
lintegrate

Release 0.1.16

Matthew Pitkin

Sep 04, 2023

CONTENTS:

1	Installation	3
1.1	Installing from source	3
1.2	PyPI	4
1.3	Conda	4
1.4	Contributing	5
2	C API	7
2.1	QNG non-adaptive Gauss-Kronrod integration	7
2.2	QAG adaptive integration	7
2.3	CQUAD doubly-adaptive integration	8
3	Python API	9
4	Examples	13
4.1	C library	13
4.2	Python interface	18
5	Citation	19
6	Indices and tables	21
	Index	23

lintegrate provides a C library for numerical integration of functions for which numerical instabilities require working with the natural logarithm of the function.

In addition to the C library, there are Python versions of several of the core methods that are accessed in a similar way to the SciPy `quad()` function.

INSTALLATION

The lintegrate library requires that you have the [GNU Scientific Library \(GSL\)](#) installed on your system. In particular, the development version of the library is required, so that the header files are present.

For Linux systems, GSL can be installed with:

Debian-based

```
$ sudo apt install libgsl-dev
```

Red Hat-based

```
$ sudo yum install gsl-devel
```

openSUSE-based

```
$ sudo zypper install gsl-devel
```

For Mac OS, GSL can be installed with:

```
$ brew install gsl
```

For Windows you can install lintegrate via Conda, which will automatically include dependencies such as GSL.

1.1 Installing from source

The C library of lintegrate can be installed on Linux and Mac OS systems from the source hosted on [GitHub](#). This requires the [SCons](#) software construction tool. The source repository can be cloned with:

HTTPS

```
$ git clone https://github.com/mattpitkin/lntegrate.git
```

ssh

```
$ git clone git@github.com:mattpitkin/lntegrate.git
```

GitHub CLI

```
$ gh repo clone mattpitkin/lntegrate
```

After cloning, build and install the library with:

```
$ sudo scons  
$ sudo scons install
```

1.1.1 Python package

The Python package can also be installed by running:

```
$ pip install .
```

1.2 PyPI

To install lntegrate using pip from the [PyPI repository](#) [GSL](#) must be already installed on your system. lntegrate can then be installed with:

```
$ pip install lntegrate
```

1.3 Conda

lntegrate can be installed on Linux, Mac OS, or Windows, without having to pre-install any dependencies by using the [conda-forge](#) version of the package. This can be done via the Anaconda Navigator or on the command line using:

```
$ conda install -c conda-forge lntegrate
```


1.4 Contributing

Contributions to lntegrate, either to the C library or the Python wrapper, are welcome. Issues can be raised via the GitHub [issue tracker](#) and [pull requests](#) are encouraged.

The C library contains a selection numerical integration functions based on equivalents in the [GSL numerical integration](#) package. To use these functions the `lrintegrate.h` header file must be included.

For all these functions the input `gsl_function` `f` should return the natural logarithm of the required integrand. The outputs will include the natural logarithm of the resulting integral.

2.1 QNG non-adaptive Gauss-Kronrod integration

```
int lintegration_qng (const gsl_function *f, double a, double b, double epsabs,  
double epsrel, double * result, double * abserr, size_t * neval)
```

This is the equivalent of the GSL `gsl_integration_qng()` function (see the link for a description of the arguments and return values). It applies a non-adaptive procedure which applied the [Gauss-Kronrod](#) integration rules. The output will include the natural logarithm of the resulting integral.

```
int lintegration_qng_split (const gsl_function *f, double *splitpts, size_t npts,  
double epsabs, double epsrel, double * result, double * abserr, size_t * neval)
```

In some cases it might be necessary to split the integral into multiple segments that are integrated separately and then combined. For example, if there is a large dynamic range for the abscissa variable. The “split” version of `lintegration_qng()` takes in an array of values `splitpts` giving the bounds (including of the upper bound point) and the length of that array `npts`. The other arguments are the same as for `lintegration_qng()`.

2.2 QAG adaptive integration

```
int lintegration_qag (const gsl_function *f, double a, double b, double epsabs,  
double epsrel, size_t limit, int key, gsl_integration_workspace * workspace,  
double * result, double * abserr)
```

This is the equivalent of the GSL `gsl_integration_qag()` function (see the links for a description of the arguments and return values). It applies a simple adaptive integration procedure. The integration region is divided into subintervals, and on each iteration the subinterval with the largest estimated error is bisected.

```
int lintegration_qag_split (const gsl_function *f, double *splitpts, size_t npts,  
double epsabs, double epsrel, size_t limit, int key, double * result, double * abserr)
```

In some cases it might be necessary to split the integral into multiple segments that are integrated separately and then combined. For example, if there is a large dynamic range for the abscissa variable. The “split” version of `lintegration_qag()` takes in an array of values `splitpts` giving the bounds (including of the upper bound point) and the length of that array `npts`. The other arguments are the same as for `lintegration_qag()` except that it does not require a `gsl_integration_workspace`.

2.3 CQUAD doubly-adaptive integration

```
int lntegration_cquad (const gsl_function * f, double a, double b, double epsabs,
double epsrel, gsl_integration_cquad_workspace * ws, double *result, double *abserr,
size_t * nevals)
```

This is the equivalent of the GSL `gsl_integration_cquad()` function (see the links for a description of the arguments and return values). The algorithm uses a doubly-adaptive scheme in which [Clenshaw-Curtis](#) quadrature rules of increasing degree are used to compute the integral in each interval. It can handle most types of singularities, non-numerical function values such as Inf or NaN, as well as some divergent integrals.

```
int lntegration_cquad_split (const gsl_function * f, double *splitpts, size_t npts,
double epsabs, double epsrel, size_t wsints, double *result, double *abserr,
size_t * nevals)
```

In some cases it might be necessary to split the integral into multiple segments that are integrated separately and then combined. For example, if there is a large dynamic range for the abscissa variable. The “split” version of `lntegration_cquad()` takes in an array of values `splitpts` giving the bounds (including of the upper bound point) and the length of that array `npts`. The other arguments are the same as for `lntegration_cquad()`. It does not require a `gsl_integration_cquad_workspace`, but does take in `wsints` to set the maximum number of intervals for each `gsl_integration_cquad_workspace` allocated via `gsl_integration_cquad_workspace_alloc()`.

PYTHON API

The functions described in the C API can be access using the Python `lintegrte` module provided with the library. These functions have a similar style to the SciPy `quad()` function.

logtrapz(*f*, *x*[, *disable_checks=False*, *args=()*])

Given the natural logarithm *f* of some function *g* (i.e., $f = \log(g)$), compute the natural logarithm of the integral of that function using the trapezium rule.

Parameters

- **f** (`numpy.ndarray`, list, or function handle) – A set of values of the logarithm of a function evaluated at points given by *x*, or a function handle of the log function to be evaluated.
- **x** (`numpy.ndarray`, list, or float) – An array of values at which *f* has been evaluated, or is to be evaluated at. Or, provided *f* is also an array, a single value giving the spacing between evaluation points (if the function has been evaluated on an evenly spaced grid).
- **disable_checks** (`bool`) – Set this to True to disable checks, such as making sure *x* values are in ascending order. Defaults to False.
- **args** (`tuple`) – A tuple of any additional parameters required by the function *f* (to be unpacked within the function).

Returns The natural logarithm of the integral of the function

Return type `float`

lqng(*func*, *a=0.*, *b=0.*[, *args=()*, *epsabs=1.49e-8*, *epsrel=1.49e-8*, *intervals=None*, *nintervals=0*, *intervaltype='linear'*])

Python wrapper to the `lintegration_qng()` function. This will integrate $\exp(func)$, whilst staying in log-space to ensure numerical precision, using a non-adaptive procedure. It uses fixed Gauss-Kronrod-Patterson abscisae to sample the integrand at a maximum of 87 points (see `gsl_integration_qng()`).

Parameters

- **func** (`function`) – A callable Python function which returns the natural logarithm of the underlying function being integrated over.
- **a** (`float`) – Lower limit of integration
- **b** (`float`) – Upper limit of integration
- **args** (`tuple`) – Extra arguments to pass to *func*. These must be unpacked within *func*, e.g.:

```
def myfunc(x, args):  
    y, z = args  
    return x + y + z
```

- **epsabs** (`float`) – The absolute error tolerance for the integral

- **epsrel** (*float*) – The relative error tolerance for the integral
- **intervals** (*numpy.ndarray* or list) – An array of values bounding intervals into which the integral will be split (this could be used, for example, if you have a very tightly peaked function and require small intervals around the peak).
- **nintervals** (*int*) – If **intervals** is not given then split the range between a and b into **nintervals** intervals
- **intervaltype** (*str*) – If splitting into **nintervals** intervals then choose whether to split the range in equal intervals in ‘linear’, ‘log’, or ‘log10’ space

Returns A tuple containing: the natural logarithm of the integral of `exp(func)`, an estimate of the absolute error in the result, and the number of evaluations used in the integration

Return type *tuple*

lqag(*func*, *a*=0., *b*=0., [*args*=(), *epsabs*=1.49e-8, *epsrel*=1.49e-8, *limit*=50, *intkey*=1, *intervals*=None, *nintervals*=0, *intervaltype*='linear'])

Python wrapper to the `lintegration_qag()` function. This will integrate `exp(func)`, whilst staying in log-space to ensure numerical precision, using a simple adaptive procedure (see `gsl_integration_qag()`).

Parameters

- **func** (*function*) – A callable Python function which returns the natural logarithm of the underlying function being integrated over.
- **a** (*float*) – Lower limit of integration
- **b** (*float*) – Upper limit of integration
- **args** (*tuple*) – Extra arguments to pass to *func*. These must be unpacked within *func*, e.g.:

```
def myfunc(x, args):  
    y, z = args  
    return x + y + z
```

- **epsabs** (*float*) – The absolute error tolerance for the integral
- **epsrel** (*float*) – The relative error tolerance for the integral
- **limit** (*int*) – The maximum number of subintervals used in the integration
- **intkey** (*int*) – A key given the integration rule following those for the `gsl_integration_qag()` function. This can be 1, 2, 3, 4, 5, or 6, corresponding to the 15, 21, 31, 41, 51 and 61 point Gauss-Kronrod rules respectively.
- **intervals** (*numpy.ndarray* or list) – An array of values bounding intervals into which the integral will be split (this could be used, for example, if you have a very tightly peaked function and require small intervals around the peak).
- **nintervals** (*int*) – If **intervals** is not given then split the range between a and b into **nintervals** intervals
- **intervaltype** (*str*) – If splitting into **nintervals** intervals then choose whether to split the range in equal intervals in ‘linear’, ‘log’, or ‘log10’ space

Returns A tuple containing: the natural logarithm of the integral of `exp(func)` and an estimate of the absolute error in the result

Return type *tuple*

```
lcquad(func, a, b[, args=(), epsabs=1.49e-8, epsrel=1.49e-8, wsintervals=100, intervals=None, nintervals=0,
intervaltype='linear'])
```

Python wrapper to the `lintegration_cquad()` function. This will integrate $\exp(\text{func})$, whilst staying in log-space to ensure numerical precision, using a doubly adaptive procedure (see `gsl_integration_cquad()`).

Parameters

- **func** (*function*) – A callable Python function which returns the natural logarithm of the underlying function being integrated over.
- **a** (*float*) – Lower limit of integration
- **b** (*float*) – Upper limit of integration
- **args** (*tuple*) – Extra arguments to pass to *func*. These must be unpacked within *func*, e.g.:

```
def myfunc(x, args):
    y, z = args
    return x + y + z
```

- **epsabs** (*float*) – The absolute error tolerance for the integral
- **epsrel** (*float*) – The relative error tolerance for the integral
- **wsintervals** (*int*) – A sufficient number of subintervals for the integration (if the workspace is full the smallest intervals will be discarded)
- **intervals** (*numpy.ndarray* or list) – An array of values bounding intervals into which the integral will be split (this could be used, for example, if you have a very tightly peaked function and require small intervals around the peak).
- **nintervals** (*int*) – If *intervals* is not given then split the range between *a* and *b* into *nintervals* intervals
- **intervaltype** (*str*) – If splitting into *nintervals* intervals then choose whether to split the range in equal intervals in 'linear', 'log', or 'log10' space

Returns A tuple containing: the natural logarithm of the integral of $\exp(\text{func})$, an estimate of the absolute error in the result, and the number of evaluations used in the integration

Return type *tuple*

EXAMPLES

4.1 C library

Here we show a selection of examples of using the `lintegr` C library functions. A Linux Makefile for the three examples can be found [here](#).

4.1.1 Example 1

In this example the `lintegration_qag()`, `lintegration_qng()` and `lintegration_cquad()` functions are used to evaluate the integral of a Gaussian function in log-space. The outputs are compared to the result of integration the original function using the GSL `gsl_integration_qag()` function.

```
/* example using integr functionality */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <gsl/gsl_math.h>
#include <gsl/gsl_integration.h>

#include <lintegr.h>

/* create function for integration */
double lintegrand(double x, void *params);

struct intparams {
    double mu;
    double sig;
};

double lintegrand(double x, void *params){
    struct intparams * p = (struct intparams *)params;
    double mu = p->mu;
    double sig = p->sig;

    return -0.5*(mu-x)*(mu-x)/(sig*sig);
}

double integrand(double x, void *params){
    struct intparams * p = (struct intparams *)params;
```

(continues on next page)

(continued from previous page)

```

double mu = p->mu;
double sig = p->sig;

return exp(-0.5*(mu-x)*(mu-x)/(sig*sig));
}

int main( int argv, char **argc ){
    gsl_function F;
    struct intparams params;
    gsl_integration_workspace *w = gsl_integration_workspace_alloc (100);
    gsl_integration_cquad_workspace *cw = gsl_integration_cquad_workspace_alloc(50);
    double qaganswer = 0., qnganswer = 0., cquadanswer = 0., answer = 0.;
    double abserr = 0.;
    size_t neval = 0;

    double minlim = -6.; /* minimum for integration range */
    double maxlim = 6.; /* maximum for integration range */

    double abstol = 1e-10; /* absolute tolerance */
    double reltol = 1e-10; /* relative tolerance */

    params.mu = 0.;
    params.sig = 1.;

    F.function = &lntegrand;
    F.params = &params;

    /* integrate log of function using QAG */
    lntegration_qag(&F, minlim, maxlim, abstol, reltol, 100, GSL_INTEG_GAUSS31, w, &
    ↪qaganswer, &abserr);

    /* integrate log of function using QNG */
    lntegration_qng(&F, minlim, maxlim, abstol, reltol, &qnganswer, &abserr, &neval);

    /* integrate log of function using CQUAD */
    lntegration_cquad(&F, minlim, maxlim, abstol, reltol, cw, &cquadanswer, &abserr, &
    ↪neval);

    /* integrate function using GSL QAG */
    F.function = &integrand;
    gsl_integration_qag(&F, minlim, maxlim, abstol, reltol, 100, GSL_INTEG_GAUSS31, w, &
    ↪answer, &abserr);

    gsl_integration_workspace_free(w);
    gsl_integration_cquad_workspace_free(cw);

    fprintf(stdout, "Answer \"lntegrate QAG\" = %.8lf\n", qaganswer);
    fprintf(stdout, "Answer \"lntegrate QNG\" = %.8lf\n", qnganswer);
    fprintf(stdout, "Answer \"lntegrate CQUAD\" = %.8lf\n", cquadanswer);
    fprintf(stdout, "Answer \"gsl_integrate_qag\" = %.8lf\n", log(answer));
    fprintf(stdout, "Analytical answer = %.8lf\n", log(sqrt(2.*M_PI)));

```

(continues on next page)

(continued from previous page)

```

return 0;
}

```

4.1.2 Example 2

In this example the `lintegration_qag()`, `lintegration_qng()` and `lintegration_cquad()` functions are used to evaluate the integral of flat function in log-space. The outputs are compared to the result of integration the original function using the GSL `gsl_integration_qag()` function.

```

/* example using lintegrate functionality */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <gsl/gsl_math.h>
#include <gsl/gsl_integration.h>

#include <lintegrate.h>

/* create function for integration */
double lintegrand(double x, void *params);

double lintegrand(double x, void *params){
    return 50.;
}

double integrand(double x, void *params){
    return exp(50.);
}

int main( int argv, char **argc ){
    gsl_function F;
    gsl_integration_workspace *w = gsl_integration_workspace_alloc (100);
    gsl_integration_cquad_workspace *cw = gsl_integration_cquad_workspace_alloc(50);
    double qaganswer = 0., qnganswer = 0., cquadanswer = 0., answer = 0.;
    double abserr = 0.;
    size_t neval = 0;

    double minlim = -6.; /* minimum for integration range */
    double maxlim = 6.; /* maximum for integration range */

    double abstol = 1e-10; /* absolute tolerance */
    double reltol = 1e-10; /* relative tolerance */

    F.function = &lintegrand;

    /* integrate log of function using QAG */
    lintegration_qag(&F, minlim, maxlim, abstol, reltol, 100, GSL_INTEG_GAUSS31, w, &
    ↪qaganswer, &abserr);

    /* integrate log of function using QNG */

```

(continues on next page)

(continued from previous page)

```

lntegration_qng(&F, minlim, maxlim, abstol, reltol, &qnganswer, &abserr, &neval);

/* integrate log of function using CQUAD */
lntegration_cquad(&F, minlim, maxlim, abstol, reltol, cw, &cquadanswer, &abserr, &
↪neval);

/* integrate function using GSL QAG */
F.function = &integrand;
gsl_integration_qag(&F, minlim, maxlim, abstol, reltol, 100, GSL_INTEG_GAUSS31, w, &
↪answer, &abserr);

gsl_integration_workspace_free(w);
gsl_integration_cquad_workspace_free(cw);

fprintf(stdout, "Answer \"lntegrate QAG\" = %.8lf\n", qaganswer);
fprintf(stdout, "Answer \"lntegrate QNG\" = %.8lf\n", qnganswer);
fprintf(stdout, "Answer \"lntegrate CQUAD\" = %.8lf\n", cquadanswer);
fprintf(stdout, "Answer \"gsl_integrate_qag\" = %.8lf\n", log(answer));
fprintf(stdout, "Analytic answer = %.8lf\n", log(maxlim-minlim) + 50.);

return 0;
}

```

4.1.3 Example 3

In this example the `lntegration_qag_split()`, `lntegration_qng_split()` and `lntegration_cquad_split()` functions are used to evaluate the integral of function that is very tightly peaked log-space. The outputs are compared to an analytical calculation of the integral of original function found using [Wolfram Alpha](#).

```

/* example using lntegrate_split functionality */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <gsl/gsl_math.h>
#include <gsl/gsl_integration.h>

#include <time.h>

#include <lntegrate.h>

/* create function for integration */
double lntegrand(double x, void *params);

double lntegrand(double x, void *params){
    return 1e-11/x;
}

int main( int argv, char **argc ){
    gsl_function F;

```

(continues on next page)

(continued from previous page)

```

double qaganswer = 0., cquadanswer = 0., qnganswer = 0.;
double abserr = 0.;
size_t neval = 0;

double minlim = 1e-13; /* minimum for integration range */
double maxlim = 1e-5; /* maximum for integration range */

/* split the interval logarithmically for integrating */
size_t nints = 50, i = 0;
double splits[nints+1];
for ( i=0; i<nints+1; i++ ){
    splits[i] = pow(10., (log10(minlim) + i*(log10(maxlim)-log10(minlim)))/
↪(double)nints));
}

double abstol = 1e-10; /* absolute tolerance */
double reltol = 1e-10; /* relative tolerance */

F.function = &lntegrand;

clock_t t1, t2;

/* integrate log of function using QAG */
t1 = clock();
lntegration_qag_split(&F, splits, nints+1, abstol, reltol, 100, GSL_INTEG_GAUSS31, &
↪qaganswer, &abserr);
t2 = clock();
fprintf(stderr, "lntegration_qag_split: run time = %ld mus\n", (t2-t1)*1000000/CLOCKS_
↪PER_SEC);

/* integrate log of function using QNG */
t1 = clock();
lntegration_qng_split(&F, splits, nints+1, abstol, reltol, &qnganswer, &abserr, &
↪neval);
t2 = clock();
fprintf(stderr, "lntegration_qng_split: run time = %ld mus\n", (t2-t1)*1000000/CLOCKS_
↪PER_SEC);

/* integrate log of function using CQUAD */
t1 = clock();
lntegration_cquad_split(&F, splits, nints+1, abstol, reltol, 50, &cquadanswer, &
↪abserr, &neval);
t2 = clock();
fprintf(stderr, "lntegration_cquad_split: run time = %ld mus\n", (t2-t1)*1000000/
↪CLOCKS_PER_SEC);

fprintf(stdout, "Answer \"lntegrate QAG\" = %.8lf\n", qaganswer);
fprintf(stdout, "Answer \"lntegrate QNG\" = %.8lf\n", qnganswer);
fprintf(stdout, "Answer \"lntegrate CQUAD\" = %.8lf\n", cquadanswer);
fprintf(stdout, "(Approximate) Analytical answer = %.8lf\n", log(2.74356e28)); // via_
↪Wolfram Alpha

```

(continues on next page)

(continued from previous page)

```

    return 0;
}

```

4.2 Python interface

Here we show an example of using the Python interface for linterate.

```

from linterate import lqag, lqng, lcquad, logtrapz
import numpy as np

# define the log of the function to be integrated
def integrand(x, args):
    mu, sig = args # unpack extra arguments
    return -0.5*((x-mu)/sig)**2

# set integration limits
xmin = -6.
xmax = 6.

# set additional arguments
mu = 0.
sig = 1.

resqag = lqag(integrand, xmin, xmax, args=(mu, sig))
resqng = lqng(integrand, xmin, xmax, args=(mu, sig))
rescquad = lcquad(integrand, xmin, xmax, args=(mu, sig))
restrapz = logtrapz(integrand, np.linspace(xmin, xmax, 100), args=(mu, sig))

```

4.2.1 Using R

The linterate Python interface can be accessed using R through the `reticulate` package. The above example would be:

```

library(reticulate)
py_install("linterate", pip = TRUE) ## run once to make sure linterate is installed
  ↪ and visible to reticulate.
lint <- import("linterate", convert = FALSE)
integrand <- function(x, args){
  mu = args[1]
  sig = args[2]
  return(-.5 * ((x-mu)/sig)^2 )
}
integrand <- Vectorize(integrand)
mu <- 0
sig <- 1
mmin <- -10
mmax <- 10
lint$lqag(py_func(integrand), r_to_py(mmin), r_to_py(mmax), c(mu, sig))

```

CITATION

If using `lintegrate` within your research, I would be grateful if you cite the associated [JOSS paper](#) for the software. The following BibTeX citation can be used:

```
@article{Pitkin2022,  
  doi = {10.21105/joss.04231},  
  url = {https://doi.org/10.21105/joss.04231},  
  year = {2022},  
  publisher = {The Open Journal},  
  volume = {7},  
  number = {73},  
  pages = {4231},  
  author = {Matthew Pitkin},  
  title = {lintegrate: A C/Python numerical integration library for working in log-space}  
  ↪,  
  journal = {Journal of Open Source Software}  
}
```

You may also want to cite the [GSL](#) reference “*M. Galassi et al, GNU Scientific Library Reference Manual (3rd Ed.), ISBN 0954612078*” and the URL <http://www.gnu.org/software/gsl/>.

INDICES AND TABLES

- `genindex`
- `search`

B

built-in function

 lcquad(), 10

 logtrapz(), 9

 lqag(), 10

 lqng(), 9

L

lcquad()

 built-in function, 10

logtrapz()

 built-in function, 9

lqag()

 built-in function, 10

lqng()

 built-in function, 9