附录

附件1：几何学预处理.cpp

用于对运动学模型的几何规律做基础分解，方便后续调用。

|  |
| --- |
| #include <algorithm>  #include <cmath>  #include <cstdio>  #include <cstring>  #include <vector>  using namespace std;  #define EPS 1e-11  #define INF 1e18  #define PI 3.14159265358979323846  #define EQ(t1, t2) (abs((t1) - (t2)) < EPS)  #define LE(t1, t2) ((t1) <= (t2)-EPS)  #define LEQ(t1, t2) ((t1) < (t2) + EPS)  #define NEXT(i, n) ((i) + 1 >= (n) ? 0 : (i) + 1)  #define PREV(i, n) ((i) > 0 ? (i)-1 : (n)-1)  inline int SGN(double t) { return LE(t, 0) ? -1 : LE(0, t) ? 1 : 0; }  struct Point {  double x, y;  bool operator==(const Point& p) const { return EQ(x, p.x) && EQ(y, p.y); }  bool operator<(const Point& p) const {  return LEQ(x, p.x) && (LE(x, p.x) || LE(y, p.y));  }  Point operator+(Point& p) { return {x + p.x, y + p.y}; }  Point operator-(Point& p) { return {x - p.x, y - p.y}; }  double operator\*(Point& p) { return x \* p.y - y \* p.x; }  Point operator\*(double value) { return {x \* value, y \* value}; }  Point operator/(double value) { return {x / value, y / value}; }  double dot(Point& p) { return x \* p.x + y \* p.y; }  double r2() { return x \* x + y \* y; }  double r() { return sqrt(x \* x + y \* y); }  double dis2(Point& p) { return (\*this - p).r2(); }  double dis(Point& p) { return (\*this - p).r(); }  // 1锛氱偣鍦ㄧ洿绾垮乏杈?-1锛氱偣鍦ㄧ洿绾垮彸杈? 0锛氱偣鍦ㄧ洿绾夸笂  int direction(Point& p1, Point& p2) {  return SGN(x \* (p1.y - p2.y) + p1.x \* (p2.y - y) + p2.x \* (y - p1.y));  }  bool onLine(Point& p1, Point& p2) { return direction(p1, p2) == 0; }  // 鍒ゆ柇鐐规槸鍚﹀湪绾挎涓?  bool onLineSeg(Point& p1, Point& p2) {  return onLine(p1, p2) && inRect(p1, p2);  }  /\*  0锛歱1p鍨傜洿p1p2  1: p2p鍨傜洿p1p2  (0,1): p鍦╬1鐐瑰瀭绾夸笌p2鐐瑰瀭绾夸箣闂?  \*/  double lineRelation(Point& p1, Point& p2) {  Point t = p2 - p1;  return t.dot(\*this - p1) / t.r2();  }  Point footPoint(Point& p1, Point& p2) {  double r = lineRelation(p1, p2);  return p1 + (p2 - p1) \* r;  }  // 涓庣洿绾跨殑璺濈  double lineDis(Point& p1, Point& p2) {  return abs((p1 - \*this) \* (p2 - \*this)) / p1.dis(p2);  }  double lineSegDis(Point& p1, Point& p2, Point& ret);  double lineSegArrayDis(Point\* p, int lineNum, Point& ret);  // 鍏充簬鐩寸嚎鐨勫绉扮偣  Point mirror(Point& p1, Point& p2) {  Point foot = footPoint(p1, p2);  return foot \* 2 - \*this;  }  // 閫嗘椂閽堟棆杞?  Point rotate(double angle) {  Point f = {sin(angle), cos(angle)};  return {\*this \* f, dot(f)};  }  Point rotate90() { return {-y, x}; }  double cosAngle(Point& p1, Point& p2) {  Point t1 = \*this - p1, t2 = \*this - p2;  return t1.dot(t2) / sqrt(t1.r2() \* t2.r2());  }  double sinAngle(Point& p1, Point& p2) {  Point t1 = \*this - p1, t2 = \*this - p2;  return abs(t1 \* t2) / sqrt(t1.r2() \* t2.r2());  }  double tanAngle(Point& o) {  if (EQ(x, o.x)) return y - o.y >= 0 ? INF : -INF;  return (y - o.y) / (x - o.x);  }  double angle(Point& p1, Point& p2) { return acos(cosAngle(p1, p2)); }  double angle(Point& o) { return atan2(y - o.y, x - o.x); }  bool inRect(Point& p1, Point& p2) {  return LEQ((p1.x - x) \* (p2.x - x), 0) &&  LEQ((p1.y - y) \* (p2.y - y), 0);  }  int inPolygon(Point\* p, int n);  int inConvex(Point\* p, int n);  int inCircle(Point& o, double r) {  double dist = dis2(o);  return SGN(r \* r - dist);  }  void pointcut(Point& o, double r, Point& ret1, Point& ret2);  Point nearnestPoint(Point& o, double r);  };  double Point::lineSegDis(Point& p1, Point& p2, Point& ret) {  double r = lineRelation(p1, p2);  if (LEQ(r, 0))  ret = p1;  else if (LEQ(1, r))  ret = p2;  else  ret = footPoint(p1, p2);  return dis(ret);  }  // input lineNum+1 points  double Point::lineSegArrayDis(Point\* p, int lineNum, Point& ret) {  Point tp;  double td, mind = INF;  for (int i = 0; i < lineNum; i++) {  td = lineSegDis(p[i], p[i + 1], tp);  if (LE(td, mind)) {  mind = td;  ret = tp;  }  }  return mind;  }  // donnot include extream points, and donnot include coincidence.  inline bool lineSegLineSegIntersect(Point& p1, Point& p2, Point& q1,  Point& q2) {  Point pq1 = p1 - q1, p12 = p2 - p1, q12 = q2 - q1;  return SGN(pq1 \* q12) \* SGN((p2 - q1) \* q12) < 0 &&  SGN(pq1 \* p12) \* SGN((p1 - q2) \* p12) < 0;  }  // include extream points and coincidence.  inline bool lineSegLineSegIntersect2(Point& p1, Point& p2, Point& q1,  Point& q2) {  if (!LEQ(min(q1.x, q2.x), max(p1.x, p2.x)) ||  !LEQ(min(p1.x, p2.x), max(q1.x, q2.x)) ||  !LEQ(min(q1.y, q2.y), max(p1.y, p2.y)) ||  !LEQ(min(p1.y, p2.y), max(q1.y, q2.y)))  return false;  Point pq1 = p1 - q1, p12 = p2 - p1, q12 = q2 - q1;  return SGN(pq1 \* q12) \* SGN((p2 - q1) \* q12) <= 0 &&  SGN(pq1 \* p12) \* SGN((p1 - q2) \* p12) <= 0;  }  // donot include extream points, and donot include coincidence.  inline bool lineLineSegIntersect(Point& l1, Point& l2, Point& p1, Point& p2) {  Point line = l2 - l1;  return SGN((p1 - l1) \* line) \* SGN((p2 - l1) \* line) < 0;  }  // donnot include coincidence.  inline bool lineLineIntersect(Point& p1, Point& p2, Point& q1, Point& q2) {  return !EQ((p2 - p1) \* (q2 - q1), 0);  }  inline Point lineLineIntersectPoint(Point& p1, Point& p2, Point& q1,  Point& q2) {  Point q12 = q2 - q1;  double k = (p2 - p1) \* q12;  if (EQ(k, 0)) return {INF \* INF, INF \* INF};  double r = ((q1 - p1) \* q12) / k;  return p1 + (p2 - p1) \* r;  }  // 澶栧績  Point circumcenter(Point& p1, Point& p2, Point& p3) {  Point t1 = (p1 + p2) \* 0.5, t2, t3 = (p2 + p3) \* 0.5, t4;  t2 = t1 + (p1 - p2).rotate90();  t4 = t3 + (p2 - p3).rotate90();  return lineLineIntersectPoint(t1, t2, t3, t4);  }  // 鍐呭績  Point incenter(Point& p1, Point& p2, Point& p3) {  double r12 = p1.dis(p2), r23 = p2.dis(p3), r31 = p3.dis(p1);  Point t1 = (p2 \* r31 + p3 \* r12) / (r12 + r31),  t2 = (p1 \* r23 + p3 \* r12) / (r12 + r23);  return lineLineIntersectPoint(p1, t1, p2, t2);  }  // 鍨傚績  Point prepencenter(Point& p1, Point& p2, Point& p3) {  Point t1 = p1 + (p2 - p3).rotate90();  Point t2 = p2 + (p1 - p3).rotate90();  return lineLineIntersectPoint(p1, t1, p2, t2);  }  // 閲嶅績  inline Point barycenter(Point& p1, Point& p2, Point& p3) {  return (p1 + p2 + p3) / 3;  }  // 鍐呭垏鍦?  inline double apothem(Point& p1, Point& p2, Point& p3) {  Point p12 = p2 - p1, p13 = p3 - p1, p23 = p3 - p2;  return abs(p12 \* p23) / (p12.r() + p13.r() + p23.r());  }  // 澶栨帴鍦?  inline double circumradius(Point& p1, Point& p2, Point& p3) {  Point p12 = p2 - p1, p13 = p3 - p1, p23 = p3 - p2;  return sqrt(p12.r2() \* p23.r2() \* p13.r2()) / (2 \* abs(p12 \* p23));  }  // 閫嗘椂閽?锛岄『鏃堕拡-1  int getPolygonDirection(Point\* p, int n) {  int index = 0;  for (int i = 1; i < n; i++) {  if (p[i] < p[index]) index = i;  }  return p[index].direction(p[NEXT(index, n)], p[PREV(index, n)]);  }  bool checkConvex(Point\* p, int n) {  int direction = p[0].direction(p[n - 1], p[1]);  if (direction == 0) return false;  if (p[n - 1].direction(p[n - 2], p[0]) != direction) return false;  for (int i = n - 2; i > 0; i--) {  if (p[i].direction(p[i - 1], p[i + 1]) != direction) return false;  }  return true;  }  // 娉ㄦ剰椤烘椂閽堥潰绉负璐?  double polygonArea(Point\* p, int n) {  double area = 0;  for (int i = n - 2; i > 0; i--) area += p[i].y \* (p[i - 1].x - p[i + 1].x);  area += p[0].y \* (p[n - 1].x - p[1].x);  area += p[n - 1].y \* (p[n - 2].x - p[0].x);  return area / 2;  }  // 鍐呴儴杩斿洖1锛岃竟鐣?锛屽閮?1  int Point::inPolygon(Point\* p, int n) {  int i, j = n - 1, odd = -1;  for (i = 0; i < n; j = i++) {  if (LE(p[i].y, y) != LE(p[j].y, y)) {  double tx =  (y - p[j].y) / (p[i].y - p[j].y) \* (p[i].x - p[j].x) + p[j].x;  if (LEQ(tx, x)) {  if (LE(tx, x))  odd = -odd;  else  return 0;  }  } else if (onLineSeg(p[i], p[j]))  return 0;  }  return odd;  }  int Point::inConvex(Point\* p, int n) {  int \_direction = p[1].direction(p[2], p[0]);  if (direction(p[0], p[1]) != \_direction) {  if (onLineSeg(p[0], p[1])) return 0;  return -1;  }  if (direction(p[n - 1], p[0]) != \_direction) {  if (onLineSeg(p[n - 1], p[0])) return 0;  return -1;  }  int left = 2, right = n - 1;  while (left < right) {  int mid = (left + right) >> 1;  if (direction(p[0], p[mid]) == \_direction)  left = mid + 1;  else  right = mid;  }  int ret = direction(p[left - 1], p[left]);  return ret == \_direction ? 1 : ret == 0 ? 0 : -1;  }  // 浠ヤ笅涓夊嚱鏁板彧鍏佽閫嗘椂閽堟柟鍚?  // angle array size >= 2\*n,return offset  int lineConvexIntersectPointInit(Point\* p, int n, double angle[]) {  int ret = 0;  for (int i = 0, j = n - 1; i < n; j = i++) angle[j] = p[i].angle(p[j]);  do  angle[ret + n] = angle[ret++];  while (LE(angle[ret - 1], angle[ret]) && ret < n);  return ret;  }  // ret鍜宺et2鍒嗗埆涓鸿窛鐩寸嚎鏈€杩戝拰鏈€杩滅殑鐐逛笅鏍?  int lineConvexIntersect(Point& p1, Point& p2, Point\* p, int n, double angle[],  int offset, int& ret1, int& ret2) {  int pos[2];  double k[2];  k[0] = p1.angle(p2);  k[1] = k[0] <= 0 ? k[0] + PI : k[0] - PI;  for (int i = 0; i < 2; i++) {  pos[i] = (upper\_bound(angle + offset, angle + offset + n, k[i] - EPS) -  angle) %  n;  if (p[pos[i]].onLine(p1, p2))  return p[NEXT(pos[i], n)].onLine(p1, p2) ? 3 : 1;  }  ret1 = pos[0];  ret2 = pos[1];  return p[pos[0]].direction(p1, p2) == p[pos[1]].direction(p1, p2) ? 0 : 2;  }  void lineConvexIntersectPoint(Point& p1, Point& p2, Point\* p, int n, int i1,  int i2, Point& ret1, Point& ret2) {  for (int i = 0, l, r; i < 2; i++) {  if (i) {  l = min(i1, i2);  r = max(i1, i2);  } else {  l = max(i1, i2);  r = min(i1, i2) + n;  }  while (l < r) {  int mid = (l + r) >> 1;  if (p[mid % n].direction(p1, p2) == p[r % n].direction(p1, p2))  r = mid;  else  l = mid + 1;  }  l %= n;  (i ? ret1 : ret2) = lineLineIntersectPoint(p1, p2, p[l], p[PREV(l, n)]);  }  }  bool lineSegInPolygon(Point p1, Point p2, Point\* p, int n) {  if (p2 < p1) swap(p1, p2);  int s1 = p1.inPolygon(p, n), s2 = p2.inPolygon(p, n), id = -1, pos = 0;  if (s1 == -1 || s2 == -1) return false;  while (p[pos].onLine(p1, p2)) pos++;  int i = pos, j = pos, d = p[j].direction(p1, p2), d1, d2 = d;  do {  i = NEXT(i, n);  d1 = d2;  d2 = p[i].direction(p1, p2);  if (d2 \* d == -1) {  if (lineSegLineSegIntersect(p[i], p[j], p1, p2)) return false;  if (d1 == 0 && p1 < p[id] && p[id] < p2 && p1 < p[j] && p[j] < p2)  return false;  d = d2;  }  if (d1 == 0 && d2 && p1 < p[j] && p[j] < p2) id = j;  if (d2 == 0 && d1 && p1 < p[i] && p[i] < p2) id = i;  } while ((j = i) != pos);  if (s1 == 0 && s2 == 0) {  if (id == -1) return ((p1 + p2) \* 0.5).inPolygon(p, n) >= 0;  Point q1 = p1, q2 = p2;  for (int i = 0; i < n; i++) {  if (p[i].onLine(p1, p2)) {  if (p[i] < p[id]) q1 = max(q1, p[i]);  if (p[id] < p[i]) q2 = min(q2, p[i]);  }  }  return ((q1 + p[id]) \* 0.5).inPolygon(p, n) >= 0 &&  ((q2 + p[id]) \* 0.5).inPolygon(p, n) >= 0;  }  return true;  }  Point gravityCenter(Point\* p, int n) {  if (n < 3) {  if (n == 1)  return p[0];  else  return (p[0] + p[1]) \* 0.5;  }  double area = 0;  Point ret = {0, 0};  for (int i = 0, j = n - 1; i < n; j = i++) {  double t = p[i] \* p[j];  area += t;  ret.x += (p[i].x + p[j].x) \* t;  ret.y += (p[i].y + p[j].y) \* t;  }  return ret / (3 \* area);  }  // sort p[] first , ret[n] must be available to visit.  int convexHullSorted(Point\* p, int n, Point\* ret) {  int j = 0;  for (int i = 0; i < n; i++) {  while (j >= 2 && p[i].direction(ret[j - 2], ret[j - 1]) != 1) j--;  ret[j++] = p[i];  }  int mid = j + 1;  for (int i = n - 2; i >= 0; i--) {  while (j >= mid && p[i].direction(ret[j - 2], ret[j - 1]) != 1) j--;  ret[j++] = p[i];  }  return j - 1;  }  void convexHullSorted(Point\* p, int n, Point\* up, int& retUp, Point\* down,  int& retDown) {  retUp = retDown = 0;  for (int i = 0; i < n; i++) {  while (retUp >= 2 && p[i].direction(up[retUp - 2], up[retUp - 1]) != -1)  retUp--;  while (retDown >= 2 &&  p[i].direction(down[retDown - 2], down[retDown - 1]) != 1)  retDown--;  up[retUp++] = p[i];  down[retDown++] = p[i];  }  }  // p2缁昿1閫嗘椂閽堣浆90搴︿唬琛ㄥ钩闈㈠唴閮紝鑷澧炲姞4涓崐骞抽潰鍋氳竟鐣?  #define judge(p, q) (p.direction(q.second[0], q.second[1]) < 0)  #define intersect(p, q) \  lineLineIntersectPoint(p.second[0], p.second[1], q.second[0], q.second[1])  int halfPlainIntersect(Point (\*p)[2], int n, Point\* ret) {  vector<pair<double, Point\*> > v(n), line(n);  for (int i = 0; i < n; i++) v[i] = make\_pair(p[i][1].angle(p[i][0]), p[i]);  sort(v.begin(), v.end());  int m = 0, l = 0, r = 0;  for (int i = 1; i < n; i++) {  if (!EQ(v[i].first, v[m].first))  v[++m] = v[i];  else if (!judge(v[i].second[0], v[m]))  v[m] = v[i];  }  if (EQ(v[0].first + 2 \* PI, v[m].first) && !judge(v[m--].second[0], v[0]))  v[0] = v[m + 1];  vector<Point> q(n);  line[0] = v[0];  for (int i = 1; i <= m; i++) {  while (l < r && judge(q[r - 1], v[i])) r--;  while (l < r && judge(q[l], v[i])) l++;  if (l == r && LEQ(line[l].first + PI, v[i].first)) return 0;  line[++r] = v[i];  q[r - 1] = intersect(line[r - 1], line[r]);  }  while (l < r && judge(q[r - 1], line[l])) r--;  if (l == r) return 0;  q[r] = intersect(line[l], line[r]);  int num = unique(q.begin() + l, q.begin() + r + 1) - q.begin() - l;  memcpy(ret, &q[l], sizeof(Point) \* num);  if (num > 1 && ret[0] == ret[num - 1]) num--;  return num;  }  // These points must be put counter-clockwise.  int polygonKernel(Point\* p, int n, Point\* ret) {  Point(\*q)[2] = new Point[n][2];  for (int i = 0, j = n - 1; i < n; j = i++) q[j][0] = p[j], q[j][1] = p[i];  int m = halfPlainIntersect(&q[0], n, ret);  delete[] q;  return m;  }  // return two points representing ax+by<=c  void getPlain(double a, double b, double c, Point\* p) {  if (EQ(a, 0))  p[0] = {0, c / b};  else  p[0] = {c / a, 0};  p[1] = {-b, a};  p[1] = p[1] + p[0];  }  // These points must be put counter-clockwise.  // Ensure p[n] exists and p[n]=p[0].  double convexDiameter(Point\* p, int n, Point& ret1, Point& ret2) {  double ret = 0;  for (int i = 0, j = 1; i < n; i++) {  double t1 = (p[i + 1] - p[i]) \* (p[j] - p[i]), t2;  for (; LE(t1, t2 = (p[i + 1] - p[i]) \* (p[j + 1] - p[i])); t1 = t2) {  if (++j == n) j = 0;  }  double td2 = p[i].dis2(p[j]);  if (ret < td2) {  ret = td2;  ret1 = p[i];  ret2 = p[j];  }  td2 = p[i + 1].dis2(p[j]);  if (ret < td2) {  ret = td2;  ret1 = p[i + 1];  ret2 = p[j];  }  }  return sqrt(ret);  }  // These points must be put counter-clockwise.  // Ensure p[n] exists and p[n]=p[0].  double convexWidth(Point\* p, int n) {  double ret = INF;  for (int i = 0, j = 1; i < n; i++) {  double t1 = (p[i + 1] - p[i]) \* (p[j] - p[i]), t2;  for (; LE(t1, t2 = (p[i + 1] - p[i]) \* (p[j + 1] - p[i])); t1 = t2) {  if (++j == n) j = 0;  }  ret = min(ret, t1 / p[i].dis(p[i + 1]));  }  return ret;  }  struct NearestPointsStruct {  Point p1, p2;  double d2;  vector<Point> v;  };  inline bool nearestPointsCmp(Point& p1, Point& p2) {  return LEQ(p1.y, p2.y) && (LE(p1.y, p2.y) || LE(p1.x, p2.x));  }  void nearestPointsInternal(Point\* p, int left, int right,  NearestPointsStruct& s) {  if (right - left < 8) {  for (int i = left; i < right; i++) {  for (int j = i + 1; j < right; j++) {  double td2 = p[j].dis2(p[i]);  if (td2 < s.d2) {  s.d2 = td2;  s.p1 = p[i];  s.p2 = p[j];  }  }  }  return;  }  int mid = (left + right) >> 1;  nearestPointsInternal(p, left, mid, s);  nearestPointsInternal(p, mid, right, s);  s.v.clear();  double l = (p[mid - 1].x + p[mid].x) / 2, d = sqrt(s.d2);  for (int i = mid - 1; i >= left && l - p[i].x < d; i--) s.v.push\_back(p[i]);  for (int i = mid; i < right && p[i].x - l < d; i++) s.v.push\_back(p[i]);  sort(s.v.begin(), s.v.end(), nearestPointsCmp);  for (unsigned int i = 0; i < s.v.size(); i++) {  for (unsigned int j = i + 1; j < s.v.size() && s.v[j].y - s.v[i].y < d;  j++) {  double td2 = s.v[j].dis2(s.v[i]);  if (td2 < s.d2) {  s.d2 = td2;  s.p1 = s.v[i];  s.p2 = s.v[j];  }  }  }  }  double nearestPointsSorted(Point\* p, int n, Point& ret1, Point& ret2) {  NearestPointsStruct s;  s.d2 = INF;  s.v.reserve(n);  nearestPointsInternal(p, 0, n, s);  ret1 = s.p1;  ret2 = s.p2;  return sqrt(s.d2);  }  double farthestPointsSorted(Point\* p, int n, Point& ret1, Point& ret2) {  vector<Point> v(n + 1);  int cnt = convexHullSorted(p, n, &\*v.begin());  v[n] = v[0];  return convexDiameter(&\*v.begin(), cnt, ret1, ret2);  }  int circleLineRelation(Point& o, double r, Point& p1, Point& p2) {  double d = o.lineDis(p1, p2);  if (LE(d, r)) return 1;  if (LE(r, d)) return -1;  return 0;  }  int circleCircleRelation(Point& o1, double r1, Point& o2, double r2) {  double r = o1.dis(o2);  if (LE(r1 + r2, r)) return 4;  if (LEQ(r1 + r2, r)) return 3;  double sub = abs(r1 - r2);  if (LE(sub, r)) return 2;  if (LEQ(sub, r)) return 1;  return 0;  }  // include extream points.  bool circleLineSegIntersect(Point& o, double r, Point& p1, Point& p2) {  int t1 = p1.inCircle(o, r), t2 = p2.inCircle(o, r);  if (t1 >= 0 || t2 >= 0) return t1 != 1 || t2 != 1;  double t = o.lineRelation(p1, p2);  if (t >= 1 || t <= 0) return false;  Point foot = p1 + (p2 - p1) \* t;  return foot.inCircle(o, r) >= 0;  }  // ret1 is near p1,ret2 is near p2.  void circleLineIntersect(Point& o, double r, Point& p1, Point& p2, Point& ret1,  Point& ret2) {  Point foot = o.footPoint(p1, p2);  double t = r \* r - o.dis2(foot);  t = LEQ(t, 0) ? 0 : sqrt(t / p1.dis2(p2));  ret1 = foot + (p1 - p2) \* t;  ret2 = foot \* 2 - ret1;  }  void circleCircleIntersectPoint(Point& o1, double r1, Point& o2, double r2,  Point& ret1, Point& ret2) {  double d2 = o1.dis2(o2);  double t1 = (r1 \* r1 - r2 \* r2) / (2 \* d2) + 0.5;  double t2 = r1 \* r1 / d2 - t1 \* t1;  t2 = LEQ(t2, 0) ? 0 : sqrt(t2);  Point foot = o1 + (o2 - o1) \* t1;  ret1 = foot + (o2 - o1).rotate90() \* t2;  ret2 = foot \* 2 - ret1;  }  double circleCircleIntersectArea(Point& o1, double r1, Point& o2, double r2) {  int r = circleCircleRelation(o1, r1, o2, r2);  if (r >= 3) return 0;  if (r <= 1) return min(r1, r2) \* min(r1, r2) \* PI;  Point p1, p2;  circleCircleIntersectPoint(o1, r1, o2, r2, p1, p2);  double ret = r1 \* r1 \* o1.angle(p1, o2) + r2 \* r2 \* o2.angle(p1, o1);  return ret - sqrt(o1.dis2(o2) \* p1.dis2(p2)) / 2;  }  void Point::pointcut(Point& o, double r, Point& ret1, Point& ret2) {  double t1 = r \* r / dis2(o);  Point foot = o + (\*this - o) \* t1;  double t2 = t1 - t1 \* t1;  t2 = LEQ(t2, 0) ? 0 : sqrt(t2);  ret1 = foot + (\*this - o).rotate90() \* t2;  ret2 = foot \* 2 - ret1;  }  // ret[0] and ret[2] are on circle o1,ret[1] and ret[3] are on circle o2.  void circleCirclePointcutOuter(Point& o1, double r1, Point& o2, double r2,  Point\* ret) {  Point o12 = o2 - o1;  double d12 = o12.r2(), r = (r1 - r2) / d12;  Point foot1 = o1 + o12 \* (r \* r1), foot2 = o2 + o12 \* (r \* r2);  double t = 1 / d12 - r \* r;  t = LEQ(t, 0) ? 0 : sqrt(t);  Point line = o12.rotate90();  ret[0] = foot1 + line \* (t \* r1);  ret[1] = foot2 + line \* (t \* r2);  ret[2] = foot1 \* 2 - ret[0];  ret[3] = foot2 \* 2 - ret[1];  }  void circleCirclePointcutInner(Point& o1, double r1, Point& o2, double r2,  Point\* ret) {  Point o12 = o2 - o1;  double d12 = o12.r2(), r = (r1 + r2) / d12;  Point foot1 = o1 + o12 \* (r \* r1), foot2 = o2 - o12 \* (r \* r2);  double t = 1 / d12 - r \* r;  t = LEQ(t, 0) ? 0 : sqrt(t);  Point line = o12.rotate90();  ret[0] = foot1 + line \* (t \* r1);  ret[1] = foot2 - line \* (t \* r2);  ret[2] = foot1 \* 2 - ret[0];  ret[3] = foot2 \* 2 - ret[1];  }  Point Point::nearnestPoint(Point& o, double r) {  Point p = \*this - o;  double d = p.r();  if (EQ(d, 0)) return o;  return o + p \* (r / d);  }  // Upset the order before using this function.  double minCoveringCircle(Point\* p, int n, Point& ret) {  if (n == 1) {  ret = p[0];  return 0;  }  double r2 = p[0].dis2(p[1]) / 4;  ret = (p[0] + p[1]) \* 0.5;  for (int i = 2; i < n; i++) {  if (LE(r2, ret.dis2(p[i]))) {  ret = (p[0] + p[i]) \* 0.5;  r2 = p[0].dis2(p[i]) / 4;  for (int j = 1; j < i; j++) {  if (LE(r2, ret.dis2(p[j]))) {  ret = (p[i] + p[j]) \* 0.5;  r2 = p[i].dis2(p[j]) / 4;  for (int k = 0; k < j; k++) {  if (LE(r2, ret.dis2(p[k]))) {  ret = circumcenter(p[i], p[j], p[k]);  r2 = ret.dis2(p[k]);  }  }  }  }  }  }  return sqrt(r2);  }  int unitCoveringCircle(Point\* p, int n, double r) {  int ret = 0;  vector<pair<double, bool> > v;  v.reserve(2 \* n);  double t = r \* r \* 4;  for (int i = 0; i < n; i++) {  v.clear();  int value = 0;  for (int j = 0; j < n; j++) {  if (LEQ(p[i].dis2(p[j]), t) && i != j) {  double a = p[j].angle(p[i]);  double b = acos(p[i].dis(p[j]) / r / 2);  double t1 = a - b, t2 = a + b;  if (t1 < -PI / 2) {  t1 += 2 \* PI;  if (t2 < -PI / 2)  t2 += 2 \* PI;  else  value++;  }  v.push\_back(make\_pair(t1, true));  v.push\_back(make\_pair(t2, false));  }  }  sort(v.begin(), v.end());  if (value > ret) ret = value;  for (unsigned int j = 0; j < v.size(); j++) {  if (v[j].second) {  value++;  if (value > ret) ret = value;  } else  value--;  }  }  return ret + 1;  }  /\*  double circlePolygonAreaIntersect(Point& o,double r,Point\* p,int n){  double area=0;  Point p1,p2;  for(int i=0,j=n-1;i<n;j=i++){  int f1=p[i].inCircle(o,r),f2=p[j].inCircle(o,r);  if(f1>=0&&f2>=0)area+=(o-p[i])\*(o-p[j]);  else if(f1>=0&&f2<0){  circleLineIntersect(o,r,p[i],p[j],p1,p2);  area+=(o-p[i])\*(o-p2);  area+=asin(o.sinAngle(p2,p[j]))\*r\*r;  }  else if(f1<0&&f2>=0){  circleLineIntersect(o,r,p[i],p[j],p1,p2);  area+=(o-p1)\*(o-p[j]);  area+=asin(o.sinAngle(p[i],p1))\*r\*r;  }  else if(circleLineSegIntersect(o,r,p[i],p[j])){  circleLineIntersect(o,r,p[i],p[j],p1,p2);  area+=(o-p1)\*(o-p2);  area+=(asin(o.sinAngle(p[i],p1))+asin(o.sinAngle(p2,p[j])))\*r\*r;  }  else area+=asin(o.sinAngle(p[i],p[j]))\*r\*r;  }  return abs(area/2);  }  \*/ |

附件2：coordinate\_transfer.cpp

用于坐标转换。

|  |
| --- |
| #include<bits/stdc++.h>  using namespace std;  typedef long long ll;  const int INF=0x3f3f3f3f;  const long long mod=100000007;  const double e=2.718281828459045;  const double pi=3.1415926535;  #define CK cout<<"OK\n";  int main(){  freopen("coordinate\_transfer\_in.txt","r",stdin);  freopen("coordinate\_transfer\_out.txt","w",stdout);  double x[100],y[100];  double ax[100],ay[100];  int n=0;  double k=15.624/27.3244;  double dx=2.998806,dy=0.220528;  while(~scanf("%lf%lf",&x[n],&y[n])){  x[n]-=dx;  y[n]-=dy;  n++;  }  for(int i=0;i<n;i++){  double a=k\*k+1;  double b=-2\*x[i]-2\*k\*y[i];  double c=x[i]\*x[i]+y[i]\*y[i];  double xi=-b/2/a;  ax[i]=-sqrt(  max(a\*xi\*xi+b\*xi+c,0.0)  )+15;    a=1+1/k/k;  b=2/k\*y[i]-2\*x[i];  c=x[i]\*x[i]+y[i]\*y[i];  xi=-b/2/a;  ay[i]=sqrt(  max(a\*xi\*xi+b\*xi+c,0.0)  );  }  for(int i=0;i<n;i++){  printf("%.7lf\t%.7lf\n",ax[i],ay[i]);  }  return 0;  } |

附件3：pic\_1.cpp

用于提供图片所需数据。

|  |
| --- |
| #include<bits/stdc++.h>  using namespace std;  typedef long long ll;  const int INF=0x3f3f3f3f;  const long long mod=100000007;  const double e=2.718281828459045;  const double pi=3.1415926535;  #define CK cout<<"OK\n";  int main(){    freopen("pic\_1\_in.txt","r",stdin);  freopen("pic\_1\_out.txt","w",stdout);  double a[100],b[100];  int n=0;  while(~scanf("%lf%lf",&a[n],&b[n])){  n++;  }  for(int i=0;i<n;i++){  printf("%.5lf,",a[i]);  }  printf("\n");  for(int i=0;i<n;i++){  printf("%.5lf,",b[i]);  }  printf("\n");  return 0;  } |

附件4：计算圆轨迹.cpp

用于计算匀速圆周运动中的轨迹。

|  |
| --- |
| /\*  姹傚渾浜?  Example锛?  缁欏嚭涓や釜鍦嗙幆锛屽渾蹇冧綅缃笉鍚岋紝浣嗘槸澶у皬鍦嗗崐寰勭浉鍚岋紝姹備袱涓渾鐜浉浜ょ殑闈㈢Н  Solution锛?  闈㈢Н涓哄ぇ鍦嗕氦闈㈢Н-2\*澶у皬鍦嗕氦闈㈢Н+灏忓渾浜ら潰绉?  \*/  using namespace std;  const double EPS = 1e-12;  const double PI = acos(-1.0);  const int inf = ~0U >> 1;  int sgn(double x) {  if (fabs(x) < EPS) return 0;  if (x < 0)  return -1;  else  return 1;  }  struct Point {  double x, y;  Point() {}  Point(double x1, double y1) {  x = x1;  y = y1;  }  Point operator-(const Point &b) const { return Point(x - b.x, y - b.y); }  double operator\*(const Point &b) const { return x \* b.x + y \* b.y; } //鐐圭Н  double operator^(const Point &b) const { return x \* b.y - y \* b.x; } //鍙夌Н  };  struct Line {  Point u, v;  Line() {}  Line(Point u1, Point v1) {  u = u1;  v = v1;  }  };  double dist(Point a, Point b) { return sqrt((a - b) \* (a - b)); }  // 涓ゅ渾鐩镐氦閮ㄥ垎闈㈢Н锛宎锛宐鏄渾蹇冿紝r1锛宺2鏄崐寰?  double Area(Point a, double r1, Point b, double r2) {  double k = dist(a, b);  if (k + EPS >= r1 + r2) return 0; //鐩稿垏鎴栬€呯浉绂?  if (k <= fabs(r1 - r2) + EPS) {  double R = min(r1, r2);  return PI \* R \* R;  }  double x = (k \* k + r1 \* r1 - r2 \* r2) / (2.0 \* k);  double w1 = acos(x / r1);  double w2 = acos((k - x) / r2);  return (w1 \* r1 \* r1 + w2 \* r2 \* r2 - k \* r1 \* sin(w1));  }  int main(int T) {  scanf("%d", &T);  int cnt = 1;  while (T--) {  double x, y, r, R;  Point p1, p2;  scanf("%lf%lf", &r, &R);  scanf("%lf%lf", &x, &y);  p1 = Point(x, y);  scanf("%lf%lf", &x, &y);  p2 = Point(x, y);  double ans =  Area(p1, R, p2, R) - 2 \* Area(p1, R, p2, r) + Area(p1, r, p2, r);  printf("Case #%d: %.6f\n", cnt++, ans);  }  return 0;  } |

附件5：角度变换.cpp

用于处理第一阶段和第三阶段中方向盘正在转动的情况。

|  |
| --- |
| /\*  鏃嬭浆鍗″３  娉ㄦ剰鍦ㄤ娇鐢ㄥ墠鍏堟眰鍑稿寘  \*/  // 鏃嬭浆鍗″３锛屾眰骞抽潰鏈€杩滅偣瀵?  double rotating\_calipers(Point p[], int n) {  double ans = 0;  Point v;  int cur = 1, ans1 = 0, ans2 = 0;  for (int i = 0; i < n; i++) {  v = p[i] - p[(i + 1) % n];  while ((v ^ (p[(cur + 1) % n] - p[cur])) < 0) cur = (cur + 1) % n;  double tmp = dist(p[i], p[cur]);  if (tmp > ans) ans = tmp, ans1 = i, ans2 = cur;  tmp = dist(p[(i + 1) % n], p[(cur + 1) % n]);  if (tmp > ans) ans = tmp, ans1 = (i + 1) % n, ans2 = (cur + 1) % n;  }  printf("%.12f %.12f %.12f %.12f\n", p[ans1].x, p[ans1].y, p[ans2].x,  p[ans2].y);  return ans;  }  // 鏃嬭浆鍗″３锛屾眰涓夎褰㈤潰绉渶澶у€肩殑涓ゅ€?  double rotating\_calipers(Point p[], int n) {  double ans = 0;  Point v;  for (int i = 0; i < n; i++) {  int j = (i + 1) % n;  int k = (j + 1) % n;  while (j != i && k != i) {  ans = max(ans, abs((p[j] - p[i]) ^ (p[k] - p[i])));  while (((p[i] - p[j]) ^ (p[(k + 1) % n] - p[k])) < 0)  k = (k + 1) % n;  j = (j + 1) % n;  }  }  return ans;  }  // 鏃嬭浆鍗″３锛屾眰涓や釜鍑稿寘鐨勬渶灏忚窛绂?  // 鐐筽0鍒扮嚎娈祊1p2鐨勮窛绂?  double pointtoseg(Point p0, Point p1, Point p2) {  return dist(p0, NearestPointToLineSeg(p0, Line(p1, p2)));  }  // 骞宠绾挎p0p1鍜宲2p3鐨勮窛绂?  double dispallseg(Point p0, Point p1, Point p2, Point p3) {  double ans1 = min(pointtoseg(p0, p2, p3), pointtoseg(p1, p2, p3));  double ans2 = min(pointtoseg(p2, p0, p1), pointtoseg(p3, p0, p1));  return min(ans1, ans2);  }  // 寰楀埌鍚戦噺a1a2鍜宐1b2鐨勪綅缃叧绯?  double Get\_angle(Point a1, Point a2, Point b1, Point b2) {  Point t = b1 - (b2 - a1);  return (a2 - a1) ^ (t - a1);  }  double rotating\_calipers(Point p[], int np, Point q[], int nq) {  int sp = 0, sq = 0;  for (int i = 0; i < np; i++)  if (sgn(p[i].y - p[sp].y) < 0) sp = i;  for (int i = 0; i < nq; i++)  if (sgn(q[i].y - q[sq].y) > 0) sq = i;  double tmp;  double ans = 1e99;  for (int i = 0; i < np; i++) {  while (sgn(tmp = Get\_angle(p[sp], p[(sp + 1) % np], q[sq],  q[(sq + 1) % nq])) < 0)  sq = (sq + 1) % nq;  if (sgn(tmp) == 0)  ans = min(ans, dispallseg(p[sp], p[(sp + 1) % np], q[sq],  q[(sq + 1) % nq]));  else  ans = min(ans, pointtoseg(q[sq], p[sp], p[(sp + 1) % np]));  sp = (sp + 1) % np;  }  return ans;  }  double solve(Point p[], int n, Point q[], int m) {  return min(rotating\_calipers(p, n, q, m), rotating\_calipers(q, m, p, n));  } |