Educational Research
Научно-методически статии

ПРОГРАМНИ ЕКСПЕРИМЕНТИ СЪС ЗАДАЧИ ОТ ТИП HERON

Павел Азълов
Пенсилвански държавен университет

Резюме. Задачата на Heron е древна математическа задача, която има точно и елегантно решение.

В статията са формулирани задачи, които представляват обобщение на задачата на Heron, но за които не са известни точни математически решения. Тези задачи имат практическо приложение и за тях са представени приближени решения чрез програми на C++ и MATLAB.

Keywords: Heron type problems, Generalized Heron problems

1. Задачата на Heron

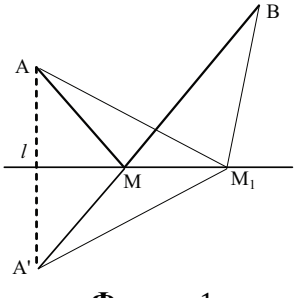
Отправна точка в тази статия е следната древна задача.

Основна задача. *Дадени са права l и две точки A и B, разположени от една и съща страна на l. Да се намери точка M от правата l, за която сборът от разстоянията от A и B до M е минимален* (Hadamard, 1962).

Задачата се среща и в разнообразни забавни формулировки. Тя има кратко и красиво решение.

Решение

Разглеждаме фиг. 1, на която *l* е дадената права, а A и B са точките, които са разположени от едната ѝ страна. Построяваме т. A', симетрична на A по отношение на правата *l*, а с M означаваме пресеч-ната точка на *l* и BA'. Нека M_1 е произволна точка върху *l*, различна от т. M.



Фигура 1

Тъй като MA' = MA и $M_1A' = M_1A$, следва, че:

$BM_1 + M_1A = BM_1 + M_1A' > BA' = BM + MA' = BM + MA$

С това исканата точка M е построена и тя е единствена.

В (Hadamard, 1962) под номер 14 е дадена задача (*задачата за билярда*), която има различна формулировка от задачата на Heron, но практически решението е същото: *Дадени са права xy и две точки A и B, които са от една и съща страна на правата. Да се намери точка M върху правата, за която* $\angle AMx = \angle BMy$.

В тази задача косвено се илюстрира закон от физиката, съгласно който ъгълът, под който пада светлинен лъч върху равнинна повърхност, е равен на ъгъла, под който лъчът се отразява.

По-нататък за краткост точката M, която е решение на задачата на Heron, ще наричаме *минималната точка* за съответната права l и двете точки P_1, P_2, а за самата задача и решението ѝ ще използваме означението $M = (l; P_1, P_2)$.

Може да се отбележи, че изискването в задачата на Heron двете точки да са от едната страна на правата, не е съществено. Когато точките са от различни страни, решението се вижда веднага и всъщност този случай подсказва за решението на основната задача.

Ако не се налагат ограничения за разположението на точките спрямо правата, тогава следва да се разгледат няколко случая. Те са представени в табл. 1, където с d_{MIN} е означено минималното разстояние $P_1M + MP_2$.

Таблица 1. Възможни случаи на взаимно разположение на точките и правата

	Разположение на точките P_1 и P_2	Разположение на минималната точка M
1	Точките са от едната страна на правата l. Те са различни и ортогоналните им проекции върху правата са също различни.	Едно решение: това е общият случай и точката M се определя по начина, посочен в основната задача (фиг. 1), $d_{MIN} = P_1M + MP_2$.
2	Точките са от различни страни на правата l.	Едно решение: точката M е пресечната точка на P_1P_2 и l, $d_{MIN} = P_1P_2$.
3	Точките P_1 и P_2 съвпадат, но не лежат върху правата l.	Едно решение: точката M е ортогоналната проекция на точките, $d_{MIN} = 2P_1M$.
4	Точките P_1 и P_2 са различни, но ортогоналните им проекции върху правата l съвпадат.	Едно решение: точката M е ортогоналната проекция на точките, $d_{MIN} = P_1M + MP_2$.
5	Една от точките е върху правата.	Едно решение: M съвпада с точката, която е върху l, $d_{MIN} = P_1P_2$.
6	Двете точки са върху правата.	Безброй много решения: M може да бъде всяка точка от отсечката P_1P_2, $d_{MIN} = P_1P_2$.

Случаите от 2 до 6 са частни и тривиални, но не са безинтересни, когато решението се представя с компютърна програма. Въпреки това, за да не се утежнява програмният код, те няма да бъдат специално разглеждани в програмните решения на някои от задачите.

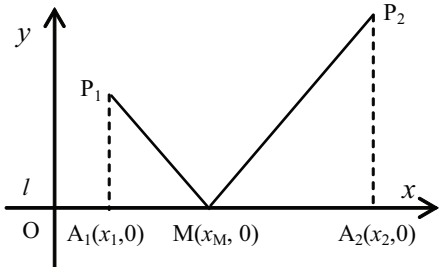
2. Задачата на Heron в координатна система

Разглеждаме задачата на Heron $M = (Ox; P_1, P_2)$ в правоъгълна координатна система Oxy, в която правата l съвпада с оста Ox, а дадените точки $P_1(x_1, y_1)$ и $P_2(x_2, y_2)$, са над оста Ox , т.е. $x_1 \neq x_2$, $y_1 > 0$, $y_2 > 0$.

Задача 1. Да се пресметнат координатите на минималната точка за M = (O*x*; P_1, P_2)

Решение

Нека ортогоналните проекции на P_1 и P_2 върху O*x* са точките $A_1(x_1, 0)$ и $A_2(x_2, 0)$, а M(, 0) е търсената минимална точка (фиг. 2). От подобието на триъгълниците P_1MA_1 и P_2MA_2 следва, че



Фигура 2

$$\frac{P_1A_1}{P_2A_2} = \frac{MA_1}{A_2M} = \frac{x_M - x_1}{x_2 - x_M}$$

Така за стойността на координата *x* на т. M се получава:

$$x_M = \frac{x_1P_2A_2 + x_2P_1A_1}{P_2A_2 + P_1A_1} \qquad (1)$$

Този резултат е обобщен в следващата задача.

Задача 2. Нека O*xy* е правоъгълна координатна система, $P_1(x_1, y_1)$ и $P_2(x_2, y_2)$ са точки в равнината, а *l* е права, минаваща през т. O и пресичаща O*x* под ъгъл α, $0 \le \alpha < \frac{\pi}{2}$. Да се пресметнат координатите (x_M, y_M) на минималната точка M = (*l*; P_1, P_2).

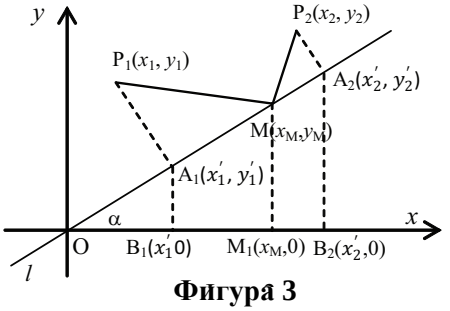
Решение

От съществено значение в тази задача е стойността на ъгъла α (фиг. 3).

– Ако α = ∢ P_1Ox, тогава т. P_1 лежи върху правата *l* и това е случай 5 от табл. 1.

– Ако α = ∢ P_2Ox, тогава т. P_2 лежи върху правата *l* и това е случай 5 от табл. 1.

– Ако ∢ P_1Ox = ∢ P_2Ox, това е случай 6 от табл. 1.



Фигура 3

– Ако правата P_1P_2 пресича *l* под ъгъл $\frac{\pi}{2}$, това съответства на случай 3 или 4 от табл. 1.

По-долу следва решението на задачата в общия случай (случай 1 от табл. 1).

Ортогоналната проекция на произволна точка P върху правата *l* с ъглов коефициент tg(α) е линейна трансформация, определена с матрицата:

$$L = \begin{pmatrix} cos^2(\alpha) & \sin(\alpha)\cos(\alpha) \\ \sin(\alpha)\cos(\alpha) & sin^2(\alpha) \end{pmatrix}$$

Оттук следва, че за координатите на ортогоналната проекция $A_1(x'_1,y'_1)$ на т. P_1 се получава:

$$A_1\begin{pmatrix} x'_1 \\ y'_1 \end{pmatrix} = L \times \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} cos^2(\alpha)\, x_1 + \sin(\alpha)\cos(\alpha)\, y_1 \\ \sin(\alpha)\cos(\alpha)\, x_1 + sin^2(\alpha)\, y_1 \end{pmatrix}$$

Аналогично за координатите на т. A_2 се получава:

$$A_2\begin{pmatrix} x'_2 \\ y'_2 \end{pmatrix} = L \times \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} cos^2(\alpha)\, x_2 + \sin(\alpha)\cos(\alpha)\, y_2 \\ \sin(\alpha)\cos(\alpha)\, x_2 + sin^2(\alpha)\, y_2 \end{pmatrix}$$

От подобието на триъгълниците A_2B_2O, MM_1O и A_1B_1O може да се запише:

$$\frac{P_1A_1}{P_2A_2} = \frac{MA_1}{A_2M} = \frac{M_1B_1}{B_2M_1} = \frac{x_M - x'_1}{x'_2 - x_M}$$

Оттук за координатите (x_M, y_M) на т. M, изразени с координатите (x'_1,y'_1) и (x'_2,y'_2) на ортогоналните проекции на т. P_1 и т. P_2 върху правата l, се получава:

$$x_M = \frac{x'_1\, P_2A_2 + x'_2\, P_1A_1}{P_2A_2 + P_1A_1} \qquad (2)$$

$$y_M = tg(\alpha)x_M = tg(\alpha)\frac{x'_1\, P_2A_2 + x'_2\, P_1A_1}{P_2A_2 + P_1A_1} \qquad (3)$$

Може да се отбележи, че специалният избор на правата l в тази и в следващите задачи не намалява общността на резултатите в тях.

Задача 3. В условията на задача 2 да се визуализира множеството от минимални точки $M(\alpha) = (l(\alpha); P_1, P_2)$ за $0 \le \alpha < \frac{\pi}{2}$.

Решение

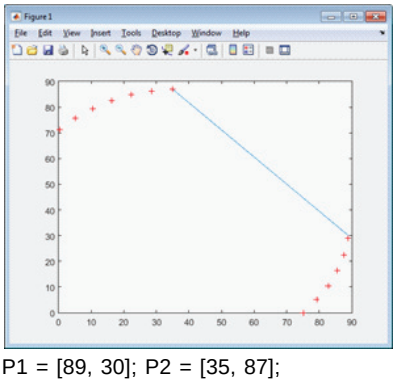
Нека $\alpha_1 = \sphericalangle P_1Ox$ и $\alpha_2 = \sphericalangle P_2Ox$. Ясно е, че ако α е в интервала $[min(\alpha_1, \alpha_2), max(\alpha_1, \alpha_2)]$, тогава задачата се свежда до някой от частните случаи 2, 5 или 6 от табл. 1.

Всъщност решението на задачата изисква намиране и визуализиране на ГМТ със свойството на т. M. Координатите на точката M са пресметнати в задача 2. Остава графично да се визуализира множеството на точките M. Това е извършено с програмния код на функцията minPoints, написана на MATLAB.

```
% P1, P2 – дадените две точки с координатите си в първи квадрант
% n – брой на ъглите alpha, за които се пресмятат минималните
% точки за съответната права

function minPoints( P1, P2, n)
  if P2(1) < P1(1)
    P = P1; P1 = P2; P2 = P;
  end

  x1 = P1(1);  y1 = P1(2);
  x2 = P2(1);  y2 = P2(2);
  alpha1 = atan(y1/x1);
  alpha2 = atan(y2/x2);
  alphaMin = min(alpha1, alpha2);
  alphaMax = max(alpha1, alpha2);

  alpha = 0.0 : pi/n : alphaMin;
  drawM(alpha);
  hold on

  x = x1:0.01:x2;
  y = y1 + (y2 - y1)/(x2 - x1)*(x - x1);
  plot(x, y);

  alpha = alphaMax : pi/n : pi/2 - eps;
  drawM(alpha);
  hold off

  function drawM(alpha)
    % (x1p, y1p) - координати на проекцията
    %              на т. P1 върху правата
    x1p = x1*cos(alpha).^2 + y1*sin(alpha).*cos(alpha);
    y1p = x1*sin(alpha).*cos(alpha) + y1*sin(alpha).^2;

    % (x2p, y2p) - координати на проекцията
    %              на т. P2 върху правата
    x2p = x2*cos(alpha).^2 + y2*sin(alpha).*cos(alpha);
    y2p = x2*sin(alpha).*cos(alpha) + y2*sin(alpha).^2;

    % Пресмятане на разстоянията от P1 и P2 до
    % ортогоналните им проекции им върху правата
```

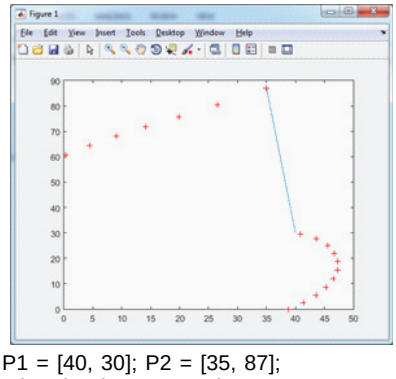
```
    d1 = sqrt((x1 - x1p).^2 + (y1 - y1p).^2);
    d2 = sqrt((x2 - x2p).^2 + (y2 - y2p).^2);
    xM = (x1p.*d2 + x2p.*d1)./(d2 + d1);
    yM = tan(alpha).*xM;
    plot(xM, yM, 'r+')
  end  % End of drawM
end  % End of minPoints
```

Резултатите от четири изпълнения на функцията `minPoints` са показани на фигури 4а, 4б, 4в и 4г. Посочени са и координатите на точките P_1 и P_2.



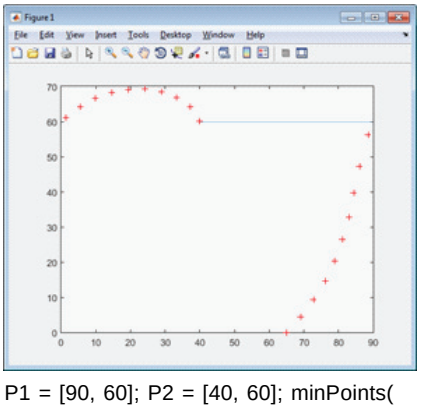
P1 = [89, 30]; P2 = [35, 87];
minPoints(P1, P2, 50);

Фигура 4а



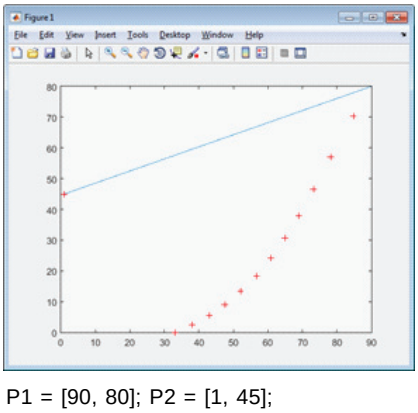
P1 = [40, 30]; P2 = [35, 87];
minPoints(P1, P2, 50);

Фигура 4б



P1 = [90, 60]; P2 = [40, 60]; minPoints(
P1, P2, 50);

Фигура 4в



P1 = [90, 80]; P2 = [1, 45];
minPoints(P1, P2, 50);

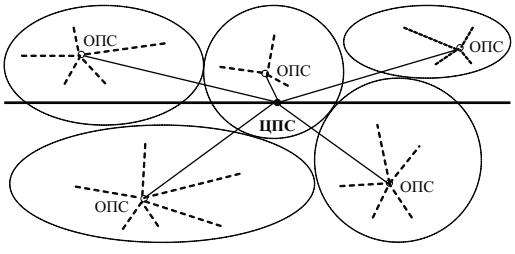
Фигура 4г

3. Задачи от тип Heron

Задачата на Heron е прекрасен пример за оптимизационна геометрична задача, подходяща за изучаване още в училище. Задача, в която се търси точка от дадена права или в общия случай от изпъкнала крива, за която сумата от разстоянията З до няколко (две или повече) други точки е минимална, представлява обобщение на задачата на Heron и затова тук ще бъде наричана *задача от тип Heron*.

Следват примери на три задачи от тип Heron.

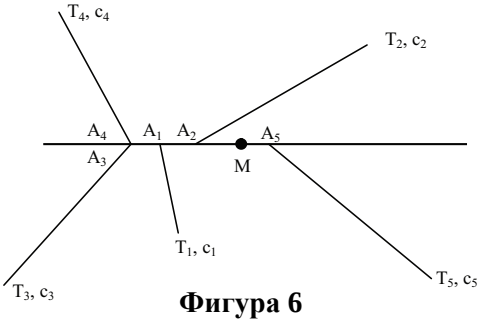
Задача 5. Структурата и организацията на пощенските услуги в една държава е следната (фиг.5). Територията на държавата е разделена на n ($n > 2$) отделни области. В рамките на всяка област има изградена областна пощенска станция (ОПС). Пощенските пратки (писма и колети) се събират и се разнасят до жителите на една и съща област чрез специализирани пощенски автомобили. Ако една пощенска пратка е адресирана до лице от друго селище в същата или друга област, тогава тя се изпраща до съответната ОПС. Областните станции комуникират помежду си чрез централна пощенска станция (ЦПС, т. нар. и хъб), в която ежедневно чрез хеликоптери се пренасят пощенските пратки от и до съответните ОПС, за които адресът на подателя и получателя са от различни области на държавата или евентуално от друга държава. За да се оптимизират разходите по транспортирането на пощенските пратки между ОПЦ и ЦПЦ, е необходимо да се избере място за хъб непосредствено до определена магистрала, така че сумата от разстоянията от всички ОПС до ЦПС да е минимална. Да се пресметнат координатите на новата ЦПС, ако се познават координатите на всяка ОПС и се приеме, че магистралата се описва с линейна функция.



Фигура 5

Задача 6. В прав участък от линията на едно метро трябва да се проектира метростанция, която да облужва жителите на квартал, живеещи в n жилищни блока. Те са разположени от двете страни на линията в рамките на правоъгълна площ с дадени размери. Улиците, по които гражданите могат да се придвижват до линията на метрото, са успоредни и вертикални на линията на метрото. Трябва да се определи точката от линията на метрото, в която да се проектира и построи метростанцията, така че сумата от разстоянията, които живеещите в тези блокове ще изминават до станцията, да е минимална. За всеки блок се предполага, че е представен с координатите на една точка и се знае броят на живеещите в него.

Задача 7. Много от жителите на n селища $T_1, T_2, \ldots, T_n$ работят в град, до който се достига по магистрала. Най-краткият път от селище T_i ($i = 1, 2, \ldots, n$) излиза на магистралата в точка A_i, а разстоянието T_iA_i е k_i километра (фиг. 6). За да се намалят транспортните разходи и да се намали трафикът на автомобили по магистралата, една компания иска да инвестира в построяването на паркинг непосредствено до магистралата.



Фигура 6

Всеки абонат на паркинга може да паркира колата си, с която идва от дома си, и да продължи с автобус до града или да се присъедини с други абонати, за да формират група за придвижване до града само с един автомобил. Да се определи точката M от магистралата, на която да се построи паркингът, така че сумарното разстояние от селищата до паркинга, което се изминава от абонатите, да бъде минимално. Предполага се, че броят за абонатите на паркинга от селище T_i е c_i, $i = 1, 2, \ldots, n$.

Решенията на тези примерни задачи имат практическа стойност, но за тях не е лесно да се посочи точно решение по подобие на решението на задачата на Heron. Ето защо в подобни случаи полезни могат да бъдат *приближени решения*, които да бъдат напълно задоволителни за реални практически нужди.

4. Един експеримент: приближено решение на задачата на Heron

По-долу е представен *експеримент*, при който пресмятането на минималната точка и на съответното разстояние в задачата на Heron ще се извърши по два начина.

Задача 8. Разглежда се задачата $M = (Ox; P_1, P_2)$, в която точките P_1, P_2 са определени с координатите си (x_1, y_1) и (x_2, y_2). Да се напише програма за приближено пресмятане на минималната точка и съответното минимално разстояние, като резултатът се сравни със стойностите, получени от точния метод, представен в задача 1.

Решение

Приемаме разглеждането на тази задача като *експеримент*. С него ще може да се сравнят точното и приближеното решение за конкретни точки, чиито координати са случайни числа. Експериментът се базира на два съществени факта от решението на основната задача. Знае се, че задачата има решение и то е единствено.

Реализацията на експеримента е извършена с програмния модул Heron, написан на C++. Във функцията `approx` се генерира редица от равноотдалечени

точки $T_i(t_i, 0)$ с дадена стъпка h, $t_0 = min(x_1, x_2) \le t_i \le max(x_1, x_2)$, $t_i = t_{i-1} + h$, $i = 1, 2, ...$ За всяка от точките T_i се пресмятат разстоянията ѝ до дадените точки P_1 и P_2 и като резултат функцията връща минималната точка и съответното разстояние.

Програмата е структурирана като програмен модул в три файла: заглавен файл, файл за реализация на функциите и главна функция, с която е проведен експериментът. Преобразуването на модула в C++ клас е тривиално, но това не е направено, за да не се усложнява излишно програмният код.

За по-компактен запис на кода в заглавния файл `Heron.h` „точката“ е дефинирана като структура (`Point`) със съответен конструктор и функция, отпечатваща текущата стойност на точка. Функциите `project` и `distPP` са помощни за функцията `approx`. С първата се пресмята ортогоналната проекция на точка върху права, минаваща през началото на кординатната система и сключваща даден ъгъл с оста O*x*. Втората функция пресмята разстоянието между две точки. С функцията `heron` се пресмятат стойностите на минималната точка съгласно (2) и (3) от задача 2.

```
// Файл: Heron.h
#ifndef HERON_H
#define HERON_H

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

const double PI = 3.1415926;

struct Point
{
  double x, y;
  Point(double x1=0.0, double y1=0.0)
  {
    x = x1;
    y = y1;
  }

  void print(int n)
  {
     cout << fixed << setprecision(n);
     cout << "x = " << setw(10) << x
          << ", y = " << setw(10) << y << endl;
  }
};
```

```
double approxL(Point p1, Point p2, double h, Point& p);
Point  project(Point p, double alpha);
Point  heron(Point p1, Point p2, double alpha, double& minD);
double distPP(Point p1, Point p2);

double approxC(Point p1, Point p2, double r, double h, Point& p);
bool   checkH(Point p1, Point p2, Point p, double eps);
#endif
```

По-долу следват дефинициите на функциите, декларирани в заглавния файл.

```
#include "Heron.h"
// Функцията пресмята минималната точка pM
// и минималното разстояние (minD) с приближение.
// Правата минава през началото на координатната система.
// p1, p2 - точки, за които се пресмята минималното разстояние.
// h - разстояние между две съседни точки от редицата.
// Функцията връща:
//   pM - точка от Ox, за която е достигнат минимумът.
//   minD – стойност на пресметнатото минимално разстояние.

double approxL(Point p1, Point p2, double h, Point& pM)
{
   double minD = DBL_MAX;
   double mD;
   double a = p1.x, b = p2.x;
   if (a > b) swap(a, b);

   double x = a;
   while (x <= b)
   {
      Point p(x, 0.0);
      mD = distPP(p1, p) + distPP(p2, p);  // mD = sumDist(pts, n, p);
      if (mD < minD)
      {
         minD = mD;
         pM = p;
      }
      x = x + h;
   }
   return minD;
}
```

```
// Фукцията пресмята координатите на ортогоналната
// проекция на точка p върху права, минаваща през началото
// на координатната система и сключваща ъгъл alpha с оста Ox.
// Функцията връща точка, която е ортогонална проекция на p.

Point project(Point p, double alpha)
{
  double x, y;
  x = p.x*pow(cos(alpha), 2)    + p.y*sin(alpha)*cos(alpha);
  y = p.x*sin(alpha)*cos(alpha) + p.y*pow(sin(alpha), 2);
  return Point(x, y);
}

// Функцията реализира алгоритъма на Heron за пресмятане на
// минималното разстояние между две точки p1 и p2 до дадена
// точка M от права, минаваща през началото на координатната
// система и сключваща ъгъл alpha с оста Ox.
// Функцията връща координатите на минималната точка M(xM, yM),
// както и стойността на съответното разстояние (minD).

Point heron(Point p1, Point p2, double alpha, double& minD)
{
  minD = DBL_MAX;
  Point p1Pr = project(p1, alpha);
  Point p2Pr = project(p2, alpha);

  double d1 = distPP(p1, p1Pr);
  double d2 = distPP(p2, p2Pr);
  double d  = d1 + d2;

  double xM = (d1*p2Pr.x + d2*p1Pr.x)/d;
  double yM = (d1*p2Pr.y + d2*p1Pr.y)/d;

  Point p(xM, yM);
  minD = distPP(p1, p) + distPP(p, p2);
  return p;
}

// Функцията пресмята разстоянието между две точки p1 и p2 в рав-
нината.
double distPP(Point p1, Point p2)
```

```
{
   return sqrt(pow(p1.x - p2.x, 2) + pow(p1.y - p2.y, 2));
}
```

Същинският експеримент се извършва в главната функция на програмата. В нея за всяка стъпка h, ($h = 10^{-s}$, s = 0, 1, 2, . . . , 7) двете точки се генерират десет пъти със случайни координати (x, y) в интервалите: $1 \leq x \leq 100$ и $0 \leq y \leq 50$. За всяка двойка точки се пресмятат приближените и „точните“ стойности на минималната точка и на съответното минимално разстояние.

```
// Файл: HeronExperiment.cpp
// Експеримент, формулиран в задача 8
#include <iostream>
#include “Heron.h”
#include <cstdlib>
#include <ctime>
using namespace std;

int main()
{
   srand(unsigned int (time(0)));// Стартиране на генератора
                                 // на псевдослучайни числа
   double a = 1.0, b = 100.0;    // Интервали за X
   double c = 0.0, d = 50.0;     // Интервали за Y
   double errM_Total, errD_Total // errM, err_Total - грешка в
                                 //   координатата X
   double errM, errD;            //  errD,  err_Total  -  грешка  в
                                 // разстоянието
   double h = 1.0;               // Стъпка по оста X
   time_t tb, te;                // Променливи за измерване на времето
   Point pM, p;
   int n = 10;
   cout << scientific << setprecision(16);

   for (int st=1; st<8; st++)
   {
     errM_Total = 0.0, errD_Total = 0.0;

     cout << left << setw(26) << “Errors for min point”
     << left << setw(26) << “Errors for min distance” << endl;
```

```
    for (int i=0; i<n; i++)
    {
       // Генерират се псевдослучайни координати в интервалите:
       double p1x = (b-a)*rand()/RAND_MAX + a;  // (a, b)
       double p1y = (d-c)*rand()/RAND_MAX + c;  // (c, d)
       double p2x = (b-a)*rand()/RAND_MAX + a;  // (a, b)
       double p2y = (d-c)*rand()/RAND_MAX + c;  // (c, d)
       Point p1(p1x,p1y), p2(p2x, p2y);         // p1, p2 - две точки

       double minD1;                            // минималното разстояние
       double alpha = 0.0;                      // ъгъл на правата
                                                // спрямо оста Ox
       pM = heron(p1, p2, alpha, minD1);

// Експеримент: Приближено (итеративно) пресмятане
       tb = time(0);                            // Начало на изпълнението
       double minD2 = approxL(p1, p2, h, p);    // Приближен (итеративен)
                                                   метод
       te = time(0);                            // Край на изпълнението
       errM = fabs(pM.x - p.x);                 // Грешка в пресмятането
                                                   на координата
       errD = fabs(minD1 - minD2);              // Грешка в пресмятането
                                                   на разстоянието
       out << setw(26) << errM << setw(26) << errD << endl;
       errM_Total += errM;
       errD_Total += errD;

    }

    cout << "Step = " << h << endl;             // Стъпка h по X
    double elapsed = ((double) (te - tb)) / CLOCKS_PER_SEC;
    cout << "Execution time for \'approx\' = " << elapsed << endl;
    cout << "Average error for minPoint  = " << errM_Total/n << endl;
    cout << "Average error for minDist   = " << errD_Total/n << endl << endl;
    h = h/10.0;                                 // Нова стъпка
  }
 return 0;
}
```

По-долу са приведени резултатите от изпълнението на програмата само за стъпки $h = 10^0$, 10^{-4} и 10^{-6}. Отпечатани са абсолютните и средните грешки от пре-

смятанията, както и времето за изпълнение на функцията за съответната стъпка (табл. 2).

Таблица 2. Извадка от резултатите на експеримента от задача 8

```
Errors for min point      Errors for min distance
2.8866672572007701e-003   1.4530472469687084e-007
1.1274219156476306e-003   1.4028344708094664e-008
3.8754061789127547e-002   3.1160067095470367e-005
1.1202230856799389e-002   9.8888237971550552e-006
4.6601264227348338e-003   1.1387409415419825e-007
4.9885086629684494e-002   2.8644056065729728e-005
2.7421482087259363e-002   1.7195142277159903e-005
1.7456261090394776e-002   8.7953738727719610e-006
2.1044828013415895e-002   7.8731189887548680e-006
4.1085869993722213e-002   5.6820032398263720e-005
Step = 1.0000000000000001e-001
Execution time for ‘approx' = 0.0000000000000000e+000
Average error for minPoint  = 2.1552403605598690e-002
Average error for minDist   = 1.6064982165886477e-005

Errors for min point      Errors for min distance
8.4881643072520774e-005   1.1924683462893881e-010
4.5223040986286378e-004   1.1749634154512023e-008
3.0460122957975955e-004   3.4031238271836628e-008
2.0810234004287054e-004   6.1453420130419545e-010
1.2876654088245232e-004   3.6835956507275114e-010
1.7103966857234809e-004   8.9126217517332407e-010
4.0696975851517436e-004   6.8856564894304029e-009
4.0098363976426299e-004   3.9265586337933200e-009
2.7341373852252104e-004   2.0927970467710111e-009
3.0623992627454300e-004   2.9986466643094900e-009
Step = 1.0000000000000000e-003
Execution time for ‘approx' = 0.0000000000000000e+000
Average error for minPoint  = 2.7372288950893163e-004
Average error for minDist   = 6.3677934036832085e-009

Errors for min point      Errors for min distance
4.2637637420739338e-008   0.0000000000000000e+000
3.4051080888275465e-007   0.0000000000000000e+000
2.9652249189382474e-007   0.0000000000000000e+000
4.0785594990211393e-007   1.4210854715202004e-014
6.8620099824556746e-008   0.0000000000000000e+000
6.1577844689963968e-007   0.0000000000000000e+000
4.5829490957771668e-007   6.3238303482648917e-013
2.7479455866341596e-007   7.1054273576010019e-015
1.9542123652627197e-007   0.0000000000000000e+000
2.1990502574453785e-007   4.2632564145606011e-014
Step = 1.0000000000000002e-006
Execution time for ‘approx' = 1.7999999999999999e-002
Average error for minPoint  = 2.9203411653355717e-007
Average error for minDist   = 6.9633188104489823e-014
```

Вижда се, че резултатите, получени от итеративния метод за приближено пресмятане на минималната точка и съответното разстояние, са много добри.

Това дава основание функцията `approx` да се обобщи за *n* точки. Нейният прототип ще съдържа масив от *n* точки:

```
double approx(Point pts[], int n, double h, Point& p)
```

Самата дефиниция на функцията apprох в по-голямата си част ще остане същата. Единствено операторът

```
mD = distPP(p1, Point(t, 0.0)) + distPP(p2, Point(t, 0.0));
```

трябва да се замени с оператора:

```
mD = sumDist(pts, n, p),
```

където параметърът T съответства на текущата точка от правата, която е кадидат за минималната точка. Следва кодът на функцията sumDist:

```
double sumDist(Point pts[], int n, Point p)
{
   double d = 0.0;
   for (int i = 0; i<n; i++)
     d = d + distPP(pts[i], p);
   return d;
}
```

Положителният резултат от този експеримент е добър повод да се пристъпи към втори експеримент, в който правата *l* ще бъде заменена с изпъкнала крива и по-конкретно с дъга от окръжност.

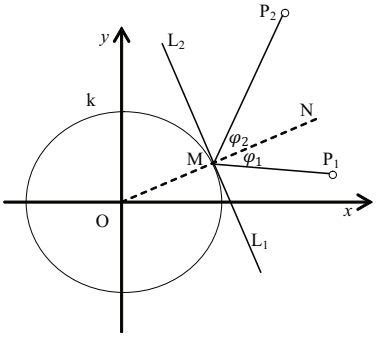
5. Втори експеримент

Разглеждаме задачата от тип Heron M = (k; P_1, P_2), в която k е дъга от окръжност. Целта е експериментално да се провери една хипотеза, чиято идея се основава на метода на Heron, а именно, че ъгълите между P_1M и P_2M с допирателната на окръжността k в т. M са равни (физичен закон за отражението). За тази цел разглеждаме следната задача.

Задача 9. Дадена е окръжност с радиус *r* и център в началото на правоъгълна координатна система, k(O, *r*). Дадени са и две точки в първи квадрант, определени с координатите си, $P_1(x_1, y_1)$ и $P_2(x_2, y_2)$.

а) Да се намери точка $M(x_M, y_M)$ от дъгата на окръжността k в първи квадрант ($x > 0$, $y > 0$), за която сборът от разстоянията от M до P_1 и P_2 да е минимален (фиг. 7).

б) За точката M, определена в а), да се провери дали $\sphericalangle P_1ML_1$ е равен на $\sphericalangle P_2ML_2$ в случая, когато правата L_1L_2 е допирателна към окръжността k в т. M.



Фигура 7

В тази задача ролята на правата *l* от предишните задачи е поета от допирателната L_1L_2 към окръжността в точка M (фиг. 7). Но тъй като $OM \perp L_1L_2$, за да се докаже, че $\sphericalangle P_1ML_1 = \sphericalangle P_2ML_2$, е достатъчно да се докаже, че . $\sphericalangle\varphi_1 = \sphericalangle\varphi_2$

В зависимост от разположението на точките спрямо окръжността могат да се разгледат множество частни случаи (табл. 3).

Таблица 3. Възможни случаи на взаимно разположение на точките и окръжността

	Разположение на точките P1 и P2	Разположение на минималната точка M
1	Точките P1 и P2 са извън кръга, правата P1P2 пресича окръжността в две точки M1 и M2.	Две решения: M = M1 или M = M2, dMIN = |P1P2|.
2	P1 и P2 са върху окръжността.	Две решения: M = P1 или M = P2, dMIN = |P1P2|.
3	Правата P1P2 е допирателна към окръжността в точката M.	Едно решение: M е търсената минимална точка, d_{MIN} = |P1P2|.
4	Една от точките е върху окръжността, а другата е вътре или извън кръга.	Едно решение: M е точката, която е върху окръжността, d_{MIN} = |P1P2|.
5	Една от точките е вътре в кръга, а другата е извън кръга.	Едно решение: M е пресечната точка на P1P2 с окръжността, d_{MIN} = |P1P2|.
6	Двете точки P1 и P2 съвпадат.	Едно решение: минималната точка M е пресечната точка на окръжността и правата OP1.
7	И двете точки P1 и P2 едновременно са извън кръга и правата P1P2 не пресича окръжността.	Общ случай (хипотеза): минималната точка M е точка от окръжността, за която правата OM е ъглополовяща на ъгъла P1MP2 ($\sphericalangle\varphi_1 = \sphericalangle\varphi_2$), фиг. 7.
8	И двете точки P1 и P2 едновременно са вътре кръга.	Общ случай: този случай е аналогичен на случай 6.

На фиг. 7 е представен общият случай 7, който се има предвид при решението на задачата. Ще бъде използван опитът от задача 8 и координатите на минималната точка ще се пресметнат с приближение. Това се реализира с функцията `approxK`. Тя е твърде сходна с функцията `approxL`, но има един допълнителен параметър *r* – радиус на окръжността.

```
// Функцията пресмята M = (k, p1, p2),
// M - точка от окръжността k(O, r)
// с център в началото на координатната система,
// за която: 0 < x(k) < max(p1.x, p2.x),
```

```
// k = 1, 2, ... е редица от равноотстоящи точки
// със стъпка h.
// p1, p2 - точки, за които се пресмята минималното разстояние.
// Функцията връща:
//   pM - точка от Ox, за която е достигнат минимумът.
//   minD – стойност на пресметнатото минимално разстояние.

double approxC(Point p1, Point p2, double r, double h, Point& pM)
{
   Point p;
   double b = min(r, max(p1.x, p2.x));
   double mD;
   double minD = DBL_MAX;

    double x = b;
   while (x >= 0)
   {
     p = Point(x, sqrt(r*r - x*x));
     mD = distPP(p1, p) + distPP(p, p2);
     if (mD < minD)
     {
        minD = mD;
        pM = p;
     }
     x = x - h;
  }
  return minD;
}
```

И тази функция, както и функцията `approxL`, може да се обобщи и броят на точките да е целочислен параметър *n*, *n* > 2, а операторът

```
mD = distPP(p1, p) + distPP(p, p2);
```

трябва да се замени с обръщение към функцията `sumDist`:

```
mD = sumDist(pts, n, p)
```

С функцията `checkH` се проверява дали ъглите са равни. Точността, с която се извършва проверката (сравняването на реалните числа), е определена от параметъра *eps*.

Алгоритъмът, заложен в кода на функцията `checkH`, използва зависимостите между ъглите на две пресичащи се прави с ъглови коефициенти съответно m_1 и m_2. Тогава за ъгъла между тях, изразен чрез ъглите θ_1 и θ_2, се получава:

$$\theta = \theta_2 - \theta_1 \text{ или } \theta = \pi - (\theta_2 - \theta_1).$$

$$tg(\theta) = tg(\theta_2 - \theta_1) = \frac{tg(\theta_2) - tg(\theta_1)}{1 + tg(\theta_2)\,tg(\theta_1)} = \frac{m_2 - m_1}{1 + m_1 m_2}$$

В нашия случай трябва да се пресметнат ъгловите коефициенти m, m_1 и m_2 на правите OM, P_1M и P_2M и съответно те са:

$$m = \frac{y_M}{x_M}, \qquad m_1 = \frac{y_M - y_1}{x_M - x_1}, \qquad m_2 = \frac{y_M - y_2}{x_M - x_2}$$

Следва кодът на функцията.

```
// Функцията проверява дали в точката M от окръжността с радиус r, в
// която изразът p1M + Mp2 достига max, ъгълът между p1 и
// допирателната в т. M е равен на ъгъла между p2 и допирателната в т. M
// Точността, с която се прави проверката, се определя от параметъра eps.
// Стъпката, с която се търси точката M, е определена от параметъра h.

bool checkH(Point p1, Point p2, Point pM, double eps)
{
   double sM = pM.y/pM.x;
   double s1 = (pM.y - p1.y)/(pM.x -p1.x);
   double s2 = (pM.y - p2.y)/(pM.x -p2.x);

   double tg1 = (sM - s1)/(1 + sM*s1);
   double tg2 = (s2 - sM)/(1 + sM*s2);

   return (tg2 - tg1) <= eps;
}
```

По-долу следва главната функция на програмата.

```
// Експеримент, формулиран в задача 9
#include <iostream>
#include "Heron.h"
#include <cstdlib>
#include <ctime>
using namespace std;

int main()
{
   srand(unsigned int (time(0)));  // Стартиране на генератора
                                   // на псевдослучайни числа
   double a1 = 1.0, a2 = 101.0;    // (a1, a2) - интервал
                                   // за абсцисата
   double r =  83.47;              // Радиус на окръжността:
                                   // 33.27,7.47, 83.47
```

```
double b1 = sqrt(r*r - a1*a1); // (b1, b2) - интервал
                               //за ординатата
double b2 = 100.0;

// Генерират се псевдослучайни координати в интервалите:
double p1x = (a2 - a1)*rand()/RAND_MAX + a1;      // (a1, a2)
double p1y = (b2 - b1)*rand()/RAND_MAX + b1;      // (b1, b2)
double p2x = (a2 - a1)*rand()/RAND_MAX + a1;      // (a1, a2)
double p2y = (b2 - b1)*rand()/RAND_MAX + b1;      // (b1, b2)
Point  p1(p1x,p1y), p2(p2x, p2y);                 // p1, p2 -
                                                  //две точки

Point pM;       // Минимална точка
double minD;    // Минимално разстояние
bool test;      // Резултат от теста:
                //1 - успешен, 0 - неуспешен
double h, eps;  // Стъпка (h), точност за теста (eps)
cout << "radius = " << r << endl;
cout << "P1:  "; p1.print(6);
cout << "P2:  "; p2.print(6);

cout << "\nstep" << setw(8) << "eps" << setw(11) << "pM.x"
     << setw(23) << "pM.y" << setw(31) << "min distance" << setw(14)
     << "test" << endl;

// Експеримент: Приближено (итеративно) пресмятане
for (double s=1; s<7; s++)
{
   h = pow(10.0, -s);
   for (double e=1; e<=s; e++)
   {
      eps = pow(10.0, -e);
      // Пресмятане на min точка и разстояние
      double minD = approxC(p1, p2, r, h, pM);
      test = checkH(p1, p2, pM, eps);    // Тестване на хипоте
                                            зата
      cout << setprecision(1) << setw(8) << h << setw(9)
           << eps;
      cout << scientific << setprecision(14);
      cout << setw(23) << pM.x << setw(23) << pM.y
           << setw(23) << minD << setw(2) << test << endl;
```

```
        }
      }
      return 0;
   }
```

Програмата е изпълнена многократно и резултатите на три от тях са дадени в табл. 4.

Таблица 4. Извадка от резултатите на експеримента от задача 9

```
radius = 7.47
P1:  x =  63.147893, y =  75.745046
P2:  x =  12.319315, y =  60.575455
```

step	eps	pM.x	pM.y	min distance	test
0.1	0.1	3.17000000000001e+000	6.76402247187278e+000	1.45993377523788e+002	1
1.0e-002	1.0e-001	3.19000000000009e+000	6.75461323837268e+000	1.45993285530871e+002	1
1.0e-002	1.0e-002	3.19000000000009e+000	6.75461323837268e+000	1.45993285530871e+002	1
1.0e-003	1.0e-001	3.19299999999893e+000	6.75319561393025e+000	1.45993283736262e+002	1
1.0e-003	1.0e-002	3.19299999999893e+000	6.75319561393025e+000	1.45993283736262e+002	1
1.0e-003	1.0e-003	3.19299999999893e+000	6.75319561393025e+000	1.45993283736262e+002	1
1.0e-004	1.0e-001	3.19320000000638e+000	6.75310104766390e+000	1.45993283728143e+002	1
1.0e-004	1.0e-002	3.19320000000638e+000	6.75310104766390e+000	1.45993283728143e+002	1
1.0e-004	1.0e-003	3.19320000000638e+000	6.75310104766390e+000	1.45993283728143e+002	1
1.0e-004	1.0e-004	3.19320000000638e+000	6.75310104766390e+000	1.45993283728143e+002	1
1.0e-005	1.0e-001	3.19322000012608e+000	6.75309159058240e+000	1.45993283728098e+002	1
1.0e-005	1.0e-002	3.19322000012608e+000	6.75309159058240e+000	1.45993283728098e+002	1
1.0e-005	1.0e-003	3.19322000012608e+000	6.75309159058240e+000	1.45993283728098e+002	1
1.0e-005	1.0e-004	3.19322000012608e+000	6.75309159058240e+000	1.45993283728098e+002	1
1.0e-005	1.0e-005	3.19322000012608e+000	6.75309159058240e+000	1.45993283728098e+002	1
1.0e-006	1.0e-001	3.19321599940220e+000	6.75309348233547e+000	1.45993283728096e+002	1
1.0e-006	1.0e-002	3.19321599940220e+000	6.75309348233547e+000	1.45993283728096e+002	1
1.0e-006	1.0e-003	3.19321599940220e+000	6.75309348233547e+000	1.45993283728096e+002	1
1.0e-006	1.0e-004	3.19321599940220e+000	6.75309348233547e+000	1.45993283728096e+002	1
1.0e-006	1.0e-005	3.19321599940220e+000	6.75309348233547e+000	1.45993283728096e+002	1
1.0e-006	1.0e-006	3.19321599940220e+000	6.75309348233547e+000	1.45993283728096e+002	1

```
radius = 83.47
P1:  x =  64.591418, y =  88.429298
P2:  x =  17.302988, y =  84.382984
```

step	eps	pM.x	pM.y	min distance	test
0.1	0.1	2.16914181951348e+001	8.06022535459386e+001	4.94006015808695e+001	1
1.0e-002	1.0e-001	2.17314181951398e+001	8.05914782295743e+001	4.94005540789146e+001	1
1.0e-002	1.0e-002	2.17314181951398e+001	8.05914782295743e+001	4.94005540789146e+001	1
1.0e-003	1.0e-001	2.17354181951946e+001	8.05903995267427e+001	4.94005537167977e+001	1
1.0e-003	1.0e-002	2.17354181951946e+001	8.05903995267427e+001	4.94005537167977e+001	1
1.0e-003	1.0e-003	2.17354181951946e+001	8.05903995267427e+001	4.94005537167977e+001	1

1.0e-004	1.0e-001	2.17352181940773e+001	8.05904534672429e+001	4.94005537160548e+001	1
1.0e-004	1.0e-002	2.17352181940773e+001	8.05904534672429e+001	4.94005537160548e+001	1
1.0e-004	1.0e-003	2.17352181940773e+001	8.05904534672429e+001	4.94005537160548e+001	1
1.0e-004	1.0e-004	2.17352181940773e+001	8.05904534672429e+001	4.94005537160548e+001	1
1.0e-005	1.0e-001	2.17352481851790e+001	8.05904453786469e+001	4.94005537160398e+001	1
1.0e-005	1.0e-002	2.17352481851790e+001	8.05904453786469e+001	4.94005537160398e+001	1
1.0e-005	1.0e-003	2.17352481851790e+001	8.05904453786469e+001	4.94005537160398e+001	1
1.0e-005	1.0e-004	2.17352481851790e+001	8.05904453786469e+001	4.94005537160398e+001	1
1.0e-005	1.0e-005	2.17352481851790e+001	8.05904453786469e+001	4.94005537160398e+001	1
1.0e-006	1.0e-001	2.17352432668690e+001	8.05904467051153e+001	4.94005537160392e+001	1
1.0e-006	1.0e-002	2.17352432668690e+001	8.05904467051153e+001	4.94005537160392e+001	1
1.0e-006	1.0e-003	2.17352432668690e+001	8.05904467051153e+001	4.94005537160392e+001	1
1.0e-006	1.0e-004	2.17352432668690e+001	8.05904467051153e+001	4.94005537160392e+001	1
1.0e-006	1.0e-005	2.17352432668690e+001	8.05904467051153e+001	4.94005537160392e+001	1
1.0e-006	1.0e-006	2.17352432668690e+001	8.05904467051153e+001	4.94005537160392e+001	1

```
radius = 33.27
P1:  x =  83.276925, y =  59.081570
P2:  x =  12.496170, y =  91.216633
```

step	eps	pM.x	pM.y	min distance	test
0.1	0.1	1.69699999999998e+001	2.86166385167792e+001	1.35730337958670e+002	1
1.0e-002	1.0e-001	1.69499999999979e+001	2.86284893069836e+001	1.35730322955179e+002	1
1.0e-002	1.0e-002	1.69499999999979e+001	2.86284893069836e+001	1.35730322955179e+002	1
1.0e-003	1.0e-001	1.69519999999846e+001	2.86273050775046e+001	1.35730322766513e+002	1
1.0e-003	1.0e-002	1.69519999999846e+001	2.86273050775046e+001	1.35730322766513e+002	1
1.0e-003	1.0e-003	1.69519999999846e+001	2.86273050775046e+001	1.35730322766513e+002	1
1.0e-004	1.0e-001	1.69519999999929e+001	2.86273050774997e+001	1.35730322766513e+002	1
1.0e-004	1.0e-002	1.69519999999929e+001	2.86273050774997e+001	1.35730322766513e+002	1
1.0e-004	1.0e-003	1.69519999999929e+001	2.86273050774997e+001	1.35730322766513e+002	1
1.0e-004	1.0e-004	1.69519999999929e+001	2.86273050774997e+001	1.35730322766513e+002	1
1.0e-005	1.0e-001	1.69520100001666e+001	2.86272991557753e+001	1.35730322766513e+002	1
1.0e-005	1.0e-002	1.69520100001666e+001	2.86272991557753e+001	1.35730322766513e+002	1
1.0e-005	1.0e-003	1.69520100001666e+001	2.86272991557753e+001	1.35730322766513e+002	1
1.0e-005	1.0e-004	1.69520100001666e+001	2.86272991557753e+001	1.35730322766513e+002	1
1.0e-005	1.0e-005	1.69520100001666e+001	2.86272991557753e+001	1.35730322766513e+002	1
1.0e-006	1.0e-001	1.69520059877378e+001	2.86273015317844e+001	1.35730322766512e+002	1
1.0e-006	1.0e-002	1.69520059877378e+001	2.86273015317844e+001	1.35730322766512e+002	1
1.0e-006	1.0e-003	1.69520059877378e+001	2.86273015317844e+001	1.35730322766512e+002	1
1.0e-006	1.0e-004	1.69520059877378e+001	2.86273015317844e+001	1.35730322766512e+002	1
1.0e-006	1.0e-005	1.69520059877378e+001	2.86273015317844e+001	1.35730322766512e+002	1
1.0e-006	1.0e-006	1.69520059877378e+001	2.86273015317844e+001	1.35730322766512e+002	1

Дотук всичко беше само експеримент върху една хипотеза. Резултатът от експеримента е отличен. Сега е ред хипотезата да се преформулира като теорема и да се докаже или... опровергае.

REFERENCES

Hadamard, J. (1962). *Lectii de geometrie elementara. Geometrie plana, Editia a doua. Traducere din limba franceza completata cu resolvarile problemelor*. Bucuresti: Editura tehnica.

PROGRAMMING EXPERIMENTS WITH PROBLEMS OF HERON TYPE

Abstract. The Heron's problem is an ancient mathematical problem, which has a precise and elegant solution. The paper discusses generalizations of the Heron problem that have a variety of practical applications. Nowadays, it is still unknown if such problems have precise mathematical solutions. Approximate solutions to similar problems written as programs in C++ and MATLAB are presented in this paper.

✉ **Dr. Pavel Azalov**
Assoc. Prof. of Computer Science
Pennsylvania State University,
Hazleton campus, U.S.A.
E-mail: pka10@psu.edu