10 Gaussian Quadrature

Lab Objective: *Learn the basics of Gaussian quadrature and its application to numerical integration. Build a class to perform numerical integration using Legendre and Chebyshev polynomials. Compare the accuracy and speed of both types of Gaussian quadrature with the built-in Scipy package. Perform multivariate Gaussian quadrature.*

Legendre and Chebyshev Gaussian Quadrature

It can be shown that for any class of orthogonal polynomials $p \in \mathbb{R}[x; 2n+1]$ with corresponding weight function $w(x)$, there exists a set of points $\{x_i\}_{i=0}^n$ and weights $\{w_i\}_{i=0}^n$ such that

$$\int_a^b p(x)w(x)dx = \sum_{i=0}^{n} p(x_i)w_i.$$

Since this relationship is exact, a good approximation for the integral

$$\int_a^b f(x)w(x)dx$$

can be expected as long as the function $f(x)$ can be reasonably interpolated by a polynomial at the points x_i for $i = 0, 1, \dots, n$. In fact, it can be shown that if $f(x)$ is $2n+1$ times differentiable, the error of the approximation will decrease as n increases.

Gaussian quadrature can be performed using any basis of orthonormal polynomials, but the most commonly used are the Legendre polynomials and the Chebyshev polynomials. Their weight functions are $w_l(x) = 1$ and $w_c(x) = \frac{1}{\sqrt{1-x^2}}$, respectively, both defined on the open interval $(-1, 1)$.

Problem 1. Define a class for performing Gaussian quadrature. The constructor should accept an integer n denoting the number of points and weights to use (this will be explained later) and a label indicating which class of polynomials to use. If the label is not either `"legendre"` or `"chebyshev"`, raise a `ValueError`; otherwise, store it as an attribute.

The weight function $w(x)$ will show up later in the denominator of certain computations. Define the reciprocal function $w(x)^{-1} = 1/w(x)$ as a `lambda` function and save it as an attribute.

Calculating Points and Weights

All sets of orthogonal polynomials $\{u_k\}_{k=0}^n$ satisfy the three-term recurrence relation

$$u_0 = 1, \qquad u_1 = x - \alpha_1, \qquad u_{k+1} = (x - \alpha_k)u_k - \beta_k u_{k-1}$$

for some coefficients $\{\alpha_k\}_{k=1}^n$ and $\{\beta_k\}_{k=1}^n$. For the Legendre polynomials, they are given by

$$\alpha_k = 0, \qquad \beta_k = \frac{k^2}{4k^2 - 1},$$

and for the Chebyshev polynomials, they are

$$\alpha_k = 0, \qquad \beta_k = \begin{cases} \frac{1}{2} & \text{if } k = 1 \\ \frac{1}{4} & \text{otherwise.} \end{cases}$$

Given these values, the corresponding *Jacobi matrix* is defined as follows.

$$J = \begin{bmatrix} \alpha_1 & \sqrt{\beta_1} & 0 & \dots & 0 \\ \sqrt{\beta_1} & \alpha_2 & \sqrt{\beta_2} & \dots & 0 \\ 0 & \sqrt{\beta_2} & \alpha_3 & \ddots & 0 \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & \dots & & & \sqrt{\beta_{n-1}} \\ 0 & \dots & & \sqrt{\beta_{n-1}} & \alpha_n \end{bmatrix}$$

According to the *Golub-Welsch algorithm*,[1] the n eigenvalues of J are the points x_i to use in Gaussian quadrature, and the corresponding weights are given by $w_i = \mu_w(\mathbb{R})v_{i,0}^2$ where $v_{i,0}$ is the first entry of the ith eigenvector and $\mu_w(\mathbb{R}) = \int_{-\infty}^{\infty} w(x)dx$ is the *measure* of the weight function. Since the weight functions for Legendre and Chebyshev polynomials have compact support on the interval $(-1, 1)$, their measures are given as follows.

$$\mu_{w_l}(\mathbb{R}) = \int_{-\infty}^{\infty} w_l(x)dx = \int_{-1}^{1} 1dx = 2 \qquad \mu_{w_c}(\mathbb{R}) = \int_{-\infty}^{\infty} w_c(x)dx = \int_{-1}^{1} \frac{1}{\sqrt{1-x^2}}dx = \pi$$

Problem 2. Write a method for your class from Problem 1 that accepts an integer n. Construct the $n \times n$ Jacobi matrix J for the polynomial family indicated in the constructor. Use SciPy to compute the eigenvalues and eigenvectors of J, then compute the points $\{x_i\}_{i=1}^n$ and weights $\{w_i\}_{i=1}^n$ for the quadrature. Return both the array of points and the array weights.

Test your method by checking your points and weights against the following values using the Legendre polynomials with $n = 5$.

x_i	$-\frac{1}{3}\sqrt{5+2\sqrt{\frac{10}{7}}}$	$-\frac{1}{3}\sqrt{5-2\sqrt{\frac{10}{7}}}$	0	$\frac{1}{3}\sqrt{5-2\sqrt{\frac{10}{7}}}$	$\frac{1}{3}\sqrt{5+2\sqrt{\frac{10}{7}}}$
w_i	$\frac{322-13\sqrt{70}}{900}$	$\frac{322+13\sqrt{70}}{900}$	$\frac{128}{225}$	$\frac{322+13\sqrt{70}}{900}$	$\frac{322-13\sqrt{70}}{900}$

[1] See `http://gubner.ece.wisc.edu/gaussquad.pdf` for a complete treatment of the Golub-Welsch algorithm, including the computation of the recurrence relation coefficients for arbitrary orthogonal polynomials.

Finally, modify the constructor of your class so that it calls your new function and stores the resulting points and weights as attributes.
(Note: The order of the points and weights in the table may differ depending on whether you used `scipy.linalg.eig()` or `scipy.linalg.eigh()`. The order doesn't matter but it is important that each point corresponds to the correct weight.)

Integrating with Given Weights and Points

Now that the points and weights have been obtained, they can be used to approximate the integrals of different functions. For a given function $f(x)$ with points x_i and weights w_i,

$$\int_{-1}^{1} f(x)w(x)dx \approx \sum_{i=1}^{n} f(x_i)w_i.$$

There are two problems with the preceding formula. First, the weight function is part of the integral being approximated, and second, the points obtained are only found on the interval $(-1, 1)$ (in the case of the Legendre and Chebyshev polynomials). To solve the first problem, define a new function $g(x) = f(x)/w(x)$ so that

$$\int_{-1}^{1} f(x)dx = \int_{-1}^{1} g(x)w(x)dx \approx \sum_{i=1}^{n} g(x_i)w_i. \tag{10.1}$$

The integral of $f(x)$ on $[-1, 1]$ can thus be approximated with the inner product $\mathbf{w}^\mathsf{T} g(\mathbf{x})$, where $g(\mathbf{x}) = [g(x_1), \dots, g(x_n)]^\mathsf{T}$ and $\mathbf{w} = [w_1, \dots, w_n]^\mathsf{T}$.

Problem 3. Write a method for your class that accepts a callable function f. Use (10.1) and the stored points and weights to approximate of the integral of f on the interval $[-1, 1]$.
(Hint: Use $w(x)^{-1}$ from Problem 1 to compute $g(x)$ without division.)

Test your method with examples that are easy to compute by hand and by comparing your results to `scipy.integrate.quad()`.

```
>>> import numpy as np
>>> from scipy.integrate import quad

# Integrate f(x) = 1 / sqrt(1 - x**2) from -1 to 1.
>>> f = lambda x: 1 / np.sqrt(1 - x**2)
>>> quad(f, -1, 1)[0]
3.141592653589591
```

NOTE

Since the points and weights for Gaussian quadrature do not depend on f, they only need to be computed once and can then be reused to approximate the integral of any function. The class structure in Problems 1–4 takes advantage of this fact, but `scipy.integrate.quad()` does not. If a larger n is needed for higher accuracy, however, the computations must be repeated to get a new set of points and weights.

Shifting the Interval of Integration

Since the weight functions for the Legendre and Chebyshev polynomials have compact support on the interval $(-1, 1)$, all of the quadrature points are found on that interval as well. To integrate a function on an arbitrary interval $[a, b]$ requires a change of variables. Let

$$u = \frac{2x - b - a}{b - a}$$

so that $u = -1$ when $x = a$ and $u = 1$ when $x = b$. Then

$$x = \frac{b-a}{2}u + \frac{a+b}{2} \qquad \text{and} \qquad dx = \frac{b-a}{2}du,$$

so the transformed integral is given by

$$\int_a^b f(x)dx = \frac{b-a}{2}\int_{-1}^{1} f\left(\frac{b-a}{2}u + \frac{a+b}{2}\right)du.$$

By defining a new function $h(x)$ as

$$h(x) = f\left(\frac{(b-a)}{2}x + \frac{(a+b)}{2}\right),$$

the integral of f can be approximated by integrating h over $[-1, 1]$ with (10.1). This results in the final quadrature formula

$$\int_a^b f(x)dx = \frac{b-a}{2}\int_{-1}^{1} h(x)dx = \frac{b-a}{2}\int_{-1}^{1} g(x)w(x)dx \approx \frac{b-a}{2}\sum_{i=1}^{n} g(x_i)w_i, \tag{10.2}$$

where now $g(x) = h(x)/w(x)$.

Problem 4. Write a method for your class that accepts a callable function f and bounds of integration a and b. Use (10.2) to approximate the integral of f from a to b.
(Hint: Define $h(x)$ and use your method from Problem 3.)

Problem 5. The *standard normal distribution* has the following probability density function.

$$f(x) = \frac{1}{\sqrt{2\pi}}e^{-x^2/2}$$

This function has no symbolic antiderivative, so it can only be integrated numerically. The following code gives an "exact" value of the integral of $f(x)$ from $-\infty$ to a specified value.

```
>>> from scipy.stats import norm
```

```
>>> norm.cdf(1)                    # Integrate f from -infty to 1.
0.84134474606854293
>>> norm.cdf(1) - norm.cdf(-1)     # Integrate f from -1 to 1.
0.68268949213708585
```

Write a function that uses `scipy.stats` to calculate the "exact" value

$$F = \int_{-3}^{2} f(x)dx.$$

Then repeat the following experiment for $n = 5, 10, 15, \ldots, 50$.

1. Use your class from Problems 1–4 with the Legendre polynomials to approximate F using n points and weights. Calculate and record the error of the approximation.
2. Use your class with the Chebyshev polynomials to approximate F using n points and weights. Calculate and record the error of the approximation.

Plot the errors against the number of points and weights n, using a log scale for the y-axis. Finally, plot a horizontal line showing the error of `scipy.integrate.quad()` (which doesn't depend on n).

Multivariate Quadrature

The extension of Gaussian quadrature to higher dimensions is fairly straightforward. The same set of points $\{z_i\}_{i=1}^n$ and weights $\{w_i\}_{i=1}^n$ can be used in each direction, so the only difference from 1-D quadrature is how the function is shifted and scaled. To begin, let $h : \mathbb{R}^2 \to \mathbb{R}$ and define $g : \mathbb{R}^2 \to \mathbb{R}$ by $g(x, y) = h(x, y)/(w(x)w(y))$ so that

$$\int_{-1}^{1}\int_{-1}^{1} h(x,y)dx\,dy. = \int_{-1}^{1}\int_{-1}^{1} g(x,y)w(x)w(y)dx\,dy \approx \sum_{i=1}^{n}\sum_{j=1}^{n} w_i w_j g(z_i, z_j). \tag{10.3}$$

To integrate $f : \mathbb{R}^2 \to \mathbb{R}$ over an arbitrary box $[a_1, b_1] \times [a_2, b_2]$, set

$$h(x,y) = f\left(\frac{b_1 - a_1}{2}x + \frac{a_1 + b_1}{2},\ \frac{b_2 - a_2}{2}y + \frac{a_2 + b_2}{2}\right)$$

so that

$$\int_{a_2}^{b_2}\int_{a_1}^{b_1} f(x)dx\,dy = \frac{(b_1 - a_1)(b_2 - a_2)}{4}\int_{-1}^{1}\int_{-1}^{1} h(x,y)dx\,dy. \tag{10.4}$$

Combining (10.3) and (10.4) gives the final 2-D Gaussian quadrature formula. Compare it to (10.2).

$$\int_{a_2}^{b_2}\int_{a_1}^{b_1} f(x)dx\,dy \approx \frac{(b_1 - a_1)(b_2 - a_2)}{4}\sum_{i=1}^{n}\sum_{j=1}^{n} w_i w_j g(z_i, z_j) \tag{10.5}$$

Problem 6. Write a method for your class that accepts a function $f : \mathbb{R}^2 \to \mathbb{R}$ (which actually accepts two separate arguments, not one array with two elements) and bounds of integration a_1, a_2, b_1, and b_2. Use (10.5) to compute the double integral

$$\int_{a_2}^{b_2} \int_{a_1}^{b_1} f(x)dx\, dy.$$

Validate your method by comparing it `scipy.integrate.nquad()`. Note carefully that this function has slightly different syntax for the bounds of integration.

```
>>> from scipy.integrate import nquad

# Integrate f(x,y) = sin(x) + cos(y) over [-10,10] in x and [-1,1] in y.
>>> f = lambda x, y: np.sin(x) + np.cos(y)
>>> nquad(f, [[-10, 10], [-1, 1]])[0]
33.658839392315855
```

Note

Although Gaussian quadrature can obtain reasonable approximations in lower dimensions, it quickly becomes intractable in higher dimensions due to the curse of dimensionality. In other words, the number of points and weights required to obtain a good approximation becomes so large that Gaussian quadrature become computationally infeasible. For this reason, high-dimensional integrals are often computed via *Monte Carlo methods*, numerical integration techniques based on random sampling. However, quadrature methods are generally significantly more accurate in lower dimensions than Monte Carlo methods.