Coding Gaussian Quadrature Integral Approximation

Math 351: Numerical Analysis Project

Noah Mifsud and Jeremy Guenza-Marcus

March 12th 2020

For our Numerical Analysis project we wrote a function in Mathematica which, when given any function of a single variable, will approximate its integral over the interval a to b using Gaussian Quadrature. The user can input a function $F(x)$, integration limits a, b, and a number n specifying the number of nodes with which the program will approximate the integral. Our program then outputs a numerical value for the Gaussian quadrature approximation of the integral and a value for the error of this approximation, along with other relevant information. The following document will outline some theory behind Gaussian quadrature approximation as it's being done by our program, provide a description of how the program works, and end with some interesting properties we observed through our testing process.

Theory

Gaussian Quadrature is a type of numerical integral approximation which involves generating unevenly spaced nodes and corresponding weights to achieve high accuracy. For our project, we decided to generate these nodes and weights using the Legendre Polynomials. Legendre Polynomials are a special set of recursively generated functions whose roots can be used as nodes for Gaussian quadrature on the interval $[-1, 1]$. Fortunately, we can always use a change of variables to map this approximation onto an interval $[a, b]$. We outline the process of Gaussian approximation of an integral over interval $[a, b]$ below.

Given an integral that we want to approximate using nodes $x_0, x_1, ..., x_n$ and weights $A_0, A_1, ..., A_n$ in this way:

$$\int_a^b F(x)dx \approx A_0F(x_0) + A_1F(x_1) + ... + A_nF(x_n)$$

we first perform a change of variables $[a, b] \to [-1, 1]$ using the function

$$g(\xi) = \frac{(a-b)}{2}F\left(\frac{(a-b)}{2}\xi + \frac{(a+b)}{2}\right)$$

so we have the integral

$$\int_a^b F(x)dx = \int_{-1}^1 g(\xi)d\xi \approx A_0g(x_0) + A_1g(x_1) + ... + A_ng(x_n)$$

We can then generate the nodes and weights using the Legendre polynomials. The Legendre polynomials of degree n, denoted $P_n(x)$, have properties:

- $P_n(x)$ has degree n
- If $i \neq j$, $\int_{-1}^1 P_i(x) * P_j(x) = 0$
- They span L_2, in other words, $[P_0(x), P_1(x), .., P_n(x)]$ has the same vector span as $[1, x, x^2, ..., x^n]$

Properties two and three can be combined to show

$$\int_{-1}^{1} P_n(x) * f(x) = 0$$

for any polynomial $f(x)$ of degree less than or equal to n because any such polynomial can be written as a linear combination of Legendre Polynomials.[2] As such, they satisfy the properties necessary for integral approximation using an arbitrary function $q(x)$ as in chapter 6 of the book. We then have that the Legendre polynomial of degree n can be generated recursively using

$$P_n(x) = \left(\frac{2n-1}{n}\right) x P_{n-1}(x) - \left(\frac{n-1}{n}\right) P_{n-2}(x)$$

with

$$P_0(x) = 1$$
$$P_1(x) = x$$

The nodes of our approximation, $(x_0, x_1, ..., x_n)$, are given by the roots of the n^{th} degree Legendre polynomial.

The weights can be calculated using the function [1]

$$W(x) = \frac{2}{(1-x^2)(\frac{d}{dx}P_n(x))^2}$$

where the i^{th} weight, A_i, is given by

$$A_i = W(x_i)$$

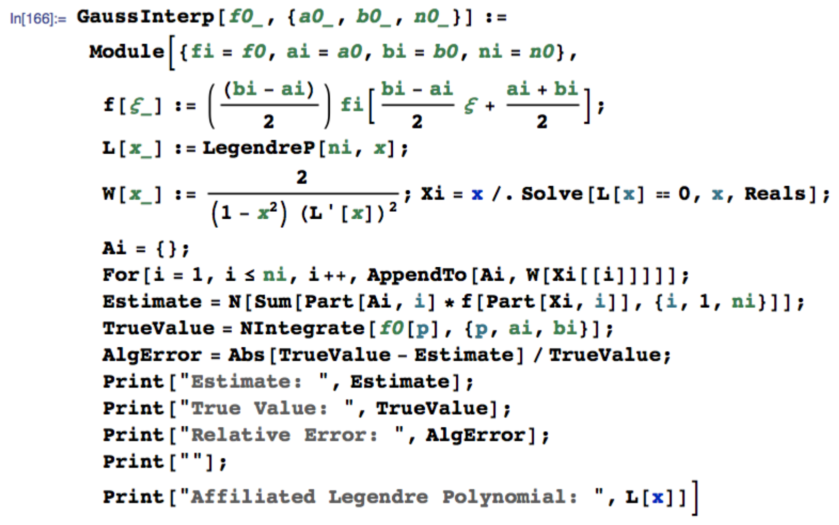
We can then approximate the integral using

$$\int_a^b F(x)dx = \int_{-1}^{1} g(\xi)d\xi \approx A_0 g(x_0) + A_1 g(x_1) + ... + A_n g(x_n) = \sum_{i=0}^{n} A_i * f(x_i)$$

In the next section we will explain how to we implimented this into Mathematica.

Program

Our program essentially follows the process outlined above.



```
In[166]:= GaussInterp[f0_, {a0_, b0_, n0_}] :=
  Module[{fi = f0, ai = a0, bi = b0, ni = n0},
   f[ξ_] := ((bi - ai)/2) fi[(bi - ai)/2 ξ + (ai + bi)/2];
   L[x_] := LegendreP[ni, x];
   W[x_] := 2/((1 - x^2) (L'[x])^2); Xi = x /. Solve[L[x] == 0, x, Reals];
   Ai = {};
   For[i = 1, i ≤ ni, i++, AppendTo[Ai, W[Xi[[i]]]]];
   Estimate = N[Sum[Part[Ai, i] * f[Part[Xi, i]], {i, 1, ni}]];
   TrueValue = NIntegrate[f0[p], {p, ai, bi}];
   AlgError = Abs[TrueValue - Estimate] / TrueValue;
   Print["Estimate:  ", Estimate];
   Print["True Value:  ", TrueValue];
   Print["Relative Error:  ", AlgError];
   Print[""];
   Print["Affiliated Legendre Polynomial:  ", L[x]]]
```

It takes as input four items: $F0$-the function over which we are estimating the integral, $a0$-the lower bound of the integration, $b0$-the upper bound of the integration and $n0$-the desired number of nodes. It then creates a module where each of these inputs is reassigned so we can freely manipulate them within the bounds of the program.

With the module created, it begins by performing the necessary change of variables and defining a new function, $f(x)$ which takes values on the interval $[-1, 1]$ and outputs equivalent values of the input-function $F0$ on its interval $[a, b]$. With this function now defined on $[-1, 1]$ we can perform Gaussian quadrature approximation on it using Legendre polynomials.

Our program first generates the n^{th} Legendre polynomial using Mathematica's built-in function LegendreP[n,x] and assigns it to a new function named $L[x]$. It then defines a function W[x] (as in the theory section):

$$W(x) = \frac{2}{(1-x^2)(L'(x))^2}$$

It finds the nodes $x_0, x_1, ..., x_n$, being the roots of L[x], using Mathematica's built in Solve function. By calling Solve on L[x]==0, limiting it to real solutions, and prefacing it with 'x/.' we can make Solve output a list of every solution (every x value which yields a node). Each entry in this list is a just a number so this single line generates the list $[x_0, x_1, ..., x_n]$. Our program then creates an empty list Ai. It populates this list with a For-loop over values of i from 1 to n. At each step i, the For-loop appends the value $W[xi]$ to the list Ai. Thus, it creates a list of the weights Ai, $[A_0, A_1, ..., A_n]$. With an ordered list of the nodes $[x_0, x_1, ..., x_n]$ and the weights $[A_0, A_1, ..., A_n]$ it then calculates the integral approximation using the equation

$$\int_a^b F(x)dx = \int_{-1}^1 f(\xi)d\xi \approx A_0 f(x_0) + A_1 f(x_1) + ... + A_n f(x_n) = \sum_{i=0}^{n} A_i * f(x_i)$$

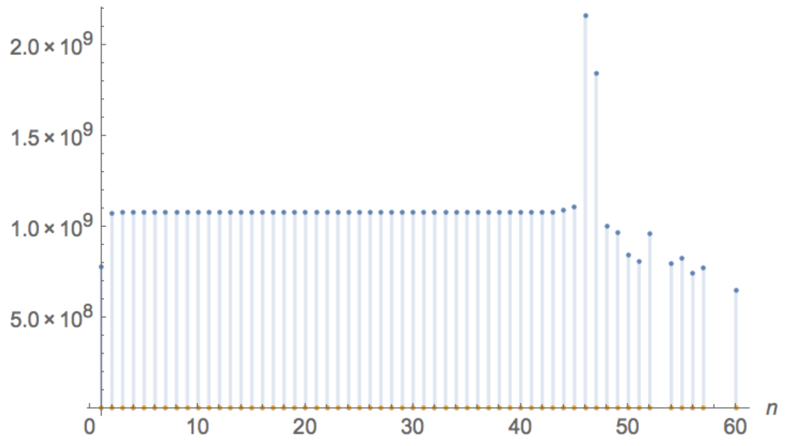
where f is the internally defined function still under the influence of the change of variables.

To check the accuracy of this estimate, our program uses Mathematica's NIntegrate function to find the 'actual' value of the integral then calculates the relative error (absolute value of the difference between the estimate and the 'actual' value, divided said 'actual' value). Finally, it returns its estimated value for the integral, the 'actual' value, the calculated relative error and the relevant Legendre polynomial.

Testing and Results

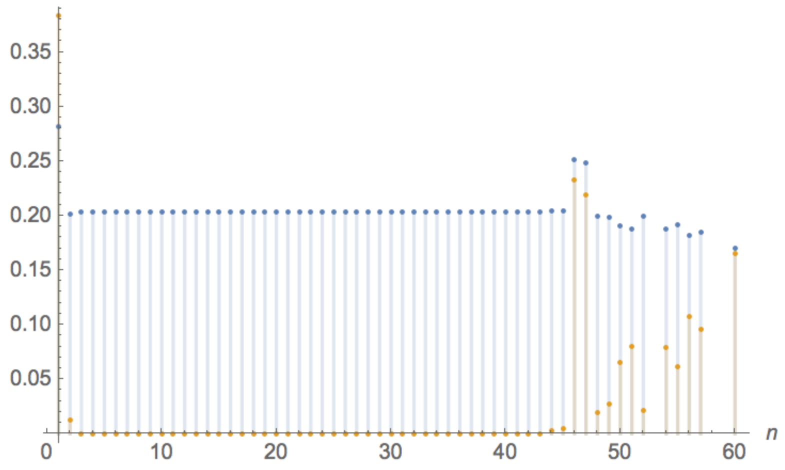
We first tested the program for a bunch of different functions and limit values to confirm it was working properly. We were consistently able to get low errors in our estimates for reasonable values of n.

To get an idea of this program's effectiveness we created a plot that illustrates estimates of a polynomial of degree 9 as a function of the number of partitions n. The blue dots are the estimated values while the orange ones are relative errors.



Observe that the program converges quickly to a consistent estimate of the integral for values of n below 45. At $n = 45$ we start to see progressively more extreme fluctuations in the estimate and past $n = 50$ the program begins failing to compute some estimates. We predict this might be related to the Runge effect for numerical approximation whereby higher numbers of nodes induce wild oscillations in the output. We also think the high order Legendre polynomials (with their very large coefficients) might be causing computation inaccuracies. Based on the error outputs during these high-n-value computations we infer that the failed computations are resulting from the appearance of complex infinite values in the computation of $W[x_i]$.

We can compare this graph to one generated by estimating the integral of Sinc[x] in the same formulation.



Note that again the estimate is very consistent up until $n = 45$ where it starts to rapidly break down. Additionally, the increases in error associated with this breakdown are much more evident. This means there is some characteristic of either our program, or of the process of gaussian quadrature approximation in general, which is not compatible with n values above 45.

We wanted to test our program's ability to generate nodes and weights against Mathematica's own GaussianQuadratureWeights function. However, we were unable to make our program output it's calculated nodes and weights in a reasonable form.

References

[1] Abramowitz and Stegun 1972, P 887

[2] James Keesling, University of Florida https://people.clas.ufl.edu/kees/files/GaussianQuadrature.pdf

[3] Numerical Mathematics and Computing, 6th Edition, Ward Cheney and David Kincaid