4.4 節末問題 4.4 的解答

問題 4.4.1 (1)

這是測試對於將多項式函數積分的方法（→**4.4.3項**）理解的問題。。

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

令F(x)為上式，則所求答案如下：

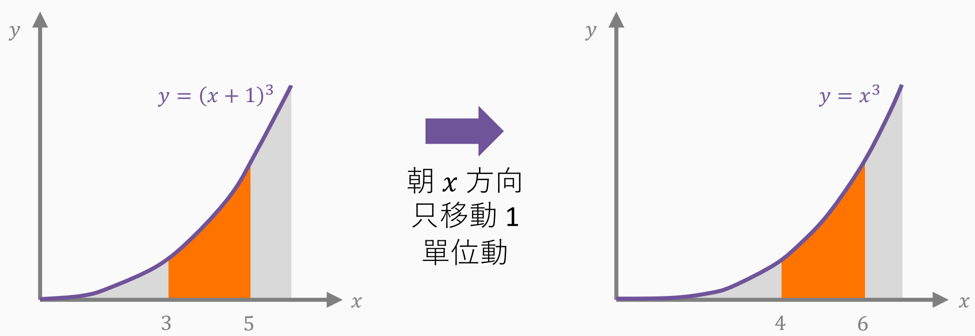
$$\int_3^5 (x^3 + 3x^2 + 3x + 1)dx = F(5) - F(3)$$

因為 $F(5) = 323.75, F(3) = 63.75$， 所以答案是 $323.75 - 63.75 = \boxed{260}$

另外，使用 $(x^3 + 3x^2 + 3x + 1) = (x+1)^3$的話可以輕鬆計算。

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

若使函數的圖形向右平行移動 1 單位，會更容易理解。



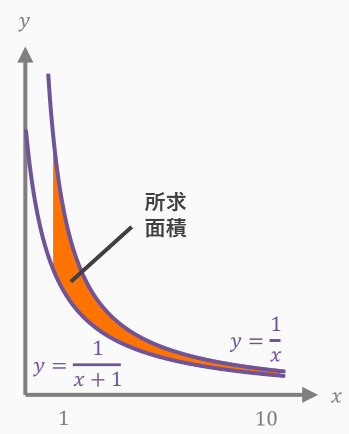
問題 4.4.1 (2)

這是測試對將$1/x$積分的方法（→**4.4.3項**）理解的問題。

由於積分對應於求附帶符號地面積的操作，因此下式成立。

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

示意圖如右圖。



分別求出紅色部分和藍色部分如下。如果不瞭解$1/(x+1)$的積分，請回到4.4.5 項確認。

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

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

因此，根據對數函數的公式（→**2.3.10項**），求得的答案如下。

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

約為 0.5978。

問題 4.4.1 (3)

實際上，下式會成立。

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

因此，所求答案與(2)相同。

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

問題 4.4.2

已知定積分的值為約 1.2882263643059391197。

那麼，如何求得這個值呢？雖然多項式函數等的積分也可以手動計算出正確的值，但在 $f(x) = 2^{x^2}$ 的情況下，由於函數 $f(x)$很複雜，縝密的計算答案非常困難。。

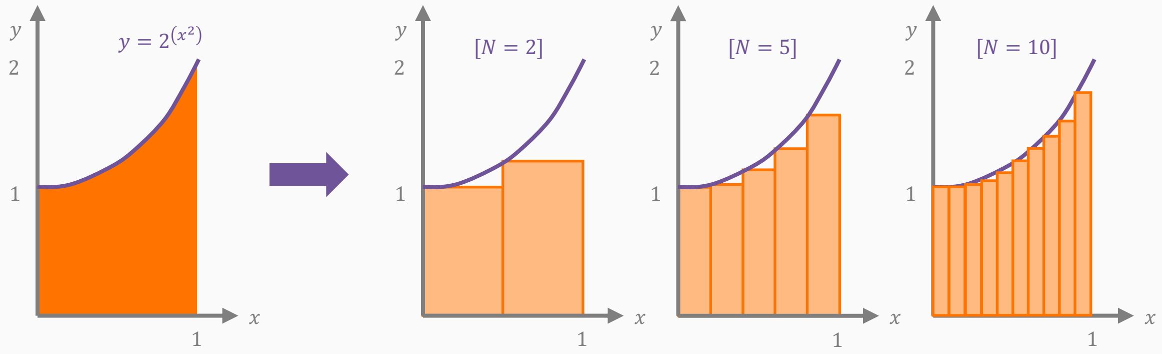
在這種時候，經常使用稱為數值計算（→**4.3.7項**）的方法，計算答案的近似值來取代。這裡介紹兩種代表性的方法。

方法 1：簡單的區間求積法

因為積分對應於求面積的操作，當$f(x) = 2^{(x^2)}$ 時，可以近似如下。

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

$N = 2, 5, 10$ 時的示意圖如下。



以這種方法求解定積分值的程式範例如下。在此，越增加N的值越可以提高近似精準度。

```
#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) 的值
		Answer += value;
	}
	printf("%.14lf\n", Answer / N);
	return 0;
}
```

然而，使用這種方法難以將絕對誤差控制在 10^{-12} 以下。實際上：

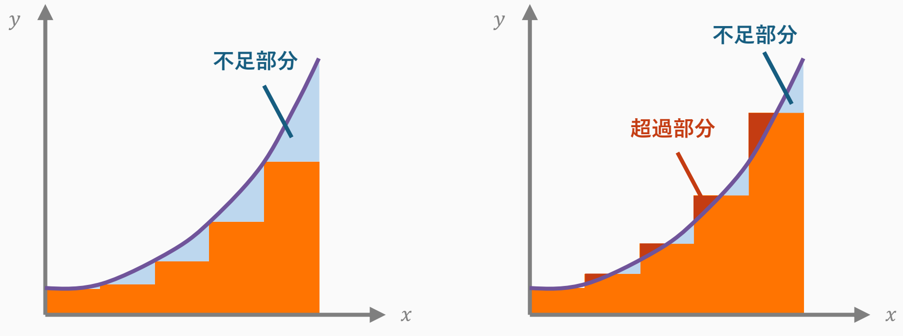
- $N = 1000$ 時，輸出為 1.28772659535497
- $N = 1000000$ 時，輸出為 1.28822586430618

僅能達到 3～6 位的一致性。

方法 2：使用中央值

方法 1 中是使用區間的左端，這裡則使用中央值 $f(1/2N), f(3/2N), \ldots$ 來求面積。

下図下圖顯示了 $N = 5$ 的例子，紅色代表超過的部分，藍色代表不足的部分。使用中央值時，紅色和藍色的面積幾乎相等，可以被正確地計數。



若以數式表示，定積分可近似如下：

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

實作後會如下所示。（Python、Java、C的程式請參閱 chap4-4.md）

```
#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((2i+1)/2N) 的值
        Answer += value;
    }
    printf("%.14lf\n", Answer / N);
    return 0;
}
```

與方法 1 相比，精準度大幅提高， $N = 1000000$ 時可實現絕對誤差10^{-12} 。

- $N = 1000$ 時，輸出為 1.28822624878143
- $N = 1000000$ 時，輸出為 1.28822636430577

此外，還有如辛普森公式等已知的更有效率求定積分近似值的方法。對此感興趣的讀者可以在網路等進行調查。

問題 4.4.3

簡單的解法如下。

> 按照 $i = 1, 2, \ldots, N$的順序，列舉出所有因數，藉此來計算$f(i)$ 。這樣就能得知答案。因為列舉因數的計算複雜度為$O(\sqrt{N})$ ，所以整體處理的計算複雜度$O(N^{1.5})$。

然而，在本問題的限制下會超過執行時間限制（TLE）。更快速地計算 $f(1), f(2), \ldots,$ $f(N)$的方法如下。

> 1. 最初，對所有 i $(1 \leq i \leq N)$ 設 $f(i) = 0$ 。
> 2. 1 的倍數：於 $f(1), f(2), f(3), f(4), \ldots$ 加 1 。
> 3. 2 的倍數：於 $f(2), f(4), f(6), f(8), \ldots$ 加 1 。
> 4. 3 的倍數：於 $f(3), f(6), f(9), f(12), \ldots$ 加 1 。
> 5. 對 $4, 5, 6, 7, \ldots, N$ 的倍數也進行同樣操作。

$N = 7$ 時，求 $f(1), f(2), \ldots, f(N)$ 的過程如下。

1	2	3	4	5	6	7
0	0	0	0	0	0	0

→

1	2	3	4	5	6	7
1	1	1	1	1	1	1

→

1	2	3	4	5	6	7
1	2	1	2	1	2	1

→

1	2	3	4	5	6	7
1	2	2	2	1	3	1

→

1	2	3	4	5	6	7
1	2	2	3	1	3	1

→

1	2	3	4	5	6	7
1	2	2	3	2	3	1

→

1	2	3	4	5	6	7
1	2	2	3	2	4	1

→

1	2	3	4	5	6	7
1	2	2	3	2	4	2

接著，來估算此演算法的計算複雜度吧。x 的倍數全部有 N/x 個，故對所有x的倍數加1的操作的計算複雜度需要 $O(N/x)$ 。因此，整體的計算次數如下。

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

即使 $N=10^7$ 也能夠高速運行（但 Python 等慢速的程式語言可能會 TLE）。實作範例如下。

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

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

int main() {
    // 輸入 → 陣列的初始化
    cin >> N;
    for (int i = 1; i <= N; i++) F[i] = 0;

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

    // 求出答案 → 輸出
    for (int i = 1; i <= N; i++) {
        Answer += 1LL * i * F[i];
    }
    cout << Answer << endl;
    return 0;
}
```

此外，利用數學考察篇中的「考慮相加次數的技巧（→**5.7節**）」可將此問題的計算複雜度降為 $O(N)$。。

問題 4.4.4

已知答案為 6,000,022,499,693 。

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

（解說在下頁繼續）

作為簡單的方法，考慮如下程式將 $1/1 + 1/2 + 1/3 + \cdots$依序相加的方法。但現實的時間上，此方法只能到$N = 23$ 的程度，否則無法完成執行。

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

int main() {
    // 參數的設定、初始化
    long long cnt = 0;
    double LIMIT = 23; // 將此設為 30 的話，答案即為所求
    double Current = 0;

    // 1 個 1 個相加
    while (Current < LIMIT) {
        cnt += 1;
        Current += 1.0 / cnt;
    }

    // 輸出答案
    cout << cnt << endl;
    return 0;
}
```

因此，舉以下兩種方法作為代表性的快速求解方法。其他還有多種方法，請思考看看。

方法 1：使用近似

如注腳所述，歐拉常數 $\gamma = 0.57721566490153286\ldots$，令從 $1/1$ 到 $1/n$ 的和為 H_n，則 H_n 值會非常接近 $\log_e n - \gamma$ 。因此，使$\log_e n - \gamma \geq 30$ 的最小的 n 為：

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

答案是一致的。

方法 2：使用並行計算

計算次數超過 10^{12} 時，通常難以在現實的時間中完成。然而，若使用CUDA 等程式語言進行並行計算，可將計算時間縮短 100 倍以上。著名的「超級電腦富岳」也是利用並行計算運行。有興趣的讀者請在網路上調查看看。