MinWook Kang

January 12, 2023

1 Implement Your Own fixed quad in [-1, 1]

우리는 앞 부분에서, 해를 추정하는 Root Finding Method를 공부했다. 그리고 이번 챕터에서는 Rimann Sum, Trapezoid Rule, Simpsion's Rule 그리고 Gaussian Quadrature을 통하여 어떤 함수 f 가 주어졌을 때, 우리가 원하는 범위 안에서의 적분 값을 실제의 적분 값과 매우 비슷하게 추정할 수 있는 방법을 배웠다. 이것으로 실제로 적분이 필요한 문제들에 대해 접근해 보겠다.

1.1 Problem Recognition

SciPy 모듈에서 fixed quad를 사용하면, Gaussian Quadrature 의 방식으로 원하는 적분 범위 안에서의 적분 값을 유추할 수 있다. 하지만, 이 모듈을 사용하지 않고, 모듈을 만들어서 문제에 적용하는 것을 보일 것이다.

$$\int_{-1}^{1} e^x dx = e - e^{-1} \approx 2.3504023872876029138 \tag{1}$$

$F(x) = e^x$ 를 [-1, 1]로 적분하게 되면 나오는 값을 구하는 모듈을 만들어야 한다. Gauss-Legendre 방식에 의하면, [-1, 1]에서 N에 대한 x_i 값과 w_i 값을 알고 이것을 곱하여 더하게 되면,

$$\int_{-1}^{1} f(x)dx \approx \sum_{i=1}^{n} w_i f(x_i) \equiv G_n(f) \tag{2}$$

위의 공식에 따라, [-1, 1] 사이의 적분 값을 추산할 수 있다. 따라서 우리는 먼저 x_i 값과 w_i 값을 구해줄 것이다. 그러기 위해서는 우선, 임의의 숫자 N에 대한 르장드르 다항식의 해가 필요한데, 이는 SciPy 모듈에서 roots_legendre 을 통해 쉽게 구할 수 있다. 먼저 roots_legendre를 보면, 르장드르 다항식에 들어가는 임의의 숫자 N에 대하여 해 roots 와 무게 weights 값을 알려주는 모듈이다. 만약 N 의 값에 3을 대입하면 아래처럼 구할 수 있다.

```
from scipy.special import roots_legendre, eval_legendre
roots, weights = roots_legendre(3)
roots, weights

(array([-0.77459667, 0.        , 0.77459667]),
 array([0.55555556, 0.88888889, 0.55555556]))
```

1.2 Development of a solution

def 함수를 이용하여 min_fixed_only_quad(f, N) 함수를 정의하고, roots 와 weights인수를 xi 와 wi 로 저장한 후, zip 함수를 통하여 각 열마다 묶어준 다음, 각각의 (wi, xi) 의 묶음에 따라 식 $w * (f(x))$ 계산한다. 그후 묶음에 대한 이 식들을 더해줌으로서 위의 (2)를 파이썬으로 구현할 것이다. 먼저 $F(x) = e^x$ 이고, 적분 범위는 [-1, 1]이라고 하면 다음과 같다.

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

def min_fixed_only_quad(f, N):
    roots, weights = roots_legendre(N)
    xi = roots
    wi = weights
    return sum(w*(f(x)) for w, x in zip(wi, xi))
print(f"""
n = 1: I  = {min_fixed_only_quad(f, 1)}, Abs error = {(f(1) - f(-1)) - min_fixed_only_quad(f, 1)}
n = 2: I = {min_fixed_only_quad(f, 2)}, Abs error = {(f(1) - f(-1)) - min_fixed_only_quad(f, 2)}
n = 3: I = {min_fixed_only_quad(f, 3)}, Abs error = {(f(1) - f(-1)) - min_fixed_only_quad(f, 3)}
n = 4: I = {min_fixed_only_quad(f, 4)}, Abs error = {(f(1) - f(-1)) - min_fixed_only_quad(f, 4)}
n = 5: I = {min_fixed_only_quad(f, 5)}, Abs error = {(f(1) - f(-1)) - min_fixed_only_quad(f, 5)}
""")

n = 1: I = 2.0, Abs error = 0.35040238728760276
n = 2: I = 2.3426960879097307, Abs error = 0.007706299377872039
n = 3: I = 2.3503369286800115, Abs error = 6.545860759121 78e-05
n = 4: I = 2.3504020921563766, Abs error = 2.9513122612456755e-07
n = 5: I = 2.350402386462826, Abs error = 8.247766913882515e-10
```

1.3 Execution and Assessment

이때, 우리는 실제 값과의 차이를 비교하기 위하여 $f(1) - f(-1)$ 값에다가 추산한 값을 빼서, N 에 따라서 얼만큼 오차가 날 수 있는지 확인해 볼 것이다. 먼저 SciPy package의 fixed_quad로 계산한

결과는 다음과 같다.

```
n = 1: approx = (2.0, None)
n = 2: approx = (2.3426960879097307, None)
n = 3: approx = (2.3503369286800115, None)
n = 4: approx = (2.350402092156377, None)
n = 5: approx = (2.350402386462826, None)
```

이 두 값을 빼보면, N = 4 인 경우 $-4.440892098500626e-16$ 를 제외하고는 0으로 맞아 떨어지는 것을 볼 수 있다. N=4 인 경우에도, 수가 매우 작으므로, 0이라 봐도 무방하다. 하지만 위의 함수는 범위가 [-1, 1] 로 정해져 있기 때문에, 이를 변수변환을 통해 어느 구간에서도 적분 값을 추산할 수 있는 모듈을 만들 것이다.

2 Own fixed quad in any range

2.1 Problem Recognition

위의 상황에서는 범위가 [-1, 1]을 만족하는 상황에서만 적분 값을 구할 수 있었다. 그럼 만약, 적분 범위가 아래와 같은 식일 경우는 어떻게 구할 수 있을까? 이는 적분의 범위를 변수변환을 통하여 바꾸어 주면 된다.

$$\int_0^1 \frac{4}{1+x^2}dx \tag{3}$$

먼저 Gauss' Method에 따르면, x 와 dx는 아래처럼 바꾸어도 무방하다.

$$t = \frac{b-a}{2}x + \frac{b+a}{2} \quad , \quad dt = \frac{b-a}{2}dx \tag{4}$$

2.2 Development of a solution

변수 변환을 통해 적분 범위는 dt에 대해서 [-1, 1]을 만족하게 된다. 그러면 위의 함수를 조금 변형하면 똑같이 적분을 시킬 수 있다. 위의 def 함수에서, x를 t에 대한 식으로 바꾸어지고, 함수에 $(b-a)/2$를 곱하게 해주면 되므로, 위의 식을 $F(x) = \frac{4}{1+x^2}$, 적분 범위는 [0, 1]로 잡고 계산하면 다음과 같다.

```
def f(t):
    return np.exp(t)
```

```python

def min_fixed_any_range(f, a, b, n):
    g = lambda x: (b - a)/2 * f((b-a)/2*x + (a+b)/2)
    roots, weights = roots_legendre(n)
    xi = roots
    wi = weights
    return sum(w*(g(x)) for w, x in zip(wi, xi))
```

먼저 def로 선언한 함수를 보면 인수를 4개로 두었는데, 이는 구하고자 하는 함수의 범위를 나타내며, 변수 변환을 통해 원하는 [a, b] 의 적분 값을 구할 수 있다. 또한 $g(x)$ 라는 함수에 대해 lambda를 통하여 아래의 식을 새롭게 정의 하였다.

$$g(x) = \frac{b-a}{2} f(\frac{b-a}{2}x + \frac{b+a}{2}) \tag{5}$$

이는 $f(t)$에서 $t = \frac{b-a}{2}x + \frac{b+a}{2}$ 을 대입한 것이다. 또한, $\frac{b+a}{2}$ 는 적분 범위를 벗어날 수 있기 때문에, 적분을 나중에 해주고, $\frac{b+a}{2}$ 를 곱해줌으로써, $g(x)$에 대하여 위의 코드가 성립됨을 볼 수 있다. 그리고 section 1에서 본 것과 같이, zip(wi, xi) 에 대하여 각각의 w, x값을 할당하여 $g(x)$와 w를 곱한 것을 더해주게 되면, 우리가 구하고자 한 적분 값을 구할 수 있게 된다. 위의 코드로 인한 결과를 보면 다음과 같다.

2.3 Execution and Assessment

```python
print(f"""
n = 1: I = {min_fixed_any_range(f, -1, 1, 1)}, Abs error = {(f(1) - f(-1)) -
    min_fixed_any_range(f, -1, 1, 1)}
n = 2: I = {min_fixed_any_range(f, -1, 1, 2)}, Abs error = {(f(1) - f(-1)) -
    min_fixed_any_range(f, -1, 1, 2)}
n = 3: I = {min_fixed_any_range(f, -1, 1, 3)}, Abs error = {(f(1) - f(-1)) -
    min_fixed_any_range(f, -1, 1, 3)}
n = 4: I = {min_fixed_any_range(f, -1, 1, 4)}, Abs error = {(f(1) - f(-1)) -
    min_fixed_any_range(f, -1, 1, 4)}
n = 5: I = {min_fixed_any_range(f, -1, 1, 5)}, Abs error = {(f(1) - f(-1)) -
    min_fixed_any_range(f, -1, 1, 5)}
""")

n = 1: I = 2.0, Abs error = 0.35040238728760276
```

```
n = 2: I = 2.3426960879097307, Abs error = 0.007706299377872039
n = 3: I = 2.3503369286800115, Abs error = 6.54586075912178e-05
n = 4: I = 2.3504020921563766, Abs error = 2.9513122612456755e-07
n = 5: I = 2.350402386462826, Abs error = 8.247766913882515e-10
```

실제의 값 Abs erorr에 우리가 만든 모듈의 차를 구했을때, N 의 값, 즉 legendre 다항식의 차수가 충분하지 않았을 때, error의 폭이 커짐을 볼 수 있고, N이 커질 수록 0에 수렵하는 것을 볼 수 있다. 이와 관련하여 오차를 줄이는 방법에서도 논하여 보겠다.

3 reduce the error

3.1 Problem Recognition

$$\int_0^1 \frac{4}{1+x^2}dx \tag{6}$$

위의 함수에서, N의 값을 임의로 지정하지 않고, 원하는 오차 범위 내에서 적분을 근사적으로 구하려면 어떻게 해야 할까? 우리는 앞서서 Root Finding에서도 똑같이 이용한 방식을 이용할 수 있다. 바로 위에서 구한, 실제값에다가 결과값을 뺀 오차가 우리가 설정한 범위보다 작아질 때 까지 N을 1씩 증가시키면서 반복시키면 된다.

3.2 Development of a solution