**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 «Методы оптимизации»

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

Вариант 15

Выполнил: Петросян А.Р.,
студент группы ИУ8-32

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. Постановка задачи

Необходимо найти линейную регрессию функции (коэффициенты наиболее подходящей прямой) по набору ее дискретных значений, заданных равномерно на интервале со случайными ошибками . Выполнить расчет параметров градиентным методом. Провести двумерный пассивный поиск оптимальных весовых коэффициентов нейронной сети (НС) регрессии.

Вариант 15

Рассмотрим линейную регрессию функции c*x+d на интервале a=-5, b=0 с известными параметрами c= 0,1 , d = 2

Набор данных содержит N=32 отсчета.

со случайными ошибками e = A * random(-0.5..0.5), A=0,1.

Необходимо выполнить расчет параметров градиентным методом и Провести двумерный пассивный поиск оптимальных весовых коэффициентов нейронной сети (НС) регрессии.

## 3. Ход работы

Рисунок 1 демонстрирует интерфейс программы.

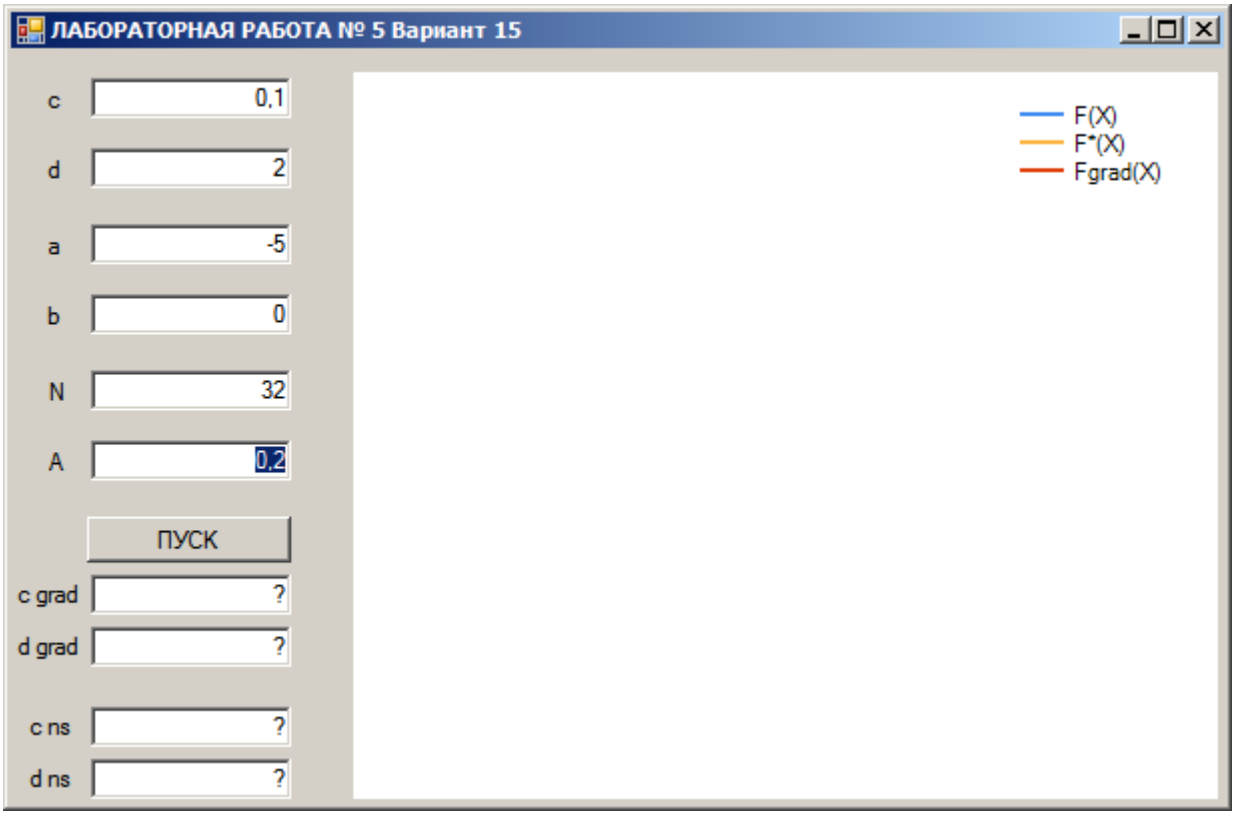


**Рисунок 1** – Интерфейс программы

Программа позволяет ввести значения коэффициентов “c” и “d” начальной функции, а также интервал (a,b) и величину вносимой ошибки «A». Также есть возможность задать количество узлов сетки N.

При вводе программа проверяет  допустимость входных данных. При вводе неверных данных поле ввода с ними подсвечивается красным цветом.

Дробная часть N, если пользователь ее ввел, отбрасывается.

Программа проверяет, что N больше 1. Количество узлов, меньшее 2, считается ошибкой.

Если a оказывается больше b,  то они меняются местами. Программа не считает это за ошибку.

После нажатия кнопки ПУСК происходит расчет коэффициентов линейной регрессии двумя методами – градиентным поиском и прямым двумерным пассивным поиском.

Рисунок 2 демонстрирует результат работы программы.

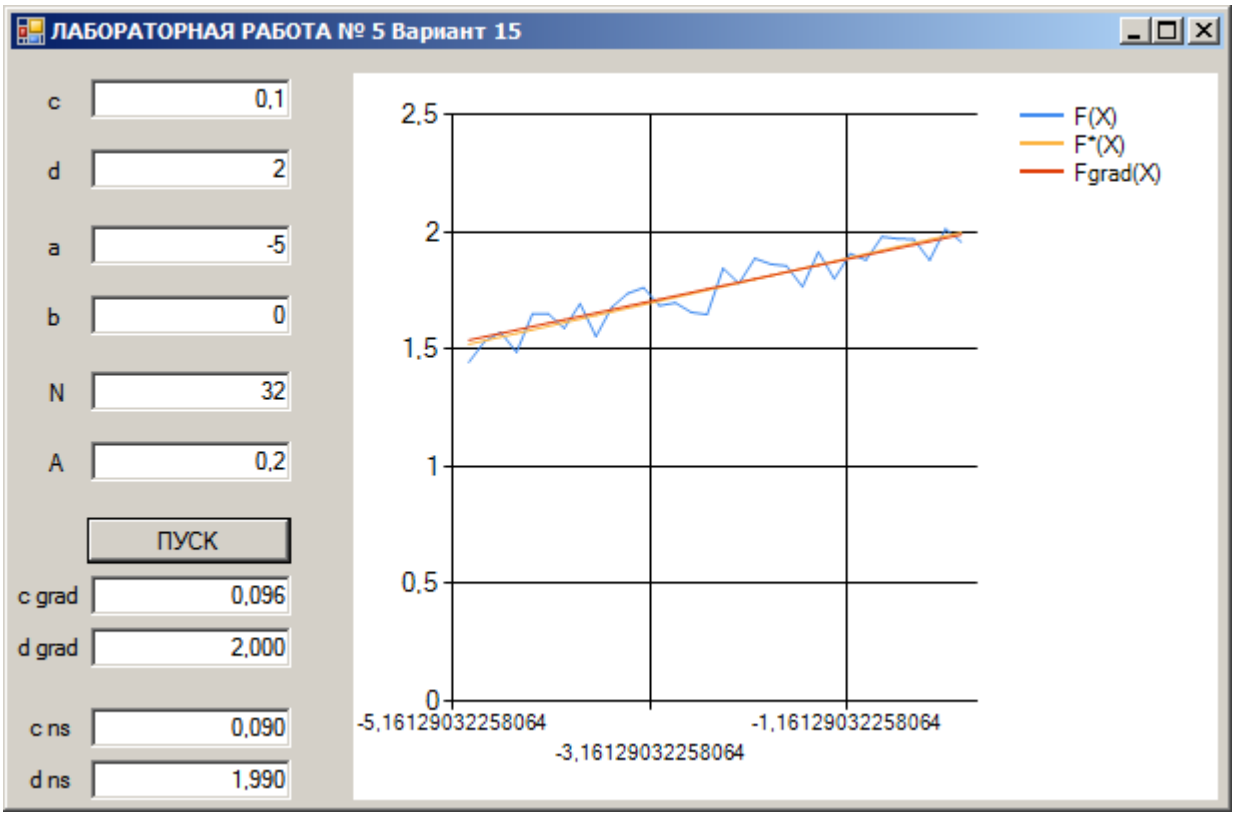


**Рисунок 1** – Результат работы программы

Найденные коэффициенты выводятся в поля внизу формы.

Исходная функция F(X) и найденные F*(X) и F grad(X) выводятся на график.

Для реализации градиентного метода используется алгоритм «градиентный спуск с постоянным шагом». Минимизируется функция «сумма квадратов отклонений на заданной сетке» (MNS). В текущей точке численно вычисляются частные производные по обеим координатам, и делается шаг фиксированной длины по направлению вектора градиента. Если в результате выполнения этого шага получается значение отклонения больше, чем было в исходной точке – считается, что результат найден.

## 4. Выводы

При реализации настройки нейронной сети в результате оптимизации (с учетом, что НС состоит из одного нейрона) описание реакции нейрона сделано непосредственно при выполнении двумерного поиска. Итоговый алгоритм: задается сетка для поиска с шагами hc и hd по каждой из координат. Для каждого узла сетки по первой координате выполняется поиск минимального значения MNS по каждому узлу сетки по второй координате. Выбирается наилучшее (наименьшее). После этого перестраивается сетка так, чтобы найденное решение было в ее центре, а шаг поиска уменьшается. Если шаг поиска становится меньше наперед заданного, то решение считается найденным.

# Приложение 1. Исходный код программы

```
#pragma endregion

		//Делегат для передачи функции
		delegate double function(double  x);
		//Не модулированная функция
		double F0(double x)
		{
			return x*x*Math::Sin(x);
		}
		//Модулированная
		double F(double x)
		{
			return F0(x) * Math::Sin(5*x);
		}

		double A, B; //Интервал
		ref struct Log //Единичная запись в журнале
		{
			double T; //Температура
			double X; //Аргумент
			double F; //Функция
		};
		List<Log^>^ list; //Журнал операций


	private: System::Void MyForm_Load(System::Object^  sender, System::EventArgs^  e) {
		list = gcnew List<Log^>();
		button_Click(sender, e); //Вызвать расчет для данных по умолчанию
	}


	private: System::Void button_Click(System::Object^  sender, System::EventArgs^  e) {
		//Прочитать данные
		if (!ReadData()) return; //В данных - ошибки
		//Заполнить график
		ShowInChart();
		//Выполнить расчет
		//function^ f = gcnew function(this,&MyForm::F0);
		double X = FindExtremum(gcnew function(this, &MyForm::F0));
		labelMinF0->Text = X.ToString("0.000");
		SaveToFile("F0.txt");
		X = FindExtremum(gcnew function(this, &MyForm::F));
		labelMinF->Text = X.ToString("0.000");
		//Вывести результат в файл
		SaveToFile("F.txt");
	}

			//Прочитать одно значение
			// ^ = ссылка на экземпляр класса
			// % = передача параметра по ссылке (может быть изменен в функции)
			bool ReadTextBox(TextBox^ textbox, double %value)
			{
				textbox->BackColor = Color::Red;
				if (!double::TryParse(textbox->Text, value)) return false;
				textbox->BackColor = Color::White;
				return true;
			}
			//Прочитать исходные данные
			bool ReadData()
			{
				bool result = true;
```

```
		result &= ReadTextBox(textBoxA, A);
		result &= ReadTextBox(textBoxB, B);
		if (A > B)
		{
			MessageBox::Show("A>B");
			return false;
		}
		return result;
}

void ShowInChart()
{
		int N = 1000; //Точек на графике
		double h = (B - A) / (N - 1);
		chart->Series[0]->Points->Clear();
		chart->Series[1]->Points->Clear();
		chart->ChartAreas[0]->AxisX->Minimum = A;
		chart->ChartAreas[0]->AxisX->Maximum = B;

		for (int i = 0; i <= N; i++)
		{
			double x = A + h*i;
			chart->Series[0]->Points->AddXY(x, F0(x));
			chart->Series[1]->Points->AddXY(x, F(x));
		}
}

double TMax = 10; //Начальная "температура"
double TMin = 1E-6; //Конечная температура
Random^ random = gcnew Random(); //Датчик случайных чисел

double FindExtremum(function^ F)
{
		double T = TMax; //Номер итерации
		double Alpha = 0.975; //Коэффициент уменьшения
		double X = (B + A) / 2; //Начальное значение приближения
		double f1 = F(X); //Начальное значение функции
		bool NeedToLog = true;
		while (T > TMin)
		{
			if (NeedToLog)
			{
				//Записывается в журнал
				Log^ log = gcnew Log();
				log->T = T;
				log->X = X;
				log->F = f1;
				list->Add(log);
				NeedToLog = false;
			}
			//Выбирается пробная точка
			double xs = random->NextDouble()*(B - A) + A;

			//Улучшение алгоритма - уменьшение величины шага с понижением
температуры
			//double xs = X + (2*random->NextDouble()-1) * (B - A)*(T / TMax);
			//if (xs<A || xs>B) continue;

			T = T * Alpha;//Следующая итерация
			double f = f1; //Значение функции на предыдущем шаге

			f1 = F(xs);
			//Разница
			double df = f1 - f;
```

```
                                        if (df <= 0)  //Если разница отрицательная - безусловный переход в новую
точку
                                        {
                                                X = xs;
                                                NeedToLog = true;
                                                continue; //и продолжить выполнение цикла
                                        }
                                        //Если разница положительная
                                        double P = Math::Exp(-df / T); //Вероятность перехода
                                        if (random->NextDouble() <= P)
                                        {
                                                X = xs; //Переход
                                                NeedToLog = true;
                                        }
                                        else
                                                f1 = f; //Иначе - остаемся на месте
                                }
                                return X;
                        }

                        //Сохранить журнал в файл, очистить
                        void SaveToFile(String^ filename)
                        {
                                StreamWriter^ writer = gcnew StreamWriter(filename);
                                writer->WriteLine("N\tT\tx\tf(x)\t"); //Заголовок
                                for (int k = 0; k < list->Count; k++)
                                {
                                        Log^ log = list[k];
                                        writer->WriteLine(
                                                (k+1).ToString()+"\t"+
                                                log->T.ToString("0.00")+ "\t" +
                                                log->X.ToString("0.00000") + "\t" +
                                                log->F.ToString("0.00000") + "\t"
                                                );
                                }

                                writer->Close();
                                list->Clear();
                        }

};
```