Simpson's Rule

Simpson's Rule is a numerical integration method used to approximate the definite integral of a function over a given interval. It provides a more accurate estimation than the trapezoidal rule by approximating the area under a curve using quadratic (second-order) polynomials.

Simpson's Rule approximates the definite integral of a function $f(x)$ over an interval $[a, b]$ by breaking the interval into n equally spaced sub-intervals. The formula is as follows:

$$\int_a^b f(x)\,dx \approx \frac{h}{3}[f(a) + 4f(a+h) + 2f(a+2h) + 4f(a+3h) + \cdots + 2f(a+(n-2)h) + 4f(a+(n-1)h) + f(b)]$$

Where:

- a and b are the lower and upper limits of integration.
- n is the number of equally spaced subdivisions.
- h is the width of each subdivision, calculated as $h = \frac{a-b}{n}$
- $f(x)$ is the function being integrated.

In essence, Simpson's Rule uses a weighted average of function values over the sub-intervals to estimate the area under the curve and, by extension, the definite integral. This method is accurate, especially when the function can be well-approximated by quadratic curves. It finds applications in various fields, such as physics and engineering, for precise numerical integration.

SageMath is a robust mathematical software system capable of conducting numerical computations, which includes the implementation of Simpson's Rule to estimate definite integrals. Below is a demonstration of how to apply Simpson's Rule in SageMath.

```
def simpsons_rule(func, a, b, n):

      if n % 2 != 0:
       raise ValueError("n must be an even number for Simpson's Rule.")

    h = (b - a) / n
    integral = func(a) + func(b)

    for i in range(1, n):
        x = a + i * h
        if i % 2 == 0:
            integral += 2 * func(x)
        else:
            integral += 4 * func(x)

    integral *= h / 3
    return integral

def f(x):
    return f(x)

a = # lower limit
b = # upper limit
n = # Number of subintervals,n must be an even number.

result = simpsons_rule(f, a, b, n)
#To approxiamte to 4 decimal places
Approx_result = round(result, 4)
print(Approx_result)
```

Figure 1: SageMath code to evaluate integrals using Simpson's rule

This SageMath code applies the Simpson's Rule to a specific function and interval to evaluate its definite integral. Here's the summary of the code:

- **def simpsons_rule(func, a, b, n)**: This line defines a function called simpsons_rule that takes four arguments: **func (the function to be integrated), a (the lower limit of the integration interval), b (the upper limit of the integration interval), and n (the number of subintervals).**
- **if** $n\%2! = 0$: This line checks if n is not an even number, because Simpson's Rule requires an even number of sub-intervals. It raises a **ValueError** with a message indicating that n must be even.
  h = (b - a) / n - Calculates the width of each sub-interval h by dividing the difference between b and a by the number of sub-intervals n.
- **integral** $= func(a) + func(b)$: Initialize the integral variable with the values of the function at the lower and upper limits.
  - **for i in range(1, n)**: Start a loop that iterates from 1 to n - 1. This loop will calculate the sum of the areas under the curve using Simpson's Rule.
  - **x = a + i * h** - Calculates the value of x at the current sub-interval, where i is the loop index.
  - **if i % 2 == 0**: Check if the index i is even. If it is, add 2 * func(x) to the integral variable.
    else:
    If the index i is odd, add 4 * func(x) to the integral variable.
  - **integral *= h / 3**: After the loop completes, multiply the integral by h / 3, which is part of the Simpson's Rule formula.
  - **return integral**: Return the final approximate integral value.
- **def f(x)**: Define a function f(x) which is the function to be integrated.
- **result = simpsons_rule(f, a, b, n)**: Call the simpsons_rule function with the provided function f, lower limit a, upper limit b, and number of sub-intervals n. This calculates the approximate integral.
- **Approx_result = round(result, 4)**: Round the result to four decimal places and assign it to the Approx_result variable.
- **print(Approx_result)**: Print the approximate integral value to the console.

Solving Examples with the Code

Example 1

Use the Simpson's rule to approximate $\int_1^{1.2} \frac{1}{x}\,dx$.

Example 2

Use the Simpson's rule to approximate $\int_0^4 e^x\,dx$.

Example 3

Use the Simpson's rule to approximate $\int_0^2 \frac{2}{\sqrt{x}}\,dx$.

Example 4

Use the Simpson's rule to approximate $\int_0^5 \sqrt{25 - x^2}\,dx$, using 10 intervals.

Example 5

With 4 intervals, use the Simpson's rule to evaluate the integral I $= \int_0^1 \frac{dx}{\sqrt{x^2 + 6x + 10}}$.

```sage
def simpsons_rule(func, a, b, n):

    if n % 2 != 0:
        raise ValueError("n must be an even number for Simpson's Rule.")

    h = (b - a) / n
    integral = func(a) + func(b)

    for i in range(1, n):
        x = a + i * h
        if i % 2 == 0:
            integral += 2 * func(x)
        else:
            integral += 4 * func(x)

    integral *= h / 3
    return integral

def f(x):
    return 1/x

a = 1
b = 1.2
n = 2

result = simpsons_rule(f, a, b, n)
Approx_result = round(result, 4)
print(Approx_result)
```

Evaluate

Language: Sage

Share

```
0.1823
```

Help | Powered by SageMath

Figure 2: Solution to example 1.

```sage
def simpsons_rule(func, a, b, n):

    if n % 2 != 0:
        raise ValueError("n must be an even number for Simpson's Rule.")

    h = (b - a) / n
    integral = func(a) + func(b)

    for i in range(1, n):
        x = a + i * h
        if i % 2 == 0:
            integral += 2 * func(x)
        else:
            integral += 4 * func(x)

    integral *= h / 3
    return integral

def f(x):
    return exp(x)

a = 0
b = 4
n = 2

result = simpsons_rule(f, a, b, n)
Approx_result = round(result, 4)
print(Approx_result)
```

Evaluate

Language: Sage

Share

```
56.7696
```

Help | Powered by SageMath

Figure 3: Solution to example 2.

```sage
def simpsons_rule(func, a, b, n):

    if n % 2 != 0:
        raise ValueError("n must be an even number for Simpson's Rule.")

    h = (b - a) / n
    integral = func(a) + func(b)

    for i in range(1, n):
        x = a + i * h
        if i % 2 == 0:
            integral += 2 * func(x)
        else:
            integral += 4 * func(x)

    integral *= h / 3
    return integral

def f(x):
    return 2/ sqrt(x)

a = 1
b = 2
n = 2

result = simpsons_rule(f, a, b, n)
Approx_result = round(result, 4)
print(Approx_result)
```

Evaluate

Language: Sage

Share

```
1.6577
```

Help | Powered by SageMath

Figure 4: Solution to example 3.

```sage
def simpsons_rule(func, a, b, n):

    if n % 2 != 0:
        raise ValueError("n must be an even number for Simpson's Rule.")

    h = (b - a) / n
    integral = func(a) + func(b)

    for i in range(1, n):
        x = a + i * h
        if i % 2 == 0:
            integral += 2 * func(x)
        else:
            integral += 4 * func(x)

    integral *= h / 3
    return integral

def f(x):
    return sqrt(25 - x^2)

a = 0
b = 5
n = 10

result = simpsons_rule(f, a, b, n)
Approx_result = round(result, 4)
print(Approx_result)
```

Evaluate

Language: Sage

Share

```
19.5438
```

Help | Powered by SageMath

Figure 5: Solution to example 4.

```
def simpsons_rule(func, a, b, n):

    if n % 2 != 0:
        raise ValueError("n must be an even number for Simpson's Rule.")

    h = (b - a) / n
    integral = func(a) + func(b)

    for i in range(1, n):
        x = a + i * h
        if i % 2 == 0:
            integral += 2 * func(x)
        else:
            integral += 4 * func(x)

    integral *= h / 3
    return integral

def f(x):
    return 1/sqrt(x^2 + 6*x + 10)

a = 0
b = 1
n = 4

result = simpsons_rule(f, a, b, n)
Approx_result = round(result, 4)
print(Approx_result)
```

Evaluate

Language: Sage

Share

```
0.2763
```

Help | Powered by SageMath

Figure 6: Solution to example 5.

Remark

In summary, SageMath's application in evaluating integrals with Simpson's Rule provides an efficient and practical solution for numerical integration. This tool balances accuracy and computational cost while offering customizable options for various mathematical challenges. In today's data-driven world, SageMath proves to be a valuable asset for students and professionals. We encourage you to explore its diverse mathematical capabilities.

Thank you for your attention.