**Integration ([scipy.integrate](http://jiffyclub.github.io/scipy/integrate.html" \l "module-scipy.integrate" \o "scipy.integrate))**

The [scipy.integrate](http://jiffyclub.github.io/scipy/integrate.html#module-scipy.integrate) sub-package provides several integration techniques including an ordinary differential equation integrator. An overview of the module is provided by the help command:

>>> help(integrate)

Methods for Integrating Functions given function object.

quad -- General purpose integration.

dblquad -- General purpose double integration.

tplquad -- General purpose triple integration.

fixed\_quad -- Integrate func(x) using Gaussian quadrature of order n.

quadrature -- Integrate with given tolerance using Gaussian quadrature.

romberg -- Integrate func using Romberg integration.

Methods for Integrating Functions given fixed samples.

trapz -- Use trapezoidal rule to compute integral from samples.

cumtrapz -- Use trapezoidal rule to cumulatively compute integral.

simps -- Use Simpson's rule to compute integral from samples.

romb -- Use Romberg Integration to compute integral from

(2\*\*k + 1) evenly-spaced samples.

See the special module's orthogonal polynomials (special) for Gaussian

quadrature roots and weights for other weighting factors and regions.

Interface to numerical integrators of ODE systems.

odeint -- General integration of ordinary differential equations.

ode -- Integrate ODE using VODE and ZVODE routines.

**General integration (**[**quad**](http://jiffyclub.github.io/scipy/generated/scipy.integrate.quad.html#scipy.integrate.quad)**)**

The function [quad](http://jiffyclub.github.io/scipy/generated/scipy.integrate.quad.html#scipy.integrate.quad) is provided to integrate a function of one variable between two points. The points can be ±∞

(± inf) to indicate infinite limits. For example, suppose you wish to integrate a bessel function jv(2.5, x) along the interval [0,4.5].

*I*=∫4.50*J*2.5(*x*)*dx*.

This could be computed using [quad](http://jiffyclub.github.io/scipy/generated/scipy.integrate.quad.html#scipy.integrate.quad):

>>> import scipy.integrate as integrate

>>> import scipy.special as special

>>> result = integrate.quad(lambda x: special.jv(2.5,x), 0, 4.5)

>>> result

(1.1178179380783249, 7.8663172481899801e-09)

>>> from numpy import sqrt, sin, cos, pi

>>> I = sqrt(2/pi)\*(18.0/27\*sqrt(2)\*cos(4.5) - 4.0/27\*sqrt(2)\*sin(4.5) +

... sqrt(2\*pi) \* special.fresnel(3/sqrt(pi))[0])

>>> I

1.117817938088701

>>> print(abs(result[0]-I))

1.03761443881e-11

The first argument to quad is a “callable” Python object (*i.e.* a function, method, or class instance). Notice the use of a lambda- function in this case as the argument. The next two arguments are the limits of integration. The return value is a tuple, with the first element holding the estimated value of the integral and the second element holding an upper bound on the error. Notice, that in this case, the true value of this integral is

*I*=2*π*−−√(18272–√cos(4.5)−4272–√sin(4.5)+2*π*−−√Si(3*π*−−√)),

where

Si(*x*)=∫*x*0sin(*π*2*t*2)*dt*.

is the Fresnel sine integral. Note that the numerically-computed integral is within 1.04×10−11

of the exact result — well below the reported error bound.

If the function to integrate takes additional parameters, the can be provided in the *args* argument. Suppose that the following integral shall be calculated:

*I*(*a*,*b*)=∫10*ax*2+*bdx*.

This integral can be evaluated by using the following code:

>>> from scipy.integrate import quad

>>> def integrand(x, a, b):

... return a\*x\*\*2 + b

...

>>> a = 2

>>> b = 1

>>> I = quad(integrand, 0, 1, args=(a,b))

>>> I

(1.6666666666666667, 1.8503717077085944e-14)

Infinite inputs are also allowed in [quad](http://jiffyclub.github.io/scipy/generated/scipy.integrate.quad.html#scipy.integrate.quad) by using ±

inf as one of the arguments. For example, suppose that a numerical value for the exponential integral:

*En*(*x*)=∫∞1*e*−*xttndt*.

is desired (and the fact that this integral can be computed as special.expn(n,x) is forgotten). The functionality of the function special.expn can be replicated by defining a new function vec\_expint based on the routine [quad](http://jiffyclub.github.io/scipy/generated/scipy.integrate.quad.html#scipy.integrate.quad):

>>> from scipy.integrate import quad

>>> def integrand(t, n, x):

... return np.exp(-x\*t) / t\*\*n

...

>>> def expint(n, x):

... return quad(integrand, 1, np.inf, args=(n, x))[0]

...

>>> vec\_expint = np.vectorize(expint)

>>> vec\_expint(3, np.arange(1.0, 4.0, 0.5))

array([ 0.1097, 0.0567, 0.0301, 0.0163, 0.0089, 0.0049])

>>> import scipy.special as special

>>> special.expn(3, np.arange(1.0,4.0,0.5))

array([ 0.1097, 0.0567, 0.0301, 0.0163, 0.0089, 0.0049])

The function which is integrated can even use the quad argument (though the error bound may underestimate the error due to possible numerical error in the integrand from the use of [quad](http://jiffyclub.github.io/scipy/generated/scipy.integrate.quad.html#scipy.integrate.quad) ). The integral in this case is

*In*=∫∞0∫∞1*e*−*xttndtdx*=1*n*.

>>> result = quad(lambda x: expint(3, x), 0, np.inf)

>>> print(result)

(0.33333333324560266, 2.8548934485373678e-09)

>>> I3 = 1.0/3.0

>>> print(I3)

0.333333333333

>>> print(I3 - result[0])

8.77306560731e-11

This last example shows that multiple integration can be handled using repeated calls to [quad](http://jiffyclub.github.io/scipy/generated/scipy.integrate.quad.html#scipy.integrate.quad).

**General multiple integration ([dblquad](http://jiffyclub.github.io/scipy/generated/scipy.integrate.dblquad.html" \l "scipy.integrate.dblquad" \o "scipy.integrate.dblquad),** [**tplquad**](http://jiffyclub.github.io/scipy/generated/scipy.integrate.tplquad.html#scipy.integrate.tplquad)**,** [**nquad**](http://jiffyclub.github.io/scipy/generated/scipy.integrate.nquad.html#scipy.integrate.nquad)**)**

The mechanics for double and triple integration have been wrapped up into the functions [dblquad](http://jiffyclub.github.io/scipy/generated/scipy.integrate.dblquad.html#scipy.integrate.dblquad) and [tplquad](http://jiffyclub.github.io/scipy/generated/scipy.integrate.tplquad.html#scipy.integrate.tplquad). These functions take the function to integrate and four, or six arguments, respectively. The limits of all inner integrals need to be defined as functions.

An example of using double integration to compute several values of *In*

is shown below:

>>> from scipy.integrate import quad, dblquad

>>> def I(n):

... return dblquad(lambda t, x: np.exp(-x\*t)/t\*\*n, 0, np.inf, lambda x: 1, lambda x: np.inf)

...

>>> print(I(4))

(0.2500000000043577, 1.29830334693681e-08)

>>> print(I(3))

(0.33333333325010883, 1.3888461883425516e-08)

>>> print(I(2))

(0.4999999999985751, 1.3894083651858995e-08)

As example for non-constant limits consider the integral

*I*=∫1/2*y*=0∫1−2*yx*=0*xydxdy*=196.

This integral can be evaluated using the expression below (Note the use of the non-constant lambda functions for the upper limit of the inner integral):

>>> from scipy.integrate import dblquad

>>> area = dblquad(lambda x, y: x\*y, 0, 0.5, lambda x: 0, lambda x: 1-2\*x)

>>> area

(0.010416666666666668, 1.1564823173178715e-16)

For n-fold integration, scipy provides the function [nquad](http://jiffyclub.github.io/scipy/generated/scipy.integrate.nquad.html#scipy.integrate.nquad). The integration bounds are an iterable object: either a list of constant bounds, or a list of functions for the non-constant integration bounds. The order of integration (and therefore the bounds) is from the innermost integral to the outermost one.

The integral from above

*In*=∫∞0∫∞1*e*−*xttndtdx*=1*n*

can be calculated as

>>> from scipy import integrate

>>> N = 5

>>> def f(t, x):

... return np.exp(-x\*t) / t\*\*N

...

>>> integrate.nquad(f, [[1, np.inf],[0, np.inf]])

(0.20000000000002294, 1.2239614263187945e-08)

Note that the order of arguments for *f* must match the order of the integration bounds; i.e. the inner integral with respect to *t*

is on the interval [1,∞] and the outer integral with respect to *x* is on the interval [0,∞]

.

Non-constant integration bounds can be treated in a similar manner; the example from above

*I*=∫1/2*y*=0∫1−2*yx*=0*xydxdy*=196.

can be evaluated by means of

>>> from scipy import integrate

>>> def f(x, y):

... return x\*y

...

>>> def bounds\_y():

... return [0, 0.5]

...

>>> def bounds\_x(y):

... return [0, 1-2\*y]

...

>>> integrate.nquad(f, [bounds\_x, bounds\_y])

(0.010416666666666668, 4.101620128472366e-16)

which is the same result as before.

**Gaussian quadrature**

A few functions are also provided in order to perform simple Gaussian quadrature over a fixed interval. The first is [fixed\_quad](http://jiffyclub.github.io/scipy/generated/scipy.integrate.fixed_quad.html#scipy.integrate.fixed_quad) which performs fixed-order Gaussian quadrature. The second function is [quadrature](http://jiffyclub.github.io/scipy/generated/scipy.integrate.quadrature.html#scipy.integrate.quadrature) which performs Gaussian quadrature of multiple orders until the difference in the integral estimate is beneath some tolerance supplied by the user. These functions both use the module special.orthogonal which can calculate the roots and quadrature weights of a large variety of orthogonal polynomials (the polynomials themselves are available as special functions returning instances of the polynomial class — e.g. [special.legendre](http://jiffyclub.github.io/scipy/generated/scipy.special.legendre.html#scipy.special.legendre)).

**Romberg Integration**

Romberg’s method [[WPR]](http://jiffyclub.github.io/scipy/tutorial/integrate.html#wpr) is another method for numerically evaluating an integral. See the help function for [romberg](http://jiffyclub.github.io/scipy/generated/scipy.integrate.romberg.html#scipy.integrate.romberg) for further details.

**Integrating using Samples**

If the samples are equally-spaced and the number of samples available is 2*k*+1

for some integer *k*

, then Romberg [romb](http://jiffyclub.github.io/scipy/generated/scipy.integrate.romb.html#scipy.integrate.romb) integration can be used to obtain high-precision estimates of the integral using the available samples. Romberg integration uses the trapezoid rule at step-sizes related by a power of two and then performs Richardson extrapolation on these estimates to approximate the integral with a higher-degree of accuracy.

In case of arbitrary spaced samples, the two functions trapz (defined in numpy [[NPT]](http://jiffyclub.github.io/scipy/tutorial/integrate.html#npt)) and [simps](http://jiffyclub.github.io/scipy/generated/scipy.integrate.simps.html#scipy.integrate.simps) are available. They are using Newton-Coates formulas of order 1 and 2 respectively to perform integration. The trapezoidal rule approximates the function as a straight line between adjacent points, while Simpson’s rule approximates the function between three adjacent points as a parabola.

For an odd number of samples that are equally spaced Simpson’s rule is exact if the function is a polynomial of order 3 or less. If the samples are not equally spaced, then the result is exact only if the function is a polynomial of order 2 or less.

>>> import numpy as np

>>> def f1(x):

... return x\*\*2

...

>>> def f2(x):

... return x\*\*3

...

>>> x = np.array([1,3,4])

>>> y1 = f1(x)

>>> from scipy.integrate import simps

>>> I1 = simps(y1, x)

>>> print(I1)

21.0

This corresponds exactly to

∫41*x*2*dx*=21,

whereas integrating the second function

>>> y2 = f2(x)

>>> I2 = integrate.simps(y2, x)

>>> print(I2)

61.5

does not correspond to

∫41*x*3*dx*=63.75

because the order of the polynomial in f2 is larger than two.

**Faster integration using low-level callback functions**

A user desiring reduced integration times may pass a C function pointer through [scipy.LowLevelCallable](http://jiffyclub.github.io/scipy/generated/scipy.LowLevelCallable.html#scipy.LowLevelCallable) to [quad](http://jiffyclub.github.io/scipy/generated/scipy.integrate.quad.html#scipy.integrate.quad), [dblquad](http://jiffyclub.github.io/scipy/generated/scipy.integrate.dblquad.html#scipy.integrate.dblquad), [tplquad](http://jiffyclub.github.io/scipy/generated/scipy.integrate.tplquad.html#scipy.integrate.tplquad) or [nquad](http://jiffyclub.github.io/scipy/generated/scipy.integrate.nquad.html#scipy.integrate.nquad) and it will be integrated and return a result in Python. The performance increase here arises from two factors. The primary improvement is faster function evaluation, which is provided by compilation of the function itself. Additionally we have a speedup provided by the removal of function calls between C and Python in [quad](http://jiffyclub.github.io/scipy/generated/scipy.integrate.quad.html#scipy.integrate.quad). This method may provide a speed improvements of ~2x for trivial functions such as sine but can produce a much more noticeable improvements (10x+) for more complex functions. This feature then, is geared towards a user with numerically intensive integrations willing to write a little C to reduce computation time significantly.

The approach can be used, for example, via [ctypes](https://docs.python.org/dev/library/ctypes.html#module-ctypes) in a few simple steps:

1.) Write an integrand function in C with the function signature double f(int n, double \*x, void \*user\_data), where x is an array containing the point the function f is evaluated at, and user\_data to arbitrary additional data you want to provide.

/\* testlib.c \*/

double f(int n, double \*x, void \*user\_data) {

double c = \*(double \*)user\_data;

return c + x[0] - x[1] \* x[2]; /\* corresponds to c + x - y \* z \*/

}

2.) Now compile this file to a shared/dynamic library (a quick search will help with this as it is OS-dependent). The user must link any math libraries, etc. used. On linux this looks like:

$ gcc -shared -fPIC -o testlib.so testlib.c

The output library will be referred to as testlib.so, but it may have a different file extension. A library has now been created that can be loaded into Python with [ctypes](https://docs.python.org/dev/library/ctypes.html#module-ctypes).

3.) Load shared library into Python using [ctypes](https://docs.python.org/dev/library/ctypes.html#module-ctypes) and set restypes and argtypes - this allows Scipy to interpret the function correctly:

import os, ctypes

from scipy import integrate, LowLevelCallable

lib = ctypes.CDLL(os.path.abspath('testlib.so'))

lib.f.restype = ctypes.c\_double

lib.f.argtypes = (ctypes.c\_int, ctypes.POINTER(ctypes.c\_double), ctypes.c\_void\_p)

c = ctypes.c\_double(1.0)

user\_data = ctypes.cast(ctypes.pointer(c), ctypes.c\_void\_p)

func = LowLevelCallable(lib.f, user\_data)

The last void \*user\_data in the function is optional and can be omitted (both in the C function and ctypes argtypes) if not needed. Note that the coordinates are passed in as an array of doubles rather than a separate argument.

4.) Now integrate the library function as normally, here using [nquad](http://jiffyclub.github.io/scipy/generated/scipy.integrate.nquad.html#scipy.integrate.nquad):

>>> integrate.nquad(func, [[0, 10], [-10, 0], [-1, 1]])

(1200.0, 1.1102230246251565e-11)

The Python tuple is returned as expected in a reduced amount of time. All optional parameters can be used with this method including specifying singularities, infinite bounds, etc.

### Tkdiff e References

|  |  |
| --- | --- |
| [[WPR]](http://jiffyclub.github.io/scipy/tutorial/integrate.html#id1) | [https://en.wikipedia.org/wiki/Romberg’s\_method](https://en.wikipedia.org/wiki/Romberg%27s_method) |

|  |  |
| --- | --- |
| [[NPT]](http://jiffyclub.github.io/scipy/tutorial/integrate.html#id2) | <https://docs.scipy.org/doc/numpy/reference/generated/numpy.trapz.html> |

|  |  |
| --- | --- |
| [[MOL]](http://jiffyclub.github.io/scipy/tutorial/integrate.html#id3) | <https://en.wikipedia.org/wiki/Method_of_lines> |