Project 3: Numerical Integration Using Orthogonal Polynomials

Harper Wang, Cameron Stephens

November 2022

Contents

1 Introduction

There are several major families of orthogonal polynomials. This report will examine the Legendre polynomials, constructed of the interval [-1,1], and will use the fifth degree Legendre polynomial to create a Gaussian quadrature that can be used to integrate an arbitrary function of both one and two variables over intervals not restricted to [-1,1].

2 Mathematical Issues

2.1 Part I: Generating the Gaussian Abscissas and Weights

2.2 The First Five Legendre Polynomials

With $P_0 = 1$, the first five Legendre polynomials over the interval [-1,1] can be calculated over using the python library scipy and is as follows.

$$P_1 = x$$
$$P_2 = 1.5x^2 - 0.5$$
$$P_3 = 2.5x^3 - 1.5x$$
$$P_4 = 4.375x^4 - 3.75x^2 + .0375$$
$$P_5 = 7.875x^5 - 8.75x^3 + 1.875x$$

2.3 Calculating the Gaussian Quadrature

We will be using the Gaussian quadrature for a 5-point rule. This quadrature will sample the function at 5 different points, with each point corresponding to a root of the 5th degree Legendre polynomial. The weights are calculated by integrating each Lagrange basis polynomial:

$$l_i(t) = \Pi_{j=1, j\neq i}^{5} \frac{t - x_j}{x_i - x_j},$$

with each weight being

$$w_i = \int_{-1}^{1} l_i(t)dt,$$

where the x_i values are the roots of the 5th-degree Legendre polynomial.

2.4 Integration Based on the 5-Point Gaussian Quadrature

2.4.1 Single Integrals

Once the weights and abscissas for the 5-point Gaussian quadrature are found, the integral of any arbitrary function on the interval from [-1,1] can be approx-

imated by

$$\int_{-1}^{1} f(x)dx \approx \sum_{i=1}^{5} w_i f(x_i).$$

We can define a change of variables in which this holds true for any range [a,b].

$$\int_{a}^{b} f(x)dx \approx \frac{b-a}{2} \sum_{i=1}^{5} w_i f(\frac{b-a}{2}x_i + \frac{b+a}{2}).$$

2.4.2 Double Integrals

It follows that if single integrals can be evaluated using this technique, double integrals can as well. For an arbitrary function u(x,y), its integral can be approximated as follows:

$$\int_{-1}^{1}\int_{-1}^{1} u(x,y)dydx \approx \sum_{i=1}^{5} w_i \sum_{j=1}^{5} w_j u(x_i, y_j).$$

If we want to integrate this function over the range [a,b] for x, and the range [g(x),f(x)] for y then we can also find an appropriate change of variables.

$$\int_{a}^{b}\int_{g(x)}^{f(x)} u(x,y)dydx \approx \frac{b-a}{2} \sum_{i=1}^{5} w_i \frac{h(x_i)-g(x_i)}{2} \sum_{j=1}^{5} w_j u(\frac{b-a}{2}x_i + \frac{b+a}{2}, \frac{h(x_i)-g(x_i)}{2}y_j + \frac{h(x_i)+g(x_i)}{2}).$$

2.5 Clenshaw-Curtis Quadrature

The Clenshaw-Curtis quadrature is another type of quadrature that makes use of orthogonal functions. The formula for this quadrature is as follows

$$\int_{0}^{\pi} f(cos(\theta)sin(\theta)d\theta \approx a_0 + \sum_{k=1}^{\frac{N}{2}-1} \frac{2a_{2k}}{1-(2k)^2} + \frac{a_N}{1-N^2}$$

where a_k and a_2k are found to be Fourier coefficients of the cosine series for $f(\cos\theta$. In the 1960 paper, A method for numerical integration on an automatic computer, by Clenshaw and Curtis; they derive a quadrature using the Chebyshev polynomials and and evaluate the integral as a sum of the function evaluated at the Chebyshev nodes of this function.

2.5.1 What makes this Quadrature Good?

On paper, this quadrature is not as good as Gaussian quadrature, but when using the fast Fourier transform to calculate the coefficients of this quadrature it becomes much faster than calculating an integral using Gaussian quadrature.

3 Programming Issues

3.1 Part I: Generating the Gaussian Abscissas and Weights

3.2 Roots of the First Five Legendre Polynomials

Using the numpy.roots function we can find the roots for all of these polynomials.

Roots for P1
0.0000000000000000

Roots for P2
0.5773502691896250
-0.5773502691896250

Roots for P3
0.7745966692414830
-0.7745966692414830
0.0000000000000000

Roots for P4
0.8611363115940530
-0.8611363115940520
0.3399810435848560
-0.3399810435848560

Roots for P5
-0.9061798459386630
-0.5384693101056830
0.9061798459386640
0.5384693101056830
0.0000000000000000

3.3 Calculating the Gaussian Quadrature

The Gaussian quadrature is created using the formulas in the previous section of this report. The Lagrange basis polynomials can easily be constructed using a basic loop once the roots of P_5 are known.

```
def li(i,t):
    i = i-1 # to correct for indexing from 0
    li = 1
    x = [-9.061798459386638527e-01,
         -5.384693101056831077e-01,
         9.061798459386645188e-01,
         5.384693101056831077e-01,
```

```
          0.00000000000000000e+00]
    for j in range(0,5):
        if j != i:
            li = li*(t-x[j])/(x[i]-x[j])
    return li
```

Once the basis polynomials are constructed,they could be integrated by hand since they are just polynomials, but due to the difficulty in calculating the coefficients, we simply integrate them using a built-in routine to find the weights and collect these weights into an array for later use.

```
wi = []
for i in range(1,6):
    w = scipy.integrate.quad(lambda x: li(i,x), -1, 1)
    wi = np.append(wi,w[0])
```

This gives us the following weights for our quadrature.

Weights
0.2369268850561890
0.4786286704993660
0.2369268850561880
0.4786286704993670
0.5688888888888880

3.4 Integration Based on the 5-Point Gaussian Quadrature

3.4.1 Single Integrals

Using the equation for single integrals of an arbitrary function on any interval [a,b], the following code to evaluate single integrals was constructed.

```
def my_single_integral(f,a,b):
    x = [-9.061798459386638527e-01,
         -5.384693101056831077e-01,
         9.061798459386645188e-01,
         5.384693101056831077e-01,
         0.000000000000000000e+00]
    w = [2.369268850561891959e-01,
         4.786286704993663599e-01,
         2.369268850561884743e-01,
         4.786286704993673036e-01,
         5.688888888888885553e-01,]
    #Apply Change of interval
    I = 0
    for i in range(0,len(x)):
        I = I + w[i]*f(((b-a)*0.5)*x[i]+(a+b)*0.5)
    return I*(b-a)*0.5
```

3.4.2 Double Integrals

Using the equation for an arbitrary function of two variables over an interval with the domain [a,b] for x, and the domain [g(x),f(x)] for y, the following code was constructed to evaluate double integrals.

```
def my_double_integral(u,A,B,g,h):
    x = [-9.061798459386638527e-01,
         -5.384693101056831077e-01,
         9.061798459386645188e-01,
         5.384693101056831077e-01,
         0.000000000000000000e+00]
    w = [2.369268850561891959e-01,
         4.786286704993663599e-01,
         2.369268850561884743e-01,
         4.786286704993673036e-01,
         5.688888888888885553e-01,]
    #explicitly make the change of variable in x to find the bounds of
    integration over y
    Xi = x
    dxdXi = (B-A)/2
    I = 0
    for i in range(0,5):
        Xi[i] = ((B-A)*0.5)*x[i]+(A+B)*0.5
        I = I + w[i]*my_single_integral(lambda y: u(Xi[i],y), g(Xi[
    i]), h(Xi[i]))
    return I*dxdXi
```

This function makes use of the single integral calculated of the previous section. Meaning this function evaluates the double sum from the previous section as a sequence of two embedded for loops.

3.5 Part III: Applying the Double Integral Evaluator

This section will be looking at two different ways to make use of the above function to evaluate the double integral:

$$\int_0^{\pi/4} \int_{\sin x}^{\pi/4} \frac{2y \sin x (\cos x)^2}{\sqrt{1-y^2}}$$

3.5.1 Evaluation Using Gaussian Quadrature

When this integral is evaluated using the Gaussian quadrature (by evaluating mydoubleintegral(u, 0, np.pi/4, lambda x: 0, np.sin)), we evaluate the integral to be 0.31134846460769783. This is accurate to 6 decimal place. Keeping in mind that this degree of accuracy was obtained with only 25 evaluations of our function, this is very impressive.

3.5.2 Evaluation by Dividing up the Domains

What if we divided the x domain up into 10 evenly spaced sub intervals and then divided up the y domain into 10 more sub intervals? How much more

accuracy could we obtain by doing this? In order to do this dividing up the x domain was simple enough, but because the domain of the y values were given by functions, we actually had to divide the y domain up based on contours that were multiples of $\frac{sin(x)}{10}$. So, after dividing the domain of this integral up into 100 pieces we get that not each piece is equal in size. Then for each of the 100 sub domains of this integral, we used the Gaussian quadrature to evaluate the integral within the subdomain and added up the values of the integral of each subdomain. This was done with the following code.

```
def u(x,y):
    return (2*y*np.sin(x)+(np.cos(x))**2)/np.sqrt(1-y**2)
a = 0
b = np.pi/4
nIntervals = 10
def h(x):
    return np.sin(x)
def g(x):
    return 0
def hj(x,j):
    return j*np.sin(x)/10
# x interval
xdom = np.linspace(a,b,nIntervals + 1)
#loop through the X intervals
I = 0
for i in range(0,len(xdom)-1):
    aNew = xdom[i]
    bNew = xdom[i+1]
    #define the new y interval
    for j in range(len(xdom)-1):
        xplot = np.linspace(aNew, bNew,100)
        I = I + my_double_integral(u, aNew, bNew, lambda x: hj(x,j)
    , lambda x: hj(x,j+1))
```

The result is that we got the total integral to be

$$\int_0^{\pi/4} \int_{\sin x}^{\pi/4} \frac{2y \sin x (\cos x)^2}{\sqrt{1-y^2}} \approx 0.3113485472432878.$$

4 Conclusion

Gaussian quadrature is an effective way to integrate general fictions, not limited to the integration of polynomials. One can compound Gaussian quadrature and use it to integrate compound integrals as well. This quadrature can evaluate double integrals to surprising precision, given that it only samples the function at 25 points over the entire domain. But, this can be made better by dividing up the integral into many sub-domains and integrating each of the sub-domains using the Gaussian quadrature.
These two answers agree to the first six digits of precision. Since the more complicated quadrature is built off of the first, but samples the function 100 times more, it is safe to assume that (since we are still not dividing up the interval on the order to where machine error is greater than quadrature error) the second method is much more accurate than the first method for evaluating this integral. The more accurate method is the Clenshaw-Curtis Quadrature in this case because the function we are evaluating is not a polynomial Clenshaw-Curtis Quadrature converges at a faster rate than Gaussian Quadrature for functions that are composed of trigonometric functions.