Numerical Integration

MAD4401 - Numerical Analysis

Devere Anthony Weaver

October 28, 2022

4.3 - Elements of Numerical Integration

Numerical quadrature is the name of the basic method that allows us to approximate a definite integral. It makes use of a summation and using a set of $n + 1$ distinct nodes on the given interval. Then one integrates the Largrange iterpolating polynomial and its truncation error term.

Note: The derivation is relatively simple, consult the textbook as needed.

The result of the derivation gives us the following quadrature formula,

$$\int_a^b f(x)dx \approx \sum_{i=0}^{n} a_i f(x_i),$$

where

$$a_i = \int_a^b L_i(x)dx, \;\; i = 0, 1, ...n.$$

The error term for this quadrature formula is given by,

$$E(f) = \frac{1}{(n+1)!} \int_a^b \prod_{i=0}^{n} (x - x_i) f^{(n+1)}(\xi(x))dx.$$

The following formulas are produced using first and second Largrange polynomials with *equally spaced nodes.*

The Trapezoidal Rule

$$\int_a^b f(x)dx = \frac{h}{2}\left[f(x_0) + f(x_1)\right] - \frac{h^3}{12} f''(\xi)$$

Check the image on p.193 of the text to visualize why this is called the trapezoidal rule. When itegrating a function with positives values, it's definite integral is approximated by the area in a trapezoid.

Since the Trapezoidal Rule involves f'', it will give the exact result when applied to a function whose second derivative is identically zero (any polynomial of degree one or less).

Simpson's Rule

Again, the derivation is in the textbook, consult as needed. The derivation gives Simpson's rule,

$$\int_{x_0}^{x_2} f(x)dx = \frac{h}{3}\left[f(x_0) + 4f(x_1) + f(x_2)\right] - \frac{h^5}{90} f^{(4)}(\xi).$$

The error term involves the fourth derivative of f so it gives an exact result when applied to any polynomials of degree three or less.

Example 1 Compare the Trapezoidal rule and Simpson's rule aapproximations to $\int_0^2 f(x)dx$ when f(x) is:

1. x^2
2. x^4
3. $(x+1)^{-1}$
4. $\sqrt{1+x^2}$
5. $\sin x$
6. e^x.

On this interval for the Trapezoidal rule we have, $x_0 = 0$, $x_1 = 2$, and $h = x_1 - x_0$. For Simpson's rule we have $x_0 = 0$, $x_2 = 2$, $x_1 = x_0 + h$, and $h = (x_0 + x_2)/2$.

[1]:
```
import numpy as np
import math
import sympy
```

[2]:
```
# implement Trapezoidal Rule
# TODO: Implement these functions such that they also compute the
# error terms as well and return these different values as
# a tuple
def trapezoidalRule(f, x0,x1):
    """
    Implementation of the Trapezoidal Rule used to approximate
    the definite integral of functions by using
    first Largrange polynomials with equally spaced nodes;
    x0 = lower endpoint, x1 = upper endpoint,
    f = function object representing mathematical
    function.
    """
    h = x1-x0
    return (h/2)*(f(x0)+f(x1))
```

[3]:
```
# implement Simpson's Rule
def simpsonsRule(f, x0, x2):
    """
    Implementation of the Simpson's Rule used to approximate
    the definite integral of functions by using
    second Largrange polynomials with equally spaced nodes;
    x0 = lower endpoint, x1 = upper endpoint,
    f = function object representing mathematical
    function.
    """
    h = (x0+x2)/2
    x1 = x0+h
    return (h/3)*(f(x0)+4*f(x1)+f(x2))
```

[4]:
```
# implement functions from problem
def a(x):
    return math.pow(x,2)

def b(x):
    return math.pow(x,4)

def c(x):
    return math.pow(x+1,-1)

def d(x):
    return math.sqrt(1+math.pow(x,2))

def e(x):
    return math.sin(x)

def f(x):
    return math.exp(x)
```

[5]:
```
# lets create a vector of function objects and
# then create vectorized versions of the rules
funcs = np.array([a,b,c,d,e,f])
trapezoidalRuleV = np.vectorize(trapezoidalRule)
simpsonsRuleV = np.vectorize(simpsonsRule)
```

[6]:
```
# compute trapezoidal rule for given functions
trapezoidalRuleV(funcs,0,2)
```

[6]:
```
array([ 4.        , 16.        ,  1.33333333,  3.23606798,  0.90929743,
        8.3890561 ])
```

[7]:
```
# compute simpsons rule for given functions
simpsonsRuleV(funcs,0,2)
```

[7]:
```
array([2.66666667, 6.66666667, 1.11111111, 2.96430741, 1.42506046,
       6.4207278 ])
```

[8]:
```
# symbolic evaluation to check one of the function
x = sympy.symbols('x')
expr = sympy.sqrt(1+x**2)
```

```
exprInt = sympy.integrate(expr, (x,0,2))
sympy.N(exprInt)
```

[8]:
```
2.9578857150892
```

As is shown above, these two integration approximations can give some very different results. For the set of approximations above, Simpson's rule gave more accurate approximations, likely due to the fact that it has more evaluation points.

Measuring Precision

Definition: The *degree of accuracy*, or *precision*, of a quadrature formula is the largest positive integer n such that the formual is exact for x^k for each $k = 0, 1, ...n$.

Closed Newton-Cotes Formulas

The $(n+1)$-point closed Newton-Cotes formula uses nodes,

- $x_i = x_0 + ih, \text{ for } i = 0, 1, ...n$
- $x_0 = a$
- $x_n = b$
- $h = (b-a)/n$

Of course it is called closed because the endpoints of the closed interval are included as nodes for evaluation.

Theorem 4.2

Suppose that $\sum_{i=0}^{n} a_i f(x_i)$ denotes the $(n+1)$-point closed Newton-Cotes formula with $x_0 = a$, $x_n = b$, and $h = (b-a)/n$. There exists $\xi \in (a, b)$ for which,

i) if n is even and $f \in C^{n+2}[a, b]$:

$$\int_a^b f(x)dx = \sum_{i=0}^{n} a_i f(x_i) + \frac{h^{n+3} f^{(n+2)}(\xi)}{(n+2)!} \int_0^n t^2(t-1)\dots(t-n)dt.$$

ii) if n is odd and $f \in C^{n+1}[a, b]$:

$$\int_a^b f(x)dx = \sum_{i=0}^{n} a_i f(x_i) + \frac{h^{n+2} f^{(n+1)}(\xi)}{(n+1)!} \int_0^n t(t-1)\dots(t-n)dt.$$

Both the Trapezoidal rule and Simpson's rule belong to the closed Newton-Cotes class. Two other common ones are,

Simpson's Three-Eigths rule

$$\int_{x_0}^{x_3} f(x)dx = \frac{3h}{8}\left[f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)\right] - \frac{3h^5}{80} f^{(4)}(\xi),$$

where $x_0 < \xi < x_3$.

n=4

$$\int_{x_0}^{x_4} f(x)dx = \frac{2h}{45}\left[7f(x_0) + 32f(x_1) + 12f(x_2) + 32f(x_3)7f(x_4)\right] - \frac{8h^7}{945}f^{(6)}(\xi),$$

where $x_0 < \xi < x_4$.

Open Newton-Cotes Formulas

These formulas don't include the endpoints of the interval as nodes. They use the following nodes,

- $x_0 = a + h$
- $x_n = b - h$
- $x_i = x_0 + ih$ for $i = 0, 1, ...n$
- $h = \frac{(b-a)}{(n+2)}$

Since the endpoints aren't included in the set of points for evaluation, we can notate them as $x_{n-1} = a$ and $x_{n+1} = b$ indicating that all the evalution points are within the open interval (a, b).

Some common open Newton-Cotes formulas with their error terms are as follows:

n=0: Midpoint Rule

$$\int_{x-1}^{x_1} f(x)dx = 2hf(x_0) + \frac{h^3}{3}f''(\xi),$$

where $x_{-1} < \xi < x_1$.

n=1:

$$\int_{x-1}^{x_2} f(x)dx = \frac{3h}{2}[f(x_0) + f(x_1)] + \frac{3h^3}{4}f''(\xi),$$

where $x_{-1} < \xi < x_2$.

n=2:

$$\int_{x-1}^{x_3} f(x)dx = \frac{4h}{3}[2f(x_0) - f(x_1) + 2f(x_2)] + \frac{14h^5}{45}f^{(4)}(\xi),$$

where $x_{-1} < \xi < x_3$.

n=3:

$$\int_{x-1}^{x_4} f(x)dx = \frac{5h}{24}[11f(x_0) + f(x_1) + f(x_2) + 11f(x_3)] + \frac{95h^5}{144}f^{(4)}(\xi),$$

where $x_{-1} < \xi < x_4$.