2mm = ants.resample_image(fi, (2,2), use_voxels=0, interp_type='linear')
>>> resampled = ants.resample_image_to_target(fi2mm, fi, verbose=True)
```

rgb_to_vector()

Convert an RGB ANTsImage to a Vector ANTsImage

Parameters
image (ants.core.ANTsImage) – RGB image to be converted

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> mni = ants.image_read(ants.get_data('mni'))
>>> mni_rgb = ants.scalar_to_rgb(mni)
>>> mni_vector = mni.rgb_to_vector()
>>> mni_rgb2 = mni.vector_to_rgb()
```

set_direction(*new_direction*)

Set image direction

Parameters
new_direction (numpy.ndarray or tuple or list) – updated direction for the image. should have one value for each dimension

Return type
None

set_origin(*new_origin*)

Set image origin

Parameters
new_origin (tuple or list) – updated origin for the image. should have one value for each dimension

Return type
None

set_spacing(*new_spacing*)

Set image spacing

Parameters
new_spacing (tuple or list) – updated spacing for the image. should have one value for each dimension

Return type
None

property shape

slice_image(*axis*, *idx*, *collapse_strategy=0*)

Slice an image.

Parameters

- **axis** (int) – Which axis.
- **idx** (int) – Which slice number.
- **collapse_strategy** (int) – Collapse strategy for sub-matrix: 0, 1, or 2. 0: collapse to sub-matrix if positive-definite. Otherwise throw an exception. Default. 1: Collapse to identity. 2: Collapse to sub-matrix if positive definite. Otherwise collapse to identity.

Example

```
>>> import ants
>>> mni = ants.image_read(ants.get_data('mni'))
>>> mni2 = ants.slice_image(mni, axis=1, idx=100)
```

smooth_image(*sigma*, *sigma_in_physical_coordinates=True*, *FWHM=False*, *max_kernel_width=32*)

Smooth an image

ANTsR function: *smoothImage*

Parameters

- **image** – Image to smooth
- **sigma** – Smoothing factor. Can be scalar, in which case the same sigma is applied to each dimension, or a vector of length dim(inimage) to specify a unique smoothness for each dimension.
- **sigma_in_physical_coordinates** (bool) – If true, the smoothing factor is in millimeters; if false, it is in pixels.
- **FWHM** (bool) – If true, sigma is interpreted as the full-width-half-max (FWHM) of the filter, not the sigma of a Gaussian kernel.
- **max_kernel_width** (scalar) – Maximum kernel width

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read( ants.get_ants_data('r16'))
>>> simage = ants.smooth_image(image, (1.2,1.5))
```

property spacing

Get image spacing

Return type
tuple

split_channels()

Split channels of a multi-channel ANTsImage into a collection of scalar ANTsImage types

Parameters
image (ants.core.ANTsImage) – multi-channel image to split

Return type
list of ANTsImage types

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'), 'float')
>>> image2 = ants.image_read(ants.get_ants_data('r16'), 'float')
>>> imagemerge = ants.merge_channels([image,image2])
>>> imagemerge.components == 2
>>> images_unmerged = ants.split_channels(imagemerge)
>>> len(images_unmerged) == 2
>>> images_unmerged[0].components == 1
```

std(*axis=None*)

Return std along specified axis

sum(*axis=None*, *keepdims=False*)

Return sum along specified axis

symmetrize_image()

Use registration and reflection to make an image symmetric

ANTsR function: N/A

Parameters
image (ants.core.ANTsImage) – image to make symmetric

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read( ants.get_ants_data('r16') , 'float')
>>> simage = ants.symimage(image)
```

threshold_image(*low_thresh=None*, *high_thresh=None*, *inval=1*, *outval=0*, *binary=True*)

Converts a scalar image into a binary image by thresholding operations

ANTsR function: *thresholdImage*

Parameters

- **image** (ants.core.ANTsImage) – Input image to operate on
- **low_thresh** (scalar (optional)) – Lower edge of threshold window
- **hight_thresh** (scalar (optional)) – Higher edge of threshold window
- **inval** (scalar) – Output value for image voxels in between lothresh and hithresh
- **outval** (scalar) – Output value for image voxels lower than lothresh or higher than hithresh
- **binary** (bool) – if true, returns binary thresholded image if false, return binary thresholded image multiplied by original image

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read( ants.get_ants_data('r16') )
>>> timage = ants.threshold_image(image, 0.5, 1e15)
```

timeseries_to_matrix(*mask=None*)

Convert a timeseries image into a matrix.

ANTsR function: *timeseries2matrix*

Parameters

- **image** (image whose slices we convert to a matrix. E.g. a 3D image of size) – x by y by z will convert to a z by x*y sized matrix
- **mask** (ANTsImage (optional)) – image containing binary mask. voxels in the mask are placed in the matrix

Returns
array with a row for each image shape = (N_IMAGES, N_VOXELS)

Return type
numpy.ndarray

Example

```
>>> import ants
>>> img = ants.make_image( (10,10,10,5 ) )
>>> mat = ants.timeseries_to_matrix( img )
```

to_file(*filename*)

Write the ANTsImage to file

Parameters
filename (str) – filepath to which the image will be written

to_filename(*filename*)

Write the ANTsImage to file

Parameters
filename (str) – filepath to which the image will be written

unique(*sort=False*)

Return unique set of values in image

vector_to_rgb()

Convert an Vector ANTsImage to a RGB ANTsImage

Parameters
image (ants.core.ANTsImage) – RGB image to be converted

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> img = ants.image_read(ants.get_data('r16'), pixeltype='unsigned char')
>>> img_rgb = ants.scalar_to_rgb(img.clone())
>>> img_vec = img_rgb.rgb_to_vector()
>>> img_rgb2 = img_vec.vector_to_rgb()
```

view(*single_components=False*)

Geet a numpy array providing direct, shared access to the image data. IMPORTANT: If you alter the view, then the underlying image data will also be altered.

Parameters
single_components (boolean (default is False)) – if True, keep the extra component dimension in returned array even if image only has one component (i.e. self.has_components == False)

Return type
numpy.ndarray

weingarten_image_curvature(*sigma=1.0*, *opt='mean'*)

Uses the weingarten map to estimate image mean or gaussian curvature

ANTsR function: *weingartenImageCurvature*

Parameters

- **image** (ants.core.ANTsImage) – image from which curvature is calculated
- **sigma** (scalar) – smoothing parameter
- **opt** (str) – mean by default, otherwise *gaussian* or *characterize*

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('mni')).resample_image((3,3,3))
>>> imagecurv = ants.weingarten_image_curvature(image)
```

1.1.2 ANTsImage IO

image_clone(*image*, *pixeltype=None*)

Clone an ANTsImage

ANTsR function: *antsImageClone*

Parameters

- **image** (ants.core.ANTsImage) – image to clone
- **pixeltype** (string (optional)) – new datatype for image

Return type

ants.core.ANTsImage

image_header_info(*filename*)

Read file info from image header

ANTsR function: *antsImageHeaderInfo*

Parameters

filename (str) – name of image file from which info will be read

Return type

dict

image_read(*filename*, *dimension=None*, *pixeltype='float'*, *reorient=False*)

Read an ANTsImage from file

ANTsR function: *antsImageRead*

Parameters

- **filename** (str) – Name of the file to read the image from.
- **dimension** (int) – Number of dimensions of the image read. This need not be the same as the dimensions of the image in the file. Allowed values: 2, 3, 4. If not provided, the dimension is obtained from the image file
- **pixeltype** (str) – C++ datatype to be used to represent the pixels read. This datatype need not be the same as the datatype used in the file. Options: unsigned char, unsigned int, float, double. Use pixeltype=None to use the datatype in the file if supported. Unsupported datatypes will be converted to float.
- **reorient** (boolean | string) – if True, the image will be reoriented to RPI if it is 3D if False, nothing will happen if string, this should be the 3-letter orientation to which the

  input image will reoriented if 3D.

  if the image is 2D, this argument is ignored

Return type

ants.core.ANTsImage

image_write(*image*, *filename*, *ri=False*)

Write an ANTsImage to file

ANTsR function: *antsImageWrite*

Parameters

- **image** (ants.core.ANTsImage) – image to save to file
- **filename** (str) – name of file to which image will be saved
- **ri** (bool) –

if True, return image. This allows for using this function in a pipeline:

```
>>> img2 = img.smooth_image(2.).image_write(file1, ri=True).
↪threshold_image(0,20).image_write(file2, ri=True)
```

if False, do not return image

make_image(*shape*, *voxval=0*, *spacing=None*, *origin=None*, *direction=None*, *has_components=False*, *pixeltype='float'*)

Make an image with given size and voxel values or fill a mask with a vector of voxel values.

ANTsR function: *makeImage*

Parameters

- **shape** (tuple/ANTsImage) – input image shape (tuple) or a mask (ANTsImage). If a mask, the output image has the same shape as the mask, and the number of voxel values in voxval should match the number of positive voxels in the mask. If a tuple, the number of voxel values in voxval should match the product of the dimensions in the tuple.
- **voxval** (scalar) – input image value. Either a scalar (single value) to be placed in all voxels, or a vector of with length matching the number of voxels required by the shape argument.
- **spacing** (tuple/list) – image spatial resolution.
- **origin** (tuple/list) – image spatial origin.
- **direction** (list/ndarray) – direction matrix to convert from index to physical space.
- **components** (bool) – whether there are components per pixel or not.
- **pixeltype** (str) – a supported pixel type for ANTsImage.

Return type

ants.core.ANTsImage

from_numpy(*data*, *origin=None*, *spacing=None*, *direction=None*, *has_components=False*, *is_rgb=False*)

Create an ANTsImage object from a numpy array

ANTsR function: *as.antsImage*

Parameters

- **data** (numpy.ndarray) – image data array
- **origin** (tuple/list) – image origin
- **spacing** (tuple/list) – image spacing
- **direction** (list/ndarray) – image direction
- **has_components** (bool) – whether the image has vector components.
- **is_rgb** (bool) – whether the image is RGB. This implies has_components=True.

Returns

image with given data and any given information

Return type

ants.core.ANTsImage

matrix_to_images(*data_matrix*, *mask*)

Unmasks rows of a matrix and writes as images

ANTsR function: *matrixToImages*

Parameters

- **data_matrix** (numpy.ndarray) – each row corresponds to an image array should have number of columns equal to non-zero voxels in the mask
- **mask** (ants.core.ANTsImage) – image containing a binary mask. Rows of the matrix are unmasked and written as images. The mask defines the output image space

Return type
list of ANTsImage types

Example

```
>>> import ants
>>> img = ants.image_read(ants.get_ants_data('r16'))
>>> msk = ants.get_mask( img )
>>> img2 = ants.image_read(ants.get_ants_data('r16'))
>>> img3 = ants.image_read(ants.get_ants_data('r16'))
>>> mat = ants.image_list_to_matrix([img,img2,img3], msk )
>>> ilist = ants.matrix_to_images( mat, msk )
```

images_from_matrix(*data_matrix*, *mask*)

Unmasks rows of a matrix and writes as images

ANTsR function: *matrixToImages*

Parameters

- **data_matrix** (numpy.ndarray) – each row corresponds to an image array should have number of columns equal to non-zero voxels in the mask
- **mask** (ants.core.ANTsImage) – image containing a binary mask. Rows of the matrix are unmasked and written as images. The mask defines the output image space

Return type
list of ANTsImage types

Example

```
>>> import ants
>>> img = ants.image_read(ants.get_ants_data('r16'))
>>> msk = ants.get_mask( img )
>>> img2 = ants.image_read(ants.get_ants_data('r16'))
>>> img3 = ants.image_read(ants.get_ants_data('r16'))
>>> mat = ants.image_list_to_matrix([img,img2,img3], msk )
>>> ilist = ants.matrix_to_images( mat, msk )
```

image_list_to_matrix(*image_list*, *mask=None*, *sigma=None*, *epsilon=0.5*)

Read images into rows of a matrix, given a mask - much faster for large datasets as it is based on C++ implementations.

ANTsR function: *imagesToMatrix*

Parameters

- **image_list** (list of ANTsImage types) – images to convert to ndarray
- **mask** (ANTsImage (optional)) – Mask image, voxels in the mask (>= epsilon) are placed in the matrix. If None, the first image in image_list is thresholded at its mean value to create a mask.

- **sigma** (scaler (optional)) – smoothing factor
- **epsilon** (scalar) – threshold for mask, values >= epsilon are included in the mask.

Returns
array with a row for each image shape = (N_IMAGES, N_VOXELS)

Return type
numpy.ndarray

Example

```
>>> import ants
>>> img = ants.image_read(ants.get_ants_data('r16'))
>>> img2 = ants.image_read(ants.get_ants_data('r16'))
>>> img3 = ants.image_read(ants.get_ants_data('r16'))
>>> mat = ants.image_list_to_matrix([img,img2,img3])
```

images_to_matrix(*image_list*, *mask=None*, *sigma=None*, *epsilon=0.5*)

Read images into rows of a matrix, given a mask - much faster for large datasets as it is based on C++ implementations.

ANTsR function: *imagesToMatrix*

Parameters

- **image_list** (list of ANTsImage types) – images to convert to ndarray
- **mask** (ANTsImage (optional)) – Mask image, voxels in the mask (>= epsilon) are placed in the matrix. If None, the first image in image_list is thresholded at its mean value to create a mask.
- **sigma** (scaler (optional)) – smoothing factor
- **epsilon** (scalar) – threshold for mask, values >= epsilon are included in the mask.

Returns
array with a row for each image shape = (N_IMAGES, N_VOXELS)

Return type
numpy.ndarray

Example

```
>>> import ants
>>> img = ants.image_read(ants.get_ants_data('r16'))
>>> img2 = ants.image_read(ants.get_ants_data('r16'))
>>> img3 = ants.image_read(ants.get_ants_data('r16'))
>>> mat = ants.image_list_to_matrix([img,img2,img3])
```

matrix_from_images(*image_list*, *mask=None*, *sigma=None*, *epsilon=0.5*)

Read images into rows of a matrix, given a mask - much faster for large datasets as it is based on C++ implementations.

ANTsR function: *imagesToMatrix*

Parameters

- **image_list** (list of ANTsImage types) – images to convert to ndarray

- **mask** (ANTsImage (optional)) – Mask image, voxels in the mask (>= epsilon) are placed in the matrix. If None, the first image in image_list is thresholded at its mean value to create a mask.
- **sigma** (scaler (optional)) – smoothing factor
- **epsilon** (scalar) – threshold for mask, values >= epsilon are included in the mask.

Returns
array with a row for each image shape = (N_IMAGES, N_VOXELS)

Return type
numpy.ndarray

Example

```
>>> import ants
>>> img = ants.image_read(ants.get_ants_data('r16'))
>>> img2 = ants.image_read(ants.get_ants_data('r16'))
>>> img3 = ants.image_read(ants.get_ants_data('r16'))
>>> mat = ants.image_list_to_matrix([img,img2,img3])
```

1.2 Transforms

1.2.1 ANTsTransform

class ANTsTransform(*precision='float'*, *dimension=3*, *transform_type='AffineTransform'*, *pointer=None*)

Bases: object

apply(*data*, *data_type='point'*, *reference=None*, ***kwargs*)

Apply transform to data

apply_to_image(*image*, *reference=None*, *interpolation='linear'*)

Apply transform to an image

Parameters

- **image** (ants.core.ANTsImage) – image to which the transform will be applied
- **reference** (ants.core.ANTsImage) – target space for transforming image
- **interpolation** (str) –

  type of interpolation to use. Options are:
  linear nearestneighbor multilabel gaussian bspline cosinewindowedsinc welchwindowedsinc hammingwindoweddinc lanczoswindowedsinc genericlabel

Returns
list

Return type
transformed vector

apply_to_point(*point*)

Apply transform to a point

Parameters
point (list/tuple) – point to which the transform will be applied

Returns
list

Return type
transformed point

Example

```
>>> import ants
>>> tx = ants.new_ants_transform()
>>> params = tx.parameters
>>> tx.set_parameters(params*2)
>>> pt2 = tx.apply_to_point((1,2,3)) # should be (2,4,6)
```

apply_to_vector(*vector*)

Apply transform to a vector

Parameters
vector (list/tuple) – vector to which the transform will be applied

Returns
list

Return type
transformed vector

property fixed_parameters

Get parameters of transform

invert()

Invert the transform

property parameters

Get parameters of transform

set_fixed_parameters(*parameters*)

Set parameters of transform

set_parameters(*parameters*)

Set parameters of transform

1.2.2 ANTsTransform IO

create_ants_transform(*transform_type='AffineTransform'*, *precision='float'*, *dimension=3*, *matrix=None*, *offset=None*, *center=None*, *translation=None*, *parameters=None*, *fixed_parameters=None*, *displacement_field=None*, *supported_types=False*)

Create and initialize an ANTsTransform

ANTsR function: *createAntsrTransform*

Parameters

- **transform_type** (str) – type of transform(s)
- **precision** (str) – numerical precision
- **dimension** (int) – spatial dimension of transform
- **matrix** (numpy.ndarray) – matrix for linear transforms
- **offset** (tuple/list) – offset for linear transforms

- **center** (tuple/list) – center for linear transforms
- **translation** (tuple/list) – translation for linear transforms
- **parameters** (ndarray/list) – array of parameters
- **fixed_parameters** (ndarray/list) – array of fixed parameters
- **displacement_field** (ants.core.ANTsImage) – multichannel ANTsImage for non-linear transform
- **supported_types** (bool) – flag that returns array of possible transforms types

Return type
ANTsTransform or list of ANTsTransform types

Example

```
>>> import ants
>>> translation = (3,4,5)
>>> tx = ants.create_ants_transform( type='Euler3DTransform',␣
↪translation=translation )
```

new_ants_transform(*precision='float'*, *dimension=3*, *transform_type='AffineTransform'*, *parameters=None*, *fixed_parameters=None*)

Create a new ANTsTransform

ANTsR function: None

This is a simplified method for creating an ANTsTransform, mostly used internally. See create_ants_transform for more options.

Example

```
>>> import ants
>>> tx = ants.new_ants_transform()
```

read_transform(*filename*, *precision='float'*)

Read a transform from file

ANTsR function: *readAntsrTransform*

Parameters

- **filename** (str) – filename of transform
- **precision** (str) – numerical precision of transform

Return type
ANTsTransform

Example

```
>>> import ants
>>> tx = ants.new_ants_transform(dimension=2)
>>> tx.set_parameters((0.9,0,0,1.1,10,11))
>>> ants.write_transform(tx, '~/desktop/tx.mat')
>>> tx2 = ants.read_transform('~/desktop/tx.mat')
```

write_transform(*transform*, *filename*)

Write ANTsTransform to file

ANTsR function: *writeAntsrTransform*

Parameters

- **transform** (ANTsTransform) – transform to save
- **filename** (str) – filename of transform (file extension is ".mat" for affine transforms)

Return type

N/A

Example

```
>>> import ants
>>> tx = ants.new_ants_transform(dimension=2)
>>> tx.set_parameters((0.9,0,0,1.1,10,11))
>>> ants.write_transform(tx, '~/desktop/tx.mat')
>>> tx2 = ants.read_transform('~/desktop/tx.mat')
```

transform_from_displacement_field(*field*)

Convert deformation field (multiChannel image) to ANTsTransform

ANTsR function: *antsrTransformFromDisplacementField*

Parameters

field (ants.core.ANTsImage) – deformation field as multi-channel ANTsImage

Return type

ants.core.ANTsImage

Example

```
>>> import ants
>>> fi = ants.image_read(ants.get_ants_data('r16') )
>>> mi = ants.image_read(ants.get_ants_data('r64') )
>>> fi = ants.resample_image(fi,(60,60),1,0)
>>> mi = ants.resample_image(mi,(60,60),1,0) # speed up
>>> mytx = ants.registration(fixed=fi, moving=mi, type_of_transform = ('SyN') )
>>> vec = ants.image_read( mytx['fwdtransforms'][0] )
>>> atx = ants.transform_from_displacement_field( vec )
```

1.3 Metrics

1.3.1 ANTsMetric

class ANTsImageToImageMetric(*metric*)

Bases: object

ANTsImageToImageMetric class

property dimension

get_value()

initialize()

property is_vector

property metrictype

property pointer

property precision

set_fixed_image(*image*)

Set Fixed ANTsImage for metric

set_fixed_mask(*image*)

Set Fixed ANTsImage Mask for metric

set_moving_image(*image*)

Set Moving ANTsImage for metric

set_moving_mask(*image*)

Set Fixed ANTsImage Mask for metric

set_sampling(*strategy='regular'*, *percentage=1.0*)

1.3.2 ANTsMetric IO

new_ants_metric(*dimension=3*, *precision='float'*, *metric_type='MeanSquares'*)

create_ants_metric(*fixed*, *moving*, *metric_type='MeanSquares'*, *fixed_mask=None*, *moving_mask=None*, *sampling_strategy='regular'*, *sampling_percentage=1*)

Parameters

metric_type (str) – which metric to use options:

MeanSquares MattesMutualInformation ANTSNeighborhoodCorrelation Correlation Demons JointHistogramMutualInformation

Example

```
>>> import ants
>>> fixed = ants.image_read(ants.get_ants_data('r16'))
>>> moving = ants.image_read(ants.get_ants_data('r64'))
>>> metric_type = 'Correlation'
>>> metric = ants.create_ants_metric(fixed, moving, metric_type)
```

supported_metrics()

CHAPTER

TWO

REGISTRATION

registration(*fixed*, *moving*, *type_of_transform='SyN'*, *initial_transform=None*, *outprefix=''*, *mask=None*, *moving_mask=None*, *mask_all_stages=False*, *grad_step=0.2*, *flow_sigma=3*, *total_sigma=0*, *aff_metric='mattes'*, *aff_sampling=32*, *aff_random_sampling_rate=0.2*, *syn_metric='mattes'*, *syn_sampling=32*, *reg_iterations=(40, 20, 0)*, *aff_iterations=(2100, 1200, 1200, 10)*, *aff_shrink_factors=(6, 4, 2, 1)*, *aff_smoothing_sigmas=(3, 2, 1, 0)*, *write_composite_transform=False*, *verbose=False*, *multivariate_extras=None*, *restrict_transformation=None*, *smoothing_in_mm=False*, *singleprecision=True*, *use_legacy_histogram_matching=False*, ***kwargs*)

Register a pair of images either through the full or simplified interface to the ANTs registration method.

ANTsR function: *antsRegistration*

Parameters

- **fixed** (ants.core.ANTsImage) – fixed image to which we register the moving image.
- **moving** (ants.core.ANTsImage) – moving image to be mapped to fixed space.
- **type_of_transform** (str) – A linear or non-linear registration type. Mutual information metric by default. See Notes below for more.
- **initial_transform** (list of strings (optional)) – transforms to prepend. If None, a translation is computed to align the image centers of mass, unless the type of transform is deformable-only (time-varying diffeomorphisms, SyNOnly, or antsRegistrationSyN*[so|bo]). To force initialization with an identity transform, set this to 'Identity'.
- **outprefix** (str) – output will be named with this prefix.
- **mask** (ANTsImage (optional)) – Registration metric mask in the fixed image space.
- **moving_mask** (ANTsImage (optional)) – Registration metric mask in the moving image space.
- **mask_all_stages** (bool) – If true, apply metric mask(s) to all registration stages, instead of just the final stage.
- **grad_step** (scalar) – gradient step size (not for all tx)
- **flow_sigma** (scalar) – smoothing for update field At each iteration, the similarity metric and gradient is calculated. That gradient field is also called the update field and is smoothed before composing with the total field (i.e., the estimate of the total transform at that iteration). This total field can also be smoothed after each iteration.
- **total_sigma** (scalar) – smoothing for total field
- **aff_metric** (str) – the metric for the affine part (GC, mattes, meansquares)
- **aff_sampling** (scalar) – number of bins for the mutual information metric

- **aff_random_sampling_rate** (scalar) – the fraction of points used to estimate the metric. this can impact speed but also reproducibility and/or accuracy.
- **syn_metric** (str) – the metric for the syn part (CC, mattes, meansquares, demons)
- **syn_sampling** (scalar) – the nbins or radius parameter for the syn metric
- **reg_iterations** (list/tuple of integers) – vector of iterations for syn. we will set the smoothing and multi-resolution parameters based on the length of this vector.
- **aff_iterations** (list/tuple of integers) – vector of iterations for low-dimensional (translation, rigid, affine) registration.
- **aff_shrink_factors** (list/tuple of integers) – vector of multi-resolution shrink factors for low-dimensional (translation, rigid, affine) registration.
- **aff_smoothing_sigmas** (list/tuple of integers) – vector of multi-resolution smoothing factors for low-dimensional (translation, rigid, affine) registration.
- **write_composite_transform** (bool) – Boolean specifying whether or not the composite transform (and its inverse, if it exists) should be written to an hdf5 composite file. This is false by default so that only the transform for each stage is written to file.
- **verbose** (bool) – request verbose output (useful for debugging)
- **multivariate_extras** (additional metrics for multi-metric registration) – list of additional images and metrics which will trigger the use of multiple metrics in the registration process in the deformable stage. Each multivariate metric needs 5 entries: name of metric, fixed, moving, weight, samplingParam. the list of lists should be of the form (("nameOfMetric2", img, img, weight, metricParam)). Another example would be (("MeanSquares", f2, m2, 0.5, 0

  ), ("CC", f2, m2, 0.5, 2)) . This is only compatible

  with the SyNOnly or antsRegistrationSyN* transformations.
- **restrict_transformation** (This option allows the user to restrict the) – optimization of the displacement field, translation, rigid or affine transform on a per-component basis. For example, if one wants to limit the deformation or rotation of 3-D volume to the first two dimensions, this is possible by specifying a weight vector of '(1,1,0)' for a 3D deformation field or '(1,1,0,1,1,0)' for a rigid transformation. Restriction currently only works if there are no preceding transformations.
- **smoothing_in_mm** (boolean ; currently only impacts low dimensional registration) –
- **singleprecision** (bool) – if True, use float32 for computations. This is useful for reducing memory usage for large datasets, at the cost of precision.
- **use_legacy_histogram_matching** (bool) – if True, use the original histogram matching in ANTs. This is not recommended, but is available for backwards compatibilty with earlier versions, where it was always turned on. The default is False. A better implementation of histogram matching is available in the ants.histogram_match_image2 function.
- **kwargs** (keyword args) – extra arguments

Returns

warpedmovout: Moving image warped to space of fixed image. *warpedfixout*: Fixed image warped to space of moving image. *fwdtransforms*: Transforms to move from moving to fixed image. *invtransforms*: Transforms to move from fixed to moving image.

Return type
dict containing follow key/value pairs

Notes

The output dict contains file names, designed to be used with ants.apply_transforms. As in ANTs, the forward affine transform .mat file is present in both the fwdtransforms and invtransforms lists. The matrix is inverted at run time by ants.apply_transforms when applying an inverse transform (see its whichtoinvert parameter).
type_of_transform can be one of:

- "Translation": Translation transformation.
- "Rigid": Rigid transformation: Only rotation and translation.
- "Similarity": Similarity transformation: uniform scaling, rotation and translation.
- **"QuickRigid": Rigid transformation: Only rotation and translation.**
  May be useful for quick visualization fixes.'
- **"DenseRigid": Rigid transformation: Only rotation and translation.**
  Employs dense sampling during metric estimation.'
- **"BOLDRigid": Rigid transformation: Parameters typical for BOLD to**
  BOLD intrasubject registration'.'
- "Affine": Affine transformation: Rigid + scaling + shear (12 parameters).
- "AffineFast": Fast version of Affine.
- **"BOLDAffine": Affine transformation: Parameters typical for BOLD to**
  BOLD intrasubject registration'.'
- **"TRSAA": translation, rigid, similarity, affine (twice). please set**
  regIterations if using this option. this would be used in cases where you want a really high quality affine mapping (perhaps with mask).
- "Elastic": Elastic deformation: Affine + deformable.
- **"ElasticSyN": Symmetric normalization: Affine + deformable**
  transformation, with mutual information as optimization metric and elastic regularization.
- **"SyN": Symmetric normalization: Affine + deformable transformation,**
  with mutual information as optimization metric.
- **"SyNRA": Symmetric normalization: Rigid + Affine + deformable**
  transformation, with mutual information as optimization metric.
- **"SyNOnly": Symmetric normalization with no rigid or affine stages.**
  Uses mutual information as optimization metric.
- "SyNCC": SyN, but with cross-correlation as the metric.
- "SyNabp": SyN optimized for abpBrainExtraction.
- "SyNBold": SyN, but optimized for registrations between BOLD and T1 images.
- **"SyNBoldAff": SyN, but optimized for registrations between BOLD**
  and T1 images, with additional affine step.
- **"SyNAggro": SyN, but with more aggressive registration**
  (fine-scale matching and more deformation). Takes more time than SyN.
- "SyNLessAggro": Does exactly the same thing as "SyNAggro".

- **"TV[n]": time-varying diffeomorphism with where 'n' indicates number of** time points in velocity field discretization. The initial transform should be computed, if needed, in a separate call to ants.registration.
- "TVMSQ": time-varying diffeomorphism with mean square metric
- "TVMSQC": time-varying diffeomorphism with mean square metric for very large deformation
- **"antsRegistrationSyN[x]": recreation of the antsRegistrationSyN.sh script in ANTs**

  where 'x' is one of the transforms available:
  t: translation (1 stage) r: rigid (1 stage) a: rigid + affine (2 stages) s: rigid + affine + deformable syn (3 stages) sr: rigid + deformable syn (2 stages) so: deformable syn only (1 stage) b: rigid + affine + deformable b-spline syn (3 stages) br: rigid + deformable b-spline syn (2 stages) bo: deformable b-spline syn only (1 stage)
- **"antsRegistrationSyNQuick[x]": recreation of the antsRegistrationSyNQuick.sh script in ANTs.**
  x options as above.
- "antsRegistrationSyNRepro[x]": reproducible registration. x options as above.
- "antsRegistrationSyNQuickRepro[x]": quick reproducible registration. x options as above.

Example

```
>>> import ants
>>> fi = ants.image_read(ants.get_ants_data('r16'))
>>> mi = ants.image_read(ants.get_ants_data('r64'))
>>> fi = ants.resample_image(fi, (60,60), 1, 0)
>>> mi = ants.resample_image(mi, (60,60), 1, 0)
>>> mytx = ants.registration(fixed=fi, moving=mi, type_of_transform = 'SyN' )
>>> mytx = ants.registration(fixed=fi, moving=mi, type_of_transform =
↪'antsRegistrationSyN[t]' )
>>> mytx = ants.registration(fixed=fi, moving=mi, type_of_transform =
↪'antsRegistrationSyN[b]' )
>>> mytx = ants.registration(fixed=fi, moving=mi, type_of_transform =
↪'antsRegistrationSyN[s]' )
```

affine_initializer(*fixed_image*, *moving_image*, *search_factor=20*, *radian_fraction=0.1*, *use_principal_axis=False*, *local_search_iterations=10*, *mask=None*, *txfn=None*)

A multi-start optimizer for affine registration Searches over the sphere to find a good initialization for further registration refinement, if needed. This is a wrapper for the ANTs function antsAffineInitializer.

ANTsR function: *affineInitializer*

Parameters

- **fixed_image** (ants.core.ANTsImage) – the fixed reference image
- **moving_image** (ants.core.ANTsImage) – the moving image to be mapped to the fixed space
- **search_factor** (scalar) – degree of increments on the sphere to search
- **radian_fraction** (scalar) – between zero and one, defines the arc to search over
- **use_principal_axis** (bool) – boolean to initialize by principal axis
- **local_search_iterations** (scalar) – gradient descent iterations
- **mask** (ANTsImage (optional)) – optional mask to restrict registration

- **txfn** (string (optional)) – filename for the transformation

Returns
transformation matrix

Return type
numpy.ndarray

Example

```
>>> import ants
>>> fi = ants.image_read(ants.get_ants_data('r16'))
>>> mi = ants.image_read(ants.get_ants_data('r27'))
>>> txfile = ants.affine_initializer( fi, mi )
>>> tx = ants.read_transform(txfile, dimension=2)
```

apply_transforms(*fixed*, *moving*, *transformlist*, *interpolator='linear'*, *imagetype=0*, *whichtoinvert=None*, *compose=None*, *defaultvalue=0*, *singleprecision=False*, *verbose=False*, ***kwargs*)

Apply a transform list to map an image from one domain to another. In image registration, one computes mappings between (usually) pairs of images. These transforms are often a sequence of increasingly complex maps, e.g. from translation, to rigid, to affine to deformation. The list of such transforms is passed to this function to interpolate one image domain into the next image domain, as below. The order matters strongly and the user is advised to familiarize with the standards established in examples.

ANTsR function: *antsApplyTransforms*

Parameters

- **fixed** (ants.core.ANTsImage) – fixed image defining domain into which the moving image is transformed. The output will have the same pixel type as this image.
- **moving** (AntsImage) – moving image to be mapped to fixed space.
- **transformlist** (list of strings) – list of transforms generated by ants.registration where each transform is a filename.
- **interpolator** (str) –

  Choice of interpolator. Supports partial matching.
  linear nearestNeighbor multiLabel for label images (deprecated, prefer genericLabel) gaussian bSpline cosineWindowedSinc welchWindowedSinc hammingWindowedSinc lanczosWindowedSinc genericLabel use this for label images

- **imagetype** (int) –

  Integer mapping to image types as one of
  0 scalar (default) 1 spatial vector in index coordinates 2 diffusion tensor in index coordinates 3 time series 4 multi-channel image (vector-valued pixels, but not spatial vectors) 6 spatial vector in physical coordinates, such as an ANTs/ITK displacement field

  Default is 0 (scalar).

  Index-based vectors (imagetype=1) and tensors (imagetype=2) will be rebased into the fixed index space. They will also be reoriented to account for the physical rotation in the transform. Physical space vectors (imagetype=6) will remain in physical coordinates, but will be reoriented to maintain appropriate alignment after transformation.

  Tensors are interpolated using a log-Euclidean framework.

  Vectors are linearly interpolated component-wise. This is not optimal but probably acceptable if the neighborhood orientations are similar. If your vector data is axial (ie,

the sign is undefined) use nearestNeighbor interpolation, or project vectors into a single hemisphere before interpolation.

- **whichtoinvert** (list of booleans (optional)) – Must be same length as transformlist. whichtoinvert[i] is True if transformlist[i] is a matrix, and the matrix should be inverted. If transformlist[i] is a warp field, whichtoinvert[i] must be False. If the transform list is a matrix followed by a warp field, whichtoinvert defaults to (True,False). Otherwise it defaults to [False]*len(transformlist)).
- **compose** (string (optional)) – if it is a string pointing to a valid file location, this will force the function to return a composite transformation filename.
- **defaultvalue** (scalar) – Default voxel value for mappings outside the image domain.
- **singleprecision** (bool) – if True, use float32 for computations. This is useful for reducing memory usage for large datasets, at the cost of precision.
- **verbose** (bool) – print command and run verbose application of transform.
- **kwargs** (keyword arguments) – extra parameters

Return type
ants.core.ANTsImage or string (transformation filename)

Example

```
>>> import ants
>>> fixed = ants.image_read( ants.get_ants_data('r16') )
>>> moving = ants.image_read( ants.get_ants_data('r64') )
>>> fixed = ants.resample_image(fixed, (64,64), 1, 0)
>>> moving = ants.resample_image(moving, (64,64), 1, 0)
>>> mytx = ants.registration(fixed=fixed , moving=moving ,
                             type_of_transform = 'SyN' )
>>> mywarpedimage = ants.apply_transforms( fixed=fixed, moving=moving,
                                           transformlist=mytx['fwdtransforms'] )
```

create_jacobian_determinant_image(*domain_image*, *tx*, *do_log=False*, *geom=False*)

Compute the jacobian determinant from a transformation file

ANTsR function: *createJacobianDeterminantImage*

Parameters

- **domain_image** (ants.core.ANTsImage) – image that defines transformation domain
- **tx** (str) – deformation transformation file name
- **do_log** (bool) – return the log jacobian
- **geom** (bool) – use the geometric jacobian calculation (boolean)

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> fi = ants.image_read( ants.get_ants_data('r16'))
>>> mi = ants.image_read( ants.get_ants_data('r64'))
>>> fi = ants.resample_image(fi,(128,128),1,0)
```

(continues on next page)

(continued from previous page)

```
>>> mi = ants.resample_image(mi,(128,128),1,0)
>>> mytx = ants.registration(fixed=fi , moving=mi, type_of_transform = ('SyN') )
>>> jac = ants.create_jacobian_determinant_image(fi,mytx['fwdtransforms'][0],1)
```

create_warped_grid(*image*, *grid_step=10*, *grid_width=2*, *grid_directions=(True, True)*, *fixed_reference_image=None*, *transform=None*, *foreground=1*, *background=0*)

Deforming a grid is a helpful way to visualize a deformation field. This function enables a user to define the grid parameters and apply a deformable map to that grid.

ANTsR function: *createWarpedGrid*

Parameters

- **image** (ants.core.ANTsImage) – input image
- **grid_step** (scalar) – width of grid blocks
- **grid_width** (scalar) – width of grid lines
- **grid_directions** (tuple of booleans) – directions in which to draw grid lines, boolean vector
- **fixed_reference_image** (ANTsImage (optional)) – reference image space
- **transform** (list/tuple of strings (optional)) – vector of transforms
- **foreground** (scalar) – intensity value for grid blocks
- **background** (scalar) – intensity value for grid lines

Return type

ants.core.ANTsImage

Example

```
>>> import ants
>>> fi = ants.image_read( ants.get_ants_data( 'r16' ) )
>>> mi = ants.image_read( ants.get_ants_data( 'r64' ) )
>>> mygr = ants.create_warped_grid( mi )
>>> mytx = ants.registration(fixed=fi, moving=mi, type_of_transform = ('SyN') )
>>> mywarpedgrid = ants.create_warped_grid( mygr, grid_directions=(False,True),
                        transform=mytx['fwdtransforms'], fixed_reference_image=fi )
```

fsl2antstransform(*matrix*, *reference*, *moving*)

Convert an FSL linear transform to an antsrTransform

ANTsR function: *fsl2antsrtransform*

Parameters

- **matrix** (ndarray/list) – 4x4 matrix of transform parameters
- **reference** (ants.core.ANTsImage) – target image
- **moving** (ants.core.ANTsImage) – moving image

Return type

ANTsTransform

Examples

```
>>> import ants
>>> import numpy as np
>>> fslmat = np.zeros((4,4))
>>> np.fill_diagonal(fslmat, 1)
>>> img = ants.image_read(ants.get_ants_data('ch2'))
>>> tx = ants.fsl2antstransform(fslmat, img, img)
```

image_mutual_information(*image1*, *image2*)

Compute mutual information between two ANTsImage types

ANTsR function: *antsImageMutualInformation*

Parameters

- **image1** (ants.core.ANTsImage) – image 1
- **image2** (ants.core.ANTsImage) – image 2

Return type

scalar

Example

```
>>> import ants
>>> fi = ants.image_read( ants.get_ants_data('r16') ).clone('float')
>>> mi = ants.image_read( ants.get_ants_data('r64') ).clone('float')
>>> mival = ants.image_mutual_information(fi, mi) # -0.1796141
```

reflect_image(*image*, *axis=None*, *tx=None*, *metric='mattes'*)

Reflect an image along an axis

ANTsR function: *reflectImage*

Parameters

- **image** (ants.core.ANTsImage) – image to reflect
- **axis** (integer (optional)) – which dimension to reflect across, numbered from 0 to imageDimension-1
- **tx** (string (optional)) – transformation type to estimate after reflection
- **metric** (str) – similarity metric for image registration. see antsRegistration.

Return type

ants.core.ANTsImage

Example

```
>>> import ants
>>> fi = ants.image_read( ants.get_ants_data('r16'), 'float' )
>>> axis = 2
>>> asym = ants.reflect_image(fi, axis, 'Affine')['warpedmovout']
>>> asym = asym - fi
```

reorient_image()

get_center_of_mass(*image*)

Compute an image center of mass in physical space which is defined as the mean of the intensity weighted voxel coordinate system.

ANTsR function: *getCenterOfMass*

Parameters

image (ants.core.ANTsImage) – image from which center of mass will be computed

Return type

scalar

Example

```
>>> fi = ants.image_read( ants.get_ants_data("r16"))
>>> com1 = ants.get_center_of_mass( fi )
>>> fi = ants.image_read( ants.get_ants_data("r64"))
>>> com2 = ants.get_center_of_mass( fi )
```

resample_image(*image*, *resample_params*, *use_voxels=False*, *interp_type=1*)

Resample image by spacing or number of voxels with various interpolators. Works with multi-channel images.

ANTsR function: *resampleImage*

Parameters

- **image** (ants.core.ANTsImage) – input image
- **resample_params** (tuple/list) – vector of size dimension with numeric values
- **use_voxels** (bool) – True means interpret resample params as voxel counts
- **interp_type** (int) – one of 0 (linear), 1 (nearest neighbor), 2 (gaussian), 3 (windowed sinc), 4 (bspline)

Return type

ants.core.ANTsImage

Example

```
>>> import ants
>>> fi = ants.image_read( ants.get_ants_data("r16"))
>>> finn = ants.resample_image(fi,(50,60),True,0)
>>> filin = ants.resample_image(fi,(1.5,1.5),False,1)
>>> img = ants.image_read( ants.get_ants_data("r16"))
>>> img = ants.merge_channels([img, img])
>>> outimg = ants.resample_image(img, (128,128), True)
```

resample_image_to_target(*image*, *target*, *interp_type='linear'*, *imagetype=0*, *verbose=False*, ***kwargs*)

Resample image by using another image as target reference. This function uses ants.apply_transform with an identity matrix to achieve proper resampling.

ANTsR function: *resampleImageToTarget*

Parameters

- **image** (ants.core.ANTsImage) – image to resample
- **target** (ants.core.ANTsImage) – image of reference, the output will be in this space and will have the same pixel type.
- **interp_type** (str) –

Choice of interpolator. Supports partial matching.
linear nearestNeighbor multiLabel for label images but genericlabel is preferred gaussian bSpline cosineWindowedSinc welchWindowedSinc hammingWindowedSinc lanczosWindowedSinc genericLabel use this for label images

- **imagetype** (int) – choose 0/1/2/3 mapping to scalar/vector/tensor/time-series
- **verbose** (bool) – print command and run verbose application of transform.
- **kwargs** (keyword arguments) – additional arugment passed to antsApplyTransforms C code

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> fi = ants.image_read(ants.get_ants_data('r16'))
>>> fi2mm = ants.resample_image(fi, (2,2), use_voxels=0, interp_type='linear')
>>> resampled = ants.resample_image_to_target(fi2mm, fi, verbose=True)
```

CHAPTER

THREE

SEGMENTATION

atropos(*a*, *x*, *i='Kmeans[3]'*, *m='[0.2,1x1]'*, *c='[5,0]'*, *priorweight=0.25*, ***kwargs*)

A finite mixture modeling (FMM) segmentation approach with possibilities for specifying prior constraints. These prior constraints include the specification of a prior label image, prior probability images (one for each class), and/or an MRF prior to enforce spatial smoothing of the labels. Similar algorithms include FAST and SPM. atropos can also perform multivariate segmentation if you pass a list of images in: e.g. a=(img1,img2).

ANTsR function: *atropos*

Parameters

- **a** (ants.core.ANTsImage or list/tuple of ANTsImage types) – One or more scalar images to segment. If priors are not used, the intensities of the first image are used to order the classes in the segmentation output, from lowest to highest intensity. Otherwise the order of the classes is dictated by the order of the prior images.
- **x** (ants.core.ANTsImage) – mask image.
- **i** (str) – initialization usually KMeans[N] for N classes or a list of N prior probability images. See Atropos in ANTs for full set of options.
- **m** (str) – mrf parameters as a string, usually "[smoothingFactor,radius]" where smoothingFactor determines the amount of smoothing and radius determines the MRF neighborhood, as an ANTs style neighborhood vector eg "1x1x1" for a 3D image. The radius must match the dimensionality of the image, eg 1x1 for 2D and The default in ANTs is smoothingFactor=0.3 and radius=1. See Atropos for more options.
- **c** (str) – convergence parameters, "[numberOfIterations,convergenceThreshold]". A threshold of 0 runs the full numberOfIterations, otherwise Atropos tests convergence by comparing the mean maximum posterior probability over the whole region of interest defined by the mask x.
- **priorweight** (scalar) – usually 0 (priors used for initialization only), 0.25 or 0.5.
- **kwargs** (keyword arguments) – more parameters, see Atropos help in ANTs

Returns

***segmentation*: ANTsImage**
actually segmented image

probabilityimages
[list of ANTsImage types] one image for each segmentation class

Return type
dictionary with the following key/value pairs

Example

```
>>> import ants
>>> img = ants.image_read(ants.get_ants_data('r16'))
>>> img = ants.resample_image(img, (64,64), 1, 0)
>>> mask = ants.get_mask(img)
>>> ants.atropos( a = img, m = '[0.2,1x1]', c = '[2,0]',  i = 'kmeans[3]', x = mask␣
↪)
>>> seg2 = ants.atropos( a = img, m = '[0.2,1x1]', c = '[2,0]', i = seg[
↪'probabilityimages'], x = mask, priorweight=0.25 )
```

joint_label_fusion(*target_image*, *target_image_mask*, *atlas_list*, *beta=4*, *rad=2*, *label_list=None*, *rho=0.01*, *usecor=False*, *r_search=3*, *nonnegative=False*, *no_zeroes=False*, *max_lab_plus_one=False*, *output_prefix=None*, *verbose=False*)

A multiple atlas voting scheme to customize labels for a new subject. This function will also perform intensity fusion. It almost directly calls the C++ in the ANTs executable so is much faster than other variants in ANTsR.

One may want to normalize image intensities for each input image before passing to this function. If no labels are passed, we do intensity fusion. Note on computation time: the underlying C++ is multithreaded. You can control the number of threads by setting the environment variable ITK_GLOBAL_DEFAULT_NUMBER_OF_THREADS e.g. to use all or some of your CPUs. This will improve performance substantially. For instance, on a macbook pro from 2015, 8 cores improves speed by about 4x.

ANTsR function: *jointLabelFusion*

Parameters

- **target_image** (`ants.core.ANTsImage`) – image to be approximated
- **target_image_mask** (`ants.core.ANTsImage`) – mask with value 1
- **atlas_list** (`list` of `ANTsImage types`) – list containing intensity images
- **beta** (`scalar`) – weight sharpness, default to 2
- **rad** (`scalar`) – neighborhood radius, default to 2
- **label_list** (`list` of `ANTsImage types (optional)`) – list containing images with segmentation labels
- **rho** (`scalar`) – ridge penalty increases robustness to outliers but also makes image converge to average
- **usecor** (`bool`) – employ correlation as local similarity
- **r_search** (`scalar`) – radius of search, default is 3
- **nonnegative** (`bool`) – constrain weights to be non-negative
- **no_zeroes** (`bool`) – this will constrain the solution only to voxels that are always non-zero in the label list
- **max_lab_plus_one** (`bool`) – this will add max label plus one to the non-zero parts of each label where the target mask is greater than one. NOTE: this will have a side effect of adding to the original label images that are passed to the program. It also guarantees that every position in the labels have some label, rather than none. Ie it guarantees to explicitly parcellate the input data.
- **output_prefix** (`str`) – file prefix for storing output probabilityimages to disk
- **verbose** (`bool`) – whether to show status updates

Returns

segmentation
[ANTsImage] segmentation image

intensity
[ANTsImage] intensity image

probabilityimages
[list of ANTsImage types] probability map image for each label

segmentation_numbers
[list of numbers] segmentation label (number, int) for each probability map

Return type
`dictionary w/ following key/value pairs`

Example

```
>>> import ants
>>> ref = ants.image_read( ants.get_ants_data('r16'))
>>> ref = ants.resample_image(ref, (50,50),1,0)
>>> ref = ants.iMath(ref,'Normalize')
>>> mi = ants.image_read( ants.get_ants_data('r27'))
>>> mi2 = ants.image_read( ants.get_ants_data('r30'))
>>> mi3 = ants.image_read( ants.get_ants_data('r62'))
>>> mi4 = ants.image_read( ants.get_ants_data('r64'))
>>> mi5 = ants.image_read( ants.get_ants_data('r85'))
>>> refmask = ants.get_mask(ref)
>>> refmask = ants.iMath(refmask,'ME',2) # just to speed things up
>>> ilist = [mi,mi2,mi3,mi4,mi5]
>>> seglist = [None]*len(ilist)
>>> for i in range(len(ilist)):
>>>     ilist[i] = ants.iMath(ilist[i],'Normalize')
>>>     mytx = ants.registration(fixed=ref , moving=ilist[i] ,
>>>         type_of_transform = ('Affine') )
>>>     mywarpedimage = ants.apply_transforms(fixed=ref,moving=ilist[i],
>>>             transformlist=mytx['fwdtransforms'])
>>>     ilist[i] = mywarpedimage
>>>     seg = ants.threshold_image(ilist[i],'Otsu', 3)
>>>     seglist[i] = ( seg ) + ants.threshold_image( seg, 1, 3 ).morphology(␣
↪operation='dilate', radius=3 )
>>> r = 2
>>> pp = ants.joint_label_fusion(ref, refmask, ilist, r_search=2,
>>>                     label_list=seglist, rad=[r]*ref.dimension )
>>> pp = ants.joint_label_fusion(ref,refmask,ilist, r_search=2, rad=[r]*ref.
↪dimension)
```

kelly_kapowski(*s*, *g*, *w*, *its=45*, *r=0.025*, *m=1.5*, *gm_label=2*, *wm_label=3*, ***kwargs*)

Compute cortical thickness using the DiReCT algorithm.

Diffeomorphic registration-based cortical thickness based on probabilistic segmentation of an image. This is an optimization algorithm.

Parameters

- **s** (`ANTsimage`) – segmentation image
- **g** (`ants.core.ANTsImage`) – gray matter probability image

- **w** (ants.core.ANTsImage) – white matter probability image
- **its** (int) – convergence params - controls iterations
- **r** (scalar) – gradient descent update parameter
- **m** (scalar) – gradient field smoothing parameter
- **gm_label** (int) – label for gray matter in the segmentation image
- **wm_label** (int) – label for white matter in the segmentation image
- **kwargs** (keyword arguments) – anything else, see KellyKapowski help in ANTs

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> img = ants.image_read( ants.get_ants_data('r16') ,2)
>>> img = ants.resample_image(img, (64,64),1,0)
>>> mask = ants.get_mask( img )
>>> segs = ants.kmeans_segmentation( img, k=3, kmask = mask)
>>> thick = ants.kelly_kapowski(s=segs['segmentation'], g=segs['probabilityimages
↪'][1],
                                w=segs['probabilityimages'][2], its=45,
                                r=0.5, m=1)
```

kmeans_segmentation(*image*, *k*, *kmask=None*, *mrf=0.1*)

K-means image segmentation that is a wrapper around *ants.atropos*

ANTsR function: *kmeansSegmentation*

Parameters

- **image** (ants.core.ANTsImage) – input image
- **k** (int) – integer number of classes
- **kmask** (ANTsImage (optional)) – segment inside this mask
- **mrf** (scalar) – smoothness, higher is smoother

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> fi = ants.image_read(ants.get_ants_data('r16'), 'float')
>>> fi = ants.n3_bias_field_correction(fi, 2)
>>> seg = ants.kmeans_segmentation(fi, 3)
```

fuzzy_spatial_cmeans_segmentation(*image*, *mask=None*, *number_of_clusters=4*, *m=2*, *p=1*, *q=1*, *radius=2*, *max_number_of_iterations=20*, *convergence_threshold=0.02*, *verbose=False*)

Fuzzy spatial c-means for image segmentation.

Image segmentation using fuzzy spatial c-means as described in

Chuang et al., Fuzzy c-means clustering with spatial information for image segmentation. CMIG: 30:9-15, 2006.

Parameters

- **image** (ants.core.ANTsImage) – Input image.
- **mask** (ants.core.ANTsImage) – Optional mask image.
- **number_of_clusters** (int) – Number of segmentation clusters.
- **m** (float) – Fuzziness parameter (default=2).
- **p** (float) – Membership importance parameter (default=1).
- **q** (float) – Spatial constraint importance parameter (default=1). q = 0 is equivalent to conventional fuzzy c-means.
- **radius** (int or tuple) – Neighborhood radius (scalar or array) for spatial constraint.
- **max_number_of_iterations** (int) – Iteration limit (default=20).
- **convergence_threshold** (float) – Convergence between iterations is measured using the Dice coefficient (default=0.02).
- **varbose** (bool) – Print progress.

Return type

dictionary containing ANTsImage and probability images

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> mask = ants.get_mask(image)
>>> fuzzy = ants.fuzzy_spatial_cmeans_segmentation(image, mask, number_of_
↪clusters=3)
```

label_geometry_measures(*label_image*, *intensity_image=None*)

Wrapper for the ANTs funtion LabelGeometryMeasures

ANTsR function: *labelGeometryMeasures*

Parameters

- **label_image** (ants.core.ANTsImage) – image on which to compute geometry. Labels must be representable as uint32.
- **intensity_image** (ANTsImage (optional)) – image with intensity values

Return type

pandas.DataFrame

Example

```
>>> import ants
>>> fi = ants.image_read( ants.get_ants_data('r16') )
>>> seg = ants.kmeans_segmentation( fi, 3 )['segmentation']
>>> geom = ants.label_geometry_measures(seg,fi)
```

prior_based_segmentation(*image*, *priors*, *mask*, *priorweight=0.25*, *mrf=0.1*, *iterations=25*)

Spatial prior-based image segmentation.

Markov random field regularized, prior-based image segmentation that is a wrapper around atropos (see ANTs and related publications).

ANTsR function: *priorBasedSegmentation*

Parameters

- **image** (ants.core.ANTsImage or list/tuple of ANTsImage types) – input image or image list for multivariate segmentation
- **priors** (list/tuple of ANTsImage types) – list of priors that cover the number of classes
- **mask** (ants.core.ANTsImage) – segment inside this mask
- **prior_weight** (scalar) – usually 0 (priors used for initialization only), 0.25 or 0.5.
- **mrf** (scalar) – regularization, higher is smoother, a numerical value in range 0.0 to 0.2
- **iterations** (int) – maximum number of iterations. could be a large value eg 25.

Returns

***segmentation*: ANTsImage**
actually segmented image

probabilityimages
[list of ANTsImage types] one image for each segmentation class

Return type
dictionary with the following key/value pairs

Example

```
>>> import ants
>>> fi = ants.image_read(ants.get_ants_data('r16'))
>>> seg = ants.kmeans_segmentation(fi,3)
>>> mask = ants.threshold_image(seg['segmentation'], 1, 1e15)
>>> priorseg = ants.prior_based_segmentation(fi, seg['probabilityimages'], mask, 0.
↪25, 0.1, 3)
```

CHAPTER

FOUR

STATISTICAL LEARNING

sparse_decom2(*inmatrix*, *inmask=(None, None)*, *sparseness=(0.01, 0.01)*, *nvecs=3*, *its=20*, *cthresh=(0, 0)*, *statdir=None*, *perms=0*, *uselong=0*, *z=0*, *smooth=0*, *robust=0*, *mycoption=0*, *initialization_list=[]*, *initialization_list2=[]*, *ell1=10*, *prior_weight=0*, *verbose=False*, *rejector=0*, *max_based=False*, *version=1*)

Decomposes two matrices into paired sparse eigenevectors to maximize canonical correlation - aka Sparse CCA. Note: we do not scale the matrices internally. We leave scaling choices to the user.

ANTsR function: *sparseDecom2*

Parameters

- **inmatrix** (typing.Tuple of ndarrays) – input as inmatrix=(mat1,mat2). n by p input matrix and n by q input matrix , spatial variable lies along columns.
- **inmask** (typing.Tuple of ANTsImage types (optional - one or both)) – optional pair of image masks
- **sparseness** (tuple) – a pair of float values e.g c(0.01,0.1) enforces an unsigned 99 percent and 90 percent sparse solution for each respective view
- **nvecs** (int) – number of eigenvector pairs
- **its** (int) – number of iterations, 10 or 20 usually sufficient
- **cthresh** (typing.Tuple) – cluster threshold pair
- **statdir** (string (optional)) – temporary directory if you want to look at full output
- **perms** (int) – number of permutations. settings permutations greater than 0 will estimate significance per vector empirically. For small datasets, these may be conservative. p-values depend on how one scales the input matrices.
- **uselong** (bool) – enforce solutions of both views to be the same - requires matrices to be the same size
- **z** (float) – subject space (low-dimensional space) sparseness value
- **smooth** (float) – smooth the data (only available when mask is used)
- **robust** (bool) – rank transform input matrices
- **mycoption** (int) – enforce 1 - spatial orthogonality, 2 - low-dimensional orthogonality or 0 - both
- **initialization_list** (list) – initialization for first view
- **initialization_list2** (list) – initialization for 2nd view
- **ell1** (float) – gradient descent parameter, if negative then l0 otherwise use l1

- **prior_weight** (scalar) – Scalar value weight on prior between 0 (prior is weak) and 1 (prior is strong). Only engaged if initialization is used
- **verbose** (bool) – activates verbose output to screen
- **rejector** (scalar) – rejects small correlation solutions
- **max_based** (bool) – whether to choose max-based thresholding

Returns

projections
[ndarray] X projections

projections2
[ndarray] Y projections

eig1
[ndarray] X components

eig2
[ndarray] Y components

summary
[pd.DataFrame] first column is canonical correlations, second column is p-values (these are *None* if perms > 0)

Return type
dict w/ following key/value pairs

Example

```
>>> import numpy as np
>>> import ants
>>> mat = np.random.randn(20, 100)
>>> mat2 = np.random.randn(20, 90)
>>> mydecom = ants.sparse_decom2(inmatrix = (mat,mat2),
                                 sparseness=(0.1,0.3), nvecs=3,
                                 its=3, perms=0)
```

eig_seg(*mask*, *img_list*, *apply_segmentation_to_images=False*, *cthresh=0*, *smooth=1*)

Segment a mask into regions based on the max value in an image list. At a given voxel the segmentation label will contain the index to the image that has the largest value. If the 3rd image has the greatest value, the segmentation label will be 3 at that voxel.

Parameters

- **mask** (ants.core.ANTsImage) – D-dimensional mask > 0 defining segmentation region.
- **img_list** (collection of ants.core.ANTsImage or np.ndarray) – images to use
- **apply_segmentation_to_images** (bool) – determines if original image list is modified by the segmentation.
- **cthresh** (int) – throw away isolated clusters smaller than this value
- **smooth** (float) – smooth the input data first by this value

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> mylist = [ants.image_read(ants.get_ants_data('r16')),
              ants.image_read(ants.get_ants_data('r27')),
              ants.image_read(ants.get_ants_data('r85'))]
>>> myseg = ants.eig_seg(ants.get_mask(mylist[0]), mylist)
```

initialize_eigenanatomy(*initmat*, *mask=None*, *initlabels=None*, *nreps=1*, *smoothing=0*)

InitializeEigenanatomy is a helper function to initialize sparseDecom and sparseDecom2. Can be used to estimate sparseness parameters per eigenvector. The user then only chooses nvecs and optional regularization parameters.

Parameters

- **initmat** (np.ndarray or ants.core.ANTsImage) – input matrix where rows provide initial vector values. alternatively, this can be an antsImage which contains labeled regions.
- **mask** (ants.core.ANTsImage) – mask if available
- **initlabels** (list/tuple of integers) – which labels in initmat to use as initial components
- **nreps** (int) – nrepetitions to use
- **smoothing** (float) – if using an initial label image, optionally smooth each roi

Returns

initlist
[list of ANTsImage types] initialization list(s) for sparseDecom(2)

mask
[ANTsImage] mask(s) for sparseDecom(2)

enames
[list of strings] string names of components for sparseDecom(2)

Return type
dict w/ the following key/value pairs

Example

```
>>> import ants
>>> import numpy as np
>>> mat = np.random.randn(4,100).astype('float32')
>>> init = ants.initialize_eigenanatomy(mat)
```

CHAPTER

FIVE

UTILITIES

n3_bias_field_correction(*image*, *downsample_factor=3*)

N3 Bias Field Correction

ANTsR function: *n3BiasFieldCorrection*

Parameters

- **image** (ants.core.ANTsImage) – image to be bias corrected
- **downsample_factor** (scalar) – how much to downsample image before performing bias correction

Return type

ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read( ants.get_ants_data('r16') )
>>> image_n3 = ants.n3_bias_field_correction(image)
```

n4_bias_field_correction(*image*, *mask=None*, *rescale_intensities=False*, *shrink_factor=4*, *convergence={'iters': [50, 50, 50, 50], 'tol': 1e-07}*, *spline_param=None*, *return_bias_field=False*, *verbose=False*, *weight_mask=None*)

N4 Bias Field Correction

ANTsR function: *n4BiasFieldCorrection*

Parameters

- **image** (ants.core.ANTsImage) – image to bias correct
- **mask** (ants.core.ANTsImage) – Input mask. If not specified, the entire image is used.
- **rescale_intensities** (bool) – At each iteration, a new intensity mapping is calculated and applied but there is nothing which constrains the new intensity range to be within certain values. The result is that the range can "drift" from the original at each iteration. This option rescales to the [min,max] range of the original image intensities within the user-specified mask. A mask is required to perform rescaling. Default is False in ANTsR/ANTsPy but True in ANTs.
- **shrink_factor** (scalar) – Shrink factor for multi-resolution correction, typically integer less than 4
- **convergence** (dict w/ keys `iters` and *tol*) – iters : vector of maximum number of iterations for each level tol : the convergence tolerance. Default tolerance is 1e-7 in ANTsR/ANTsPy but 0.0 in ANTs.

- **spline_param** (float or vector) – Parameter controlling number of control points in spline. Either single value, indicating the spacing in each direction, or vector with one entry per dimension of image, indicating the mesh size. If None, defaults to mesh size of 1 in all dimensions.
- **return_bias_field** (bool) – Return bias field instead of bias corrected image.
- **verbose** (bool) – enables verbose output.
- **weight_mask** (ANTsImage (optional)) – antsImage of weight mask

Return type
ants.core.ANTsImage

Example

```
>>> image = ants.image_read( ants.get_ants_data('r16') )
>>> image_n4 = ants.n4_bias_field_correction(image)
```

abp_n4(*image*, *intensity_truncation=(0.025, 0.975, 256)*, *mask=None*, *usen3=False*)

Truncate outlier intensities and bias correct with the N4 algorithm.

ANTsR function: *abpN4*

Parameters

- **image** (ants.core.ANTsImage) – image to correct and truncate
- **intensity_truncation** (typing.Tuple) – quantiles for intensity truncation
- **mask** (ANTsImage (optional)) – mask for bias correction
- **usen3** (bool) – if True, use N3 bias correction instead of N4

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> image2 = ants.abp_n4(image)
```

merge_channels(*image_list*, *channels_first=False*)

Merge channels of multiple scalar ANTsImage types into one multi-channel ANTsImage

ANTsR function: *mergeChannels*

Parameters
image_list (list/tuple of ANTsImage types) – scalar images to merge

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> image2 = ants.image_read(ants.get_ants_data('r16'))
>>> image3 = ants.merge_channels([image,image2])
>>> image3 = ants.merge_channels([image,image2], channels_first=True)
```

(continues on next page)

(continued from previous page)

```
>>> image3.numpy()
>>> image3.components == 2
```

split_channels(*image*)

Split channels of a multi-channel ANTsImage into a collection of scalar ANTsImage types

Parameters

image (ants.core.ANTsImage) – multi-channel image to split

Return type

list of ANTsImage types

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'), 'float')
>>> image2 = ants.image_read(ants.get_ants_data('r16'), 'float')
>>> imagemerge = ants.merge_channels([image,image2])
>>> imagemerge.components == 2
>>> images_unmerged = ants.split_channels(imagemerge)
>>> len(images_unmerged) == 2
>>> images_unmerged[0].components == 1
```

crop_image(*image*, *label_image=None*, *label=1*)

Use a label image to crop a smaller ANTsImage from within a larger ANTsImage

ANTsR function: *cropImage*

Parameters

- **image** (ants.core.ANTsImage) – image to crop
- **label_image** (ants.core.ANTsImage) – image with label values. If not supplied, estimated from data.
- **label** (int) – the label value to use

Return type

ants.core.ANTsImage

Example

```
>>> import ants
>>> fi = ants.image_read( ants.get_ants_data('r16') )
>>> cropped = ants.crop_image(fi)
>>> fi2 = ants.merge_channels([fi,fi])
>>> cropped2 = ants.crop_image(fi2)
>>> cropped = ants.crop_image(fi, fi, 100 )
```

crop_indices(*image*, *lowerind*, *upperind*)

Create a proper ANTsImage sub-image by indexing the image with indices. This is similar to but different from array sub-setting in that the resulting sub-image can be decropped back into its place without having to store its original index locations explicitly.

ANTsR function: *cropIndices*

Parameters

- **image** (ants.core.ANTsImage) – image to crop

- **lowerind** (list/tuple of integers) – vector of lower index, should be length image dimensionality
- **upperind** (list/tuple of integers) – vector of upper index, should be length image dimensionality

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> fi = ants.image_read( ants.get_ants_data("r16"))
>>> cropped = ants.crop_indices( fi, (10,10), (100,100) )
>>> cropped = ants.smooth_image( cropped, 5 )
>>> decropped = ants.decrop_image( cropped, fi )
```

decrop_image(*cropped_image*, *full_image*)

The inverse function for *ants.crop_image*

ANTsR function: *decropImage*

Parameters

- **cropped_image** (ants.core.ANTsImage) – cropped image
- **full_image** (ants.core.ANTsImage) – image in which the cropped image will be put back

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> fi = ants.image_read(ants.get_ants_data('r16'))
>>> mask = ants.get_mask(fi)
>>> cropped = ants.crop_image(fi, mask, 1)
>>> cropped = ants.smooth_image(cropped, 1)
>>> decropped = ants.decrop_image(cropped, fi)
```

denoise_image(*image*, *mask=None*, *shrink_factor=1*, *p=1*, *r=2*, *noise_model='Rician'*, *v=0*)

Denoise an image using a spatially adaptive filter originally described in J. V. Manjon, P. Coupe, Luis Marti-Bonmati, D. L. Collins, and M. Robles. Adaptive Non-Local Means Denoising of MR Images With Spatially Varying Noise Levels, Journal of Magnetic Resonance Imaging, 31:192-203, June 2010.

ANTsR function: *denoiseImage*

Parameters

- **image** (ants.core.ANTsImage) – scalar image to denoise.
- **mask** (ants.core.ANTsImage) – to limit the denoise region.
- **shrink_factor** (scalar) – downsampling level performed within the algorithm.
- **p** (int or character of format '2x2' where the x separates vector entries) – patch radius for local sample.
- **r** (int or character of format '2x2' where the x separates vector entries) – search radius from which to choose extra local samples.

- **noise_model** (str) – 'Rician' or 'Gaussian'

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> import numpy as np
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> # add fairly large salt and pepper noise
>>> imagenoise = image + np.random.randn(*image.shape).astype('float32')*5
>>> imagedenoise = ants.denoise_image(imagenoise, ants.get_mask(image))
```

fit_bspline_object_to_scattered_data(*scattered_data*, *parametric_data*, *parametric_domain_origin*, *parametric_domain_spacing*, *parametric_domain_size*, *is_parametric_dimension_closed=None*, *data_weights=None*, *number_of_fitting_levels=4*, *mesh_size=1*, *spline_order=3*)

Fit a b-spline object to scattered data. This is basically a wrapper for the ITK filter

https://itk.org/Doxygen/html/classitk_1_1BSplineScatteredDataPointSetToImageFilter.html

This filter is flexible in the possible objects that can be approximated. Possibilities include:

- 1/2/3/4-D curve
- 2-D surface in 3-D space (not available/templated)
- 2/3/4-D scalar field
- 2/3-D displacement field
- 2/3-D time-varying velocity field

In order to understand the input parameters, it is important to understand the difference between the parametric and data dimensions. A curve as one parametric dimension but the data dimension can be 1-D, 2-D, 3-D, or 4-D. In contrast, a 3-D displacement field has a parametric and data dimension of 3. The scattered data is what's approximated by the B-spline object and the parametric point is the location of scattered data within the domain of the B-spline object.

ANTsR function: *fitBsplineObjectToScatteredData*

Parameters

- **scattered_data** (2-D numpy array) – Defines the scattered data input to be approximated. Data is organized by row –> data v, column —> data dimension.
- **parametric_data** (2-D numpy array) – Defines the parametric location of the scattered data. Data is organized by row –> parametric point, column –> parametric dimension. Note that each row corresponds to the same row in the scatteredData.
- **data_weights** (1-D numpy array) – Defines the individual weighting of the corresponding scattered data value. Default = None meaning all values are weighted the same.
- **parametric_domain_origin** (typing.Tuple) – Defines the parametric origin of the B-spline object.
- **parametric_domain_spacing** (typing.Tuple) – Defines the parametric spacing of the B-spline object. Defines the sampling rate in the parametric domain.
- **parametric_domain_size** (typing.Tuple) – Defines the size (length) of the B-spline object. Note that the length of the B-spline object in dimension d is defined as parametric_domain_spacing[d] * parametric_domain_size[d]-1.

- **is_parametric_dimension_closed** (typing.Tuple) – Booleans defining whether or not the corresponding parametric dimension is closed (e.g., closed loop). Default = None.
- **number_of_fitting_levels** (int) – Specifies the number of fitting levels.
- **mesh_size** (typing.Tuple) – Defines the mesh size at the initial fitting level.
- **spline_order** (int) – Spline order of the B-spline object. Default = 3.

Returns

- returns numpy array for B-spline curve (parametric dimension = 1). Otherwise,
- returns an ANTsImage.

Example

```
>>> # Perform 2-D curve example
>>>
>>> import ants, numpy
>>> import matplotlib.pyplot as plt
>>> x = numpy.linspace(-4, 4, num=100)
>>> y = numpy.exp(-numpy.multiply(x, x)) + numpy.random.uniform(-0.1, 0.1, len(x))
>>> u = numpy.linspace(0, 1.0, num=len(x))
>>> scattered_data = numpy.column_stack((x, y))
>>> parametric_data = numpy.expand_dims(u, axis=-1)
>>> spacing = 1/(len(x)-1) * 1.0;
>>> bspline_curve = ants.fit_bspline_object_to_scattered_data(scattered_data,
>>>    parametric_data,
>>>   parametric_domain_origin=[0.0], parametric_domain_spacing=[spacing],
>>>   parametric_domain_size=[len(x)], is_parametric_dimension_closed=None,
>>>   number_of_fitting_levels=5, mesh_size=1)
>>> plt.plot(x, y, label='Noisy points')
>>> plt.plot(bspline_curve[:,0], bspline_curve[:,1], label='B-spline curve')
>>> plt.grid(True)
>>> plt.axis('tight')
>>> plt.legend(loc='upper left')
>>> plt.show()
>>>
>>> ##############################################################################
>>>
>>> # Perform 2-D scalar field (i.e., image) example
>>>
>>> import ants, numpy
>>> number_of_random_points = 10000
>>> img = ants.image_read( ants.get_ants_data("r16"))
>>> img_array = img.numpy()
>>> row_indices = numpy.random.choice(range(2, img_array.shape[0]), number_of_
↪random_points)
>>> col_indices = numpy.random.choice(range(2, img_array.shape[1]), number_of_
↪random_points)
>>> scattered_data = numpy.zeros((number_of_random_points, 1))
>>> parametric_data = numpy.zeros((number_of_random_points, 2))
>>> for i in range(number_of_random_points):
```

(continues on next page)

(continued from previous page)

```
>>>     scattered_data[i,0] = img_array[row_indices[i], col_indices[i]]
>>>     parametric_data[i,0] = row_indices[i]
>>>     parametric_data[i,1] = col_indices[i]
>>> bspline_img = ants.fit_bspline_object_to_scattered_data(
>>>     scattered_data, parametric_data,
>>>     parametric_domain_origin=[0.0, 0.0],
>>>     parametric_domain_spacing=[1.0, 1.0],
>>>     parametric_domain_size = img.shape,
>>>     number_of_fitting_levels=7, mesh_size=1)
>>>
>>> ants.plot(img, title="Original")
>>> ants.plot(bspline_img, title="B-spline approximation")
```

fit_bspline_displacement_field(*displacement_field=None*, *displacement_weight_image=None*, *displacement_origins=None*, *displacements=None*, *displacement_weights=None*, *origin=None*, *spacing=None*, *size=None*, *direction=None*, *number_of_fitting_levels=4*, *mesh_size=1*, *spline_order=3*, *enforce_stationary_boundary=True*, *estimate_inverse=False*, *rasterize_points=False*)

Fit a b-spline object to a dense displacement field image and/or a set of points with associated displacements and smooths them using B-splines. The inverse can also be estimated.. This is basically a wrapper for the ITK filter

https://itk.org/Doxygen/html/classitk_1_1DisplacementFieldToBSplineImageFilter.html}

which, in turn is a wrapper for the ITK filter used for the function fit_bspline_object_to_scattered_data.

ANTsR function: *fitBsplineToDisplacementField*

Parameters

- **displacement_field** (`ANTs image`) – Input displacement field. Either this and/or the points must be specified.
- **displacement_weight_image** (`ANTs image`) – Input image defining weighting of the voxelwise displacements in the displacement_field. I If None, defaults to identity weighting for each displacement. Default = None.
- **displacement_origins** (`2-D numpy array`) – Matrix (number_of_points x dimension) defining the origins of the input displacement points. Default = None.
- **displacements** (`2-D numpy array`) – Matrix (number_of_points x dimension) defining the displacements of the input displacement points. Default = None.
- **displacement_weights** (`1-D numpy array`) – Array defining the individual weighting of the corresponding scattered data value. Default = None meaning all values are weighted the same.
- **origin** (`typing.Tuple`) – Defines the physical origin of the B-spline object.
- **spacing** (`typing.Tuple`) – Defines the physical spacing of the B-spline object.
- **size** (`typing.Tuple`) – Defines the size (length) of the B-spline object. Note that the length of the B-spline object in dimension d is defined as spacing[d] * size[d]-1.
- **direction** (`2-D numpy array`) – Booleans defining whether or not the corresponding parametric dimension is closed (e.g., closed loop). Default = None.
- **number_of_fitting_levels** (`int`) – Specifies the number of fitting levels.
- **mesh_size** (`typing.Tuple`) – Defines the mesh size at the initial fitting level.

- **spline_order** (int) – Spline order of the B-spline object. Default = 3.
- **enforce_stationary_boundary** (bool) – Ensure no displacements on the image boundary. Default = True.
- **estimate_inverse** (bool) – Estimate the inverse displacement field. Default = False.
- **rasterize_points** (bool) – Use nearest neighbor rasterization of points for estimating the field (potential speed-up). Default = False.

Return type
Returns an ANTsImage.

Example

```
>>> import ants
>>> import numpy as np
>>> points = np.array([[-50, -50]])
>>> deltas = np.array([[10, 10]])
>>> bspline_field = ants.fit_bspline_displacement_field(
>>>   displacement_origins=points, displacements=deltas,
>>>   origin=[0.0, 0.0], spacing=[1.0, 1.0], size=[100, 100],
>>>   direction=np.array([[-1, 0], [0, -1]]),
>>>   number_of_fitting_levels=4, mesh_size=(1, 1))
```

functional_lung_segmentation(*image*, *mask=None*, *number_of_iterations=2*, *number_of_atropos_iterations=5*, *mrf_parameters='[0.7,2x2x2]'*, *number_of_clusters=6*, *cluster_centers=None*, *bias_correction='n4'*, *verbose=True*)

Ventilation-based segmentation of hyperpolarized gas lung MRI.

Lung segmentation into classes based on ventilation as described in this paper: https://pubmed.ncbi.nlm.nih.gov/21837781/

Parameters

- **image** (ANTs image) – Input proton-weighted MRI.
- **mask** (ANTs image) – Mask image designating the region to segment. 0/1 = background/foreground.
- **number_of_iterations** (int) – Number of Atropos <–> bias correction iterations (outer loop).
- **number_of_atropos_iterations** (int) – Number of Atropos iterations (inner loop). If number_of_atropos_iterations = 0, this is equivalent to K-means with no MRF priors.
- **mrf_parameters** (str) – Parameters for MRF in Atropos.
- **number_of_clusters** (int) – Number of tissue classes.
- **cluster_centers** (numpy.ndarray or tuple) – Initialization centers for k-means.
- **bias_correction** (str) – Apply "n3", "n4", or no bias correction (default = "n4").
- **verbose** (bool) – Print progress to the screen.

Returns

- Dictionary with segmentation image, probability images, and
- processed image.

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_data("mni")).resample_image((4,4,4))
>>> mask = image.get_mask()
>>> seg = ants.functional_lung_segmentation(image, mask, verbose=True,
                                            number_of_iterations=1,
                                            number_of_clusters=2,
                                            number_of_atropos_iterations=1)
```

get_ants_data(*file_id=None*, *target_file_name=None*, *antsx_cache_directory=None*)

Get ANTsPy test data file

ANTsR function: *getANTsRData*

Parameters

name (str) – name of test image tag to retrieve Options:

- 'r16'
- 'r27'
- 'r30'
- 'r62'
- 'r64'
- 'r85'
- 'ch2'
- 'mni'
- 'surf'
- 'pcasl'

Returns

filepath of test image

Return type

str

Example

```
>>> import ants
>>> mnipath = ants.get_ants_data('mni')
```

get_centroids(*image*, *clustparam=0*)

Reduces a variate/statistical/network image to a set of centroids describing the center of each stand-alone non-zero component in the image

ANTsR function: *getCentroids*

Parameters

- **image** (ants.core.ANTsImage) – image from which centroids will be calculated
- **clustparam** (int) – look at regions greater than or equal to this size

Return type

numpy.ndarray

Example

```
>>> import ants
>>> image = ants.image_read( ants.get_ants_data( "r16" ) )
>>> image = ants.threshold_image( image, 90, 120 )
>>> image = ants.label_clusters( image, 10 )
>>> cents = ants.get_centroids( image )
```

get_mask(*image*, *low_thresh=None*, *high_thresh=None*, *cleanup=2*)

Get a binary mask image from the given image after thresholding

ANTsR function: *getMask*

Parameters

- **image** (ants.core.ANTsImage) – image from which mask will be computed. Can be an antsImage of 2, 3 or 4 dimensions.
- **low_thresh** (scalar (optional)) – An inclusive lower threshold for voxels to be included in the mask. If not given, defaults to image mean.
- **high_thresh** (scalar (optional)) – An inclusive upper threshold for voxels to be included in the mask. If not given, defaults to image max
- **cleanup** (int) – If > 0, morphological operations will be applied to clean up the mask by eroding away small or weakly-connected areas, and closing holes. If cleanup is >0, the following steps are applied
  1. Erosion with radius 2 voxels
  2. Retain largest component
  3. Dilation with radius 1 voxel
  4. Morphological closing

Return type

ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read( ants.get_ants_data('r16') )
>>> mask = ants.get_mask(image)
```

image_similarity(*fixed_image*, *moving_image*, *metric_type='MeanSquares'*, *fixed_mask=None*, *moving_mask=None*, *sampling_strategy='regular'*, *sampling_percentage=1.0*)

Measure similarity between two images. NOTE: Similarity is actually returned as distance (i.e. dissimilarity) per ITK/ANTs convention. E.g. using Correlation metric, the similarity of an image with itself returns -1.

ANTsR function: *imageSimilarity*

Parameters

- **fixed** (ants.core.ANTsImage) – the fixed image
- **moving** (ants.core.ANTsImage) – the moving image
- **metric_type** (str) –

  image metric to calculate

  MeanSquares Correlation ANTSNeighborhoodCorrelation MattesMutualInformation JointHistogramMutualInformation Demons

- **fixed_mask** (ANTsImage (optional)) – mask for the fixed image
- **moving_mask** (ANTsImage (optional)) – mask for the moving image
- **sampling_strategy** (string (optional)) –

  sampling strategy, default is full sampling
  None (Full sampling) random regular
- **sampling_percentage** (scalar) – percentage of data to sample when calculating metric Must be between 0 and 1

Return type
scalar

Example

```
>>> import ants
>>> x = ants.image_read(ants.get_ants_data('r16'))
>>> y = ants.image_read(ants.get_ants_data('r30'))
>>> metric = ants.image_similarity(x,y,metric_type='MeanSquares')
```

image_to_cluster_images(*image*, *min_cluster_size=50*, *min_thresh=1e-06*, *max_thresh=1*)

Converts an image to several independent images.

Produces a unique image for each connected component 1 through N of size > min_cluster_size

ANTsR function: *image2ClusterImages*

Parameters

- **image** (ants.core.ANTsImage) – input image
- **min_cluster_size** (int) – throw away clusters smaller than this value
- **min_thresh** (scalar) – threshold to a statistical map
- **max_thresh** (scalar) – threshold to a statistical map

Return type
list of ANTsImage types

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> image = ants.threshold_image(image, 1, 1e15)
>>> image_cluster_list = ants.image_to_cluster_images(image)
```

iMath(*image*, *operation*, **args*)

Perform various (often mathematical) operations on the input image/s. Additional parameters should be specific for each operation. See the the full iMath in ANTs, on which this function is based.

ANTsR function: *iMath*

Parameters

- **image** (ants.core.ANTsImage) – input object, usually antsImage
- **operation** – a string e.g. "GetLargestComponent" ... the special case of "GetOperations" or "GetOperationsFull" will return a list of operations and brief description. Some operations may not be valid (WIP), but most are.
- ***args** (non-keyword arguments) – additional parameters specific to the operation

Example

```
>>> import ants
>>> img = ants.image_read(ants.get_ants_data('r16'))
>>> img2 = ants.iMath(img, 'Canny', 1, 5, 12)
```

label_clusters(*image*, *min_cluster_size=50*, *min_thresh=1e-06*, *max_thresh=1*, *fully_connected=False*)

This will give a unique ID to each connected component 1 through N of size > min_cluster_size

ANTsR function: *labelClusters*

Parameters

- **image** (ants.core.ANTsImage) – input image e.g. a statistical map
- **min_cluster_size** (int) – throw away clusters smaller than this value
- **min_thresh** (scalar) – threshold to a statistical map
- **max_thresh** (scalar) – threshold to a statistical map
- **fully_connected** (bool) – boolean sets neighborhood connectivity pattern

Return type

ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read( ants.get_ants_data('r16') )
>>> timageFully = ants.label_clusters( image, 10, 128, 150, True )
>>> timageFace = ants.label_clusters( image, 10, 128, 150, False )
```

label_image_centroids(*image*, *physical=False*, *convex=True*, *verbose=False*)

Converts a label image to coordinates summarizing their positions

ANTsR function: *labelImageCentroids*

Parameters

- **image** (ants.core.ANTsImage) – image of integer labels
- **physical** (bool) – whether you want physical space coordinates or not
- **convex** (bool) – if True, return centroid if False return point with min average distance to other points with same label

Returns

labels
[1D-ndarray] array of label values

vertices
[pd.DataFrame] coordinates of label centroids

Return type

dictionary w/ following key-value pairs

Example

```
>>> import ants
>>> import numpy as np
>>> image = ants.from_numpy(np.asarray([[[0,2],[1,3]],[[4,6],[5,7]]]).astype(
↪'float32'))
>>> labels = ants.label_image_centroids(image)
```

label_overlap_measures(*source_image*, *target_image*)

Get overlap measures from two label images (e.g., Dice)

ANTsR function: *labelOverlapMeasures*

Parameters

- **image** (*source*) – Source image
- **target_image** (ants.core.ANTsImage) – Target image

Return type

data frame with measures for each label and all labels combined

Example

```
>>> import ants
>>> r16 = ants.image_read( ants.get_ants_data('r16') )
>>> r64 = ants.image_read( ants.get_ants_data('r64') )
>>> s16 = ants.kmeans_segmentation( r16, 3 )['segmentation']
>>> s64 = ants.kmeans_segmentation( r64, 3 )['segmentation']
>>> stats = ants.label_overlap_measures(s16, s64)
```

mask_image(*image*, *mask*, *level=1*, *binarize=False*)

Mask an input image by a mask image. If the mask image has multiple labels, it is possible to specify which label(s) to mask at.

ANTsR function: *maskImage*

Parameters

- **image** (ants.core.ANTsImage) – Input image.
- **mask** (ants.core.ANTsImage) – Mask or label image.
- **level** (scalar or tuple of scalars) – Level(s) at which to threshold the mask. Can be a single value or an iterable of values.
- **binarize** (bool) – whether to binarize the output image. This computes an intersection between (image != 0) and (mask == i), for all i in level.

Return type

ants.core.ANTsImage

Example

```
>>> import ants
>>> myimage = ants.image_read(ants.get_ants_data('r16'))
>>> mask = ants.get_mask(myimage)
>>> myimage_mask = ants.mask_image(myimage, mask, 3)
>>> seg = ants.kmeans_segmentation(myimage, 3)
>>> myimage_mask = ants.mask_image(myimage, seg['segmentation'], (1,3))
```

mni2tal(*xin*)

mni2tal for converting from ch2/mni space to tal - very approximate.

This is a standard approach but it's not very accurate.

ANTsR function: *mni2tal*

Parameters

xin (tuple) – point in mni152 space.

Return type

tuple

Example

```
>>> import ants
>>> ants.mni2tal( (10,12,14) )
```

References

http://bioimagesuite.yale.edu/mni2tal/501_95733_More%20Accurate%20Talairach%20Coordinates%20SLIDES.pdf
http://imaging.mrc-cbu.cam.ac.uk/imaging/MniTalairach

morphology(*image*, *operation*, *radius*, *mtype='binary'*, *value=1*, *shape='ball'*, *radius_is_parametric=False*, *thickness=1*, *lines=3*, *include_center=False*)

Apply morphological operations to an image

ANTsR function: *morphology*

Parameters

- **input** (ants.core.ANTsImage) – input image
- **operation** (str) –

  operation to apply
  "close" Morpholgical closing "dilate" Morpholgical dilation "erode" Morpholgical erosion "open" Morpholgical opening
- **radius** (scalar) – radius of structuring element
- **mtype** (str) –

  type of morphology
  "binary" Binary operation on a single value "grayscale" Grayscale operations
- **value** (scalar) – value to operation on (type='binary' only)
- **shape** (str) –

  shape of the structuring element (type='binary' only)
  "ball" spherical structuring element "box" box shaped structuring element "cross" cross shaped structuring element "annulus" annulus shaped structuring element "polygon" polygon structuring element
- **radius_is_parametric** (bool) – used parametric radius boolean (shape='ball' and shape='annulus' only)
- **thickness** (scalar) – thickness (shape='annulus' only)
- **lines** (int) – number of lines in polygon (shape='polygon' only)
- **include_center** (bool) – include center of annulus boolean (shape='annulus' only)

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> fi = ants.image_read( ants.get_ants_data('r16') , 2 )
>>> mask = ants.get_mask( fi )
>>> dilated_ball = ants.morphology( mask, operation='dilate', radius=3, mtype=
↪'binary', shape='ball')
>>> eroded_box = ants.morphology( mask, operation='erode', radius=3, mtype='binary',
↪ shape='box')
>>> opened_annulus = ants.morphology( mask, operation='open', radius=5, mtype=
↪'binary', shape='annulus', thickness=2)
```

simulate_displacement_field(*domain_image*, *field_type='bspline'*, *number_of_random_points=1000*, *sd_noise=10.0*, *enforce_stationary_boundary=True*, *number_of_fitting_levels=4*, *mesh_size=1*, *sd_smoothing=4.0*)

simulate displacement field using either b-spline or exponential transform

ANTsR function: *simulateDisplacementField*

Parameters

- **domain_image** (ants.core.ANTsImage) – Domain image
- **field_type** (str) – Either "bspline" or "exponential".
- **number_of_random_points** (int) – Number of displacement points.
- **sd_noise** (float) – Standard deviation of the displacement field noise.
- **enforce_stationary_boundary** (bool) – Determines fixed boundary conditions.
- **number_of_fitting_levels** (int) – Number of fitting levels (b-spline only).
- **mesh_size** (int or typing.Tuple) – Determines fitting resolution at base level (b-spline only).
- **sd_smoothing** (float) – Standard deviation of the Gaussian smoothing in mm (exponential only).

Return type
ANTs vector image.

Example

```
>>> import ants
>>> domain = ants.image_read( ants.get_ants_data('r16'))
>>> exp_field = ants.simulate_displacement_field(domain, field_type="exponential")
>>> bsp_field = ants.simulate_displacement_field(domain, field_type="bspline")
>>> bsp_xfrm = ants.transform_from_displacement_field(bsp_field * 3)
>>> domain_warped = ants.apply_ants_transform_to_image(bsp_xfrm, domain, domain)
```

smooth_image(*image*, *sigma*, *sigma_in_physical_coordinates=True*, *FWHM=False*, *max_kernel_width=32*)

Smooth an image

ANTsR function: *smoothImage*

Parameters

- **image** – Image to smooth
- **sigma** – Smoothing factor. Can be scalar, in which case the same sigma is applied to each dimension, or a vector of length dim(inimage) to specify a unique smoothness for each dimension.
- **sigma_in_physical_coordinates** (bool) – If true, the smoothing factor is in millimeters; if false, it is in pixels.
- **FWHM** (bool) – If true, sigma is interpreted as the full-width-half-max (FWHM) of the filter, not the sigma of a Gaussian kernel.
- **max_kernel_width** (scalar) – Maximum kernel width

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read( ants.get_ants_data('r16'))
>>> simage = ants.smooth_image(image, (1.2,1.5))
```

threshold_image(*image*, *low_thresh=None*, *high_thresh=None*, *inval=1*, *outval=0*, *binary=True*)

Converts a scalar image into a binary image by thresholding operations

ANTsR function: *thresholdImage*

Parameters

- **image** (ants.core.ANTsImage) – Input image to operate on
- **low_thresh** (scalar (optional)) – Lower edge of threshold window
- **hight_thresh** (scalar (optional)) – Higher edge of threshold window
- **inval** (scalar) – Output value for image voxels in between lothresh and hithresh
- **outval** (scalar) – Output value for image voxels lower than lothresh or higher than hithresh
- **binary** (bool) – if true, returns binary thresholded image if false, return binary thresholded image multiplied by original image

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read( ants.get_ants_data('r16') )
>>> timage = ants.threshold_image(image, 0.5, 1e15)
```

weingarten_image_curvature(*image*, *sigma=1.0*, *opt='mean'*)

Uses the weingarten map to estimate image mean or gaussian curvature

ANTsR function: *weingartenImageCurvature*

Parameters

- **image** (ants.core.ANTsImage) – image from which curvature is calculated
- **sigma** (scalar) – smoothing parameter

- **opt** (str) – mean by default, otherwise *gaussian* or *characterize*

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('mni')).resample_image((3,3,3))
>>> imagecurv = ants.weingarten_image_curvature(image)
```

from_sitk(*sitk_image: SimpleITK.Image*) → *ANTsImage*

Converts a SimpleITK image into an ANTsPy image. Requires SimpleITK.

Parameters
sitk_image (SimpleITK.Image) – The SimpleITK image to convert.

Returns
ants_image – The converted ANTsPy image.

Return type
ants.core.ANTsImage

Raises
ValueError – If the image dimensionality is unsupported.

Examples

```
>>> import SimpleITK as sitk
>>> import ants
>>> img = sitk.ReadImage('brain.nii.gz')
>>> ants_img = from_sitk(img)
```

to_sitk(*ants_image:* ANTsImage) → SimpleITK.Image

Converts an ANTsPy image into a SimpleITK image. Requires SimpleITK.

Parameters
ants_image (ants.core.ANTsImage) – The ANTsPy image to convert.

Returns
sitk_image – The converted SimpleITK image.

Return type
SimpleITK.Image

Raises
ValueError – If the image dimensionality is unsupported.

Examples

```
>>> import ants
>>> ants_img = ants.image_read('brain.nii.gz')
>>> sitk_img = ants.to_sitk(ants_img)
```

CHAPTER

SIX

DEEPLEARN

extract_image_patches(*image*, *patch_size*, *max_number_of_patches='all'*, *stride_length=1*, *mask_image=None*, *random_seed=None*, *return_as_array=False*, *randomize=True*)

Extract 2-D or 3-D image patches.

Parameters

- **image** (`ants.core.ANTsImage`) – Input image with one or more components.
- **patch_size** (`n-D tuple (depending on dimensionality).`) – Width, height, and depth (if 3-D) of patches.
- **max_number_of_patches** (`int` or `str`) – Maximum number of patches returned. If "all" is specified, then all patches in sequence (defined by the stride_length are extracted.
- **stride_length** (`int` or `typing.Tuple`) – Defines the sequential patch overlap for max_number_of_patches = "all". Can be a image-dimensional vector or a scalar.
- **mask_image** (`ANTsImage (optional)`) – Optional image specifying the sampling region for the patches when max_number_of_patches does not equal "all". The way we constrain patch selection using a mask is by forcing each returned patch to have a masked voxel at its center.
- **random_seed** (`integer (optional)`) – Seed value that allows reproducible patch extraction across runs.
- **return_as_array** (`bool`) – Specifies the return type of the function. If False (default) the return type is a list where each element is a single patch. Otherwise the return type is an array of size dim(number_of_patches, patch_size).
- **randomize** (`bool`) – Boolean controlling whether we randomize indices when masking.

Return type

`A list (or array)` of `patches.`

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> image_patches = extract_image_patches(image, patch_size=(32, 32))
```

reconstruct_image_from_patches(*patches*, *domain_image*, *stride_length=1*, *domain_image_is_mask=False*)

Reconstruct image from a list of patches.

Parameters

- **patches** (`list` or `numpy.ndarray` of `patches`) – List or array of patches defining an image. Patches are assumed to have the same format as returned by extract_image_patches.

- **domain_image** (ANTs image) – Image or mask to define the geometric information of the reconstructed image. If this is a mask image, the reconstruction will only use patches in the mask.
- **stride_length** (int or typing.Tuple) – Defines the sequential patch overlap for max_number_of_patches = "all". Can be a image-dimensional vector or a scalar.
- **domain_image_is_mask** (bool) – Boolean specifying whether the domain image is a mask used to limit the region of reconstruction from the patches.

Return type
An ANTs image.

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> image_patches = extract_image_patches(image, patch_size=(16, 16), stride_
↪length=4)
>>> reconstructed_image = reconstruct_image_from_patches(image_patches, image,␣
↪stride_length=4)
```

regression_match_image(*source_image*, *reference_image*, *mask=None*, *poly_order=1*, *truncate=True*)

Image intensity normalization using linear regression.

Parameters

- **source_image** (ants.core.ANTsImage) – Image whose intensities are matched to the reference image.
- **reference_image** (ants.core.ANTsImage) – Defines the reference intensity function.
- **poly_order** (int) – Polynomial order of fit. Default is 1 (linear fit).
- **mask** (ants.core.ANTsImage) – Defines voxels for regression modeling.
- **truncate** (bool) – Turns on/off the clipping of intensities.

Return type
ANTs image (i.e., source_image) matched to the (reference_image).

Example

```
>>> import ants
>>> source_image = ants.image_read(ants.get_ants_data('r16'))
>>> reference_image = ants.image_read(ants.get_ants_data('r64'))
>>> matched_image = regression_match_image(source_image, reference_image)
```

crop_image_center(*image*, *crop_size*)

Crop the center of an image.

Parameters

- **image** (ants.core.ANTsImage) – Input image
- **crop_size** (typing.Tuple) – Width, height, depth (if 3-D), and time (if 4-D) of crop region.

Return type
ANTs image.

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> cropped_image = crop_image_center(image, crop_size=(64, 64))
```

pad_or_crop_image_to_size(*image*, *size*)

Pad or crop an image to a specified size

Parameters

- **image** (ants.core.ANTsImage) – Input image
- **size** (tuple) – size of output image

Return type

A cropped or padded image

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> padded_image = pad_or_crop_image_to_size(image, (333, 333))
```

pad_image_by_factor(*image*, *factor*)

Pad an image based on a factor.

Pad image of size (x, y, z) to (x', y', z') where (x', y', z') is a divisible by a user-specified factor.

Parameters

- **image** (ants.core.ANTsImage) – Input image
- **factor** (scalar or typing.Tuple) – Padding factor

Return type

ANTs image.

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> padded_image = pad_image_by_factor(image, factor=4)
```

histogram_warp_image_intensities(*image*, *break_points=(0.25, 0.5, 0.75)*, *displacements=None*, *clamp_end_points=(False, False)*, *sd_displacements=0.05*, *transform_domain_size=20*)

Transform image intensities based on histogram mapping.

Apply B-spline 1-D maps to an input image for intensity warping.

Parameters

- **image** (ants.core.ANTsImage) – Input image.
- **break_points** (int or tuple) – Parametric points at which the intensity transform displacements are specified between [0, 1]. Alternatively, a single number can be given and the sequence is linearly spaced in [0, 1].
- **displacements** (tuple) – displacements to define intensity warping. Length must be equal to the breakPoints. Alternatively, if None random displacements are chosen (random normal: mean = 0, sd = sd_displacements).

- **sd_displacements** (float) – Characterize the randomness of the intensity displacement.
- **clamp_end_points** (2-element tuple of booleans) – Specify non-zero intensity change at the ends of the histogram.
- **transform_domain_size** (int) – Defines the sampling resolution of the B-spline warping.

Return type
ANTs image

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data("r64"))
>>> transformed_image = histogram_warp_image_intensities( image )
```

simulate_bias_field(*domain_image*, *number_of_points=10*, *sd_bias_field=1.0*, *number_of_fitting_levels=4*, *mesh_size=1*)

Simulate random bias field

Low frequency, spatial varying simulated random bias field using random points and B-spline fitting.

Parameters

- **domain_image** (ants.core.ANTsImage) – Image to define the spatial domain of the bias field.
- **number_of_points** (int) – Number of randomly defined points to define the bias field (default = 10).
- **sd_bias_field** (float) – Characterize the standard deviation of the amplitude (default = 1).
- **number_of_fitting_levels** (int) – B-spline fitting parameter.

Return type
ANTs image

Example

```
>>> import ants
>>> import numpy as np
>>> image = ants.image_read(ants.get_ants_data("r64"))
>>> log_field = ants.simulate_bias_field(image, number_of_points=10, sd_bias_
↪field=1.0,
...    number_of_fitting_levels=2, mesh_size=10)
>>> log_field = log_field.iMath("Normalize")
>>> field_array = np.power(np.exp(log_field.numpy()), 4)
>>> image = image * ants.from_numpy(field_array, origin=image.origin,
...    spacing=image.spacing, direction=image.direction)
```

randomly_transform_image_data(*reference_image*, *input_image_list*, *segmentation_image_list=None*, *number_of_simulations=10*, *transform_type='affine'*, *sd_affine=0.02*, *deformation_transform_type='bspline'*, *number_of_random_points=1000*, *sd_noise=10.0*, *number_of_fitting_levels=4*, *mesh_size=1*, *sd_smoothing=4.0*, *input_image_interpolator='linear'*, *segmentation_image_interpolator='nearestNeighbor'*)

Randomly transform image data (optional: with corresponding segmentations).

Apply rigid, affine and/or deformable maps to an input set of training images. The reference image domain defines the space in which this happens.

Parameters

- **reference_image** (ants.core.ANTsImage) – Defines the spatial domain for all output images. If the input images do not match the spatial domain of the reference image, we internally resample the target to the reference image. This could have unexpected consequences. Resampling to the reference domain is performed by testing using ants.image_physical_space_consistency then calling ants.resample_image_to_target with failure.
- **input_image_list** (list of lists of ANTsImages) – List of lists of input images to warp. The internal list sets contain one or more images (per subject) which are assumed to be mutually aligned. The outer list contains multiple subject lists which are randomly sampled to produce output image list.
- **segmentation_image_list** (list of ANTsImages) – List of segmentation images corresponding to the input image list (optional).
- **number_of_simulations** (int) – Number of simulated output image sets.
- **transform_type** (str) – One of the following options: "translation", "rotation", "rigid", "scaleShear", "affine", "deformation", "affineAndDeformation".
- **sd_affine** (float) – Parameter dictating deviation amount from identity for random linear transformations.
- **deformation_transform_type** (str) – "bspline" or "exponential".
- **number_of_random_points** (int) – Number of displacement points for the deformation field.
- **sd_noise** (float) – Standard deviation of the displacement field.
- **number_of_fitting_levels** (int) – Number of fitting levels (bspline deformation only).
- **mesh_size** (int or typing.Tuple) – Determines fitting resolution (bspline deformation only).
- **sd_smoothing** (float) – Standard deviation of the Gaussian smoothing in mm (exponential field only).
- **input_image_interpolator** (str) – One of the following options: "nearestNeighbor", "linear", "gaussian", "bSpline".
- **segmentation_image_interpolator** (str) – Only "nearestNeighbor" is currently available.

Return type

list of lists of transformed images

Example

```
>>> import ants
>>> image1_list = list()
>>> image1_list.append(ants.image_read(ants.get_ants_data("r16")))
>>> image2_list = list()
```

(continues on next page)

(continued from previous page)

```
>>> image2_list.append(ants.image_read(ants.get_ants_data("r64")))
>>> input_segmentations = list()
>>> input_segmentations.append(ants.threshold_image(image1, "Otsu", 3))
>>> input_segmentations.append(ants.threshold_image(image2, "Otsu", 3))
>>> input_images = list()
>>> input_images.append(image1_list)
>>> input_images.append(image2_list)
>>> data = antspynet.randomly_transform_image_data(image1,
>>>     input_images, input_segmentations, sd_affine=0.02,
>>>     transform_type = "affineAndDeformation" )
```

data_augmentation(*input_image_list*, *segmentation_image_list=None*, *pointset_list=None*, *number_of_simulations=10*, *reference_image=None*, *transform_type='affineAndDeformation'*, *noise_model='additivegaussian'*, *noise_parameters=(0.0, 0.05)*, *sd_simulated_bias_field=1.0*, *sd_histogram_warping=0.05*, *sd_affine=0.05*, *sd_deformation=0.2*, *output_numpy_file_prefix=None*, *verbose=False*)

Randomly transform image data.

Given an input image list (possibly multi-modal) and an optional corresponding segmentation image list, this function will perform data augmentation with the following augmentation possibilities:

- spatial transformations
- added image noise
- simulated bias field
- histogram warping

Parameters

- **input_image_list** (list of lists of ANTsImages) – List of lists of input images to warp. The internal list sets contain one or more images (per subject) which are assumed to be mutually aligned. The outer list contains multiple subject lists which are randomly sampled to produce output image list.
- **segmentation_image_list** (list of ANTsImages) – List of segmentation images corresponding to the input image list (optional).
- **pointset_list** (list of pointsets) – Numpy arrays corresponding to the input image list (optional). If using this option, the transform_type must be invertible.
- **number_of_simulations** (int) – Number of simulated output image sets. Default = 10.
- **reference_image** (ants.core.ANTsImage) – Defines the spatial domain for all output images. If one is not specified, we used the first image in the input image list.
- **transform_type** (str) – One of the following options: "translation", "rigid", "scaleShear", "affine", "deformation", "affineAndDeformation".
- **noise_model** (str) – 'additivegaussian', 'saltandpepper', 'shot', and 'speckle'. Alternatively, one can specify a tuple or list of one or more of the options and one is selected at random with reasonable, randomized parameters. Note that the "speckle" model takes much longer than the others.
- **noise_parameters** (tuple or numpy.ndarray or float) – 'additivegaussian': (mean, standardDeviation) 'saltandpepper': (probability, saltValue, pepperValue) 'shot': scale 'speckle': standardDeviation Note that the standard deviation, scale, and probability values are *max* values and are randomly selected in the range [0,

noise_parameter]. Also, the "mean", "saltValue" and "pepperValue" are assumed to be in the intensity normalized range of [0, 1].

- **sd_simulated_bias_field** (float) – Characterize the standard deviation of the amplitude.
- **sd_histogram_warping** (float) – Determines the strength of the bias field.
- **sd_affine** (float) – Determines the amount of affine transformation.
- **sd_deformation** (float) – Determines the amount of deformable transformation.
- **output_numpy_file_prefix** (str) – Filename of output numpy array containing all the simulated images and segmentations.

Return type

list of lists of transformed images and/or outputs to a numpy array.

Example

```
>>> image1_list = list()
>>> image1_list.append(ants.image_read(ants.get_ants_data("r16")))
>>> image2_list = list()
>>> image2_list.append(ants.image_read(ants.get_ants_data("r64")))
>>> segmentation1 = ants.threshold_image(image1_list[0], "Otsu", 3)
>>> segmentation2 = ants.threshold_image(image2_list[0], "Otsu", 3)
>>> input_segmentations = list()
>>> input_segmentations.append(segmentation1)
>>> input_segmentations.append(segmentation2)
>>> points1 = ants.get_centroids(segmentation1)[:,0:2]
>>> points2 = ants.get_centroids(segmentation2)[:,0:2]
>>> input_points = list()
>>> input_points.append(points1)
>>> input_points.append(points2)
>>> input_images = list()
>>> input_images.append(image1_list)
>>> input_images.append(image2_list)
>>> data = data_augmentation(input_images,
                             input_segmentations,
                             input_points,
                             tranform_type="scaleShear")
```

CHAPTER

SEVEN

VISUALIZATION

plot(*image*, *overlay=None*, *blend=False*, *alpha=1*, *cmap='Greys_r'*, *overlay_cmap='turbo'*, *overlay_alpha=0.9*, *vminol=None*, *vmaxol=None*, *cbar=False*, *cbar_length=0.8*, *cbar_dx=0.0*, *cbar_vertical=True*, *axis=0*, *nslices=12*, *slices=None*, *ncol=None*, *slice_buffer=None*, *black_bg=True*, *bg_thresh_quant=0.01*, *bg_val_quant=0.99*, *domain_image_map=None*, *crop=False*, *scale=False*, *reverse=False*, *title=None*, *title_fontsize=20*, *title_dx=0.0*, *title_dy=0.0*, *filename=None*, *dpi=500*, *figsize=1.5*, *reorient=True*, *resample=True*)

Plot an ANTsImage.

Use mask_image and/or threshold_image to preprocess images to be be overlaid and display the overlays in a given range. See the wiki examples.

By default, images will be reoriented to 'LAI' orientation before plotting. So, if axis == 0, the images will be ordered from the left side of the brain to the right side of the brain. If axis == 1, the images will be ordered from the anterior (front) of the brain to the posterior (back) of the brain. And if axis == 2, the images will be ordered from the inferior (bottom) of the brain to the superior (top) of the brain.

ANTsR function: *plot.antsImage*

Parameters

- **image** (`ants.core.ANTsImage`) – image to plot
- **overlay** (`ants.core.ANTsImage`) – image to overlay on base image
- **cmap** (`str`) – colormap to use for base image. See matplotlib.
- **overlay_cmap** (`str`) – colormap to use for overlay images, if applicable. See matplotlib.
- **overlay_alpha** (`float`) – level of transparency for any overlays. Smaller value means the overlay is more transparent. See matplotlib.
- **axis** (`int`) – which axis to plot along if image is 3D
- **nslices** (`int`) – number of slices to plot if image is 3D
- **slices** (`list` or `tuple` of `integers`) – specific slice indices to plot if image is 3D. If given, this will override *nslices*. This can be absolute array indices (e.g. (80,100,120)), or this can be relative array indices (e.g. (0.4,0.5,0.6))
- **ncol** (`int`) – Number of columns to have on the plot if image is 3D.
- **slice_buffer** (`int`) – how many slices to buffer when finding the non-zero slices of a 3D images. So, if slice_buffer = 10, then the first slice in a 3D image will be the first non-zero slice index plus 10 more slices.
- **black_bg** (`bool`) – if True, the background of the image(s) will be black. if False, the background of the image(s) will be determined by the

values *bg_thresh_quant* and *bg_val_quant*.

- **bg_thresh_quant** (float) – if white_bg=True, the background will be determined by thresholding the image at the *bg_thresh* quantile value and setting the background intensity to the *bg_val* quantile value. This value should be in [0, 1] - somewhere around 0.01 is recommended.
  - equal to 1 will threshold the entire image
  - equal to 0 will threshold none of the image
- **bg_val_quant** (float) – if white_bg=True, the background will be determined by thresholding the image at the *bg_thresh* quantile value and setting the background intensity to the *bg_val* quantile value. This value should be in [0, 1]
  - equal to 1 is pure white
  - equal to 0 is pure black
  - somewhere in between is gray
- **domain_image_map** (ants.core.ANTsImage) – this input ANTsImage or list of ANTsImage types contains a reference image *domain_image* and optional reference mapping named *domainMap*. If supplied, the image(s) to be plotted will be mapped to the domain image space before plotting - useful for non-standard image orientations.
- **crop** (bool) – if true, the image(s) will be cropped to their bounding boxes, resulting in a potentially smaller image size. if false, the image(s) will not be cropped
- **scale** (bool or typing.Tuple) – if true, nothing will happen to intensities of image(s) and overlay(s) if false, dynamic range will be maximized when visualizing overlays if 2-tuple, the image will be dynamically scaled between these quantiles
- **reverse** (bool) – if true, the order in which the slices are plotted will be reversed. This is useful if you want to plot from the front of the brain first to the back of the brain, or vice-versa
- **title** (str) – add a title to the plot
- **filename** (str) – if given, the resulting image will be saved to this file
- **dpi** (int) – determines resolution of image if saved to file. Higher values result in higher resolution images, but at a cost of having a larger file size
- **resample** (bool) – if true, resample image if spacing is very unbalanced.

Example

```
>>> import ants
>>> import numpy as np
>>> img = ants.image_read(ants.get_data('r16'))
>>> segs = img.kmeans_segmentation(k=3)['segmentation']
>>> ants.plot(img, segs*(segs==1), crop=True)
>>> ants.plot(img, segs*(segs==1), crop=False)
>>> mni = ants.image_read(ants.get_data('mni'))
>>> segs = mni.kmeans_segmentation(k=3)['segmentation']
>>> ants.plot(mni, segs*(segs==1), crop=False)
```

CHAPTER

EIGHT

ANTSPYX

ants	

8.1 ants

Modules

config	
contrib	
core	
decorators	
deeplearn	
internal	
label	
learn	
lib	
math	
ops	
plotting	
registration(fixed, moving[, ...])	Register a pair of images either through the full or simplified interface to the ANTs registration method.
segmentation	
utils	

8.1.1 ants.config

Functions

set_ants_deterministic([on, seed_value])	Set deterministic behavior globally for the package.

set_ants_deterministic(*on=True*, *seed_value=123*)

Set deterministic behavior globally for the package.

Parameters

- **on** (bool) – Whether to enable deterministic mode.
- **seed_value** (int or None) – Random seed to use if deterministic mode is enabled.

8.1.2 ants.contrib

Modules

sampling	

ants.contrib.sampling

Modules

affine2d	Affine transforms
affine3d	Affine transforms
transforms	Various data augmentation transforms for ANTsImage types

ants.contrib.sampling.affine2d

Affine transforms

See http://www.cs.cornell.edu/courses/cs4620/2010fa/lectures/03transforms3D.pdf

Classes

RandomRotate2D(rotation_range[, reference, lazy])	Apply a Rotated2D transform to an image, but with the zoom parameters randomly generated from a user-specified range.
RandomShear2D(shear_range[, reference, lazy])	Apply a Shear2D transform to an image, but with the shear parameters randomly generated from a user-specified range.
RandomTranslate2D(translation_range[, ...])	Apply a Translate2D transform to an image, but with the parameters randomly generated from a user-specified range.
RandomZoom2D(zoom_range[, reference, lazy])	Apply a Zoom2D transform to an image, but with the zoom parameters randomly generated from a user-specified range.

continues on next page

Table 6 – continued from previous page

Rotate2D(rotation[, reference, lazy])	Create an ANTs Affine Transform with a specified level of rotation.
Shear2D(shear[, reference, lazy])	Create an ANTs Affine Transform with a specified shear.
Translate2D(translation[, reference, lazy])	Create an ANTs Affine Transform with a specified translation.
Zoom2D(zoom[, reference, lazy])	Create an ANTs Affine Transform with a specified level of zoom.

class RandomRotate2D(*rotation_range*, *reference=None*, *lazy=False*)

Bases: object

Apply a Rotated2D transform to an image, but with the zoom parameters randomly generated from a user-specified range. The range is determined by a mean (first parameter) and standard deviation (second parameter) via calls to random.gauss.

transform(*X=None*, *y=None*)

Transform an image using an Affine transform with rotation parameters randomly generated from the user-specified range. Return the transform if X=None.

Parameters

- **X** (ants.core.ANTsImage) – Image to transform
- **y** (ANTsImage (optional)) – Another image to transform

Return type

ANTsImage if y is None, else a tuple of ANTsImage types

Examples

```
>>> import ants
>>> img = ants.image_read(ants.get_data('r16'))
>>> tx = ants.contrib.RandomRotate2D(rotation_range=(-10,10))
>>> img2 = tx.transform(img)
```

class RandomShear2D(*shear_range*, *reference=None*, *lazy=False*)

Bases: object

Apply a Shear2D transform to an image, but with the shear parameters randomly generated from a user-specified range. The range is determined by a mean (first parameter) and standard deviation (second parameter) via calls to random.gauss.

transform(*X=None*, *y=None*)

Transform an image using an Affine transform with shear parameters randomly generated from the user-specified range. Return the transform if X=None.

Parameters

- **X** (ants.core.ANTsImage) – Image to transform
- **y** (ANTsImage (optional)) – Another image to transform

Return type

ANTsImage if y is None, else a tuple of ANTsImage types

Examples

```
>>> import ants
>>> img = ants.image_read(ants.get_data('r16'))
>>> tx = ants.contrib.RandomShear2D(shear_range=(-10,10))
>>> img2 = tx.transform(img)
```

class RandomTranslate2D(*translation_range*, *reference=None*, *lazy=False*)

Bases: object

Apply a Translate2D transform to an image, but with the parameters randomly generated from a user-specified range. The range is determined by a mean (first parameter) and standard deviation (second parameter) via calls to random.gauss.

transform(*X=None*, *y=None*)

Transform an image using an Affine transform with translation parameters randomly generated from the user-specified range. Return the transform if X=None.

Parameters

- **X** (ants.core.ANTsImage) – Image to transform
- **y** (ANTsImage (optional)) – Another image to transform

Return type

ANTsImage if y is None, else a tuple of ANTsImage types

Examples

```
>>> import ants
>>> img = ants.image_read(ants.get_data('r16'))
>>> tx = ants.contrib.RandomShear2D(translation_range=(-10,10))
>>> img2 = tx.transform(img)
```

class RandomZoom2D(*zoom_range*, *reference=None*, *lazy=False*)

Bases: object

Apply a Zoom2D transform to an image, but with the zoom parameters randomly generated from a user-specified range. The range is determined by a mean (first parameter) and standard deviation (second parameter) via calls to random.gauss.

transform(*X=None*, *y=None*)

Transform an image using an Affine transform with zoom parameters randomly generated from the user-specified range. Return the transform if X=None.

Parameters

- **X** (ants.core.ANTsImage) – Image to transform
- **y** (ANTsImage (optional)) – Another image to transform

Return type

ANTsImage if y is None, else a tuple of ANTsImage types

Examples

```
>>> import ants
>>> img = ants.image_read(ants.get_data('r16'))
```

(continues on next page)

(continued from previous page)

```
>>> tx = ants.contrib.RandomZoom2D(zoom_range=(0.8,0.9))
>>> img2 = tx.transform(img)
```

class Rotate2D(*rotation*, *reference=None*, *lazy=False*)

Bases: object

Create an ANTs Affine Transform with a specified level of rotation.

transform(*X=None*, *y=None*)

Transform an image using an Affine transform with the given rotation parameters. Return the transform if X=None.

Parameters

- **X** (ants.core.ANTsImage) – Image to transform
- **y** (ANTsImage (optional)) – Another image to transform

Return type

ANTsImage if y is None, else a tuple of ANTsImage types

Examples

```
>>> import ants
>>> img = ants.image_read(ants.get_data('r16'))
>>> tx = ants.contrib.Rotate2D(rotation=(10,-5,12))
>>> img2 = tx.transform(img)
```

class Shear2D(*shear*, *reference=None*, *lazy=False*)

Bases: object

Create an ANTs Affine Transform with a specified shear.

transform(*X=None*, *y=None*)

Transform an image using an Affine transform with the given shear parameters. Return the transform if X=None.

Parameters

- **X** (ants.core.ANTsImage) – Image to transform
- **y** (ANTsImage (optional)) – Another image to transform

Return type

ANTsImage if y is None, else a tuple of ANTsImage types

Examples

```
>>> import ants
>>> img = ants.image_read(ants.get_data('r16'))
>>> tx = ants.contrib.Shear2D(shear=(10,0,0))
>>> img2_x = tx.transform(img)# x axis stays same
>>> tx = ants.contrib.Shear2D(shear=(-10,0,0)) # other direction
>>> img2_x = tx.transform(img)# x axis stays same
>>> tx = ants.contrib.Shear2D(shear=(0,10,0))
>>> img2_y = tx.transform(img) # y axis stays same
>>> tx = ants.contrib.Shear2D(shear=(0,0,10))
```

(continues on next page)

(continued from previous page)

```
>>> img2_z = tx.transform(img) # z axis stays same
>>> tx = ants.contrib.Shear2D(shear=(10,10,10))
>>> img2 = tx.transform(img)
```

class Translate2D(*translation*, *reference=None*, *lazy=False*)

Bases: object

Create an ANTs Affine Transform with a specified translation.

transform(*X=None*, *y=None*)

Transform an image using an Affine transform with the given translation parameters. Return the transform if X=None.

Parameters

- **X** (ants.core.ANTsImage) – Image to transform
- **y** (ANTsImage (optional)) – Another image to transform

Return type

ANTsImage if y is None, else a tuple of ANTsImage types

Examples

```
>>> import ants
>>> img = ants.image_read(ants.get_data('r16'))
>>> tx = ants.contrib.Translate2D(translation=(10,0))
>>> img2_x = tx.transform(img)
>>> tx = ants.contrib.Translate2D(translation=(-10,0)) # other direction
>>> img2_x = tx.transform(img)
>>> tx = ants.contrib.Translate2D(translation=(0,10))
>>> img2_z = tx.transform(img)
>>> tx = ants.contrib.Translate2D(translation=(10,10))
>>> img2 = tx.transform(img)
```

class Zoom2D(*zoom*, *reference=None*, *lazy=False*)

Bases: object

Create an ANTs Affine Transform with a specified level of zoom. Any value greater than 1 implies a "zoom-out" and anything less than 1 implies a "zoom-in".

transform(*X=None*, *y=None*)

Transform an image using an Affine transform with the given zoom parameters. Return the transform if X=None.

Parameters

- **X** (ants.core.ANTsImage) – Image to transform
- **y** (ANTsImage (optional)) – Another image to transform

Return type

ANTsImage if y is None, else a tuple of ANTsImage types

Examples

```
>>> import ants
>>> img = ants.image_read(ants.get_data('r16'))
>>> tx = ants.contrib.Zoom2D(zoom=(0.8,0.8,0.8))
>>> img2 = tx.transform(img)
```

ants.contrib.sampling.affine3d

Affine transforms

See http://www.cs.cornell.edu/courses/cs4620/2010fa/lectures/03transforms3d.pdf

Classes

Affine3D(transformation[, reference, lazy])	Create a specified ANTs Affine Transform
RandomRotate3D(rotation_range[, reference, lazy])	Apply a Rotate3D transform to an image, but with the zoom parameters randomly generated from a user-specified range.
RandomShear3D(shear_range[, reference, lazy])	Apply a Shear3D transform to an image, but with the shear parameters randomly generated from a user-specified range.
RandomTranslate3D(translation_range[, ...])	Apply a Translate3D transform to an image, but with the shear parameters randomly generated from a user-specified range.
RandomZoom3D(zoom_range[, reference, lazy])	Apply a Zoom3D transform to an image, but with the zoom parameters randomly generated from a user-specified range.
Rotate3D(rotation[, reference, lazy])	Create an ANTs Affine Transform with a specified level of rotation.
Shear3D(shear[, reference, lazy])	Create an ANTs Affine Transform with a specified shear.
Translate3D(translation[, reference, lazy])	Create an ANTs Affine Transform with a specified translation.
Zoom3D(zoom[, reference, lazy])	Create an ANTs Affine Transform with a specified level of zoom.

class Affine3D(*transformation*, *reference=None*, *lazy=False*)

Bases: object

Create a specified ANTs Affine Transform

transform(*X=None*, *y=None*)

Transform an image using an Affine transform with the given translation parameters. Return the transform if X=None.

Parameters

- **X** (ants.core.ANTsImage) – Image to transform
- **y** (ANTsImage (optional)) – Another image to transform

Return type

ANTsImage if y is None, else a tuple of ANTsImage types

Examples

```
>>> import ants
>>> img = ants.image_read(ants.get_data('ch2'))
>>> tx = ants.contrib.Affine3D(transformation=np.array([[1, 0, 0, dx], [0, 1,␣
↪0, dy],[0, 0, 1, dz]])
>>> img2_x = tx.transform(img)# image translated by (dx, dy, dz)
```

class RandomRotate3D(*rotation_range*, *reference=None*, *lazy=False*)

Bases: object

Apply a Rotate3D transform to an image, but with the zoom parameters randomly generated from a user-specified range. The range is determined by a mean (first parameter) and standard deviation (second parameter) via calls to random.gauss.

transform(*X=None*, *y=None*)

Transform an image using an Affine transform with rotation parameters randomly generated from the user-specified range. Return the transform if X=None.

Parameters

- **X** (ants.core.ANTsImage) – Image to transform
- **y** (ANTsImage (optional)) – Another image to transform

Return type

ANTsImage if y is None, else a tuple of ANTsImage types

Examples

```
>>> import ants
>>> img = ants.image_read(ants.get_data('ch2'))
>>> tx = ants.contrib.RandomRotate3D(rotation_range=(-10,10))
>>> img2 = tx.transform(img)
```

class RandomShear3D(*shear_range*, *reference=None*, *lazy=False*)

Bases: object

Apply a Shear3D transform to an image, but with the shear parameters randomly generated from a user-specified range. The range is determined by a mean (first parameter) and standard deviation (second parameter) via calls to random.gauss.

transform(*X=None*, *y=None*)

Transform an image using an Affine transform with shear parameters randomly generated from the user-specified range. Return the transform if X=None.

Parameters

- **X** (ants.core.ANTsImage) – Image to transform
- **y** (ANTsImage (optional)) – Another image to transform

Return type

ANTsImage if y is None, else a tuple of ANTsImage types

Examples

```
>>> import ants
>>> img = ants.image_read(ants.get_data('ch2'))
>>> tx = ants.contrib.RandomShear3D(shear_range=(-10,10))
>>> img2 = tx.transform(img)
```

class RandomTranslate3D(*translation_range*, *reference=None*, *lazy=False*)

Bases: object

Apply a Translate3D transform to an image, but with the shear parameters randomly generated from a user-specified range. The range is determined by a mean (first parameter) and standard deviation (second parameter) via calls to random.gauss.

transform(*X=None*, *y=None*)

Transform an image using an Affine transform with translation parameters randomly generated from the user-specified range. Return the transform if X=None.

Parameters

- **X** (ants.core.ANTsImage) – Image to transform
- **y** (ANTsImage (optional)) – Another image to transform

Return type

ANTsImage if y is None, else a tuple of ANTsImage types

Examples

```
>>> import ants
>>> img = ants.image_read(ants.get_data('ch2'))
>>> tx = ants.contrib.RandomShear3D(translation_range=(-10,10))
>>> img2 = tx.transform(img)
```

class RandomZoom3D(*zoom_range*, *reference=None*, *lazy=False*)

Bases: object

Apply a Zoom3D transform to an image, but with the zoom parameters randomly generated from a user-specified range. The range is determined by a mean (first parameter) and standard deviation (second parameter) via calls to random.gauss.

transform(*X=None*, *y=None*)

Transform an image using an Affine transform with zoom parameters randomly generated from the user-specified range. Return the transform if X=None.

Parameters

- **X** (ants.core.ANTsImage) – Image to transform
- **y** (ANTsImage (optional)) – Another image to transform

Return type

ANTsImage if y is None, else a tuple of ANTsImage types

Examples

```
>>> import ants
>>> img = ants.image_read(ants.get_data('ch2'))
```

(continues on next page)

(continued from previous page)

```
>>> tx = ants.contrib.RandomZoom3D(zoom_range=(0.8,0.9))
>>> img2 = tx.transform(img)
```

class Rotate3D(*rotation*, *reference=None*, *lazy=False*)

Bases: object

Create an ANTs Affine Transform with a specified level of rotation.

transform(*X=None*, *y=None*)

Transform an image using an Affine transform with the given rotation parameters. Return the transform if X=None.

Parameters

- **X** (ants.core.ANTsImage) – Image to transform
- **y** (ANTsImage (optional)) – Another image to transform

Return type

ANTsImage if y is None, else a tuple of ANTsImage types

Examples

```
>>> import ants
>>> img = ants.image_read(ants.get_data('ch2'))
>>> tx = ants.contrib.Rotate3D(rotation=(10,-5,12))
>>> img2 = tx.transform(img)
```

class Shear3D(*shear*, *reference=None*, *lazy=False*)

Bases: object

Create an ANTs Affine Transform with a specified shear.

transform(*X=None*, *y=None*)

Transform an image using an Affine transform with the given shear parameters. Return the transform if X=None.

Parameters

- **X** (ants.core.ANTsImage) – Image to transform
- **y** (ANTsImage (optional)) – Another image to transform

Return type

ANTsImage if y is None, else a tuple of ANTsImage types

Examples

```
>>> import ants
>>> img = ants.image_read(ants.get_data('ch2'))
>>> tx = ants.contrib.Shear3D(shear=(10,0,0))
>>> img2_x = tx.transform(img)# x axis stays same
>>> tx = ants.contrib.Shear3D(shear=(-10,0,0)) # other direction
>>> img2_x = tx.transform(img)# x axis stays same
>>> tx = ants.contrib.Shear3D(shear=(0,10,0))
>>> img2_y = tx.transform(img) # y axis stays same
>>> tx = ants.contrib.Shear3D(shear=(0,0,10))
```

(continues on next page)

(continued from previous page)

```
>>> img2_z = tx.transform(img) # z axis stays same
>>> tx = ants.contrib.Shear3D(shear=(10,10,10))
>>> img2 = tx.transform(img)
```

class Translate3D(*translation*, *reference=None*, *lazy=False*)

Bases: `object`

Create an ANTs Affine Transform with a specified translation.

transform(*X=None*, *y=None*)

Transform an image using an Affine transform with the given translation parameters. Return the transform if X=None.

Parameters

- **X** (`ants.core.ANTsImage`) – Image to transform
- **y** (`ANTsImage (optional)`) – Another image to transform

Return type

`ANTsImage if y is None, else a tuple of ANTsImage types`

Examples

```
>>> import ants
>>> img = ants.image_read(ants.get_data('ch2'))
>>> tx = ants.contrib.Translate3D(translation=(10,0,0))
>>> img2_x = tx.transform(img)# x axis stays same
>>> tx = ants.contrib.Translate3D(translation=(-10,0,0)) # other direction
>>> img2_x = tx.transform(img)# x axis stays same
>>> tx = ants.contrib.Translate3D(translation=(0,10,0))
>>> img2_y = tx.transform(img) # y axis stays same
>>> tx = ants.contrib.Translate3D(translation=(0,0,10))
>>> img2_z = tx.transform(img) # z axis stays same
>>> tx = ants.contrib.Translate3D(translation=(10,10,10))
>>> img2 = tx.transform(img)
```

class Zoom3D(*zoom*, *reference=None*, *lazy=False*)

Bases: `object`

Create an ANTs Affine Transform with a specified level of zoom. Any value greater than 1 implies a "zoom-out" and anything less than 1 implies a "zoom-in".

transform(*X=None*, *y=None*)

Transform an image using an Affine transform with the given zoom parameters. Return the transform if X=None.

Parameters

- **X** (`ants.core.ANTsImage`) – Image to transform
- **y** (`ANTsImage (optional)`) – Another image to transform

Return type

`ANTsImage if y is None, else a tuple of ANTsImage types`

Examples

```
>>> import ants
>>> img = ants.image_read(ants.get_data('ch2'))
>>> tx = ants.contrib.Zoom3D(zoom=(0.8,0.8,0.8))
>>> img2 = tx.transform(img)
```

ants.contrib.sampling.transforms

Various data augmentation transforms for ANTsImage types

List of Transformations:

- CastIntensity
- BlurIntensity
- NormalizeIntensity
- RescaleIntensity
- ShiftScaleIntensity
- SigmoidIntensity

Todo

- RotateImage
- ShearImage
- ScaleImage
- DeformImage
- PadImage
- HistogramEqualizeIntensity
- TruncateIntensity
- SharpenIntensity
- **MorpholigicalIntensity**
  - MD
  - ME
  - MO
  - MC
  - GD
  - GE
  - GO
  - GC

Classes

BlurIntensity(sigma, width)	Transform for blurring the intensity of an ANTsImage using a Gaussian Filter
CastIntensity(pixeltype)	Cast the pixeltype of an ANTsImage to a given type.
FlipImage(axis1, axis2)	Transform an image by flipping two axes.
LocallyBlurIntensity([conductance, iters])	Blur an ANTsImage locally using a gradient anisotropic diffusion filter, thereby preserving the sharpeness of edges as best as possible.
MultiResolutionImage([levels, keep_shape])	Generate a set of images at multiple resolutions from an original image
NormalizeIntensity()	Normalize the intensity values of an ANTsImage to have zero mean and unit variance
RescaleIntensity(min_val, max_val)	Rescale the pixeltype of an ANTsImage linearly to be between a given minimum and maximum value.
ScaleImage(scale[, reference, interp])	Scale an image in physical space.
ShiftScaleIntensity(shift, scale)	Shift and scale the intensity of an ANTsImage
SigmoidIntensity(min_val, max_val, alpha, beta)	Transform an image using a sigmoid function
TranslateImage(translation[, reference, interp])	Translate an image in physical space.

class BlurIntensity(*sigma*, *width*)

Bases: object

Transform for blurring the intensity of an ANTsImage using a Gaussian Filter

transform(*X*, *y=None*)

Blur an image by applying a gaussian filter.

Parameters

- **X** (ants.core.ANTsImage) – image to transform
- **y** (ANTsImage (optional)) – another image to transform.

Example

```
>>> import ants
>>> blur = ants.contrib.BlurIntensity(2,3)
>>> img2d = ants.image_read(ants.get_data('r16'))
>>> img2d_b = blur.transform(img2d)
>>> ants.plot(img2d)
>>> ants.plot(img2d_b)
>>> img3d = ants.image_read(ants.get_data('mni'))
>>> img3d_b = blur.transform(img3d)
>>> ants.plot(img3d)
>>> ants.plot(img3d_b)
```

class CastIntensity(*pixeltype*)

Bases: object

Cast the pixeltype of an ANTsImage to a given type. This code uses the C++ ITK library directly, so it is fast.

NOTE: This offers a ~2.5x speedup over using img.clone(pixeltype):

Timings vs Cloning

```
>>> import ants
>>> import time
>>> caster = ants.contrib.CastIntensity('float')
>>> img = ants.image_read(ants.get_data('mni')).clone('unsigned int')
>>> s = time.time()
>>> for i in range(1000):
...     img_float = caster.transform(img)
>>> e = time.time()
>>> print(e - s) # 9.6s
>>> s = time.time()
>>> for i in range(1000):
...     img_float = img.clone('float')
>>> e = time.time()
>>> print(e - s) # 25.3s
```

transform(*X*, *y=None*)

Transform an image by casting its type

Parameters

- **X** (ants.core.ANTsImage) – image to cast
- **y** (ANTsImage (optional)) – another image to cast.

Example

```
>>> import ants
>>> caster = ants.contrib.CastIntensity('float')
>>> img2d = ants.image_read(ants.get_data('r16')).clone('unsigned int')
>>> img2d_float = caster.transform(img2d)
>>> print(img2d.pixeltype, '- ', img2d_float.pixeltype)
>>> img3d = ants.image_read(ants.get_data('mni')).clone('unsigned int')
>>> img3d_float = caster.transform(img3d)
>>> print(img3d.pixeltype, ' - ' , img3d_float.pixeltype)
```

class FlipImage(*axis1*, *axis2*)

Bases: object

Transform an image by flipping two axes.

transform(*X*, *y=None*)

Transform an image by applying a sigmoid function.

Parameters

- **X** (ants.core.ANTsImage) – image to transform
- **y** (ANTsImage (optional)) – another image to transform.

Example

```
>>> import ants
>>> flipper = ants.contrib.FlipImage(0,1)
>>> img2d = ants.image_read(ants.get_data('r16'))
```

(continues on next page)

(continued from previous page)

```
>>> img2d_r = flipper.transform(img2d)
>>> ants.plot(img2d)
>>> ants.plot(img2d_r)
>>> flipper2 = ants.contrib.FlipImage(1,0)
>>> img2d = ants.image_read(ants.get_data('r16'))
>>> img2d_r = flipper2.transform(img2d)
>>> ants.plot(img2d)
>>> ants.plot(img2d_r)
```

class LocallyBlurIntensity(*conductance=1*, *iters=5*)

Bases: object

Blur an ANTsImage locally using a gradient anisotropic diffusion filter, thereby preserving the sharpeness of edges as best as possible.

transform(*X*, *y=None*)

Locally blur an image by applying a gradient anisotropic diffusion filter.

Parameters

- **X** (ants.core.ANTsImage) – image to transform
- **y** (ANTsImage (optional)) – another image to transform.

Example

```
>>> import ants
>>> blur = ants.contrib.LocallyBlurIntensity(1,5)
>>> img2d = ants.image_read(ants.get_data('r16'))
>>> img2d_b = blur.transform(img2d)
>>> ants.plot(img2d)
>>> ants.plot(img2d_b)
>>> img3d = ants.image_read(ants.get_data('mni'))
>>> img3d_b = blur.transform(img3d)
>>> ants.plot(img3d)
>>> ants.plot(img3d_b)
```

class MultiResolutionImage(*levels=4*, *keep_shape=False*)

Bases: object

Generate a set of images at multiple resolutions from an original image

transform(*X*, *y=None*)

Generate a set of multi-resolution ANTsImage types

Parameters

- **X** (ants.core.ANTsImage) – image to transform
- **y** (ANTsImage (optional)) – another image to transform

Example

```
>>> import ants
>>> multires = ants.contrib.MultiResolutionImage(levels=4)
>>> img = ants.image_read(ants.get_data('r16'))
>>> imgs = multires.transform(img)
```

class NormalizeIntensity

Bases: object

Normalize the intensity values of an ANTsImage to have zero mean and unit variance

NOTE: this transform is more-or-less the same in speed as an equivalent numpy+scikit-learn solution.

Timing vs Numpy+Scikit-Learn

```
>>> import ants
>>> import numpy as np
>>> from sklearn.preprocessing import StandardScaler
>>> import time
>>> img = ants.image_read(ants.get_data('mni'))
>>> arr = img.numpy().reshape(1,-1)
>>> normalizer = ants.contrib.NormalizeIntensity()
>>> normalizer2 = StandardScaler()
>>> s = time.time()
>>> for i in range(100):
...     img_scaled = normalizer.transform(img)
>>> e = time.time()
>>> print(e - s) # 3.3s
>>> s = time.time()
>>> for i in range(100):
...     arr_scaled = normalizer2.fit_transform(arr)
>>> e = time.time()
>>> print(e - s) # 3.5s
```

transform(*X*, *y=None*)

Transform an image by normalizing its intensity values to have zero mean and unit variance.

Parameters

- **X** (ants.core.ANTsImage) – image to transform
- **y** (ANTsImage (optional)) – another image to transform.

Example

```
>>> import ants
>>> normalizer = ants.contrib.NormalizeIntensity()
>>> img2d = ants.image_read(ants.get_data('r16'))
>>> img2d_r = normalizer.transform(img2d)
>>> print(img2d.mean(), ',', img2d.std(), ' -> ', img2d_r.mean(), ',', img2d_r.
↪std())
>>> img3d = ants.image_read(ants.get_data('mni'))
>>> img3d_r = normalizer.transform(img3d)
>>> print(img3d.mean(), ',' , img3d.std(), ',', ' -> ', img3d_r.mean(), ',' ,␣
↪img3d_r.std())
```

class RescaleIntensity(*min_val*, *max_val*)

Bases: object

Rescale the pixeltype of an ANTsImage linearly to be between a given minimum and maximum value. This code uses the C++ ITK library directly, so it is fast.

NOTE: this offered a ~5x speedup over using built-in arithmetic operations in ANTs. It is also more-or-less the same in speed as an equivalent numpy+scikit-learn solution.

Timing vs Built-in Operations

```
>>> import ants
>>> import time
>>> rescaler = ants.contrib.RescaleIntensity(0,1)
>>> img = ants.image_read(ants.get_data('mni'))
>>> s = time.time()
>>> for i in range(100):
...     img_float = rescaler.transform(img)
>>> e = time.time()
>>> print(e - s) # 2.8s
>>> s = time.time()
>>> for i in range(100):
...     maxval = img.max()
...     img_float = (img - maxval) / (maxval - img.min())
>>> e = time.time()
>>> print(e - s) # 13.9s
```

Timing vs Numpy+Scikit-Learn

```
>>> import ants
>>> import numpy as np
>>> from sklearn.preprocessing import MinMaxScaler
>>> import time
>>> img = ants.image_read(ants.get_data('mni'))
>>> arr = img.numpy().reshape(1,-1)
>>> rescaler = ants.contrib.RescaleIntensity(-1,1)
>>> rescaler2 = MinMaxScaler((-1,1)).fit(arr)
>>> s = time.time()
>>> for i in range(100):
...     img_scaled = rescaler.transform(img)
>>> e = time.time()
>>> print(e - s) # 2.8s
>>> s = time.time()
>>> for i in range(100):
...     arr_scaled = rescaler2.transform(arr)
>>> e = time.time()
>>> print(e - s) # 3s
```

transform(*X*, *y=None*)

Transform an image by linearly rescaling its intensity to be between a minimum and maximum value

Parameters

- **X** (`ants.core.ANTsImage`) – image to transform
- **y** (`ANTsImage (optional)`) – another image to transform.

Example

```
>>> import ants
>>> rescaler = ants.contrib.RescaleIntensity(0,1)
>>> img2d = ants.image_read(ants.get_data('r16'))
>>> img2d_r = rescaler.transform(img2d)
>>> print(img2d.min(), ',', img2d.max(), ' -> ', img2d_r.min(), ',', img2d_r.
↪max())
>>> img3d = ants.image_read(ants.get_data('mni'))
>>> img3d_r = rescaler.transform(img3d)
>>> print(img3d.min(), ',' , img3d.max(), ' -> ', img3d_r.min(), ',' , img3d_r.
↪max())
```

class ScaleImage(*scale*, *reference=None*, *interp='linear'*)

Bases: object

Scale an image in physical space. This function calls highly optimized ITK/C++ code.

transform(*X*, *y=None*)

Example

```
>>> import ants
>>> scaler = ants.contrib.ScaleImage((1.2,1.2))
>>> img2d = ants.image_read(ants.get_data('r16'))
>>> img2d_r = scaler.transform(img2d)
>>> ants.plot(img2d, img2d_r)
>>> scaler = ants.contrib.ScaleImage((1.2,1.2,1.2))
>>> img3d = ants.image_read(ants.get_data('mni'))
>>> img3d_r = scaler.transform(img3d)
>>> ants.plot(img3d, img3d_r)
```

class ShiftScaleIntensity(*shift*, *scale*)

Bases: object

Shift and scale the intensity of an ANTsImage

transform(*X*, *y=None*)

Transform an image by shifting and scaling its intensity values.

Parameters

- **X** (ants.core.ANTsImage) – image to transform
- **y** (ANTsImage (optional)) – another image to transform.

Example

```
>>> import ants
>>> shiftscaler = ants.contrib.ShiftScaleIntensity(10,2.)
>>> img2d = ants.image_read(ants.get_data('r16'))
>>> img2d_r = shiftscaler.transform(img2d)
>>> print(img2d.min(), ',', img2d.max(), ' -> ', img2d_r.min(), ',', img2d_r.
↪max())
>>> img3d = ants.image_read(ants.get_data('mni'))
>>> img3d_r = shiftscaler.transform(img3d)
```

(continues on next page)

(continued from previous page)

```
>>> print(img3d.min(), ',' , img3d.max(), ',', ' -> ', img3d_r.min(), ',' ,␣
↪img3d_r.max())
```

class SigmoidIntensity(*min_val*, *max_val*, *alpha*, *beta*)

Bases: object

Transform an image using a sigmoid function

transform(*X*, *y=None*)

Transform an image by applying a sigmoid function.

Parameters

- **X** (ants.core.ANTsImage) – image to transform
- **y** (ANTsImage (optional)) – another image to transform.

Example

```
>>> import ants
>>> sigscaler = ants.contrib.SigmoidIntensity(0,1,1,1)
>>> img2d = ants.image_read(ants.get_data('r16'))
>>> img2d_r = sigscaler.transform(img2d)
>>> img3d = ants.image_read(ants.get_data('mni'))
>>> img3d_r = sigscaler.transform(img3d)
```

class TranslateImage(*translation*, *reference=None*, *interp='linear'*)

Bases: object

Translate an image in physical space. This function calls highly optimized ITK/C++ code.

transform(*X*, *y=None*)

Example

```
>>> import ants
>>> translater = ants.contrib.TranslateImage((40,0))
>>> img2d = ants.image_read(ants.get_data('r16'))
>>> img2d_r = translater.transform(img2d)
>>> ants.plot(img2d, img2d_r)
>>> translater = ants.contrib.TranslateImage((40,0,0))
>>> img3d = ants.image_read(ants.get_data('mni'))
>>> img3d_r = translater.transform(img3d)
>>> ants.plot(img3d, img3d_r, axis=2)
```

8.1.3 ants.core

Modules

ants_image	
ants_image_io	Image IO
ants_metric	ANTs ImageToImageMetric class

continues on next page

Table 9 – continued from previous page

ants_metric_io	
ants_transform	
ants_transform_io	

ants.core.ants_image

Functions

copy_image_info(reference, target)	Copy origin, direction, and spacing from one antsImage to another
from_pointer(pointer)	
get_direction(image)	Get direction of ANTsImage
get_origin(image)	Get origin of ANTsImage
get_spacing(image)	Get spacing of ANTsImage
is_image(object)	
set_direction(image, direction)	Set direction of ANTsImage
set_origin(image, origin)	Set origin of ANTsImage
set_spacing(image, spacing)	Set spacing of ANTsImage

Classes

ANTsImage(pointer)	

class ANTsImage(*pointer*)

Bases: `object`

abp_n4(*intensity_truncation=(0.025, 0.975, 256)*, *mask=None*, *usen3=False*)

Truncate outlier intensities and bias correct with the N4 algorithm.

ANTsR function: *abpN4*

Parameters

- **image** (`ants.core.ANTsImage`) – image to correct and truncate
- **intensity_truncation** (`typing.Tuple`) – quantiles for intensity truncation
- **mask** (`ANTsImage (optional)`) – mask for bias correction
- **usen3** (`bool`) – if True, use N3 bias correction instead of N4

Return type

`ants.core.ANTsImage`

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> image2 = ants.abp_n4(image)
```

abs(*axis=None*)

Return absolute value of image

add_noise_to_image(*noise_model*, *noise_parameters*)

Add noise to an image using additive Gaussian, salt-and-pepper, shot, or speckle noise.

Parameters

- **image** (ants.core.ANTsImage) – scalar image.
- **noise_model** (str) – 'additivegaussian', 'saltandpepper', 'shot', or 'speckle'.
- **noise_parameters** (tuple or numpy.ndarray or float) – 'additivegaussian': (mean, standardDeviation) 'saltandpepper': (probability, saltValue, pepperValue) 'shot': scale 'speckle': standardDeviation

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> noise_image = ants.add_noise_to_image(image, 'additivegaussian', (0.0, 1.
↪0))
>>> noise_image = ants.add_noise_to_image(image, 'saltandpepper', (0.1, 0.0,␣
↪100.0))
>>> noise_image = ants.add_noise_to_image(image, 'shot', 1.0)
>>> noise_image = ants.add_noise_to_image(image, 'speckle', 1.0)
```

allclose(*image2*)

Check if two images have the same array values

anti_alias()

Apply Anti-Alias filter to a binary image

ANTsR function: N/A

Parameters
image (ants.core.ANTsImage) – binary image to which anti-aliasing will be applied

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> img = ants.image_read(ants.get_data('r16'))
>>> mask = ants.get_mask(img)
>>> mask_aa = ants.anti_alias(mask)
>>> ants.plot(mask)
>>> ants.plot(mask_aa)
```

apply(*fn*)

Apply an arbitrary function to ANTsImage.

Parameters
fn (python function or lambda) – function to apply to ENTIRE image at once

Returns
image with function applied to it

Return type
ants.core.ANTsImage

argmax(*axis=None*)

Return argmax along specified axis

argmin(*axis=None*)

Return argmin along specified axis

argrange(*axis=None*)

Return argrange along specified axis

astype(*dtype*)

Cast & clone an ANTsImage to a given numpy datatype.

Map:
uint8 : unsigned char uint32 : unsigned int float32 : float float64 : double

clone(*pixeltype=None*)

Create a copy of the given ANTsImage with the same data and info, possibly with a different data type for the image data. Only supports casting to uint8 (unsigned char), uint32 (unsigned int), float32 (float), and float64 (double)

Parameters
pixeltype (string (optional)) – if None, the pixeltype will be the same as the cloned ANTsImage. Otherwise, the data will be cast to this type. This can be a numpy type or an ITK type. Options:

'unsigned char' or 'uint8', 'unsigned int' or 'uint32', 'float' or 'float32', 'double' or 'float64'

Return type
ants.core.ANTsImage

property components

crop_image(*label_image=None*, *label=1*)

Use a label image to crop a smaller ANTsImage from within a larger ANTsImage

ANTsR function: *cropImage*

Parameters

- **image** (ants.core.ANTsImage) – image to crop
- **label_image** (ants.core.ANTsImage) – image with label values. If not supplied, estimated from data.
- **label** (int) – the label value to use

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> fi = ants.image_read( ants.get_ants_data('r16') )
>>> cropped = ants.crop_image(fi)
>>> fi2 = ants.merge_channels([fi,fi])
>>> cropped2 = ants.crop_image(fi2)
>>> cropped = ants.crop_image(fi, fi, 100 )
```

crop_indices(*lowerind*, *upperind*)

Create a proper ANTsImage sub-image by indexing the image with indices. This is similar to but different from array sub-setting in that the resulting sub-image can be decropped back into its place without having to store its original index locations explicitly.

ANTsR function: *cropIndices*

Parameters

- **image** (ants.core.ANTsImage) – image to crop
- **lowerind** (list/tuple of integers) – vector of lower index, should be length image dimensionality
- **upperind** (list/tuple of integers) – vector of upper index, should be length image dimensionality

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> fi = ants.image_read( ants.get_ants_data("r16"))
>>> cropped = ants.crop_indices( fi, (10,10), (100,100) )
>>> cropped = ants.smooth_image( cropped, 5 )
>>> decropped = ants.decrop_image( cropped, fi )
```

decrop_image(*full_image*)

The inverse function for *ants.crop_image*

ANTsR function: *decropImage*

Parameters

- **cropped_image** (ants.core.ANTsImage) – cropped image
- **full_image** (ants.core.ANTsImage) – image in which the cropped image will be put back

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> fi = ants.image_read(ants.get_ants_data('r16'))
>>> mask = ants.get_mask(fi)
>>> cropped = ants.crop_image(fi, mask, 1)
```

(continues on next page)

(continued from previous page)

```
>>> cropped = ants.smooth_image(cropped, 1)
>>> decropped = ants.decrop_image(cropped, fi)
```

denoise_image(*mask=None*, *shrink_factor=1*, *p=1*, *r=2*, *noise_model='Rician'*, *v=0*)

Denoise an image using a spatially adaptive filter originally described in J. V. Manjon, P. Coupe, Luis Marti-Bonmati, D. L. Collins, and M. Robles. Adaptive Non-Local Means Denoising of MR Images With Spatially Varying Noise Levels, Journal of Magnetic Resonance Imaging, 31:192-203, June 2010.

ANTsR function: *denoiseImage*

Parameters

- **image** (ants.core.ANTsImage) – scalar image to denoise.
- **mask** (ants.core.ANTsImage) – to limit the denoise region.
- **shrink_factor** (scalar) – downsampling level performed within the algorithm.
- **p** (int or character of format '2x2' where the x separates vector entries) – patch radius for local sample.
- **r** (int or character of format '2x2' where the x separates vector entries) – search radius from which to choose extra local samples.
- **noise_model** (str) – 'Rician' or 'Gaussian'

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> import numpy as np
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> # add fairly large salt and pepper noise
>>> imagenoise = image + np.random.randn(*image.shape).astype('float32')*5
>>> imagedenoise = ants.denoise_image(imagenoise, ants.get_mask(image))
```

property dimension

property direction

Get image direction

Return type
tuple

property dtype

flatten()

Flatten image data

get_center_of_mass()

Compute an image center of mass in physical space which is defined as the mean of the intensity weighted voxel coordinate system.

ANTsR function: *getCenterOfMass*

Parameters
image (ants.core.ANTsImage) – image from which center of mass will be computed

Return type
scalar

Example

```
>>> fi = ants.image_read( ants.get_ants_data("r16"))
>>> com1 = ants.get_center_of_mass( fi )
>>> fi = ants.image_read( ants.get_ants_data("r64"))
>>> com2 = ants.get_center_of_mass( fi )
```

get_centroids(*clustparam=0*)

Reduces a variate/statistical/network image to a set of centroids describing the center of each stand-alone non-zero component in the image

ANTsR function: *getCentroids*

Parameters

- **image** (ants.core.ANTsImage) – image from which centroids will be calculated
- **clustparam** (int) – look at regions greater than or equal to this size

Return type
numpy.ndarray

Example

```
>>> import ants
>>> image = ants.image_read( ants.get_ants_data( "r16" ) )
>>> image = ants.threshold_image( image, 90, 120 )
>>> image = ants.label_clusters( image, 10 )
>>> cents = ants.get_centroids( image )
```

get_mask(*low_thresh=None*, *high_thresh=None*, *cleanup=2*)

Get a binary mask image from the given image after thresholding

ANTsR function: *getMask*

Parameters

- **image** (ants.core.ANTsImage) – image from which mask will be computed. Can be an antsImage of 2, 3 or 4 dimensions.
- **low_thresh** (scalar (optional)) – An inclusive lower threshold for voxels to be included in the mask. If not given, defaults to image mean.
- **high_thresh** (scalar (optional)) – An inclusive upper threshold for voxels to be included in the mask. If not given, defaults to image max
- **cleanup** (int) – If > 0, morphological operations will be applied to clean up the mask by eroding away small or weakly-connected areas, and closing holes. If cleanup is >0, the following steps are applied
  1. Erosion with radius 2 voxels
  2. Retain largest component
  3. Dilation with radius 1 voxel
  4. Morphological closing

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read( ants.get_ants_data('r16') )
>>> mask = ants.get_mask(image)
```

get_neighborhood_at_voxel(*center*, *kernel*, *physical_coordinates=False*)

Get a hypercube neighborhood at a voxel. Get the values in a local neighborhood of an image.

ANTsR function: *getNeighborhoodAtVoxel*

Parameters

- **image** (ants.core.ANTsImage) – image to get values from.
- **center** (tuple/list) – indices for neighborhood center
- **kernel** (tuple/list) – either a collection of values for neighborhood radius (in voxels) or a binary collection of the same dimension as the image, specifying the shape of the neighborhood to extract
- **physical_coordinates** (bool) – whether voxel indices and offsets should be in voxel or physical coordinates

Returns

values
[ndarray] array of neighborhood values at the voxel

indices
[ndarray] matrix providing the coordinates for each value

Return type
dictionary w/ following key-value pairs

Example

```
>>> import ants
>>> img = ants.image_read(ants.get_ants_data('r16'))
>>> center = (2,2)
>>> radius = (3,3)
>>> retval = ants.get_neighborhood_at_voxel(img, center, radius)
```

get_neighborhood_in_mask(*mask*, *radius*, *physical_coordinates=False*, *boundary_condition=None*, *spatial_info=False*, *get_gradient=False*)

Get neighborhoods for voxels within mask.

This converts a scalar image to a matrix with rows that contain neighbors around a center voxel

ANTsR function: *getNeighborhoodInMask*

Parameters

- **image** (ants.core.ANTsImage) – image to get values from
- **mask** (ants.core.ANTsImage) – image indicating which voxels to examine. Each voxel > 0 will be used as the center of a neighborhood

- **radius** (tuple/list) – array of values for neighborhood radius (in voxels)
- **physical_coordinates** (bool) – whether voxel indices and offsets should be in voxel or physical coordinates
- **boundary_condition** (string (optional)) –

  how to handle voxels in a neighborhood, but not in the mask.
  None : fill values with *NaN image* : use image value, even if not in mask *mean* : use mean of all non-NaN values for that neighborhood
- **spatial_info** (bool) – whether voxel locations and neighborhood offsets should be returned along with pixel values.
- **get_gradient** (bool) – whether a matrix of gradients (at the center voxel) should be returned in addition to the value matrix (WIP)

Returns

- if spatial_info is False –

  if get_gradient is False:

  ndarray
  an array of pixel values where the number of rows is the size of the neighborhood and there is a column for each voxel

  else if get_gradient is True:

  dictionary w/ following key-value pairs:

  values
  [ndarray] array of pixel values where the number of rows is the size of the neighborhood and there is a column for each voxel.

  gradients
  [ndarray] array providing the gradients at the center voxel of each neighborhood
- else if spatial_info is True –

  dictionary w/ following key-value pairs:

  values
  [ndarray] array of pixel values where the number of rows is the size of the neighborhood and there is a column for each voxel.

  indices
  [ndarray] array provinding the center coordinates for each neighborhood

  offsets
  [ndarray] array providing the offsets from center for each voxel in a neighborhood

Example

```
>>> import ants
>>> r16 = ants.image_read(ants.get_ants_data('r16'))
>>> mask = ants.get_mask(r16)
>>> mat = ants.get_neighborhood_in_mask(r16, mask, radius=(2,2))
```

get_orientation()

property has_components

hausdorff_distance(*image2*)

Get Hausdorff distance between non-zero pixels in two images

ANTsR function: *hausdorffDistance*

Parameters

- **image** (*source*) – Source image
- **target_image** (ants.core.ANTsImage) – Target image

Return type
data frame with "Distance" and "AverageDistance"

Example

```
>>> import ants
>>> r16 = ants.image_read( ants.get_ants_data('r16') )
>>> r64 = ants.image_read( ants.get_ants_data('r64') )
>>> s16 = ants.kmeans_segmentation( r16, 3 )['segmentation']
>>> s64 = ants.kmeans_segmentation( r64, 3 )['segmentation']
>>> stats = ants.hausdorff_distance(s16, s64)
```

hessian_objectness(*object_dimension=1*, *is_bright_object=True*, *sigma_min=0.1*, *sigma_max=10*, *number_of_sigma_steps=10*, *use_sigma_logarithmic_spacing=True*, *alpha=0.5*, *beta=0.5*, *gamma=5.0*, *set_scale_objectness_measure=True*)

Interface to ITK filter. Based on the paper by Westin et al., "Geometrical Diffusion Measures for MRI from Tensor Basis Analysis" and Luca Antiga's Insight Journal paper http://hdl.handle.net/1926/576.

Parameters

- **image** (ants.core.ANTsImage) – scalar image.
- **object_dimension** (unsigned int) – 0: 'sphere', 1: 'line', or 2: 'plane'.
- **is_bright_object** (bool) – Set 'true' for enhancing bright objects and 'false' for dark objects.
- **sigma_min** (float) – Define scale domain for feature extraction.
- **sigma_max** (float) – Define scale domain for feature extraction.
- **number_of_sigma_steps** (unsigned int) – Define number of samples for scale space.
- **use_sigma_logarithmic_spacing** (bool) – Define sample spacing the for scale space.
- **alpha** (float) – Hessian filter parameter.
- **beta** (float) – Hessian filter parameter.
- **gamma** (float) – Hessian filter parameter.
- **set_scale_objectness_measure** (bool) – ...

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> hessian_object_image = ants.hessian_objectness(image)
```

histogram_equalize_image(*number_of_histogram_bins=256*)

Histogram equalize image

from http://www.janeriksolem.net/histogram-equalization-with-python-and.html

Parameters

- **image** (ants.core.ANTsImage) – source image
- **number_of_histogram_bins** (int) – number of bins for cumulative histogram

Example

```
>>> import ants
>>> src_img = ants.image_read(ants.get_data('r16'))
>>> src_img_eq = ants.histogram_equalize_image(src_img)
```

histogram_match_image(*reference_image*, *number_of_histogram_bins=255*, *number_of_match_points=64*, *use_threshold_at_mean_intensity=False*)

Histogram match source image to reference image.

Parameters

- **source_image** (ants.core.ANTsImage) – source image
- **reference_image** (ants.core.ANTsImage) – reference image
- **number_of_histogram_bins** (int) – number of bins for source and reference histograms
- **number_of_match_points** (int) – number of points for histogram matching
- **use_threshold_at_mean_intensity** (bool) – see ITK description.

Example

```
>>> import ants
>>> src_img = ants.image_read(ants.get_data('r16'))
>>> ref_img = ants.image_read(ants.get_data('r64'))
>>> src_ref = ants.histogram_match_image(src_img, ref_img)
```

histogram_match_image2(*reference_image*, *source_mask=None*, *reference_mask=None*, *match_points=64*, *transform_domain_size=255*)

Transform image intensities based on histogram mapping.

Apply B-spline 1-D maps to an input image for intensity warping.

Parameters

- **source_image** (ants.core.ANTsImage) – source image
- **reference_image** (ants.core.ANTsImage) – reference image
- **source_mask** (ants.core.ANTsImage) – source mask

- **reference_mask** (ants.core.ANTsImage) – reference mask
- **match_points** (int or tuple) – Parametric points at which the intensity transform displacements are specified between [0, 1], i.e. quantiles. Alternatively, a single number can be given and the sequence is linearly spaced in [0, 1].
- **transform_domain_size** (int) – Defines the sampling resolution of the B-spline warping.

Return type
ANTs image

Example

```
>>> import ants
>>> src_img = ants.image_read(ants.get_data('r16'))
>>> ref_img = ants.image_read(ants.get_data('r64'))
>>> src_ref = ants.histogram_match_image2(src_img, ref_img)
```

iMath(*operation*, **args*)

Perform various (often mathematical) operations on the input image/s. Additional parameters should be specific for each operation. See the the full iMath in ANTs, on which this function is based.

ANTsR function: *iMath*

Parameters

- **image** (ants.core.ANTsImage) – input object, usually antsImage
- **operation** – a string e.g. "GetLargestComponent" ... the special case of "GetOperations" or "GetOperationsFull" will return a list of operations and brief description. Some operations may not be valid (WIP), but most are.
- ***args** (non-keyword arguments) – additional parameters specific to the operation

Example

```
>>> import ants
>>> img = ants.image_read(ants.get_ants_data('r16'))
>>> img2 = ants.iMath(img, 'Canny', 1, 5, 12)
```

iMath_GC(*radius=1*)

iMath_GD(*radius=1*)

iMath_GE(*radius=1*)

iMath_GO(*radius=1*)

iMath_MC(*radius=1*, *value=1*, *shape=1*, *parametric=False*, *lines=3*, *thickness=1*, *include_center=False*)

iMath_MD(*radius=1*, *value=1*, *shape=1*, *parametric=False*, *lines=3*, *thickness=1*, *include_center=False*)

iMath_ME(*radius=1*, *value=1*, *shape=1*, *parametric=False*, *lines=3*, *thickness=1*, *include_center=False*)

iMath_MO(*radius=1*, *value=1*, *shape=1*, *parametric=False*, *lines=3*, *thickness=1*, *include_center=False*)

iMath_canny(*sigma*, *lower*, *upper*)

iMath_fill_holes(*hole_type=2*)

iMath_get_largest_component(*min_size=50*)

iMath_grad(*sigma=0.5*, *normalize=False*)

iMath_histogram_equalization(*alpha*, *beta*)

iMath_laplacian(*sigma=0.5*, *normalize=False*)

iMath_maurer_distance(*foreground=1*)

iMath_normalize()

iMath_pad(*padding*)

iMath_perona_malik(*conductance=0.25*, *n_iterations=1*)

iMath_propagate_labels_through_mask(*labels*, *stopping_value=100*, *propagation_method=0*)

```
>>> import ants
>>> wms = ants.image_read('~/desktop/wms.nii.gz')
>>> thal = ants.image_read('~/desktop/thal.nii.gz')
>>> img2 = ants.iMath_propagate_labels_through_mask(wms, thal, 500, 0)
```

iMath_sharpen()

iMath_truncate_intensity(*lower_q*, *upper_q*, *n_bins=64*)

```
>>> import ants
>>> img = ants.image_read(ants.get_ants_data('r16'))
>>> ants.iMath_truncate_intensity( img, 0.2, 0.8 )
```

image_mutual_information(*image2*)

Compute mutual information between two ANTsImage types

ANTsR function: *antsImageMutualInformation*

Parameters

- **image1** (ants.core.ANTsImage) – image 1
- **image2** (ants.core.ANTsImage) – image 2

Return type
scalar

Example

```
>>> import ants
>>> fi = ants.image_read( ants.get_ants_data('r16') ).clone('float')
>>> mi = ants.image_read( ants.get_ants_data('r64') ).clone('float')
>>> mival = ants.image_mutual_information(fi, mi) # -0.1796141
```

image_physical_space_consistency(*image2*, *tolerance=0.01*, *datatype=False*)

Check if two or more ANTsImage objects occupy the same physical space

ANTsR function: *antsImagePhysicalSpaceConsistency*

Parameters

- ***images** (ANTsImages) – images to compare
- **tolerance** (float) – tolerance when checking origin and spacing
- **data_type** (bool) – If true, also check that the image data types are the same

Returns
true if images share same physical space, false otherwise

Return type
bool

image_similarity(*moving_image*, *metric_type='MeanSquares'*, *fixed_mask=None*, *moving_mask=None*, *sampling_strategy='regular'*, *sampling_percentage=1.0*)

Measure similarity between two images. NOTE: Similarity is actually returned as distance (i.e. dissimilarity) per ITK/ANTs convention. E.g. using Correlation metric, the similarity of an image with itself returns -1.

ANTsR function: *imageSimilarity*

Parameters

- **fixed** (ants.core.ANTsImage) – the fixed image
- **moving** (ants.core.ANTsImage) – the moving image
- **metric_type** (str) –

  image metric to calculate
  MeanSquares Correlation ANTSNeighborhoodCorrelation MattesMutualInformation JointHistogramMutualInformation Demons
- **fixed_mask** (ANTsImage (optional)) – mask for the fixed image
- **moving_mask** (ANTsImage (optional)) – mask for the moving image
- **sampling_strategy** (string (optional)) –

  sampling strategy, default is full sampling
  None (Full sampling) random regular
- **sampling_percentage** (scalar) – percentage of data to sample when calculating metric Must be between 0 and 1

Return type
scalar

Example

```
>>> import ants
>>> x = ants.image_read(ants.get_ants_data('r16'))
>>> y = ants.image_read(ants.get_ants_data('r30'))
>>> metric = ants.image_similarity(x,y,metric_type='MeanSquares')
```

image_to_cluster_images(*min_cluster_size=50*, *min_thresh=1e-06*, *max_thresh=1*)

Converts an image to several independent images.

Produces a unique image for each connected component 1 through N of size > min_cluster_size

ANTsR function: *image2ClusterImages*

Parameters

- **image** (ants.core.ANTsImage) – input image

- **min_cluster_size** (int) – throw away clusters smaller than this value
- **min_thresh** (scalar) – threshold to a statistical map
- **max_thresh** (scalar) – threshold to a statistical map

Return type
list of ANTsImage types

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> image = ants.threshold_image(image, 1, 1e15)
>>> image_cluster_list = ants.image_to_cluster_images(image)
```

image_write(*filename*, *ri=False*)

Write an ANTsImage to file

ANTsR function: *antsImageWrite*

Parameters

- **image** (ants.core.ANTsImage) – image to save to file
- **filename** (str) – name of file to which image will be saved
- **ri** (bool) –

  if True, return image. This allows for using this function in a pipeline:

  ```
  >>> img2 = img.smooth_image(2.).image_write(file1,␣
  ↪ri=True).threshold_image(0,20).image_write(file2,␣
  ↪ri=True)
  ```

  if False, do not return image

property is_rgb

label_clusters(*min_cluster_size=50*, *min_thresh=1e-06*, *max_thresh=1*, *fully_connected=False*)

This will give a unique ID to each connected component 1 through N of size > min_cluster_size

ANTsR function: *labelClusters*

Parameters

- **image** (ants.core.ANTsImage) – input image e.g. a statistical map
- **min_cluster_size** (int) – throw away clusters smaller than this value
- **min_thresh** (scalar) – threshold to a statistical map
- **max_thresh** (scalar) – threshold to a statistical map
- **fully_connected** (bool) – boolean sets neighborhood connectivity pattern

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read( ants.get_ants_data('r16') )
>>> timageFully = ants.label_clusters( image, 10, 128, 150, True )
>>> timageFace = ants.label_clusters( image, 10, 128, 150, False )
```

label_geometry_measures(*intensity_image=None*)

Wrapper for the ANTs funtion LabelGeometryMeasures

ANTsR function: *labelGeometryMeasures*

Parameters

- **label_image** (ants.core.ANTsImage) – image on which to compute geometry. Labels must be representable as uint32.
- **intensity_image** (ANTsImage (optional)) – image with intensity values

Return type
pandas.DataFrame

Example

```
>>> import ants
>>> fi = ants.image_read( ants.get_ants_data('r16') )
>>> seg = ants.kmeans_segmentation( fi, 3 )['segmentation']
>>> geom = ants.label_geometry_measures(seg,fi)
```

label_image_centroids(*physical=False*, *convex=True*, *verbose=False*)

Converts a label image to coordinates summarizing their positions

ANTsR function: *labelImageCentroids*

Parameters

- **image** (ants.core.ANTsImage) – image of integer labels
- **physical** (bool) – whether you want physical space coordinates or not
- **convex** (bool) – if True, return centroid if False return point with min average distance to other points with same label

Returns

labels
[1D-ndarray] array of label values

vertices
[pd.DataFrame] coordinates of label centroids

Return type
dictionary w/ following key-value pairs

Example

```
>>> import ants
>>> import numpy as np
>>> image = ants.from_numpy(np.asarray([[[0,2],[1,3]],[[4,6],[5,7]]]).astype(
```

(continues on next page)

(continued from previous page)

```
↪'float32'))
>>> labels = ants.label_image_centroids(image)
```

label_overlap_measures(*target_image*)

Get overlap measures from two label images (e.g., Dice)

ANTsR function: *labelOverlapMeasures*

Parameters

- **image** (*source*) – Source image
- **target_image** (ants.core.ANTsImage) – Target image

Return type
data frame with measures for each label and all labels combined

Example

```
>>> import ants
>>> r16 = ants.image_read( ants.get_ants_data('r16') )
>>> r64 = ants.image_read( ants.get_ants_data('r64') )
>>> s16 = ants.kmeans_segmentation( r16, 3 )['segmentation']
>>> s64 = ants.kmeans_segmentation( r64, 3 )['segmentation']
>>> stats = ants.label_overlap_measures(s16, s64)
```

label_stats(*label_image*)

Get label statistics from an image. The labels must be representable as uint32.

ANTsR function: *labelStats*

Parameters

- **image** (ants.core.ANTsImage) – Image from which statistics will be calculated
- **label_image** (ants.core.ANTsImage) – Label image

Return type
pandas.DataFrame

Example

```
>>> import ants
>>> image = ants.image_read( ants.get_ants_data('r16') , 2 )
>>> image = ants.resample_image( image, (64,64), 1, 0 )
>>> mask = ants.get_mask(image)
>>> segs1 = ants.kmeans_segmentation( image, 3 )
>>> stats = ants.label_stats(image, segs1['segmentation'])
```

labels_to_matrix(*mask*, *target_labels=None*, *missing_val=nan*)

Convert a labeled image to an n x m binary matrix where n = number of voxels and m = number of labels. Only includes values inside the provided mask while including background (image == 0) for consistency with timeseries2matrix and other image to matrix operations.

ANTsR function: *labels2matrix*

Parameters

- **image** (ants.core.ANTsImage) – input label image

- **mask** (ants.core.ANTsImage) – defines domain of interest
- **target_labels** (list/tuple) – defines target regions to be returned. if the target label does not exist in the input label image, then the matrix will contain a constant value of missing_val (default None) in that row.
- **missing_val** (scalar) – value to use for missing label values

Return type
numpy.ndarray

Example

```
>>> import ants
>>> fi = ants.image_read(ants.get_ants_data('r16')).resample_image((60,60),1,0)
>>> mask = ants.get_mask(fi)
>>> labs = ants.kmeans_segmentation(fi,3)['segmentation']
>>> labmat = ants.labels_to_matrix(labs, mask)
```

list_to_ndimage(*image_list*)

Merge list of multiple scalar ANTsImage types of dimension into one ANTsImage of dimension plus one

ANTsR function: *mergeListToNDImage*

Parameters

- **image** (target image space) –
- **image_list** (list/tuple of ANTsImage types) – scalar images to merge into target image space

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> image2 = ants.image_read(ants.get_ants_data('r16'))
>>> imageTar = ants.make_image( ( *image2.shape, 2 ) )
>>> image3 = ants.list_to_ndimage( imageTar, [image,image2])
>>> image3.dimension == 3
```

mask_image(*mask*, *level=1*, *binarize=False*)

Mask an input image by a mask image. If the mask image has multiple labels, it is possible to specify which label(s) to mask at.

ANTsR function: *maskImage*

Parameters

- **image** (ants.core.ANTsImage) – Input image.
- **mask** (ants.core.ANTsImage) – Mask or label image.
- **level** (scalar or tuple of scalars) – Level(s) at which to threshold the mask. Can be a single value or an iterable of values.
- **binarize** (bool) – whether to binarize the output image. This computes an intersection between (image != 0) and (mask == i), for all i in level.

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> myimage = ants.image_read(ants.get_ants_data('r16'))
>>> mask = ants.get_mask(myimage)
>>> myimage_mask = ants.mask_image(myimage, mask, 3)
>>> seg = ants.kmeans_segmentation(myimage, 3)
>>> myimage_mask = ants.mask_image(myimage, seg['segmentation'], (1,3))
```

matrix_to_timeseries(*matrix*, *mask=None*)

converts a matrix to a ND image.

ANTsR function: *matrix2timeseries*

Parameters

- **image** (reference ND image) –
- **matrix** (matrix to convert to image) –
- **mask** (mask image defining voxels of interest) –

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> img = ants.make_image( (10,10,10,5 ) )
>>> mask = ants.ndimage_to_list( img )[0] * 0
>>> mask[ 4:8, 4:8, 4:8 ] = 1
>>> mat = ants.timeseries_to_matrix( img, mask = mask )
>>> img2 = ants.matrix_to_timeseries( img,  mat, mask)
```

max(*axis=None*)

Return max along specified axis

mean(*axis=None*)

Return mean along specified axis

median(*axis=None*)

Return median along specified axis

min(*axis=None*)

Return min along specified axis

morphology(*operation*, *radius*, *mtype='binary'*, *value=1*, *shape='ball'*, *radius_is_parametric=False*, *thickness=1*, *lines=3*, *include_center=False*)

Apply morphological operations to an image

ANTsR function: *morphology*

Parameters

- **input** (ants.core.ANTsImage) – input image
- **operation** (str) –

operation to apply
”close” Morpholgical closing “dilate” Morpholgical dilation “erode” Morpholgical erosion “open” Morpholgical opening

- **radius** (scalar) – radius of structuring element
- **mtype** (str) –

  type of morphology
  ”binary” Binary operation on a single value “grayscale” Grayscale operations
- **value** (scalar) – value to operation on (type=’binary’ only)
- **shape** (str) –

  shape of the structuring element (type=’binary’ only)
  ”ball” spherical structuring element “box” box shaped structuring element “cross” cross shaped structuring element “annulus” annulus shaped structuring element “polygon” polygon structuring element
- **radius_is_parametric** (bool) – used parametric radius boolean (shape=’ball’ and shape=’annulus’ only)
- **thickness** (scalar) – thickness (shape=’annulus’ only)
- **lines** (int) – number of lines in polygon (shape=’polygon’ only)
- **include_center** (bool) – include center of annulus boolean (shape=’annulus’ only)

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> fi = ants.image_read( ants.get_ants_data('r16') , 2 )
>>> mask = ants.get_mask( fi )
>>> dilated_ball = ants.morphology( mask, operation='dilate', radius=3, mtype=
↪'binary', shape='ball')
>>> eroded_box = ants.morphology( mask, operation='erode', radius=3, mtype=
↪'binary', shape='box')
>>> opened_annulus = ants.morphology( mask, operation='open', radius=5, mtype=
↪'binary', shape='annulus', thickness=2)
```

movie(*filename=None*, *writer=None*, *fps=30*)

Create and save a movie - mp4, gif, etc - of the various 2D slices of a 3D ants image

Try this:
conda install -c conda-forge ffmpeg

Example

```
>>> import ants
>>> mni = ants.image_read(ants.get_data('mni'))
>>> ants.movie(mni, filename='~/desktop/movie.mp4')
```

multi_label_morphology(*operation*, *radius*, *dilation_mask=None*, *label_list=None*, *force=False*)

Morphology on multi label images.

Wraps calls to iMath binary morphology. Additionally, dilation and closing operations preserve pre-existing labels. The choices of operation are:

Dilation: dilates all labels sequentially, but does not overwrite original labels. This reduces dependence on the intensity ordering of adjoining labels. Ordering dependence can still arise if two or more labels dilate into the same space - in this case, the label with the lowest intensity is retained. With a mask, dilated labels are multiplied by the mask and then added to the original label, thus restricting dilation to the mask region.

Erosion: Erodes labels independently, equivalent to calling iMath iteratively.

Closing: Close holes in each label sequentially, but does not overwrite original labels.

Opening: Opens each label independently, equivalent to calling iMath iteratively.

Parameters

- **image** (ants.core.ANTsImage) – Input image should contain only 0 for background and positive integers for labels.
- **operation** (str) – One of MD, ME, MC, MO, passed to iMath.
- **radius** (int) – radius of the morphological operation.
- **dilation_mask** (ants.core.ANTsImage) – Optional binary mask to constrain dilation only (eg dilate cortical label into WM).
- **label_list** (list or tuple or numpy.ndarray) – Optional list of labels, to perform operation upon. Defaults to all unique intensities in image.

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> img = ants.image_read(ants.get_data('r16'))
>>> labels = ants.get_mask(img,1,150) + ants.get_mask(img,151,225) * 2
>>> labels_dilated = ants.multi_label_morphology(labels, 'MD', 2)
>>> # should see original label regions preserved in dilated version
>>> # label N should have mean N and 0 variance
>>> print(ants.label_stats(labels_dilated, labels))
```

multiply_images(*image2*)

n3_bias_field_correction(*downsample_factor=3*)

N3 Bias Field Correction

ANTsR function: *n3BiasFieldCorrection*

Parameters

- **image** (ants.core.ANTsImage) – image to be bias corrected
- **downsample_factor** (scalar) – how much to downsample image before performing bias correction

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read( ants.get_ants_data('r16') )
>>> image_n3 = ants.n3_bias_field_correction(image)
```

n3_bias_field_correction2(*mask=None*, *rescale_intensities=False*, *shrink_factor=4*, *convergence={'iters': 50, 'tol': 1e-07}*, *spline_param=None*, *number_of_fitting_levels=4*, *return_bias_field=False*, *verbose=False*, *weight_mask=None*)

N3 Bias Field Correction

ANTsR function: *n3BiasFieldCorrection2*

Parameters

- **image** (ants.core.ANTsImage) – image to bias correct
- **mask** (ants.core.ANTsImage) – Input mask. If not specified, the entire image is used.
- **rescale_intensities** (bool) – At each iteration, a new intensity mapping is calculated and applied but there is nothing which constrains the new intensity range to be within certain values. The result is that the range can "drift" from the original at each iteration. This option rescales to the [min,max] range of the original image intensities within the user-specified mask. A mask is required to perform rescaling. Default is False in ANTsR/ANTsPy but True in ANTs.
- **shrink_factor** (scalar) – Shrink factor for multi-resolution correction, typically integer less than 4
- **convergence** (dict w/ keys `iters` and *tol*) – iters : maximum number of iterations tol : the convergence tolerance. Default tolerance is 1e-7 in ANTsR/ANTsPy but 0.0 in ANTs.
- **spline_param** (float or vector Parameter controlling number of control) – points in spline. Either single value, indicating the spacing in each direction, or vector with one entry per dimension of image, indicating the mesh size. If None, defaults to mesh size of 1 in all dimensions.
- **number_of_fitting_levels** (int) – Number of fitting levels per iteration.
- **return_bias_field** (bool) – Return bias field instead of bias corrected image.
- **verbose** (bool) – enables verbose output.
- **weight_mask** (ANTsImage (optional)) – antsImage of weight mask

Return type
ants.core.ANTsImage

Example

```
>>> image = ants.image_read( ants.get_ants_data('r16') )
>>> image_n3 = ants.n3_bias_field_correction2(image)
```

n4_bias_field_correction(*mask=None*, *rescale_intensities=False*, *shrink_factor=4*, *convergence={'iters': [50, 50, 50, 50], 'tol': 1e-07}*, *spline_param=None*, *return_bias_field=False*, *verbose=False*, *weight_mask=None*)

N4 Bias Field Correction

ANTsR function: *n4BiasFieldCorrection*

Parameters

- **image** (ants.core.ANTsImage) – image to bias correct
- **mask** (ants.core.ANTsImage) – Input mask. If not specified, the entire image is used.
- **rescale_intensities** (bool) – At each iteration, a new intensity mapping is calculated and applied but there is nothing which constrains the new intensity range to be within certain values. The result is that the range can "drift" from the original at each iteration. This option rescales to the [min,max] range of the original image intensities within the user-specified mask. A mask is required to perform rescaling. Default is False in ANTsR/ANTsPy but True in ANTs.
- **shrink_factor** (scalar) – Shrink factor for multi-resolution correction, typically integer less than 4
- **convergence** (dict w/ keys `iters` and *tol*) – iters : vector of maximum number of iterations for each level tol : the convergence tolerance. Default tolerance is 1e-7 in ANTsR/ANTsPy but 0.0 in ANTs.
- **spline_param** (float or vector) – Parameter controlling number of control points in spline. Either single value, indicating the spacing in each direction, or vector with one entry per dimension of image, indicating the mesh size. If None, defaults to mesh size of 1 in all dimensions.
- **return_bias_field** (bool) – Return bias field instead of bias corrected image.
- **verbose** (bool) – enables verbose output.
- **weight_mask** (ANTsImage (optional)) – antsImage of weight mask

Return type
ants.core.ANTsImage

Example

```
>>> image = ants.image_read( ants.get_ants_data('r16') )
>>> image_n4 = ants.n4_bias_field_correction(image)
```

ndimage_to_list()

Split a n dimensional ANTsImage into a list of n-1 dimensional ANTsImages

Parameters
image (ants.core.ANTsImage) – n-dimensional image to split

Return type
list of ANTsImage types

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> image2 = ants.image_read(ants.get_ants_data('r16'))
>>> imageTar = ants.make_image( ( *image2.shape, 2 ) )
>>> image3 = ants.list_to_ndimage( imageTar, [image,image2])
>>> image3.dimension == 3
```

(continues on next page)

(continued from previous page)

```
>>> images_unmerged = ants.ndimage_to_list( image3 )
>>> len(images_unmerged) == 2
>>> images_unmerged[0].dimension == 2
```

new_image_like(*data*)

Create a new ANTsImage with the same header information, but with a new image array.

Parameters

data (numpy.ndarray or py::capsule) – New array or pointer for the image. It must have the same shape as the current image data.

Return type

ants.core.ANTsImage

nonzero()

Return non-zero indices of image

numpy(*single_components=False*)

Get a numpy array copy representing the underlying image data. Altering this ndarray will have NO effect on the underlying image data.

Parameters

single_components (boolean (default is False)) – if True, keep the extra component dimension in returned array even if image only has one component (i.e. self.has_components == False)

Return type

numpy.ndarray

property orientation

property origin

Get image origin

Return type

tuple

pad_image(*shape=None*, *pad_width=None*, *value=0.0*, *return_padvals=False*)

Pad an image to have the given shape or to be isotropic.

Parameters

- **image** (ants.core.ANTsImage) – image to pad
- **shape** (tuple) –
  - if shape is given, the image will be padded in each dimension until it has this shape
  - if shape and pad_width are both None, the image will be padded along each dimension to match the largest existing dimension so that it has isotropic dimensions.
- **pad_width** (list of integers or list-of-list of integers) – How much to pad in each direction. If a single list is supplied (e.g., [4,4,4]), then the image will be padded by half that amount on both sides. If a list-of-list is supplied (e.g., [(0,4),(0,4),(0,4)]), then the image will be padded unevenly on the different sides
- **pad_value** (scalar) – value with which image will be padded

Example

```
>>> import ants
>>> img = ants.image_read(ants.get_data('r16'))
>>> img2 = ants.pad_image(img, shape=(300,300))
>>> mni = ants.image_read(ants.get_data('mni'))
>>> mni2 = ants.pad_image(mni)
>>> mni3 = ants.pad_image(mni, pad_width=[(0,4),(0,4),(0,4)])
>>> mni4 = ants.pad_image(mni, pad_width=(4,4,4))
```

property physical_shape

property pixeltype

plot(*overlay=None*, *blend=False*, *alpha=1*, *cmap='Greys_r'*, *overlay_cmap='turbo'*, *overlay_alpha=0.9*, *vminol=None*, *vmaxol=None*, *cbar=False*, *cbar_length=0.8*, *cbar_dx=0.0*, *cbar_vertical=True*, *axis=0*, *nslices=12*, *slices=None*, *ncol=None*, *slice_buffer=None*, *black_bg=True*, *bg_thresh_quant=0.01*, *bg_val_quant=0.99*, *domain_image_map=None*, *crop=False*, *scale=False*, *reverse=False*, *title=None*, *title_fontsize=20*, *title_dx=0.0*, *title_dy=0.0*, *filename=None*, *dpi=500*, *figsize=1.5*, *reorient=True*, *resample=True*)

Plot an ANTsImage.

Use mask_image and/or threshold_image to preprocess images to be be overlaid and display the overlays in a given range. See the wiki examples.

By default, images will be reoriented to 'LAI' orientation before plotting. So, if axis == 0, the images will be ordered from the left side of the brain to the right side of the brain. If axis == 1, the images will be ordered from the anterior (front) of the brain to the posterior (back) of the brain. And if axis == 2, the images will be ordered from the inferior (bottom) of the brain to the superior (top) of the brain.

ANTsR function: *plot.antsImage*

Parameters

- **image** (`ants.core.ANTsImage`) – image to plot
- **overlay** (`ants.core.ANTsImage`) – image to overlay on base image
- **cmap** (`str`) – colormap to use for base image. See matplotlib.
- **overlay_cmap** (`str`) – colormap to use for overlay images, if applicable. See matplotlib.
- **overlay_alpha** (`float`) – level of transparency for any overlays. Smaller value means the overlay is more transparent. See matplotlib.
- **axis** (`int`) – which axis to plot along if image is 3D
- **nslices** (`int`) – number of slices to plot if image is 3D
- **slices** (`list` or `tuple` of `integers`) – specific slice indices to plot if image is 3D. If given, this will override *nslices*. This can be absolute array indices (e.g. (80,100,120)), or this can be relative array indices (e.g. (0.4,0.5,0.6))
- **ncol** (`int`) – Number of columns to have on the plot if image is 3D.
- **slice_buffer** (`int`) – how many slices to buffer when finding the non-zero slices of a 3D images. So, if slice_buffer = 10, then the first slice in a 3D image will be the first non-zero slice index plus 10 more slices.
- **black_bg** (`bool`) – if True, the background of the image(s) will be black. if False, the background of the image(s) will be determined by the

values *bg_thresh_quant* and *bg_val_quant*.

- **bg_thresh_quant** (float) – if white_bg=True, the background will be determined by thresholding the image at the *bg_thresh* quantile value and setting the background intensity to the *bg_val* quantile value. This value should be in [0, 1] - somewhere around 0.01 is recommended.
  - equal to 1 will threshold the entire image
  - equal to 0 will threshold none of the image
- **bg_val_quant** (float) – if white_bg=True, the background will be determined by thresholding the image at the *bg_thresh* quantile value and setting the background intensity to the *bg_val* quantile value. This value should be in [0, 1]
  - equal to 1 is pure white
  - equal to 0 is pure black
  - somewhere in between is gray
- **domain_image_map** (ants.core.ANTsImage) – this input ANTsImage or list of ANTsImage types contains a reference image *domain_image* and optional reference mapping named *domainMap*. If supplied, the image(s) to be plotted will be mapped to the domain image space before plotting - useful for non-standard image orientations.
- **crop** (bool) – if true, the image(s) will be cropped to their bounding boxes, resulting in a potentially smaller image size. if false, the image(s) will not be cropped
- **scale** (bool or typing.Tuple) – if true, nothing will happen to intensities of image(s) and overlay(s) if false, dynamic range will be maximized when visualizing overlays if 2-tuple, the image will be dynamically scaled between these quantiles
- **reverse** (bool) – if true, the order in which the slices are plotted will be reversed. This is useful if you want to plot from the front of the brain first to the back of the brain, or vice-versa
- **title** (str) – add a title to the plot
- **filename** (str) – if given, the resulting image will be saved to this file
- **dpi** (int) – determines resolution of image if saved to file. Higher values result in higher resolution images, but at a cost of having a larger file size
- **resample** (bool) – if true, resample image if spacing is very unbalanced.

Example

```
>>> import ants
>>> import numpy as np
>>> img = ants.image_read(ants.get_data('r16'))
>>> segs = img.kmeans_segmentation(k=3)['segmentation']
>>> ants.plot(img, segs*(segs==1), crop=True)
>>> ants.plot(img, segs*(segs==1), crop=False)
>>> mni = ants.image_read(ants.get_data('mni'))
>>> segs = mni.kmeans_segmentation(k=3)['segmentation']
>>> ants.plot(mni, segs*(segs==1), crop=False)
```

plot_hist(*threshold=0.0*, *fit_line=False*, *normfreq=True*, *title=None*, *grid=True*, *xlabel=None*, *ylabel=None*, *facecolor='green'*, *alpha=0.75*)

Plot a histogram from an ANTsImage

Parameters
image (ants.core.ANTsImage) – image from which histogram will be created

plot_ortho(*overlay=None*, *reorient=True*, *blend=False*, *xyz=None*, *xyz_lines=True*, *xyz_color='red'*, *xyz_alpha=0.6*, *xyz_linewidth=2*, *xyz_pad=5*, *orient_labels=True*, *alpha=1*, *cmap='Greys_r'*, *overlay_cmap='jet'*, *overlay_alpha=0.9*, *cbar=False*, *cbar_length=0.8*, *cbar_dx=0.0*, *cbar_vertical=True*, *black_bg=True*, *bg_thresh_quant=0.01*, *bg_val_quant=0.99*, *crop=False*, *scale=False*, *domain_image_map=None*, *title=None*, *titlefontsize=24*, *title_dx=0*, *title_dy=0*, *text=None*, *textfontsize=24*, *textfontcolor='white'*, *text_dx=0*, *text_dy=0*, *filename=None*, *dpi=500*, *figsize=1.0*, *flat=False*, *transparent=True*, *resample=False*, *allow_xyz_change=True*)

Plot an orthographic view of a 3D image

Use mask_image and/or threshold_image to preprocess images to be be overlaid and display the overlays in a given range. See the wiki examples.

ANTsR function: N/A

image
[ANTsImage] image to plot

overlay
[ANTsImage] image to overlay on base image

xyz
[list or tuple of 3 integers] selects index location on which to center display if given, solid lines will be drawn to converge at this coordinate. This is useful for pinpointing a specific location in the image.

flat
[boolean] if true, the ortho image will be plot in one row if false, the ortho image will be a 2x2 grid with the bottom

left corner blank

cmap
[string] colormap to use for base image. See matplotlib.

overlay_cmap
[string] colormap to use for overlay images, if applicable. See matplotlib.

overlay_alpha
[float] level of transparency for any overlays. Smaller value means the overlay is more transparent. See matplotlib.

cbar: boolean
if true, a colorbar will be added to the plot

cbar_length: float
length of the colorbar relative to the image

cbar_dx: float
horizontal shift of the colorbar relative to the image

cbar_vertical: boolean
if true, the colorbar will be vertical, if false, it will be horizontal underneath the image

axis
[integer] which axis to plot along if image is 3D

black_bg
[boolean] if True, the background of the image(s) will be black. if False, the background of the image(s) will be determined by the

values *bg_thresh_quant* and *bg_val_quant*.

bg_thresh_quant
[float] if white_bg=True, the background will be determined by thresholding the image at the *bg_thresh* quantile value and setting the background intensity to the *bg_val* quantile value. This value should be in [0, 1] - somewhere around 0.01 is recommended.

- equal to 1 will threshold the entire image
- equal to 0 will threshold none of the image

bg_val_quant
[float] if white_bg=True, the background will be determined by thresholding the image at the *bg_thresh* quantile value and setting the background intensity to the *bg_val* quantile value. This value should be in [0, 1]

- equal to 1 is pure white
- equal to 0 is pure black
- somewhere in between is gray

domain_image_map
[ANTsImage] this input ANTsImage or list of ANTsImage types contains a reference image *domain_image* and optional reference mapping named *domainMap*. If supplied, the image(s) to be plotted will be mapped to the domain image space before plotting - useful for non-standard image orientations.

crop
[boolean] if true, the image(s) will be cropped to their bounding boxes, resulting in a potentially smaller image size. if false, the image(s) will not be cropped

scale
[boolean or 2-tuple] if true, nothing will happen to intensities of image(s) and overlay(s) if false, dynamic range will be maximized when visualizing overlays if 2-tuple, the image will be dynamically scaled between these quantiles

title
[string] add a title to the plot

filename
[string] if given, the resulting image will be saved to this file

dpi
[integer] determines resolution of image if saved to file. Higher values result in higher resolution images, but at a cost of having a larger file size

resample : resample image in case of unbalanced spacing

allow_xyz_change : boolean will attempt to adjust xyz after padding

```
>>> import ants
>>> mni = ants.image_read(ants.get_data('mni'))
>>> ants.plot_ortho(mni, xyz=(100,100,100))
>>> mni2 = mni.threshold_image(7000, mni.max())
>>> ants.plot_ortho(mni, overlay=mni2)
>>> ants.plot_ortho(mni, overlay=mni2, flat=True)
```

(continues on next page)

(continued from previous page)

```
>>> ants.plot_ortho(mni, overlay=mni2, xyz=(110,110,110), xyz_
↪lines=False,
                    text='Lines Turned Off', textfontsize=22)
>>> ants.plot_ortho(mni, mni2, xyz=(120,100,100),
                    text=' Example
```

Ortho Text', textfontsize=26,
title='Example Ortho Title', titlefontsize=26)

quantile(*q*, *nonzero=True*)

Get the quantile values from an ANTsImage

Examples

```
>>> img = ants.image_read(ants.get_data('r16'))
>>> ants.quantile(img, 0.5)
>>> ants.quantile(img, (0.5, 0.75))
```

range(*axis=None*)

Return range tuple along specified axis

reflect_image(*axis=None*, *tx=None*, *metric='mattes'*)

Reflect an image along an axis

ANTsR function: *reflectImage*

Parameters

- **image** (ants.core.ANTsImage) – image to reflect
- **axis** (integer (optional)) – which dimension to reflect across, numbered from 0 to imageDimension-1
- **tx** (string (optional)) – transformation type to estimate after reflection
- **metric** (str) – similarity metric for image registration. see antsRegistration.

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> fi = ants.image_read( ants.get_ants_data('r16'), 'float' )
>>> axis = 2
>>> asym = ants.reflect_image(fi, axis, 'Affine')['warpedmovout']
>>> asym = asym - fi
```

reorient_image2(*orientation='RAS'*)

Reorient an image. .. rubric:: Example

```
>>> import ants
>>> mni = ants.image_read(ants.get_data('mni'))
>>> mni2 = mni.reorient_image2()
```

resample_image(*resample_params*, *use_voxels=False*, *interp_type=1*)

Resample image by spacing or number of voxels with various interpolators. Works with multi-channel images.

ANTsR function: *resampleImage*

Parameters

- **image** (ants.core.ANTsImage) – input image
- **resample_params** (tuple/list) – vector of size dimension with numeric values
- **use_voxels** (bool) – True means interpret resample params as voxel counts
- **interp_type** (int) – one of 0 (linear), 1 (nearest neighbor), 2 (gaussian), 3 (windowed sinc), 4 (bspline)

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> fi = ants.image_read( ants.get_ants_data("r16"))
>>> finn = ants.resample_image(fi,(50,60),True,0)
>>> filin = ants.resample_image(fi,(1.5,1.5),False,1)
>>> img = ants.image_read( ants.get_ants_data("r16"))
>>> img = ants.merge_channels([img, img])
>>> outimg = ants.resample_image(img, (128,128), True)
```

resample_image_to_target(*target*, *interp_type='linear'*, *imagetype=0*, *verbose=False*, ***kwargs*)

Resample image by using another image as target reference. This function uses ants.apply_transform with an identity matrix to achieve proper resampling.

ANTsR function: *resampleImageToTarget*

Parameters

- **image** (ants.core.ANTsImage) – image to resample
- **target** (ants.core.ANTsImage) – image of reference, the output will be in this space and will have the same pixel type.
- **interp_type** (str) –

  Choice of interpolator. Supports partial matching.
  linear nearestNeighbor multiLabel for label images but genericlabel is preferred gaussian bSpline cosineWindowedSinc welchWindowedSinc hammingWindowedSinc lanczosWindowedSinc genericLabel use this for label images
- **imagetype** (int) – choose 0/1/2/3 mapping to scalar/vector/tensor/time-series
- **verbose** (bool) – print command and run verbose application of transform.
- **kwargs** (keyword arguments) – additional arugment passed to antsApplyTransforms C code

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> fi = ants.image_read(ants.get_ants_data('r16'))
>>> fi2mm = ants.resample_image(fi, (2,2), use_voxels=0, interp_type='linear')
>>> resampled = ants.resample_image_to_target(fi2mm, fi, verbose=True)
```

rgb_to_vector()

Convert an RGB ANTsImage to a Vector ANTsImage

Parameters
image (ants.core.ANTsImage) – RGB image to be converted

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> mni = ants.image_read(ants.get_data('mni'))
>>> mni_rgb = ants.scalar_to_rgb(mni)
>>> mni_vector = mni.rgb_to_vector()
>>> mni_rgb2 = mni.vector_to_rgb()
```

set_direction(*new_direction*)

Set image direction

Parameters
new_direction (numpy.ndarray or tuple or list) – updated direction for the image. should have one value for each dimension

Return type
None

set_origin(*new_origin*)

Set image origin

Parameters
new_origin (tuple or list) – updated origin for the image. should have one value for each dimension

Return type
None

set_spacing(*new_spacing*)

Set image spacing

Parameters
new_spacing (tuple or list) – updated spacing for the image. should have one value for each dimension

Return type
None

property shape

slice_image(*axis*, *idx*, *collapse_strategy=0*)

Slice an image.

Parameters

- **axis** (int) – Which axis.
- **idx** (int) – Which slice number.
- **collapse_strategy** (int) – Collapse strategy for sub-matrix: 0, 1, or 2. 0: collapse to sub-matrix if positive-definite. Otherwise throw an exception. Default. 1: Collapse to identity. 2: Collapse to sub-matrix if positive definite. Otherwise collapse to identity.

Example

```
>>> import ants
>>> mni = ants.image_read(ants.get_data('mni'))
>>> mni2 = ants.slice_image(mni, axis=1, idx=100)
```

smooth_image(*sigma*, *sigma_in_physical_coordinates=True*, *FWHM=False*, *max_kernel_width=32*)

Smooth an image

ANTsR function: *smoothImage*

Parameters

- **image** – Image to smooth
- **sigma** – Smoothing factor. Can be scalar, in which case the same sigma is applied to each dimension, or a vector of length dim(inimage) to specify a unique smoothness for each dimension.
- **sigma_in_physical_coordinates** (bool) – If true, the smoothing factor is in millimeters; if false, it is in pixels.
- **FWHM** (bool) – If true, sigma is interpreted as the full-width-half-max (FWHM) of the filter, not the sigma of a Gaussian kernel.
- **max_kernel_width** (scalar) – Maximum kernel width

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read( ants.get_ants_data('r16'))
>>> simage = ants.smooth_image(image, (1.2,1.5))
```

property spacing

Get image spacing

Return type
tuple

split_channels()

Split channels of a multi-channel ANTsImage into a collection of scalar ANTsImage types

Parameters
image (ants.core.ANTsImage) – multi-channel image to split

Return type
list of ANTsImage types

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'), 'float')
>>> image2 = ants.image_read(ants.get_ants_data('r16'), 'float')
>>> imagemerge = ants.merge_channels([image,image2])
>>> imagemerge.components == 2
>>> images_unmerged = ants.split_channels(imagemerge)
>>> len(images_unmerged) == 2
>>> images_unmerged[0].components == 1
```

std(*axis=None*)

Return std along specified axis

sum(*axis=None*, *keepdims=False*)

Return sum along specified axis

symmetrize_image()

Use registration and reflection to make an image symmetric

ANTsR function: N/A

Parameters

image (ants.core.ANTsImage) – image to make symmetric

Return type

ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read( ants.get_ants_data('r16') , 'float')
>>> simage = ants.symimage(image)
```

threshold_image(*low_thresh=None*, *high_thresh=None*, *inval=1*, *outval=0*, *binary=True*)

Converts a scalar image into a binary image by thresholding operations

ANTsR function: *thresholdImage*

Parameters

- **image** (ants.core.ANTsImage) – Input image to operate on
- **low_thresh** (scalar (optional)) – Lower edge of threshold window
- **hight_thresh** (scalar (optional)) – Higher edge of threshold window
- **inval** (scalar) – Output value for image voxels in between lothresh and hithresh
- **outval** (scalar) – Output value for image voxels lower than lothresh or higher than hithresh
- **binary** (bool) – if true, returns binary thresholded image if false, return binary thresholded image multiplied by original image

Return type

ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read( ants.get_ants_data('r16') )
>>> timage = ants.threshold_image(image, 0.5, 1e15)
```

timeseries_to_matrix(*mask=None*)

Convert a timeseries image into a matrix.

ANTsR function: *timeseries2matrix*

Parameters

- **image** (image whose slices we convert to a matrix. E.g. a 3D image of size) – x by y by z will convert to a z by x*y sized matrix
- **mask** (ANTsImage (optional)) – image containing binary mask. voxels in the mask are placed in the matrix

Returns

array with a row for each image shape = (N_IMAGES, N_VOXELS)

Return type

numpy.ndarray

Example

```
>>> import ants
>>> img = ants.make_image( (10,10,10,5 ) )
>>> mat = ants.timeseries_to_matrix( img )
```

to_file(*filename*)

Write the ANTsImage to file

Parameters

filename (str) – filepath to which the image will be written

to_filename(*filename*)

Write the ANTsImage to file

Parameters

filename (str) – filepath to which the image will be written

unique(*sort=False*)

Return unique set of values in image

vector_to_rgb()

Convert an Vector ANTsImage to a RGB ANTsImage

Parameters

image (ants.core.ANTsImage) – RGB image to be converted

Return type

ants.core.ANTsImage

Example

```
>>> import ants
>>> img = ants.image_read(ants.get_data('r16'), pixeltype='unsigned char')
>>> img_rgb = ants.scalar_to_rgb(img.clone())
>>> img_vec = img_rgb.rgb_to_vector()
>>> img_rgb2 = img_vec.vector_to_rgb()
```

view(*single_components=False*)

Geet a numpy array providing direct, shared access to the image data. IMPORTANT: If you alter the view, then the underlying image data will also be altered.

Parameters

single_components (boolean (default is False)) – if True, keep the extra component dimension in returned array even if image only has one component (i.e. self.has_components == False)

Return type

numpy.ndarray

weingarten_image_curvature(*sigma=1.0*, *opt='mean'*)

Uses the weingarten map to estimate image mean or gaussian curvature

ANTsR function: *weingartenImageCurvature*

Parameters

- **image** (ants.core.ANTsImage) – image from which curvature is calculated
- **sigma** (scalar) – smoothing parameter
- **opt** (str) – mean by default, otherwise *gaussian* or *characterize*

Return type

ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('mni')).resample_image((3,3,3))
>>> imagecurv = ants.weingarten_image_curvature(image)
```

copy_image_info(*reference*, *target*)

Copy origin, direction, and spacing from one antsImage to another

ANTsR function: *antsCopyImageInfo*

Parameters

- **reference** (ants.core.ANTsImage) – Image to get values from.
- **target** (ANTsImAGE) – Image to copy values to

Returns

Target image with reference header information

Return type

ants.core.ANTsImage

from_pointer(*pointer*)

get_direction(*image*)

Get direction of ANTsImage

ANTsR function: *antsGetDirection*

get_origin(*image*)

Get origin of ANTsImage

ANTsR function: *antsGetOrigin*

get_spacing(*image*)

Get spacing of ANTsImage

ANTsR function: *antsGetSpacing*

is_image(*object*)

set_direction(*image*, *direction*)

Set direction of ANTsImage

ANTsR function: *antsSetDirection*

set_origin(*image*, *origin*)

Set origin of ANTsImage

ANTsR function: *antsSetOrigin*

set_spacing(*image*, *spacing*)

Set spacing of ANTsImage

ANTsR function: *antsSetSpacing*

ants.core.ants_image_io

Image IO

Functions

clone(image[, pixeltype])	Create a copy of the given ANTsImage with the same data and info, possibly with a different data type for the image data.
copy(image[, pixeltype])	Create a copy of the given ANTsImage with the same data and info, possibly with a different data type for the image data.
dicom_read(directory[, pixeltype])	Read a set of dicom files in a directory into a single ANTsImage.
from_numpy(data[, origin, spacing, ...])	Create an ANTsImage object from a numpy array
from_numpy_like(data, image)	
image_clone(image[, pixeltype])	Clone an ANTsImage
image_header_info(filename)	Read file info from image header
image_read(filename[, dimension, pixeltype, ...])	Read an ANTsImage from file
image_write(image, filename[, ri])	Write an ANTsImage to file
make_image(shape[, voxval, spacing, origin, ...])	Make an image with given size and voxel values or fill a mask with a vector of voxel values.
new_image_like(image, data)	Create a new ANTsImage with the same header information, but with a new image array.

continues on next page

Table 12 – continued from previous page

read_image_metadata(filename)	Read metadata dictionary from image file.

dicom_read(*directory*, *pixeltype='float'*)

Read a set of dicom files in a directory into a single ANTsImage. The origin of the resulting 3D image will be the origin of the first dicom image read.

Parameters

directory (str) – folder in which all the dicom images exist

Return type

ants.core.ANTsImage

Example

```
>>> import ants
>>> img = ants.dicom_read('~/desktop/dicom-subject/')
```

from_numpy(*data*, *origin=None*, *spacing=None*, *direction=None*, *has_components=False*, *is_rgb=False*)

Create an ANTsImage object from a numpy array

ANTsR function: *as.antsImage*

Parameters

- **data** (numpy.ndarray) – image data array
- **origin** (tuple/list) – image origin
- **spacing** (tuple/list) – image spacing
- **direction** (list/ndarray) – image direction
- **has_components** (bool) – whether the image has vector components.
- **is_rgb** (bool) – whether the image is RGB. This implies has_components=True.

Returns

image with given data and any given information

Return type

ants.core.ANTsImage

from_numpy_like(*data*, *image*)

image_clone(*image*, *pixeltype=None*)

Clone an ANTsImage

ANTsR function: *antsImageClone*

Parameters

- **image** (ants.core.ANTsImage) – image to clone
- **pixeltype** (string (optional)) – new datatype for image

Return type

ants.core.ANTsImage

image_header_info(*filename*)

Read file info from image header

ANTsR function: *antsImageHeaderInfo*

Parameters
filename (str) – name of image file from which info will be read

Return type
dict

image_read(*filename*, *dimension=None*, *pixeltype='float'*, *reorient=False*)

Read an ANTsImage from file

ANTsR function: *antsImageRead*

Parameters

- **filename** (str) – Name of the file to read the image from.
- **dimension** (int) – Number of dimensions of the image read. This need not be the same as the dimensions of the image in the file. Allowed values: 2, 3, 4. If not provided, the dimension is obtained from the image file
- **pixeltype** (str) – C++ datatype to be used to represent the pixels read. This datatype need not be the same as the datatype used in the file. Options: unsigned char, unsigned int, float, double. Use pixeltype=None to use the datatype in the file if supported. Unsupported datatypes will be converted to float.
- **reorient** (boolean | string) – if True, the image will be reoriented to RPI if it is 3D if False, nothing will happen if string, this should be the 3-letter orientation to which the

  input image will reoriented if 3D.

  if the image is 2D, this argument is ignored

Return type
ants.core.ANTsImage

image_write(*image*, *filename*, *ri=False*)

Write an ANTsImage to file

ANTsR function: *antsImageWrite*

Parameters

- **image** (ants.core.ANTsImage) – image to save to file
- **filename** (str) – name of file to which image will be saved
- **ri** (bool) –

  if True, return image. This allows for using this function in a pipeline:

  ```
  >>> img2 = img.smooth_image(2.).image_write(file1, ri=True).
  ↪threshold_image(0,20).image_write(file2, ri=True)
  ```

  if False, do not return image

make_image(*shape*, *voxval=0*, *spacing=None*, *origin=None*, *direction=None*, *has_components=False*, *pixeltype='float'*)

Make an image with given size and voxel values or fill a mask with a vector of voxel values.

ANTsR function: *makeImage*

Parameters

- **shape** (tuple/ANTsImage) – input image shape (tuple) or a mask (ANTsImage). If a mask, the output image has the same shape as the mask, and the number of voxel values

in voxval should match the number of positive voxels in the mask. If a tuple, the number of voxel values in voxval should match the product of the dimensions in the tuple.

- **voxval** (scalar) – input image value. Either a scalar (single value) to be placed in all voxels, or a vector of with length matching the number of voxels required by the shape argument.
- **spacing** (tuple/list) – image spatial resolution.
- **origin** (tuple/list) – image spatial origin.
- **direction** (list/ndarray) – direction matrix to convert from index to physical space.
- **components** (bool) – whether there are components per pixel or not.
- **pixeltype** (str) – a supported pixel type for ANTsImage.

Return type
ants.core.ANTsImage

new_image_like(*image*, *data*)

Create a new ANTsImage with the same header information, but with a new image array.

Parameters
data (numpy.ndarray or py::capsule) – New array or pointer for the image. It must have the same shape as the current image data.

Return type
ants.core.ANTsImage

read_image_metadata(*filename*)

Read metadata dictionary from image file. This uses ITK ImageIO to efficiently read the metadata without reading the pixel data.

Parameters
filename (str) – name of image file from which metadata will be read.

Return type
dict

ants.core.ants_metric

ANTs ImageToImageMetric class

Classes

ANTsImageToImageMetric(metric)	ANTsImageToImageMetric class

class ANTsImageToImageMetric(*metric*)

Bases: object

ANTsImageToImageMetric class

property dimension

get_value()

initialize()

property is_vector

property metrictype

property pointer

property precision

set_fixed_image(*image*)

Set Fixed ANTsImage for metric

set_fixed_mask(*image*)

Set Fixed ANTsImage Mask for metric

set_moving_image(*image*)

Set Moving ANTsImage for metric

set_moving_mask(*image*)

Set Fixed ANTsImage Mask for metric

set_sampling(*strategy='regular'*, *percentage=1.0*)

ants.core.ants_metric_io

Functions

create_ants_metric(fixed, moving[, ...])	
new_ants_metric([dimension, precision, ...])	
supported_metrics()	

create_ants_metric(*fixed*, *moving*, *metric_type='MeanSquares'*, *fixed_mask=None*, *moving_mask=None*, *sampling_strategy='regular'*, *sampling_percentage=1*)

Parameters

metric_type (str) – which metric to use options:

MeanSquares MattesMutualInformation ANTSNeighborhoodCorrelation Correlation Demons JointHistogramMutualInformation

Example

```
>>> import ants
>>> fixed = ants.image_read(ants.get_ants_data('r16'))
>>> moving = ants.image_read(ants.get_ants_data('r64'))
>>> metric_type = 'Correlation'
>>> metric = ants.create_ants_metric(fixed, moving, metric_type)
```

new_ants_metric(*dimension=3*, *precision='float'*, *metric_type='MeanSquares'*)

supported_metrics()

ants.core.ants_transform

Functions

apply_ants_transform(transform, data[, ...])	Apply ANTsTransform to data
apply_ants_transform_to_image(transform, ...)	Apply transform to an image
apply_ants_transform_to_point(transform, point)	Apply transform to a point
apply_ants_transform_to_vector(transform, vector)	Apply transform to a vector
compose_ants_transforms(transform_list)	Compose multiple ANTsTransform's together
get_ants_transform_fixed_parameters(transform)	Get fixed parameters of an ANTsTransform
get_ants_transform_parameters(transform)	Get parameters of an ANTsTransform
invert_ants_transform(transform)	Invert ANTsTransform
set_ants_transform_fixed_parameters(...)	Set fixed parameters of an ANTsTransform
set_ants_transform_parameters(transform, ...)	Set parameters of an ANTsTransform
transform_index_to_physical_point(image, index)	Get spatial point from index of an image.
transform_physical_point_to_index(image, point)	Get index from spatial point of an image.

Classes

ANTsTransform([precision, dimension, ...])	

class ANTsTransform(*precision='float'*, *dimension=3*, *transform_type='AffineTransform'*, *pointer=None*)

Bases: `object`

apply(*data*, *data_type='point'*, *reference=None*, ***kwargs*)

Apply transform to data

apply_to_image(*image*, *reference=None*, *interpolation='linear'*)

Apply transform to an image

Parameters

- **image** (`ants.core.ANTsImage`) – image to which the transform will be applied
- **reference** (`ants.core.ANTsImage`) – target space for transforming image
- **interpolation** (`str`) –

  type of interpolation to use. Options are:
  linear nearestneighbor multilabel gaussian bspline cosinewindowedsinc welchwindowedsinc hammingwindoweddinc lanczoswindowedsinc genericlabel

Returns
list

Return type
`transformed vector`

apply_to_point(*point*)

Apply transform to a point

Parameters
point (`list/tuple`) – point to which the transform will be applied

Returns
list

Return type
transformed point

Example

```
>>> import ants
>>> tx = ants.new_ants_transform()
>>> params = tx.parameters
>>> tx.set_parameters(params*2)
>>> pt2 = tx.apply_to_point((1,2,3)) # should be (2,4,6)
```

apply_to_vector(*vector*)

Apply transform to a vector

Parameters
vector (list/tuple) – vector to which the transform will be applied

Returns
list

Return type
transformed vector

property fixed_parameters

Get parameters of transform

invert()

Invert the transform

property parameters

Get parameters of transform

set_fixed_parameters(*parameters*)

Set parameters of transform

set_parameters(*parameters*)

Set parameters of transform

apply_ants_transform(*transform*, *data*, *data_type='point'*, *reference=None*, ***kwargs*)

Apply ANTsTransform to data

ANTsR function: *applyAntsrTransform*

Parameters

- **transform** (*ANTsTransform*) – transform to apply to image
- **data** (ndarray/list/tuple) – data to which transform will be applied
- **data_type** (str) – type of data Options :

  'point' 'vector' 'image'
- **reference** (ants.core.ANTsImage) – target space for transforming image
- **kwargs** (kwargs) – additional options passed to *apply_ants_transform_to_image*

Returns

- ANTsImage if data_type == 'image'

- OR
- tuple if data_type == 'point' or data_type == 'vector'

apply_ants_transform_to_image(*transform*, *image*, *reference*, *interpolation='linear'*)

Apply transform to an image

ANTsR function: *applyAntsrTransformToImage*

Parameters

- **image** (ants.core.ANTsImage) – image to which the transform will be applied
- **reference** (ants.core.ANTsImage) – reference image
- **interpolation** (str) – type of interpolation to use. Options are: linear nearestneighbor multilabel gaussian bspline cosinewindowedsinc welchwindowedsinc hammingwindoweddinc lanczoswindowedsinc genericlabel

Returns

list

Return type

transformed vector

Example

```
>>> import ants
>>> img = ants.image_read(ants.get_ants_data("r16")).clone('float')
>>> tx = ants.new_ants_transform(dimension=2)
>>> tx.set_parameters((0.9,0,0,1.1,10,11))
>>> img2 = tx.apply_to_image(img, img)
```

apply_ants_transform_to_point(*transform*, *point*)

Apply transform to a point

ANTsR function: *applyAntsrTransformToPoint*

Parameters

point (list/tuple) – point to which the transform will be applied

Returns

tuple

Return type

transformed point

Example

```
>>> import ants
>>> tx = ants.new_ants_transform()
>>> params = tx.parameters
>>> tx.set_parameters(params*2)
>>> pt2 = tx.apply_to_point((1,2,3)) # should be (2,4,6)
```

apply_ants_transform_to_vector(*transform*, *vector*)

Apply transform to a vector

ANTsR function: *applyAntsrTransformToVector*

Parameters

vector (list/tuple) – vector to which the transform will be applied

Returns
tuple

Return type
transformed vector

compose_ants_transforms(*transform_list*)

Compose multiple ANTsTransform's together

ANTsR function: *composeAntsrTransforms*

Parameters
transform_list (list/tuple of ANTsTransform object) – list of transforms to compose together

Returns
one transform that contains all given transforms

Return type
ANTsTransform

Example

```
>>> import ants
>>> img = ants.image_read(ants.get_ants_data("r16")).clone('float')
>>> tx = ants.new_ants_transform(dimension=2)
>>> tx.set_parameters((0.9,0,0,1.1,10,11))
>>> inv_tx = tx.invert()
>>> single_tx = ants.compose_ants_transforms([tx, inv_tx])
>>> img_orig = single_tx.apply_to_image(img, img)
>>> rRotGenerator = ants.contrib.RandomRotate2D( ( 0, 40 ), reference=img )
>>> rShearGenerator=ants.contrib.RandomShear2D( (0,50), reference=img )
>>> tx1 = rRotGenerator.transform()
>>> tx2 = rShearGenerator.transform()
>>> rSrR = ants.compose_ants_transforms([tx1, tx2])
>>> rSrR.apply_to_image( img )
```

get_ants_transform_fixed_parameters(*transform*)

Get fixed parameters of an ANTsTransform

ANTsR function: *getAntsrTransformFixedParameters*

get_ants_transform_parameters(*transform*)

Get parameters of an ANTsTransform

ANTsR function: *getAntsrTransformParameters*

invert_ants_transform(*transform*)

Invert ANTsTransform

ANTsR function: *invertAntsrTransform*

Example

```
>>> import ants
>>> img = ants.image_read(ants.get_ants_data("r16")).clone('float')
>>> tx = ants.new_ants_transform(dimension=2)
>>> tx.set_parameters((0.9,0,0,1.1,10,11))
```

(continues on next page)

(continued from previous page)

```
>>> img_transformed = tx.apply_to_image(img, img)
>>> inv_tx = tx.invert()
>>> img_orig = inv_tx.apply_to_image(img_transformed, img_transformed)
```

set_ants_transform_fixed_parameters(*transform*, *parameters*)

Set fixed parameters of an ANTsTransform

ANTsR function: *setAntsrTransformFixedParameters*

set_ants_transform_parameters(*transform*, *parameters*)

Set parameters of an ANTsTransform

ANTsR function: *setAntsrTransformParameters*

transform_index_to_physical_point(*image*, *index*)

Get spatial point from index of an image.

ANTsR function: *antsTransformIndexToPhysicalPoint*

Parameters

- **img** (ants.core.ANTsImage) – image to get values from
- **index** (list or tuple or numpy.ndarray) – location in image

Return type

tuple

Example

```
>>> import ants
>>> import numpy as np
>>> img = ants.make_image((10,10),np.random.randn(100))
>>> pt = ants.transform_index_to_physical_point(img, (2,2))
```

transform_physical_point_to_index(*image*, *point*)

Get index from spatial point of an image.

ANTsR function: *antsTransformPhysicalPointToIndex*

Parameters

- **image** (ants.core.ANTsImage) – image to get values from
- **point** (list or tuple or numpy.ndarray) – point in image

Return type

tuple

Example

```
>>> import ants
>>> import numpy as np
>>> img = ants.make_image((10,10),np.random.randn(100))
>>> idx = ants.transform_physical_point_to_index(img, (2,2))
>>> img.set_spacing((2,2))
>>> idx2 = ants.transform_physical_point_to_index(img, (4,4))
```

ants.core.ants_transform_io

Functions

create_ants_transform([transform_type, ...])	Create and initialize an ANTsTransform
fsl2antstransform(matrix, reference, moving)	Convert an FSL linear transform to an antsrTransform
new_ants_transform([precision, dimension, ...])	Create a new ANTsTransform
read_transform(filename[, precision])	Read a transform from file
transform_from_displacement_field(field)	Convert deformation field (multiChannel image) to ANTsTransform
transform_to_displacement_field(xfrm, ref)	Convert displacement field ANTsTransform to displacement field
write_transform(transform, filename)	Write ANTsTransform to file

create_ants_transform(*transform_type='AffineTransform'*, *precision='float'*, *dimension=3*, *matrix=None*, *offset=None*, *center=None*, *translation=None*, *parameters=None*, *fixed_parameters=None*, *displacement_field=None*, *supported_types=False*)

Create and initialize an ANTsTransform

ANTsR function: *createAntsrTransform*

Parameters

- **transform_type** (str) – type of transform(s)
- **precision** (str) – numerical precision
- **dimension** (int) – spatial dimension of transform
- **matrix** (numpy.ndarray) – matrix for linear transforms
- **offset** (tuple/list) – offset for linear transforms
- **center** (tuple/list) – center for linear transforms
- **translation** (tuple/list) – translation for linear transforms
- **parameters** (ndarray/list) – array of parameters
- **fixed_parameters** (ndarray/list) – array of fixed parameters
- **displacement_field** (ants.core.ANTsImage) – multichannel ANTsImage for non-linear transform
- **supported_types** (bool) – flag that returns array of possible transforms types

Return type

ANTsTransform or list of ANTsTransform types

Example

```
>>> import ants
>>> translation = (3,4,5)
>>> tx = ants.create_ants_transform( type='Euler3DTransform',␣
↪translation=translation )
```

fsl2antstransform(*matrix*, *reference*, *moving*)

Convert an FSL linear transform to an antsrTransform

ANTsR function: *fsl2antsrtransform*

Parameters

- **matrix** (ndarray/list) – 4x4 matrix of transform parameters
- **reference** (ants.core.ANTsImage) – target image
- **moving** (ants.core.ANTsImage) – moving image

Return type
ANTsTransform

Examples

```
>>> import ants
>>> import numpy as np
>>> fslmat = np.zeros((4,4))
>>> np.fill_diagonal(fslmat, 1)
>>> img = ants.image_read(ants.get_ants_data('ch2'))
>>> tx = ants.fsl2antstransform(fslmat, img, img)
```

new_ants_transform(*precision='float'*, *dimension=3*, *transform_type='AffineTransform'*, *parameters=None*, *fixed_parameters=None*)

Create a new ANTsTransform

ANTsR function: None

This is a simplified method for creating an ANTsTransform, mostly used internally. See create_ants_transform for more options.

Example

```
>>> import ants
>>> tx = ants.new_ants_transform()
```

read_transform(*filename*, *precision='float'*)

Read a transform from file

ANTsR function: *readAntsrTransform*

Parameters

- **filename** (str) – filename of transform
- **precision** (str) – numerical precision of transform

Return type
ANTsTransform

Example

```
>>> import ants
>>> tx = ants.new_ants_transform(dimension=2)
>>> tx.set_parameters((0.9,0,0,1.1,10,11))
>>> ants.write_transform(tx, '~/desktop/tx.mat')
>>> tx2 = ants.read_transform('~/desktop/tx.mat')
```

transform_from_displacement_field(*field*)

Convert deformation field (multiChannel image) to ANTsTransform

ANTsR function: *antsrTransformFromDisplacementField*

Parameters
field (ants.core.ANTsImage) – deformation field as multi-channel ANTsImage

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> fi = ants.image_read(ants.get_ants_data('r16') )
>>> mi = ants.image_read(ants.get_ants_data('r64') )
>>> fi = ants.resample_image(fi,(60,60),1,0)
>>> mi = ants.resample_image(mi,(60,60),1,0) # speed up
>>> mytx = ants.registration(fixed=fi, moving=mi, type_of_transform = ('SyN') )
>>> vec = ants.image_read( mytx['fwdtransforms'][0] )
>>> atx = ants.transform_from_displacement_field( vec )
```

transform_to_displacement_field(*xfrm*, *ref*)

Convert displacement field ANTsTransform to displacement field

ANTsR function: *antsrTransformToDisplacementField*

Parameters

- **xfrm** (displacement field ANTsTransform) – displacement field ANTsTransform
- **ref** (ANTs Image) –

Return type
ANTsVectorImage

Example

```
>>> import ants
>>> fi = ants.image_read(ants.get_ants_data('r16') )
>>> mi = ants.image_read(ants.get_ants_data('r64') )
>>> fi = ants.resample_image(fi,(60,60),1,0)
>>> mi = ants.resample_image(mi,(60,60),1,0) # speed up
>>> mytx = ants.registration(fixed=fi, moving=mi, type_of_transform = ('SyN') )
>>> vec = ants.image_read( mytx['fwdtransforms'][0] )
>>> atx = ants.transform_from_displacement_field( vec )
>>> field = ants.transform_to_displacement_field( atx, fi )
```

write_transform(*transform*, *filename*)

Write ANTsTransform to file

ANTsR function: *writeAntsrTransform*

Parameters

- **transform** (ANTsTransform) – transform to save
- **filename** (str) – filename of transform (file extension is ".mat" for affine transforms)

Return type
N/A

Example

```
>>> import ants
>>> tx = ants.new_ants_transform(dimension=2)
>>> tx.set_parameters((0.9,0,0,1.1,10,11))
>>> ants.write_transform(tx, '~/desktop/tx.mat')
>>> tx2 = ants.read_transform('~/desktop/tx.mat')
```

8.1.4 ants.decorators

Functions

image_method(func)	

image_method(*func*)

8.1.5 ants.deeplearn

Modules

cropping_and_padding_utilities	
data_augmentation(input_image_list[, ...])	Randomly transform image data.
extract_image_patches(image, patch_size[, ...])	Extract 2-D or 3-D image patches.
histogram_warp_image_intensities(image[, ...])	Transform image intensities based on histogram mapping.
one_hot_segmentation	
randomly_transform_image_data(...[, ...])	Randomly transform image data (optional: with corresponding segmentations).
reconstruct_image_from_patches(patches, ...)	Reconstruct image from a list of patches.
regression_match_image(source_image, ...[, ...])	Image intensity normalization using linear regression.
simulate_bias_field(domain_image[, ...])	Simulate random bias field

ants.deeplearn.cropping_and_padding_utilities

Functions

crop_image_center(image, crop_size)	Crop the center of an image.
crop_image_from_center_point(image, ...)	Crop a patch from an image around a center point
pad_image_by_factor(image, factor)	Pad an image based on a factor.
pad_or_crop_image_to_size(image, size)	Pad or crop an image to a specified size

crop_image_center(*image*, *crop_size*)

Crop the center of an image.

Parameters

- **image** (ants.core.ANTsImage) – Input image
- **crop_size** (typing.Tuple) – Width, height, depth (if 3-D), and time (if 4-D) of crop region.

Return type
ANTs image.

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> cropped_image = crop_image_center(image, crop_size=(64, 64))
```

crop_image_from_center_point(*image*, *center_point*, *patch_size*)

Crop a patch from an image around a center point

image: ANTsImage
Input image

center_point: tuple
Physical space coordinates of center point.

patch_size: tuple
List defining patch size.

pad_image_by_factor(*image*, *factor*)

Pad an image based on a factor.

Pad image of size (x, y, z) to (x', y', z') where (x', y', z') is a divisible by a user-specified factor.

Parameters

- **image** (ants.core.ANTsImage) – Input image
- **factor** (scalar or typing.Tuple) – Padding factor

Return type
ANTs image.

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> padded_image = pad_image_by_factor(image, factor=4)
```

pad_or_crop_image_to_size(*image*, *size*)

Pad or crop an image to a specified size

Parameters

- **image** (ants.core.ANTsImage) – Input image
- **size** (tuple) – size of output image

Return type
A cropped or padded image

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> padded_image = pad_or_crop_image_to_size(image, (333, 333))
```

ants.deeplearn.data_augmentation

data_augmentation(*input_image_list*, *segmentation_image_list=None*, *pointset_list=None*, *number_of_simulations=10*, *reference_image=None*, *transform_type='affineAndDeformation'*, *noise_model='additivegaussian'*, *noise_parameters=(0.0, 0.05)*, *sd_simulated_bias_field=1.0*, *sd_histogram_warping=0.05*, *sd_affine=0.05*, *sd_deformation=0.2*, *output_numpy_file_prefix=None*, *verbose=False*)

Randomly transform image data.

Given an input image list (possibly multi-modal) and an optional corresponding segmentation image list, this function will perform data augmentation with the following augmentation possibilities:

- spatial transformations
- added image noise
- simulated bias field
- histogram warping

Parameters

- **input_image_list** (list of lists of ANTsImages) – List of lists of input images to warp. The internal list sets contain one or more images (per subject) which are assumed to be mutually aligned. The outer list contains multiple subject lists which are randomly sampled to produce output image list.
- **segmentation_image_list** (list of ANTsImages) – List of segmentation images corresponding to the input image list (optional).
- **pointset_list** (list of pointsets) – Numpy arrays corresponding to the input image list (optional). If using this option, the transform_type must be invertible.
- **number_of_simulations** (int) – Number of simulated output image sets. Default = 10.
- **reference_image** (ants.core.ANTsImage) – Defines the spatial domain for all output images. If one is not specified, we used the first image in the input image list.
- **transform_type** (str) – One of the following options: "translation", "rigid", "scaleShear", "affine", "deformation", "affineAndDeformation".
- **noise_model** (str) – 'additivegaussian', 'saltandpepper', 'shot', and 'speckle'. Alternatively, one can specify a tuple or list of one or more of the options and one is selected at random with reasonable, randomized parameters. Note that the "speckle" model takes much longer than the others.
- **noise_parameters** (tuple or numpy.ndarray or float) – 'additivegaussian': (mean, standardDeviation) 'saltandpepper': (probability, saltValue, pepperValue) 'shot': scale 'speckle': standardDeviation Note that the standard deviation, scale, and probability values are *max* values and are randomly selected in the range [0, noise_parameter]. Also, the "mean", "saltValue" and "pepperValue" are assumed to be in the intensity normalized range of [0, 1].
- **sd_simulated_bias_field** (float) – Characterize the standard deviation of the amplitude.
- **sd_histogram_warping** (float) – Determines the strength of the bias field.
- **sd_affine** (float) – Determines the amount of affine transformation.
- **sd_deformation** (float) – Determines the amount of deformable transformation.
- **output_numpy_file_prefix** (str) – Filename of output numpy array containing all the simulated images and segmentations.

Return type
list of lists of transformed images and/or outputs to a numpy array.

Example

```
>>> image1_list = list()
>>> image1_list.append(ants.image_read(ants.get_ants_data("r16")))
>>> image2_list = list()
>>> image2_list.append(ants.image_read(ants.get_ants_data("r64")))
>>> segmentation1 = ants.threshold_image(image1_list[0], "Otsu", 3)
>>> segmentation2 = ants.threshold_image(image2_list[0], "Otsu", 3)
>>> input_segmentations = list()
>>> input_segmentations.append(segmentation1)
>>> input_segmentations.append(segmentation2)
>>> points1 = ants.get_centroids(segmentation1)[:,0:2]
>>> points2 = ants.get_centroids(segmentation2)[:,0:2]
>>> input_points = list()
>>> input_points.append(points1)
>>> input_points.append(points2)
>>> input_images = list()
>>> input_images.append(image1_list)
>>> input_images.append(image2_list)
>>> data = data_augmentation(input_images,
                             input_segmentations,
                             input_points,
                             tranform_type="scaleShear")
```

ants.deeplearn.extract_image_patches

extract_image_patches(*image*, *patch_size*, *max_number_of_patches='all'*, *stride_length=1*, *mask_image=None*, *random_seed=None*, *return_as_array=False*, *randomize=True*)

Extract 2-D or 3-D image patches.

Parameters

- **image** (ants.core.ANTsImage) – Input image with one or more components.
- **patch_size** (n-D tuple (depending on dimensionality).) – Width, height, and depth (if 3-D) of patches.
- **max_number_of_patches** (int or str) – Maximum number of patches returned. If "all" is specified, then all patches in sequence (defined by the stride_length are extracted.
- **stride_length** (int or typing.Tuple) – Defines the sequential patch overlap for max_number_of_patches = "all". Can be a image-dimensional vector or a scalar.
- **mask_image** (ANTsImage (optional)) – Optional image specifying the sampling region for the patches when max_number_of_patches does not equal "all". The way we constrain patch selection using a mask is by forcing each returned patch to have a masked voxel at its center.
- **random_seed** (integer (optional)) – Seed value that allows reproducible patch extraction across runs.
- **return_as_array** (bool) – Specifies the return type of the function. If False (default) the return type is a list where each element is a single patch. Otherwise the return type is an array of size dim(number_of_patches, patch_size).
- **randomize** (bool) – Boolean controlling whether we randomize indices when masking.

Return type
A list (or array) of patches.

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> image_patches = extract_image_patches(image, patch_size=(32, 32))
```

ants.deeplearn.histogram_warp_image_intensities

histogram_warp_image_intensities(*image*, *break_points=(0.25, 0.5, 0.75)*, *displacements=None*, *clamp_end_points=(False, False)*, *sd_displacements=0.05*, *transform_domain_size=20*)

Transform image intensities based on histogram mapping.

Apply B-spline 1-D maps to an input image for intensity warping.

Parameters

- **image** (ants.core.ANTsImage) – Input image.
- **break_points** (int or tuple) – Parametric points at which the intensity transform displacements are specified between [0, 1]. Alternatively, a single number can be given and the sequence is linearly spaced in [0, 1].
- **displacements** (tuple) – displacements to define intensity warping. Length must be equal to the breakPoints. Alternatively, if None random displacements are chosen (random normal: mean = 0, sd = sd_displacements).
- **sd_displacements** (float) – Characterize the randomness of the intensity displacement.
- **clamp_end_points** (2-element tuple of booleans) – Specify non-zero intensity change at the ends of the histogram.
- **transform_domain_size** (int) – Defines the sampling resolution of the B-spline warping.

Return type
ANTs image

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data("r64"))
>>> transformed_image = histogram_warp_image_intensities( image )
```

ants.deeplearn.one_hot_segmentation

Functions

one_hot_to_segmentation(one_hot_array, ...)	Inverse of basic one-hot transformation of segmentations array
segmentation_to_one_hot(segmentations_array)	Basic one-hot transformation of segmentations array

ants.deeplearn.randomly_transform_image_data

randomly_transform_image_data(*reference_image*, *input_image_list*, *segmentation_image_list=None*, *number_of_simulations=10*, *transform_type='affine'*, *sd_affine=0.02*, *deformation_transform_type='bspline'*, *number_of_random_points=1000*, *sd_noise=10.0*, *number_of_fitting_levels=4*, *mesh_size=1*, *sd_smoothing=4.0*, *input_image_interpolator='linear'*, *segmentation_image_interpolator='nearestNeighbor'*)

Randomly transform image data (optional: with corresponding segmentations).

Apply rigid, affine and/or deformable maps to an input set of training images. The reference image domain defines the space in which this happens.

Parameters

- **reference_image** (ants.core.ANTsImage) – Defines the spatial domain for all output images. If the input images do not match the spatial domain of the reference image, we internally resample the target to the reference image. This could have unexpected consequences. Resampling to the reference domain is performed by testing using ants.image_physical_space_consistency then calling ants.resample_image_to_target with failure.
- **input_image_list** (list of lists of ANTsImages) – List of lists of input images to warp. The internal list sets contain one or more images (per subject) which are assumed to be mutually aligned. The outer list contains multiple subject lists which are randomly sampled to produce output image list.
- **segmentation_image_list** (list of ANTsImages) – List of segmentation images corresponding to the input image list (optional).
- **number_of_simulations** (int) – Number of simulated output image sets.
- **transform_type** (str) – One of the following options: "translation", "rotation", "rigid", "scaleShear", "affine", "deformation", "affineAndDeformation".
- **sd_affine** (float) – Parameter dictating deviation amount from identity for random linear transformations.
- **deformation_transform_type** (str) – "bspline" or "exponential".
- **number_of_random_points** (int) – Number of displacement points for the deformation field.
- **sd_noise** (float) – Standard deviation of the displacement field.
- **number_of_fitting_levels** (int) – Number of fitting levels (bspline deformation only).
- **mesh_size** (int or typing.Tuple) – Determines fitting resolution (bspline deformation only).
- **sd_smoothing** (float) – Standard deviation of the Gaussian smoothing in mm (exponential field only).
- **input_image_interpolator** (str) – One of the following options: "nearestNeighbor", "linear", "gaussian", "bSpline".
- **segmentation_image_interpolator** (str) – Only "nearestNeighbor" is currently available.

Return type

list of lists of transformed images

Example

```
>>> import ants
>>> image1_list = list()
>>> image1_list.append(ants.image_read(ants.get_ants_data("r16")))
>>> image2_list = list()
>>> image2_list.append(ants.image_read(ants.get_ants_data("r64")))
>>> input_segmentations = list()
>>> input_segmentations.append(ants.threshold_image(image1, "Otsu", 3))
>>> input_segmentations.append(ants.threshold_image(image2, "Otsu", 3))
>>> input_images = list()
>>> input_images.append(image1_list)
>>> input_images.append(image2_list)
>>> data = antspynet.randomly_transform_image_data(image1,
>>>     input_images, input_segmentations, sd_affine=0.02,
>>>     transform_type = "affineAndDeformation" )
```

ants.deeplearn.reconstruct_image_from_patches

reconstruct_image_from_patches(*patches*, *domain_image*, *stride_length=1*, *domain_image_is_mask=False*)

Reconstruct image from a list of patches.

Parameters

- **patches** (list or numpy.ndarray of patches) – List or array of patches defining an image. Patches are assumed to have the same format as returned by extract_image_patches.
- **domain_image** (ANTs image) – Image or mask to define the geometric information of the reconstructed image. If this is a mask image, the reconstruction will only use patches in the mask.
- **stride_length** (int or typing.Tuple) – Defines the sequential patch overlap for max_number_of_patches = "all". Can be a image-dimensional vector or a scalar.
- **domain_image_is_mask** (bool) – Boolean specifying whether the domain image is a mask used to limit the region of reconstruction from the patches.

Return type

An ANTs image.

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> image_patches = extract_image_patches(image, patch_size=(16, 16), stride_
↪length=4)
>>> reconstructed_image = reconstruct_image_from_patches(image_patches, image,␣
↪stride_length=4)
```

ants.deeplearn.regression_match_image

regression_match_image(*source_image*, *reference_image*, *mask=None*, *poly_order=1*, *truncate=True*)

Image intensity normalization using linear regression.

Parameters

- **source_image** (ants.core.ANTsImage) – Image whose intensities are matched to the reference image.

- **reference_image** (ants.core.ANTsImage) – Defines the reference intensity function.
- **poly_order** (int) – Polynomial order of fit. Default is 1 (linear fit).
- **mask** (ants.core.ANTsImage) – Defines voxels for regression modeling.
- **truncate** (bool) – Turns on/off the clipping of intensities.

Return type
ANTs image (i.e., source_image) matched to the (reference_image).

Example

```
>>> import ants
>>> source_image = ants.image_read(ants.get_ants_data('r16'))
>>> reference_image = ants.image_read(ants.get_ants_data('r64'))
>>> matched_image = regression_match_image(source_image, reference_image)
```

ants.deeplearn.simulate_bias_field

simulate_bias_field(*domain_image*, *number_of_points=10*, *sd_bias_field=1.0*, *number_of_fitting_levels=4*, *mesh_size=1*)

Simulate random bias field

Low frequency, spatial varying simulated random bias field using random points and B-spline fitting.

Parameters

- **domain_image** (ants.core.ANTsImage) – Image to define the spatial domain of the bias field.
- **number_of_points** (int) – Number of randomly defined points to define the bias field (default = 10).
- **sd_bias_field** (float) – Characterize the standard deviation of the amplitude (default = 1).
- **number_of_fitting_levels** (int) – B-spline fitting parameter.

Return type
ANTs image

Example

```
>>> import ants
>>> import numpy as np
>>> image = ants.image_read(ants.get_ants_data("r64"))
>>> log_field = ants.simulate_bias_field(image, number_of_points=10, sd_bias_
↪field=1.0,
...     number_of_fitting_levels=2, mesh_size=10)
>>> log_field = log_field.iMath("Normalize")
>>> field_array = np.power(np.exp(log_field.numpy()), 4)
>>> image = image * ants.from_numpy(field_array, origin=image.origin,
...     spacing=image.spacing, direction=image.direction)
```

8.1.6 ants.internal

Functions

get_lib_fn(string)	
get_pointer_string(image)	
infer_dtype(dtype)	
process_arguments(args)	
short_ptype(pixeltype)	

get_lib_fn(*string*)

get_pointer_string(*image*)

infer_dtype(*dtype*)

process_arguments(*args*)

short_ptype(*pixeltype*)

8.1.7 ants.label

Modules

image_to_cluster_images(image[, ...])	Converts an image to several independent images.
label_clusters(image[, min_cluster_size, ...])	This will give a unique ID to each connected component 1 through N of size > min_cluster_size
label_geometry_measures(label_image[, ...])	Wrapper for the ANTs funtion LabelGeometryMeasures
label_image_centroids(image[, physical, ...])	Converts a label image to coordinates summarizing their positions
label_overlap_measures(source_image, ...)	Get overlap measures from two label images (e.g., Dice)
label_stats(image, label_image)	Get label statistics from an image.
labels_to_matrix(image, mask[, ...])	Convert a labeled image to an n x m binary matrix where n = number of voxels and m = number of labels.
make_points_image(pts, target[, radius])	Create label image from physical space points
multi_label_morphology(image, operation, radius)	Morphology on multi label images.

ants.label.image_to_cluster_images

image_to_cluster_images(*image*, *min_cluster_size=50*, *min_thresh=1e-06*, *max_thresh=1*)

Converts an image to several independent images.

Produces a unique image for each connected component 1 through N of size > min_cluster_size

ANTsR function: *image2ClusterImages*

Parameters

- **image** (ants.core.ANTsImage) – input image
- **min_cluster_size** (int) – throw away clusters smaller than this value

- **min_thresh** (scalar) – threshold to a statistical map
- **max_thresh** (scalar) – threshold to a statistical map

Return type
list of ANTsImage types

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> image = ants.threshold_image(image, 1, 1e15)
>>> image_cluster_list = ants.image_to_cluster_images(image)
```

ants.label.label_clusters

label_clusters(*image*, *min_cluster_size=50*, *min_thresh=1e-06*, *max_thresh=1*, *fully_connected=False*)

This will give a unique ID to each connected component 1 through N of size > min_cluster_size

ANTsR function: *labelClusters*

Parameters

- **image** (ants.core.ANTsImage) – input image e.g. a statistical map
- **min_cluster_size** (int) – throw away clusters smaller than this value
- **min_thresh** (scalar) – threshold to a statistical map
- **max_thresh** (scalar) – threshold to a statistical map
- **fully_connected** (bool) – boolean sets neighborhood connectivity pattern

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read( ants.get_ants_data('r16') )
>>> timageFully = ants.label_clusters( image, 10, 128, 150, True )
>>> timageFace = ants.label_clusters( image, 10, 128, 150, False )
```

ants.label.label_geometry_measures

label_geometry_measures(*label_image*, *intensity_image=None*)

Wrapper for the ANTs funtion LabelGeometryMeasures

ANTsR function: *labelGeometryMeasures*

Parameters

- **label_image** (ants.core.ANTsImage) – image on which to compute geometry. Labels must be representable as uint32.
- **intensity_image** (ANTsImage (optional)) – image with intensity values

Return type
pandas.DataFrame

Example

```
>>> import ants
>>> fi = ants.image_read( ants.get_ants_data('r16') )
>>> seg = ants.kmeans_segmentation( fi, 3 )['segmentation']
>>> geom = ants.label_geometry_measures(seg,fi)
```

ants.label.label_image_centroids

label_image_centroids(*image*, *physical=False*, *convex=True*, *verbose=False*)

Converts a label image to coordinates summarizing their positions

ANTsR function: *labelImageCentroids*

Parameters

- **image** (ants.core.ANTsImage) – image of integer labels
- **physical** (bool) – whether you want physical space coordinates or not
- **convex** (bool) – if True, return centroid if False return point with min average distance to other points with same label

Returns

labels
[1D-ndarray] array of label values

vertices
[pd.DataFrame] coordinates of label centroids

Return type
dictionary w/ following key-value pairs

Example

```
>>> import ants
>>> import numpy as np
>>> image = ants.from_numpy(np.asarray([[[0,2],[1,3]],[[4,6],[5,7]]]).astype(
↪'float32'))
>>> labels = ants.label_image_centroids(image)
```

ants.label.label_overlap_measures

label_overlap_measures(*source_image*, *target_image*)

Get overlap measures from two label images (e.g., Dice)

ANTsR function: *labelOverlapMeasures*

Parameters

- **image** (*source*) – Source image
- **target_image** (ants.core.ANTsImage) – Target image

Return type
data frame with measures for each label and all labels combined

Example

```
>>> import ants
>>> r16 = ants.image_read( ants.get_ants_data('r16') )
>>> r64 = ants.image_read( ants.get_ants_data('r64') )
>>> s16 = ants.kmeans_segmentation( r16, 3 )['segmentation']
>>> s64 = ants.kmeans_segmentation( r64, 3 )['segmentation']
>>> stats = ants.label_overlap_measures(s16, s64)
```

ants.label.label_stats

label_stats(*image*, *label_image*)

Get label statistics from an image. The labels must be representable as uint32.

ANTsR function: *labelStats*

Parameters

- **image** (ants.core.ANTsImage) – Image from which statistics will be calculated
- **label_image** (ants.core.ANTsImage) – Label image

Return type

pandas.DataFrame

Example

```
>>> import ants
>>> image = ants.image_read( ants.get_ants_data('r16') , 2 )
>>> image = ants.resample_image( image, (64,64), 1, 0 )
>>> mask = ants.get_mask(image)
>>> segs1 = ants.kmeans_segmentation( image, 3 )
>>> stats = ants.label_stats(image, segs1['segmentation'])
```

ants.label.labels_to_matrix

labels_to_matrix(*image*, *mask*, *target_labels=None*, *missing_val=nan*)

Convert a labeled image to an n x m binary matrix where n = number of voxels and m = number of labels. Only includes values inside the provided mask while including background (image == 0) for consistency with timeseries2matrix and other image to matrix operations.

ANTsR function: *labels2matrix*

Parameters

- **image** (ants.core.ANTsImage) – input label image
- **mask** (ants.core.ANTsImage) – defines domain of interest
- **target_labels** (list/tuple) – defines target regions to be returned. if the target label does not exist in the input label image, then the matrix will contain a constant value of missing_val (default None) in that row.
- **missing_val** (scalar) – value to use for missing label values

Return type

numpy.ndarray

Example

```
>>> import ants
>>> fi = ants.image_read(ants.get_ants_data('r16')).resample_image((60,60),1,0)
>>> mask = ants.get_mask(fi)
>>> labs = ants.kmeans_segmentation(fi,3)['segmentation']
>>> labmat = ants.labels_to_matrix(labs, mask)
```

ants.label.make_points_image

make_points_image(*pts*, *target*, *radius=5*)

Create label image from physical space points

Creates spherical points in the coordinate space of the target image based on the n-dimensional matrix of points that the user supplies. The image defines the dimensionality of the data so if the input image is 3D then the input points should be 2D or 3D.

ANTsR function: *makePointsImage*

Parameters

- **pts** (numpy.ndarray) – input points
- **target** (ants.core.ANTsImage) – Image defining target space
- **radius** (int) – radius for the points

Return type

ants.core.ANTsImage

Example

```
>>> import ants
>>> import pandas as pd
>>> mni = ants.image_read(ants.get_data('mni'))
>>> powers_pts = pd.read_csv(ants.get_data('powers_mni_itk'))
>>> powers_labels = ants.make_points_image(powers_pts.iloc[:,:3].values, mni,␣
↪radius=3)
```

ants.label.multi_label_morphology

multi_label_morphology(*image*, *operation*, *radius*, *dilation_mask=None*, *label_list=None*, *force=False*)

Morphology on multi label images.

Wraps calls to iMath binary morphology. Additionally, dilation and closing operations preserve pre-existing labels. The choices of operation are:

Dilation: dilates all labels sequentially, but does not overwrite original labels. This reduces dependence on the intensity ordering of adjoining labels. Ordering dependence can still arise if two or more labels dilate into the same space - in this case, the label with the lowest intensity is retained. With a mask, dilated labels are multiplied by the mask and then added to the original label, thus restricting dilation to the mask region.

Erosion: Erodes labels independently, equivalent to calling iMath iteratively.

Closing: Close holes in each label sequentially, but does not overwrite original labels.

Opening: Opens each label independently, equivalent to calling iMath iteratively.

Parameters

- **image** (ants.core.ANTsImage) – Input image should contain only 0 for background and positive integers for labels.
- **operation** (str) – One of MD, ME, MC, MO, passed to iMath.
- **radius** (int) – radius of the morphological operation.
- **dilation_mask** (ants.core.ANTsImage) – Optional binary mask to constrain dilation only (eg dilate cortical label into WM).
- **label_list** (list or tuple or numpy.ndarray) – Optional list of labels, to perform operation upon. Defaults to all unique intensities in image.

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> img = ants.image_read(ants.get_data('r16'))
>>> labels = ants.get_mask(img,1,150) + ants.get_mask(img,151,225) * 2
>>> labels_dilated = ants.multi_label_morphology(labels, 'MD', 2)
>>> # should see original label regions preserved in dilated version
>>> # label N should have mean N and 0 variance
>>> print(ants.label_stats(labels_dilated, labels))
```

8.1.8 ants.learn

Modules

decomposition

ants.learn.decomposition

Functions

eig_seg(mask, img_list[, ...])	Segment a mask into regions based on the max value in an image list.
initialize_eigenanatomy(initmat[, mask, ...])	InitializeEigenanatomy is a helper function to initialize sparseDecom and sparseDecom2.
sparse_decom2(inmatrix[, inmask, ...])	Decomposes two matrices into paired sparse eigenevectors to maximize canonical correlation - aka Sparse CCA.

eig_seg(*mask*, *img_list*, *apply_segmentation_to_images=False*, *cthresh=0*, *smooth=1*)

Segment a mask into regions based on the max value in an image list. At a given voxel the segmentation label will contain the index to the image that has the largest value. If the 3rd image has the greatest value, the segmentation label will be 3 at that voxel.

Parameters

- **mask** (ants.core.ANTsImage) – D-dimensional mask > 0 defining segmentation region.
- **img_list** (collection of ants.core.ANTsImage or np.ndarray) – images to use

- **apply_segmentation_to_images** (bool) – determines if original image list is modified by the segmentation.
- **cthresh** (int) – throw away isolated clusters smaller than this value
- **smooth** (float) – smooth the input data first by this value

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> mylist = [ants.image_read(ants.get_ants_data('r16')),
              ants.image_read(ants.get_ants_data('r27')),
              ants.image_read(ants.get_ants_data('r85'))]
>>> myseg = ants.eig_seg(ants.get_mask(mylist[0]), mylist)
```

initialize_eigenanatomy(*initmat*, *mask=None*, *initlabels=None*, *nreps=1*, *smoothing=0*)

InitializeEigenanatomy is a helper function to initialize sparseDecom and sparseDecom2. Can be used to estimate sparseness parameters per eigenvector. The user then only chooses nvecs and optional regularization parameters.

Parameters

- **initmat** (np.ndarray or ants.core.ANTsImage) – input matrix where rows provide initial vector values. alternatively, this can be an antsImage which contains labeled regions.
- **mask** (ants.core.ANTsImage) – mask if available
- **initlabels** (list/tuple of integers) – which labels in initmat to use as initial components
- **nreps** (int) – nrepetitions to use
- **smoothing** (float) – if using an initial label image, optionally smooth each roi

Returns

initlist
[list of ANTsImage types] initialization list(s) for sparseDecom(2)

mask
[ANTsImage] mask(s) for sparseDecom(2)

enames
[list of strings] string names of components for sparseDecom(2)

Return type
dict w/ the following key/value pairs

Example

```
>>> import ants
>>> import numpy as np
>>> mat = np.random.randn(4,100).astype('float32')
>>> init = ants.initialize_eigenanatomy(mat)
```

sparse_decom2(*inmatrix*, *inmask=(None, None)*, *sparseness=(0.01, 0.01)*, *nvecs=3*, *its=20*, *cthresh=(0, 0)*, *statdir=None*, *perms=0*, *uselong=0*, *z=0*, *smooth=0*, *robust=0*, *mycoption=0*, *initialization_list=[]*, *initialization_list2=[]*, *ell1=10*, *prior_weight=0*, *verbose=False*, *rejector=0*, *max_based=False*, *version=1*)

Decomposes two matrices into paired sparse eigenevectors to maximize canonical correlation - aka Sparse CCA. Note: we do not scale the matrices internally. We leave scaling choices to the user.

ANTsR function: *sparseDecom2*

Parameters

- **inmatrix** (typing.Tuple of ndarrays) – input as inmatrix=(mat1,mat2). n by p input matrix and n by q input matrix , spatial variable lies along columns.
- **inmask** (typing.Tuple of ANTsImage types (optional - one or both)) – optional pair of image masks
- **sparseness** (tuple) – a pair of float values e.g c(0.01,0.1) enforces an unsigned 99 percent and 90 percent sparse solution for each respective view
- **nvecs** (int) – number of eigenvector pairs
- **its** (int) – number of iterations, 10 or 20 usually sufficient
- **cthresh** (typing.Tuple) – cluster threshold pair
- **statdir** (string (optional)) – temporary directory if you want to look at full output
- **perms** (int) – number of permutations. settings permutations greater than 0 will estimate significance per vector empirically. For small datasets, these may be conservative. p-values depend on how one scales the input matrices.
- **uselong** (bool) – enforce solutions of both views to be the same - requires matrices to be the same size
- **z** (float) – subject space (low-dimensional space) sparseness value
- **smooth** (float) – smooth the data (only available when mask is used)
- **robust** (bool) – rank transform input matrices
- **mycoption** (int) – enforce 1 - spatial orthogonality, 2 - low-dimensional orthogonality or 0 - both
- **initialization_list** (list) – initialization for first view
- **initialization_list2** (list) – initialization for 2nd view
- **ell1** (float) – gradient descent parameter, if negative then l0 otherwise use l1
- **prior_weight** (scalar) – Scalar value weight on prior between 0 (prior is weak) and 1 (prior is strong). Only engaged if initialization is used
- **verbose** (bool) – activates verbose output to screen
- **rejector** (scalar) – rejects small correlation solutions
- **max_based** (bool) – whether to choose max-based thresholding

Returns

projections
[ndarray] X projections

projections2
[ndarray] Y projections

eig1
[ndarray] X components

eig2
[ndarray] Y components

summary
[pd.DataFrame] first column is canonical correlations, second column is p-values (these are *None* if perms > 0)

Return type
`dict w/ following key/value pairs`

Example

```
>>> import numpy as np
>>> import ants
>>> mat = np.random.randn(20, 100)
>>> mat2 = np.random.randn(20, 90)
>>> mydecom = ants.sparse_decom2(inmatrix = (mat,mat2),
                                 sparseness=(0.1,0.3), nvecs=3,
                                 its=3, perms=0)
```

8.1.9 ants.lib

8.1.10 ants.math

Modules

averaging	
get_centroids(image[, clustparam])	Reduces a variate/statistical/network image to a set of centroids describing the center of each stand-alone non-zero component in the image
get_neighborhood	
hausdorff_distance(image1, image2)	Get Hausdorff distance between non-zero pixels in two images
image_similarity(fixed_image, moving_image)	Measure similarity between two images.
metrics	
quantile(image, q[, nonzero])	Get the quantile values from an ANTsImage

ants.math.averaging

Functions

average_images(x[, normalize, mask, ...])	average a list of images

average_images(*x*, *normalize=True*, *mask=None*, *imagetype=0*, *sum_image_threshold=3*, *return_sum_image=False*, *verbose=False*)

average a list of images

images will be resampled automatically to the largest image space; this is not a registration so images should be in the same physical space to begin with.

x : a list containing either filenames or antsImages

normalize : boolean

mask
[None or integer; this will perform a masked averaging which can] be useful when images have only partial coverage. integer greater than zero will perform morphological closing.

imagetype
[integer] choose 0/1/2/3 mapping to scalar/vector/tensor/time-series

sum_image_threshold
[integer] only average regions with overlap greater than or equal to this value

return_sum_image
[boolean] returns the average and the image that show ROI overlap; primarily for debugging

verbose
[boolean] will print progress

Return type
`ants.core.ANTsImage`

Example

```
>>> import ants
>>> x0=[ ants.get_data('r16'), ants.get_data('r27'), ants.get_data('r62'), ants.get_
↪data('r64') ]
>>> x1=[]
>>> for k in range(len(x0)):
>>>     x1.append( ants.image_read( x0[k] ) )
>>> avg=ants.average_images(x0)
>>> avg1=ants.average_images(x1)
>>> avg2=ants.average_images(x1,mask=0)
    >>> avg3=ants.average_images(x1,mask=1,normalize=True)
```

ants.math.get_centroids

get_centroids(*image*, *clustparam=0*)

Reduces a variate/statistical/network image to a set of centroids describing the center of each stand-alone non-zero component in the image

ANTsR function: *getCentroids*

Parameters

- **image** (`ants.core.ANTsImage`) – image from which centroids will be calculated
- **clustparam** (`int`) – look at regions greater than or equal to this size

Return type
`numpy.ndarray`

Example

```
>>> import ants
>>> image = ants.image_read( ants.get_ants_data( "r16" ) )
>>> image = ants.threshold_image( image, 90, 120 )
>>> image = ants.label_clusters( image, 10 )
>>> cents = ants.get_centroids( image )
```

ants.math.get_neighborhood

Functions

get_neighborhood_at_voxel(image, center, kernel)	Get a hypercube neighborhood at a voxel.
get_neighborhood_in_mask(image, mask, radius)	Get neighborhoods for voxels within mask.

get_neighborhood_at_voxel(*image*, *center*, *kernel*, *physical_coordinates=False*)

Get a hypercube neighborhood at a voxel. Get the values in a local neighborhood of an image.

ANTsR function: *getNeighborhoodAtVoxel*

Parameters

- **image** (ants.core.ANTsImage) – image to get values from.
- **center** (tuple/list) – indices for neighborhood center
- **kernel** (tuple/list) – either a collection of values for neighborhood radius (in voxels) or a binary collection of the same dimension as the image, specifying the shape of the neighborhood to extract
- **physical_coordinates** (bool) – whether voxel indices and offsets should be in voxel or physical coordinates

Returns

values
[ndarray] array of neighborhood values at the voxel

indices
[ndarray] matrix providing the coordinates for each value

Return type
dictionary w/ following key-value pairs

Example

```
>>> import ants
>>> img = ants.image_read(ants.get_ants_data('r16'))
>>> center = (2,2)
>>> radius = (3,3)
>>> retval = ants.get_neighborhood_at_voxel(img, center, radius)
```

get_neighborhood_in_mask(*image*, *mask*, *radius*, *physical_coordinates=False*, *boundary_condition=None*, *spatial_info=False*, *get_gradient=False*)

Get neighborhoods for voxels within mask.

This converts a scalar image to a matrix with rows that contain neighbors around a center voxel

ANTsR function: *getNeighborhoodInMask*

Parameters

- **image** (ants.core.ANTsImage) – image to get values from
- **mask** (ants.core.ANTsImage) – image indicating which voxels to examine. Each voxel > 0 will be used as the center of a neighborhood
- **radius** (tuple/list) – array of values for neighborhood radius (in voxels)

- **physical_coordinates** (bool) – whether voxel indices and offsets should be in voxel or physical coordinates
- **boundary_condition** (string (optional)) –

  how to handle voxels in a neighborhood, but not in the mask.
  None : fill values with *NaN image* : use image value, even if not in mask *mean* : use mean of all non-NaN values for that neighborhood
- **spatial_info** (bool) – whether voxel locations and neighborhood offsets should be returned along with pixel values.
- **get_gradient** (bool) – whether a matrix of gradients (at the center voxel) should be returned in addition to the value matrix (WIP)

Returns

- if spatial_info is False –

  if get_gradient is False:

  ndarray
  an array of pixel values where the number of rows is the size of the neighborhood and there is a column for each voxel

  else if get_gradient is True:

  dictionary w/ following key-value pairs:

  values
  [ndarray] array of pixel values where the number of rows is the size of the neighborhood and there is a column for each voxel.

  gradients
  [ndarray] array providing the gradients at the center voxel of each neighborhood
- else if spatial_info is True –

  dictionary w/ following key-value pairs:

  values
  [ndarray] array of pixel values where the number of rows is the size of the neighborhood and there is a column for each voxel.

  indices
  [ndarray] array provinding the center coordinates for each neighborhood

  offsets
  [ndarray] array providing the offsets from center for each voxel in a neighborhood

Example

```
>>> import ants
>>> r16 = ants.image_read(ants.get_ants_data('r16'))
>>> mask = ants.get_mask(r16)
>>> mat = ants.get_neighborhood_in_mask(r16, mask, radius=(2,2))
```

ants.math.hausdorff_distance

hausdorff_distance(*image1*, *image2*)

Get Hausdorff distance between non-zero pixels in two images

ANTsR function: *hausdorffDistance*

Parameters

- **image** (*source*) – Source image
- **target_image** (ants.core.ANTsImage) – Target image

Return type
data frame with "Distance" and "AverageDistance"

Example

```
>>> import ants
>>> r16 = ants.image_read( ants.get_ants_data('r16') )
>>> r64 = ants.image_read( ants.get_ants_data('r64') )
>>> s16 = ants.kmeans_segmentation( r16, 3 )['segmentation']
>>> s64 = ants.kmeans_segmentation( r64, 3 )['segmentation']
>>> stats = ants.hausdorff_distance(s16, s64)
```

ants.math.image_similarity

image_similarity(*fixed_image*, *moving_image*, *metric_type='MeanSquares'*, *fixed_mask=None*, *moving_mask=None*, *sampling_strategy='regular'*, *sampling_percentage=1.0*)

Measure similarity between two images. NOTE: Similarity is actually returned as distance (i.e. dissimilarity) per ITK/ANTs convention. E.g. using Correlation metric, the similarity of an image with itself returns -1.

ANTsR function: *imageSimilarity*

Parameters

- **fixed** (ants.core.ANTsImage) – the fixed image
- **moving** (ants.core.ANTsImage) – the moving image
- **metric_type** (str) –

  image metric to calculate
  MeanSquares Correlation ANTSNeighborhoodCorrelation MattesMutualInformation JointHistogramMutualInformation Demons
- **fixed_mask** (ANTsImage (optional)) – mask for the fixed image
- **moving_mask** (ANTsImage (optional)) – mask for the moving image
- **sampling_strategy** (string (optional)) –

  sampling strategy, default is full sampling
  None (Full sampling) random regular
- **sampling_percentage** (scalar) – percentage of data to sample when calculating metric Must be between 0 and 1

Return type
scalar

Example

```
>>> import ants
>>> x = ants.image_read(ants.get_ants_data('r16'))
>>> y = ants.image_read(ants.get_ants_data('r30'))
>>> metric = ants.image_similarity(x,y,metric_type='MeanSquares')
```

ants.math.metrics

Functions

image_mutual_information(image1, image2)	Compute mutual information between two ANTsImage types

image_mutual_information(*image1*, *image2*)

Compute mutual information between two ANTsImage types

ANTsR function: *antsImageMutualInformation*

Parameters

- **image1** (ants.core.ANTsImage) – image 1
- **image2** (ants.core.ANTsImage) – image 2

Return type
scalar

Example

```
>>> import ants
>>> fi = ants.image_read( ants.get_ants_data('r16') ).clone('float')
>>> mi = ants.image_read( ants.get_ants_data('r64') ).clone('float')
>>> mival = ants.image_mutual_information(fi, mi) # -0.1796141
```

ants.math.quantile

quantile(*image*, *q*, *nonzero=True*)

Get the quantile values from an ANTsImage

Examples

```
>>> img = ants.image_read(ants.get_data('r16'))
>>> ants.quantile(img, 0.5)
>>> ants.quantile(img, (0.5, 0.75))
```

8.1.11 ants.ops

Modules

add_noise_to_image(image, noise_model, ...)	Add noise to an image using additive Gaussian, salt-and-pepper, shot, or speckle noise.
anti_alias(image)	Apply Anti-Alias filter to a binary image

continues on next page

Table 30 – continued from previous page

bias_correction	
crop_image(image[, label_image, label])	Use a label image to crop a smaller ANTsImage from within a larger ANTsImage
denoise_image(image[, mask, shrink_factor, ...])	Denoise an image using a spatially adaptive filter originally described in J.
get_mask(image[, low_thresh, high_thresh, ...])	Get a binary mask image from the given image after thresholding
hessian_objectness(image[, ...])	Interface to ITK filter.
histogram_equalize_image(image[, ...])	Histogram equalize image
histogram_match_image(source_image, ...[, ...])	Histogram match source image to reference image.
iMath(image, operation, *args)	Perform various (often mathematical) operations on the input image/s.
image_type_cast(image_list[, pixeltype])	Cast a list of images to the highest pixeltype present in the list or all to a specified type
mask_image(image, mask[, level, binarize])	Mask an input image by a mask image.
morphology(image, operation, radius[, ...])	Apply morphological operations to an image
pad_image(image[, shape, pad_width, value, ...])	Pad an image to have the given shape or to be isotropic.
reflect_image(image[, axis, tx, metric])	Reflect an image along an axis
reorient_image	
resample_image(image, resample_params[, ...])	Resample image by spacing or number of voxels with various interpolators.
slice_image(image, axis, idx[, ...])	Slice an image.
smooth_image(image, sigma[, ...])	Smooth an image
symmetrize_image(image)	Use registration and reflection to make an image symmetric
threshold_image(image[, low_thresh, ...])	Converts a scalar image into a binary image by thresholding operations
weingarten_image_curvature(image[, sigma, opt])	Uses the weingarten map to estimate image mean or gaussian curvature

ants.ops.add_noise_to_image

add_noise_to_image(*image*, *noise_model*, *noise_parameters*)

Add noise to an image using additive Gaussian, salt-and-pepper, shot, or speckle noise.

Parameters

- **image** (ants.core.ANTsImage) – scalar image.
- **noise_model** (str) – 'additivegaussian', 'saltandpepper', 'shot', or 'speckle'.
- **noise_parameters** (tuple or numpy.ndarray or float) – 'additivegaussian': (mean, standardDeviation) 'saltandpepper': (probability, saltValue, pepperValue) 'shot': scale 'speckle': standardDeviation

Return type

ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> noise_image = ants.add_noise_to_image(image, 'additivegaussian', (0.0, 1.0))
```

(continues on next page)

(continued from previous page)

```
>>> noise_image = ants.add_noise_to_image(image, 'saltandpepper', (0.1, 0.0, 100.0))
>>> noise_image = ants.add_noise_to_image(image, 'shot', 1.0)
>>> noise_image = ants.add_noise_to_image(image, 'speckle', 1.0)
```

ants.ops.anti_alias

anti_alias(*image*)

Apply Anti-Alias filter to a binary image

ANTsR function: N/A

Parameters

image (ants.core.ANTsImage) – binary image to which anti-aliasing will be applied

Return type

ants.core.ANTsImage

Example

```
>>> import ants
>>> img = ants.image_read(ants.get_data('r16'))
>>> mask = ants.get_mask(img)
>>> mask_aa = ants.anti_alias(mask)
>>> ants.plot(mask)
>>> ants.plot(mask_aa)
```

ants.ops.bias_correction

Functions

abp_n4(image[, intensity_truncation, mask, ...])	Truncate outlier intensities and bias correct with the N4 algorithm.
n3_bias_field_correction(image[, ...])	N3 Bias Field Correction
n3_bias_field_correction2(image[, mask, ...])	N3 Bias Field Correction
n4_bias_field_correction(image[, mask, ...])	N4 Bias Field Correction

abp_n4(*image*, *intensity_truncation=(0.025, 0.975, 256)*, *mask=None*, *usen3=False*)

Truncate outlier intensities and bias correct with the N4 algorithm.

ANTsR function: *abpN4*

Parameters

- **image** (ants.core.ANTsImage) – image to correct and truncate
- **intensity_truncation** (typing.Tuple) – quantiles for intensity truncation
- **mask** (ANTsImage (optional)) – mask for bias correction
- **usen3** (bool) – if True, use N3 bias correction instead of N4

Return type

ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> image2 = ants.abp_n4(image)
```

n3_bias_field_correction(*image*, *downsample_factor=3*)

N3 Bias Field Correction

ANTsR function: *n3BiasFieldCorrection*

Parameters

- **image** (ants.core.ANTsImage) – image to be bias corrected
- **downsample_factor** (scalar) – how much to downsample image before performing bias correction

Return type

ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read( ants.get_ants_data('r16') )
>>> image_n3 = ants.n3_bias_field_correction(image)
```

n3_bias_field_correction2(*image*, *mask=None*, *rescale_intensities=False*, *shrink_factor=4*, *convergence={'iters': 50, 'tol': 1e-07}*, *spline_param=None*, *number_of_fitting_levels=4*, *return_bias_field=False*, *verbose=False*, *weight_mask=None*)

N3 Bias Field Correction

ANTsR function: *n3BiasFieldCorrection2*

Parameters

- **image** (ants.core.ANTsImage) – image to bias correct
- **mask** (ants.core.ANTsImage) – Input mask. If not specified, the entire image is used.
- **rescale_intensities** (bool) – At each iteration, a new intensity mapping is calculated and applied but there is nothing which constrains the new intensity range to be within certain values. The result is that the range can "drift" from the original at each iteration. This option rescales to the [min,max] range of the original image intensities within the user-specified mask. A mask is required to perform rescaling. Default is False in ANTsR/ANTsPy but True in ANTs.
- **shrink_factor** (scalar) – Shrink factor for multi-resolution correction, typically integer less than 4
- **convergence** (dict w/ keys `iters` and *tol*) – iters : maximum number of iterations tol : the convergence tolerance. Default tolerance is 1e-7 in ANTsR/ANTsPy but 0.0 in ANTs.
- **spline_param** (float or vector Parameter controlling number of control) – points in spline. Either single value, indicating the spacing in each direction, or vector with one entry per dimension of image, indicating the mesh size. If None, defaults to mesh size of 1 in all dimensions.
- **number_of_fitting_levels** (int) – Number of fitting levels per iteration.

- **return_bias_field** (bool) – Return bias field instead of bias corrected image.
- **verbose** (bool) – enables verbose output.
- **weight_mask** (ANTsImage (optional)) – antsImage of weight mask

Return type
ants.core.ANTsImage

Example

```
>>> image = ants.image_read( ants.get_ants_data('r16') )
>>> image_n3 = ants.n3_bias_field_correction2(image)
```

n4_bias_field_correction(*image*, *mask=None*, *rescale_intensities=False*, *shrink_factor=4*, *convergence={'iters': [50, 50, 50, 50], 'tol': 1e-07}*, *spline_param=None*, *return_bias_field=False*, *verbose=False*, *weight_mask=None*)

N4 Bias Field Correction

ANTsR function: *n4BiasFieldCorrection*

Parameters

- **image** (ants.core.ANTsImage) – image to bias correct
- **mask** (ants.core.ANTsImage) – Input mask. If not specified, the entire image is used.
- **rescale_intensities** (bool) – At each iteration, a new intensity mapping is calculated and applied but there is nothing which constrains the new intensity range to be within certain values. The result is that the range can "drift" from the original at each iteration. This option rescales to the [min,max] range of the original image intensities within the user-specified mask. A mask is required to perform rescaling. Default is False in ANTsR/ANTsPy but True in ANTs.
- **shrink_factor** (scalar) – Shrink factor for multi-resolution correction, typically integer less than 4
- **convergence** (dict w/ keys `iters` and *tol*) – iters : vector of maximum number of iterations for each level tol : the convergence tolerance. Default tolerance is 1e-7 in ANTsR/ANTsPy but 0.0 in ANTs.
- **spline_param** (float or vector) – Parameter controlling number of control points in spline. Either single value, indicating the spacing in each direction, or vector with one entry per dimension of image, indicating the mesh size. If None, defaults to mesh size of 1 in all dimensions.
- **return_bias_field** (bool) – Return bias field instead of bias corrected image.
- **verbose** (bool) – enables verbose output.
- **weight_mask** (ANTsImage (optional)) – antsImage of weight mask

Return type
ants.core.ANTsImage

Example

```
>>> image = ants.image_read( ants.get_ants_data('r16') )
>>> image_n4 = ants.n4_bias_field_correction(image)
```

ants.ops.crop_image

crop_image(*image*, *label_image=None*, *label=1*)

Use a label image to crop a smaller ANTsImage from within a larger ANTsImage

ANTsR function: *cropImage*

Parameters

- **image** (ants.core.ANTsImage) – image to crop
- **label_image** (ants.core.ANTsImage) – image with label values. If not supplied, estimated from data.
- **label** (int) – the label value to use

Return type

ants.core.ANTsImage

Example

```
>>> import ants
>>> fi = ants.image_read( ants.get_ants_data('r16') )
>>> cropped = ants.crop_image(fi)
>>> fi2 = ants.merge_channels([fi,fi])
>>> cropped2 = ants.crop_image(fi2)
>>> cropped = ants.crop_image(fi, fi, 100 )
```

ants.ops.denoise_image

denoise_image(*image*, *mask=None*, *shrink_factor=1*, *p=1*, *r=2*, *noise_model='Rician'*, *v=0*)

Denoise an image using a spatially adaptive filter originally described in J. V. Manjon, P. Coupe, Luis Marti-Bonmati, D. L. Collins, and M. Robles. Adaptive Non-Local Means Denoising of MR Images With Spatially Varying Noise Levels, Journal of Magnetic Resonance Imaging, 31:192-203, June 2010.

ANTsR function: *denoiseImage*

Parameters

- **image** (ants.core.ANTsImage) – scalar image to denoise.
- **mask** (ants.core.ANTsImage) – to limit the denoise region.
- **shrink_factor** (scalar) – downsampling level performed within the algorithm.
- **p** (int or character of format '2x2' where the x separates vector entries) – patch radius for local sample.
- **r** (int or character of format '2x2' where the x separates vector entries) – search radius from which to choose extra local samples.
- **noise_model** (str) – 'Rician' or 'Gaussian'

Return type

ants.core.ANTsImage

Example

```
>>> import ants
>>> import numpy as np
>>> image = ants.image_read(ants.get_ants_data('r16'))
```

(continues on next page)

(continued from previous page)

```
>>> # add fairly large salt and pepper noise
>>> imagenoise = image + np.random.randn(*image.shape).astype('float32')*5
>>> imagedenoise = ants.denoise_image(imagenoise, ants.get_mask(image))
```

ants.ops.get_mask

get_mask(*image*, *low_thresh=None*, *high_thresh=None*, *cleanup=2*)

Get a binary mask image from the given image after thresholding

ANTsR function: *getMask*

Parameters

- **image** (ants.core.ANTsImage) – image from which mask will be computed. Can be an antsImage of 2, 3 or 4 dimensions.
- **low_thresh** (scalar (optional)) – An inclusive lower threshold for voxels to be included in the mask. If not given, defaults to image mean.
- **high_thresh** (scalar (optional)) – An inclusive upper threshold for voxels to be included in the mask. If not given, defaults to image max
- **cleanup** (int) – If > 0, morphological operations will be applied to clean up the mask by eroding away small or weakly-connected areas, and closing holes. If cleanup is >0, the following steps are applied
  1. Erosion with radius 2 voxels
  2. Retain largest component
  3. Dilation with radius 1 voxel
  4. Morphological closing

Return type

ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read( ants.get_ants_data('r16') )
>>> mask = ants.get_mask(image)
```

ants.ops.hessian_objectness

hessian_objectness(*image*, *object_dimension=1*, *is_bright_object=True*, *sigma_min=0.1*, *sigma_max=10*, *number_of_sigma_steps=10*, *use_sigma_logarithmic_spacing=True*, *alpha=0.5*, *beta=0.5*, *gamma=5.0*, *set_scale_objectness_measure=True*)

Interface to ITK filter. Based on the paper by Westin et al., "Geometrical Diffusion Measures for MRI from Tensor Basis Analysis" and Luca Antiga's Insight Journal paper http://hdl.handle.net/1926/576.

Parameters

- **image** (ants.core.ANTsImage) – scalar image.
- **object_dimension** (unsigned int) – 0: 'sphere', 1: 'line', or 2: 'plane'.
- **is_bright_object** (bool) – Set 'true' for enhancing bright objects and 'false' for dark objects.
- **sigma_min** (float) – Define scale domain for feature extraction.

- **sigma_max** (float) – Define scale domain for feature extraction.
- **number_of_sigma_steps** (unsigned int) – Define number of samples for scale space.
- **use_sigma_logarithmic_spacing** (bool) – Define sample spacing the for scale space.
- **alpha** (float) – Hessian filter parameter.
- **beta** (float) – Hessian filter parameter.
- **gamma** (float) – Hessian filter parameter.
- **set_scale_objectness_measure** (bool) – ...

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> hessian_object_image = ants.hessian_objectness(image)
```

ants.ops.histogram_equalize_image

histogram_equalize_image(*image*, *number_of_histogram_bins=256*)

Histogram equalize image

from http://www.janeriksolem.net/histogram-equalization-with-python-and.html

Parameters

- **image** (ants.core.ANTsImage) – source image
- **number_of_histogram_bins** (int) – number of bins for cumulative histogram

Example

```
>>> import ants
>>> src_img = ants.image_read(ants.get_data('r16'))
>>> src_img_eq = ants.histogram_equalize_image(src_img)
```

ants.ops.histogram_match_image

histogram_match_image(*source_image*, *reference_image*, *number_of_histogram_bins=255*, *number_of_match_points=64*, *use_threshold_at_mean_intensity=False*)

Histogram match source image to reference image.

Parameters

- **source_image** (ants.core.ANTsImage) – source image
- **reference_image** (ants.core.ANTsImage) – reference image
- **number_of_histogram_bins** (int) – number of bins for source and reference histograms
- **number_of_match_points** (int) – number of points for histogram matching
- **use_threshold_at_mean_intensity** (bool) – see ITK description.

Example

```
>>> import ants
>>> src_img = ants.image_read(ants.get_data('r16'))
>>> ref_img = ants.image_read(ants.get_data('r64'))
>>> src_ref = ants.histogram_match_image(src_img, ref_img)
```

ants.ops.iMath

iMath(*image*, *operation*, **args*)

Perform various (often mathematical) operations on the input image/s. Additional parameters should be specific for each operation. See the the full iMath in ANTs, on which this function is based.

ANTsR function: *iMath*

Parameters

- **image** (ants.core.ANTsImage) – input object, usually antsImage
- **operation** – a string e.g. "GetLargestComponent" ... the special case of "GetOperations" or "GetOperationsFull" will return a list of operations and brief description. Some operations may not be valid (WIP), but most are.
- ***args** (non-keyword arguments) – additional parameters specific to the operation

Example

```
>>> import ants
>>> img = ants.image_read(ants.get_ants_data('r16'))
>>> img2 = ants.iMath(img, 'Canny', 1, 5, 12)
```

ants.ops.image_type_cast

image_type_cast(*image_list*, *pixeltype=None*)

Cast a list of images to the highest pixeltype present in the list or all to a specified type

ANTsR function: *antsImageTypeCast*

Parameters

- **image_list** (list/tuple) – images to cast
- **pixeltype** (string (optional)) – pixeltype to cast to. If None, images will be cast to the highest precision pixeltype found in image_list

Returns

given images casted to new type

Return type

list of ANTsImages

ants.ops.mask_image

mask_image(*image*, *mask*, *level=1*, *binarize=False*)

Mask an input image by a mask image. If the mask image has multiple labels, it is possible to specify which label(s) to mask at.

ANTsR function: *maskImage*

Parameters

- **image** (ants.core.ANTsImage) – Input image.

- **mask** (ants.core.ANTsImage) – Mask or label image.
- **level** (scalar or tuple of scalars) – Level(s) at which to threshold the mask. Can be a single value or an iterable of values.
- **binarize** (bool) – whether to binarize the output image. This computes an intersection between (image != 0) and (mask == i), for all i in level.

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> myimage = ants.image_read(ants.get_ants_data('r16'))
>>> mask = ants.get_mask(myimage)
>>> myimage_mask = ants.mask_image(myimage, mask, 3)
>>> seg = ants.kmeans_segmentation(myimage, 3)
>>> myimage_mask = ants.mask_image(myimage, seg['segmentation'], (1,3))
```

ants.ops.morphology

morphology(*image*, *operation*, *radius*, *mtype='binary'*, *value=1*, *shape='ball'*, *radius_is_parametric=False*, *thickness=1*, *lines=3*, *include_center=False*)

Apply morphological operations to an image

ANTsR function: *morphology*

Parameters

- **input** (ants.core.ANTsImage) – input image
- **operation** (str) –

  operation to apply
  ”close” Morpholgical closing “dilate” Morpholgical dilation “erode” Morpholgical erosion “open” Morpholgical opening
- **radius** (scalar) – radius of structuring element
- **mtype** (str) –

  type of morphology
  ”binary” Binary operation on a single value “grayscale” Grayscale operations
- **value** (scalar) – value to operation on (type=’binary’ only)
- **shape** (str) –

  shape of the structuring element (type=’binary’ only)
  ”ball” spherical structuring element “box” box shaped structuring element “cross” cross shaped structuring element “annulus” annulus shaped structuring element “polygon” polygon structuring element
- **radius_is_parametric** (bool) – used parametric radius boolean (shape=’ball’ and shape=’annulus’ only)
- **thickness** (scalar) – thickness (shape=’annulus’ only)
- **lines** (int) – number of lines in polygon (shape=’polygon’ only)
- **include_center** (bool) – include center of annulus boolean (shape=’annulus’ only)

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> fi = ants.image_read( ants.get_ants_data('r16') , 2 )
>>> mask = ants.get_mask( fi )
>>> dilated_ball = ants.morphology( mask, operation='dilate', radius=3, mtype=
↪'binary', shape='ball')
>>> eroded_box = ants.morphology( mask, operation='erode', radius=3, mtype='binary',
↪ shape='box')
>>> opened_annulus = ants.morphology( mask, operation='open', radius=5, mtype=
↪'binary', shape='annulus', thickness=2)
```

ants.ops.pad_image

pad_image(*image*, *shape=None*, *pad_width=None*, *value=0.0*, *return_padvals=False*)

Pad an image to have the given shape or to be isotropic.

Parameters

- **image** (ants.core.ANTsImage) – image to pad
- **shape** (tuple) –
  - if shape is given, the image will be padded in each dimension until it has this shape
  - if shape and pad_width are both None, the image will be padded along each dimension to match the largest existing dimension so that it has isotropic dimensions.
- **pad_width** (list of integers or list-of-list of integers) – How much to pad in each direction. If a single list is supplied (e.g., [4,4,4]), then the image will be padded by half that amount on both sides. If a list-of-list is supplied (e.g., [(0,4),(0,4),(0,4)]), then the image will be padded unevenly on the different sides
- **pad_value** (scalar) – value with which image will be padded

Example

```
>>> import ants
>>> img = ants.image_read(ants.get_data('r16'))
>>> img2 = ants.pad_image(img, shape=(300,300))
>>> mni = ants.image_read(ants.get_data('mni'))
>>> mni2 = ants.pad_image(mni)
>>> mni3 = ants.pad_image(mni, pad_width=[(0,4),(0,4),(0,4)])
>>> mni4 = ants.pad_image(mni, pad_width=(4,4,4))
```

ants.ops.reflect_image

reflect_image(*image*, *axis=None*, *tx=None*, *metric='mattes'*)

Reflect an image along an axis

ANTsR function: *reflectImage*

Parameters

- **image** (ants.core.ANTsImage) – image to reflect

- **axis** (integer (optional)) – which dimension to reflect across, numbered from 0 to imageDimension-1
- **tx** (string (optional)) – transformation type to estimate after reflection
- **metric** (str) – similarity metric for image registration. see antsRegistration.

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> fi = ants.image_read( ants.get_ants_data('r16'), 'float' )
>>> axis = 2
>>> asym = ants.reflect_image(fi, axis, 'Affine')['warpedmovout']
>>> asym = asym - fi
```

ants.ops.reorient_image

Functions

get_center_of_mass(image)	Compute an image center of mass in physical space which is defined as the mean of the intensity weighted voxel coordinate system.
get_orientation(image)	
get_possible_orientations()	
reorient_image2(image[, orientation])	Reorient an image.

get_center_of_mass(*image*)

Compute an image center of mass in physical space which is defined as the mean of the intensity weighted voxel coordinate system.

ANTsR function: *getCenterOfMass*

Parameters
image (ants.core.ANTsImage) – image from which center of mass will be computed

Return type
scalar

Example

```
>>> fi = ants.image_read( ants.get_ants_data("r16"))
>>> com1 = ants.get_center_of_mass( fi )
>>> fi = ants.image_read( ants.get_ants_data("r64"))
>>> com2 = ants.get_center_of_mass( fi )
```

get_orientation(*image*)

get_possible_orientations()

reorient_image2(*image*, *orientation='RAS'*)

Reorient an image. .. rubric:: Example

```
>>> import ants
>>> mni = ants.image_read(ants.get_data('mni'))
>>> mni2 = mni.reorient_image2()
```

ants.ops.resample_image

resample_image(*image*, *resample_params*, *use_voxels=False*, *interp_type=1*)

Resample image by spacing or number of voxels with various interpolators. Works with multi-channel images.

ANTsR function: *resampleImage*

Parameters

- **image** (ants.core.ANTsImage) – input image
- **resample_params** (tuple/list) – vector of size dimension with numeric values
- **use_voxels** (bool) – True means interpret resample params as voxel counts
- **interp_type** (int) – one of 0 (linear), 1 (nearest neighbor), 2 (gaussian), 3 (windowed sinc), 4 (bspline)

Return type

ants.core.ANTsImage

Example

```
>>> import ants
>>> fi = ants.image_read( ants.get_ants_data("r16"))
>>> finn = ants.resample_image(fi,(50,60),True,0)
>>> filin = ants.resample_image(fi,(1.5,1.5),False,1)
>>> img = ants.image_read( ants.get_ants_data("r16"))
>>> img = ants.merge_channels([img, img])
>>> outimg = ants.resample_image(img, (128,128), True)
```

ants.ops.slice_image

slice_image(*image*, *axis*, *idx*, *collapse_strategy=0*)

Slice an image.

Parameters

- **axis** (int) – Which axis.
- **idx** (int) – Which slice number.
- **collapse_strategy** (int) – Collapse strategy for sub-matrix: 0, 1, or 2. 0: collapse to sub-matrix if positive-definite. Otherwise throw an exception. Default. 1: Collapse to identity. 2: Collapse to sub-matrix if positive definite. Otherwise collapse to identity.

Example

```
>>> import ants
>>> mni = ants.image_read(ants.get_data('mni'))
>>> mni2 = ants.slice_image(mni, axis=1, idx=100)
```

ants.ops.smooth_image

smooth_image(*image*, *sigma*, *sigma_in_physical_coordinates=True*, *FWHM=False*, *max_kernel_width=32*)

Smooth an image

ANTsR function: *smoothImage*

Parameters

- **image** – Image to smooth
- **sigma** – Smoothing factor. Can be scalar, in which case the same sigma is applied to each dimension, or a vector of length dim(inimage) to specify a unique smoothness for each dimension.
- **sigma_in_physical_coordinates** (bool) – If true, the smoothing factor is in millimeters; if false, it is in pixels.
- **FWHM** (bool) – If true, sigma is interpreted as the full-width-half-max (FWHM) of the filter, not the sigma of a Gaussian kernel.
- **max_kernel_width** (scalar) – Maximum kernel width

Return type

ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read( ants.get_ants_data('r16'))
>>> simage = ants.smooth_image(image, (1.2,1.5))
```

ants.ops.symmetrize_image

symmetrize_image(*image*)

Use registration and reflection to make an image symmetric

ANTsR function: N/A

Parameters

image (ants.core.ANTsImage) – image to make symmetric

Return type

ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read( ants.get_ants_data('r16') , 'float')
>>> simage = ants.symimage(image)
```

ants.ops.threshold_image

threshold_image(*image*, *low_thresh=None*, *high_thresh=None*, *inval=1*, *outval=0*, *binary=True*)

Converts a scalar image into a binary image by thresholding operations

ANTsR function: *thresholdImage*

Parameters

- **image** (ants.core.ANTsImage) – Input image to operate on

- **low_thresh** (scalar (optional)) – Lower edge of threshold window
- **hight_thresh** (scalar (optional)) – Higher edge of threshold window
- **inval** (scalar) – Output value for image voxels in between lothresh and hithresh
- **outval** (scalar) – Output value for image voxels lower than lothresh or higher than hithresh
- **binary** (bool) – if true, returns binary thresholded image if false, return binary thresholded image multiplied by original image

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read( ants.get_ants_data('r16') )
>>> timage = ants.threshold_image(image, 0.5, 1e15)
```

ants.ops.weingarten_image_curvature

weingarten_image_curvature(*image*, *sigma=1.0*, *opt='mean'*)

Uses the weingarten map to estimate image mean or gaussian curvature

ANTsR function: *weingartenImageCurvature*

Parameters

- **image** (ants.core.ANTsImage) – image from which curvature is calculated
- **sigma** (scalar) – smoothing parameter
- **opt** (str) – mean by default, otherwise *gaussian* or *characterize*

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('mni')).resample_image((3,3,3))
>>> imagecurv = ants.weingarten_image_curvature(image)
```

8.1.12 ants.plotting

Modules

movie(image[, filename, writer, fps])	Create and save a movie - mp4, gif, etc - of the various 2D slices of a 3D ants image
plot(image[, overlay, blend, alpha, cmap, ...])	Plot an ANTsImage.
plot_directory(directory[, recursive, ...])	Create and save an ANTsPy plot for every image matching a given regular expression in a directory, optionally recursively.
plot_grid(images[, slices, axes, figsize, ...])	Plot a collection of images in an arbitrarily-defined grid
plot_hist(image[, threshold, fit_line, ...])	Plot a histogram from an ANTsImage

continues on next page

Table 33 – continued from previous page

plot_ortho(image[, overlay, reorient, ...])	Plot an orthographic view of a 3D image
plot_ortho_stack(images[, overlays, ...])	Create a stack of orthographic plots with optional overlays.

ants.plotting.movie

movie(*image*, *filename=None*, *writer=None*, *fps=30*)

Create and save a movie - mp4, gif, etc - of the various 2D slices of a 3D ants image
Try this:
conda install -c conda-forge ffmpeg

Example

```
>>> import ants
>>> mni = ants.image_read(ants.get_data('mni'))
>>> ants.movie(mni, filename='~/desktop/movie.mp4')
```

ants.plotting.plot

plot(*image*, *overlay=None*, *blend=False*, *alpha=1*, *cmap='Greys_r'*, *overlay_cmap='turbo'*, *overlay_alpha=0.9*, *vminol=None*, *vmaxol=None*, *cbar=False*, *cbar_length=0.8*, *cbar_dx=0.0*, *cbar_vertical=True*, *axis=0*, *nslices=12*, *slices=None*, *ncol=None*, *slice_buffer=None*, *black_bg=True*, *bg_thresh_quant=0.01*, *bg_val_quant=0.99*, *domain_image_map=None*, *crop=False*, *scale=False*, *reverse=False*, *title=None*, *title_fontsize=20*, *title_dx=0.0*, *title_dy=0.0*, *filename=None*, *dpi=500*, *figsize=1.5*, *reorient=True*, *resample=True*)

Plot an ANTsImage.

Use mask_image and/or threshold_image to preprocess images to be be overlaid and display the overlays in a given range. See the wiki examples.

By default, images will be reoriented to 'LAI' orientation before plotting. So, if axis == 0, the images will be ordered from the left side of the brain to the right side of the brain. If axis == 1, the images will be ordered from the anterior (front) of the brain to the posterior (back) of the brain. And if axis == 2, the images will be ordered from the inferior (bottom) of the brain to the superior (top) of the brain.

ANTsR function: *plot.antsImage*

Parameters

- **image** (`ants.core.ANTsImage`) – image to plot
- **overlay** (`ants.core.ANTsImage`) – image to overlay on base image
- **cmap** (`str`) – colormap to use for base image. See matplotlib.
- **overlay_cmap** (`str`) – colormap to use for overlay images, if applicable. See matplotlib.
- **overlay_alpha** (`float`) – level of transparency for any overlays. Smaller value means the overlay is more transparent. See matplotlib.
- **axis** (`int`) – which axis to plot along if image is 3D
- **nslices** (`int`) – number of slices to plot if image is 3D
- **slices** (`list` or `tuple` of `integers`) – specific slice indices to plot if image is 3D. If given, this will override *nslices*. This can be absolute array indices (e.g. (80,100,120)), or this can be relative array indices (e.g. (0.4,0.5,0.6))

- **ncol** (int) – Number of columns to have on the plot if image is 3D.
- **slice_buffer** (int) – how many slices to buffer when finding the non-zero slices of a 3D images. So, if slice_buffer = 10, then the first slice in a 3D image will be the first non-zero slice index plus 10 more slices.
- **black_bg** (bool) – if True, the background of the image(s) will be black. if False, the background of the image(s) will be determined by the

  values *bg_thresh_quant* and *bg_val_quant*.
- **bg_thresh_quant** (float) – if white_bg=True, the background will be determined by thresholding the image at the *bg_thresh* quantile value and setting the background intensity to the *bg_val* quantile value. This value should be in [0, 1] - somewhere around 0.01 is recommended.
  - equal to 1 will threshold the entire image
  - equal to 0 will threshold none of the image
- **bg_val_quant** (float) – if white_bg=True, the background will be determined by thresholding the image at the *bg_thresh* quantile value and setting the background intensity to the *bg_val* quantile value. This value should be in [0, 1]
  - equal to 1 is pure white
  - equal to 0 is pure black
  - somewhere in between is gray
- **domain_image_map** (ants.core.ANTsImage) – this input ANTsImage or list of ANTsImage types contains a reference image *domain_image* and optional reference mapping named *domainMap*. If supplied, the image(s) to be plotted will be mapped to the domain image space before plotting - useful for non-standard image orientations.
- **crop** (bool) – if true, the image(s) will be cropped to their bounding boxes, resulting in a potentially smaller image size. if false, the image(s) will not be cropped
- **scale** (bool or typing.Tuple) – if true, nothing will happen to intensities of image(s) and overlay(s) if false, dynamic range will be maximized when visualizing overlays if 2-tuple, the image will be dynamically scaled between these quantiles
- **reverse** (bool) – if true, the order in which the slices are plotted will be reversed. This is useful if you want to plot from the front of the brain first to the back of the brain, or vice-versa
- **title** (str) – add a title to the plot
- **filename** (str) – if given, the resulting image will be saved to this file
- **dpi** (int) – determines resolution of image if saved to file. Higher values result in higher resolution images, but at a cost of having a larger file size
- **resample** (bool) – if true, resample image if spacing is very unbalanced.

Example

```
>>> import ants
>>> import numpy as np
>>> img = ants.image_read(ants.get_data('r16'))
>>> segs = img.kmeans_segmentation(k=3)['segmentation']
>>> ants.plot(img, segs*(segs==1), crop=True)
```

(continues on next page)

(continued from previous page)

```
>>> ants.plot(img, segs*(segs==1), crop=False)
>>> mni = ants.image_read(ants.get_data('mni'))
>>> segs = mni.kmeans_segmentation(k=3)['segmentation']
>>> ants.plot(mni, segs*(segs==1), crop=False)
```

ants.plotting.plot_directory

plot_directory(*directory*, *recursive=False*, *regex='*'*, *save_prefix=''*, *save_suffix=''*, *axis=None*, ***kwargs*)

Create and save an ANTsPy plot for every image matching a given regular expression in a directory, optionally recursively. This is a good function for quick visualize exploration of all of images in a directory

ANTsR function: N/A

Parameters

- **directory** (str) – directory in which to search for images and plot them
- **recursive** (bool) – If true, this function will search through all directories under the given directory recursively to make plots. If false, this function will only create plots for images in the given directory
- **regex** (str) – regular expression used to filter out certain filenames or suffixes
- **save_prefix** (str) – sub-string that will be appended to the beginning of all saved plot filenames. Default is to add nothing.
- **save_suffix** (str) – sub-string that will be appended to the end of all saved plot filenames. Default is add nothing.
- **kwargs** (keyword arguments) – any additional arguments to pass onto the *ants.plot* function. e.g. overlay, alpha, cmap, etc. See *ants.plot* for more options.

Example

```
>>> import ants
>>> ants.plot_directory(directory='~/desktop/testdir',
                        recursive=False, regex='*')
```

ants.plotting.plot_grid

plot_grid(*images*, *slices=None*, *axes=2*, *figsize=1.0*, *rpad=0*, *cpad=0*, *vmin=None*, *vmax=None*, *colorbar=True*, *cmap='Greys_r'*, *title=None*, *tfontsize=20*, *title_dx=0*, *title_dy=0*, *rlabels=None*, *rfontsize=14*, *rfontcolor='white'*, *rfacecolor='black'*, *clabels=None*, *cfontsize=14*, *cfontcolor='white'*, *cfacecolor='black'*, *filename=None*, *dpi=400*, *transparent=True*, ***kwargs*)

Plot a collection of images in an arbitrarily-defined grid

Matplotlib named colors: https://matplotlib.org/examples/color/named_colors.html

Parameters

- **images** (list of ANTsImage types) – image(s) to plot. if one image, this image will be used for all grid locations. if multiple images, they should be arrange in a list the same shape as the *gridsize* argument.
- **slices** (int or list of integers) – slice indices to plot if one integer, this slice index will be used for all images if multiple integers, they should be arranged in a list the same shape as the *gridsize* argument

- **axes** (int or list of integers) – axis or axes along which to plot image slices if one integer, this axis will be used for all images if multiple integers, they should be arranged in a list the same shape as the *gridsize* argument

Example

```
>>> import ants
>>> import numpy as np
>>> mni1 = ants.image_read(ants.get_data('mni'))
>>> mni2 = mni1.smooth_image(1.)
>>> mni3 = mni1.smooth_image(2.)
>>> mni4 = mni1.smooth_image(3.)
>>> images = np.asarray([[mni1, mni2],
...                      [mni3, mni4]])
>>> slices = np.asarray([[100, 100],
...                      [100, 100]])
>>> ants.plot_grid(images=images, slices=slices, title='2x2 Grid')
>>> images2d = np.asarray([[mni1.slice_image(2,100), mni2.slice_image(2,100)],
...                        [mni3.slice_image(2,100), mni4.slice_image(2,100)]])
>>> ants.plot_grid(images=images2d, title='2x2 Grid Pre-Sliced')
>>> ants.plot_grid(images.reshape(1,4), slices.reshape(1,4), title='1x4 Grid')
>>> ants.plot_grid(images.reshape(4,1), slices.reshape(4,1), title='4x1 Grid')
```

```
>>> # Padding between rows and/or columns
>>> ants.plot_grid(images, slices, cpad=0.02, title='Col Padding')
>>> ants.plot_grid(images, slices, rpad=0.02, title='Row Padding')
>>> ants.plot_grid(images, slices, rpad=0.02, cpad=0.02, title='Row and Col Padding
↪')
```

```
>>> # Adding plain row and/or column labels
>>> ants.plot_grid(images, slices, title='Adding Row Labels', rlabels=['Row #1',
↪'Row #2'])
>>> ants.plot_grid(images, slices, title='Adding Col Labels', clabels=['Col #1',
↪'Col #2'])
>>> ants.plot_grid(images, slices, title='Row and Col Labels',
                   rlabels=['Row 1', 'Row 2'], clabels=['Col 1', 'Col 2'])
```

```
>>> # Making a publication-quality image
>>> images = np.asarray([[mni1, mni2, mni2],
...                      [mni3, mni4, mni4]])
>>> slices = np.asarray([[100, 100, 100],
...                      [100, 100, 100]])
>>> axes = np.asarray([[0, 1, 2],
                       [0, 1, 2]])
>>> ants.plot_grid(images, slices, axes, title='Publication Figures with ANTsPy',
                   tfontsize=20, title_dy=0.03, title_dx=-0.04,
                   rlabels=['Row 1', 'Row 2'],
                   clabels=['Col 1', 'Col 2', 'Col 3'],
                   rfontsize=16, cfontsize=16)
```

ants.plotting.plot_hist

plot_hist(*image*, *threshold=0.0*, *fit_line=False*, *normfreq=True*, *title=None*, *grid=True*, *xlabel=None*, *ylabel=None*, *facecolor='green'*, *alpha=0.75*)

Plot a histogram from an ANTsImage

Parameters

image (ants.core.ANTsImage) – image from which histogram will be created

ants.plotting.plot_ortho

plot_ortho(*image*, *overlay=None*, *reorient=True*, *blend=False*, *xyz=None*, *xyz_lines=True*, *xyz_color='red'*, *xyz_alpha=0.6*, *xyz_linewidth=2*, *xyz_pad=5*, *orient_labels=True*, *alpha=1*, *cmap='Greys_r'*, *overlay_cmap='jet'*, *overlay_alpha=0.9*, *cbar=False*, *cbar_length=0.8*, *cbar_dx=0.0*, *cbar_vertical=True*, *black_bg=True*, *bg_thresh_quant=0.01*, *bg_val_quant=0.99*, *crop=False*, *scale=False*, *domain_image_map=None*, *title=None*, *titlefontsize=24*, *title_dx=0*, *title_dy=0*, *text=None*, *textfontsize=24*, *textfontcolor='white'*, *text_dx=0*, *text_dy=0*, *filename=None*, *dpi=500*, *figsize=1.0*, *flat=False*, *transparent=True*, *resample=False*, *allow_xyz_change=True*)

Plot an orthographic view of a 3D image

Use mask_image and/or threshold_image to preprocess images to be be overlaid and display the overlays in a given range. See the wiki examples.

ANTsR function: N/A

image
[ANTsImage] image to plot

overlay
[ANTsImage] image to overlay on base image

xyz
[list or tuple of 3 integers] selects index location on which to center display if given, solid lines will be drawn to converge at this coordinate. This is useful for pinpointing a specific location in the image.

flat
[boolean] if true, the ortho image will be plot in one row if false, the ortho image will be a 2x2 grid with the bottom

left corner blank

cmap
[string] colormap to use for base image. See matplotlib.

overlay_cmap
[string] colormap to use for overlay images, if applicable. See matplotlib.

overlay_alpha
[float] level of transparency for any overlays. Smaller value means the overlay is more transparent. See matplotlib.

cbar: boolean
if true, a colorbar will be added to the plot

cbar_length: float
length of the colorbar relative to the image

cbar_dx: float
horizontal shift of the colorbar relative to the image

cbar_vertical: boolean
if true, the colorbar will be vertical, if false, it will be horizontal underneath the image

axis
[integer] which axis to plot along if image is 3D

black_bg
[boolean] if True, the background of the image(s) will be black. if False, the background of the image(s) will be determined by the

values *bg_thresh_quant* and *bg_val_quant*.

bg_thresh_quant
[float] if white_bg=True, the background will be determined by thresholding the image at the *bg_thresh* quantile value and setting the background intensity to the *bg_val* quantile value. This value should be in [0, 1] - somewhere around 0.01 is recommended.

- equal to 1 will threshold the entire image
- equal to 0 will threshold none of the image

bg_val_quant
[float] if white_bg=True, the background will be determined by thresholding the image at the *bg_thresh* quantile value and setting the background intensity to the *bg_val* quantile value. This value should be in [0, 1]

- equal to 1 is pure white
- equal to 0 is pure black
- somewhere in between is gray

domain_image_map
[ANTsImage] this input ANTsImage or list of ANTsImage types contains a reference image *domain_image* and optional reference mapping named *domainMap*. If supplied, the image(s) to be plotted will be mapped to the domain image space before plotting - useful for non-standard image orientations.

crop
[boolean] if true, the image(s) will be cropped to their bounding boxes, resulting in a potentially smaller image size. if false, the image(s) will not be cropped

scale
[boolean or 2-tuple] if true, nothing will happen to intensities of image(s) and overlay(s) if false, dynamic range will be maximized when visualizing overlays if 2-tuple, the image will be dynamically scaled between these quantiles

title
[string] add a title to the plot

filename
[string] if given, the resulting image will be saved to this file

dpi
[integer] determines resolution of image if saved to file. Higher values result in higher resolution images, but at a cost of having a larger file size

resample : resample image in case of unbalanced spacing

allow_xyz_change : boolean will attempt to adjust xyz after padding

```
>>> import ants
>>> mni = ants.image_read(ants.get_data('mni'))
>>> ants.plot_ortho(mni, xyz=(100,100,100))
>>> mni2 = mni.threshold_image(7000, mni.max())
>>> ants.plot_ortho(mni, overlay=mni2)
>>> ants.plot_ortho(mni, overlay=mni2, flat=True)
>>> ants.plot_ortho(mni, overlay=mni2, xyz=(110,110,110), xyz_lines=False,
                    text='Lines Turned Off', textfontsize=22)
>>> ants.plot_ortho(mni, mni2, xyz=(120,100,100),
                    text=' Example
```

Ortho Text', textfontsize=26,
title='Example Ortho Title', titlefontsize=26)

ants.plotting.plot_ortho_stack

plot_ortho_stack(*images*, *overlays=None*, *reorient=True*, *xyz=None*, *xyz_lines=False*, *xyz_color='red'*, *xyz_alpha=0.6*, *xyz_linewidth=2*, *xyz_pad=5*, *cmap='Greys_r'*, *alpha=1*, *overlay_cmap='jet'*, *overlay_alpha=0.9*, *black_bg=True*, *bg_thresh_quant=0.01*, *bg_val_quant=0.99*, *crop=False*, *scale=False*, *domain_image_map=None*, *title=None*, *titlefontsize=24*, *title_dx=0*, *title_dy=0*, *text=None*, *textfontsize=24*, *textfontcolor='white'*, *text_dx=0*, *text_dy=0*, *filename=None*, *dpi=500*, *figsize=1.0*, *colpad=0*, *rowpad=0*, *transpose=False*, *transparent=True*, *orient_labels=True*)

Create a stack of orthographic plots with optional overlays.

Use mask_image and/or threshold_image to preprocess images to be be overlaid and display the overlays in a given range. See the wiki examples.

Example

```
>>> import ants
>>> mni = ants.image_read(ants.get_data('mni'))
>>> ch2 = ants.image_read(ants.get_data('ch2'))
>>> ants.plot_ortho_stack([mni,mni,mni])
```

8.1.13 ants.registration

registration(*fixed*, *moving*, *type_of_transform='SyN'*, *initial_transform=None*, *outprefix=''*, *mask=None*, *moving_mask=None*, *mask_all_stages=False*, *grad_step=0.2*, *flow_sigma=3*, *total_sigma=0*, *aff_metric='mattes'*, *aff_sampling=32*, *aff_random_sampling_rate=0.2*, *syn_metric='mattes'*, *syn_sampling=32*, *reg_iterations=(40, 20, 0)*, *aff_iterations=(2100, 1200, 1200, 10)*, *aff_shrink_factors=(6, 4, 2, 1)*, *aff_smoothing_sigmas=(3, 2, 1, 0)*, *write_composite_transform=False*, *verbose=False*, *multivariate_extras=None*, *restrict_transformation=None*, *smoothing_in_mm=False*, *singleprecision=True*, *use_legacy_histogram_matching=False*, ***kwargs*)

Register a pair of images either through the full or simplified interface to the ANTs registration method.

ANTsR function: *antsRegistration*

Parameters

- **fixed** (ants.core.ANTsImage) – fixed image to which we register the moving image.
- **moving** (ants.core.ANTsImage) – moving image to be mapped to fixed space.

- **type_of_transform** (str) – A linear or non-linear registration type. Mutual information metric by default. See Notes below for more.
- **initial_transform** (list of strings (optional)) – transforms to prepend. If None, a translation is computed to align the image centers of mass, unless the type of transform is deformable-only (time-varying diffeomorphisms, SyNOnly, or antsRegistrationSyN*[so|bo]). To force initialization with an identity transform, set this to 'Identity'.
- **outprefix** (str) – output will be named with this prefix.
- **mask** (ANTsImage (optional)) – Registration metric mask in the fixed image space.
- **moving_mask** (ANTsImage (optional)) – Registration metric mask in the moving image space.
- **mask_all_stages** (bool) – If true, apply metric mask(s) to all registration stages, instead of just the final stage.
- **grad_step** (scalar) – gradient step size (not for all tx)
- **flow_sigma** (scalar) – smoothing for update field At each iteration, the similarity metric and gradient is calculated. That gradient field is also called the update field and is smoothed before composing with the total field (i.e., the estimate of the total transform at that iteration). This total field can also be smoothed after each iteration.
- **total_sigma** (scalar) – smoothing for total field
- **aff_metric** (str) – the metric for the affine part (GC, mattes, meansquares)
- **aff_sampling** (scalar) – number of bins for the mutual information metric
- **aff_random_sampling_rate** (scalar) – the fraction of points used to estimate the metric. this can impact speed but also reproducibility and/or accuracy.
- **syn_metric** (str) – the metric for the syn part (CC, mattes, meansquares, demons)
- **syn_sampling** (scalar) – the nbins or radius parameter for the syn metric
- **reg_iterations** (list/tuple of integers) – vector of iterations for syn. we will set the smoothing and multi-resolution parameters based on the length of this vector.
- **aff_iterations** (list/tuple of integers) – vector of iterations for low-dimensional (translation, rigid, affine) registration.
- **aff_shrink_factors** (list/tuple of integers) – vector of multi-resolution shrink factors for low-dimensional (translation, rigid, affine) registration.
- **aff_smoothing_sigmas** (list/tuple of integers) – vector of multi-resolution smoothing factors for low-dimensional (translation, rigid, affine) registration.
- **write_composite_transform** (bool) – Boolean specifying whether or not the composite transform (and its inverse, if it exists) should be written to an hdf5 composite file. This is false by default so that only the transform for each stage is written to file.
- **verbose** (bool) – request verbose output (useful for debugging)
- **multivariate_extras** (additional metrics for multi-metric registration) – list of additional images and metrics which will trigger the use of multiple metrics in the registration process in the deformable stage. Each multivariate metric needs 5 entries: name of metric, fixed, moving, weight, samplingParam. the list of lists should be of the form (("nameOfMetric2", img, img, weight, metricParam)). Another example would be (("MeanSquares", f2, m2, 0.5, 0

), (“CC”, f2, m2, 0.5, 2)) . This is only compatible

with the SyNOnly or antsRegistrationSyN* transformations.

- **restrict_transformation** (This option allows the user to restrict the) – optimization of the displacement field, translation, rigid or affine transform on a per-component basis. For example, if one wants to limit the deformation or rotation of 3-D volume to the first two dimensions, this is possible by specifying a weight vector of ‘(1,1,0)’ for a 3D deformation field or ‘(1,1,0,1,1,0)’ for a rigid transformation. Restriction currently only works if there are no preceding transformations.
- **smoothing_in_mm** (boolean ; currently only impacts low dimensional registration) –
- **singleprecision** (bool) – if True, use float32 for computations. This is useful for reducing memory usage for large datasets, at the cost of precision.
- **use_legacy_histogram_matching** (bool) – if True, use the original histogram matching in ANTs. This is not recommended, but is available for backwards compatibilty with earlier versions, where it was always turned on. The default is False. A better implementation of histogram matching is available in the ants.histogram_match_image2 function.
- **kwargs** (keyword args) – extra arguments

Returns

warpedmovout: Moving image warped to space of fixed image. *warpedfixout*: Fixed image warped to space of moving image. *fwdtransforms*: Transforms to move from moving to fixed image. *invtransforms*: Transforms to move from fixed to moving image.

Return type

dict containing follow key/value pairs

Notes

The output dict contains file names, designed to be used with ants.apply_transforms. As in ANTs, the forward affine transform .mat file is present in both the fwdtransforms and invtransforms lists. The matrix is inverted at run time by ants.apply_transforms when applying an inverse transform (see its whichtoinvert parameter).
type_of_transform can be one of:

- “Translation”: Translation transformation.
- “Rigid”: Rigid transformation: Only rotation and translation.
- “Similarity”: Similarity transformation: uniform scaling, rotation and translation.
- **“QuickRigid”: Rigid transformation: Only rotation and translation.**
  May be useful for quick visualization fixes.’
- **“DenseRigid”: Rigid transformation: Only rotation and translation.**
  Employs dense sampling during metric estimation.’
- **“BOLDRigid”: Rigid transformation: Parameters typical for BOLD to**
  BOLD intrasubject registration’.’
- “Affine”: Affine transformation: Rigid + scaling + shear (12 parameters).
- “AffineFast”: Fast version of Affine.
- **“BOLDAffine”: Affine transformation: Parameters typical for BOLD to**
  BOLD intrasubject registration’.’

- **"TRSAA": translation, rigid, similarity, affine (twice). please set**
  regIterations if using this option. this would be used in cases where you want a really high quality affine mapping (perhaps with mask).
- "Elastic": Elastic deformation: Affine + deformable.
- **"ElasticSyN": Symmetric normalization: Affine + deformable**
  transformation, with mutual information as optimization metric and elastic regularization.
- **"SyN": Symmetric normalization: Affine + deformable transformation,**
  with mutual information as optimization metric.
- **"SyNRA": Symmetric normalization: Rigid + Affine + deformable**
  transformation, with mutual information as optimization metric.
- **"SyNOnly": Symmetric normalization with no rigid or affine stages.**
  Uses mutual information as optimization metric.
- "SyNCC": SyN, but with cross-correlation as the metric.
- "SyNabp": SyN optimized for abpBrainExtraction.
- "SyNBold": SyN, but optimized for registrations between BOLD and T1 images.
- **"SyNBoldAff": SyN, but optimized for registrations between BOLD**
  and T1 images, with additional affine step.
- **"SyNAggro": SyN, but with more aggressive registration**
  (fine-scale matching and more deformation). Takes more time than SyN.
- "SyNLessAggro": Does exactly the same thing as "SyNAggro".
- **"TV[n]": time-varying diffeomorphism with where 'n' indicates number of**
  time points in velocity field discretization. The initial transform should be computed, if needed, in a separate call to ants.registration.
- "TVMSQ": time-varying diffeomorphism with mean square metric
- "TVMSQC": time-varying diffeomorphism with mean square metric for very large deformation
- **"antsRegistrationSyN[x]": recreation of the antsRegistrationSyN.sh script in ANTs**

  where 'x' is one of the transforms available:
  t: translation (1 stage) r: rigid (1 stage) a: rigid + affine (2 stages) s: rigid + affine + deformable syn (3 stages) sr: rigid + deformable syn (2 stages) so: deformable syn only (1 stage) b: rigid + affine + deformable b-spline syn (3 stages) br: rigid + deformable b-spline syn (2 stages) bo: deformable b-spline syn only (1 stage)
- **"antsRegistrationSyNQuick[x]": recreation of the antsRegistrationSyNQuick.sh script in ANTs.**
  x options as above.
- "antsRegistrationSyNRepro[x]": reproducible registration. x options as above.
- "antsRegistrationSyNQuickRepro[x]": quick reproducible registration. x options as above.

Example

```
>>> import ants
>>> fi = ants.image_read(ants.get_ants_data('r16'))
>>> mi = ants.image_read(ants.get_ants_data('r64'))
>>> fi = ants.resample_image(fi, (60,60), 1, 0)
>>> mi = ants.resample_image(mi, (60,60), 1, 0)
```

(continues on next page)

(continued from previous page)

```
>>> mytx = ants.registration(fixed=fi, moving=mi, type_of_transform = 'SyN' )
>>> mytx = ants.registration(fixed=fi, moving=mi, type_of_transform =
↪'antsRegistrationSyN[t]' )
>>> mytx = ants.registration(fixed=fi, moving=mi, type_of_transform =
↪'antsRegistrationSyN[b]' )
>>> mytx = ants.registration(fixed=fi, moving=mi, type_of_transform =
↪'antsRegistrationSyN[s]' )
```

8.1.14 ants.segmentation

Modules

atropos(a, x[, i, m, c, priorweight])	A finite mixture modeling (FMM) segmentation approach with possibilities for specifying prior constraints.
functional_lung_segmentation(image[, mask, ...])	Ventilation-based segmentation of hyperpolarized gas lung MRI.
fuzzy_spatial_cmeans_segmentation(image[, ...])	Fuzzy spatial c-means for image segmentation.
joint_label_fusion(target_image, ...[, ...])	A multiple atlas voting scheme to customize labels for a new subject.
kelly_kapowski(s, g, w[, its, r, m, ...])	Compute cortical thickness using the DiReCT algorithm.
kmeans	
otsu	
prior_based_segmentation(image, priors, mask)	Spatial prior-based image segmentation.

ants.segmentation.atropos

atropos(*a*, *x*, *i='Kmeans[3]'*, *m='[0.2,1x1]'*, *c='[5,0]'*, *priorweight=0.25*, ***kwargs*)

A finite mixture modeling (FMM) segmentation approach with possibilities for specifying prior constraints. These prior constraints include the specification of a prior label image, prior probability images (one for each class), and/or an MRF prior to enforce spatial smoothing of the labels. Similar algorithms include FAST and SPM. atropos can also perform multivariate segmentation if you pass a list of images in: e.g. a=(img1,img2).

ANTsR function: *atropos*

Parameters

- **a** (`ants.core.ANTsImage` or `list/tuple` of `ANTsImage types`) – One or more scalar images to segment. If priors are not used, the intensities of the first image are used to order the classes in the segmentation output, from lowest to highest intensity. Otherwise the order of the classes is dictated by the order of the prior images.
- **x** (`ants.core.ANTsImage`) – mask image.
- **i** (`str`) – initialization usually KMeans[N] for N classes or a list of N prior probability images. See Atropos in ANTs for full set of options.
- **m** (`str`) – mrf parameters as a string, usually "[smoothingFactor,radius]" where smoothingFactor determines the amount of smoothing and radius determines the MRF neighborhood, as an ANTs style neighborhood vector eg "1x1x1" for a 3D image. The radius must match the dimensionality of the image, eg 1x1 for 2D and The default in ANTs is smoothingFactor=0.3 and radius=1. See Atropos for more options.

- **c** (str) – convergence parameters, "[numberOfIterations,convergenceThreshold]". A threshold of 0 runs the full numberOfIterations, otherwise Atropos tests convergence by comparing the mean maximum posterior probability over the whole region of interest defined by the mask x.
- **priorweight** (scalar) – usually 0 (priors used for initialization only), 0.25 or 0.5.
- **kwargs** (keyword arguments) – more parameters, see Atropos help in ANTs

Returns

***segmentation*: ANTsImage**
actually segmented image

probabilityimages
[list of ANTsImage types] one image for each segmentation class

Return type
dictionary with the following key/value pairs

Example

```
>>> import ants
>>> img = ants.image_read(ants.get_ants_data('r16'))
>>> img = ants.resample_image(img, (64,64), 1, 0)
>>> mask = ants.get_mask(img)
>>> ants.atropos( a = img, m = '[0.2,1x1]', c = '[2,0]',  i = 'kmeans[3]', x = mask␣
↪)
>>> seg2 = ants.atropos( a = img, m = '[0.2,1x1]', c = '[2,0]', i = seg[
↪'probabilityimages'], x = mask, priorweight=0.25 )
```

ants.segmentation.functional_lung_segmentation

functional_lung_segmentation(*image*, *mask=None*, *number_of_iterations=2*, *number_of_atropos_iterations=5*, *mrf_parameters='[0.7,2x2x2]'*, *number_of_clusters=6*, *cluster_centers=None*, *bias_correction='n4'*, *verbose=True*)

Ventilation-based segmentation of hyperpolarized gas lung MRI.

Lung segmentation into classes based on ventilation as described in this paper:
https://pubmed.ncbi.nlm.nih.gov/21837781/

Parameters

- **image** (ANTs image) – Input proton-weighted MRI.
- **mask** (ANTs image) – Mask image designating the region to segment. 0/1 = background/foreground.
- **number_of_iterations** (int) – Number of Atropos <–> bias correction iterations (outer loop).
- **number_of_atropos_iterations** (int) – Number of Atropos iterations (inner loop). If number_of_atropos_iterations = 0, this is equivalent to K-means with no MRF priors.
- **mrf_parameters** (str) – Parameters for MRF in Atropos.
- **number_of_clusters** (int) – Number of tissue classes.
- **cluster_centers** (numpy.ndarray or tuple) – Initialization centers for k-means.

- **bias_correction** (str) – Apply "n3", "n4", or no bias correction (default = "n4").
- **verbose** (bool) – Print progress to the screen.

Returns

- Dictionary with segmentation image, probability images, and
- processed image.

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_data("mni")).resample_image((4,4,4))
>>> mask = image.get_mask()
>>> seg = ants.functional_lung_segmentation(image, mask, verbose=True,
                                            number_of_iterations=1,
                                            number_of_clusters=2,
                                            number_of_atropos_iterations=1)
```

ants.segmentation.fuzzy_spatial_cmeans_segmentation

fuzzy_spatial_cmeans_segmentation(*image*, *mask=None*, *number_of_clusters=4*, *m=2*, *p=1*, *q=1*, *radius=2*, *max_number_of_iterations=20*, *convergence_threshold=0.02*, *verbose=False*)

Fuzzy spatial c-means for image segmentation.

Image segmentation using fuzzy spatial c-means as described in

Chuang et al., Fuzzy c-means clustering with spatial information for image segmentation. CMIG: 30:9-15, 2006.

Parameters

- **image** (ants.core.ANTsImage) – Input image.
- **mask** (ants.core.ANTsImage) – Optional mask image.
- **number_of_clusters** (int) – Number of segmentation clusters.
- **m** (float) – Fuzziness parameter (default=2).
- **p** (float) – Membership importance parameter (default=1).
- **q** (float) – Spatial constraint importance parameter (default=1). q = 0 is equivalent to conventional fuzzy c-means.
- **radius** (int or tuple) – Neighborhood radius (scalar or array) for spatial constraint.
- **max_number_of_iterations** (int) – Iteration limit (default=20).
- **convergence_threshold** (float) – Convergence between iterations is measured using the Dice coefficient (default=0.02).
- **varbose** (bool) – Print progress.

Return type

dictionary containing ANTsImage and probability images

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> mask = ants.get_mask(image)
>>> fuzzy = ants.fuzzy_spatial_cmeans_segmentation(image, mask, number_of_
↪clusters=3)
```

ants.segmentation.joint_label_fusion

joint_label_fusion(*target_image*, *target_image_mask*, *atlas_list*, *beta=4*, *rad=2*, *label_list=None*, *rho=0.01*, *usecor=False*, *r_search=3*, *nonnegative=False*, *no_zeroes=False*, *max_lab_plus_one=False*, *output_prefix=None*, *verbose=False*)

A multiple atlas voting scheme to customize labels for a new subject. This function will also perform intensity fusion. It almost directly calls the C++ in the ANTs executable so is much faster than other variants in ANTsR.

One may want to normalize image intensities for each input image before passing to this function. If no labels are passed, we do intensity fusion. Note on computation time: the underlying C++ is multithreaded. You can control the number of threads by setting the environment variable ITK_GLOBAL_DEFAULT_NUMBER_OF_THREADS e.g. to use all or some of your CPUs. This will improve performance substantially. For instance, on a macbook pro from 2015, 8 cores improves speed by about 4x.

ANTsR function: *jointLabelFusion*

Parameters

- **target_image** (`ants.core.ANTsImage`) – image to be approximated
- **target_image_mask** (`ants.core.ANTsImage`) – mask with value 1
- **atlas_list** (`list` of `ANTsImage types`) – list containing intensity images
- **beta** (`scalar`) – weight sharpness, default to 2
- **rad** (`scalar`) – neighborhood radius, default to 2
- **label_list** (`list` of `ANTsImage types (optional)`) – list containing images with segmentation labels
- **rho** (`scalar`) – ridge penalty increases robustness to outliers but also makes image converge to average
- **usecor** (`bool`) – employ correlation as local similarity
- **r_search** (`scalar`) – radius of search, default is 3
- **nonnegative** (`bool`) – constrain weights to be non-negative
- **no_zeroes** (`bool`) – this will constrain the solution only to voxels that are always non-zero in the label list
- **max_lab_plus_one** (`bool`) – this will add max label plus one to the non-zero parts of each label where the target mask is greater than one. NOTE: this will have a side effect of adding to the original label images that are passed to the program. It also guarantees that every position in the labels have some label, rather than none. Ie it guarantees to explicitly parcellate the input data.
- **output_prefix** (`str`) – file prefix for storing output probabilityimages to disk
- **verbose** (`bool`) – whether to show status updates

Returns

segmentation
[ANTsImage] segmentation image

intensity
[ANTsImage] intensity image

probabilityimages
[list of ANTsImage types] probability map image for each label

segmentation_numbers
[list of numbers] segmentation label (number, int) for each probability map

Return type
`dictionary w/ following key/value pairs`

Example

```
>>> import ants
>>> ref = ants.image_read( ants.get_ants_data('r16'))
>>> ref = ants.resample_image(ref, (50,50),1,0)
>>> ref = ants.iMath(ref,'Normalize')
>>> mi = ants.image_read( ants.get_ants_data('r27'))
>>> mi2 = ants.image_read( ants.get_ants_data('r30'))
>>> mi3 = ants.image_read( ants.get_ants_data('r62'))
>>> mi4 = ants.image_read( ants.get_ants_data('r64'))
>>> mi5 = ants.image_read( ants.get_ants_data('r85'))
>>> refmask = ants.get_mask(ref)
>>> refmask = ants.iMath(refmask,'ME',2) # just to speed things up
>>> ilist = [mi,mi2,mi3,mi4,mi5]
>>> seglist = [None]*len(ilist)
>>> for i in range(len(ilist)):
>>>     ilist[i] = ants.iMath(ilist[i],'Normalize')
>>>     mytx = ants.registration(fixed=ref , moving=ilist[i] ,
>>>         type_of_transform = ('Affine') )
>>>     mywarpedimage = ants.apply_transforms(fixed=ref,moving=ilist[i],
>>>             transformlist=mytx['fwdtransforms'])
>>>     ilist[i] = mywarpedimage
>>>     seg = ants.threshold_image(ilist[i],'Otsu', 3)
>>>     seglist[i] = ( seg ) + ants.threshold_image( seg, 1, 3 ).morphology(␣
↪operation='dilate', radius=3 )
>>> r = 2
>>> pp = ants.joint_label_fusion(ref, refmask, ilist, r_search=2,
>>>                     label_list=seglist, rad=[r]*ref.dimension )
>>> pp = ants.joint_label_fusion(ref,refmask,ilist, r_search=2, rad=[r]*ref.
↪dimension)
```

ants.segmentation.kelly_kapowski

kelly_kapowski(*s*, *g*, *w*, *its=45*, *r=0.025*, *m=1.5*, *gm_label=2*, *wm_label=3*, ***kwargs*)

Compute cortical thickness using the DiReCT algorithm.

Diffeomorphic registration-based cortical thickness based on probabilistic segmentation of an image. This is an optimization algorithm.

Parameters

- **s** (`ANTsimage`) – segmentation image

- **g** (ants.core.ANTsImage) – gray matter probability image
- **w** (ants.core.ANTsImage) – white matter probability image
- **its** (int) – convergence params - controls iterations
- **r** (scalar) – gradient descent update parameter
- **m** (scalar) – gradient field smoothing parameter
- **gm_label** (int) – label for gray matter in the segmentation image
- **wm_label** (int) – label for white matter in the segmentation image
- **kwargs** (keyword arguments) – anything else, see KellyKapowski help in ANTs

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> img = ants.image_read( ants.get_ants_data('r16') ,2)
>>> img = ants.resample_image(img, (64,64),1,0)
>>> mask = ants.get_mask( img )
>>> segs = ants.kmeans_segmentation( img, k=3, kmask = mask)
>>> thick = ants.kelly_kapowski(s=segs['segmentation'], g=segs['probabilityimages
↪'][1],
                                w=segs['probabilityimages'][2], its=45,
                                r=0.5, m=1)
```

ants.segmentation.kmeans

Functions

kmeans_segmentation(image, k[, kmask, mrf])	K-means image segmentation that is a wrapper around *ants.atropos*

kmeans_segmentation(*image*, *k*, *kmask=None*, *mrf=0.1*)

K-means image segmentation that is a wrapper around *ants.atropos*

ANTsR function: *kmeansSegmentation*

Parameters

- **image** (ants.core.ANTsImage) – input image
- **k** (int) – integer number of classes
- **kmask** (ANTsImage (optional)) – segment inside this mask
- **mrf** (scalar) – smoothness, higher is smoother

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> fi = ants.image_read(ants.get_ants_data('r16'), 'float')
>>> fi = ants.n3_bias_field_correction(fi, 2)
>>> seg = ants.kmeans_segmentation(fi, 3)
```

ants.segmentation.otsu

Functions

otsu_segmentation(image, k[, mask])	Otsu image segmentation

otsu_segmentation(*image*, *k*, *mask=None*)

Otsu image segmentation

This is a very fast segmentation algorithm good for quick explortation, but does not return probability maps.

ANTsR function: *thresholdImage(image, 'Otsu', k)*

Parameters

- **image** (ants.core.ANTsImage) – input image
- **k** (int) – integer number of classes. Note that a background class will be added to this, so the resulting segmentation will have k+1 unique values.
- **mask** (ants.core.ANTsImage) – segment inside this mask

Return type

ants.core.ANTsImage

Example

```
>>> import ants
>>> mni = ants.image_read(ants.get_data('mni'))
>>> seg = mni.otsu_segmentation(k=3) #0=bg,1=csf,2=gm,3=wm
```

ants.segmentation.prior_based_segmentation

prior_based_segmentation(*image*, *priors*, *mask*, *priorweight=0.25*, *mrf=0.1*, *iterations=25*)

Spatial prior-based image segmentation.

Markov random field regularized, prior-based image segmentation that is a wrapper around atropos (see ANTs and related publications).

ANTsR function: *priorBasedSegmentation*

Parameters

- **image** (ants.core.ANTsImage or list/tuple of ANTsImage types) – input image or image list for multivariate segmentation
- **priors** (list/tuple of ANTsImage types) – list of priors that cover the number of classes
- **mask** (ants.core.ANTsImage) – segment inside this mask
- **prior_weight** (scalar) – usually 0 (priors used for initialization only), 0.25 or 0.5.
- **mrf** (scalar) – regularization, higher is smoother, a numerical value in range 0.0 to 0.2

- **iterations** (int) – maximum number of iterations. could be a large value eg 25.

Returns

***segmentation*: ANTsImage**
actually segmented image

probabilityimages
[list of ANTsImage types] one image for each segmentation class

Return type
dictionary with the following key/value pairs

Example

```
>>> import ants
>>> fi = ants.image_read(ants.get_ants_data('r16'))
>>> seg = ants.kmeans_segmentation(fi,3)
>>> mask = ants.threshold_image(seg['segmentation'], 1, 1e15)
>>> priorseg = ants.prior_based_segmentation(fi, seg['probabilityimages'], mask, 0.
↪25, 0.1, 3)
```

8.1.15 ants.utils

Modules

channels	
consistency	
convergence_monitoring(values[, window_size])	
get_ants_data([file_id, target_file_name, ...])	Get ANTsPy test data file
matrix_image	
mni2tal(xin)	mni2tal for converting from ch2/mni space to tal - very approximate.
ndimage_to_list(image)	Split a n dimensional ANTsImage into a list of n-1 dimensional ANTsImages
nibabel_nifti_to_ants	
nifti_utils	
polar_decomposition(X)	Perform a polar decomposition of a matrix.
scalar_rgb_vector	
sitk_to_ants	

ants.utils.channels

Functions

merge_channels(image_list[, channels_first])	Merge channels of multiple scalar ANTsImage types into one multi-channel ANTsImage
split_channels(image)	Split channels of a multi-channel ANTsImage into a collection of scalar ANTsImage types

merge_channels(*image_list*, *channels_first=False*)

Merge channels of multiple scalar ANTsImage types into one multi-channel ANTsImage

ANTsR function: *mergeChannels*

Parameters

image_list (list/tuple of ANTsImage types) – scalar images to merge

Return type

ants.core.ANTsImage

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> image2 = ants.image_read(ants.get_ants_data('r16'))
>>> image3 = ants.merge_channels([image,image2])
>>> image3 = ants.merge_channels([image,image2], channels_first=True)
>>> image3.numpy()
>>> image3.components == 2
```

split_channels(*image*)

Split channels of a multi-channel ANTsImage into a collection of scalar ANTsImage types

Parameters

image (ants.core.ANTsImage) – multi-channel image to split

Return type

list of ANTsImage types

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'), 'float')
>>> image2 = ants.image_read(ants.get_ants_data('r16'), 'float')
>>> imagemerge = ants.merge_channels([image,image2])
>>> imagemerge.components == 2
>>> images_unmerged = ants.split_channels(imagemerge)
>>> len(images_unmerged) == 2
>>> images_unmerged[0].components == 1
```

ants.utils.consistency

Functions

allclose(image1, image2)	Check if two images have the same array values
image_physical_space_consistency(image1, image2)	Check if two or more ANTsImage objects occupy the same physical space

allclose(*image1*, *image2*)

Check if two images have the same array values

image_physical_space_consistency(*image1*, *image2*, *tolerance=0.01*, *datatype=False*)

Check if two or more ANTsImage objects occupy the same physical space

ANTsR function: *antsImagePhysicalSpaceConsistency*

Parameters

- ***images** (ANTsImages) – images to compare
- **tolerance** (float) – tolerance when checking origin and spacing
- **data_type** (bool) – If true, also check that the image data types are the same

Returns

true if images share same physical space, false otherwise

Return type

bool

ants.utils.convergence_monitoring

convergence_monitoring(*values*, *window_size=10*)

ants.utils.get_ants_data

get_ants_data(*file_id=None*, *target_file_name=None*, *antsx_cache_directory=None*)

Get ANTsPy test data file

ANTsR function: *getANTsRData*

Parameters

name (str) – name of test image tag to retrieve Options:

- 'r16'
- 'r27'
- 'r30'
- 'r62'
- 'r64'
- 'r85'
- 'ch2'
- 'mni'
- 'surf'
- 'pcasl'

Returns

filepath of test image

Return type

str

Example

```
>>> import ants
>>> mnipath = ants.get_ants_data('mni')
```

ants.utils.matrix_image

Functions

image_list_to_matrix(image_list[, mask, ...])	Read images into rows of a matrix, given a mask - much faster for large datasets as it is based on C++ implementations.
images_from_matrix(data_matrix, mask)	Unmasks rows of a matrix and writes as images
images_to_matrix(image_list[, mask, sigma, ...])	Read images into rows of a matrix, given a mask - much faster for large datasets as it is based on C++ implementations.
matrix_from_images(image_list[, mask, ...])	Read images into rows of a matrix, given a mask - much faster for large datasets as it is based on C++ implementations.
matrix_to_images(data_matrix, mask)	Unmasks rows of a matrix and writes as images
matrix_to_timeseries(image, matrix[, mask])	converts a matrix to a ND image.
timeseries_to_matrix(image[, mask])	Convert a timeseries image into a matrix.

image_list_to_matrix(*image_list*, *mask=None*, *sigma=None*, *epsilon=0.5*)

Read images into rows of a matrix, given a mask - much faster for large datasets as it is based on C++ implementations.

ANTsR function: *imagesToMatrix*

Parameters

- **image_list** (list of ANTsImage types) – images to convert to ndarray
- **mask** (ANTsImage (optional)) – Mask image, voxels in the mask (>= epsilon) are placed in the matrix. If None, the first image in image_list is thresholded at its mean value to create a mask.
- **sigma** (scaler (optional)) – smoothing factor
- **epsilon** (scalar) – threshold for mask, values >= epsilon are included in the mask.

Returns

array with a row for each image shape = (N_IMAGES, N_VOXELS)

Return type

numpy.ndarray

Example

```
>>> import ants
>>> img = ants.image_read(ants.get_ants_data('r16'))
>>> img2 = ants.image_read(ants.get_ants_data('r16'))
>>> img3 = ants.image_read(ants.get_ants_data('r16'))
>>> mat = ants.image_list_to_matrix([img,img2,img3])
```

images_from_matrix(*data_matrix*, *mask*)

Unmasks rows of a matrix and writes as images

ANTsR function: *matrixToImages*

Parameters

- **data_matrix** (numpy.ndarray) – each row corresponds to an image array should have number of columns equal to non-zero voxels in the mask
- **mask** (ants.core.ANTsImage) – image containing a binary mask. Rows of the matrix are unmasked and written as images. The mask defines the output image space

Return type

list of ANTsImage types

Example

```
>>> import ants
>>> img = ants.image_read(ants.get_ants_data('r16'))
>>> msk = ants.get_mask( img )
>>> img2 = ants.image_read(ants.get_ants_data('r16'))
>>> img3 = ants.image_read(ants.get_ants_data('r16'))
>>> mat = ants.image_list_to_matrix([img,img2,img3], msk )
>>> ilist = ants.matrix_to_images( mat, msk )
```

images_to_matrix(*image_list*, *mask=None*, *sigma=None*, *epsilon=0.5*)

Read images into rows of a matrix, given a mask - much faster for large datasets as it is based on C++ implementations.

ANTsR function: *imagesToMatrix*

Parameters

- **image_list** (list of ANTsImage types) – images to convert to ndarray
- **mask** (ANTsImage (optional)) – Mask image, voxels in the mask (>= epsilon) are placed in the matrix. If None, the first image in image_list is thresholded at its mean value to create a mask.
- **sigma** (scaler (optional)) – smoothing factor
- **epsilon** (scalar) – threshold for mask, values >= epsilon are included in the mask.

Returns

array with a row for each image shape = (N_IMAGES, N_VOXELS)

Return type

numpy.ndarray

Example

```
>>> import ants
>>> img = ants.image_read(ants.get_ants_data('r16'))
>>> img2 = ants.image_read(ants.get_ants_data('r16'))
>>> img3 = ants.image_read(ants.get_ants_data('r16'))
>>> mat = ants.image_list_to_matrix([img,img2,img3])
```

matrix_from_images(*image_list*, *mask=None*, *sigma=None*, *epsilon=0.5*)

Read images into rows of a matrix, given a mask - much faster for large datasets as it is based on C++ implementations.

ANTsR function: *imagesToMatrix*

Parameters

- **image_list** (list of ANTsImage types) – images to convert to ndarray
- **mask** (ANTsImage (optional)) – Mask image, voxels in the mask (>= epsilon) are placed in the matrix. If None, the first image in image_list is thresholded at its mean value to create a mask.
- **sigma** (scaler (optional)) – smoothing factor
- **epsilon** (scalar) – threshold for mask, values >= epsilon are included in the mask.

Returns

array with a row for each image shape = (N_IMAGES, N_VOXELS)

Return type

numpy.ndarray

Example

```
>>> import ants
>>> img = ants.image_read(ants.get_ants_data('r16'))
>>> img2 = ants.image_read(ants.get_ants_data('r16'))
>>> img3 = ants.image_read(ants.get_ants_data('r16'))
>>> mat = ants.image_list_to_matrix([img,img2,img3])
```

matrix_to_images(*data_matrix*, *mask*)

Unmasks rows of a matrix and writes as images

ANTsR function: *matrixToImages*

Parameters

- **data_matrix** (numpy.ndarray) – each row corresponds to an image array should have number of columns equal to non-zero voxels in the mask
- **mask** (ants.core.ANTsImage) – image containing a binary mask. Rows of the matrix are unmasked and written as images. The mask defines the output image space

Return type

list of ANTsImage types

Example

```
>>> import ants
>>> img = ants.image_read(ants.get_ants_data('r16'))
>>> msk = ants.get_mask( img )
>>> img2 = ants.image_read(ants.get_ants_data('r16'))
>>> img3 = ants.image_read(ants.get_ants_data('r16'))
>>> mat = ants.image_list_to_matrix([img,img2,img3], msk )
>>> ilist = ants.matrix_to_images( mat, msk )
```

matrix_to_timeseries(*image*, *matrix*, *mask=None*)

converts a matrix to a ND image.

ANTsR function: *matrix2timeseries*

Parameters

- **image** (reference ND image) –

- **matrix** (matrix to convert to image) –
- **mask** (mask image defining voxels of interest) –

Return type
ants.core.ANTsImage

Example

```
>>> import ants
>>> img = ants.make_image( (10,10,10,5 ) )
>>> mask = ants.ndimage_to_list( img )[0] * 0
>>> mask[ 4:8, 4:8, 4:8 ] = 1
>>> mat = ants.timeseries_to_matrix( img, mask = mask )
>>> img2 = ants.matrix_to_timeseries( img,  mat, mask)
```

timeseries_to_matrix(*image*, *mask=None*)

Convert a timeseries image into a matrix.

ANTsR function: *timeseries2matrix*

Parameters

- **image** (image whose slices we convert to a matrix. E.g. a 3D image of size) – x by y by z will convert to a z by x*y sized matrix
- **mask** (ANTsImage (optional)) – image containing binary mask. voxels in the mask are placed in the matrix

Returns
array with a row for each image shape = (N_IMAGES, N_VOXELS)

Return type
numpy.ndarray

Example

```
>>> import ants
>>> img = ants.make_image( (10,10,10,5 ) )
>>> mat = ants.timeseries_to_matrix( img )
```

ants.utils.mni2tal

mni2tal(*xin*)

mni2tal for converting from ch2/mni space to tal - very approximate.

This is a standard approach but it's not very accurate.

ANTsR function: *mni2tal*

Parameters
xin (tuple) – point in mni152 space.

Return type
tuple

Example

```
>>> import ants
>>> ants.mni2tal( (10,12,14) )
```

References

http://bioimagesuite.yale.edu/mni2tal/501_95733_More%20Accurate%20Talairach%20Coordinates%20SLIDES.pdf
http://imaging.mrc-cbu.cam.ac.uk/imaging/MniTalairach

ants.utils.ndimage_to_list

ndimage_to_list(*image*)

Split a n dimensional ANTsImage into a list of n-1 dimensional ANTsImages

Parameters

image (ants.core.ANTsImage) – n-dimensional image to split

Return type

list of ANTsImage types

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> image2 = ants.image_read(ants.get_ants_data('r16'))
>>> imageTar = ants.make_image( ( *image2.shape, 2 ) )
>>> image3 = ants.list_to_ndimage( imageTar, [image,image2])
>>> image3.dimension == 3
>>> images_unmerged = ants.ndimage_to_list( image3 )
>>> len(images_unmerged) == 2
>>> images_unmerged[0].dimension == 2
```

ants.utils.nibabel_nifti_to_ants

Functions

from_nibabel_nifti(nib_image[, ...])	Converts a given nibabel Nifti image into an ANTsPy image.
to_nibabel_nifti(ants_image[, header])	Converts a given ANTsPy image into a nibabel Nifti image.

ants.utils.nifti_utils

Functions

deshear_affine_transform_matrix(A[, ...])	Deshear an affine transform by removing shear components.
deshear_nifti_sform(metadata[, ...])	Deshear an sform matrix in the provided metadata dict.
get_nifti_qform_spatial_info(metadata)	Extract qform-derived spacing, origin, direction.
get_nifti_sform_shear(metadata)	Get the shear parameters from the sform matrix in metadata dictionary.
get_nifti_sform_spatial_info(metadata[, ...])	Extract sform-derived spacing, origin, direction.

continues on next page

Table 42 – continued from previous page

get_nifti_spatial_transform_from_metadata(...)	Return a dict containing origin/spacing/direction in ITK (LPS) coordinates, derived from NIfTI header metadata.
set_nifti_spatial_transform_from_metadata(...)	Set the spatial transform of an ANTsImage from NIfTI header metadata.

get_nifti_qform_spatial_info(*metadata*)

Extract qform-derived spacing, origin, direction. Uses the 4x4 'qto_xyz' from the metadata dict. This is the rotation matrix derived from quaternion parameters in the NIfTI header, multiplied by the pixdim scales and with the qoffset translation.

Note: output is in ITK LPS coordinates

Returns

pixdim_spacing = [sx, sy, sz] (from pixdim[1..3]) transform_spacing : [sx, sy, sz] origin : [ox, oy, oz] direction : 3x3 list (direction cosines)

Return type

`dict with keys`

Example

```
>>> import ants
>>> metadata = ants.read_image_metadata( ants.get_ants_data('mni') )
>>> ants.get_nifti_qform_spatial_info(metadata)
```

get_nifti_sform_shear(*metadata*)

Get the shear parameters from the sform matrix in metadata dictionary.

Return type

`[shear_xy, shear_xz, shear_yz] (cosines between axes; 0 means orthogonal)`

Example

```
>>> import ants
>>> metadata = ants.read_image_metadata( ants.get_ants_data('mni') )
>>> ants.get_nifti_sform_shear(metadata)
```

get_nifti_sform_spatial_info(*metadata*, *deshear_threshold=1e-06*, *max_angle_deviation=0.5*)

Extract sform-derived spacing, origin, direction.

Note: output is in ITK LPS coordinates.

Parameters

- **metadata** (`dict`) – The NIfTI header metadata dictionary.
- **deshear_threshold** (`float`) – Shear threshold for deshearing sform, if the shear is beneath this value, the sform is not modified.
- **max_angle_deviation** (`float`) – Maximum angle deviation for directions after deshearing sform. If the desheared directions deviate from the original directions by more than this value (in degrees), deshearing fails and the return value is None.

Returns

- `dict with keys` – pixdim_spacing = [sx, sy, sz] (from pixdim[1..3]) transform_spacing : [sx, sy, sz] origin : [ox, oy, oz] direction : 3x3 list (direction cosines) desheared : bool original_shear : [shear_xy, shear_xz, shear_yz]

- Returns None if no sform is present or if deshearing fails.

Example

```
>>> import ants
>>> metadata = ants.read_image_metadata( ants.get_ants_data('mni') )
>>> ants.get_nifti_sform_spatial_info(metadata)
```

get_nifti_spatial_transform_from_metadata(*metadata*, *prefer_sform=True*, *deshear_threshold=1e-06*, *max_angle_deviation=0.5*, *verbose=False*)

Return a dict containing origin/spacing/direction in ITK (LPS) coordinates, derived from NIfTI header metadata.

This function only returns the 3D spatial transform information, including spacing for the first three dimensions. It does not modify time spacing or direction for 4D images.

Note that the spacing is always taken from the pixdim fields in the NIfTI header, as is standard in ITK. The transforms are checked for consistency with the pixdim spacing, if they are not the same, it indicates that the transform is something other than a reorientation to the native physical space.

If prefer_sform is True (default), the sform transform is used if it is present (sform_code > 0) and either has shear beneath the threshold, or the shear can be removed without exceeding the max_axis_deviation.

If prefer_sform is False, qform is used if it is present (qform_code > 0), otherwise sform is used if present and usable.

Parameters

- **image** (ants.ANTsImage) – The image to modify.
- **metadata** (dict) – The NIfTI header metadata dictionary.
- **prefer_sform** (bool) – Whether to prefer sform over qform if both are available.
- **deshear_threshold** (float) – Shear threshold for deshearing sform.
- **max_angle_deviation** (float) – Maximum angle deviation for deshearing sform.
- **verbose** (bool) – Whether to print verbose output.

Returns

- { – “origin”: [ox, oy, oz], “pixdim_spacing”: [sx, sy, sz], # from the pixdims “transform_spacing”: [sx, sy, sz], # from the transform “direction”: [[...],[...],[...]], “transform_source”: “sform” | “qform”
- }

Example

```
>>> import ants
>>> metadata = ants.read_image_metadata( ants.get_ants_data('mni') )
>>> ants.get_nifti_spatial_transform_from_metadata(metadata)
```

set_nifti_spatial_transform_from_metadata(*image*, *metadata*, *prefer_sform=True*, *deshear_threshold=1e-06*, *max_angle_deviation=0.5*, *verbose=False*)

Set the spatial transform of an ANTsImage from NIfTI header metadata. This sets the 3D spatial transform but does not modify spacing or the fourth row / column of the direction matrix for 4D images. This function does not support 2D images, because the projection of a 3D spatial transform to 2D is not preserved in the ANTsImage.

The spacing is not modified but it is checked for consistency with the pixdim fields in the NIfTI header, as is standard in ITK. If the spacing does not match the pixdim, an error is raised.

Parameters

- **image** (ants.ANTsImage) – The image to modify.
- **metadata** (dict) – The NIfTI header metadata dictionary.
- **prefer_sform** (bool) – Whether to prefer sform over qform if both are available.
- **deshear_threshold** (float) – Shear threshold for deshearing sform, shear below this will be ignored.
- **max_angle_deviation** (float) – Maximum angle deviation for directions after deshearing sform. If the desheared directions deviate from the original directions by more than this value (in degrees), deshearing fails, and qform is used if available.
- **verbose** (bool) – Whether to print verbose output.

Returns

The modified image with updated spatial transform.

Return type

ants.ANTsImage

Example

```
>>> import ants
>>> img = ants.image_read( ants.get_ants_data('mni') )
>>> metadata = ants.read_image_metadata( ants.get_ants_data('mni') )
>>> ants.set_nifti_spatial_transform_from_metadata(img, metadata)
```

ants.utils.polar_decomposition

polar_decomposition(*X*)

Perform a polar decomposition of a matrix.

The polar decomposition of a matrix X is a factorization of the form:

X = P @ Z

where P is an orthogonal projection matrix and Z is an orthogonal matrix.

Parameters

X (numpy.ndarray) – Input matrix to decompose. Should be a square matrix.

Returns

A dictionary containing: - "P": the orthogonal projection matrix. - "Z": the orthogonal part of the decomposition. - "Xtilde": the reconstructed matrix from the decomposition (Xtilde = P @ Z).

Return type

dict

ants.utils.scalar_rgb_vector

Functions

rgb_to_vector(image)	Convert an RGB ANTsImage to a Vector ANTsImage

continues on next page

Table 43 – continued from previous page

scalar_to_rgb(image[, mask, filename, cmap, ...])	Usage: ConvertScalarImageToRGB imageDimension inputImage outputImage mask colormap [customColormapFile] [minimumInput] [maximumInput] [minimumRGBOutput=0] [maximumRGBOutput=255] <vtkLookupTable> Possible colormaps: grey, red, green, blue, copper, jet, hsv, spring, summer, autumn, winter, hot, cool, overunder, custom
vector_to_rgb(image)	Convert an Vector ANTsImage to a RGB ANTsImage

rgb_to_vector(*image*)

Convert an RGB ANTsImage to a Vector ANTsImage

Parameters

image (ants.core.ANTsImage) – RGB image to be converted

Return type

ants.core.ANTsImage

Example

```
>>> import ants
>>> mni = ants.image_read(ants.get_data('mni'))
>>> mni_rgb = ants.scalar_to_rgb(mni)
>>> mni_vector = mni.rgb_to_vector()
>>> mni_rgb2 = mni.vector_to_rgb()
```

scalar_to_rgb(*image*, *mask=None*, *filename=None*, *cmap='red'*, *custom_colormap_file=None*, *min_input=None*, *max_input=None*, *min_rgb_output=None*, *max_rgb_output=None*, *vtk_lookup_table=None*)

Usage: ConvertScalarImageToRGB imageDimension inputImage outputImage mask colormap [customColormapFile] [minimumInput] [maximumInput] [minimumRGBOutput=0] [maximumRGBOutput=255] <vtkLookupTable> Possible colormaps: grey, red, green, blue, copper, jet, hsv, spring, summer, autumn, winter, hot, cool, overunder, custom

Example

```
>>> import ants
>>> img = ants.image_read(ants.get_data('r16'))
>>> img_color = ants.scalar_to_rgb(img, cmap='jet')
```

vector_to_rgb(*image*)

Convert an Vector ANTsImage to a RGB ANTsImage

Parameters

image (ants.core.ANTsImage) – RGB image to be converted

Return type

ants.core.ANTsImage

Example

```
>>> import ants
>>> img = ants.image_read(ants.get_data('r16'), pixeltype='unsigned char')
>>> img_rgb = ants.scalar_to_rgb(img.clone())
>>> img_vec = img_rgb.rgb_to_vector()
>>> img_rgb2 = img_vec.vector_to_rgb()
```

ants.utils.sitk_to_ants

Functions

from_sitk(sitk_image)	Converts a SimpleITK image into an ANTsPy image.
to_sitk(ants_image)	Converts an ANTsPy image into a SimpleITK image.

from_sitk(*sitk_image: SimpleITK.Image*) → *ANTsImage*

Converts a SimpleITK image into an ANTsPy image. Requires SimpleITK.

Parameters

sitk_image (SimpleITK.Image) – The SimpleITK image to convert.

Returns

ants_image – The converted ANTsPy image.

Return type

ants.core.ANTsImage

Raises

ValueError – If the image dimensionality is unsupported.

Examples

```
>>> import SimpleITK as sitk
>>> import ants
>>> img = sitk.ReadImage('brain.nii.gz')
>>> ants_img = from_sitk(img)
```

to_sitk(*ants_image:* ANTsImage) → SimpleITK.Image

Converts an ANTsPy image into a SimpleITK image. Requires SimpleITK.

Parameters

ants_image (ants.core.ANTsImage) – The ANTsPy image to convert.

Returns

sitk_image – The converted SimpleITK image.

Return type

SimpleITK.Image

Raises

ValueError – If the image dimensionality is unsupported.

Examples

```
>>> import ants
>>> ants_img = ants.image_read('brain.nii.gz')
>>> sitk_img = ants.to_sitk(ants_img)
```

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex

PYTHON MODULE INDEX

a

INDEX

A

B

C

D

E

F

G

H

I

J

K

L

M

N

T

U

V

W

Z