# 4.1 節末問題 4.1 の解答

## 問題 4.1.1

**(1)** $\vec{A}+\vec{B}=(2+3,4-9)=(5,-5)$ なので、答えは以下のようになります。

- $|\vec{A}|=\sqrt{2^2+4^2}=\sqrt{20}=2\sqrt{5}$（$\sqrt{5}$ の 2 倍）
- $|\vec{B}|=\sqrt{3^2+(-9)^2}=\sqrt{90}=3\sqrt{10}$（$\sqrt{10}$ の 3 倍）
- $|\vec{A}+\vec{B}|=\sqrt{5^2+(-5)^2}=\sqrt{50}=5\sqrt{2}$（$\sqrt{5}$ の 2 倍）

**(2)** 内積の公式（→**4.1.4項**）にしたがって計算すると、

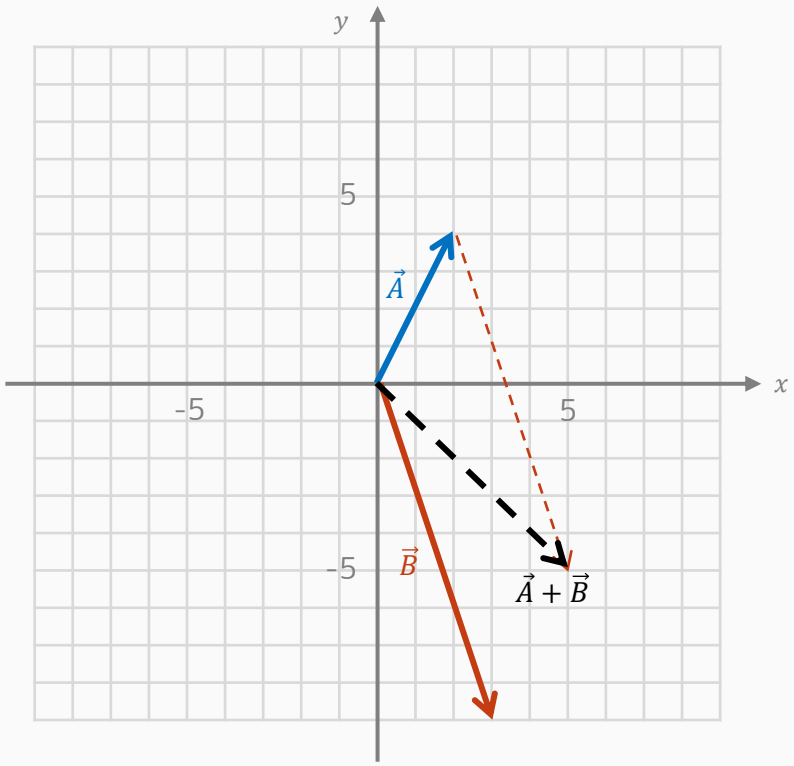
$\vec{A}\cdot\vec{B}=2\times3+4\times(-9)=-30$ となります。

**(3)** 下図を見れば一瞬で分かるのですが、あえて内積を利用して求めましょう。

(2) の答えより内積が負であるため、なす角は 90 度を超えます。

**(4)** 外積の公式（→**4.1.5項**）にしたがって計算すると、

$|A\times B|=|2\times(-9)-3\times4|=30$ となります。



## 問題 4.1.2

点 $(x_i, y_i)$ と点 $(x_j, y_j)$ の間の距離は以下の式で表されます：

$$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

したがって、以下のようにすべての点の組 $(i, j)$ を全探索するプログラムを書くと、正解が得られます。なお、ルートは `sqrt` 関数を用いて計算することができます。

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

int N;
double x[2009], y[2009];
double Answer = 1000000000.0; // 非常に大きい値に初期化

int main() {
    // 入力
    cin >> N;
    for (int i = 1; i <= N; i++) cin >> x[i] >> y[i];

    // 全探索
    for (int i = 1; i <= N; i++) {
        for (int j = i + 1; j <= N; j++) {
            // dist は i 番目の点と j 番目の点の距離
            double dist = sqrt((x[i]-x[j]) * (x[i]-x[j]) + (y[i]-y[j]) * (y[i]-y[j]));
            Answer = min(Answer, dist);
        }
    }

    // 答えの出力
    printf("%.12lf¥n", Answer);
    return 0;
}
```

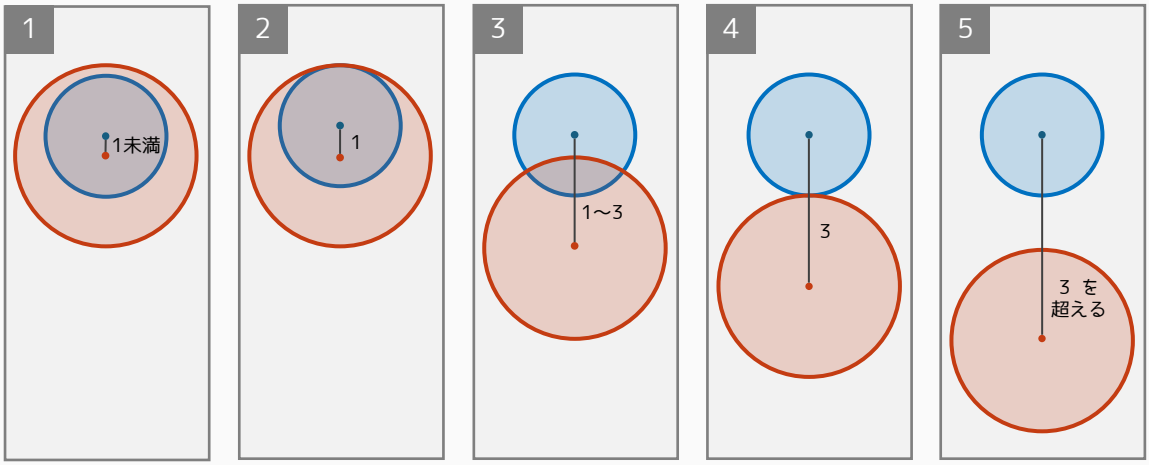
※ Python などのソースコードは chap4-1.md をご覧ください。

## 問題 4.1.3

2 つの円の中心間距離を $d$ とするとき、円の重なり具合は右の表のようになります。（パターンの番号は書籍の問題文を参照してください）

| 中心間距離 $d$ | 重なり具合 |
|---|---|
| $d < \lvert r_1 - r_2 \rvert$ | パターン [1] |
| $d = \lvert r_1 - r_2 \rvert$ | パターン [2] |
| $\lvert r_1 - r_2 \rvert < d < r_1 + r_2$ | パターン [3] |
| $d = r_1 + r_2$ | パターン [4] |
| $r_1 + r_2 < d$ | パターン [5] |

たとえば、半径 2 の円と半径 3 の円を少しずつ離していくと下図のようになります。前ページの表の通り、距離が 1 のときに内側で接し（**内接**し）、距離が 5 のときに外側で接する（**外接**する）ことが分かります。




したがって、以下のように円の中心間距離 d を求めるプログラムを書くと、正解が得られます。2 点間距離を求める方法は、節末問題 4.1.2 で扱った通りです。

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

double X1, Y1, R1;
double X2, Y2, R2;

int main() {
    // 入力
    cin >> X1 >> Y1 >> R1;
    cin >> X2 >> Y2 >> R2;

    // 円の中心間距離を求める
    double d = sqrt((X1 - X2) * (X1 - X2) + (Y1 - Y2) * (Y1 - Y2));

    // 答えを出力
    if (d < abs(R1 - R2)) cout << "1" << endl;
    else if (d == abs(R1 - R2)) cout << "2" << endl;
    else if (d < R1 + R2) cout << "3" << endl;
    else if (d == R1 + R2) cout << "4" << endl;
    else cout << "5" << endl;
    return 0;
}
```

※ Python などのソースコードは chap4-1.md をご覧ください。

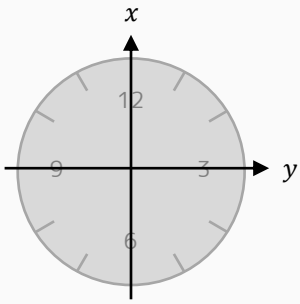
## 問題 4.1.4

この問題は三角関数（→**コラム4**）を利用すると解くことができます。まず、12 時の方向を 0° とするとき、$H$ 時 $M$ 分における角度は次のようになります。

- 時針の角度： $30H + 0.5M°$
- 分針の角度： $6M°$

したがって、時計の中心を座標 $(0, 0)$ とするとき、各針の座標は次のようになります。ただし、12 時の方向が x 軸になっていることに注意してください。

- 時針の先端： $(A\cos(30H + 0.5)°, A\sin(30H + 0.5M)°)$
- 分針の先端： $(B\cos 6H°, B\sin 6H°)$



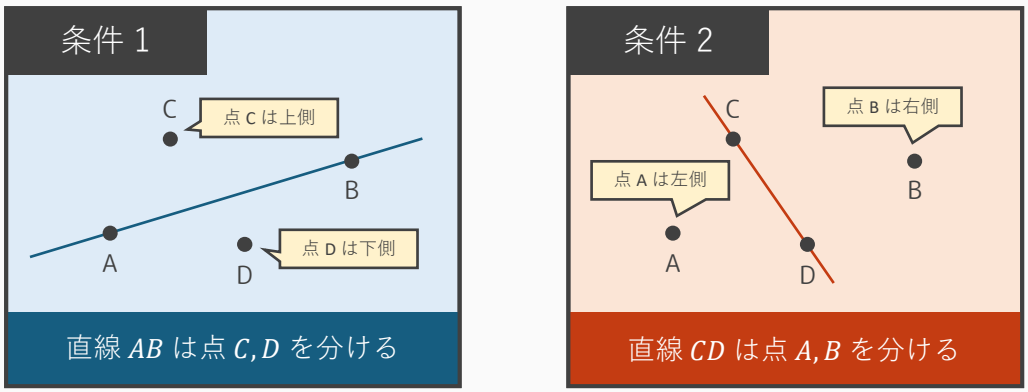

この 2 点間の距離を求めるプログラムを作成すると、正解となります。なお、余弦定理（本書の範囲外）を使って解く方法もあります。

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

const double PI = 3.14159265358979;

int main() {
	// 入力
	double A, B, H, M;
	cin >> A >> B >> H >> M;

	// 座標を求める
	double AngleH = 30.0 * H + 0.5 * M;
	double AngleM = 6.0 * M;
	double Hx = A * cos(AngleH * PI / 180.0), Hy = A * sin(AngleH * PI / 180.0);
	double Mx = B * cos(AngleM * PI / 180.0), My = B * sin(AngleM * PI / 180.0);

	// 距離を求める → 出力
	double d = sqrt((Hx - Mx) * (Hx - Mx) + (Hy - My) * (Hy - My));
	printf("%.12lf¥n", d);
	return 0;
}
```

※ Python などのソースコードは chap4-1.md をご覧ください。

## 問題 4.1.5

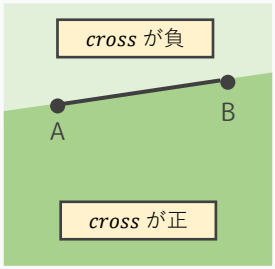
1 つ目の線分の端点を $A,B$ とし、2 つ目の線分の端点を $C,D$ とするとき、2 つの線分が交差する（共通する点を持つ）ための必要十分条件は、基本的には以下の 2 つの条件両方を満たすことです。




そこで、線分 $AB$ が点 $C,D$ を分けるかどうかは、

- $cross(\overrightarrow{AB},\overrightarrow{AC})$ の符号（正負）
- $cross(\overrightarrow{AB},\overrightarrow{AD})$ の符号（正負）

が異なるかどうかで判定することができます。なお、cross 関数は **4.1.5 項**に戻って確認しましょう。




したがって、以下のように実装すると答えを求めることができます。なお、点 $A,B,C,D$ が一直線上に並んでいる場合、$cross(\overrightarrow{AB},\overrightarrow{AC})=0$ の場合など、特殊なケース（**コーナーケース**といいます）では場合分けをする必要があることに注意してください。

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

long long cross(long long ax, long long ay, long long bx, long long by) {
    // ベクトル (ax, ay) と (bx, by) の外積の大きさ
    return ax * by - ay * bx;
}

int main() {
        // 入力
        long long X1, Y1, X2, Y2, X3, Y3, X4, Y4;
        cin >> X1 >> Y1; // 点 A の座標を入力
        cin >> X2 >> Y2; // 点 B の座標を入力
        cin >> X3 >> Y3; // 点 C の座標を入力
        cin >> X4 >> Y4; // 点 D の座標を入力
```

```
    // cross の値を計算
    long long ans1 = cross(X2-X1, Y2-Y1, X3-X1, Y3-Y1);
    long long ans2 = cross(X2-X1, Y2-Y1, X4-X1, Y4-Y1);
    long long ans3 = cross(X4-X3, Y4-Y3, X1-X3, Y1-Y3);
    long long ans4 = cross(X4-X3, Y4-Y3, X2-X3, Y2-Y3);

    // すべて一直線上に並んでいる場合（コーナーケース）
    if (ans1 == 0 && ans2 == 0 && ans3 == 0 && ans4 == 0) {
        // A, B, C, D を数値（正確には pair 型）とみなす
        // 適切に swap することで A<B, C<D を仮定できる
        // そうすると、区間が重なるかの判定（節末問題 2.5.6）に帰着できる
        pair<long long, long long> A = make_pair(X1, Y1);
        pair<long long, long long> B = make_pair(X2, Y2);
        pair<long long, long long> C = make_pair(X3, Y3);
        pair<long long, long long> D = make_pair(X4, Y4);
        if (A > B) swap(A, B);
        if (C > D) swap(C, D);
        if (max(A, C) <= min(B, D)) cout << "Yes" << endl;
        else cout << "No" << endl;
        return 0;
    }

    // そうでない場合
    // IsAB: 線分 AB が点 C, D を分けるか？
    // IsCD: 線分 CD が点 A, B を分けるか？
    bool IsAB = false, IsCD = false;
    if (ans1 >= 0 && ans2 <= 0) IsAB = true;
    if (ans1 <= 0 && ans2 >= 0) IsAB = true;
    if (ans3 >= 0 && ans4 <= 0) IsCD = true;
    if (ans3 <= 0 && ans4 >= 0) IsCD = true;

    // 答えの出力
    if (IsAB == true && IsCD == true) {
        cout << "Yes" << endl;
    }
    else {
        cout << "No" << endl;
    }
    return 0;
}
```

※ Python などのソースコードは chap4-1.md をご覧ください。