Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления» (ИУ)

КАФЕДРА «Информационная безопасность» (ИУ8)

Отчёт

по лабораторной работе № 5
по дисциплине «Теория систем и системный анализ»

Тема: «Двумерный поиск для подбора коэффициентов простейшей нейронной сети на примере решения задачи линейной регрессии экспериментальных данных»

Вариант 12

Выполнил:
Николаева Е.Д., студент
группы ИУ8-31

Проверил: Коннова Н.С.,
доцент каф. ИУ8

г. Москва,
2020 г.

1. Цель работы

Знакомство с простейшей нейронной сетью и реализация алгоритма поиска ее весовых коэффициентов на примере решения задачи регрессии экспериментальных данных.

2. Условие задачи

Вариант № 12.

В зависимости от варианта работы (табл. 1) найти линейную регрессию функции $y(x)$ (коэффициенты наиболее подходящей прямой c, d) по набору ее N дискретных значений, заданных равномерно на интервале $[a,b]$ со случайными ошибками $e_i = A\,\mathrm{rnd}(-0.5; 0.5)$. Выполнить расчет параметров c, d градиентным методом. Провести двумерный пассивный поиск оптимальных весовых коэффициентов нейронной сети (НС) регрессии.

w1 = -1, w0 = 3, a = 0, b = 3, N = 10, A = 3. Алгоритм поиска c - золотое сечение, алгоритм поиска d — метод Фибоначчи.

3. Графики

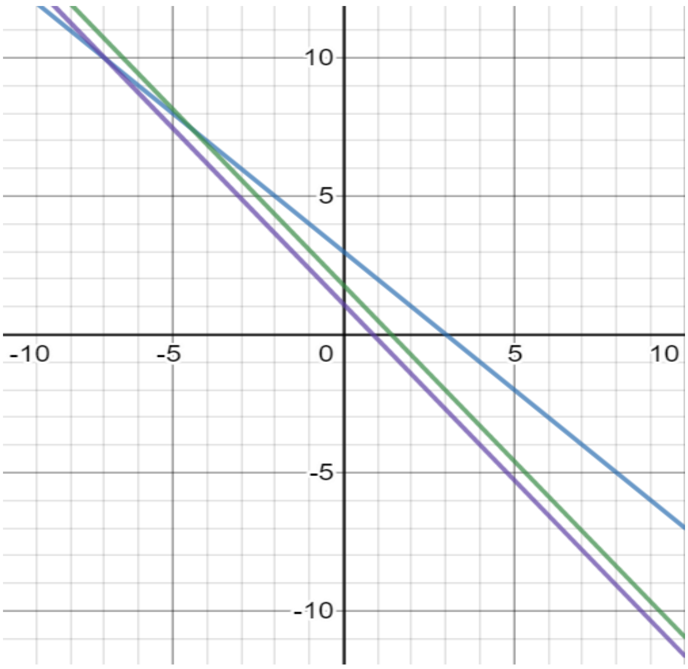

Рисунок 1. Графики, постороенные по результатам работы программы

Зеленый график – y=-x+3, синий график построен при шуме A=0, фиолетовый график построен при шуме A=3.

4. Результат работы программы

```
C:\Users\Edn\CLionProjects\untitled5\cmake-build-debug\untitled5.exe
Function y = -x + 3
Without noise
Cmin = 1.82215e-307
Cmax = 1.27344e+231
Dmin = 1.7891e-307
Dmax = 1.09974e-303
w1 = -1.27344e+231
w0 = 1.7891e-307
With noise A = 3
Cmin = 1.82218e-307
Cmax = 1.27344e+231
Dmin = 1.09974e-303
Dmax = 1.09974e-303
w1 = -1.27344e+231
w0 = 1.09974e-303
```

5. Выводы

В результате работы был реализован алгоритм поиска весовых коэффициентов функции на примере решения задачи регрессии экспериментальных данных.

6. Ответ на контрольный вопрос

1. Поясните суть метода наименьших квадратов.

Задача заключается в нахождении коэффициентов линейной зависимости, при

которых функция двух переменных *a* и *b*

$$E^2(w_1, w_0) = \sum_{i=1}^{N} \left[y(x_i) - t_i \right]^2 \to \min_{c,d}$$

принимает наименьшее значение. То есть, при данных *a* и *b* сумма квадратов отклонений экспериментальных данных от найденной прямой будет наименьшей. В этом вся суть метода наименьших квадратов. Таким образом, решение сводится к нахождению экстремума функции двух переменных.

Приложение 1. Исходный код программы «Задача 1»

```
#include <iostream>
#include <random>
#include <ctime>
#include <vector>
#include <algorithm>

const double a = 0.0;
const double b = 3.0;
const double c = -1.0;
const double d = 3.0;
const size_t N = 10;
const double A = 3.0;
const double step = (double)((b - a) / (N - 1));

struct Point {
   double x;
   double y;
};

double func (const double &x) {
   return c * x + d;
}

std::vector<Point> random(const size_t N, const double noise) {
   std::vector<Point> points (N);
   std::random_device rd;
   std::mt19937 gen(rd());
   std::uniform_real_distribution<double> error(-0.5, 0.5);
   for (size_t i = 0; i < N; ++i) {
      points[i].x = a + i * step;
      points[i].y = func( a + i * step) + noise * error(gen);
   }
   return points;
}

std::vector<Point> edge(const double lower, const double upper,
                        const size_t num, const double noise) {
   std::vector<Point> points(num);
   const double step = (upper - lower) / static_cast<double>(num - 1);
   for (size_t i = 0; i < num; ++i) {
      points[i].x = lower + i * step;
      points[i].y = func(points[i].x) -noise/2 + rand() * 1./RAND_MAX * (noise);
   }
   return points;
}

double error(const std::vector<Point>& right, const double c, const double d) {
   double sum = 0.;
   for (auto point : right) {
      sum += pow(point.y - (c * func(point.x)), 2);
   }
```

```
    return sum;
}

double golden_ratio(std::vector<Point>& p, double Cmin, double Cmax) {
    double l = std::abs(Cmax - Cmin);
    std::swap(Cmin, Cmax);
    Cmin = std::fabs(Cmin);
    Cmax = std::fabs(Cmax);
    const double e = 0.1;
    const double t = (std::sqrt(5) + 1) / 2;
    double c_k1 = Cmin + (1 - 1/t)*Cmax;
    double c_k2 = Cmin + Cmax / t;
    double f_k1 = func(-c_k1);
    double f_k2 = func(-c_k2);
    while (l > e){
        if (f_k1 < f_k2){
            Cmax = c_k2;
            c_k2 = Cmin + Cmax - c_k1;
            f_k2 = func(-c_k2);
        } else {
            Cmin = c_k1;
            c_k1 = Cmin + Cmax - c_k2;
            f_k1 = func(-c_k1);
        }
        if (c_k1 > c_k2){
            std::swap(c_k1, c_k2);
            std::swap(f_k1, f_k2);
        }
        l = std::abs(Cmax - Cmin);
    }
    return -((Cmax + Cmin) / 2);
}

int F(int f)
{
    if (f==1) return 1;
    else if (f==2) return 1;
    else if (f>2)
        return (F(f-1)+F(f-2));
}

double Fibonacci( std::vector<Point>& p, double Dmin, double Dmax) {
    double ak = Dmin, bk = Dmax, x1, x2, y1, y2;
    int G = 10;
    x1 = ak + (double)F(G - 2) / F(G) * (bk - ak);
    x2 = ak + (double)F(G - 1) / F(G) * (bk - ak);
        y1 = error(p, x1, 0);
        y2 = error(p, x2, 0);
    for (int i=G; i >= 1; --i) {
        if (y1 > y2) {
            ak = x1;
            x1 = x2;
            x2 = bk - (x2 - ak);
```

```
            y1 = y2;
            y2 = error(p, x2, 0);
        }
        else {
            bk = x2;
            x2 = x1;
            x1 = ak + (bk - x2);
            y2 = y1;
            y1 = error(p, x1, 0);
        }
    }
    return (x1 + x2) / 2;
}


void print(const double noise)
{
    std::vector<Point> p = random(N, noise);
    double Cmin, Cmax, Dmin, Dmax;
    edge( Cmin, Cmax, Dmin, Dmax);
    std::cout << "Cmin = " << Cmin << "\nCmax = " << Cmax << "\nDmin = "
              << Dmin << "\nDmax = " << Dmax << std::endl;
    double w1 = golden_ratio(p, Cmin, Cmax);
    double w0 = Fibonacci(p, Dmin, Dmax);
    std::cout << "w1 = " << w1 <<std::endl;
    std::cout<< "w0 = " << w0 << std::endl;
}

int main() {
    std::cout << "Function y = -x + 3"<<std::endl;
    std::cout << "Without noise"<<std::endl;
    print(0.0);
    std::cout << "With noise A = " << A <<std::endl;
    print(A);
    return 0;
}
```