Numerical Integration

September 24, 2018

1 Numerical Integration

In practice, we often encounter functions whose antiderivative has no closed-form formula. In such cases, analytically computing the value of a definite integral is not possible and we need to resort to numerical methods.

A natural idea is to partition the integration interval into small subintervals, and on each subinterval approximate the original function with a simpler type of function whose antiderivative is known. We can then integrate these proxy functions to approximate the true integral value. Here we present three methods in this spirit, differing only in their choice of the proxy functions: the Midpoint rule uses constant functions, the Trapezoidal rule uses linear functions, and the Simpson's rule uses quadratic functions.

As partition gets finer, under some smoothness conditions, all three methods converge to the exact integral value. More specifically, the Midpoint and Trapezoidal rules are quadratically convergent, while the simpson's rule is fourth order convergent, albeit requiring more smoothness.

In the following, we consider an integrable function $f : [a, b] \to \mathbb{R}$ and the integral

$$I = \int_a^b f(x)dx$$

We partition the interval $[a, b]$ into n intervals of equal size $h = \frac{b-a}{h}$ by using the nodes $a_i = a + ih$, for $i = 0 : n$, i.e.,

$$a = a_0 < a_1 < a_2 < \cdots < a_{n-1} < a_n = b.$$

Let x_i be the midpoint of the interval $[a_{i-1}, a_i]$, i.e.,

$$x_i = \frac{a_{i-1} + a_i}{2}, \ \forall i = 1 : n.$$

1.1 Midpoint Rule

The Midpoint rule approximates $f(x)$ on the interval $[a_{i-1}, a_i]$ by the constant function $c_i(x)$ equal to the value of the function f at the midpoint x_i of the interval $[a_{i-1}, a_i]$, i.e.,

$$c_i(x) = f(x_i), \ \forall x \in [a_{i-1}, a_i].$$

The Midpoint approximation is given by

$$I_n^M = h \sum_{i=1}^{n} f(x_i).$$

The following code provides an implementation of the rule.

```
In [2]: def I_midpoint(f, a, b, n):
            """Approximate the value of a definite integral with Midpoint rule.

            Args:
                f: A routine evaluating the function to be integrated.
                a: Left end point of the integration interval.
                n: Number of partition intervals.

            Returns:
                The approximation value of the integral under Midpoint rule.
            """

            h = (b - a) / n
            result = 0
            for i in range(1, n+1):
                result += f(a + (i - 1/2)*h)
            result =  h * result
            return result
```

1.2 Trapezoidal Rule

The Trapezoidal rule approximates $f(x)$ on the interval $[a_{i-1}, a_i]$ by the linear function $l_i(x)$ equal to $f(x)$ at the end points a_{i-1} and a_i, i.e.,

$$l_i(a_{i-1}) = f(a_{i-1}) \quad \text{and} \quad l_i(a_i) = f(a_i)$$

.

By linear interpolation, we have

$$l_i(x) = \frac{x - a_{i-1}}{a_i - a_{i-1}} f(a_i) + \frac{a_i - x}{a_i - a_{i-1}} f(a_{i-1}), \; \forall x \in [a_{i-1}, a_i].$$

The Trapezoidal approximation is given by

$$I_n^T = \frac{h}{2}\left(f(a_0) + 2\sum_{i=1}^{n-1} f(a_i) + f(a_n) \right)$$

.

The following code provides an implementation of the rule.

```
In [3]: def I_trap(f, a, b, n):
            """Approximate the value of a definite integral with Trapezoidal rule."""
            h = (b - a) / n
            result = f(a)/2 + f(b)/2
            for i in range(1, n):
                result += f(a + i*h)
            result = h * result
            return result
```

1.3 Simpson's rule

The Simpson's rule approximates $f(x)$ on the interval $[a_{i-1}, a_i]$ by the quadratic function $q_i(x)$ equal to $f(x)$ at a_{i-1}, a_i, and at the midpoint $x_i = \frac{a_{i-1}+a_i}{2}$, i.e.,

$$q_i(a_{i-1}) = f(a_{i-1}); \quad q_i(x_i) = f(x_i) \quad \textit{and} \quad q_i(a_i) = f(a_i).$$

By quadratic interpolation, we have

$$q_i(x) = \frac{(x-a_{i-1})(x-x_i)}{(a_i-a_{i-1})(a_i-x_i)} f(a_i) + \frac{(a_i-x)(x-a_{i-1})}{(a_i-x_i)(x_i-a_{i-1})} f(x_i) + \frac{(a_i-x)(x_i-x)}{(a_i-a_{i-1})(x_i-a_{i-1})} f(a_{i-1}), \; \forall x \in [a_{i-1}, a_i].$$

The Simpson's approximation is given by

$$I_n^S = \frac{h}{6}\left(f(a_0) + 2\sum_{i=1}^{n-1} f(a_i) + f(a_n) + 4\sum_{i=1}^{n} f(x_i) \right).$$

The following code provides an implementation of the rule.

```
In [4]: def I_simpson(f, a, b, n):
            """Approximate the value of a definite integral with Simpson's rule."""
            h = (b - a) / n
            result = f(a) / 6 + f(b) / 6
            for i in range(1, n):
                result += f(a + i*h) / 3
            for i in range(1, n+1):
                result += 2 * f(a + (i - 1/2)*h) /3
            result = h * result
            return result
```

1.4 Approximation with stopping criterion

In ractice, we usually approximate an integral given a certain tolerance. The above-mentioned methods could not be of use directly because we do not know the number of partitions needed in advance. Instead, we repeatedly compute approximations with increasing number of partitions until two consecutive results are close enough with each other, at which point we stop the algorithm and return the last result as our final approximation.

The following code implements this algorithm where we double the number of partions at each iteration.

```
In [17]: def approx_integral(f, a, b, method, tol):
             """Approximate the value of a definite integral with required accurary.

             Args:
                 f: A routine evaluating the function to be integrated.
                 a: Left end point of the integration interval.
                 n: Number of partition intervals.
                 method: The partitioning-based approximation method used, with f, a,
                     b, and no. of partitions as inputs.
```

```
        tol: Tolerance for deviation from the exact integral value

    Returns:
        An approximate value within tolerance of the exact value.
    """

    n = 4  # initialize no. of intervals
    I_old = I_new = method(f, a, b, n)  # previous approximation
    n = 2 * n
    I_new = I_new = method(f, a, b, n)  # current approximation
    while (abs(I_new - I_old) > tol):  # check stopping criterion
        I_old = I_new
        n = 2 * n
        I_new = method(f, a, b, n)
    return I_new
```

1.5 Test

We test the three numerical integration methods as well as approximation with stopping criterion on the integral

$$I = \int_1^3 \frac{1}{(x+1)^2} dx$$

for $n = 8$ partition intervals. Note that the exact value of this integral is $\frac{1}{4}$.

```
In [38]: def f_test(x):
             return 1 / pow(x + 1, 2)

         print('Exact:', 0.25)
         print('Midpoint:', I_midpoint(f_test, 1, 3, 8))
         print('Trapezoidal:', I_trap(f_test, 1, 3, 8))
         print('Simpson\'s:', I_simpson(f_test, 1, 3, 8))
         print('Midpoint (tol=0.0001):', approx_integral(f_test, 1, 3, I_midpoint, 0.0001))
```

```
Exact: 0.25
Midpoint: 0.24943374496382814
Trapezoidal: 0.2511354251631682
Simpson's: 0.2500009716969415
Midpoint (tol=0.0001): 0.24999109988161783
```