1. Проект интерфейса

Программная система принимает на вход файл с исходными данными в формате CSV и файл с входными параметрами и выводит результаты вычислений в файл в формате HTML.

Входной файл в формате CSV

Входной файл в формате CSV имеет имя `input.csv`.

Записи в файле разделяются символами конца строки (CRLF). После последней записи могут идти и могут не идти символы CRLF.

Все записи должны состоять из трёх полей. Поля записи разделяются символом «точка с запятой» (`;`).

В качестве первой записи должен быть заголовок, состоящий из полей: «Категория», «Частота», «Ранг». Регистр в заголовке важен, кодировка Windows-1251.

В следующих записях поле «Частота» должно содержать вещественное положительное число (разделитель целой и дробной части — запятая (`,`)). Поле «Ранг» должно содержать натуральное число.

Значения поля «Ранг» должны быть 1, 2, 3, …. Значения поля «Частота» должны идти по возрастанию.

Входной файл с входными параметрами

Входной файл с входными параметрами имеет имя `input.txt`.

Этот файл должен содержать три числа: δ_0 ν_0 x.

δ_0 — пороговое значение для половины минимальной высоты полосы, содержащей все точки промежутка (вещественное неотрицательное число).

ν_0 — допустимое количество аномальных точек (целое неотрицательное число).

x — код функциональной зависимости (1 или 2).

Используется функциональная зависимость

$$\ln w \cong -\gamma \ln R + c \equiv -\gamma \ln\left(\frac{N-r}{r}\right) + c\,.$$

Если $x = 1$, то применяется модифицированный В.П. Масловым закон Ципфа I: $N = n + 1$.

Если $x = 2$, то применяется модифицированный В.П. Масловым закон Ципфа II: $N = 2n + 1$.

Выходной файл в формате HTML

Выходной файл имеет имя `output.html`.

Выходной файл содержит таблицу с входными данными и входные параметры (см. рис. 1).

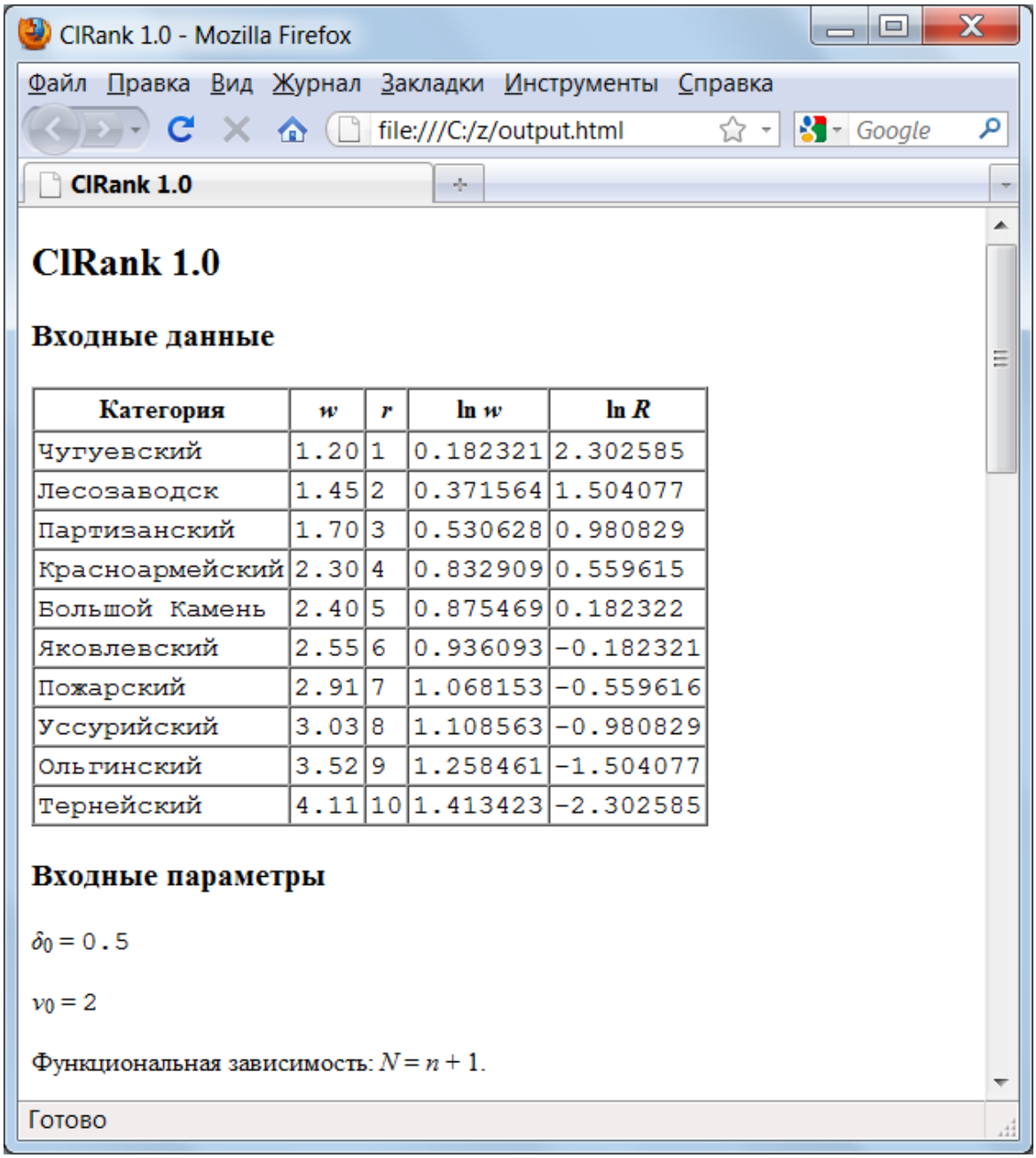

Рис. 1. Снимок экрана

Далее следуют три таблицы: таблица со значениями половины минимальной высоты полосы с ν_0 аномальных точек (значения величины $\delta_{\min}(\nu_0,\infty)$), таблица со значениями минимального количества аномальных точек (значения величины $\nu_{\min}(\delta_0,\infty)$) и таблица со значениями параметра γ (см. рис. 2–4).

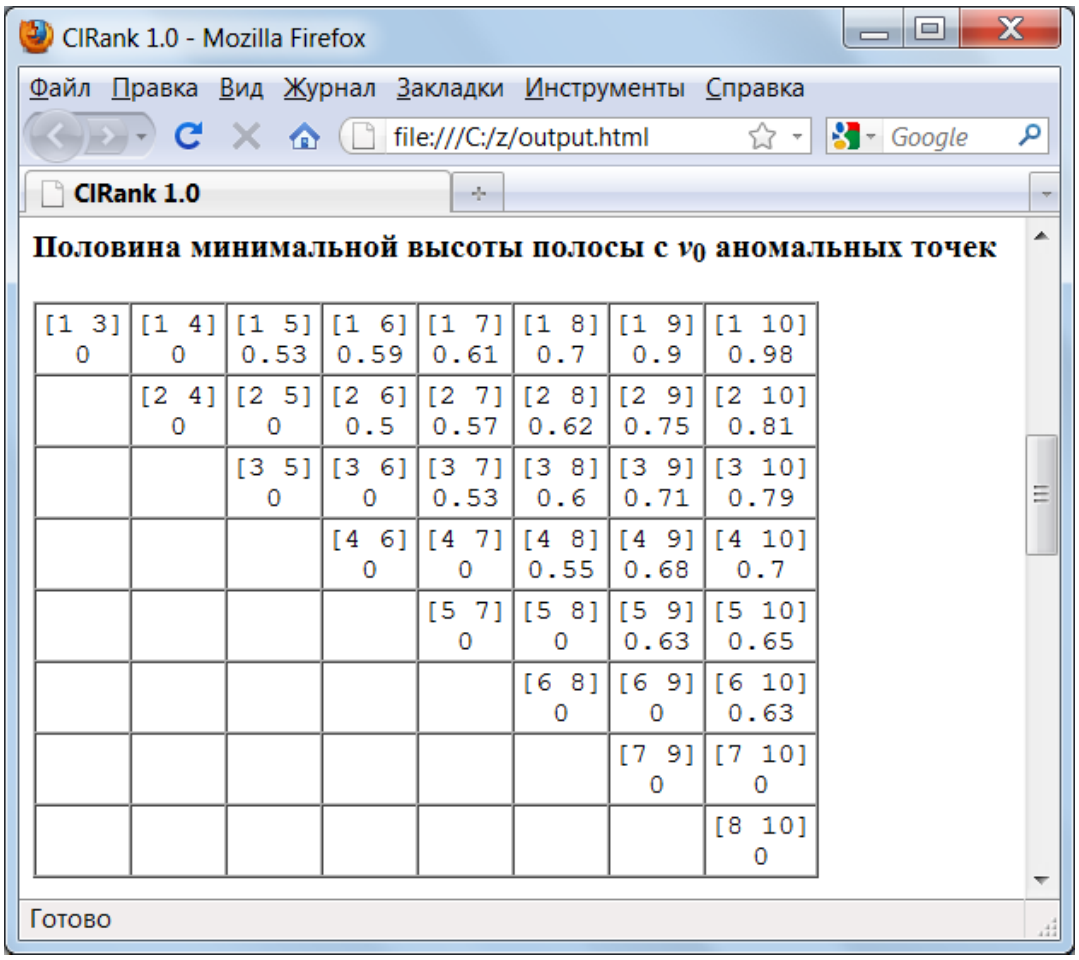

Рис. 2. Снимок экрана

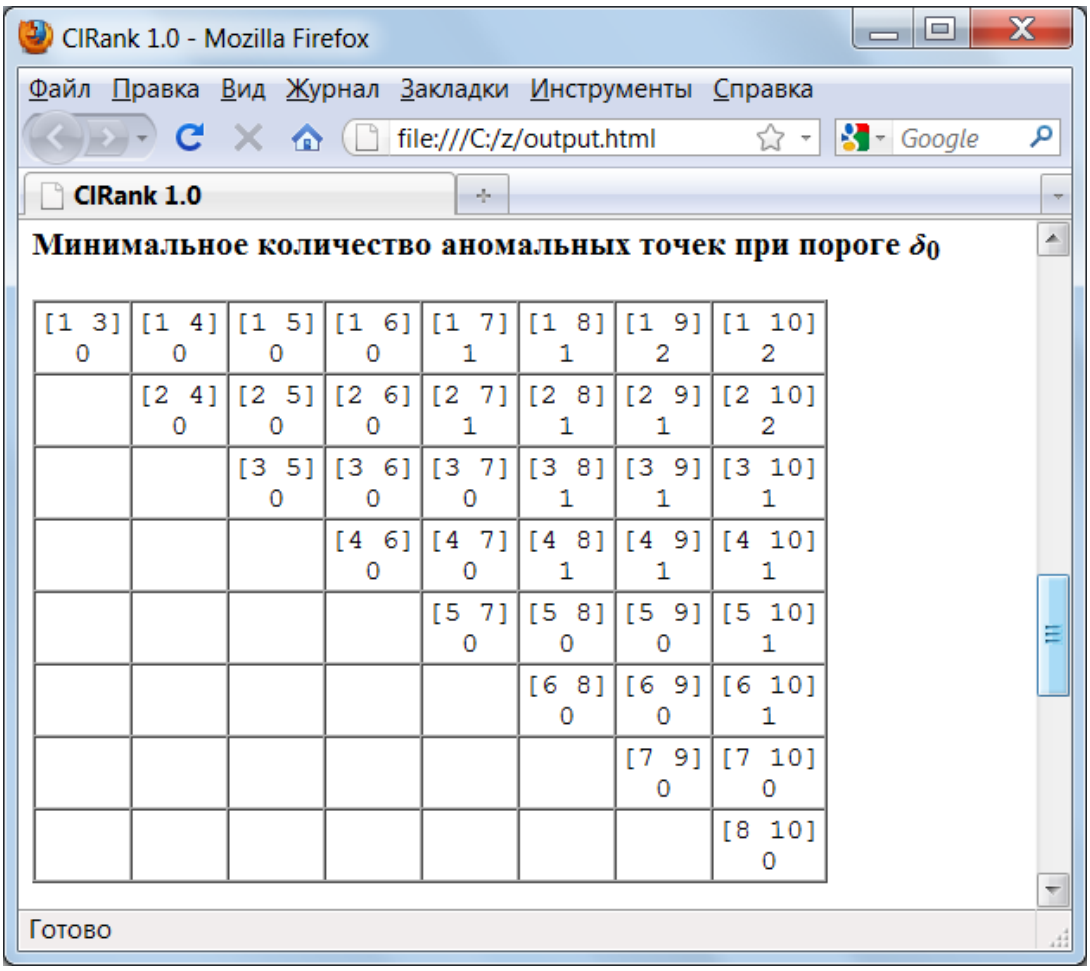

Рис. 3. Снимок экрана

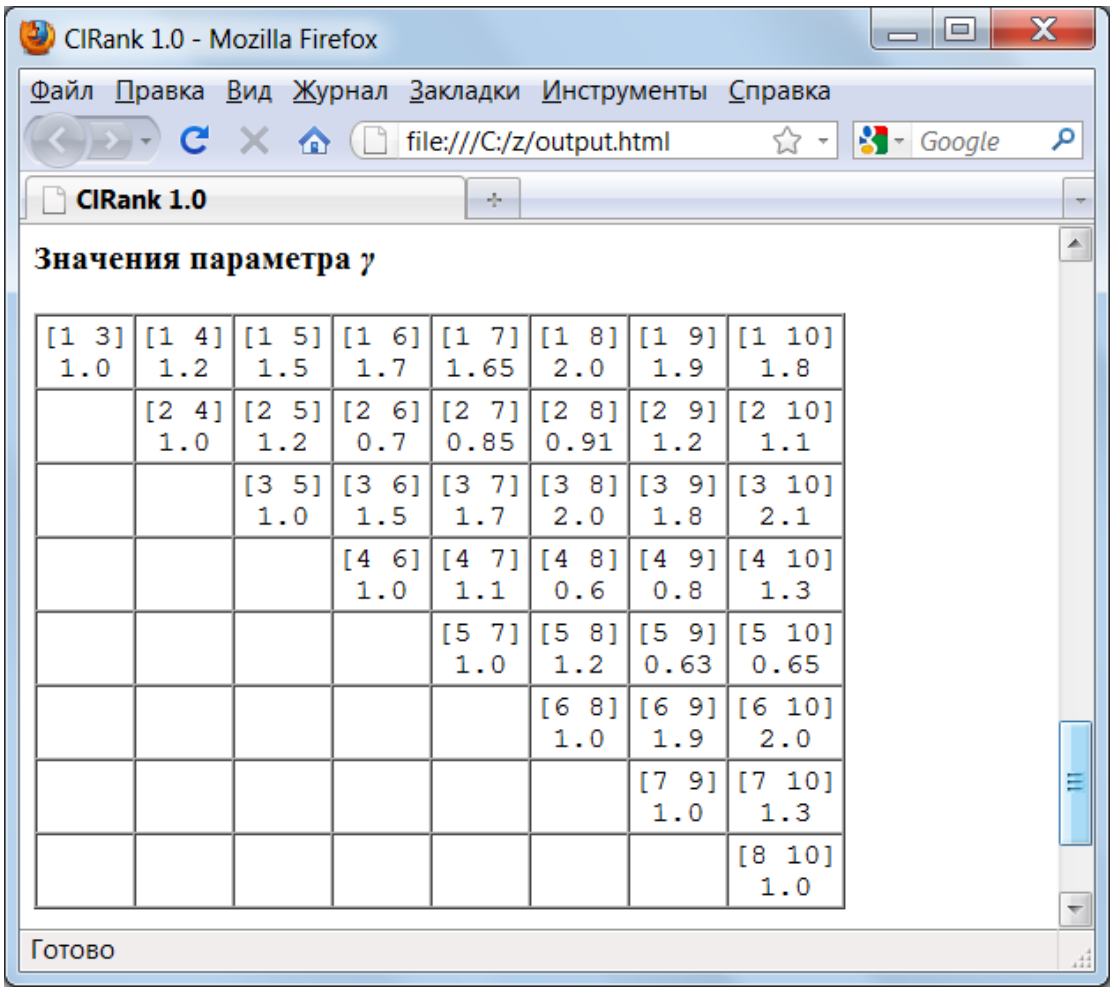

Рис. 4. Снимок экрана

Далее в выходном файле следует множество максимальных промежутков (см. рис. 5).

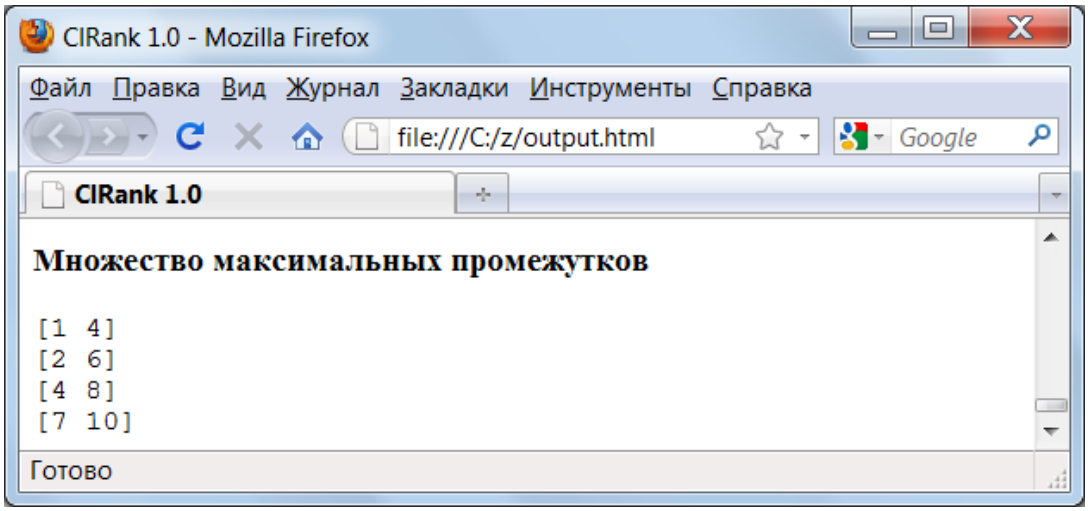

Рис. 5. Снимок экрана

2. Средства реализации

В качестве средства реализации был выбран язык программирования C++.

3. Модули

Модуль `main.cpp` отвечает за ввод входных параметров и вывод в HTML-файл.

Модули `csv.h` и `csv.cpp` отвечают за ввод входных данных из CSV-файла. Ввод входных данных осуществляет функция

```
InputData ReadInputData(std::ifstream&);
```

В модуле `types.h` описаны типы данных `InputData` и `Points`.

Модули `deltamin.h` и `deltamin.cpp` отвечают за вычисление $\delta_{\min}(\nu_0,\infty)$. Вычисление $\delta_{\min}(\nu_0,\infty)$ производит функция

```
double Deltamin(Points &points, int a, int b, int nu0, double &gamma);
```

Модули `numin.h` и `numin.cpp` отвечают за вычисление $\nu_{\min}(\delta_0,\infty)$. Вычисление $\nu_{\min}(\delta_0,\infty)$ производит функция

```
int Numin(Points &points, int a, int b, double delta0);
```

Модули `maxintervals.h` и `maxintervals.cpp` отвечают за вычисление множества максимальных промежутков. Вычисление множества максимальных промежутков производит функция

```
MaxIntervals MaxIntervalsSet(std::vector<std::vector<double> > &deltamin, double delta0, int nu0);
```

4. Реализация

```
double Deltamin(Points &points, int a, int b, int nu0, double &gamma)
//должны выполняться неравенства:
//  b-a+1 >= 3,
//  nu0 <= b-a+1
{
	if (!(b-a+1 >= 3))
		throw;
	if (!(nu0 <= b-a+1))
		throw;
	if (nu0 == b-a+1) {
		gamma = 1.0;
		return 0.0;
	}
	std::vector<Polyline> max_polylines = MaxPolylines(points, a, b, nu0);
	std::vector<Polyline> min_polylines = MinPolylines(points, a, b, nu0);
	double m;
	for (int nu = 0; nu <= nu0; ++nu) {
		double gamma1;
		double md = MinDifference(points, max_polylines[nu],
min_polylines[nu0-nu], gamma1);
		if (nu == 0 || md < m) {
			m = md;
			gamma = gamma1;
		}
	}
	return m/2;
}

std::vector<Polyline> MaxPolylines(Points &points, int a, int b, int nu0)
//должны выполняться неравенства:
//  b-a+1 >= 3,
//  nu0 < b-a+1
{
	if (!(b-a+1 >= 3))
		throw;
	if (!(nu0 < b-a+1))
		throw;
```

```
	std::vector<Polyline> k;
	k.resize(nu0+1);
	k[0].push_back(b);
	k[0].push_back(b-1);
	if (nu0 > 0) {
		k[1].push_back(b-1);
		k[1].push_back(b);
	}
	for (int q = b-2; q >= a; --q) {
		std::vector<int> old_tail, new_tail;
		for (int nu = 0; nu <= nu0; ++nu) {
			if (nu == b-q) {
				k[nu].push_back(q);
				for (int i = 0; i < old_tail.size(); ++i)
					k[nu].push_back(old_tail[i]);
				break;
			}
			else {
				int l = k[nu].size()-1;
				while (l > 0) {
					double gamma = gamma2(points[k[nu][l]],
points[k[nu][l-1]]);
					if (phi(points[k[nu][l]], gamma) >
phi(points[q], gamma))
						break;
					--l;
				}
				new_tail.clear();
				for (int i = l; i < k[nu].size(); ++i)
					new_tail.push_back(k[nu][i]);
				int s = k[nu].size();
				for (int i = l+1; i < s; ++i)
					k[nu].pop_back();
				k[nu].push_back(q);
				for (int i = 0; i < old_tail.size(); ++i)
					k[nu].push_back(old_tail[i]);
				old_tail = new_tail;
			}
		}
	}
	return k;
}

std::vector<Polyline> MinPolylines(Points &points, int a, int b, int nu0)
//должны выполняться неравенства:
//  b-a+1 >= 3,
//  nu0 < b-a+1
{
	if (!(b-a+1 >= 3))
		throw;
	if (!(nu0 < b-a+1))
		throw;
	std::vector<Polyline> k;
	k.resize(nu0+1);
	k[0].push_back(b);
	k[0].push_back(b-1);
	if (nu0 > 0) {
		k[1].push_back(b-1);
		k[1].push_back(b);
	}
	for (int q = b-2; q >= a; --q) {
		std::vector<int> old_tail, new_tail;
		for (int nu = 0; nu <= nu0; ++nu) {
			if (nu == b-q) {
```

```
                        k[nu].push_back(q);
                        for (int i = 0; i < old_tail.size(); ++i)
                              k[nu].push_back(old_tail[i]);
                        break;
                  }
                  else {
                        int l = k[nu].size()-1;
                        while (l > 0) {
                              double gamma = gamma2(points[k[nu][l]],
points[k[nu][l-1]]);
                              if (phi(points[k[nu][l]], gamma) <
phi(points[q], gamma))
                                    break;
                              --l;
                        }
                        new_tail.clear();
                        for (int i = l; i < k[nu].size(); ++i)
                              new_tail.push_back(k[nu][i]);
                        int s = k[nu].size();
                        for (int i = l+1; i < s; ++i)
                              k[nu].pop_back();
                        k[nu].push_back(q);
                        for (int i = 0; i < old_tail.size(); ++i)
                              k[nu].push_back(old_tail[i]);
                        old_tail = new_tail;
                  }
            }
      }
      return k;
}

      double gamma2(Point p1, Point p2)
      {
            return -(p1.y - p2.y)/(p1.x - p2.x);
      }

      double phi(Point p, double gamma)
      {
            return p.y + gamma*p.x;
      }

      double MinDifference(Points &points, Polyline &max_p, Polyline &min_p,
double &gamma)
      {
            double m;
            double g1, g2, g;
            int i = 0;
            int j = min_p.size()-1;
            g1 = gamma2(points[max_p[i]], points[max_p[i+1]]);
            g2 = gamma2(points[min_p[j]], points[min_p[j-1]]);
            g = g1 < g2 ? g1 : g2;
            m = phi(points[max_p[i]], g) - phi(points[min_p[j]], g);
            gamma = g;
            while (1) {
                  if (i == max_p.size()-1 && j == 0)
                        break;
                  if (i == max_p.size()-1) {
                        g = gamma2(points[min_p[j]], points[min_p[j-1]]);
                        --j;
                  }
                  else if (j == 0) {
                        g = gamma2(points[max_p[i]], points[max_p[i+1]]);
                        ++i;
                  }
```

```
                        else {
                              g1 = gamma2(points[max_p[i]], points[max_p[i+1]]);
                              g2 = gamma2(points[min_p[j]], points[min_p[j-1]]);
                              if (g1 < g2) {
                                    g = g1;
                                    ++i;
                              }
                              else {
                                    g = g2;
                                    --j;
                              }
                        }
                        double phi1 = phi(points[max_p[i]], g) -
phi(points[min_p[j]], g);
                        if (phi1 < m) {
                              m = phi1;
                              gamma = g;
                        }
                  }
                  return m;
            }

int Numin(Points &points, int a, int b, double delta0)
{
            int numin = b-a+1;
            for (int nu = 0; nu <= b-a+1; ++nu) {
                  double gamma1;
                  if (Deltamin(points, a, b, nu, gamma1) <= delta0) {
                        numin = nu;
                        break;
                  }
            }
            return numin;
}

MaxIntervals MaxIntervalsSet(std::vector<std::vector<double> > &deltamin,
double delta0, int nu0)
{
            MaxIntervals max_intervals;
            int n = deltamin.size();
            int k = 2 > nu0-1 ? 2 : nu0-1;
            for (int a = 0; a < n; ++a) {
                  for (int b = a+k; b < n; ++b) {
                        if (deltamin[a][b] <= delta0 &&
                              (a == 0 || deltamin[a-1][b] > delta0) &&
                              (b == n-1 || deltamin[a][b+1] > delta0))
                        {
                              Interval interval(a, b);
                              max_intervals.push_back(interval);
                        }
                  }
            }
            return max_intervals;
}
```

5. Тестирование

Для тестирования реализации обобщённого алгоритма Грэхема реализован другой алгоритм (менее эффективный) и произведено сравнение результатов, выдаваемых двумя реализациями.

```
struct PointOfIntersection {
            int i, j;
```

```
	double gamma;
};

class CmpPointsOfIntersection {
public:
	bool operator() (const PointOfIntersection &p1, const
PointOfIntersection &p2) const
	{
		return p1.gamma < p2.gamma;
	}
};

std::vector<Polyline> TestPolylines(Points &points, int a, int b)
{
	std::vector<Polyline> res;
	res.resize(b-a+1);
	std::vector<PointOfIntersection> pi;
	for (int i = a; i <= b; ++i) {
		for (int j = i+1; j <= b; ++j) {
			PointOfIntersection p;
			p.i = i;
			p.j = j;
			p.gamma = gamma2(points[i], points[j]);
			pi.push_back(p);
		}
	}
	sort(pi.begin(), pi.end(), CmpPointsOfIntersection());
	std::vector<int> order;
	for (int i = b; i >= a; --i)
		order.push_back(i);
	for (int i = 0; i < b-a+1; ++i)
		res[i].push_back(b-i);
	for (int i = 0; i < pi.size(); ++i) {
		for (int j = 0; j < b-a+1; ++j) {
			if (order[j] == pi[i].i) {
				order[j] = pi[i].j;
				res[j].push_back(pi[i].j);
			}
			else if (order[j] == pi[i].j) {
				order[j] = pi[i].i;
				res[j].push_back(pi[i].i);
			}
		}
	}
	return res;
}
```

6. Примеры

Приведём примеры применения программной системы для кластеризации данных из статьи М.А. Гузева и Е.В. Черныш «Ранговый анализ в задачах кластеризации» (Информатика и системы управления. — 2009. — №3(21). — С. 13–19).

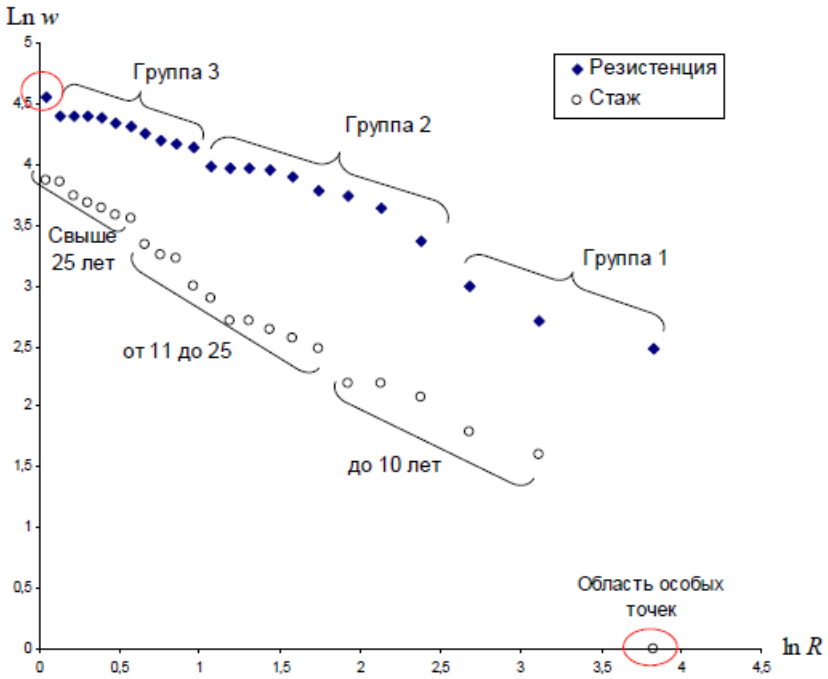

Рис. 1. Выраженность фазы «резистенции» *синдрома эмоционального выгорания* медицинских сестер по стажу работы

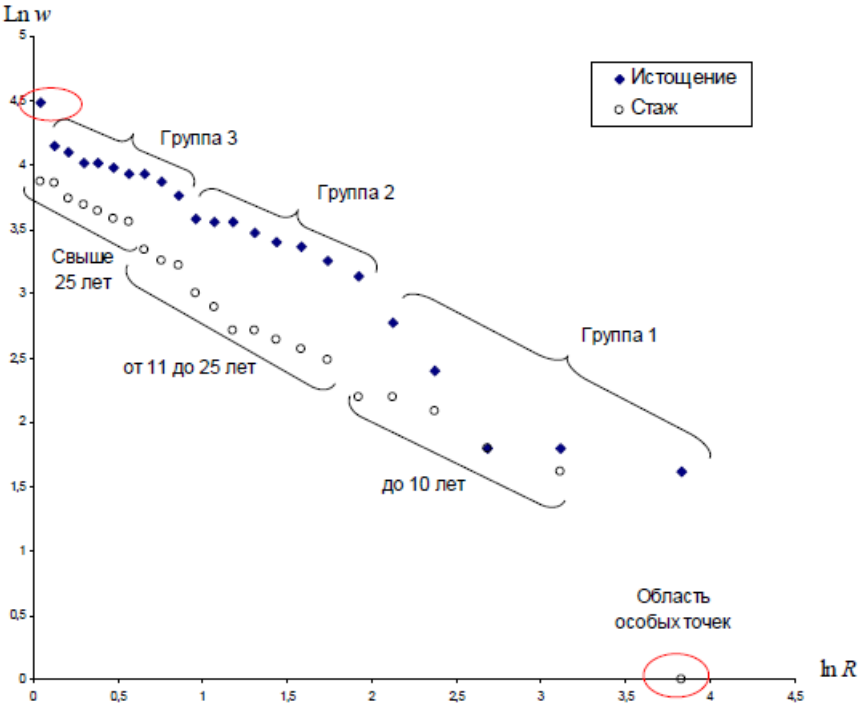

Рис. 2. Выраженность фазы «истощения» *синдрома эмоционального выгорания* медицинских сестер по стажу работы

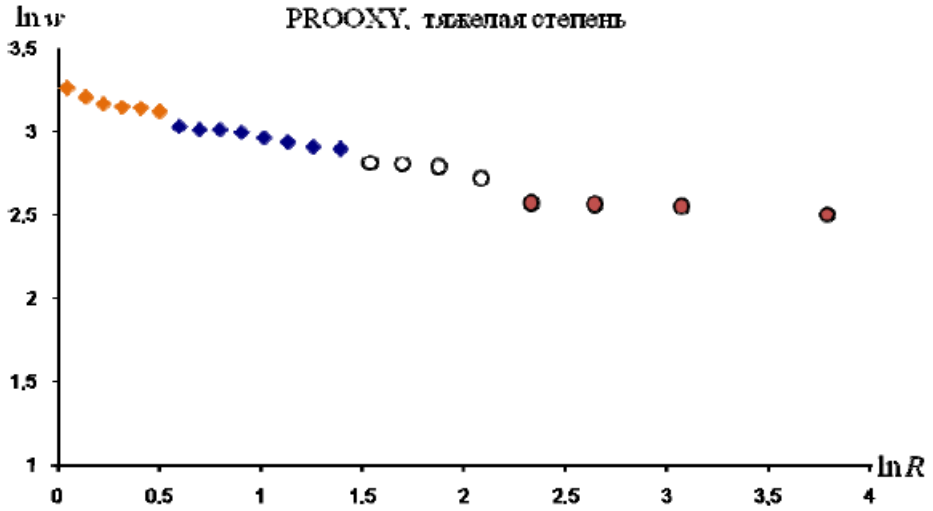

Рис. 3. Кластеризация «тяжелых» пациентов по результатам анализов крови.

Результаты, выданные программной системой, представлены в файлах `output1.html`, `output2.html`, `output3.html`.