ANTsPy

Release master

Nov 20, 2017

Package Reference

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

CHAPTER 1

Core

1.1 Images

1.1.1 ANTsImage

class `ants.core.ants_image.`**`ANTsImage`**(*pixeltype='float'*, *dimension=3*, *components=1*, *pointer=None*, *is_rgb=False*, *label_image=None*)

`abs`(*axis=None*)
Return absolute value of image

`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 *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 **`dtype`** (*`string (optional)`*) – if None, the dtype 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’

Returns

Return type *ANTsImage*

`copy`(*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 **`dtype`** (*`string (optional)`*) – if None, the dtype 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’

Returns

Return type *ANTsImage*

`direction`
Get image direction

Returns

Return type tuple

`flatten`()
Flatten image data

`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

`new_image_like`(*data*)
Create a new ANTsImage with the same header information, but with a new image array.

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

Returns

Return type *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)

Returns

Return type ndarray

origin
Get image origin

Returns

Return type tuple

range(*axis=None*)
Return range tuple along specified axis

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

Returns

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

Returns

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

Returns

Return type None

spacing
Get image spacing

Returns

Return type tuple

std(*axis=None*)
Return std along specified axis

sum(*axis=None*, *keepdims=False*)
Return sum along specified axis

to_file (*filename*)
Write the ANTsImage to file

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

to_filename (*filename*)
Write the ANTsImage to file

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

unique (*sort=False*)
Return unique set of values in image

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)

Returns

Return type ndarray

1.1.2 ANTsImage IO

ants.**image_clone** (*image*, *pixeltype=None*)
Clone an ANTsImage

ANTsR function: *antsImageClone*

Parameters

- **image** (ANTsImage) – image to clone
- **dtype** (*string (optional)*) – new datatype for image

Returns

Return type *ANTsImage*

ants.**image_header_info** (*filename*)
Read file info from image header

ANTsR function: *antsImageHeaderInfo*

Parameters **filename** (*string*) – name of image file from which info will be read

Returns

Return type dict

ants.**image_read** (*filename*, *dimension=None*, *pixeltype='float'*, *reorient=False*)
Read an ANTsImage from file

ANTsR function: *antsImageRead*

Parameters

- **filename** (*string*) – 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** (*string*) – 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
- **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

Returns

Return type *ANTsImage*

`ants.image_write`(*image*, *filename*, *ri=False*)

Write an ANTsImage to file

ANTsR function: *antsImageWrite*

Parameters

- **image** (*ANTsImage*) – image to save to file
- **filename** (*string*) – name of file to which image will be saved
- **ri** (*boolean*) –

  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

`ants.make_image`(*imagesize*, *voxval=0*, *spacing=None*, *origin=None*, *direction=None*, *has_components=False*, *pixeltype='float'*)

Make an image with given size and voxel value or given a mask and vector

ANTsR function: *makeImage*

Parameters

- **shape** (*tuple/ANTsImage*) – input image size or mask
- **voxval** (*scalar*) – input image value or vector, size of mask
- **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** (*boolean*) – whether there are components per pixel or not
- **pixeltype** (*float*) – data type of image values

Returns

Return type *ANTsImage*

`ants.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** (*ndarray*) – image data array
- **origin** (*tuple/list*) – image origin
- **spacing** (*tuple/list*) – image spacing
- **direction** (*list/ndarray*) – image direction
- **has_components** (*boolean*) – whether the image has components

Returns image with given data and any given information

Return type *ANTsImage*

ants.**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** (ANTsImage) – image containing a binary mask. Rows of the matrix are unmasked and written as images. The mask defines the output image space

Returns

Return type list of ANTsImage types

ants.**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** (ANTsImage) – image containing a binary mask. Rows of the matrix are unmasked and written as images. The mask defines the output image space

Returns

Return type list of ANTsImage types

ants.**image_list_to_matrix**(*image_list*, *mask=None*, *sigma=None*, *epsilon=0*)

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 python:types*) – images to convert to ndarray
- **mask** (ANTsImage *(optional)*) – image containing binary mask. voxels in the mask are placed in the matrix
- **sigma** (*scaler (optional)*) – smoothing factor
- **epsilon** (*scalar*) – threshold for mask

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

Return type 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])
```

`ants.`**`images_to_matrix`**(*image_list*, *mask=None*, *sigma=None*, *epsilon=0*)

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 python:types*) – images to convert to ndarray
- **mask** (ANTsImage *(optional)*) – image containing binary mask. voxels in the mask are placed in the matrix
- **sigma** (*scaler (optional)*) – smoothing factor
- **epsilon** (*scalar*) – threshold for mask

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

Return type 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])
```

`ants.`**`matrix_from_images`**(*image_list*, *mask=None*, *sigma=None*, *epsilon=0*)

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 python:types*) – images to convert to ndarray
- **mask** (ANTsImage *(optional)*) – image containing binary mask. voxels in the mask are placed in the matrix
- **sigma** (*scaler (optional)*) – smoothing factor
- **epsilon** (*scalar*) – threshold for mask

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

Return type 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 `ants.core.ants_transform.`**`ANTsTransform`**(*precision='float'*, *dimension=3*, *transform_type='AffineTransform'*, *pointer=None*)

`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`** (`ANTsImage`) – image to which the transform will be applied
- **`reference`** (`ANTsImage`) – target space for transforming image
- **`interpolation`** (*`string`*) – type of interpolation to use

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

fixed_parameters
Get parameters of transform

invert()
Invert the transform

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

ants.**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** (*string*) – type of transform(s)
- **precision** (*string*) – numerical precision
- **dimension** (*integer*) – spatial dimension of transform
- **matrix** (*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** (*ANTsImage*) – multichannel ANTsImage for non-linear transform
- **supported_types** (*boolean*) – flag that returns array of possible transforms types

Returns

Return type *ANTsTransform* or list of ANTsTransform types

Example

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

```
```

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

Create a new ANTsTransform

ANTsR function: None

Example

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

ants.**read_transform**(*filename*, *dimension=2*, *precision='float'*)

Read a transform from file

ANTsR function: *readAntsrTransform*

Parameters

- **filename** (*string*) – filename of transform
- **dimension** (*integer*) – spatial dimension of transform
- **precision** (*string*) – numerical precision of transform

Returns

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')
```

ants.**write_transform**(*transform*, *filename*)

Write ANTsTransform to file

ANTsR function: *writeAntsrTransform*

Parameters

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

Returns

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')
```

ants.**transform_from_displacement_field**(*field*)
Convert deformation field (multiChannel image) to ANTsTransform

ANTsR function: *antsrTransformFromDisplacementField*

Parameters **field** (ANTsImage) – deformation field as multi-channel ANTsImage

Returns

Return type *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') )
>>> compfield = ants.compose_transforms_to_field( fi, mytx['fwd'] )
>>> atx = ants.transform_from_displacement_field( compfield )
```

1.3 Metrics

1.3.1 ANTsMetric

class ants.core.ants_metric.**ANTsImageToImageMetric**(*metric*)
ANTsImageToImageMetric class

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

1.3.2 ANTsMetric IO

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

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

Parameters **metric_type** (*string*) – 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)
```

ants.**supported_metrics**()

CHAPTER 2

Registration

`ants.`**`registration`**(*fixed*, *moving*, *type_of_transform='SyN'*, *initial_transform=None*, *outprefix=''*, *mask=None*, *grad_step=0.2*, *flow_sigma=3*, *total_sigma=0*, *aff_metric='mattes'*, *aff_sampling=32*, *syn_metric='mattes'*, *syn_sampling=32*, *reg_iterations=(40, 20, 0)*, *verbose=False*, ***kwargs*)

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

ANTsR function: *antsRegistration*

Parameters

- **`fixed`** (`ANTsImage`) – fixed image to which we register the moving image.
- **`moving`** (`ANTsImage`) – moving image to be mapped to fixed space.
- **`type_of_transform`** (*`string`*) – 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
- **`outprefix`** (*`string`*) – output will be named with this prefix.
- **`mask`** (`ANTsImage` *`(optional)`*) – mask the registration.
- **`grad_step`** (*`scalar`*) – gradient step size (not for all tx)
- **`flow_sigma`** (*`scalar`*) – smoothing for update field
- **`total_sigma`** (*`scalar`*) – smoothing for total field
- **`aff_metric`** (*`string`*) – the metric for the affine part (GC, mattes, meansquares)
- **`aff_sampling`** (*`scalar`*) – the nbins or radius parameter for the syn metric
- **`syn_metric`** (*`string`*) – 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 python:integers`*) – vector of iterations for syn. we will set the smoothing and multi-resolution parameters based on the length of this vector.

- **verbose** (*boolean*) – request verbose output (useful for debugging)
- **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

typeofTransform can be one of:

- "Translation": Translation transformation.
- "Rigid": Rigid transformation: Only rotation and translation.
- "Similarity": Similarity transformation: 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.
- "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).
- **"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: no initial transformation,** with mutual information as optimization metric. Assumes images are aligned by an inital transformation. Can be useful if you want to run an unmasked affine followed by masked deformable registration.
- "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.

- "TVMSQ": time-varying diffeomorphism with mean square metric
- "TVMSQC": time-varying diffeomorphism with mean square metric for very large deformation

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' )
```

`ants.`**`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 arapper for the ANTs function antsAffineInitializer.

ANTsR function: *affineInitializer*

Parameters

- **fixed_image** (`ANTsImage`) – the fixed reference image
- **moving_image** (`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** (*boolean*) – 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 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)
```

`ants.`**`apply_transforms`**(*fixed*, *moving*, *transformlist*, *interpolator='linear'*, *imagetype=0*, *whichtoinvert=None*, *compose=None*, *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** (ANTsImage) – fixed image defining domain into which the moving image is transformed.
- **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** (*string*) –

  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** (*integer*) – choose 0/1/2/3 mapping to scalar/vector/tensor/time-series
- **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.
- **verbose** (*boolean*) – print command and run verbose application of transform.
- **kwargs** (*keyword arguments*) – extra parameters

Returns

Return type *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'] )
```

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

Returns

Return type *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)
>>> 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)
```

ants.**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** (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

Returns

Return type *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( mi, grid_directions=(False,True),
                        transform=mytx['fwdtransforms'], fixed_reference_image=fi␣
↪)
```

ants.**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** (ANTsImage) – target image
- **moving** (ANTsImage) – moving image

Returns

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)
```

`ants.`**`image_mutual_information`**(*image1*, *image2*)

Compute mutual information between two ANTsImage types

ANTsR function: *antsImageMutualInformation*

Parameters

- **image1** (ANTsImage) – image 1
- **image2** (ANTsImage) – image 2

Returns

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

Reflect an image along an axis

ANTsR function: *reflectImage*

Parameters

- **image** (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** (*string*) – similarity metric for image registration. see antsRegistration.

Returns

Return type *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.**reorient_image**(*image*, *axis1*, *axis2=None*, *doreflection=False*, *doscale=0*, *txfn=None*)

Align image along a specified axis

ANTsR function: *reorientImage*

Parameters

- **image** (ANTsImage) – image to reorient
- **axis1** (*list/tuple of python:integers*) – vector of size dim, might need to play w/axis sign
- **axis2** (*list/tuple of python:integers*) – vector of size dim for 3D
- **doreflection** (*boolean*) – whether to reflect
- **doscale** (*scalar value*) – 1 allows automated estimate of scaling
- **txfn** (*string*) – file name for transformation

Returns

Return type *ANTsImage*

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'))
>>> ants.reorient_image(image, (1,0))
```

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

Returns

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 )
```

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

Returns

Return type *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)
```

ants.**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** (ANTsImage) – image to resample
- **target** (ANTsImage) – image of reference, the output will be in this space
- **interp_type** (*string*) –

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

Returns

Return type *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 3

Segmentation

`ants.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** (*ANTsImage or list/tuple of ANTsImage python: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** (*ANTsImage*) – mask image.
- **i** (*string*) – 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** (*string*) – 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** (*string*) – 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 )
```

ants.**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*, *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** (`ANTsImage`) – image to be approximated
- **target_image_mask** (`ANTsImage`) – mask with value 1
- **atlas_list** (*list of ANTsImage python:types*) – list containing intensity images
- **beta** (*scalar*) – weight sharpness, default to 2
- **rad** (*scalar*) – neighborhood radius, default to 2
- **label_list** (*list of ANTsImage python:types (optional)*) – list containing images with segmentation labels
- **rho** (*scalar*) – ridge penalty increases robustness to outliers but also makes image converge to average
- **usecor** (*boolean*) – employ correlation as local similarity
- **r_search** (*scalar*) – radius of search, default is 3
- **nonnegative** (*boolean*) – constrain weights to be non-negative
- **verbose** (*boolean*) – 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

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] ,
>>>         typeofTransform = ('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
>>> 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.**kelly_kapowski**(*s, g, w, its=50, r=0.025, m=1.5, **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** (*ANTsImage*) – gray matter probability image
- **w** (*ANTsImage*) – white matter probability image
- **its** (*integer*) – convergence params - controls iterations
- **r** (*scalar*) – gradient descent update parameter
- **m** (*scalar*) – gradient field smoothing parameter
- **kwargs** (*keyword arguments*) – anything else, see KellyKapowski help in ANTs

Returns

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

Returns

Return type *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.**label_geometry_measures**(*label_image*, *intensity_image=None*)

Wrapper for the ANTs funtion labelGeometryMeasures

ANTsR function: *labelGeometryMeasures*

Parameters

- **label_image** (ANTsImage) – image on which to compute geometry
- **intensity_image** (ANTsImage *(optional)*) – image with intensity values

Returns

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.**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** (*ANTsImage or list/tuple of ANTsImage python:types*) – input image or image list for multivariate segmentation
- **priors** (*list/tuple of ANTsImage python:types*) – list of priors that cover the number of classes
- **mask** (*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** (*integer*) – 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 4

Statistical Learning

`ants.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** (*2-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** (*2-tuple of ANTsImage python: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** (*integer*) – number of eigenvector pairs
- **its** (*integer*) – number of iterations, 10 or 20 usually sufficient
- **cthresh** (*2-tuple*) – cluster threshold pair
- **statdir** (*string (optional)*) – temporary directory if you want to look at full output
- **perms** (*integer*) – 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** (*boolean*) – 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** (*boolean*) – rank transform input matrices
- **mycoption** (*integer*) – 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** (*boolean*) – activates verbose output to screen
- **rejector** (*scalar*) – rejects small correlation solutions
- **max_based** (*boolean*) – 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)
```

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

Returns

Return type *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)
```

ants.**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 ANTsImage*) – input matrix where rows provide initial vector values. alternatively, this can be an antsImage which contains labeled regions.
- **mask** (*ANTsImage*) – mask if available
- **initlabels** (*list/tuple of python:integers*) – which labels in initmat to use as initial components
- **nreps** (*integer*) – 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 5

Utilities

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

N3 Bias Field Correction

ANTsR function: *n3BiasFieldCorrection*

Parameters

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

Returns

Return type *ANTsImage*

Example

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

ants.**n4_bias_field_correction**(*image, mask=None, shrink_factor=4, convergence={'iters': [50, 50, 50, 50], 'tol': 1e-07}, spline_param=200, verbose=False, weight_mask=None*)

N4 Bias Field Correction

ANTsR function: *n4BiasFieldCorrection*

Parameters

- **image** (ANTsImage) – image to bias correct
- **mask** (ANTsImage) – input mask, if one is not passed one will be made
- **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 shrinkage factor tol : the convergence tolerance.
- **spline_param** (*integer*) – Parameter controlling number of control points in spline. Either single value, indicating how many control points, or vector with one entry per dimension of image, indicating the spacing in each direction.
- **verbose** (*boolean*) – enables verbose output.
- **weight_mask** (ANTsImage *(optional)*) – antsImage of weight mask

Returns

Return type *ANTsImage*

Example

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

ants.**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** (ANTsImage) – image to correct and truncate
- **intensity_truncation** (*3-tuple*) – quantiles for intensity truncation
- **mask** (ANTsImage *(optional)*) – mask for bias correction
- **usen3** (*boolean*) – if True, use N3 bias correction instead of N4

Returns

Return type *ANTsImage*

Example

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

ants.**merge_channels**(*image_list*)

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

ANTsR function: *mergeChannels*

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

Returns

Return type *ANTsImage*

Example

```
>>> import ants
>>> image = ants.image_read(ants.get_ants_data('r16'), 'float')
>>> image2 = ants.image_read(ants.get_ants_data('r16'), 'float')
>>> image3 = ants.merge_channels([image,image2])
>>> image3.components == 2
```

`ants.`**`split_channels`**(*image*)

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

Parameters **`image`** (`ANTsImage`) – multi-channel image to split

Returns

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

Returns

Return type *ANTsImage*

Example

```
>>> import ants
>>> fi = ants.image_read( ants.get_ants_data('r16') )
>>> cropped = ants.crop_image(fi)
>>> cropped = ants.crop_image(fi, fi, 100 )
```

`ants.`**`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** (ANTsImage) – image to crop
- **lowerind** (*list/tuple of python:integers*) – vector of lower index, should be length image dimensionality
- **upperind** (*list/tuple of python:integers*) – vector of upper index, should be length image dimensionality

Returns

Return type *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 )
```

ants.**decrop_image**(*cropped_image*, *full_image*)

The inverse function for *ants.crop_image*

ANTsR function: *decropImage*

Parameters

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

Returns

Return type *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)
```

ants.**denoise_image**(*image*, *mask=None*, *shrink_factor=1*, *p=1*, *r=3*, *noise_model='Rician'*)

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** (ANTsImage) – scalar image to denoise.
- **mask** (ANTsImage) – to limit the denoise region.

- **shrink_factor** (*scalar*) – downsampling level performed within the algorithm.
- **p** (*integer*) – patch radius for local sample.
- **r** (*integer*) – search radius from which to choose extra local samples.
- **noise_model** (*string*) – 'Rician' or 'Gaussian'

Returns

Return type *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))
```

ants.**get_ants_data**(*name=None*)

Get ANTsPy test data filename

ANTsR function: *getANTsRData*

Parameters **name** (*string*) – name of test image tag to retrieve Options:

- 'r16'
- 'r27'
- 'r64'
- 'r85'
- 'ch2'
- 'mni'
- 'surf'

Returns filepath of test image

Return type string

Example

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

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

Returns

Return type 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.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** (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** (*integer*) – 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

Returns

Return type *ANTsImage*

Example

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

`ants.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

ANTsR function: *imageSimilarity*

Parameters

- **fixed** (ANTsImage) – the fixed image

- **moving** (ANTsImage) – the moving image
- **metric_type** (*string*) –

  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

Returns

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

Returns

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.**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** (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.**impute**(*data*, *method='mean'*, *value=None*, *nan_value=nan*)
Impute missing values on a numpy ndarray in a column-wise manner.

ANTsR function: *antsrimpute*

Parameters

- **data** (*numpy.ndarray*) – data to impute
- **method** (*string or float*) – type of imputation method to use Options:

  mean median constant KNN BiScaler NuclearNormMinimization SoftImpute IterativeSVD

- **value** (*scalar (optional)*) – optional arguments for different methods if method == 'constant'

  constant value

  if method == 'KNN' number of nearest neighbors to use

- **nan_value** (*scalar*) – value which is interpreted as a missing value

Returns

- *ndarray if ndarray was given*
- *OR*
- *pd.DataFrame if pd.DataFrame was given*

Example

```
>>> import ants
>>> import numpy as np
>>> data = np.random.randn(4,10)
>>> data[2,3] = np.nan
>>> data[3,5] = np.nan
>>> data_imputed = ants.impute(data, 'mean')
```

KNN: Nearest neighbor imputations which weights samples using the mean squared difference on features for which two rows both have observed data.

SoftImpute: Matrix completion by iterative soft thresholding of SVD decompositions. Inspired by the softImpute package for R, which is based on Spectral Regularization Algorithms for Learning Large Incomplete Matrices by Mazumder et. al.

IterativeSVD: Matrix completion by iterative low-rank SVD decomposition. Should be similar to SVDimpute from Missing value estimation methods for DNA microarrays by Troyanskaya et. al.

MICE: Reimplementation of Multiple Imputation by Chained Equations.

MatrixFactorization: Direct factorization of the incomplete matrix into low-rank U and V, with an L1 sparsity penalty on the elements of U and an L2 penalty on the elements of V. Solved by gradient descent.

NuclearNormMinimization: Simple implementation of Exact Matrix Completion via Convex Optimization by Emmanuel Candes and Benjamin Recht using cvxpy. Too slow for large matrices.

BiScaler: Iterative estimation of row/column means and standard deviations to get doubly normalized matrix. Not guaranteed to converge but works well in practice. Taken from Matrix Completion and Low-Rank SVD via Fast Alternating Least Squares.

`ants.`**`invariant_image_similarity`**(*image1*, *image2*, *local_search_iterations=0*, *metric='MI'*, *thetas=array([0., 90., 180., 270., 360.])*, *thetas2=array([0., 90., 180., 270., 360.])*, *thetas3=array([0., 90., 180., 270., 360.])*, *scale_image=1*, *do_reflection=False*, *txfn=None*, *transform='Affine'*)

Similarity metrics between two images as a function of geometry

Compute similarity metric between two images as image is rotated about its center w/ or w/o optimization

ANTsR function: *invariantImageSimilarity*

Parameters

- **`image1`** (`ANTsImage`) – reference image
- **`image2`** (`ANTsImage`) – moving image
- **`local_search_iterations`** (*`integer`*) – integer controlling local search in multistart
- **`metric`** (*`string`*) –

  which metric to use MI GC
- **`thetas`** (*`1D-ndarray/list/tuple`*) – numeric vector of search angles in degrees
- **`thetas2`** (*`1D-ndarray/list/tuple`*) – numeric vector of search angles in degrees around principal axis 2 (3D)
- **`thetas3`** (*`1D-ndarray/list/tuple`*) – numeric vector of search angles in degrees around principal axis 3 (3D)
- **`scale_image`** (*`scalar`*) – global scale
- **`do_reflection`** (*`boolean`*) – whether to reflect image about principal axis
- **`txfn`** (*`string (optional)`*) – if present, write optimal tx to .mat file
- **`transform`** (*`string`*) –

  type of transform to use Rigid Similarity Affine

Returns dataframe with metric values and transformation parameters

Return type pd.DataFrame

Example

```
>>> import ants
>>> img1 = ants.image_read(ants.get_ants_data('r16'))
>>> img2 = ants.image_read(ants.get_ants_data('r64'))
>>> metric = ants.invariant_image_similarity(img1,img2)
```

ants.**convolve_image**(*image*, *kernel_image*, *crop=True*)
Convolve one image with another

ANTsR function: *convolveImage*

Parameters

- **image** (ANTsImage) – image to convolve
- **kernel_image** (ANTsImage) – image acting as kernel
- **crop** (*boolean*) – whether to automatically crop kernel_image

Returns

Return type *ANTsImage*

Example

```
>>> import ants
>>> fi = ants.image_read(ants.get_ants_data('r16'))
>>> convimg = ants.make_image( (3,3), (1,0,1,0,-4,0,1,0,1) )
>>> convout = ants.convolve_image( fi, convimg )
>>> convimg2 = ants.make_image( (3,3), (0,1,0,1,0,-1,0,-1,0) )
>>> convout2 = ants.convolve_image( fi, convimg2 )
```

ants.**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** (ANTsImage) – input image e.g. a statistical map
- **min_cluster_size** (*integer*) – 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** (*boolean*) – boolean sets neighborhood connectivity pattern

Returns

Return type *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_image_centroids`**(*image*, *physical=False*, *convex=True*, *verbose=False*)

Converts a label image to coordinates summarizing their positions

ANTsR function: *labelImageCentroids*

Parameters

- **`image`** (`ANTsImage`) – image of integer labels
- **`physical`** (*`boolean`*) – whether you want physical space coordinates or not
- **`convex`** (*`boolean`*) – 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.`**`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`** (`ANTsImage`) – Input image.
- **`mask`** (`ANTsImage`) – Mask or label image.
- **`level`** (*`scalar or tuple of scalars`*) – Level(s) at which to mask image. If vector or list of values, output image is non-zero at all locations where label image matches any of the levels specified.
- **`binarize`** (*`boolean`*) – whether binarize the output image

Returns

Return type *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.`**`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.

Returns

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.`**`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`** (`ANTsImage`) – input image
- **`operation`** (*`string`*) –

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

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

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** (*boolean*) – used parametric radius boolean (shape='ball' and shape='annulus' only)
- **thickness** (*scalar*) – thickness (shape='annulus' only)
- **lines** (*integer*) – number of lines in polygon (shape='polygon' only)
- **include_center** (*boolean*) – include center of annulus boolean (shape='annulus' only)

Returns

Return type *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.**get_pointer_string**(*image*)

ants.**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** (*boolean*) – If true, the smoothing factor is in millimeters; if false, it is in pixels.
- **FWHM** (*boolean*) – 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

Returns

Return type *ANTsImage*

Example

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

`ants.`**`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** (`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** (*boolean*) – if true, returns binary thresholded image if false, return binary thresholded image multiplied by original image

Returns

Return type *ANTsImage*

Example

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

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

Returns

Return type *ANTsImage*

Example

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

CHAPTER 6

Visualization

ants.**plot**(*image*, *overlay=None*, *blend=False*, *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*, *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*)

Plot an ANTsImage.

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** (ANTsImage) – image to plot
- **overlay** (ANTsImage) – image to overlay on base image
- **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.
- **axis** (*integer*) – which axis to plot along if image is 3D
- **nslices** (*integer*) – number of slices to plot if image is 3D
- **slices** (*list or tuple of python: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** (*integer*) – Number of columns to have on the plot if image is 3D.

- **slice_buffer** (*integer*) – 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** (*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
- **reverse** (*boolean*) – 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** (*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

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)
```

ants.**surf**(*x*, *y=None*, *z=None*, *quantlimits=(0.1, 0.9)*, *colormap='jet'*, *grayscale=0.7*, *bg_grayscale=0.9*, *alpha=None*, *inflation_factor=0*, *tol=0.03*, *smoothing_sigma=0.0*, *rotation_params=(90, 0, 270)*, *overlay_limits=None*, *filename=None*, *verbose=False*)

Render a function onto a surface.

ANTsR function: ***antsrSurf*** NOTE: the ANTsPy version of this function does NOT make a function call to ANTs, unlike the ANTsR version, so you don't have to worry about paths.

Parameters

- **x** (ANTsImage) – input image defining the surface on which to render
- **y** (ANTsImage) – input image list defining the function to render on the surface. these image(s) should be in the same space as x.
- **z** (ANTsImage) – input image list mask for each y function to render on the surface. these image(s) should be in the same space as y.
- **quantlimits** (*tuple/list*) – lower and upper quantile limits for overlay
- **colormap** (*string*) – one of: grey, red, green, blue, copper, jet, hsv, spring, summer, autumn, winter, hot, cool, overunder, custom
- **alpha** (*scalar*) – transparency vector for underlay and each overlay, default zero
- **inflation_factor** (*integer*) – number of inflation iterations to run
- **tol** (*float*) – error tolerance for surface reconstruction. Smaller values will lead to better surfaces, at the cost of taking longer. Try decreasing this value if your surfaces look very block-y.
- **smoothing_sigma** (*scalar*) – gaussian smooth the overlay by this sigma
- **rotation_params** (*tuple/list/ndarray*) – 3 Rotation angles expressed in degrees or a matrix of rotation parameters that will be applied in sequence.
- **overlay_limits** (*tuple (optional)*) – absolute lower and upper limits for functional overlay. this parameter will override quantlimits. Currently, this will set levels above overlayLimits[2] to overlayLimits[1]. Can be a list of length of y.
- **filename** (*string*) – prefix filename for output pngs
- **verbose** (*boolean*) – prints the command used to call antsSurf

Returns

Return type N/A

Example

```
>>> import ants
>>> ch2i = ants.image_read( ants.get_ants_data("ch2") )
>>> ch2seg = ants.threshold_image( ch2i, "Otsu", 3 )
>>> wm   = ants.threshold_image( ch2seg, 3, 3 )
>>> wm2 = wm.smooth_image( 1 ).threshold_image( 0.5, 1e15 )
>>> kimg = ants.weingarten_image_curvature( ch2i, 1.5  ).smooth_image( 1 )
>>> wmz = wm2.iMath("MD",3)
>>> rp = [(90,180,90), (90,180,270), (90,180,180)]
>>> ants.surf( x=wm2, y=[kimg], z=[wmz],
            inflation_factor=255, overlay_limits=(-0.3,0.3), verbose = True,
            rotation_params = rp, filename='/users/ncullen/desktop/surface.png')
```

`ants.vol` (*volume*, *overlays=None*, *quantlimits=(0.1, 0.9)*, *colormap='jet'*, *rotation_params=(90, 0, 270)*, *overlay_limits=None*, *magnification_factor=1.0*, *intensity_truncation=(0.0, 1.0)*, *filename=None*, *verbose=False*)

Render an ANTsImage as a volume with optional ANTsImage functional overlay. This function is beautiful, and runs very fast. It requires VTK.

ANTsR function: *antsrVol* NOTE: the ANTsPy version of this function does NOT make a function call to ANTs, unlike the ANTsR version, so you don't have to worry about paths.

Parameters

- **volume** (`ANTsImage`) – base volume to render
- **overlay** (*list of ANTsImages*) – functional overlay to render on the volume image. These images should be in the same space
- **colormap** (*string*) –

  possible values: grey, red, green, blue, copper, jet, hsv, spring, summer, autumn, winter, hot, cool, overunder, custom
- **rotation_params** (*tuple or collection of tuples or np.ndarray w/ shape (N,3)*) – rotation parameters to render. The final image will be a stitch of each image from the given rotation params. e.g. if rotation_params = [(90,90,90),(180,180,180)], then the final

  stiched image will have 2 brain renderings at those angles
- **overlay_limts** –
- **magnification_factor** (*float*) – how much to zoom in on the image before rendering. If the stitched images are too far apart, try increasing this value. If the brain volume gets cut off in the image, try decreasing this value
- **intensity_truncation** (*2-tuple of python:float*) – percentile to truncate intensity of overlay
- **filename** (*string*) – final filename to which the final rendered volume stitch image will be saved this will always be a .png file
- **verbose** (*boolean*) – whether to print updates during rendering

Returns

- *- a numpy array representing the final stitched image.*
- *Effects*
- *——-*
- *- saves a few png files to disk*

Example

```
>>> import ants
>>> ch2i = ants.image_read( ants.get_ants_data("mni") )
>>> ch2seg = ants.threshold_image( ch2i, "Otsu", 3 )
>>> wm   = ants.threshold_image( ch2seg, 2, 2 )
>>> kimg = ants.weingarten_image_curvature( ch2i, 1.5  ).smooth_image( 1 )
>>> rp = [(90,180,90), (90,180,270), (90,180,180)]
>>> result = ants.vol( wm, [kimg], quantlimits=(0.01,0.99), filename='/users/
↪ncullen/desktop/voltest.png')
```

ants.**render_surface_function**(*surfimg*, *funcimg=None*, *alphasurf=0.2*, *alphafunc=1.0*, *isosurf=0.5*, *isofunc=0.5*, *smoothsurf=None*, *smoothfunc=None*, *cmapsurf='grey'*, *cmapfunc='red'*, *filename=None*, *notebook=False*, *auto_open=False*)

Render an image as a base surface and an optional collection of other image.

ANTsR function: ***renderSurfaceFunction*** NOTE: The ANTsPy version of this function is actually completely different than the ANTsR version, although they should produce similar results.

Parameters

- **surfimg** (ANTsImage) – Input image to use as rendering substrate.
- **funcimg** (ANTsImage) – Input list of images to use as functional overlays.
- **alphasurf** (*scalar*) – alpha for the surface contour
- **alphafunc** (*scalar*) – alpha value for functional blobs
- **isosurf** (*scalar*) – intensity level that defines lower threshold for surface image
- **isofunc** (*scalar*) – intensity level that defines lower threshold for functional image
- **smoothsurf** (*scalar (optional)*) – smoothing for the surface image
- **smoothfunc** (*scalar (optional)*) – smoothing for the functional image
- **cmapsurf** (*string*) – color map for surface image
- **cmapfunc** (*string*) – color map for functional image
- **filename** (*string*) – where to save rendering. if None, will plot interactively
- **notebook** (*boolean*) – whether you're in a jupyter notebook.

Returns

Return type N/A

Example

```
>>> import ants
>>> mni = ants.image_read(ants.get_ants_data('mni'))
>>> mnia = ants.image_read(ants.get_ants_data('mnia'))
>>> ants.render_surface_function(mni, mnia, alphasurf=0.1, filename='/users/
↪ncullen/desktop/surffnc.png')
```

CHAPTER 7

Indices and tables

- genindex
- modindex

Python Module Index

Index