Quadrature Methods for Numerical Integration

Undergraduate Capstone Research Project Marshall University Fall 2017

Dillon Buskirk **Advisor: Dr. Scott Sarra**

Introduction

Numerical integration is necessary for many different integrals. An example of this is when an integrand, such as $f(x) = e^{x^2}$, can not be expressed in the form of an elementary function on a given interval. *Quadrature methods* use the equation

$$I_N(f) = \sum_{j=0}^{N} \omega_j f(x_j)$$

to numerically approximate the integral as the sum of *quadrature weights* times the integrand evaluated at N different *quadrature points* x_j.

I look at two different methods of numerical integration on the interval $[-1,1]$: *Gaussian quadrature* and *Clenshaw-Curtis quadrature*. Gaussian quadrature has the highest *degree of polynomial accuracy* out of any quadrature method and can exactly integrate polynomials of degree up to $2N+1$.

Clenshaw-Curtis has a degree of accuracy almost half of that of Gaussian $(N+1)$. My goal is to use each method to evaluate different integrals used by Trefethen and determine if this "factor-of-two" advantage of Gaussian quadrature is as considerable as it sounds.[3]

Orthogonal Polynomials

The quadrature points (x_j) are obtained from *orthogonal polynomials*, the *Legendre Polynomial* and *Chebyshev Polynomial*. These have three term recursion relationships.

Let $p_k(x)_{k=0}^{n}$ be a set of orthogonal polynomials. Since they are orthogonal, taking the inner product gives us the relation...

$$\int_a^b w(x)p_m(x)p_n(x)dx = \delta_{mn}c_n$$

where...

$$\delta_{mn} = \begin{cases} 1 & n = m \\ 0 & n \neq m \end{cases}$$

Gaussian Quadrature

Gaussian quadrature uses the zeros of the Legendre polynomial for its quadrature points. Instead of calculating the zeros of the Legendre polynomial implicitly, they can be found by solving a *tridiagonal eigenvalue problem*.

The relationship between the quadrature points and eigenvalues, and weights and eigenvectors is discussed by Trefethen.[5] Below is the Matlab code implementing the $(N+1)$ point Gaussian quadrature of an integrand f.[3]

```
function I = gauss(f,n)
beta = .5./sqrt(1-(2*(1:n)).^(-2)); %3TRR
T = diag(beta,1) + diag(beta,-1); %Jacobi
[V,D] = eig(T);  % eigen decomp.
x = diag(D); [x,i] = sort(x);  % nodes
w = 2*V(1,i).^2;   %weights
I = w*feval(f,x); %the integral
```

Figures 1 compares the error of the two quadrature methods for increasing number of quadrature points. For the integrand of the simple polynomial function in Figure 1, the $2N+1$ advantage of Gaussian quadrature is obvious since the 20th order polynomial is approximated to machine precision with 9 quadrature points. However, integrals requiring numerical methods to evaluate are not usually so basic.

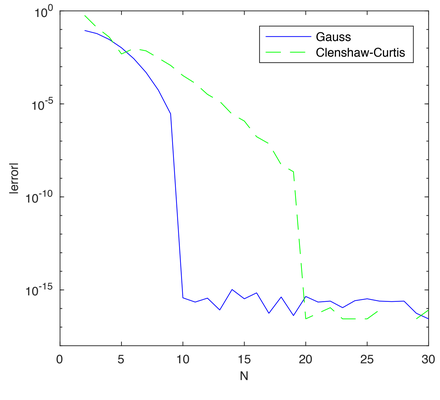

Figure 1: Error convergence for $f(x) = x^{20}$.

Clenshaw–Curtis Quadrature

Clenshaw-Curtis quadrature uses *Chebyshev-Gauss-Lobatto (CGL) points* for its quadrature points. These are the extrema and endpoints of the Chebyshev polynomial. This approximation becomes the sum of the product of the Chebyshev coefficients[4] and the quadrature weights.[2]

Using a *Fast Fourier Transform* when solving for the expansion coefficients reduces the computational cost of this implementation. Using the built in Fast Cosine Transform in Python, quicker results could be obtained. Below is the code implementing $(N+1)$ point Clenshaw-Curtis quadrature on an integrand f.[3]

```
function I = clenshaw_curtis(f,n)
x = cos(pi*(0:n)'/n);  %CGL points
fx = feval(f,x)/(2*n); %f evaluated
g = real(fft(fx([1:n+1 n:-1:2])));  %FFT
%Cheby Coeff.
a = [g(1); g(2:n)+g(2*n:-1:n+2); g(n+1)];
%weight vector
w = 0*a'; w(1:2:end) = 2./(1-(0:2:n).^2);
I = w*a;   %the integral
```

In Figure 2, both quadrature methods have similar error curves and the "factor-of-two" advantage is not evident for this integral with a non-polynomial integrand.

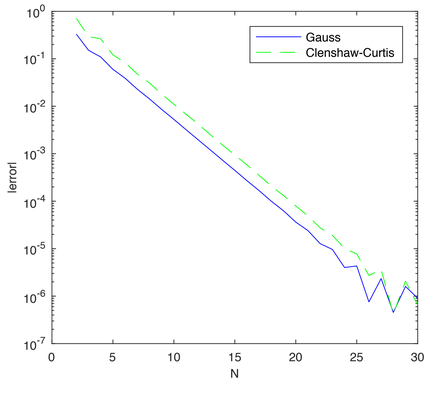

Figure 2: Error convergence for $f(x) = \frac{1}{(1+16x^2)}$.

Results

Nearly all of the integrands in the Trefethen paper gave error curves that were similar for each quadrature method and the "factor-of-two" advantage proved to be fallacious. Each quadrature method is too complex to be judged only by their degree of polynomial accuracy and saying there is a "factor-of-two" advantage is misleading.

Since each quadrature method's accuracy was nearly the same, computational efficiency is necessary to take into consideration. In Figure 3, as N becomes large, the computational advantage of Clenshaw-Curtis quadrature is enormous. The integrand used to compare the times was used due to the accuracy of each implementation being near identical.[1]

Figure 3: Averaged computation time.

References

[1] C. W. Clenshaw and A. R. Curtis, *A method for numerical integration on an automatic computer*, Numerische Mathematik **2** (1960), no. 1, 197–205.

[2] Scott A. Sarra, *Numerical analysis and scientific computing with Python, Matlab, and C++*, 2017, unpublished notes.

[3] Lloyd N. Trefethen, *Is Gauss quadrature better than Clenshaw–Curtis?*, SIAM Review **50** (2008), no. 1, 67–87.

[4] _____, *Approximation theory and approximation practice*, Society for Industrial and Applied Mathematics, 2012.

[5] Lloyd N. Trefethen and David Bau III, *Numerical linear algebra*, SIAM: Society for Industrial and Applied Mathematics, 1997.