

## 9.4: Gaussian Quadratures

Previously, we have assumed in our analysis of numerical integration schemes that the discretization points are equally-spaced. This assumption is not strictly necessary; for instance, we can easily modify the mid-point rule and trapezium rule formulas to work with non-equally-spaced points.

However, if we are free to choose the discretization points for performing numerical integration, and these points need not be equally spaced, then it is possible to exploit this freedom to further improve the accuracy of the numerical integration. The idea is to optimize the placement of the discretization points, so as to minimize the resulting numerical error. This is the basic idea behind the method of integration by **Gaussian quadratures**.

We will not discuss the details of this numerical integration method. To use it, you can call the `scipy.integrate.quad` function. This function uses a low-level numerical library named **QUADPACK**, which performs quadrature integration with **adaptive quadratures**—i.e., it automatically figures out how many discretization points should be used, and where they should be located, in order to produce a result with the desired numerical accuracy.

Because QUADPACK figures out the discretization points for itself, you have to pass `quad` a *function* representing the integrand, rather than an array of integrand values as with `trapz` or `sims`. The standard way to call the function is

```
t = quad(f, a, b)
```

which calculates the integral

$$t = \int_a^b f(x) dx. \quad (9.4.1)$$

The return value is a tuple of the form `(t, err)`, where `t` is the value of the integral and `err` is an estimate of the numerical error. The `quad` function also accepts many other optional inputs, which can be used to specify additional inputs for passing to the integrand function, the error tolerance, the number of subintervals to use, etc. See the [documentation](#) for details.

The choice of whether to perform numerical integration using Simpson's rule (`sims`) or Gaussian quadratures (`quad`) is situational. If you already know the values of the integrands at a pre-determined set of discretization points (e.g., from the result of a finite-difference calculation), then use `sims`. If you can define a function that can quickly calculate the value of the integrand at any point, use `quad`.

Here is an example of using `quad` to compute the integral  $\int_0^\infty \frac{dx}{x^2+1}$ :

```
from scipy import *
from scipy.integrate import quad

def f(x):
    return 1.0 / (x*x+1)

integ, _ = quad(f, 0, 1000)
print(integ)
```

(Note that `quad` returns two values; the first is the computed value of the integral, and the other is the absolute error, which we're not interested in, so we toss it in the "throwaway" variable named `_`. See the [documentation](#) for details.) Running the above program prints the result `1.569796...`, which differs from the exact result,  $\pi/2 = 1.570796...$ , by a relative error of 0.06%.

Here is another example, where the integrand takes an additional parameter:  $\int_0^\infty x e^{-\lambda x} dx$ :

```
from scipy import *
from scipy.integrate import quad
```

```
def f(x, lambd):  
    return x * exp(-lambd * x)  
  
integ, _ = quad(f, 0, 100, args=(0.5))  
print(integ)
```

Running the program prints the result 4.0, which agrees with the exact result of  $1/\lambda^2$  for the chosen value of the parameter  $\lambda = 0.5$ .

---

This page titled [9.4: Gaussian Quadratures](#) is shared under a [CC BY-SA 4.0](#) license and was authored, remixed, and/or curated by [Y. D. Chong](#) via [source content](#) that was edited to the style and standards of the LibreTexts platform.