In [1]:

```python
from scipy import cluster
```

In [2]:

```python
help(cluster)
```

```
Help on package scipy.cluster in scipy:

NAME
    scipy.cluster

DESCRIPTION
    =========================================
    Clustering package (:mod:`scipy.cluster`)
    =========================================

    .. currentmodule:: scipy.cluster

    :mod:`scipy.cluster.vq`

    Clustering algorithms are useful in information theory, target detection,
    communications, compression, and other areas. The `vq` module only
    supports vector quantization and the k-means algorithms.

    :mod:`scipy.cluster.hierarchy`

    The `hierarchy` module provides functions for hierarchical and
    agglomerative clustering.  Its features include generating hierarchical
    clusters from distance matrices,
    calculating statistics on clusters, cutting linkages
    to generate flat clusters, and visualizing clusters with dendrograms.

PACKAGE CONTENTS
    _hierarchy
    _optimal_leaf_ordering
    _vq
    hierarchy
    setup
    tests (package)
    vq

DATA
    __all__ = ['vq', 'hierarchy']

FILE
    c:\users\dell\appdata\local\programs\python\python39\lib\site-packages\scipy\cluster\__init__.py
```

In [3]:

```python
help()
```

```
Welcome to Python 3.9's help utility!

If this is your first time using Python, you should definitely check out
the tutorial on the Internet at https://docs.python.org/3.9/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules.  To quit this help utility and
return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics, type
"modules", "keywords", "symbols", or "topics".  Each module also comes
with a one-line summary of what it does; to list the modules whose name
or summary contain a given string such as "spam", type "modules spam".

help> scipy,cluster
No Python documentation found for 'scipy,cluster'.
Use help() to get the interactive help utility.
Use help(str) for help on the str class.

help> quit

You are now leaving help and returning to the Python interpreter.
If you want to ask for help on a particular object directly from the
interpreter, you can type "help(object)".  Executing "help('string')"
has the same effect as typing a particular string at the help> prompt.
```

In [4]:

```python
import scipy
scipy.info(cluster)
```

```
=========================================
Clustering package (:mod:`scipy.cluster`)
=========================================

.. currentmodule:: scipy.cluster

:mod:`scipy.cluster.vq`

Clustering algorithms are useful in information theory, target detection,
communications, compression, and other areas. The `vq` module only
supports vector quantization and the k-means algorithms.

:mod:`scipy.cluster.hierarchy`

The `hierarchy` module provides functions for hierarchical and
agglomerative clustering.  Its features include generating hierarchical
clusters from distance matrices,
calculating statistics on clusters, cutting linkages
to generate flat clusters, and visualizing clusters with dendrograms.
<ipython-input-4-ef44e9371b0d>:2: DeprecationWarning: scipy.info is deprecated and will be removed in SciPy 2.0.0, use numpy.info instead
  scipy.info(cluster)
```

In [5]:

```python
scipy.source(cluster)
```

```
In file: c:\users\dell\appdata\local\programs\python\python39\lib\site-packages\scipy\cluster\__init__.py

"""
=========================================
Clustering package (:mod:`scipy.cluster`)
=========================================

.. currentmodule:: scipy.cluster

:mod:`scipy.cluster.vq`

Clustering algorithms are useful in information theory, target detection,
communications, compression, and other areas. The `vq` module only
supports vector quantization and the k-means algorithms.

:mod:`scipy.cluster.hierarchy`

The `hierarchy` module provides functions for hierarchical and
agglomerative clustering.  Its features include generating hierarchical
clusters from distance matrices,
calculating statistics on clusters, cutting linkages
to generate flat clusters, and visualizing clusters with dendrograms.

"""
__all__ = ['vq', 'hierarchy']

from . import vq, hierarchy

from scipy._lib._testutils import PytestTester
test = PytestTester(__name__)
del PytestTester

<ipython-input-5-7c3327541f9e>:1: DeprecationWarning: scipy.source is deprecated and will be removed in SciPy 2.0.0, use numpy.source instead
  scipy.source(cluster)
```

Special

In [6]:

```python
from scipy import special
a=special.exp10(2)
print(a)
```

```
100.0
```

In [7]:

```python
a=special.exp2(3)
print(a)
```

```
8.0
```

In [8]:

```python
a=special.sindg(90)
print(a)
```

```
1.0
```

In [9]:

```python
a=special.cosdg(90)
print(a)
```

```
-0.0
```

Integration

In [10]:

```python
from scipy import integrate
help(integrate.quad)
```

```
Help on function quad in module scipy.integrate.quadpack:

quad(func, a, b, args=(), full_output=0, epsabs=1.49e-08, epsrel=1.49e-08, limit=50, points=None, weight=None, wvar=None, wopts=None, maxp1=50, limlst=50)
    Compute a definite integral.

    Integrate func from `a` to `b` (possibly infinite interval) using a
    technique from the Fortran library QUADPACK.

    Parameters
    ----------
    func : {function, scipy.LowLevelCallable}
        A Python function or method to integrate. If `func` takes many
        arguments, it is integrated along the axis corresponding to the
        first argument.

        If the user desires improved integration performance, then `f` may
        be a `scipy.LowLevelCallable` with one of the signatures::

            double func(double x)
            double func(double x, void *user_data)
            double func(int n, double *xx)
            double func(int n, double *xx, void *user_data)

        The ``user_data`` is the data contained in the `scipy.LowLevelCallable`.
        In the call forms with ``xx``,  ``n`` is the length of the ``xx``
        array which contains ``xx[0] == x`` and the rest of the items are
        numbers contained in the ``args`` argument of quad.

        In addition, certain ctypes call signatures are supported for
        backward compatibility, but those should not be used in new code.
    a : float
        Lower limit of integration (use -numpy.inf for -infinity).
    b : float
        Upper limit of integration (use numpy.inf for +infinity).
    args : tuple, optional
        Extra arguments to pass to `func`.
    full_output : int, optional
        Non-zero to return a dictionary of integration information.
        If non-zero, warning messages are also suppressed and the
        message is appended to the output tuple.

    Returns
    -------
    y : float
        The integral of func from `a` to `b`.
    abserr : float
        An estimate of the absolute error in the result.
    infodict : dict
        A dictionary containing additional information.
        Run scipy.integrate.quad_explain() for more information.
    message
        A convergence message.
    explain
        Appended only with 'cos' or 'sin' weighting and infinite
        integration limits, it contains an explanation of the codes in
        infodict['ierlst']

    Other Parameters
    ----------------
    epsabs : float or int, optional
        Absolute error tolerance. Default is 1.49e-8. `quad` tries to obtain
        an accuracy of ``abs(i-result) <= max(epsabs, epsrel*abs(i))``
        where ``i`` = integral of `func` from `a` to `b`, and ``result`` is the
        numerical approximation. See `epsrel` below.
    epsrel : float or int, optional
        Relative error tolerance. Default is 1.49e-8.
        If ``epsabs <= 0``, `epsrel` must be greater than both 5e-29
        and ``50 * (machine epsilon)``. See `epsabs` above.
    limit : float or int, optional
        An upper bound on the number of subintervals used in the adaptive
        algorithm.
    points : (sequence of floats,ints), optional
        A sequence of break points in the bounded integration interval
        where local difficulties of the integrand may occur (e.g.,
        singularities, discontinuities). The sequence does not have
        to be sorted. Note that this option cannot be used in conjunction
        with ``weight``.
    weight : float or int, optional
        String indicating weighting function. Full explanation for this
        and the remaining arguments can be found below.
    wvar : optional
        Variables for use with weighting functions.
    wopts : optional
        Optional input for reusing Chebyshev moments.
    maxp1 : float or int, optional
        An upper bound on the number of Chebyshev moments.
    limlst : int, optional
        Upper bound on the number of cycles (>=3) for use with a sinusoidal
        weighting and an infinite end-point.

    See Also
    --------
    dblquad : double integral
    tplquad : triple integral
    nquad : n-dimensional integrals (uses `quad` recursively)
    fixed_quad : fixed-order Gaussian quadrature
    quadrature : adaptive Gaussian quadrature
    odeint : ODE integrator
    ode : ODE integrator
    simpson : integrator for sampled data
    romb : integrator for sampled data
    scipy.special : for coefficients and roots of orthogonal polynomials

    Notes
    -----

    **Extra information for quad() inputs and outputs**

    If full_output is non-zero, then the third output argument
    (infodict) is a dictionary with entries as tabulated below. For
    infinite limits, the range is transformed to (0,1) and the
    optional outputs are given with respect to this transformed range.
    Let M be the input argument limit and let K be infodict['last'].
    The entries are:

    'neval'
        The number of function evaluations.
    'last'
        The number, K, of subintervals produced in the subdivision process.
    'alist'
        A rank-1 array of length M, the first K elements of which are the
        left end points of the subintervals in the partition of the
        integration range.
    'blist'
        A rank-1 array of length M, the first K elements of which are the
        right end points of the subintervals.
    'rlist'
        A rank-1 array of length M, the first K elements of which are the
        integral approximations on the subintervals.
    'elist'
        A rank-1 array of length M, the first K elements of which are the
        moduli of the absolute error estimates on the subintervals.
    'iord'
        A rank-1 integer array of length M, the first L elements of
        which are pointers to the error estimates over the subintervals
        with ``L=K`` if ``K<=M/2+2`` or ``L=M+1-K`` otherwise. Let I be the
        sequence ``infodict['iord']`` and let E be the sequence
        ``infodict['elist']``.  Then ``E[I[1]], ..., E[I[L]]`` forms a
        decreasing sequence.

    If the input argument points is provided (i.e., it is not None),
    the following additional outputs are placed in the output
    dictionary. Assume the points sequence is of length P.

    'pts'
        A rank-1 array of length P+2 containing the integration limits
        and the break points of the intervals in ascending order.
        This is an array giving the subintervals over which integration
        will occur.
    'level'
        A rank-1 integer array of length M (=limit), containing the
        subdivision levels of the subintervals, i.e., if (aa,bb) is a
        subinterval of ``(pts[1], pts[2])`` where ``pts[0]`` and ``pts[2]``
        are adjacent elements of ``infodict['pts']``, then (aa,bb) has level l
        if ``|bb-aa| = |pts[2]-pts[1]| * 2**(-l)``.
    'ndin'
        A rank-1 integer array of length P+2. After the first integration
        over the intervals (pts[1], pts[2]), the error estimates over some
        of the intervals may have been increased artificially in order to
        put their subdivision forward. This array has ones in slots
        corresponding to the subintervals for which this happens.

    **Weighting the integrand**

    The input variables, *weight* and *wvar*, are used to weight the
    integrand by a select list of functions. Different integration
    methods are used to compute the integral with these weighting
    functions, and these do not support specifying break points. The
    possible values of weight and the corresponding weighting functions are.

    ==========  ===================================   =====================
    ``weight``  Weight function used                  ``wvar``
    ==========  ===================================   =====================
    'cos'       cos(w*x)                              wvar = w
    'sin'       sin(w*x)                              wvar = w
    'alg'       g(x) = ((x-a)**alpha)*((b-x)**beta)   wvar = (alpha, beta)
    'alg-loga'  g(x)*log(x-a)                         wvar = (alpha, beta)
    'alg-logb'  g(x)*log(b-x)                         wvar = (alpha, beta)
    'alg-log'   g(x)*log(x-a)*log(b-x)                wvar = (alpha, beta)
    'cauchy'    1/(x-c)                               wvar = c
    ==========  ===================================   =====================

    wvar holds the parameter w, (alpha, beta), or c depending on the weight
    selected. In these expressions, a and b are the integration limits.

    For the 'cos' and 'sin' weighting, additional inputs and outputs are
    available.

    For finite integration limits, the integration is performed using a
    Clenshaw-Curtis method which uses Chebyshev moments. For repeated
    calculations, these moments are saved in the output dictionary:

    'momcom'
        The maximum level of Chebyshev moments that have been computed,
        i.e., if ``M_c`` is ``infodict['momcom']`` then the moments have been
        computed for intervals of length ``|b-a| * 2**(-l)``,
        ``l=0,1,...,M_c``.
    'nnlog'
        A rank-1 integer array of length M(=limit), containing the
        subdivision levels of the subintervals, i.e., an element of this
        array is equal to l if the corresponding subinterval is
        ``|b-a|* 2**(-l)``.
    'chebmo'
        A rank-2 array of shape (25, maxp1) containing the computed
        Chebyshev moments. These can be passed on to an integration
        over the same interval by passing this array as the second
        element of the sequence wopts and passing infodict['momcom'] as
        the first element.

    If one of the integration limits is infinite, then a Fourier integral is
    computed (assuming w neq 0). If full_output is 1 and a numerical error
    is encountered, besides the error message attached to the output tuple,
    a dictionary is also appended to the output tuple which translates the
    error codes in the array ``info['ierlst']`` to English messages. The
    output information dictionary contains the following entries instead of
    'last', 'alist', 'blist', 'rlist', and 'elist':

    'lst'
        The number of subintervals needed for the integration (call it ``K_f``).
    'rslst'
        A rank-1 array of length M_f=limlst, whose first ``K_f`` elements
        contain the integral contribution over the interval
        ``(a+(k-1)c, a+kc)`` where ``c = (2*floor(|w|) + 1) * pi / |w|``
        and ``k=1,2,...,K_f``.
    'erlst'
        A rank-1 array of length ``M_f`` containing the error estimate
        corresponding to the interval in the same position in
        ``infodict['rslist']``.
    'ierlst'
        A rank-1 integer array of length ``M_f`` containing an error flag
        corresponding to the interval in the same position in
        ``infodict['rslist']``.  See the explanation dictionary (last entry
        in the output tuple) for the meaning of the codes.

    Examples
    --------
    Calculate :math:`\int^4_0 x^2 dx` and compare with an analytic result

    >>> from scipy import integrate
    >>> x2 = lambda x: x**2
    >>> integrate.quad(x2, 0, 4)
    (21.333333333333332, 2.3684757858670003e-13)
    >>> print(4**3 / 3.)  # analytical result
    21.3333333333

    Calculate :math:`\int^\infty_0 e^{-x} dx`

    >>> invexp = lambda x: np.exp(-x)
    >>> integrate.quad(invexp, 0, np.inf)
    (1.0, 5.842605999138044e-11)

    >>> f = lambda x,a : a*x
    >>> y, err = integrate.quad(f, 0, 1, args=(1,))
    >>> y
    0.5
    >>> y, err = integrate.quad(f, 0, 1, args=(3,))
    >>> y
    1.5

    Calculate :math:`\int^1_0 x^2 + y^2 dx` with ctypes, holding
    y parameter as 1::

        testlib.c =>
            double func(int n, double args[n]){
                return args[0]*args[0] + args[1]*args[1];}
        compile to library testlib.*

    ::

       from scipy import integrate
       import ctypes
       lib = ctypes.CDLL('/home/.../testlib.*') #use absolute path
       lib.func.restype = ctypes.c_double
       lib.func.argtypes = (ctypes.c_int,ctypes.c_double)
       integrate.quad(lib.func,0,1,(1))
       #(1.3333333333333333, 1.4802973661668752e-14)
       print((1.0**3/3.0 + 1.0) - (0.0**3/3.0 + 0.0)) #Analytic result
       # 1.3333333333333333

    Be aware that pulse shapes and other sharp features as compared to the
    size of the integration interval may not be integrated correctly using
    this method. A simplified example of this limitation is integrating a
    y-axis reflected step function with many zero values within the integrals
    bounds.

    >>> y = lambda x: 1 if x<=0 else 0
    >>> integrate.quad(y, -1, 1)
    (1.0, 1.1102230246251565e-14)
    >>> integrate.quad(y, -1, 100)
    (1.0000000002199108, 1.0189464580163188e-08)
    >>> integrate.quad(y, -1, 10000)
    (0.0, 0.0)
```

In [11]:

```python
a=scipy.integrate.quad(lambda x:special.exp10(x),0,1)
print(a)
```

```
(3.9086503371292665, 4.3394735994897923e-14)
```

In [12]:

```python
a=lambda x,y:x*y**2
b=lambda x:1
c=lambda x:-1
integrate.dblquad(a,0,2,b,c)
```

Out[12]:

```
(-0.0, 4.405142707569776e-14)
```

Fourier Transformation

In [13]:

```python
from scipy.fftpack import fft,ifft
import numpy as np
x=np.array([1,2,3,4])
y=fft(x)
print(y)
```

```
[10.-0.j -2.+2.j -2.-0.j -2.-2.j]
```

In [14]:

```python
from scipy.fftpack import fft,ifft
import numpy as np
x=np.array([1,2,3,4])
y=ifft(x)
print(y)
```

```
[ 2.5-0.j  -0.5-0.5j -0.5-0.j  -0.5+0.5j]
```

Linear Algebra

In [15]:

```python
from scipy import linalg
a=np.array([[1,2],[3,4]])
b=linalg.inv(a)
print(b)
```

```
[[-2.   1. ]
 [ 1.5 -0.5]]
```

In [ ]:

```python

```