Assignment 1: Gaussian Quadrature

Due: January 16; Pass-Fail deadline: Jan 23

Gaussian Quadrature is a powerful numerical integration method that uses N function evaluations to produce an integral estimate accurate to order $2N-1$, wow!

This approximation uses a specially selected set of sample points x_k and weights w_k to approximate an integral:

$$\int_{-1}^{1} f(x)\ dx \approx \sum_{k=1}^{N} w_k f(x_k). \tag{1}$$

By convention, the points and weights are determined for the integral domain $[-1, 1]$. For a general (finite) domain $[a, b]$ a simple linear change of variables can be done:

$$x'_k = \frac{b-a}{2} x_k + \frac{a+b}{2}, \tag{2}$$

$$w'_k = \frac{b-a}{2} w_k, \tag{3}$$

$$\int_{a}^{b} f(x)\ dx \approx \sum_{k=1}^{N} w'_k f(x'_k). \tag{4}$$

For a given N, the set of $\{x_k\}$ and $\{w_k\}$ are unique and fully specify the scheme. It takes some work to compute them, but once known they never need be computed again! In this assignment you'll use simple root finding and numerical integration to generate the x_k and w_k for yourself, giving you a new high-powered integrator.

1 Find The Sample Points x_k

Gaussian quadrature uses the convenient properties of orthogonal polynomials to construct a $2N-1$ order accurate (it is exact for polynomials of order $2N-1$ or less) integral estimate using only N sample points! The sample points for the N-point method for an integral over the standard domain $[-1, 1]$ must occur exactly at the roots of $P_N(x)$: the N-th degree Legendre polynomial.

Write a `python` function to numerically find *all* the roots of a given Legendre polynomial. The first few Legendre polynomials are given in Table 1, use your function to find the roots for all the polynomials in the table. These will be your sample points x_k. They should be as accurate as possible, within 10^{-14}.

Remember: $P_n(x)$ has n distinct roots. Every $P_n(x)$ is even or odd, so $P_n(x_*) = 0 \implies P_n(-x_*) = 0$. You can test your results by plugging them back into the Legendre polynomials, you should get 0!

0	1
1	x
2	$(3x^2 - 1)/2$
3	$(5x^3 - 3x)/2$
4	$(35x^4 - 30x^2 + 3)/8$
5	$(63x^5 - 70x^3 + 15x)/8$
6	$(231x^6 - 315x^4 + 105x^2 - 5)/16$
7	$(429x^7 - 693x^5 + 315x^3 - 35x)/16$

Table 1: First few Legendre polynomials

2 Find The Weights w_k

Now that you have the sample points for a few N, you can compute the weights for those N as well! Gaussian quadrature constructs a degree $N-1$ interpolating polynomial $\Phi(x)$ for the N sample points, which is used to determine the weights w_k. The interpolating polynomial is formed from N indicator polynomials $\phi_k(x)$, each of degree $N-1$ with the property $\phi_m(x_k) = \delta_{mk}$. That is, for scheme N, each indicator polynomial is 1 at one sample point and 0 at every other sample point.

The indicator polynomials have the formula:

$$\phi_k(x) = \prod_{m=1\ldots N, m\neq k} \frac{x - x_m}{x_k - x_m} \tag{5}$$

$$= \frac{x - x_1}{x_k - x_1} \times \cdots \times \frac{x - x_{k-1}}{x_k - x_{k-1}} \times \frac{x - x_{k+1}}{x_k - x_{k+1}} \times \cdots \times \frac{x - x_N}{x_k - x_N} \tag{6}$$

Write a `python` function to compute the indicator polynomials $\phi_k(x)$ for a given set of sample points, and make a plot of $P_N(x)$ and all $\phi_k(x)$ for $N = 7$. Mark the sample points (the roots of $P_7(x)$) on your plot.

The interpolating polynomial $\Phi(x)$ is simply:

$$\Phi(x) = \sum_{k=1}^{N} f(x_k)\ \phi_k(x). \tag{7}$$

Gaussian quadrature approximates the integral of $f(x)$ with the integral of $\Phi(x)$.

$$\int_{-1}^{1} f(x)\ dx \approx \int_{-1}^{1} \Phi(x)\ dx = \sum_{k=1}^{N} \left[f(x_k) \left(\int_{-1}^{1} \phi_k(x)\ dx \right) \right] \tag{8}$$

Now we can see the weights must be:

$$w_k = \int_{-1}^{1} \phi_k(x)\ dx. \tag{9}$$

Write a `python` function to compute the weights numerically w_k for each set of sample points you found in Part 1. Use one of the simple integrators demonstrated in class. You may use the sample code. Your weights should be as accurate as possible.

3 Integrate!

You now have, for several values of N, a set of sample points x_k and a set of weights w_k defining the Gaussian quadrature scheme, time to use them!

High-order schemes put higher smoothness requirements on the integrand. For complicated functions, it is often more convenient to break the integration domain into sub-intervals, and apply Gaussian quadrature to each sub-interval separately:

$$\int_a^b f(x)\ dx = \sum_{j=0}^{N_{\text{sub}}-1} \int_{a_j}^{a_{j+1}} f(x)dx \tag{10}$$

$$a_j = a + \frac{b-a}{N_{\text{sub}}} j \tag{11}$$

An N-point Gaussian quadrature applied to N_{sub} sub-intervals will then use $N \times N_{\text{sub}}$ function evaluations.

Write a `python` function that integrates a given function $f(x)$ over (a, b) using Gaussian Quadrature for each of the schemes you have sample points and weights for. This function should also take an argument `Nsub` which indicates the number of sub-intervals to divide the integration domain into. For each integral below, make a convergence plot showing the error in the integral as a function of number of evaluations for each of your Gaussian Quadrature schemes, the maximum number of evaluations should be several thousand. The plot should have log-scaled horizontal and vertical axes to compare the convergence rates.

Test functions:

- $\int_0^1 e^x\ dx$
- $\int_1^{10} \left(x^{1/3} + \frac{1}{1+100(x-5)^2}\right) dx$ It takes some evaluations before the schemes start to converge here. Why? What is special about the number of points needed?
- $\int_{-\pi}^{4-\pi} x^5|x|\ dx$ Some schemes converge at the expected rate, some slower, why might that be?