Lab 12

Gaussian Quadrature

Lab Objective: *Numerical quadrature is an important numerical integration technique. The popular Newton-Cotes quadrature uses uniformly spaced points to approximate the integral, but Runge's phenomenon prevents Newton-Cotes from being effective for many functions. The Gaussian Quadrature method, on the other hand, uses carefully chosen points and weights to mitigate this problem.*

Shifting the Interval of Integration

As with all quadrature methods, we begin by choosing a set of points x_i and weights w_i to approximate an integral.

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} w_i f(x_i).$$

When we do Gaussian quadrature, we are required to choose a weight function $W(x)$. This function determines both the x_i's and the w_i's. Theoretically, the weight function determines a set of orthogonal polynomials to approximate the function f.

The weight function also determines the interval over which the integration will occur. For example, we may choose the weight function as $W(x) = 1$ over $[-1, 1]$ to integrate functions on $[-1, 1]$. To calculate the definite integrate over a the arbitrary interval $[a, b]$, we perform a u-substitution to shift the interval of integration from $[a, b]$ to $[-1, 1]$. Let

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

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

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

so the transformed integral is given by

$$\int_a^b f(x)dx = \frac{b-a}{2}\int_{-1}^{1} f\left(\frac{b-a}{2}u + \frac{a+b}{2}\right) du. \tag{12.1}$$

The general quadrature formula is then given by the following equation.

$$\int_a^b f(x)dx \approx \frac{b-a}{2}\sum_i w_i f\left(\frac{(b-a)}{2}x_i + \frac{(a+b)}{2}\right)$$

Problem 1. Write a function that accepts a function f, interval endpoints a and b, and a keyword argument `plot` that defaults to `False`. Use (12.1) to construct a function g such that

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

(Hint: this can be done in a single line using the `lambda` keyword.)

If `plot` is `True`, plot f over $[a,b]$ and g over $[-1,1]$ in separate subplots. The functions probably have similar shapes, but note the difference in the scaling of the y-axis. Try $f(x) = x^2$ and $f(x) = (x-3)^3$ over $1,4]$ as examples.

Finally, return the new function g.

Integrating with Given Weights and Points

With the new shifted function g from Problem 1, we can write the quadrature formula as follows.

$$\int_a^b f(x)dx \approx \frac{b-a}{2}\sum_i w_i g(x_i) \tag{12.2}$$

Suppose for a given weight function $W(x)$ and a given number of points n, we have the sample points $\mathbf{x} = [x_1,\ldots,x_n]^\mathsf{T}$ and weights $\mathbf{w} = [w_1,\ldots,w_n]^\mathsf{T}$ stored as 1-d NumPy arrays. The summation in (12.2) can then be calculated with the vector multiplication $\mathbf{w}^\mathsf{T} g(\mathbf{x})$, where $g(\mathbf{x}) = [g(x_1),\ldots,g(x_n)]^\mathsf{T}$.

Problem 2. Write a function that accepts a function f, interval endpoints a and b, an array of points $\mathbf{x}$, and an array of weights $\mathbf{w}$. Use (12.2) and your function from Problem 1 to calculate the integral of f over $[a,b]$. Return the value of the integral.

To test this function, use the following 5 points and weights that accompany the constant weight function $W(x) = 1$ (this weight function corresponds to the *Legendre polynomials*). See the next page for their definitions using NumPy.

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	$\frac{322-13\sqrt{70}}{900}$	$\frac{322+13\sqrt{70}}{900}$	$\frac{128}{225}$	$\frac{322+13\sqrt{70}}{900}$	$\frac{322-13\sqrt{70}}{900}$

```python
import numpy as np
from math import sqrt

s1 = 2 * sqrt(10. / 7.)
points = np.array([-sqrt(5 + s1) / 3.,
                   -sqrt(5 - s1) / 3.,
                                   0.,
                    sqrt(5 - s1) / 3.,
                    sqrt(5 + s1) / 3.])

s2 = 13 * sqrt(70)
weights = np.array([(322 - s2) / 900.,
                    (322 + s2) / 900.,
                           128 / 225.,
                    (322 + s2) / 900.,
                    (322 - s2) / 900.])
```

Using these points and weights should yield the approximations

$$\int_{-\pi}^{\pi} \sin(x)dx \approx 0 \quad \text{and} \quad \int_{-\pi}^{\pi} \cos(x)dx \approx 0.000196.$$

Calculating Weights and Points

Calculating an integral when the weights and points are given is straightforward. But how are these weights and points found? There are many publications that will give tables of points for various weight functions. We will demonstrate how to find such a list using the Golub-Welsch algorithm.

The Golub-Welsch Algorithm

The Golub-Welsch algorithm builds a tri-diagonal matrix and finds its eigenvalues. These eigenvalues are the points at which a function is evaluated for Guassian quadrature. The weights are the length of the shifted interval of integration times the first coordinate of each eigenvector squared. We note that finding eigenvalues for a tridiagonal matrix is a well conditioned, relatively painless problem. Using a good eigenvalue solver gives the Golub-Welsch algorithm a complexity of $O(n^2)$. A full treatment of the Golub-Welsch algorithm may be found at `http://gubner.ece.wisc.edu/gaussquad.pdf`.

We mentioned that the choice of weight function corresponds to a class of orthogonal polynomials. An important fact about orthogonal polynomials is that any set of orthogonal polynomials $\{u_i\}_{i=1}^N$ satisfies a three term recurrence relation

$$u_i(x) = (\gamma_{i-1}x - \alpha_i)u_{i-1}(x) - \beta_i u_{i-2}(x)$$

where $u_{-1}(x) = 0$ and $u_0(x) = 1$. The coefficients $\{\gamma_i, \alpha_i, \beta_i\}$ have been calculated for several classes of orthogonal polynomials, and may be determined for an arbitrary class using the procedure found in “Calculation of Gauss Quadrature Rules” by Golub and Welsch. Using these coefficients we may create a tri-diagonal matrix

$$J = \begin{bmatrix} a_1 & b_1 & 0 & 0 & \dots & 0 \\ b_1 & a_2 & b_2 & 0 & \dots & 0 \\ 0 & b_2 & a_3 & b_3 & \dots & 0 \\ \vdots & & & & & \vdots \\ \vdots & & & & & \vdots \\ 0 & \dots & & & & b_{N-1} \\ 0 & \dots & & & b_{N-1} & a_N \end{bmatrix}$$

Where $a_i = \frac{-\beta_i}{\alpha_i}$ and $b_i = (\frac{\gamma_{i+1}}{\alpha_i \alpha_{i+1}})^{\frac{1}{2}}$. This matrix is called the Jacobi matrix.

Problem 3. Write a function that will accept three arrays representing the coefficients $\{\gamma_i, \alpha_i, \beta_i\}$ from the recurrence relation above and return the Jacobi matrix.

The eigenvalues of the Jacobi matrix are the sample points x_i. The length of the shifted interval of integration (in this case 2) times the squares of the first entries of the corresponding eigenvectors give the weights.

Problem 4. The coefficients of the Legendre polynomials (which correspond to the weight function $W(x) = 1$ on $[-1, 1]$) are given by

$$\gamma_k = \frac{k-1}{k} \qquad \alpha_k = \frac{2k-1}{k} \qquad \beta_k = 0$$

Write a function that accepts an integer n representing the number of points to use in the quadrature. Calculate α, β, and γ as above, form the Jacobi matrix, then use it to find the points x_i and weights w_i that correspond to this weight function. Verify that when $n = 5$ the points and weights match the ones given in the first part of this lab.

Problem 5. Write a new function that accepts a function f, bounds a and b, and n for the number of points to use. Use the previously defined functions to estimate $\int_a^b f(x)dx$ using the coefficients of the Legendre polynomials.

This completes our implementation of the Gaussian Quadrature for a particular set orthogonal polynomials.

Numerical Integration with SciPy

There are many other techniques for finding the weights and points for a given weighting function. SciPy's `integrate` module provides general-purpose integration tools. For example, `scipy.integrate.quadrature()` offers a reasonably fast Gaussian quadrature implementation.

Problem 6. The standard normal distribution is an important object of study in probability and statistic. It is defined by the probability density function $p(x) = \frac{1}{\sqrt{2\pi}}e^{-x^2/2}$ (here we are assuming a mean of 0 and a variance of 1). This is a function that cannot be integrated symbolically.

The probability that a normally distributed random variable X will take on a value less than (or equal to) a given value x is

$$P(X \leq x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}}e^{-t^2/2}dt$$

This function is essentially zero for values of x that lie reasonably far from the mean, so we can estimate this probability by integrating from -5 to x instead of from $-\infty$ to x.

Write a function that uses `scipy.integrate.quad()` to estimate the probability that this normally distributed random variable will take a value less than a given number x that lies relatively close to the mean. You can test your result at $x = 1$ by comparing it with the following code:

```python
from scipy.stats import norm
N = norm()                          # Make a standard normal random variable.
N.cdf(1)                            # Integrate the pdf from -infinity to 1.
```