THINK FAST

Competitive Programming

From Problem 2 Solution in O(1)

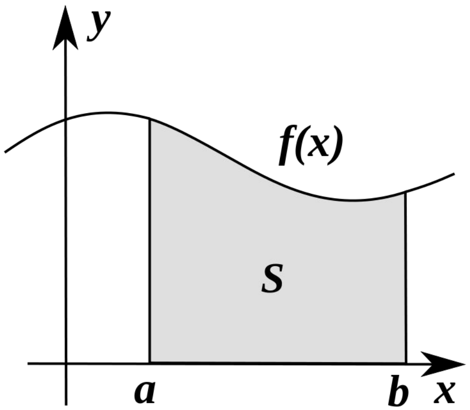
Numerical integration

Mostafa Saad Ibrahim
PhD Student @ Simon Fraser University

Numerical integration

- Family of algorithms for calculating the numerical value of a definite integral
- **Approximate** definite integral: $\int_a^b f(x)\,dx$
- E.g. evaluate 3 sin(x) in [1, 2]
- Let's focus on **1 D**

I will refer in nutshell for several ones, but one to use in competitions is last one



- **Differences**: Convergence rate and error bound

Brute force it

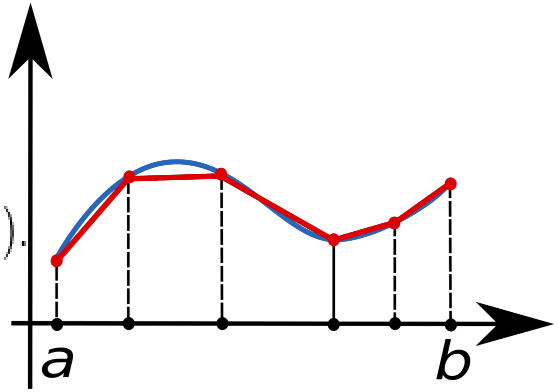
- Just evaluate it cross interval...can be slow for wide ranges

```
double f(double x) {
    return exp(-x * x);
    //return 3 * sin(x);
}

double bruteForceIntegration(double x1, double x2) {
    double area = 0;
    double w = (x2 - x1) / 5000000; // width

    for (double x = x1 + w / 2; x <= x2 - w / 2; x += w)
        area += w * f(x);
    return area;
}
```

Trapezoidal rule

$$\int_a^b f(x)\,dx \approx \frac{h}{2}\sum_{k=1}^{N}\left(f(x_{k+1})+f(x_k)\right)$$

$$= \frac{b-a}{2N}\left(f(x_1)+2f(x_2)+2f(x_3)+2f(x_4)+\cdots+2f(x_N)+f(x_{N+1})\right).$$




```
double trapezoidalRuleIntegration(double a, double b, int n = 5000000) {
    double area = 0;    // Uniform grid

    for (int k = 0; k <= n; k++) {
        double x = a + k * (b - a) / n;
        if (k == 0 || k == n)
            area += f(x);
        else
            area += 2 * f(x);
    }
    return area * (b - a) / (2 * n);
}
```

Simpson’s rule

- Given an interval [a, b] and, Simpson’s rule **approximates** the integral of f(x) in this range
- $\int_a^b f(x)\,dx \approx \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right].$
- Given n, **even** number, we can split interval and use **composite** Simpson's rule:

$$\int_a^b f(x)\,dx \approx \frac{h}{3}\left[f(x_0) + 2\sum_{j=1}^{n/2-1} f(x_{2j}) + 4\sum_{j=1}^{n/2} f(x_{2j-1}) + f(x_n)\right],$$

Simpson's rule

```
double compositeSimpsonsIntegration(double a, double b, int n = 5000000) {
    double h = (b - a) / n;
    int m = 0;
    double area = 0.0;

    for (double x = a; x <= b; x += h) {
        double r = f(x);
        if (x == a || x == b)
            area += r;
        else
            m = !m, area += r * (m + 1) * 2.0;
    }
    return area * (h / 3.0);
}
```

Adaptive Simpson's method

- This method uses an **estimate** of the error we get from calculating a definite integral using Simpson's rule.
- If the error **exceeds** a user-specified **tolerance**, The algorithm calls for **subdividing** the interval of integration in two and applying adaptive Simpson's method to each subinterval in a recursive manner
- Use that in **competitions**

Adaptive Simpson's method

```
double simpsons_f(double a, double b) {
    return (f(a) + 4 * f((a + b) / 2) + f(b)) * (b - a) / 6;
}

double adaptiveSimpsonIntegration(double a, double b) {
    double m = (a + b) / 2;
    double l = simpsons_f(a, m), r = simpsons_f(m, b), all = simpsons_f(a, b);

    if (fabs(l + r - all) < 1e-12)  // 1e-15 is requested accuracy
        return all;

    return adaptiveSimpsonIntegration(a, m) + adaptiveSimpsonIntegration(m, b);
}
```

```
double s = 2, e = 10;

cout << bruteForceIntegration(s, e) << "\n";
cout << compositeSimpsonsIntegration(s, e) << "\n";
cout << adaptiveSimpsonIntegration(s, e) << "\n";
cout << trapezoidalRuleIntegration(s, e) << "\n";
```

```
0.00414553
0.00414553
0.00414553
0.00414553
```

تم بحمد الله

علمكم الله ما ينفعكم

ونفعكم بما تعلمتم

وزادكم علماً

Problems

- SGU 217, ZOJ 2675, UVA (12528, 1280), Timus 1562, SPOJ (CIRU)