Search the docs ...

Clustering package (scipy.cluster)
K-means clustering and vector quantization (scipy.cluster.vq)
Hierarchical clustering (scipy.cluster.hierarchy)
Constants (scipy.constants)
Discrete Fourier transforms (scipy.fft)
Legacy discrete Fourier transforms (scipy.fftpack)
Integration and ODEs (scipy.integrate)
Interpolation (scipy.interpolate)
Input and output (scipy.io)
Linear algebra (scipy.linalg)
Low-level BLAS functions (scipy.linalg.blas)
Low-level LAPACK functions (scipy.linalg.lapack)
BLAS Functions for Cython
LAPACK functions for Cython
Interpolative matrix decomposition (scipy.linalg.interpolative)
Miscellaneous routines (scipy.misc)
Multidimensional image processing (scipy.ndimage)

SciPy API

Importing from SciPy

In Python the distinction between what is the public API of a library and what are private implementation details is not always clear. Unlike in other languages like Java, it is possible in Python to access “private” function or objects. Occasionally this may be convenient, but be aware that if you do so your code may break without warning in future releases. Some widely understood rules for what is and isn't public in Python are:

- Methods / functions / classes and module attributes whose names begin with a leading underscore are private.
- If a class name begins with a leading underscore, none of its members are public, whether or not they begin with a leading underscore.
- If a module name in a package begins with a leading underscore none of its members are public, whether or not they begin with a leading underscore.
- If a module or package defines `__all__`, that authoritatively defines the public interface.
- If a module or package doesn't define `__all__`, then all names that don't start with a leading underscore are public.

> **Note**
>
> Reading the above guidelines one could draw the conclusion that every private module or object starts with an underscore. This is not the case; the presence of underscores do mark something as private, but the absence of underscores do not mark something as public.

In SciPy there are modules whose names don't start with an underscore, but that should be considered private. To clarify which modules these are, we define below what the public API is for SciPy, and give some recommendations for how to import modules/functions/objects from SciPy.

Guidelines for importing functions from SciPy

The scipy namespace itself only contains functions imported from numpy. These functions still exist for backwards compatibility, but should be imported from numpy directly.

Everything in the namespaces of scipy submodules is public. In general, it is recommended to import functions from submodule namespaces. For example, the function `curve_fit` (defined in scipy/optimize/_minpack_py.py) should be imported like this:

```
from scipy import optimize
result = optimize.curve_fit(...)
```

This form of importing submodules is preferred for all submodules except `scipy.io` (because `io` is also the name of a module in the Python stdlib):

```
from scipy import interpolate
from scipy import integrate
import scipy.io as spio
```

In some cases, the public API is one level deeper. For example, the `scipy.sparse.linalg` module is public, and the functions it contains are not available in the `scipy.sparse` namespace. Sometimes it may result in more easily understandable code if functions are imported from one level deeper. For example, in the following it is immediately clear that `lomax` is a distribution if the second form is chosen:

```
# first form
from scipy import stats
stats.lomax(...)

# second form
from scipy.stats import distributions
distributions.lomax(...)
```

In that case, the second form can be chosen **if** it is documented in the next section that the submodule in question is public.

API definition

Every submodule listed below is public. That means that these submodules are unlikely to be renamed or changed in an incompatible way, and if that is necessary, a deprecation warning will be raised for one SciPy release before the change is made.

- **scipy.cluster**
  - **scipy.cluster.vq**
  - **scipy.cluster.hierarchy**
- **scipy.constants**
- **scipy.fft**
- **scipy.fftpack**
- **scipy.integrate**
- **scipy.interpolate**
- **scipy.io**
  - **scipy.io.arff**
  - **scipy.io.matlab**
  - **scipy.io.wavfile**
- **scipy.linalg**
  - **scipy.linalg.blas**
  - **scipy.linalg.cython_blas**
  - **scipy.linalg.lapack**
  - **scipy.linalg.cython_lapack**
  - **scipy.linalg.interpolative**
- **scipy.misc**
- **scipy.ndimage**
- **scipy.odr**
- **scipy.optimize**
- **scipy.signal**
  - **scipy.signal.windows**
- **scipy.sparse**
  - **scipy.sparse.linalg**
  - **scipy.sparse.csgraph**
- **scipy.spatial**
  - **scipy.spatial.distance**
  - **scipy.spatial.transform**
- **scipy.special**
- **scipy.stats**
  - **scipy.stats.contingency**
  - scipy.stats.distributions
  - **scipy.stats.mstats**
  - **scipy.stats.qmc**
  - **scipy.stats.sampling**

SciPy structure

All SciPy modules should follow the following conventions. In the following, a *SciPy module* is defined as a Python package, say `yyy`, that is located in the scipy/ directory.

- Ideally, each SciPy module should be as self-contained as possible. That is, it should have minimal dependencies on other packages or modules. Even dependencies on other SciPy modules should be kept to a minimum. A dependency on NumPy is of course assumed.
- Directory `yyy/` contains:
  - A file `setup.py` that defines `configuration(parent_package='',top_path=None)` function for *numpy.distutils*.
  - A directory `tests/` that contains files `test_<name>.py` corresponding to modules `yyy/<name>{.py,.so,/}`.
- Private modules should be prefixed with an underscore _, for instance `yyy/_somemodule.py`.
- User-visible functions should have good documentation following the NumPy documentation style.
- The `__init__.py` of the module should contain the main reference documentation in its docstring. This is connected to the Sphinx documentation under `doc/` via Sphinx's automodule directive.

  The reference documentation should first give a categorized list of the contents of the module using `autosummary::` directives, and after that explain points essential for understanding the use of the module.

  Tutorial-style documentation with extensive examples should be separate and put under `doc/source/tutorial/`.

See the existing SciPy submodules for guidance.

For further details on NumPy distutils, see NumPy Distutils - User's Guide.

Previous
File IO (scipy.io)

Next
Clustering package (scipy.cluster)

On this page

Importing from SciPy
Guidelines for importing functions from SciPy
API definition
SciPy structure