anndata Documentation

Release 0.6.10+22.ge3e6c90

Alex Wolf, Philipp Angerer

Sep 26, 2018

Contents

Install via `pip install anndata` or `conda install anndata -c bioconda`.

Report issues and see the code on GitHub.

anndata is for simple (functional) high-level APIs for data analysis pipelines. In this context, it provides an efficient, scalable way of keeping track of data together with learned annotations and reduces the code overhead typically encountered when using a mostly object-oriented library such as *scikit-learn*.

The prime use is currently for Scanpy, for which *anndata* was initially developed. Both packages have been introduced in Genome Biology (2018).

See all releases here. The following lists selected improvements.

September 14, 2018: on master

1. *layers()* inspired by .loom files allows their information lossless reading via *read_loom()*
2. initialatization from pandas DataFrames
3. iteration over chunks *chunked_X()* and *chunk_X()*
4. support for reading zarr files: *read_zarr()*

May 1, 2018: version 0.6

1. compatibility with Seurat converter
2. tremendous speedup for *concatenate()*

April 17, 2018: versions 0.5.9 - 0.5.10

1. bug fix for deep copy of unstructured annotation after slicing

March 16, 2018: versions 0.5.1 - 0.5.8

1. bug fix for reading HDF5 stored single-category annotations
2. 'outer join' concatenation: adds zeros for concatenation of sparse data and nans for dense data
3. better memory efficiency in loom exports
4. consistency and documentation updates
5. prettified print output

Warning: There was a bug in *concatenate()* in versions 0.5.2, 0.5.3 and 0.5.4: variable names were not assigned correctly. Was fixed in version 0.5.5.

February 9, 2018: version 0.5

1. inform about duplicates in *var_names* and resolve them using *var_names_make_unique()*
2. automatically remove unused categories after slicing
3. read/write *.loom* files using loompy 2
4. some IDE-backed improvements

December 29, 2017: version 0.4.2

1. fixed read/write for a few text file formats
2. read UMI tools files: *read_umi_tools()*

December 23, 2017: version 0.4

1. towards a common file format for exchanging *AnnData* with packages such as Seurat and SCDE by reading and writing .loom files
2. *AnnData* provides scalability beyond dataset sizes that fit into memory: see this blog post
3. *AnnData* has a *raw* attribute that simplifies storing the data matrix when you consider it "raw": see the clustering tutorial

CHAPTER 1

API

The central class:

AnnData([X, obs, var, uns, obsm, varm, ...])	An annotated data matrix.

1.1 anndata.AnnData

class anndata.**AnnData**(*X=None*, *obs=None*, *var=None*, *uns=None*, *obsm=None*, *varm=None*, *layers=None*, *raw=None*, *dtype='float32'*, *shape=None*, *filename=None*, *filemode=None*, *asview=False*, *, *oidx=None*, *vidx=None*)

An annotated data matrix.

AnnData stores a data matrix `.X` together with annotations of observations `.obs`, variables `.var` and unstructured annotations `.uns`.

An *AnnData* object `adata` can be sliced like a pandas dataframe, for instance, `adata_subset = adata[:, list_of_variable_names]`. *AnnData*'s basic structure is similar to R's ExpressionSet *[Huber15]*. If setting an `.h5ad`-formatted HDF5 backing file `.filename`, data remains on the disk but is automatically loaded into memory if needed. See this blog post for more details.

Parameters

X : Union[ndarray, spmatrix, DataFrame, None] A #observations × #variables data matrix. A view of the data is used if the data type matches, otherwise, a copy is made.

obs : Union[DataFrame, Mapping[Any, Iterable[Any]], ndarray, None] Key-indexed one-dimensional observations annotation of length #observations.

var : Union[DataFrame, Mapping[Any, Iterable[Any]], ndarray, None] Key-indexed one-dimensional variables annotation of length #variables.

uns : Optional[Mapping[Any, Any]] Key-index unstructured annotation.

obsm : Union[ndarray, Mapping[str, Sequence[Any]], None] Key-indexed multi-dimensional observations annotation of length #observations.

varm : Union[ndarray, Mapping[str, Sequence[Any]], None] Key-indexed multi-dimensional variables annotation of length #observations.

dtype : Union[dtype, str] Data type used for storage.

shape : Optional[tuple] Shape tuple (#observations, #variables). Can only be provided if `X` is `None`.

filename : Optional[PathLike] Name of backing file. See *anndata.h5py.File*.

filemode : Optional[str] Open mode of backing file. See *anndata.h5py.File*.

layers : Optional[Mapping[~KT, +VT_co]] Dictionary with keys as layers' names and values as matrices of the same dimensions as X.

See also:

read_h5ad, *read_csv*, *read_excel*, *read_hdf*, *read_loom*, *read_mtx*, *read_text*, *read_umi_tools*

Notes

Multi-dimensional annotations are stored in `.obsm` and `.varm`.

Indexing into an AnnData object with a numeric is supposed to be positional, like pandas `.iloc` method, while indexing with a string/ categorical is supposed to behave like `.loc`.

If the unstructured annotations `.uns` contain a sparse matrix of shape `.n_obs` × `.n_obs`, these are sliced when calling `[]`.

A data matrix is flattened if either `n_obs` or `n_vars` is 1, so that numpy's slicing behavior is reproduced:

```
adata = AnnData(np.ones((2, 2)))
adata[:, 0].X == adata.X[:, 0]
```

AnnData stores observations (samples) of variables (features) in the rows of a matrix. This is the convention of the modern classics of statistics *[Hastie09]* and machine learning *[Murphy12]*, the convention of dataframes both in R and Python and the established statistics and machine learning packages in Python (statsmodels, scikit-learn).

Attributes

T	Transpose whole object.
X	Data matrix of shape *n_obs* × *n_vars*.
filename	Change to backing mode by setting the filename of a `.h5ad` file.
isbacked	`True` if object is backed on disk, `False` otherwise.
isview	`True` if object is view of another AnnData object, `False` otherwise.
layers	Dictionary-like object with values of the same dimensions as .X.
n_obs	Number of observations.
n_vars	Number of variables/features.

Continued on next page

Table 2 – continued from previous page

obs	One-dimensional annotation of observations (*pd.DataFrame*).
obs_names	Names of observations (alias for *.obs.index*).
obsm	Multi-dimensional annotation of observations (mutable structured `np.ndarray`).
raw	Store raw version of `.X` and `.var` as `.raw.X` and `.raw.var`.
shape	*Shape of data matrix* – (n_obs, n_vars).
uns	Unstructured annotation (ordered dictionary).
var	One-dimensional annotation of variables/ features (*pd.DataFrame*).
var_names	Names of variables (alias for *.var.index*).
varm	Multi-dimensional annotation of variables/ features (mutable structured `np.ndarray`).

1.1.1 anndata.AnnData.T

AnnData.**T**

Transpose whole object.

Data matrix is transposed, observations and variables are interchanged.

1.1.2 anndata.AnnData.X

AnnData.**X**

Data matrix of shape *n_obs* × *n_vars*.

Return type Union[ndarray, spmatrix, None]

1.1.3 anndata.AnnData.filename

AnnData.**filename**

Change to backing mode by setting the filename of a `.h5ad` file.

- Setting the filename writes the stored data to disk.
- Setting the filename when the filename was previously another name moves the backing file from the previous file to the new file. If you want to copy the previous file, use `copy(filename='new_filename')`.

1.1.4 anndata.AnnData.isbacked

AnnData.**isbacked**

`True` if object is backed on disk, `False` otherwise.

1.1.5 anndata.AnnData.isview

AnnData.**isview**

`True` if object is view of another AnnData object, `False` otherwise.

1.1.6 anndata.AnnData.layers

AnnData.**layers**
Dictionary-like object with values of the same dimensions as .X.

Layers in AnnData have API similar to loompy http://linnarssonlab.org/loompy/apiwalkthrough/index.html#layers Return the layer named "unspliced":

```
adata.layers["unspliced"]
```

Create or replace the "spliced" layer:

```
adata.layers["spliced"] = ...
```

Assign the 10th column of layer "spliced" to the variable a:

```
a = adata.layers["spliced"][:, 10]
```

Delete the "spliced" layer:

```
del adata.layers["spliced"]
```

Return layers' names:

```
adata.layers.keys()
```

Setting subsets of items in layers writes to the original data also if AnnData is a view.

1.1.7 anndata.AnnData.n_obs

AnnData.**n_obs**
Number of observations.

1.1.8 anndata.AnnData.n_vars

AnnData.**n_vars**
Number of variables/features.

1.1.9 anndata.AnnData.obs

AnnData.**obs**
One-dimensional annotation of observations (*pd.DataFrame*).

1.1.10 anndata.AnnData.obs_names

AnnData.**obs_names**
Names of observations (alias for *.obs.index*).

1.1.11 anndata.AnnData.obsm

AnnData.**obsm**
Multi-dimensional annotation of observations (mutable structured `np.ndarray`).

Stores for each key, a two or higher-dimensional `np.ndarray` of length `n_obs`. Is sliced with `data` and `obs` but behaves otherwise like a `dict`.

1.1.12 anndata.AnnData.raw

AnnData.**raw**
Store raw version of `.X` and `.var` as `.raw.X` and `.raw.var`.

The *.raw* attribute is initialized with the current content of an object by setting:

```
adata.raw = adata
```

Its content can be deleted by setting it back to `None`:

```
adata.raw = None
```

Upon slicing an AnnData object along the observations (row) axis, `.raw` is also sliced. Slicing an AnnData object along the variables (columns) axis, leaves `.raw` unaffected. Note that you can call:

```
adata.raw[:, 'orig_variable_name'].X
```

to retrieve the data associated with a variable that might have been filtered out or "compressed away" in `.X`.

1.1.13 anndata.AnnData.shape

AnnData.**shape**
Shape of data matrix – (n_obs, n_vars).

1.1.14 anndata.AnnData.uns

AnnData.**uns**
Unstructured annotation (ordered dictionary).

1.1.15 anndata.AnnData.var

AnnData.**var**
One-dimensional annotation of variables/ features (*pd.DataFrame*).

1.1.16 anndata.AnnData.var_names

AnnData.**var_names**
Names of variables (alias for *.var.index*).

1.1.17 anndata.AnnData.varm

AnnData.**varm**
Multi-dimensional annotation of variables/ features (mutable structured np.ndarray).

Stores for each key, a two or higher-dimensional np.ndarray of length n_vars. Is sliced with data and var but behaves otherwise like a dict.

Methods

chunk_X([select, replace])	Return a chunk of the data matrix *.X* with random or specified indices.
chunked_X([chunk_size])	Return an iterator over the rows of the data matrix *.X*.
concatenate(*adatas[, join, batch_key, ...])	Concatenate along the observations axis.
copy([filename])	Full copy, optionally on disk.
obs_keys()	List keys of observation annotation .obs.
obs_names_make_unique([join])	Makes the index unique by appending '1', '2', etc.
obsm_keys()	List keys of observation annotation obsm.
rename_categories(key, categories)	Rename categories of annotation *key* in *.obs*, *.var* and *.uns*.
transpose()	Transpose whole object.
uns_keys()	List keys of unstructured annotation.
var_keys()	List keys of variable annotation var.
var_names_make_unique([join])	Makes the index unique by appending '1', '2', etc.
varm_keys()	List keys of variable annotation varm.
write([filename, compression, ...])	Write *.h5ad*-formatted hdf5 file and close a potential backing file.
write_csvs(dirname[, skip_data, sep])	Write annotation to *.csv* files.
write_loom(filename)	Write *.loom*-formatted hdf5 file.
write_zarr(store, chunks)	

1.1.18 anndata.AnnData.chunk_X

AnnData.**chunk_X**(*select=1000*, *replace=True*)
Return a chunk of the data matrix *.X* with random or specified indices.

Parameters

select : Union[int, List[int], Tuple[int], ndarray] If select is an integer, a random chunk of row size = select will be returned. If select is a list, tuple or numpy array of integers, then a chunk with these indices will be returned.

replace : bool If select is an integer then replace = True specifies random sampling of indices with replacement, replace = False - without replacement.

1.1.19 anndata.AnnData.chunked_X

AnnData.**chunked_X**(*chunk_size=None*)
Return an iterator over the rows of the data matrix *.X*.

Parameters

chunk_size : Optional[int] Row size of a single chunk.

1.1.20 anndata.AnnData.concatenate

AnnData.**concatenate**(**adatas*, *join='inner'*, *batch_key='batch'*, *batch_categories=None*, *index_unique='-'*)

Concatenate along the observations axis.

The `.uns`, `.varm` and `.obsm` attributes are ignored.

Currently, this works only in `'memory'` mode.

Parameters

adatas : AnnData AnnData matrices to concatenate with. Each matrix is referred to as a "batch".

join : str Use intersection (`'inner'`) or union (`'outer'`) of variables.

batch_key : str Add the batch annotation to `.obs` using this key.

batch_categories : Optional[Sequence[Any]] Use these as categories for the batch annotation. By default, use increasing numbers.

index_unique : Optional[str] Make the index unique by joining the existing index names with the batch category, using `index_unique='-'`, for instance. Provide `None` to keep existing indices.

Returns The concatenated *AnnData*, where `adata.obs[batch_key]` stores a categorical variable labeling the batch.

Return type *AnnData*

Notes

Warning: If you use `join='outer'` this fills 0s for sparse data when variables are absent in a batch. Use this with care. Dense data is filled with ‘‘NaN‘‘s. See the examples.

Examples

Joining on intersection of variables.

```
>>> adata1 = AnnData(np.array([[1, 2, 3], [4, 5, 6]]),
>>>                  {'obs_names': ['s1', 's2'],
>>>                   'anno1': ['c1', 'c2']},
>>>                  {'var_names': ['a', 'b', 'c'],
>>>                   'annoA': [0, 1, 2]})
>>> adata2 = AnnData(np.array([[1, 2, 3], [4, 5, 6]]),
>>>                  {'obs_names': ['s3', 's4'],
>>>                   'anno1': ['c3', 'c4']},
>>>                  {'var_names': ['d', 'c', 'b'],
>>>                   'annoA': [0, 1, 2]})
>>> adata3 = AnnData(np.array([[1, 2, 3], [4, 5, 6]]),
>>>                  {'obs_names': ['s1', 's2'],
>>>                   'anno2': ['d3', 'd4']},
>>>                  {'var_names': ['d', 'c', 'b'],
>>>                   'annoA': [0, 2, 3],
>>>                   'annoB': [0, 1, 2]})
>>>
```

(continues on next page)

(continued from previous page)

```
>>> adata = adata1.concatenate(adata2, adata3)
>>> adata
AnnData object with n_obs × n_vars = 6 × 2
    obs_keys = ['anno1', 'anno2', 'batch']
    var_keys = ['annoA-0', 'annoA-1', 'annoB-2', 'annoA-2']
>>> adata.X
array([[2., 3.],
       [5., 6.],
       [3., 2.],
       [6., 5.],
       [3., 2.],
       [6., 5.]], dtype=float32)
>>> adata.obs
     anno1 anno2 batch
s1-0    c1   NaN     0
s2-0    c2   NaN     0
s3-1    c3   NaN     1
s4-1    c4   NaN     1
s1-2   NaN    d3     2
s2-2   NaN    d4     2
>>> adata.var.T
         b  c
annoA-0  1  2
annoA-1  2  1
annoB-2  2  1
annoA-2  3  2
```

Joining on the union of variables.

```
>>> adata = adata1.concatenate(adata2, adata3, join='outer')
>>> adata
AnnData object with n_obs × n_vars = 6 × 4
    obs_keys = ['anno1', 'anno2', 'batch']
    var_keys = ['annoA-0', 'annoA-1', 'annoB-2', 'annoA-2']
>>> adata.var.T
index      a    b    c    d
annoA-0  0.0  1.0  2.0  NaN
annoA-1  NaN  2.0  1.0  0.0
annoB-2  NaN  2.0  1.0  0.0
annoA-2  NaN  3.0  2.0  0.0
>>> adata.var_names
Index(['a', 'b', 'c', 'd'], dtype='object')
>>> adata.X
array([[ 1.,  2.,  3., nan],
       [ 4.,  5.,  6., nan],
       [nan,  3.,  2.,  1.],
       [nan,  6.,  5.,  4.],
       [nan,  3.,  2.,  1.],
       [nan,  6.,  5.,  4.]], dtype=float32)
>>> adata.X.sum(axis=0)
array([nan, 25., 23., nan], dtype=float32)
>>> import pandas as pd
>>> Xdf = pd.DataFrame(adata.X, columns=adata.var_names)
index    a    b    c    d
0      1.0  2.0  3.0  NaN
1      4.0  5.0  6.0  NaN
2      NaN  3.0  2.0  1.0
```

(continues on next page)

(continued from previous page)

```
3      NaN  6.0  5.0  4.0
4      NaN  3.0  2.0  1.0
5      NaN  6.0  5.0  4.0
>>> Xdf.sum()
index
a     5.0
b    25.0
c    23.0
d    10.0
dtype: float32
>>> from numpy import ma
>>> adata.X = ma.masked_invalid(adata.X)
>>> adata.X
masked_array(
  data=[[1.0, 2.0, 3.0, --],
        [4.0, 5.0, 6.0, --],
        [--, 3.0, 2.0, 1.0],
        [--, 6.0, 5.0, 4.0],
        [--, 3.0, 2.0, 1.0],
        [--, 6.0, 5.0, 4.0]],
  mask=[[False, False, False,  True],
        [False, False, False,  True],
        [ True, False, False, False],
        [ True, False, False, False],
        [ True, False, False, False],
        [ True, False, False, False]],
  fill_value=1e+20,
  dtype=float32)
>>> adata.X.sum(axis=0).data
array([ 5., 25., 23., 10.], dtype=float32)
```

The masked array is not saved but has to be reinstantiated after saving.

```
>>> adata.write('./test.h5ad')
>>> from anndata import read_h5ad
>>> adata = read_h5ad('./test.h5ad')
>>> adata.X
array([[ 1.,  2.,  3., nan],
       [ 4.,  5.,  6., nan],
       [nan,  3.,  2.,  1.],
       [nan,  6.,  5.,  4.],
       [nan,  3.,  2.,  1.],
       [nan,  6.,  5.,  4.]], dtype=float32)
```

For sparse data, everything behaves similarly, except that for *join='outer'*, zeros are added.

```
>>> from scipy.sparse import csr_matrix
>>> adata1 = AnnData(csr_matrix([[0, 2, 3], [0, 5, 6]]),
>>>                  {'obs_names': ['s1', 's2'],
>>>                   'anno1': ['c1', 'c2']},
>>>                  {'var_names': ['a', 'b', 'c']})
>>> adata2 = AnnData(csr_matrix([[0, 2, 3], [0, 5, 6]]),
>>>                  {'obs_names': ['s3', 's4'],
>>>                   'anno1': ['c3', 'c4']},
>>>                  {'var_names': ['d', 'c', 'b']})
>>> adata3 = AnnData(csr_matrix([[1, 2, 0], [0, 5, 6]]),
>>>                  {'obs_names': ['s5', 's6'],
```

(continues on next page)

(continued from previous page)

```
>>>                   'anno2': ['d3', 'd4']},
>>>                  {'var_names': ['d', 'c', 'b']})
>>>
>>> adata = adata1.concatenate(adata2, adata3, join='outer')
>>> adata.var_names
Index(['a', 'b', 'c', 'd'], dtype='object')
>>> adata.X.toarray()
array([[0., 2., 3., 0.],
       [0., 5., 6., 0.],
       [0., 3., 2., 0.],
       [0., 6., 5., 0.],
       [0., 0., 2., 1.],
       [0., 6., 5., 0.]], dtype=float32)
```

1.1.21 anndata.AnnData.copy

AnnData.**copy**(*filename=None*)
Full copy, optionally on disk.

1.1.22 anndata.AnnData.obs_keys

AnnData.**obs_keys**()
List keys of observation annotation `.obs`.

1.1.23 anndata.AnnData.obs_names_make_unique

AnnData.**obs_names_make_unique**(*join='-'*)
Makes the index unique by appending '1', '2', etc.

The first occurance of a non-unique value is ignored.

Parameters

join : *str*, optional (default: '') The connecting string between name and integer.

Examples

```
>>> adata1 = sc.AnnData(np.ones((3, 2)), {'obs_names': ['a', 'b', 'c']})
>>> adata2 = sc.AnnData(np.zeros((3, 2)), {'obs_names': ['d', 'b', 'b']})
>>> adata = adata1.concatenate(adata2)
>>> adata.obs_names
Index(['a', 'b', 'c', 'd', 'b', 'b'], dtype='object')
>>> adata.obs_names_make_unique()
>>> adata.obs_names
Index(['a', 'b', 'c', 'd', 'b-1', 'b-2'], dtype='object')
```

1.1.24 anndata.AnnData.obsm_keys

AnnData.**obsm_keys**()
List keys of observation annotation `obsm`.

1.1.25 anndata.AnnData.rename_categories

AnnData.**rename_categories**(*key*, *categories*)
Rename categories of annotation *key* in *.obs*, *.var* and *.uns*.

Besides calling `self.obs[key].cat.rename_categories(categories)` - similar for *.var* - this also renames categories in unstructured annotation that uses the categorical annotation *key*.

1.1.26 anndata.AnnData.transpose

AnnData.**transpose**()
Transpose whole object.

Data matrix is transposed, observations and variables are interchanged.

1.1.27 anndata.AnnData.uns_keys

AnnData.**uns_keys**()
List keys of unstructured annotation.

1.1.28 anndata.AnnData.var_keys

AnnData.**var_keys**()
List keys of variable annotation `var`.

1.1.29 anndata.AnnData.var_names_make_unique

AnnData.**var_names_make_unique**(*join='-'*)
Makes the index unique by appending '1', '2', etc.

The first occurance of a non-unique value is ignored.

Parameters

join : *str*, optional (default: '') The connecting string between name and integer.

Examples

```
>>> adata1 = sc.AnnData(np.ones((3, 2)), {'obs_names': ['a', 'b', 'c']})
>>> adata2 = sc.AnnData(np.zeros((3, 2)), {'obs_names': ['d', 'b', 'b']})
>>> adata = adata1.concatenate(adata2)
>>> adata.obs_names
Index(['a', 'b', 'c', 'd', 'b', 'b'], dtype='object')
>>> adata.obs_names_make_unique()
>>> adata.obs_names
Index(['a', 'b', 'c', 'd', 'b-1', 'b-2'], dtype='object')
```

1.1.30 anndata.AnnData.varm_keys

AnnData.**varm_keys**()
List keys of variable annotation `varm`.

1.1.31 anndata.AnnData.write

AnnData.**write**(*filename=None*, *compression='gzip'*, *compression_opts=None*, *force_dense=None*)
Write *.h5ad*-formatted hdf5 file and close a potential backing file.

Parameters

filename : Optional[PathLike] Filename of data file. Defaults to backing file.

compression : None or {'gzip', 'lzf'} See the h5py filter pipeline.

compression_opts : Union[int, Any, None] See the h5py filter pipeline.

1.1.32 anndata.AnnData.write_csvs

AnnData.**write_csvs**(*dirname*, *skip_data=True*, *sep=', '*)
Write annotation to *.csv* files.

It is not possible to recover the full *AnnData* from the output of this function. Use `write()` for this.

Parameters

dirname : PathLike Name of directory to which to export.

skip_data : bool Skip the data matrix *.X*.

sep : str Separator for the data.

1.1.33 anndata.AnnData.write_loom

AnnData.**write_loom**(*filename*)
Write *.loom*-formatted hdf5 file.

Parameters

filename : PathLike The filename.

1.1.34 anndata.AnnData.write_zarr

AnnData.**write_zarr**(*store*, *chunks*)

1.2 Reading

Reading anndata's native file format *.h5ad*.

read_h5ad(filename[, backed])	Read `.h5ad`-formatted hdf5 file.

1.2.1 anndata.read_h5ad

anndata.**read_h5ad**(*filename*, *backed=False*)
Read `.h5ad`-formatted hdf5 file.

Parameters

filename File name of data file.

backed : {False, True, 'r', 'r+'} Load *AnnData* in *backed* mode instead of fully loading it into memory (*memory* mode). *True* and 'r' are equivalent. If you want to modify backed attributes of the AnnData object, you need to choose 'r+'.

Reading other file formats.

read_csv(filename[, delimiter, ...])	Read `.csv` file.
read_excel(filename, sheet)	Read `.xlsx` (Excel) file.
read_hdf(filename, key)	Read `.h5` (hdf5) file.
read_loom(filename[, sparse, cleanup, ...])	Read `.loom`-formatted hdf5 file.
read_mtx(filename[, dtype])	Read `.mtx` file.
read_text(filename[, delimiter, ...])	Read `.txt`, `.tab`, `.data` (text) file.
read_umi_tools(filename)	Read a gzipped condensed count matrix from umi_tools.
read_zarr(store)	

1.2.2 anndata.read_csv

anndata.**read_csv**(*filename*, *delimiter=','*, *first_column_names=None*, *dtype='float32'*)

Read `.csv` file.

Same as *read_text()* but with default delimiter `','`.

Parameters

filename : Union[PathLike, Iterator[str]] Data file.

delimiter : Optional[str] Delimiter that separates data within text file. If *None*, will split at arbitrary number of white spaces, which is different from enforcing splitting at single white space ' '.

first_column_names : Optional[bool] Assume the first column stores row names.

dtype : str Numpy data type.

Return type AnnData

1.2.3 anndata.read_excel

anndata.**read_excel**(*filename*, *sheet*)

Read `.xlsx` (Excel) file.

Assumes that the first columns stores the row names and the first row the column names.

Parameters

filename : PathLike File name to read from.

sheet : Union[str, int] Name of sheet in Excel file.

Return type AnnData

1.2.4 anndata.read_hdf

anndata.**read_hdf**(*filename*, *key*)

Read `.h5` (hdf5) file.

Note: Also looks for fields `row_names` and `col_names`.

Parameters

filename : PathLike Filename of data file.

key : str Name of dataset in the file.

Return type AnnData

1.2.5 anndata.read_loom

anndata.**read_loom**(*filename*, *sparse=True*, *cleanup=False*, *X_name='spliced'*, *obs_names='CellID'*, *var_names='Gene'*)

Read `.loom`-formatted hdf5 file.

This reads the whole file into memory.

Beware that you have to explicitly state when you want to read the file as sparse data.

Parameters

filename : PathLike The filename.

sparse : bool Whether to read the data matrix as sparse.

Return type AnnData

1.2.6 anndata.read_mtx

anndata.**read_mtx**(*filename*, *dtype='float32'*)

Read `.mtx` file.

Parameters

filename : PathLike The filename.

dtype : str Numpy data type.

Return type AnnData

1.2.7 anndata.read_text

anndata.**read_text**(*filename*, *delimiter=None*, *first_column_names=None*, *dtype='float32'*)

Read `.txt`, `.tab`, `.data` (text) file.

Same as *read_csv()* but with default delimiter `None`.

Parameters

filename : Union[PathLike, Iterator[str]] Data file, filename or stream.

delimiter : Optional[str] Delimiter that separates data within text file. If *None*, will split at arbitrary number of white spaces, which is different from enforcing splitting at single white space ' '.

first_column_names : Optional[bool] Assume the first column stores row names.

dtype : str Numpy data type.

Return type AnnData

1.2.8 anndata.read_umi_tools

anndata.**read_umi_tools**(*filename*)
Read a gzipped condensed count matrix from umi_tools.

Parameters

filename : PathLike File name to read from.

Return type AnnData

1.2.9 anndata.read_zarr

anndata.**read_zarr**(*store*)

1.3 Writing

Writing to anndata's native file format *.h5ad*.

AnnData.write([filename, compression, ...])	Write *.h5ad*-formatted hdf5 file and close a potential backing file.

Writing to other formats.

AnnData.write_csvs(dirname[, skip_data, sep])	Write annotation to *.csv* files.
AnnData.write_loom(filename)	Write *.loom*-formatted hdf5 file.

1.4 h5py

Independent of *AnnData*, the submodule *anndata.h5py* provides a thin wrapper of h5py that is able to handle sparse matrices in addition to the standard functionality.

h5py	Wraps h5py to handle sparse matrices.

1.4.1 anndata.h5py

Wraps h5py to handle sparse matrices.

anndata.h5py is based on and uses the conventions of h5sparse by Appier Inc.. See the copyright and license note in the source code.

The design choices of *anndata.h5py*, however, are different. In particular, *anndata.h5py* allows handling sparse and non-sparse data at the same time. It achieves this by extending the functionality of *File* and *Group* objects in the h5py API, and by providing a new *SparseDataset* object.

For examples and further information, see this blog post.

File(name[, mode, driver, libver, ...])	Like h5py.File, but able to handle sparse matrices.
Group(h5py_group[, force_dense])	Like h5py.Group, but able to handle sparse matrices.
Dataset	Equivalent to h5py.Dataset.
SparseDataset(h5py_group)	Analogous to h5py.Dataset, but for sparse matrices.

anndata.h5py.File

class anndata.h5py.**File**(*name*, *mode=None*, *driver=None*, *libver=None*, *userblock_size=None*, *swmr=False*, *force_dense=False*, ***kwds*)

Like h5py.File, but able to handle sparse matrices.

Attributes

filename	
id	

anndata.h5py.File.filename

File.**filename**

anndata.h5py.File.id

File.**id**

Methods

close()	
create_dataset(name[, data, chunk_size])	Create a new HDF5 dataset
keys()	

anndata.h5py.File.close

File.**close**()

anndata.h5py.File.create_dataset

File.**create_dataset**(*name*, *data=None*, *chunk_size=6000*, ***kwargs*)

Create a new HDF5 dataset

name Name of the dataset (absolute or relative). Provide None to make an anonymous dataset.

shape Dataset shape. Use "()" for scalar datasets. Required if "data" isn't provided.

dtype Numpy dtype or string. If omitted, dtype('f') will be used. Required if "data" isn't provided; otherwise, overrides data array's dtype.

data Provide data to initialize the dataset. If used, you can omit shape and dtype arguments.

Keyword-only arguments:

chunks (Tuple) Chunk shape, or True to enable auto-chunking.

maxshape (Tuple) Make the dataset resizable up to this shape. Use None for axes you want to be unlimited.

compression (String or int) Compression strategy. Legal values are 'gzip', 'szip', 'lzf'. If an integer in range(10), this indicates gzip compression level. Otherwise, an integer indicates the number of a dynamically loaded compression filter.

compression_opts Compression settings. This is an integer for gzip, 2-tuple for szip, etc. If specifying a dynamically loaded compression filter number, this must be a tuple of values.

scaleoffset (Integer) Enable scale/offset filter for (usually) lossy compression of integer or floating-point data. For integer data, the value of scaleoffset is the number of bits to retain (pass 0 to let HDF5 determine the minimum number of bits necessary for lossless compression). For floating point data, scaleoffset is the number of digits after the decimal place to retain; stored values thus have absolute error less than 0.5*10**(-scaleoffset).

shuffle (T/F) Enable shuffle filter.

fletcher32 (T/F) Enable fletcher32 error detection. Not permitted in conjunction with the scale/offset filter.

fillvalue (Scalar) Use this value for uninitialized parts of the dataset.

track_times (T/F) Enable dataset creation timestamps.

anndata.h5py.File.keys

`File.`**`keys`**`()`

anndata.h5py.Group

class `anndata.h5py.`**`Group`**(*h5py_group*, *force_dense=False*)

Like h5py.Group, but able to handle sparse matrices.

Methods

create_dataset(name[, data, chunk_size])	Create a new HDF5 dataset
keys()	

anndata.h5py.Group.create_dataset

`Group.`**`create_dataset`**(*name*, *data=None*, *chunk_size=6000*, ***kwargs*)

Create a new HDF5 dataset

name Name of the dataset (absolute or relative). Provide None to make an anonymous dataset.

shape Dataset shape. Use "()" for scalar datasets. Required if "data" isn't provided.

dtype Numpy dtype or string. If omitted, dtype('f') will be used. Required if "data" isn't provided; otherwise, overrides data array's dtype.

data Provide data to initialize the dataset. If used, you can omit shape and dtype arguments.

Keyword-only arguments:

chunks (Tuple) Chunk shape, or True to enable auto-chunking.

maxshape (Tuple) Make the dataset resizable up to this shape. Use None for axes you want to be unlimited.

compression (String or int) Compression strategy. Legal values are 'gzip', 'szip', 'lzf'. If an integer in range(10), this indicates gzip compression level. Otherwise, an integer indicates the number of a dynamically loaded compression filter.

compression_opts Compression settings. This is an integer for gzip, 2-tuple for szip, etc. If specifying a dynamically loaded compression filter number, this must be a tuple of values.

scaleoffset (Integer) Enable scale/offset filter for (usually) lossy compression of integer or floating-point data. For integer data, the value of scaleoffset is the number of bits to retain (pass 0 to let HDF5 determine the minimum number of bits necessary for lossless compression). For floating point data, scaleoffset is the number of digits after the decimal place to retain; stored values thus have absolute error less than 0.5*10**(-scaleoffset).

shuffle (T/F) Enable shuffle filter.

fletcher32 (T/F) Enable fletcher32 error detection. Not permitted in conjunction with the scale/offset filter.

fillvalue (Scalar) Use this value for uninitialized parts of the dataset.

track_times (T/F) Enable dataset creation timestamps.

anndata.h5py.Group.keys

`Group.`**`keys`**`()`

anndata.h5py.Dataset

class anndata.h5py.**Dataset**

Equivalent to h5py.Dataset.

Attributes

attrs	Attributes attached to this object
chunks	Dataset chunks (or None)
compression	Compression strategy (or None)
compression_opts	Compression setting.
dims	Access dimension scales attached to this dataset.
dtype	Numpy dtype representing the datatype
file	Return a File instance associated with this object
fillvalue	Fill value for this dataset (0 by default)
fletcher32	Fletcher32 filter is present (T/F)
flush	Flush the dataset data and metadata to the file.
id	Low-level identifier appropriate for this object
maxshape	Shape up to which this dataset can be resized.
name	Return the full name of this object.

Continued on next page

Table 13 – continued from previous page

ndim	Numpy-style attribute giving the number of dimensions
parent	Return the parent group of this object.
ref	An (opaque) HDF5 reference to this object
refresh	Refresh the dataset metadata by reloading from the file.
regionref	Create a region reference (Datasets only).
scaleoffset	Scale/offset filter settings.
shape	Numpy-style shape tuple giving dataset dimensions
shuffle	Shuffle filter present (T/F)
size	Numpy-style attribute giving the total dataset size
value	Alias for dataset[()]

anndata.h5py.Dataset.attrs

Dataset.**attrs**
Attributes attached to this object

anndata.h5py.Dataset.chunks

Dataset.**chunks**
Dataset chunks (or None)

anndata.h5py.Dataset.compression

Dataset.**compression**
Compression strategy (or None)

anndata.h5py.Dataset.compression_opts

Dataset.**compression_opts**
Compression setting. Int(0-9) for gzip, 2-tuple for szip.

anndata.h5py.Dataset.dims

Dataset.**dims**
Access dimension scales attached to this dataset.

anndata.h5py.Dataset.dtype

Dataset.**dtype**
Numpy dtype representing the datatype

anndata.h5py.Dataset.file

Dataset.**file**
Return a File instance associated with this object

anndata.h5py.Dataset.fillvalue

Dataset.**fillvalue**
Fill value for this dataset (0 by default)

anndata.h5py.Dataset.fletcher32

Dataset.**fletcher32**
Fletcher32 filter is present (T/F)

anndata.h5py.Dataset.flush

Dataset.**flush**
Flush the dataset data and metadata to the file. If the dataset is chunked, raw data chunks are written to the file.

This is part of the SWMR features and only exist when the HDF5 library version >=1.9.178

anndata.h5py.Dataset.id

Dataset.**id**
Low-level identifier appropriate for this object

anndata.h5py.Dataset.maxshape

Dataset.**maxshape**
Shape up to which this dataset can be resized. Axes with value None have no resize limit.

anndata.h5py.Dataset.name

Dataset.**name**
Return the full name of this object. None if anonymous.

anndata.h5py.Dataset.ndim

Dataset.**ndim**
Numpy-style attribute giving the number of dimensions

anndata.h5py.Dataset.parent

Dataset.**parent**
Return the parent group of this object.

This is always equivalent to obj.file[posixpath.dirname(obj.name)]. ValueError if this object is anonymous.

anndata.h5py.Dataset.ref

Dataset.**ref**
An (opaque) HDF5 reference to this object

anndata.h5py.Dataset.refresh

Dataset.**refresh**
Refresh the dataset metadata by reloading from the file.

This is part of the SWMR features and only exist when the HDF5 library version >=1.9.178

anndata.h5py.Dataset.regionref

Dataset.**regionref**
Create a region reference (Datasets only).

The syntax is regionref[<slices>]. For example, dset.regionref[...] creates a region reference in which the whole dataset is selected.

Can also be used to determine the shape of the referenced dataset (via .shape property), or the shape of the selection (via the .selection property).

anndata.h5py.Dataset.scaleoffset

Dataset.**scaleoffset**
Scale/offset filter settings. For integer data types, this is the number of bits stored, or 0 for auto-detected. For floating point data types, this is the number of decimal places retained. If the scale/offset filter is not in use, this is None.

anndata.h5py.Dataset.shape

Dataset.**shape**
Numpy-style shape tuple giving dataset dimensions

anndata.h5py.Dataset.shuffle

Dataset.**shuffle**
Shuffle filter present (T/F)

anndata.h5py.Dataset.size

Dataset.**size**
Numpy-style attribute giving the total dataset size

anndata.h5py.Dataset.value

Dataset.**value**
Alias for dataset[()]

Methods

astype(dtype)	Get a context manager allowing you to perform reads to a different destination type, e.g.:
len()	The size of the first axis.
read_direct(dest[, source_sel, dest_sel])	Read data directly from HDF5 into an existing NumPy array.
resize(size[, axis])	Resize the dataset, or the specified axis.
write_direct(source[, source_sel, dest_sel])	Write data directly to HDF5 from a NumPy array.

anndata.h5py.Dataset.astype

Dataset.**astype**(*dtype*)
Get a context manager allowing you to perform reads to a different destination type, e.g.:

```
>>> with dataset.astype('f8'):
...     double_precision = dataset[0:100:2]
```

anndata.h5py.Dataset.len

Dataset.**len**()
The size of the first axis. TypeError if scalar.

Use of this method is preferred to len(dset), as Python's built-in len() cannot handle values greater then 2**32 on 32-bit systems.

anndata.h5py.Dataset.read_direct

Dataset.**read_direct**(*dest*, *source_sel=None*, *dest_sel=None*)
Read data directly from HDF5 into an existing NumPy array.

The destination array must be C-contiguous and writable. Selections must be the output of numpy.s_[<args>].

Broadcasting is supported for simple indexing.

anndata.h5py.Dataset.resize

Dataset.**resize**(*size*, *axis=None*)
Resize the dataset, or the specified axis.

The dataset must be stored in chunked format; it can be resized up to the "maximum shape" (keyword maxshape) specified at creation time. The rank of the dataset cannot be changed.

"Size" should be a shape tuple, or if an axis is specified, an integer.

BEWARE: This functions differently than the NumPy resize() method! The data is not "reshuffled" to fit in the new shape; each axis is grown or shrunk independently. The coordinates of existing data are fixed.

anndata.h5py.Dataset.write_direct

Dataset.**write_direct**(*source*, *source_sel=None*, *dest_sel=None*)
Write data directly to HDF5 from a NumPy array.

The source array must be C-contiguous. Selections must be the output of numpy.s_[<args>].

Broadcasting is supported for simple indexing.

anndata.h5py.SparseDataset

class anndata.h5py.**SparseDataset**(*h5py_group*)
Analogous to h5py.Dataset, but for sparse matrices.

Attributes

dtype
format_str
shape
value

anndata.h5py.SparseDataset.dtype

SparseDataset.**dtype**

anndata.h5py.SparseDataset.format_str

SparseDataset.**format_str**

anndata.h5py.SparseDataset.shape

SparseDataset.**shape**

anndata.h5py.SparseDataset.value

SparseDataset.**value**

Methods

append(sparse_matrix)

anndata.h5py.SparseDataset.append

SparseDataset.**append**(*sparse_matrix*)

CHAPTER 2

Benchmarks

2.1 Reading and writing h5ad files

Of course, we want the associated notebook to be called “reading-writing.h5ad”.

https://github.com/Koncopd/anndata-scanpy-benchmarks/blob/master/Benchmarks-Loading.ipynb

The notebook should also cover reading chunks.

https://github.com/Koncopd/anndata-scanpy-benchmarks/blob/master/chunk_X.ipynb

It should compare the results with loom.

We don’t need images here, at this stage.

2.2 File sizes

Another notebook should cover file sizes.

https://github.com/Koncopd/anndata-scanpy-benchmarks/blob/master/File_sizes.ipynb

CHAPTER 3

References

Bibliography

[Hastie09] Hastie *et al.* (2009), *The Elements of Statistical Learning*, Springer.

[Huber15] Huber *et al.* (2015), *Orchestrating high-throughput genomic analysis with Bioconductor*, Nature Methods.

[Murphy12] Murphy (2012, *Machine Learning: A Probabilisitc Perspective*, MIT Press.

[Wolf17] Wolf *et al.* (2018), *Scanpy: large-scale single-cell gene expression data analysis*, Genome Biology.

Python Module Index

a

Index

S

T

U

V

W

X