基礎計算幾何 1

計算幾何的焦點與一般數學的幾何題不大相同，主要會圍繞著向量以及解析幾何（有架座標的幾何）進行。在高中的競賽（如 TOI / IOI ）中，雖然幾何的題目不常出現，但每次出現往往伴隨著不小的實作量以及拆解包裝的過程。本講義將會以高中的競賽為主，著重於二維平面的計算幾何以及其實作。

本章節將會介紹如何實作基礎的計算幾何元件，以及一些經典的計算幾何習題。

先備知識與技能

- 了解三角函數的定義
- 基礎的向量操作（會用紙筆算）
- 基礎的 STL 容器與函式使用
- 一顆相信自己的心（你一定可以實作出來的 !）

開始前的碎碎念

- 這份講義的數學部份並不嚴謹，例如會混用 A 表示平面上點 A 以及向量 $\vec{OA}$ 。
- 「內部」、「上」的定義若未特別說明，皆是非嚴格的（在邊界或端點上也算）。
- 計算幾何的實作流派非常多，建議多多參考高手的 submission 以及 codebook ，往往可以學到許多特殊的技巧。本講義也會盡量提及所有實作方式。
- 相較於其他主題，計算幾何的內容較為繁雜，請搭配實作練習幫助吸收，不要一次吃到撐（?）

- 練習建議：筆者自己在學習的時候，每一題都會重新打一遍需要的模板。推薦每次不要寫太多題（只寫一題也可以），但請多練習憑記憶打出模板，直到你認為在比賽等高壓環境下也能毫無錯誤的產出為止再使用複製貼上。
- 記得好好看測資限制（三點不共線等），往往可以發現更簡單的作法或是關鍵的觀察。
- 不要科技中毒！好好利用性質簡化問題吧！

1.1 處理浮點數誤差

浮點數誤差麻煩又可怕，運算時常常可以打破基本的數學規則[1]。然而在計算幾何中，不論多努力避免，還有必須使用浮點數的時候，例如：歐氏距離、線段交點。這些誤差會在我們做「點是否在線上」等二元判斷的時候造成很大的問題，因此，我們會需要使用一些工具來處理誤差。廢話不多說，直接上 code ：

```
const double eps = 1e-8;
int sign(double a) { return fabs(a) < eps ? 0 : (a > 0) - (a
   < 0); }
```

從程式碼中不難理解 `eps` （epsilon 的縮寫，常以 ε 表示）代表誤差的容許範圍，也就是「多小以下的數字可以忽略」，而 `sign` （數學中常以 sgn 或 signum 表示）則是回傳考慮了誤差後，一個浮點數應該是正數、負數還是 0 。ε 的大小通常是 $10^{-8} \sim 10^{-10}$ 之間的數，大小基本上是個經驗法則，賽中如果出錯了也會調整 ε 看看是不是誤差導致的。如果想要知道怎麼好好估計 ε 的大小，請參考進階計算幾何講義。

1.2 向量

眾所周知，C++ 裡面的 `vector` 不是向量[2]。為了能讓程式進行基本的向量運算，我們需要自行實作向量型別。以下會以筆者熟悉的風格撰寫。

一樣直接上程式碼：

```
// 向量本體
struct Vec {
    double x, y;
    Vec(double x = 0, double y = 0): x(x), y(y) {}
```

[1]不信你可以試試看 `0.1 + 0.2 == 0.3` ，答案是 `false` 。
[2]也有人說 `std::array` 和 `std::vector` 應該要互換名字比較正確。

```
};

// 加減乘除
Vec operator + (Vec a, Vec b)
{ return Vec(a.x + b.x, a.y + b.y); }
Vec operator - (Vec a, Vec b)
{ return Vec(a.x - b.x, a.y - b.y); }
Vec operator * (Vec a, double b)
{ return Vec(a.x * b, a.y * b); }
Vec operator / (Vec a, double b)
{ return Vec(a.x / b, a.y / b); }

// 內外積
double dot(Vec a, Vec b)
{ return a.x * b.x + a.y * b.y; }
double cross(Vec a, Vec b)
{ return a.x * b.y + a.y * b.x; }

// 向量長
double abs2(Vec a)
{ return dot(a, a); }
double abs(Vec a)
{ return sqrt(dot(a, a)); }

// 可有可無的輸入輸出， debug 時好用
istream& operator >> (istream& is, Vec v)
{ return is >> v.x >> v.y; }
ostream& operator << (ostream& os, Vec v)
{ return os << '(' << v.x << ',' << v.y << ')'; }
```

其實 + * / 很少用，通常只會打 - 而已。善用複製貼上會事半功倍喔！

而在整數向量底下，abs2 也會是整數，因此要取最常的向量的長度時不妨先以 abs2 比較，最後再開根號。

跟著蕭電這樣做

常見的向量實作方式還有使用 `pair<double, double>` 或是 `std::complex<double>`，後者可以省去加減乘除以及輸入輸出的實作，內外積也較簡短，但是會把 x 和 y 鎖死，不能作為變數名稱。各個流派各自有優缺點，建議多看看各家實作再決定一個喜歡的。

內外積的應用

根據數學上的定義，我們知道 $\vec{A}\cdot\vec{B}=|\vec{A}||\vec{B}|\cos\theta, \vec{A}\times\vec{B}=|\vec{A}||\vec{B}|\sin\theta$ 。因為向量長度（通常）恆正，因此內外積的正負號就可以來判斷兩向量的角度關係：

輸出	1	0	-1
`sign(dot(A, B))`	同向 $\lvert\theta\rvert < 90°$	正交 $\lvert\theta\rvert = 90°$	反向 $\lvert\theta > 90$
`sign(cross(A, B))`	逆時針 $0° < \theta < 180°$	共線 $\theta = 0° \cup 180°$	順時針 $-180° < \theta < 0\theta$

Table 1.1: 內外積正負號與向量夾角的關係

除此之外 $\vec{A}\times\vec{B}$ 也等於兩向量所張平行四邊形的有向面積，同時也是三角形 $\triangle OAB$ 有向面積。

為了實作上的方便，通常我們會實作兩個函式，判斷三點 A, B, C 所成向量 $\vec{AB}, \vec{AC}$ 的角度關係：

```
int ori(Vec a, Vec b, Vec c)
{ return sign(cross(b - a, c - a)); }
int dir(Vec a, Vec b, Vec c)
{ return sign(dot(b - a, c - a)); }
```

1.3 線段、射線與直線

兩點可以決定唯一的線段、射線與直線，程式碼也是如此實作：

```
typedef pair<Vec, Vec> Seg;
```

點到直線的距離

雖然有公式可以帶，但是這裡有個簡單的方式：剛剛有提到兩向量外積大小就是其所張平行四邊形面積，而平行四邊形面積又等於底乘高，假設我們要求 C 到 $\overleftrightarrow{AB}$ 的距離，那就讓 $\overline{AB}$ 當底，高就是所求距離！

$dis(C, \overleftrightarrow{AB}) = \frac{|\vec{AB} \times \vec{AC}|}{|\vec{AB}|}$