Package 'rpyANTs'

July 16, 2023

Title An Alternative Advanced Normalization Tools ('ANTs')

Version 0.0.2

Description Provides portable access from 'R' to biomedical image processing toolbox 'ANTs' by Avants et al. (2009) <doi:10.54294/uvnhin> via seamless integration with the 'Python' implementation 'ANTsPy'. Allows biomedical images to be processed in 'Python' and analyzed in 'R', and vice versa via shared memory. See 'citation(``rpyANTs'')' for more reference information.

License Apache License 2.0

Language en-US

Encoding UTF-8

RoxygenNote 7.2.3

Imports grDevices, rpymat (>= 0.1.6), reticulate (>= 1.26)

Suggests RNifti (>= 1.4.4), testthat (>= 3.0.0)

Config/testthat/edition 3

URL http://dipterix.org/rpyANTs/

BugReports https://github.com/dipterix/rpyANTs/issues

Copyright For the rpyANTs package - Zhengjia Wang; For ANTs - ConsortiumOfANTS; For ANTsPy - ANTs Contributors.

NeedsCompilation no

Author Zhengjia Wang [aut, cre]

Maintainer Zhengjia Wang <dipterix.wang@gmail.com>

Repository CRAN

Date/Publication 2023-07-16 13:50:02 UTC

R topics documented:

`ants` *Get 'ANTsPy' module*

Description

Get 'ANTsPy' module

Usage

```
ants

load_ants(force = FALSE, error_if_missing = TRUE)
```

Arguments

`force`	whether to force reloading `ants` module; default is false
`error_if_missing`	whether to raise errors when the module is unable to load; default is true.

Value

A 'Python' module if successfully loaded. If `error_if_missing` is set to false and module is unable to load, return `NULL`

See Also

`antspynet`

antspynet *Get 'ANTsPyNet' module*

Description

Get 'ANTsPyNet' module

Usage

```
antspynet

load_antspynet(force = FALSE, error_if_missing = TRUE)
```

Arguments

force	whether to force reloading antspynet module; default is false
error_if_missing	whether to raise errors when the module is unable to load; default is true.

Value

A 'Python' module if successfully loaded. If error_if_missing is set to false and module is unable to load, return NULL

See Also

ants

antspynet_brain_extraction *Extract brain and strip skull*

Description

Print antspynet$brain_extraction to see the original documentation.

Usage

```
antspynet_brain_extraction(
  x,
 modality = c("t1", "t1nobrainer", "t1combined", "flair", "t2", "t2star", "bold", "fa",
    "t1t2infant", "t1infant", "t2infant"),
  verbose = FALSE
)
```

Arguments

x	input image or image path
modality	modality type
verbose	whether to print out process to the screen

Value

Brain mask image

---

antspynet_preprocess_brain_image

Process brain image prior to segmentation

Description

Strip skulls, normalize intensity, align and re-sample to template. This procedure is needed for many antspynet functions since the deep neural networks are trained in template spaces

Usage

```
antspynet_preprocess_brain_image(
  x,
  truncate_intensity = c(0.01, 0.99),
 brain_extraction_modality = c("none", "t1", "t1v0", "t1nobrainer", "t1combined",
    "flair", "t2", "bold", "fa", "t1infant", "t2infant"),
  template_transform_type = c("None", "Affine", "Rigid"),
  template = c("biobank", "croppedMni152"),
  do_bias_correction = TRUE,
  return_bias_field = FALSE,
  do_denoising = TRUE,
  intensity_matching_type = c("regression", "histogram"),
  reference_image = NULL,
  intensity_normalization_type = NULL,
  verbose = TRUE
)
```

Arguments

x	'ANTsImage' or path to image to process
truncate_intensity	defines the quantile threshold for truncating the image intensity
brain_extraction_modality	character of length 1, perform brain extraction modality
template_transform_type	either 'Rigid' or 'Affine' align to template brain

`template` template image (not skull-stripped) or string, e.g. `'biobank'`, `'croppedMni152'`

`do_bias_correction`
whether to perform bias field correction

`return_bias_field`
return bias field as an additional output without bias correcting the image

`do_denoising` whether to remove noises using non-local means

`intensity_matching_type`
either `'regression'` or `'histogram'`; only is performed if `reference_image` is not `NULL`.

`reference_image`
`'ANTsImage'` or path to image, or `NULL`

`intensity_normalization_type`
either re-scale the intensities to `c(0, 1)` (`'01'`), or for zero-mean, unit variance (`'0mean'`); if `NULL` normalization is not performed

`verbose` print progress to the screen

Value

Dictionary with images after process. The images are registered and re-sampled into template.

See Also

`antspynet$preprocess_brain_image`

Examples

```
library(rpyANTs)
if(interactive() && ants_available("antspynet")) {
  image_path <- ants$get_ants_data('r30')
  preprocessed <- antspynet_preprocess_brain_image(
    image_path, verbose = FALSE
  )

  # Compare
  orig_img <- as_ANTsImage(image_path)
  new_img <- preprocessed$preprocessed_image
  pal <- grDevices::gray.colors(256, start = 0, end = 1)

  par(mfrow = c(1, 2), mar = c(0.1, 0.1, 0.1, 0.1),
      bg = "black", fg = "white")
  image(orig_img[], asp = 1, axes = FALSE,
        col = pal, ylim = c(1, 0))
  image(new_img[], asp = 1, axes = FALSE,
        col = pal, ylim = c(1, 0))

}
```

antspynet_segmentation *Imaging segmentation using* antspynet

Description

Supports Desikan-Killiany-Tourville labeling and deep 'Atropos'.

Usage

```
antspynet_desikan_killiany_tourville_labeling(
  x,
  do_preprocessing = TRUE,
  return_probability_images = FALSE,
  do_lobar_parcellation = FALSE,
  verbose = TRUE
)

antspynet_deep_atropos(
  x,
  do_preprocessing = TRUE,
  use_spatial_priors = TRUE,
  aseg_only = TRUE,
  verbose = TRUE
)
```

Arguments

x	'NIfTI' image or path to the image that is to be segmented
do_preprocessing	whether x is in native space and needs the be registered to template brain before performing segmentation; default is true since the model is trained with template brain. If you want to manually process the image, see antspynet_preprocess_brain_image
return_probability_images	whether to return probability images
do_lobar_parcellation	whether to perform lobar 'parcellation'
verbose	whether to print out the messages
use_spatial_priors	whether to use 'MNI' partial tissue priors
aseg_only	whether to just return the segmented image

Value

One or a list of 'ANTsImage' image instances. Please print out antspynet$desikan_killiany_tourville_labeling or antspynet$deep_atropos to see the details.

See Also

antspynet$desikan_killiany_tourville_labeling, antspynet$deep_atropos

Examples

```
# Print Python documents
if(interactive() && ants_available("antspynet")) {
  antspynet <- load_antspynet()

  print(antspynet$deep_atropos)

  print(antspynet$desikan_killiany_tourville_labeling)
}
```

---

ants_apply_transforms *Apply a transform list to map an image from one domain to another*

Description

See ants$apply_transforms for more details.

Usage

```
ants_apply_transforms(
  fixed,
  moving,
  transformlist,
 interpolator = c("linear", "nearestNeighbor", "gaussian", "genericLabel", "bSpline",
    "cosineWindowedSinc", "welchWindowedSinc", "hammingWindowedSinc",
    "lanczosWindowedSinc"),
  imagetype = 0L,
  whichtoinvert = NULL,
  compose = NULL,
  defaultvalue = 0,
  verbose = FALSE,
  ...
)
```

Arguments

fixed	fixed image defining domain into which the moving image is transformed
moving	moving image to be mapped to fixed space
transformlist	list of strings (path to transforms) generated by ants_registration where each transform is a file name

interpolator	how to interpolate the image; see 'Usage'
imagetype	integer: 0 (scalar), 1 (vector), 2 (tensor), 3 (time-series), used when the fixed and moving images have different mode (dimensions)
whichtoinvert	either NULL, None ('Python'), or a vector of logical with same length as transformlist; print ants$apply_transforms to see detailed descriptions
compose	optional character pointing to a valid file location
defaultvalue	numerical value for mappings outside the image domain
verbose	whether to verbose application of transform
...	must be named arguments passing to further methods

Value

Transformed image. The image will share the same space as fixed.

See Also

print(ants$apply_transforms)

Examples

```
if(interactive() && ants_available()) {
  ants <- load_ants()
  fixed <- as_ANTsImage( ants$get_ants_data('r16') )
  moving <- as_ANTsImage( ants$get_ants_data('r64') )
  fixed <- ants_resample_image(fixed, c(64, 64), TRUE, "linear")
  moving <- ants_resample_image(moving, c(64,64), TRUE, "linear")

  mytx <- ants_registration(fixed = fixed,
                            moving = moving,
                            type_of_transform = 'SyN')
  mywarpedimage <- ants_apply_transforms(
    fixed = fixed,
    moving = moving,
    transformlist = mytx$fwdtransforms
  )

  par(mfrow = c(1,3), mar = c(0,0,3,0))
  pal <- gray.colors(256)
  image(fixed[], asp = 1, axes = FALSE, col = pal,
        ylim = c(1, 0), main = "Reference")
  image(moving[], asp = 1, axes = FALSE, col = pal,
        ylim = c(1, 0), main = "Moving")
  image(mywarpedimage[], asp = 1, axes = FALSE, col = pal,
        ylim = c(1, 0), main = "Moving reg+resamp into Reference")
}
```

ants_apply_transforms_to_points *Apply a transform list to map points from one domain to another*

Description

See ants$apply_transforms_to_points for more details. Please note point mapping goes the opposite direction of image mapping (see ants_apply_transforms), for both reasons of convention and engineering.

Usage

```
ants_apply_transforms_to_points(
  dim,
  points,
  transformlist,
  whichtoinvert = NULL,
  verbose = FALSE,
  ...
)
```

Arguments

dim	dimensions of the transformation
points	data frame containing columns 'x', 'y', 'z', 't' (depending on dim)
transformlist	list of strings (path to transforms) generated by ants_registration where each transform is a file name
whichtoinvert	either NULL, None ('Python'), or a vector of logical with same length as transformlist; print ants$apply_transforms_to_points to see detailed descriptions
verbose	whether to verbose application of transform
...	ignored

Value

Transformed points in data frame (R object)

See Also

```
print(ants$apply_transforms_to_points)
```

Examples

```
if(interactive() && ants_available()) {
  ants <- load_ants()
  fixed <- as_ANTsImage( ants$get_ants_data('r16') )
```

```
  moving <- as_ANTsImage( ants$get_ants_data('r27') )

  reg <- ants_registration(
    fixed = fixed, moving = moving,
    type_of_transform = "antsRegistrationSyNRepro[a]")

  pts <- data.frame(
    x = c(128, 127),
    y = c(101, 111)
  )

  ptsw = ants_apply_transforms_to_points(2, pts, reg$fwdtransforms)
  ptsw
}
```

`ants_available` *Check if 'ANTs' is available*

Description

Check if 'ANTs' is available

Usage

```
ants_available(module = c("ants", "antspynet"))
```

Arguments

`module`	either `'ants'` or `'antspynet'`; default is `'ants'`

Value

Logical, whether `'ANTs'` or `'ANTsPyNet'` is available

See Also

`install_ants`

`ants_motion_correction` *Motion correction*

Description

Print `ants$motion_correction` to see the original document

Usage

```
ants_motion_correction(
  x,
  fixed = NULL,
  type_of_transform = "BOLDRigid",
  mask = NULL,
  fdOffset = 50,
  outprefix = "",
  verbose = FALSE,
  ...
)
```

Arguments

`x`	input image, usually 'fMRI' series
`fixed`	fixed image to register all timepoints to
`type_of_transform`	see `ants_registration`
`mask`	mask for image
`fdOffset`	offset value to use in frame-wise displacement calculation
`outprefix`	save path
`verbose`	whether to verbose the messages
`...`	passed to registration methods

Value

Motion-corrected image

Examples

```
if(interactive() && ants_available()) {
  fi <- as_ANTsImage(ants$get_ants_data('ch2'))
  mytx <- ants_motion_correction( fi )

  par(mfrow = c(1, 2), mar = c(1,1,1,1))
  image(fi[,,91], asp = 1, axes = FALSE)
```

```
  image(mytx$motion_corrected[,,91], asp = 1, axes = FALSE)
}
```

---

ants_plot *Plot single* 'ANTsImage'

Description

Plot single 'ANTsImage'

Usage

```
ants_plot(
  image,
  overlay = NULL,
  blend = FALSE,
  alpha = 1,
  cmap = "Greys_r",
  overlay_cmap = "turbo",
  overlay_alpha = 0.9,
  vminol = NULL,
  vmaxol = NULL,
  cbar = FALSE,
  cbar_length = 0.8,
  cbar_dx = 0,
  cbar_vertical = TRUE,
  axis = 0,
  nslices = 12,
  slices = NULL,
  ncol = NULL,
  slice_buffer = NULL,
  black_bg = TRUE,
  bg_thresh_quant = 0.01,
  bg_val_quant = 0.99,
  domain_image_map = NULL,
  crop = FALSE,
  scale = FALSE,
  reverse = FALSE,
  title = "",
  title_fontsize = 20,
  title_dx = 0,
  title_dy = 0,
  filename = NULL,
  dpi = 500,
  figsize = 1.5,
  reorient = TRUE,
```

```
  resample = TRUE,
  force_agg = FALSE,
  close_figure = TRUE
)
```

Arguments

`image`	`'ANTsImage'`, or something can be converted to `'ANTsImage'`
`overlay`	overlay `'ANTsImage'`, can be `NULL`, optional
`blend`	whether to blend image with overlay; default is false
`cmap, alpha`	image color map and transparency
`overlay_cmap, overlay_alpha`	overlay color map and transparency
`vminol, vmaxol`	I could not find its usage
`cbar`	whether to draw color legend
`cbar_length, cbar_dx, cbar_vertical`	legend position and size
`axis`	see 'Details'
`nslices, slices, ncol`	controls slice to show
`slice_buffer`	performance
`black_bg, bg_thresh_quant, bg_val_quant`	controls background
`domain_image_map`	optional `'ANTsImage'`
`crop, scale, reverse`	whether to crop, scale, or reverse the image according to background
`title, title_fontsize, title_dx, title_dy`	image title
`filename, dpi, figsize`	needed when saving to file
`reorient`	whether to reorient to `'LAI'` before plotting; default is true
`resample`	whether to resample
`force_agg`	whether to force graphic engine to use `'agg'` device; default is false
`close_figure`	whether to close figure when returning the function

Details

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.

Value

Nothing

Examples

```
if(interactive() && ants_available()) {
  ants <- load_ants()
  img <- ants$image_read(ants$get_ants_data('mni'))

  ants_plot(
    img, nslices = 12, black_bg = FALSE,
    bg_thresh_quant = 0.05, bg_val_quant = 1.0, axis = 2,
    cbar = TRUE, crop = TRUE, reverse = TRUE, cbar_vertical = FALSE,
    ncol = 4, title = "Axial view of MNI brain"
  )
}
```

---

ants_plot_grid *Plot multiple* 'ANTsImage'

Description

R-friendly wrapper function for ants$plot_grid

Usage

```
ants_plot_grid(
  images,
  shape = NULL,
  slices = 0,
  axes = 2,
  figsize = 1,
  rpad = 0,
  cpad = 0,
  vmin = NULL,
  vmax = NULL,
  colorbar = TRUE,
  cmap = "Greys_r",
  title = "",
  tfontsize = 20,
  title_dx = 0,
  title_dy = 0,
  rlabels = NULL,
  rfontsize = 14,
```

```
  rfontcolor = "black",
  rfacecolor = "white",
  clabels = NULL,
  cfontsize = 14,
  cfontcolor = "black",
  cfacecolor = "white",
  filename = NULL,
  dpi = 400,
  transparent = TRUE,
  ...,
  force_agg = FALSE,
  close_figure = TRUE
)
```

Arguments

`images`	a single `'ANTsImage'`, list, or nested list of `'ANTsImage'`
`shape`	shape of grid, default is using dimensions of `images`
`slices`	length of one or equaling to length of `slices`, slice number to plot
`axes`	`0` for `'sagittal'`, `1` for `'coronal'`, `2` for `'axial'`; default is `2`
`figsize, rpad, cpad, colorbar, cmap, transparent`	graphical parameters
`vmin, vmax`	value threshold for the image
`title`	title of figure
`title_dx, title_dy, tfontsize`	controls title margin and size
`rlabels, clabels`	row and column labels
`rfontsize, rfontcolor, rfacecolor, cfontsize, cfontcolor, cfacecolor`	row and column font size, color, and background color
`filename, dpi`	parameters to save figures
`...`	passed to `ants$plot_grid`; make sure all entries are named
`force_agg`	whether to force graphic engine to use `'agg'` device; default is false
`close_figure`	whether to close figure when returning the function

Value

Nothing

Examples

```
if(interactive() && ants_available()) {
  ants <- load_ants()
  image1 <- ants$image_read(ants$get_ants_data('mni'))
  image2 <- image1$smooth_image(1.0)
```

```
  image3 <- image1$smooth_image(2.0)
  image4 <- image1$smooth_image(3.0)

  ants_plot_grid(
    list(image1, image2, image3, image4),
    slices = 100, title = "4x1 Grid"
  )

  ants_plot_grid(
    list(image1, image2, image3, image4),
    shape = c(2, 2),
    slices = 100, title = "2x2 Grid"
  )
  ants_plot_grid(
    list(image1, image2, image3, image4),
    shape = c(2, 2), axes = c(0,1,2,1),
    slices = 100, title = "2x2 Grid (diff. anatomical slices)"
  )

}
```

---

ants_registration *Register two images using 'ANTs'*

Description

Register two images using 'ANTs'

Usage

```
ants_registration(
  fixed,
  moving,
  type_of_transform = "SyN",
  initial_transform = NULL,
  outprefix = tempfile(),
  mask = NULL,
  grad_step = 0.2,
  flow_sigma = 3,
  total_sigma = 0,
  aff_metric = c("mattes", "GC", "meansquares"),
  aff_sampling = 32,
  aff_random_sampling_rate = 0.2,
  syn_metric = c("mattes", "CC", "meansquares", "demons"),
  syn_sampling = 32,
  reg_iterations = c(40, 20, 0),
```

```
  aff_iterations = c(2100, 1200, 1200, 10),
  aff_shrink_factors = c(6, 4, 2, 1),
  aff_smoothing_sigmas = c(3, 2, 1, 0),
  write_composite_transform = FALSE,
  verbose = FALSE,
  smoothing_in_mm = FALSE,
  ...
)
```

Arguments

`fixed`	fixed image to which we register the moving image, can be character path to 'NIfTI' image, or `'ANTsImage'` instance, `'oro.nifti'` object, `'niftiImage'` from package `'RNifti'`, or `'threeBrain.nii'` from package `'threeBrain'`; see also `as_ANTsImage`
`moving`	moving image to be mapped to fixed space; see also `as_ANTsImage`
`type_of_transform`	a linear or non-linear registration type; print `ants$registration` to see details
`initial_transform`	optional list of strings; transforms to apply prior to registration
`outprefix`	output file to save results
`mask`	image mask; see also `as_ANTsImage`
`grad_step, flow_sigma, total_sigma`	optimization parameters
`aff_metric`	the metric for the 'affine' transformation, choices are `'GC'`, `'mattes'`, `'meansquares'`
`aff_sampling, aff_random_sampling_rate, aff_iterations, aff_shrink_factors, aff_smoothing_sigmas`	controls 'affine' transform
`syn_metric`	the metric for the `'SyN'` transformation, choices are `'GC'`, `'mattes'`, `'meansquares'`, `'demons'`
`syn_sampling, reg_iterations`	controls the `'SyN'` transform
`write_composite_transform`	whether the composite transform (and its inverse, if it exists) should be written to an 'HDF5' composite file; default is false
`verbose`	verbose the progress
`smoothing_in_mm`	logical, currently only impacts low dimensional registration
`...`	others passed to `ants$registration`

Value

A 'Python' dictionary of aligned images and transform files.

Examples

```
if(interactive() && ants_available()) {

  ants <- load_ants()

  # check the python documentation here for detailed explanation
  print(ants$registration)

  # example to register
  fi <- ants$image_read(ants$get_ants_data('r16'))
  mo <- ants$image_read(ants$get_ants_data('r64'))

  # resample to speed up this example
  fi <- ants$resample_image(fi, list(60L,60L), TRUE, 0L)
  mo <- ants$resample_image(mo, list(60L,60L), TRUE, 0L)

  # SDR transform
  transform <- ants_registration(
    fixed=fi, moving=mo, type_of_transform = 'SyN' )

  ants$plot(fi, overlay = transform$warpedmovout, overlay_alpha = 0.3)


}
```

`ants_resample_image` *Resample image*

Description

See `ants$resample_image` for more details

Usage

```
ants_resample_image(
  x,
  resample_params,
  use_voxels = FALSE,
  interp_type = c("linear", "nn", "guassian", "sinc", "bspline")
)
```

Arguments

`x`	input image
`resample_params`	either relative number or absolute integers

use_voxels	whether the resample_params should be treated as new dimension use_voxels=TRUE, or the new dimension should be calculated based on current dimension and resample_params combined (use_voxels=FALSE then resample_params will be treated as relative number); default is FALSE
interp_type	interpolation type; either integer or character; see 'Usage' for available options

Value

Resampled image

Examples

```
if(interactive() && ants_available()) {
  ants <- load_ants()
  fi <- as_ANTsImage(ants$get_ants_data("r16"))

  # linear (interp_type = 0 or "linear)
  filin <- ants_resample_image(fi, c(50, 60), TRUE, "linear")

  # nearest neighbor (interp_type = 1 or "nn)
  finn <- ants_resample_image(fi, c(50, 60), TRUE, "nn")

  par(mfrow = c(1, 3), mar = c(0, 0, 0, 0))
  pal <- gray.colors(256, start = 0)

  image(fi[], asp = 1, axes = FALSE,
        ylim = c(1,0), col = pal)
  image(filin[], asp = 1, axes = FALSE,
        ylim = c(1,0), col = pal)
  image(finn[], asp = 1, axes = FALSE,
        ylim = c(1,0), col = pal)
}
```

---

as_ANTsImage *Load data as 'ANTsImage' class*

Description

Load data as 'ANTsImage' class

Usage

```
as_ANTsImage(x, strict = FALSE)
```

Arguments

x	data to be converted; this can be an 'ANTsImage' instance, character, 'oro.nifti' object, 'niftiImage' from package 'RNifti', or 'threeBrain.nii' from package 'threeBrain'
strict	whether x should not be NULL

Value

An 'ANTsImage' instance; use ants$ANTsImage to see the 'Python' documentation

Examples

```
if(interactive() && ants_available()) {

  ants <- load_ants()

  # Python string
  x1 <- ants$get_ants_data('r16')
  as_ANTsImage( x1 )

  # R character
  nii_path <- system.file(package = "RNifti",
                          "extdata", "example.nii.gz")
  as_ANTsImage( nii_path )

  # niftiImage object
  x2 <- RNifti::readNifti(nii_path)
  as_ANTsImage( x2 )

}
```

---

as_ANTsTransform *Convert to 'ANTsTransform'*

Description

Convert to 'ANTsTransform'

Usage

```
as_ANTsTransform(x, ...)

## Default S3 method:
as_ANTsTransform(x, dimension = 3, ...)

## S3 method for class 'ants.core.ants_transform.ANTsTransform'
```

```
as_ANTsTransform(x, ...)

## S3 method for class 'ants.core.ants_image.ANTsImage'
as_ANTsTransform(x, ...)

## S3 method for class 'numpy.ndarray'
as_ANTsTransform(x, ...)

## S3 method for class 'character'
as_ANTsTransform(x, ...)
```

Arguments

x	'affine' matrix or 'numpy' array, character path to the matrix, 'ANTsTransform', 'ANTsImage' as displacement field.
...	passed to other methods
dimension	expected transform space dimension; default is 3

Value

An 'ANTsTransform' object

Examples

```
if(interactive() && ants_available()) {

  mat <- matrix(c(
    0, -1, 0, 128,
    1, 0, 0, -128,
    0, 0, -1, 128,
    0, 0,  0,   1
  ), ncol = 4, byrow = TRUE)

  trans <- as_ANTsTransform(mat)
  trans

  # apply transform
  trans$apply_to_point(c(120, 400, 1))

  # same results
  mat %*% c(120, 400, 1, 1)

  trans[] == mat

}
```

install_ants *Install 'ANTs' via 'ANTsPy'*

Description

Install 'ANTs' via 'ANTsPy'

Usage

```
install_ants(python_ver = "3.9", verbose = TRUE)
```

Arguments

python_ver	'Python' version, see configure_conda; default is "3.9" since 'ANTsPy' is compiled for all
verbose	whether to print the installation messages

Value

This function returns nothing.

is_affine3D *Check if an object is a 3D 'affine' transform matrix*

Description

Check if an object is a 3D 'affine' transform matrix

Usage

```
is_affine3D(x, ...)

## Default S3 method:
is_affine3D(x, strict = TRUE, ...)

## S3 method for class 'ants.core.ants_transform.ANTsTransform'
is_affine3D(x, ...)
```

Arguments

x	R or Python object, accepted forms are numeric matrix, 'ANTsTransform', or character (path to transform matrix)
...	passed to other methods
strict	whether the last element should be always 1

Value

A logical value whether the object can be loaded as a 4-by-4 matrix.

Examples

```
# not affine
is_affine3D(1)

# 3x3 matrix is not as it is treated as 2D transform
is_affine3D(matrix(rnorm(9), nrow = 3))

# 3x4 matrix
x <- matrix(rnorm(12), nrow = 3)
is_affine3D(x)

# 4x4 matrix
x <- rbind(x, c(0,0,0,1))
is_affine3D(x)

if(interactive() && ants_available()) {

  ants <- load_ants()
  x <- ants$new_ants_transform(dimension = 3L)
  is_affine3D(x)

  # save the parameters
  f <- tempfile(fileext = ".mat")
  ants$write_transform(x, f)
  is_affine3D(f)

}
```

---

py *Get 'Python' main process environment*

Description

Get 'Python' main process environment

Usage

```
py
```

Format

An object of class `python.builtin.module` (inherits from `python.builtin.object`) of length 1.

Value

The 'Python' main process as a module

`py_builtin` *Get 'Python' built-in object*

Description

Get 'Python' built-in object

Usage

```
py_builtin(name, convert = TRUE)
```

Arguments

`name`	object name
`convert`	see `import_builtins`

Value

A python built-in object specified by `name`

Examples

```
if(interactive() && ants_available()) {


# ------ Basic case: use python `int` as an R function ---------
py_int <- py_builtin("int")

# a is an R object now
a <- py_int(9)
print(a)
class(a)

# ------ Use python `int` as a Python function -----------------
py_int2 <- py_builtin("int", convert = FALSE)

# b in a python object
b <- py_int2(9)

# There is no '[1] ' when printing
print(b)
```

```
class(b)

# convert to R object
py_to_r(b)



}
```

`py_list` *List in 'Python'*

Description

List in 'Python'

Usage

```
py_list(..., convert = FALSE)
```

Arguments

`...`	passing to `list` ('Python')
`convert`	whether to convert the results back into R; default is no

Value

List instance, or an R vector if converted

Examples

```
if(interactive() && ants_available()) {

  py_list(list(1,2,3))
  py_list(c(1,2,3))

  py_list(array(1:9, c(3,3)))
  py_list(list(list(1:3), letters[1:3]))

}
```

py_slice *Slice index in 'Python' arrays*

Description

Slice index in 'Python' arrays

Usage

```
py_slice(...)
```

Arguments

...	passing to `slice` ('Python')

Value

Index slice instance

Examples

```
if(interactive() && ants_available()) {

  x <- np_array(array(seq(20), c(4, 5)))

  # equivalent to x[::2]
  x[py_slice(NULL, NULL, 2L)]

}
```

Index