4.3 節末問題 4.3 の解答

問題 4.3.1

この問題は、多項式関数の微分（→**4.3.3項**）の理解を問う問題です。答えは以下のようになります。

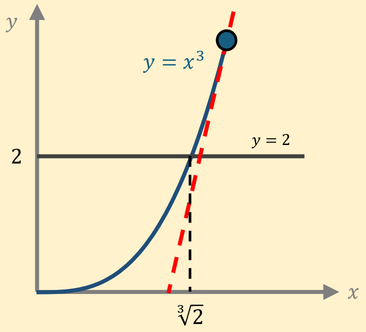
1. $f'(x) = 7$
2. $f'(x) = 2x + 4$
3. $f'(x) = 5x^4 + 4x^3 + 3x^2 + 2x + 1$

問題 4.3.2

$\sqrt[3]{2}$ の値は、以下のような方針で求めることができます。（→**4.3.6項**）

- $f(x) = x^3$ とする。ここで $f'(x) = 3x^2$。
- 最初、適当な初期値 a を設定する。
- その後、a の値を以下に更新し続ける。

> 点 $(a, f(a))$ における接線と直線
> $y = 2$ の交点の x 座標



このため、以下のようなプログラムを書けば良いです。

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

int main() {
    double r = 2.0; // √2 を求めたいから
    double a = 2.0; // 初期値を適当に 2.0 にセットする

    for (int i = 1; i <= 5; i++) {
        // 点 (a, f(a)) の x 座標と y 座標を求める
        double zahyou_x = a;
        double zahyou_y = a * a * a;  ← コード 4.3.1 からの変更部分

        // 接線の傾きを求める [y = (sessen_a)x + sessen_b とする]
```

```
        double sessen_a = 3.0 * zahyou_x * zahyou_x;    ← コード 4.3.1 からの変更部分
        double sessen_b = zahyou_y - sessen_a * zahyou_x;

        // 次の a の値 next_a を求める
        double next_a = (r - sessen_b) / sessen_a;
        printf("Step #%d: a = %.12lf -> %.12lf¥n", i, a, next_a);
        a = next_a;
    }
    return 0;
}
```

このとき、出力は以下のようになります。急激に $\sqrt[3]{2} = 1.259921049894 \ldots$ に近づき、たった 5 回で 12 桁目まで一致します。

```
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
```

なお、Python・JAVA・C のソースコードにつきましては、GitHub の chap4-3.md をご覧ください。

問題 4.3.3

二分探索法を用いて、手計算で $\sqrt{2}$ を求める過程を以下の表に示します。

操作回数	l	r	m	$m^2 < 2$ か？	範囲のイメージ
1 回目	1.00000	2.00000	1.50000	No	
2 回目	1.00000	1.50000	1.25000	Yes	
3 回目	1.25000	1.50000	1.37500	Yes	
4 回目	1.37500	1.50000	1.43750	No	
5 回目	1.37500	1.43750	1.40625	Yes	
6 回目	1.40625	1.43750	1.42188	No	
7 回目	1.40625	1.42188	1.41406	Yes	
8 回目	1.41406	1.42188	1.41797	No	
9 回目	1.41406	1.41797	1.41602	No	

9 回操作を行いましたが、$\sqrt{2} = 1.41421 \ldots$ の下 6 桁とはなかなか一致しません。

そこで、以下のようなプログラムを作成し、何回の操作で一致する桁数が 6 桁に達するかを調べてみましょう。（Python・JAVA・C のプログラムは chap4-3.md をご覧ください）

```
#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;
}
```

このとき、出力は以下のようになり、15 回目の操作でやっと 6 桁一致することが分かります。ニュートン法は 3 回なので、それに比べれば遅いです。

```
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
```

なお、このような二分探索法では、1 回の操作で精度が 2 倍になるため、精度を P 倍に上げるにはおよそ $\log_2 P$ 回の操作が必要です。今回の場合は $P = 10^5$ であり、操作回数は $\log_2 P \fallingdotseq 16$ 回とほぼ一致します。

問題 4.3.4

指数法則（→**2.3.9項**）より、$10^{0.3} = 1000^{0.1} = \sqrt[10]{1000}$ です。このため、たとえば以下のような方法が考えられます。

なお、x^5 のような累乗（整数乗）は、pow 関数を使わなくても `x * x * x * x * x` といったように四則演算だけで計算することができます。

方法1

$f(x) = x^{10}, r = 2$ として、一般化したニュートン法（→**4.3.6項**）を適用する。ここで、$f'(x) = 10x^9$ となる。

方法2

$x^{10} = 1000$ となるような x の値を二分探索（→**節末問題4.3.3**）によって求める。明らかに $1 < x < 2$ なので、初期値としては $l = 1, r = 2$ などを設定すれば良い。

ほかにも多数の方法がありますので、是非考えてみてください。