4.3 節末問題 4.3 的解答

問題 4.3.1

這是測試對多項式函數的微分（→**4.3.3項**）的理解的問題。答案如下所示。。

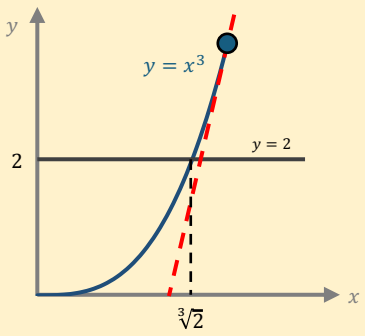
1. $f'(x) = 7$
2. $f'(x) = 2x + 4$
3. $f'(x) = 5x^4 + 4x^3 + 3x^2 + 2x + 1$

問題 4.3.2

$\sqrt[3]{2}$ 的值可以根據以下方法求得（→**4.3.6項**）。

- 令 $f(x) = x^3$ 。此時 $f'(x) = 3x^2$。
- 首先，設定一個任意的初始值a 。
- 然後，持續將 a 的值更新如下。

> 通過點 $(a, f(a))$ 之切線與直線 $y = 2$ 的交點的 x 座標



因此，撰寫如下程式即可。

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

int main() {
    double r = 2.0; // 因為想求 √2
    double a = 2.0; // 將初始值隨意的設為 2.0

    for (int i = 1; i <= 5; i++) {
        // 求點 (a, f(a)) 的 x 座標與 y 座標
        double zahyou_x = a;
        double zahyou_y = a * a * a;

        // 求切線的斜率 [令 y = (sessen_a)x + sessen_b]
```

← 從程式碼 4.3.1 變更的部分

```
        double sessen_a = 3.0 * zahyou_x * zahyou_x;
        double sessen_b = zahyou_y - sessen_a * zahyou_x;

        // 求下一個 a 的值 next_a
        double next_a = (r - sessen_b) / sessen_a;
        printf("Step #%d: a = %.12lf -> %.12lf\n", i, a, next_a);
        a = next_a;
    }
    return 0;
}
```

← 從程式碼 4.3.1 變更的部分

此時，輸出會如下所示。急遽地逼近に $\sqrt[3]{2} = 1.259921049894\ldots$，僅用5次而達到小數點後 12 位數的一致性。

```
Step #1: a = 2.000000000000 -> 1.500000000000
Step #2: a = 1.500000000000 -> 1.296296296296
Step #3: a = 1.296296296296 -> 1.260932224742
Step #4: a = 1.260932224742 -> 1.259921860566
Step #5: a = 1.259921860566 -> 1.259921049895
```

另外，有關Python、Java、C的原始碼，請參閱 GitHub 上的 chap4-3.md。

問題 4.3.3

使用二元搜尋法，手動計算 $\sqrt{2}$ 的過程如下表所示。。

操作次數	l	r	m	$m^2 < 2$ 嗎？	範圍示意圖
第 1 次	1.00000	2.00000	1.50000	No	
第 2 次	1.00000	1.50000	1.25000	Yes	
第 3 次	1.25000	1.50000	1.37500	Yes	
第 4 次	1.37500	1.50000	1.43750	No	
第 5 次	1.37500	1.43750	1.40625	Yes	
第 6 次	1.40625	1.43750	1.42188	No	
第 7 次	1.40625	1.42188	1.41406	Yes	
第 8 次	1.41406	1.42188	1.41797	No	
第 9 次	1.41406	1.41797	1.41602	No	

雖然進行了 9 次操作，但 $\sqrt{2} = 1.41421\ldots$ 的小數點後 6 位仍未能一致。

因此，製作如下程式，來檢查需要多少次操作才能達到小數點後6位一致吧。（Python、Java、C的程式請參閱chap4-3.md）

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

int main() {
	double l = 1.0;
	double r = 2.0;

	for (int i = 1; i <= 20; i++) {
		double m = (l + r) / 2.0;
		if (m * m < 2.0) l = m;
		else r = m;
		printf("Step #%d: m = %.12lf\n", i, m);
	}
	return 0;
}
```

輸出如下所示，可知在 第15次 操作時，終於達到小數點後 6 位一致。由於牛頓法只需3次操作，與之相比較慢。

```
Step #1: m = 1.500000000000
Step #2: m = 1.250000000000
Step #3: m = 1.375000000000
Step #4: m = 1.437500000000
Step #5: m = 1.406250000000
Step #6: m = 1.421875000000
Step #7: m = 1.414062500000
Step #8: m = 1.417968750000
Step #9: m = 1.416015625000
Step #10: m = 1.415039062500
Step #11: m = 1.414550781250
Step #12: m = 1.414306640625
Step #13: m = 1.414184570312
Step #14: m = 1.414245605469
Step #15: m = 1.414215087891
Step #16: m = 1.414199829102
Step #17: m = 1.414207458496
Step #18: m = 1.414211273193
Step #19: m = 1.414213180542
Step #20: m = 1.414214134216
```

這樣的二元搜尋法，藉由 1 次操作將精準度變成 2 倍，因此要將精準度提高 P 倍，大約需要 $\log_2 P$ 次操作。本次操作中$P = 10^5$，因此操作次數為 $\log_2 P \fallingdotseq 16$ 次，與實際次數幾乎一致。

問題 4.3.4

根據指數法則（→**2.3.9項**） $10^{0.3} = 1000^{0.1} = \sqrt[10]{1000}$ 。因此，可以考慮例如以下的方法。。

注意，如 x^5 的乘方（整數次方），即使不使用 pow 函數，也可以只用四則運算如 x * x * x * x * x 來計算。

方法1

令 $f(x) = x^{10}, r = 2$ ，適用一般化的牛頓法（→**4.3.6項**） 。在此， $f'(x) = 10x^9$ 。

方法2

透過二元搜尋法（→**節末問題4.3.3**）求出使 $x^{10} = 1000$ 的 x 值。顯然 $1 < x < 2$ ，因此可以設初始值為 $l = 1, r = 2$ 。

還有許多其他方法，請務必思考看看。