二维平面几何

基本定义

点

```
typedef double dbl;
const dbl pi = acos(-1), eps = 1e-8;
int sgn(dbl x) { return x < -eps ? -1 : x > eps; }

struct vec { dbl x, y; };
vec operator+(vec v1, vec v2) { return { v1.x + v2.x, v1.y + v2.y }; }
vec operator-(vec v1, vec v2) { return { v1.x - v2.x, v1.y - v2.y }; }
dbl operator*(vec v1, vec v2) { return v1.x * v2.x + v1.y * v2.y; }
dbl operator^(vec v1, vec v2) { return v1.x * v2.y - v1.y * v2.x; }
vec operator*(vec v, dbl k) { return { v.x * k, v.y * k }; }
vec operator/(vec v, dbl k) { return { v.x / k, v.y / k }; }

bool operator<(vec v1, vec v2) { return v1.x == v2.x ? v1.y < v2.y : v1.x <
v2.x; }
bool operator==(vec v1, vec v2) { return v1.x == v2.x && v1.y == v2.y; }
bool operator>(vec v1, vec v2) { return v2 < v1; }

dbl dot(vec v0, vec v1, vec v2) { return (v1 - v0) * (v2 - v0); }
dbl crx(vec v0, vec v1, vec v2) { return (v1 - v0) ^ (v2 - v0); }
dbl len(vec v) { return hypot(v.x, v.y); }
dbl len2(vec v) { return v * v; }
//  方位角，范围在[0,2pi)
dbl arg(vec v) { dbl r = atan2(v.y, v.x); return r < 0? 2 * pi + r : r; }
//  归一化
vec unif(vec v) { return v / len(v); }
//  方位角为f的单位向量
vec univ(dbl f) { return { cos(f), sin(f) }; }
//  将p绕原点旋转f度
vec rot(vec p, dbl f) { return { cos(f) * p.x - sin(f) * p.y, sin(f) * p.x +
cos(f) * p.y }; }
//  将p绕点o旋转f度
vec rot(vec o, vec p, dbl f) { return o + rot(p - o, f); }
//  将p绕原点逆时针旋转pi/2度
vec t90(vec p) { return { -p.y, p.x }; }
```

直线

```
struct line { vec p, v; };
//  求p在l上的投影
vec proj(line l, vec p) { return p + l.v * ((l.p - p) * l.v / len2(l.v)); }
//  求p关于o的对称点
vec refl(vec o, vec p) { return o + o - p; }
//  求p关于l的对称点
vec refl(line l, vec p) { return refl(proj(l, p), p); }
//  求l1和l2的交点，平行时结果未定义
vec litsc(line l1, line l2) { return l2.p + l2.v * ((l1.v ^ (l2.p - l1.p)) /
(l2.v ^ l1.v)); }
//  求p到l的距离，将fabs去掉后符号为正则p在l的右侧
dbl lpdis(line l, vec p) { return fabs(crx(l.p, p, l.p + l.v)) / len(l.v); }
```

线段

```
struct seg { vec p1, p2; };
bool onseg(seg s, vec p){return!sgn(crx(p,s.p1,s.p2))&&sgn(dot(p, s.p1,
s.p2))==-1;}

//  0为不相交，1为严格相交，2表示交点为某线段端点，3为线段平行且部分重合
int sitsc(seg s1, seg s2) {
    vec p1 = s1.p1, p2 = s1.p2, q1 = s2.p1, q2 = s2.p2;
    if (max(p1.x,p2.x)<min(q1.x,q2.x)||min(p1.x,p2.x)>max(q1.x,q2.x)) return 0;
    if (max(p1.y,p2.y)<min(q1.y,q2.y)||min(p1.y,p2.y)>max(q1.y,q2.y)) return 0;
    dbl x=crx(p2,p1,q1),y=crx(p2,p1,q2),z=crx(q2,q1,p1),w=crx(q2,q1,p2);
    if (sgn(x)==0&&sgn(y)==0) return 3;
    if (sgn(x)*sgn(y)<0&&sgn(z)*sgn(w)<0) return 1;
    if (sgn(x)*sgn(y)<=0&&sgn(z)*sgn(w)<=0) return 2;
    return 0;
}

//  求p到线段s的距离
dbl spdis(seg s, vec p) {
    if (dot(s.p1, s.p2, p) < eps) return len(p - s.p1);
    if (dot(s.p2, s.p1, p) < eps) return len(p - s.p2);
    return fabs(crx(p, s.p1, s.p2)) / len(s.p1 - s.p2);
}
```

圆

```
struct cir { vec o; dbl r; };
//  求两个圆相交面积
dbl ccarea(cir c1, cir c2) {
    if (c1.r > c2.r) swap(c1, c2);
    dbl d = len(c1.o - c2.o);
    if (sgn(d - (c1.r + c2.r)) >=0) return 0;
    if (sgn(d - abs(c1.r - c2.r)) <= 0) {
        dbl r = min(c1.r, c2.r);
        return r * r * pi;
    }
    dbl x = (d * d + c1.r * c1.r - c2.r * c2.r) / (2 * d);
    dbl t1 = acos(x / c1.r), t2 = acos((d - x) / c2.r);
    return c1.r * c1.r * t1 + c2.r * c2.r * t2 - d * c1.r * sin(t1);
}

//  求圆与直线的交点，没有返回false
```

```
bool clitsc(cir c, line l, vec& p1, vec& p2) {
    dbl x = l.v * (l.p - c.o), y = l.v * l.v;
    dbl d = x * x - y * ((l.p - c.o) * (l.p - c.o) - c.r * c.r);
    if (sgn(d) == -1) return false; d = max(d, (dbl)0);
    vec p = l.p - l.v * (x / y), w = l.v * (sqrt(d) / y);
    p1 = p + w; p2 = p - w; return true;
}

//  求圆与圆的交点，没有返回false
bool ccitsc(cir c1, cir c2, vec& p1, vec& p2) {
    dbl s1 = len(c1.o - c2.o);
    if (sgn(s1 - c1.r - c2.r) == 1 || sgn(s1 - abs(c1.r - c2.r)) == -1) return
false;
    dbl s2 = (c1.r * c1.r - c2.r * c2.r) / s1, a = (s1 + s2) / 2, b = (s1 - s2)
/ 2;
    vec o = (c2.o - c1.o) * (a / (a + b)) + c1.o;
    vec d = t90(unif(c2.o - c1.o)) * sqrt(c1.r * c1.r - a * a);
    p1 = o + d; p2 = o - d; return true;
}
//  求点到圆的切线，没有返回false
bool cptan(cir c, vec p, vec& p1, vec& p2) {
    dbl x = (p-c.o)*(p-c.o), y=x-c.r*c.r;
    if (y < eps) return false;
    vec o = (p-c.o)*(c.r*c.r/x);
    vec d =t90((p-c.o)*(-c.r*sqrt(y)/x));
    o = o + c.o; p1 = o + d; p2 = o - d;
    return true;
}
//  求两个圆的外侧公切线，没有返回false
bool ccetan(cir c1, cir c2, line& l1, line& l2) {
    //  assert(c1 != c2)
    if (!sgn(c1.r - c2.r)) {
        vec v = t90(unif(c2.o - c1.o) * c1.r);
        l1 = { c1.o + v, c2.o - c1.o };
        l2 = { c1.o - v, c2.o - c1.o };
        return true;
    }
    else {
        vec p = (c2.o*c1.r-c1.o*c2.r) / (c1.r-c2.r);
        vec p1, p2, q1, q2;
        if (cptan(c1,p,p1,p2)&&cptan(c2,p,q1,q2)) {
            if (c1.r < c2.r) swap(p1, p2), swap(q1, q2);
            l1 = { p1, q1 - p1 }; l2 = { p2, q2 - p2 };
            return true;
        }
    }
    return false;
}
//  求两个圆的内侧公切线，没有返回false
bool ccitan(cir c1, cir c2, line& l1, line& l2) {
    vec p = (c1.o * c2.r + c2.o * c1.r) / (c1.r + c2.r);
    vec p1, p2, q1, q2;
    if (cptan(c1, p, p1, p2) && cptan(c2, p, q1, q2)) {
        l1 = { p1, q1 - p1 }; l2 = { p2, q2 - p2 };
        return true;
    }
    return false;
}
```

三角形

内切圆圆心

```
vec incenter(vec a, vec b, vec c) {
    dbl d1 = len(b-c),d2=len(c-a),d3=len(a-b),
        s = fabs(crx(a,b,c));
    return (1/(d1+d2+d3))*(d1*a+d2*b+d3*c);
}
```

外接圆圆心

```
vec circumcenter(vec a, vec b, vec c) {
    b=b-a; c=c-a; dbl d1 = b*b, d2 = c*c, d = 2*(b^c);
    return a - (1/d)*vec{b.y*d2-c.y*d1,c.x*d1-b.x*d2};
}
```