Floating point error <u>handling</u>

Discrete Fourier Transform

(numpy.fft)

Functional programming

NumPy-specific help **functions**

Input and output

numpy.load

numpy.save

numpy.savez

numpy.savez compressed

numpy.loadtxt

numpy.savetxt

numpy.genfromtxt

numpy.fromregex

numpy.fromstring

numpy.ndarray.tofile

numpy.ndarray.tolist

numpy.array2string

<u>numpy.array_repr</u>

numpy.array_str

numpy.format float position

numpy.format float scientif

numpy.memmap

numpy.lib.format.open men

numpy.set printoptions

numpy.get printoptions

numpy.set string function

numpy.printoptions

numpy.binary repr

numpy.base repr

numpy.DataSource

numpy.lib.format

Linear algebra (

numpy.linalg_)

Logic functions

Masked array operations

Mathematical functions

Matrix library (

numpy.matlib)

Miscellaneous routines

Padding Arrays

numpy.load

numpy.load(file, mmap_mode=None, allow_pickle=False, fix_imports=True, encoding='ASCII')

[source]

Load arrays or pickled objects from .npy, .npz or pickled files.

Warning

Loading files that contain object arrays uses the pickle module, which is not secure against erroneous or maliciously constructed data. Consider passing allow_pickle=False to load data that is known not to contain object arrays for the safer handling of untrusted sources.

Parameters: file: file-like object, string, or pathlib.Path

The file to read. File-like objects must support the seek() and read() methods. Pickled files require that the file-like object support the readline() method as well.

mmap_mode : {None, 'r+', 'r', 'w+', 'c'}, optional

If not None, then memory-map the file, using the given mode (see numpy.memmap for a detailed description of the modes). A memory-mapped array is kept on disk. However, it can be accessed and sliced like any ndarray. Memory mapping is especially useful for accessing small fragments of large files without reading the entire file into memory.

allow_pickle: bool, optional

Allow loading pickled object arrays stored in npy files. Reasons for disallowing pickles include security, as loading pickled data can execute arbitrary code. If pickles are disallowed, loading object arrays will fail. Default: False Changed in version 1.16.3: Made default False in response to CVE-2019-6446.

fix_imports: bool, optional

Only useful when loading Python 2 generated pickled files on Python 3, which includes npy/npz files containing object arrays. If fix_imports is True, pickle will try to map the old Python 2 names to the new names used in Python 3.

encoding: str, optional

What encoding to use when reading Python 2 strings. Only useful when loading Python 2 generated pickled files in Python 3, which includes npy/npz files containing object arrays. Values other than 'latin1', 'ASCII', and 'bytes' are not allowed, as they can corrupt numerical data. Default: 'ASCII'

Returns: result: array, tuple, dict, etc.

> Data stored in the file. For .npz files, the returned instance of NpzFile class must be closed to avoid leaking file descriptors.

Raises: OSError

If the input file does not exist or cannot be read.

UnpicklingError

If allow pickle=True, but the file cannot be loaded as a pickle.

ValueError

The file contains an object array, but allow_pickle=False given.

```
See also
```

```
save, savez, savez_compressed, loadtxt
```

<u>memmap</u>

Create a memory-map to an array stored in a file on disk.

lib.format.open_memmap

Create or load a memory-mapped .npy file.

Notes

- If the file contains pickle data, then whatever object is stored in the pickle is returned.
- If the file is a .npy file, then a single array is returned.
- If the file is a .npz file, then a dictionary-like object is returned, containing {filename: array} key-value pairs, one for each file in the archive.
- If the file is a .npz file, the returned value supports the context manager protocol in a similar fashion to the open function:

```
with load('foo.npz') as data:
    a = data['a']
```

The underlying file descriptor is closed when exiting the 'with' block.

Examples

Store data to disk, and load it again:

Store compressed data to disk, and load it again:

Mem-map the stored array, and then access the second row directly from disk:

```
>>> X = np.load('/tmp/123.npy', mmap_mode='r')
>>> X[1, :]
memmap([4, 5, 6])
```

Previous
Input and output

numpy.save >