4.4 節末問題 4.4 の解答

問題 4.4.1 (1)

この問題は、多項式関数を積分する方法（→**4.4.3項**）の理解を問う問題です。

$$F(x) = \frac{1}{4}x^4 + x^3 + \frac{3}{2}x^2 + x$$

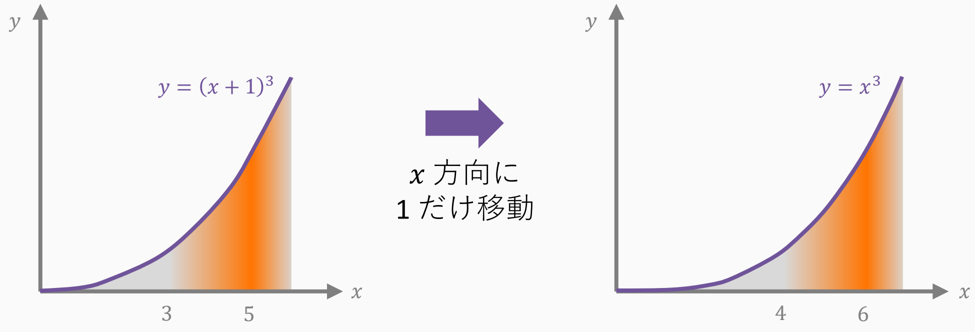
とすると、求める答えは以下のようになります。

$$\int_3^5 (x^3 + 3x^2 + 3x + 1)dx = F(5) - F(3)$$

そこで $F(5) = 323.75, F(3) = 63.75$ であるため、答えは $323.75 - 63.75 = \boxed{260}$ です。
なお、$(x^3 + 3x^2 + 3x + 1) = (x+1)^3$ であることを使うと、楽に計算できます。

$$\int_3^5 (x+1)^3\,dx = \int_4^6 x^3 dx = \frac{1}{4}(6^4 - 4^4) = 260$$

関数のグラフを右方向に 1 だけ平行移動させることを考えると、分かりやすいです。

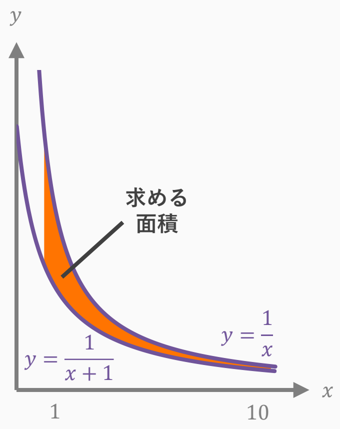


問題 4.4.1 (2)

この問題は、$1/x$ を積分する方法（→**4.4.3項**）の理解を問う問題です。
積分は符号付き面積を求める操作に対応するため、

$$F(x) = \int_1^{10} \frac{1}{x} - \frac{1}{x+1}\,dx = \int_1^{10} \frac{1}{x}dx - \int_1^{10} \frac{1}{x+1}dx$$

が成り立ちます。イメージは右図の通りです。



さて、赤色部分・青色部分をそれぞれ求めると、以下のようになります。$1/(x+1)$ の積分が分からない人は、4.4.5 項に戻って確認しましょう。

$$\int_1^{10} \frac{1}{x} dx = \log_e 10 - \log_e 1 = \log_e 10$$

$$\int_1^{10} \frac{1}{x+1} dx = \int_2^{11} \frac{1}{x} dx = \log_e 11 - \log_e 2 = \log_e 11/2$$

そこで、対数関数の公式（→**2.3.10項**）より、求める答えは

$$\log_e 10 - \log_e \frac{11}{2} = \log_e \left(10 \div \frac{11}{2}\right) = \boxed{\log_e \frac{20}{11}}$$

です。およそ 0.5978 となります。

問題 4.4.1 (3)

実は、以下の式が成り立ちます。

$$\frac{1}{x^2+x} = \frac{1}{x} - \frac{1}{x+1}$$

したがって、求める答えは (2) と同じになります。

$$\int_1^{10} \frac{1}{x^2+x} dx = \int_1^{10} \frac{1}{x} - \frac{1}{x+1} dx = \boxed{\log_e \frac{20}{11}}$$

問題 4.4.2

定積分の値は、約 1.2882263643059391197 であることが知られています。

それでは、どうやってこの値を求めるのでしょうか。多項式関数などの積分は手計算でも正確な値を計算できますが、$f(x) = 2^{x^2}$ の場合は関数 $f(x)$ が複雑であるため、厳密な答えを計算するのは非常に難しいです。

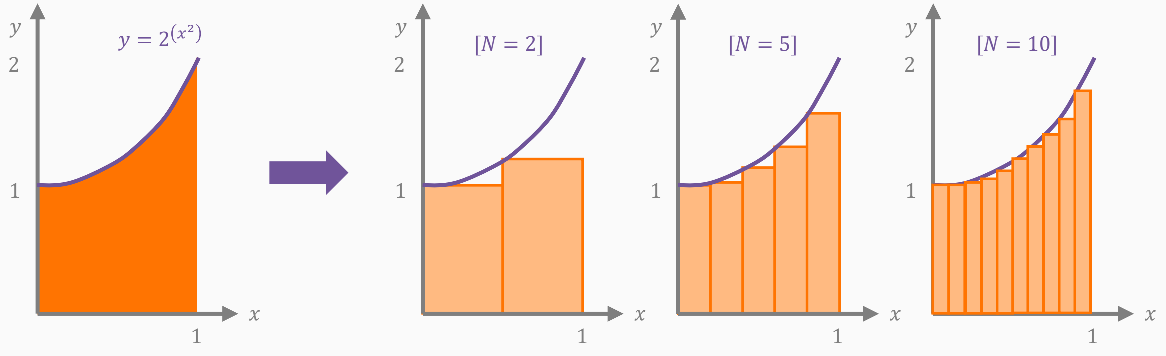
このような場合、代わりに答えの近似値を計算する数値計算（→**4.3.7項**）と呼ばれる手法がよく使われます。ここでは代表的な方法を 2 つ紹介します。

方法 1：単純な区分求積法

積分は面積を求める操作に対応するため、$f(x) = 2^{(x^2)}$ とするとき、

$$\int_0^1 f(x)dx = \frac{f(0) + f\left(\frac{1}{N}\right) + f\left(\frac{2}{N}\right) + \cdots + f\left(\frac{N-1}{N}\right)}{N}$$

と近似することができます。$N = 2, 5, 10$ の場合のイメージ図は以下の通りです。



この方法で定積分の値を求めるプログラムの例として、以下が考えられます。ここで、N の値を増やせば増やすほど近似精度が上がります。

```
#include <iostream>
#include <cmath>
using namespace std;

int main() {
	int N = 1000000;
	double Answer = 0.0;

	for (int i = 0; i < N; i++) {
		double x = 1.0 * i / N;
		double value = pow(2.0, x * x); // f(i/N) の値
		Answer += value;
	}
	printf("%.14lf¥n", Answer / N);
	return 0;
}
```

しかし、この方法では絶対誤差を 10^{-12} 以下にすることは難しいです。実際、

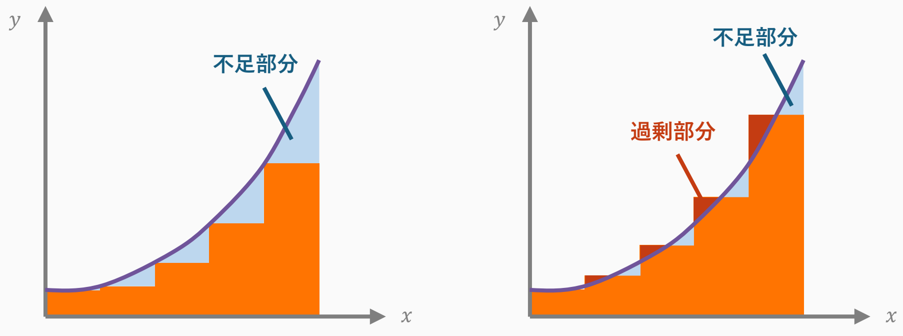
- $N = 1000$ のとき、出力は 1.28772659535497
- $N = 1000000$ のとき、出力は 1.28822586430618

となり、3〜6 桁しか一致していません。

方法 2：中央の値を使う

方法 1 では区間の左端を使いましたが、ここでは中央の値 $f(1/2N), f(3/2N), \ldots$ を使って面積を求めてみましょう。

下図は $N=5$ の場合の例を示しており、赤色は過剰な部分、青色は不足している部分を意味します。中央の値を使った場合、赤色と青色の面積がほぼ同じであり、正確に数えられているように思えます。



数式で表すと、定積分は以下のように近似することができます。

$$\int_0^1 f(x)dx = \frac{f\left(\frac{1}{2N}\right) + f\left(\frac{3}{2N}\right) + \cdots + f\left(\frac{2N-1}{2N}\right)}{N}$$

これを実装すると、以下のようになります。（Python・JAVA・C のプログラムは chap4-4.md をご覧ください）

```
#include <iostream>
#include <cmath>
using namespace std;

int main() {
    int N = 1000000;
    double Answer = 0.0;

    for (int i = 0; i < N; i++) {
        double x = 1.0 * (2 * i + 1) / (2 * N);
        double value = pow(2.0, x * x); // f(i/N) の値
        Answer += value;
    }
    printf("%.14lf¥n", Answer / N);
    return 0;
}
```

方法 1 と比べて精度がかなり良くなり、$N = 1000000$ のとき絶対誤差 10^{-12} を実現することができます。

- $N = 1000$ のとき、出力は 1.28822624878143
- $N = 1000000$ のとき、出力は 1.28822636430577

なお、さらに効率的に定積分の近似値を求める方法として、シンプソンの公式などが知られています。興味のある方は、インターネットなどで調べてみてください。

問題 4.4.3

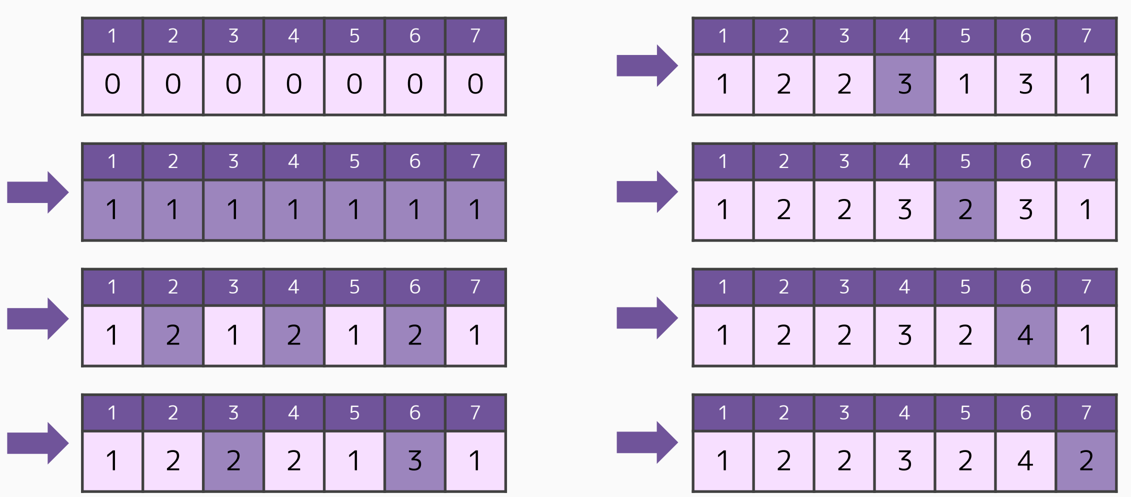
単純な解法として、以下のものが考えられます。

> $i = 1, 2, \ldots, N$ の順に、約数をすべて列挙することによって $f(i)$ を計算する。そうすると答えが分かる。約数列挙の計算量は $O(\sqrt{N})$ であるため、処理全体の計算量は $O(N^{1.5})$ である。

しかし、本問題の制約では実行時間制限超過（TLE）となってしまいます。より高速に $f(1), f(2), \ldots, f(N)$ を計算する方法として、以下が考えられます。

> 1. 最初、すべての i $(1 \leq i \leq N)$ について $f(i) = 0$ とする。
> 2. 1 の倍数：$f(1), f(2), f(3), f(4), \ldots$ に 1 を加算する。
> 3. 2 の倍数：$f(2), f(4), f(6), f(8), \ldots$ に 1 を加算する。
> 4. 3 の倍数：$f(3), f(6), f(9), f(12), \ldots$ に 1 を加算する。
> 5. $4, 5, 6, 7, \ldots, N$ の倍数についても同じような操作をする。

$N = 7$ の場合における、$f(1), f(2), \ldots, f(N)$ を求める過程は以下のようになります。



それでは、このアルゴリズムの計算量を見積もってみましょう。x の倍数は全部で N/x 個あるため、x の倍数すべてに 1 を足す操作には計算量 $O(N/x)$ かかります。したがって、全体の計算回数は、

$$\frac{N}{1}+\frac{N}{2}+\cdots+\frac{N}{N}=O(N\log N)$$

となり、$N=10^7$ の場合でも十分高速に動作します（Python などの低速なプログラミング言語では TLE するかもしれません）。実装例は以下の通りです。

```
#include <iostream>
using namespace std;

long long N;
long long F[10000009];
long long Answer = 0;

int main() {
    // 入力 → 配列の初期化
    cin >> N;
    for (int i = 1; i <= N; i++) F[i] = 0;

    // F[1], F[2], ..., F[N] を計算する
    for (int i = 1; i <= N; i++) {
        // F[i], F[2i], F[3i], ... に 1 を加算
        for (int j = i; j <= N; j += i) F[j] += 1;
    }

    // 答えを求める → 出力
    for (int i = 1; i <= N; i++) {
        Answer += 1LL * i * F[i];
    }
    cout << Answer << endl;
    return 0;
}
```

なお、数学的考察編の「足された回数を考えるテクニック（→**5.7節**）」を使うと、この問題を計算量 $O(N)$ で解くことも可能です。

問題 4.4.4

答えは 6,000,022,499,693 であることが知られています。

- 出典：https://oeis.org/A004080/b004080.txt

（解説は次ページへ続きます）

単純な方法として、以下のプログラムのように $1/1+1/2+1/3+\cdots$ を順番に足していく方法が考えられます。しかし、現実的な時間では $N=23$ 程度までしか実行が終わりません。

```
#include <iostream>
using namespace std;

int main() {
    // パラメータの設定・初期化
    long long cnt = 0;
    double LIMIT = 23; // これを 30 にすれば答えが求められる
    double Current = 0;

    // 1 つずつ足していく
    while (Current < LIMIT) {
        cnt += 1;
        Current += 1.0 / cnt;
    }

    // 答えを出力
    cout << cnt << endl;
    return 0;
}
```

そこで高速に求める代表的な方法として、以下の 2 つが挙げられます。他にも様々な方法がありますので、ぜひ考えてみてください。

方法 1：近似を使う

脚注で述べたように、オイラー定数を $\gamma=0.57721566490153286\ldots$、$1/1$ から $1/n$ までの和を H_n とすると、H_n の値は $\log_e n+\gamma$ に非常に近い値となります。そこで、$\log_e n+\gamma\geq 30$ となる最小の n は、

$$\lfloor e^{30-\gamma}\rfloor=\lfloor 6000022499693.369\ldots\rfloor=6000022499693$$

となり、何と答えと一致します。

方法 2：並列計算を使う

計算回数が 10^{12} 回を超える場合、通常は現実的な時間で計算が終わりません。しかし、CUDA などのプログラミング言語を使って並列計算をすると、計算時間が 100 倍以上短縮されます。有名な「スパコン富岳」も実は並列計算で動いています。興味のある人は、インターネットなどで調べてみてください。