Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт информационных технологий и прикладной математики

Кафедра 806 «Вычислительная математика и программирование»

Курсовой проект

по курсу “ Объектно-ориентированное программирование”

Работу выполнил: Сыроежкин К.Г.

Группа: М8О-204Б-18

Руководитель: Кузнецова С.В

Оценка: _____

Подпись: _____

Дата: _____

Москва 2019 г.

Постановка задачи

Вариант 39. Разработать windows-приложение на языке программирования c#, наглядно демонстрирующее работу алгоритма оптимизации роя частиц (Particle Swarm Optimization).

Описание алгоритма

Метод оптимизации роя частиц(PSO) – это метод стохастической оптимизации, несколько похожий на эволюционные алгоритмы, однако, сильно отличающийся. Системы роевого интеллекта представляют собой модели разнообразных типов коллективного поведения в колониях различных живых организмов — стаях птиц, косяках рыб, колониях муравьев и т. п. Отличительной особенностью таких моделей является то, что каждый отдельный организм в этой модели выполняет очень простые действия, подчиненные своей локальной цели, взаимодействуя с ограниченным числом других организмов в рассматриваемой колонии. Иначе говоря, PSO поддерживает единую статическую популяцию, члены которой изменяются в ответ на новые изменения системы. Метод роя частиц предназначен для решения задач многомерной непрерывной оптимизации и основан на моделировании социального поведения колоний животных, выполняющих коллективный поиск мест с наилучшими условиями существования. Этот метод по существу

является формой направленной мутации (directed mutation). PSO был разработан Джеймсом Кеннеди и Расселом Эберхартом в 1995 году.

Алгоритм. Пусть задана функция $f : R^n \to R$, требуется найти глобальный максимум этой функции, т. е. точку x_0 такую, $f(x_0) \geq f(x)$ для любого $x \in R^n$. Суть подхода, на котором основан метод роя частиц, заключается в том, что глобальный максимум функции f ищется с помощью системы (роя), состоящей из m частиц. Частицы выполняют поиск, перемещаясь по пространству решений R^n. Положение i-ой частицы задается вектором $x_i \in R^n$, значение $f(x_i)$ определяет функцию качества этой частицы в текущий момент времени. Каждая частица в рое обладает своей собственной скоростью $v_i \in R^n$, которая определяет как изменяются координаты частицы со временем:

$$x_i \leftarrow x_i + \tau v_i$$

где τ — некоторая единица измерения скорости (продолжительность одного такта работы алгоритма, например, можно положить $\tau = 1$). Последовательность (дискретная) положений i-ой частицы будем называть ее траекторией.

Ключевая особенность метода роя частиц заключается в способе обновления скорости отдельных частиц, которое выполняется по формуле:

$$v_i \leftarrow \alpha v_i + b(p_i - x_i) + c(g - x_i) + d(k_i - x_i)$$

Первое слагаемое в этой формуле представляет собой инерцию частицы — ее скорость на следующем временном

шаге получается изменением текущей скорости. Коэффициент α является показателем того, насколько сохраняется скорость.

Вектор p_i, фигурирующий во втором слагаемом, служит простейшей моделью индивидуальной памяти — он равен лучшей точке траектории i-ой частицы за все время ее существования (от начала работы алгоритма до текущего момента времени). Говорят, что второе слагаемое реализует принцип простой ностальгии — каждая частица «хочет» вернуться в ту точку, где ею было достигнуто лучшее значение функции f. Коэффициент b характеризует насколько сильно точка будет стремиться к своему личному лучшему значению.

Вектор g, участвующий в вычислении третьего слагаемого, представляет собой лучшую точку, обнаруженной за время своего существования всем роем, т. е. представляет собою некую коллективную память. Следовательно, само третье слагаемое определяет некоторую простую схему социального взаимодействия между отдельными частицами роя. При большом коэффициенте c точки будут иметь тенденцию больше стремиться к глобальному рекорду, и тогда алгоритм становится больше похож на hill-climbers.

Вектор k представляет собой наиболее лучшее положение, которое нашли соседи (информаторы). Во многих современных реализациях этого алгоритма соседи назначаются случайным образом или слагаемое обнуляется. Коэффициент d является чем-то средним между c и b.

Коэффициенты при слагаемых могут выбираться из разных соображений. Численные эксперименты показали, что

лучшей наиболее простой и универсальной является вероятностная схема, где коэффициенты выбираются случайным образом из диапазона [0, 1].

Реализация на псевдокоде.

Давайте, для удобства, введем несколько обозначений:

Best – лучшее положение;

x_{random} – случайно сгенерированное положение частицы;

Случайное распределение частиц по пространству решений

Случайная инициализация скоростей частиц

$Best \leftarrow x_{random}$

while не найден максимум (или другое условие):

for each (считываем очередную точку x_i из роя)

if $f(x_i) > f(Best)$

$Best = x_i$

for each (считываем очередную точку x_i из роя)

$p_i \leftarrow$ собственное лучшее положение

$g \leftarrow$ лучшее положение системы $k_i \leftarrow$ лучшее положен

for each (размерность измерения)

$\alpha \leftarrow random(0, \alpha)$

$b \leftarrow random(0, b)$

$c \leftarrow random(0, c)$

$d \leftarrow random(0, d)$

$v_i \leftarrow \alpha v_i + b(p_i - x_i) + c(g - x_i) + d(k_i - x_i)$

for each (считываем очередную точку x_i из роя)

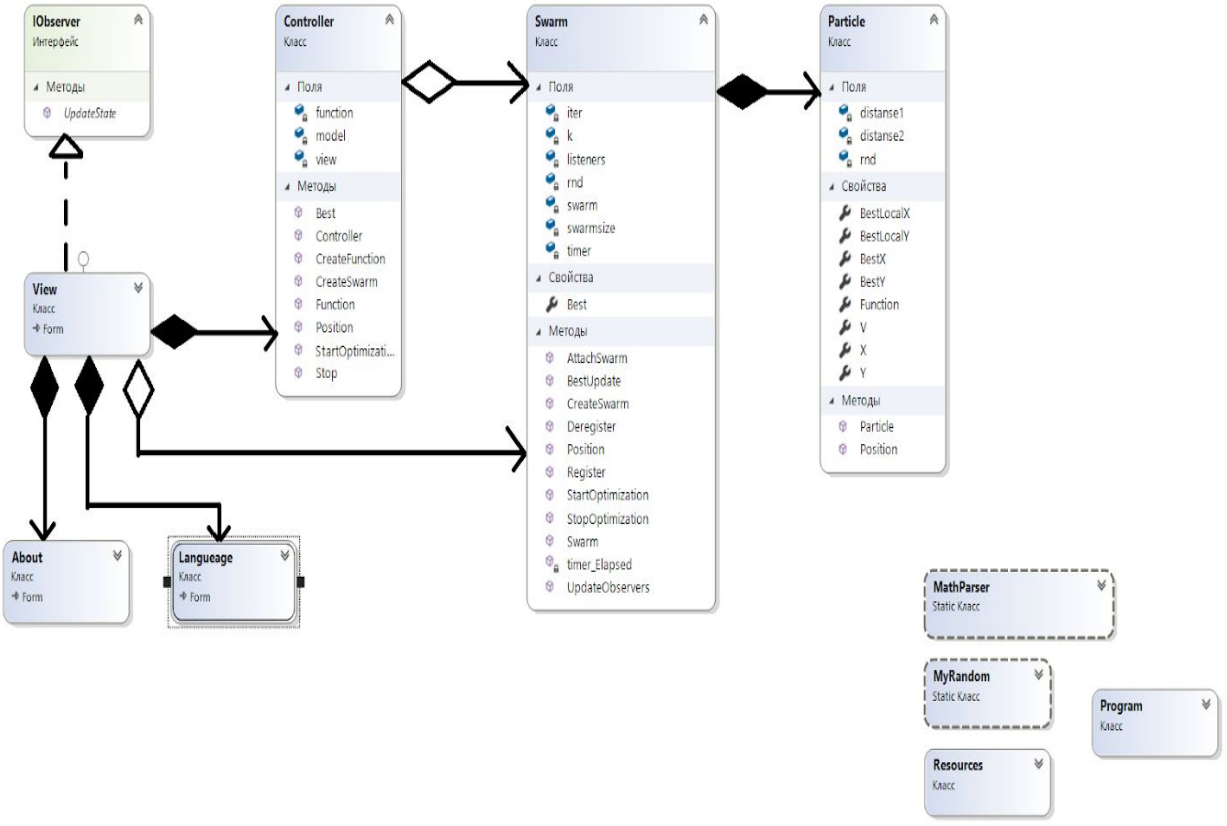
$x_i \leftarrow x_i + \tau v_i$

Реализация

Моя реализация данного алгоритма представляет из себя 2 класса: Particle и Swarm.
Particle – класс частицы. Частица хранит в себе свое положение, лучший собственный результат и лучший результат системы. Этот класс нужен удобного хранения и обновления данных.

Swarm – класс роя. Он является основной моделью и именно в нем реализован алгоритм. В качестве он использует объекты класса Particle.

Диаграмма классов



Реализация алгоритма на c#

Класс частицы:

```
    public class Particle
    {
        public Func<double, double> Function { get; set; }
        private Random rnd;
        private double distanse1, distanse2;

        public Particle(Func<double, double> function, double distanse1, double
distanse2, Random rnd)
        {
            this.distanse1 = distanse1;
            this.distanse2 = distanse2;
            this.rnd = rnd;
            X = rnd.NextDouble(distanse1, distanse2);
            BestLocalX = X;
            V = rnd.NextDouble();
            this.Function = function;
        }
        /// <summary>
        /// Лучший x, который нашел рой
```

```
        /// </summary>
        public double BestX { get; set; }

        /// <summary>
        /// X в настоящий момент
        /// </summary>
        public double X { get; set; }

        /// <summary>
        /// Лучший X, который находила эта точка
        /// </summary>
        public double BestLocalX { get; set; }

        /// <summary>
        /// Скорость
        /// </summary>
        public double V { get; set; }

        /// <summary>
        /// Функция от X
        /// </summary>
        public double Y
        {
            get
            {
                return Function(X);
            }
        }

        /// <summary>
        /// Функция от лучшего локального X
        /// </summary>
        public double BestLocalY
        {
            get
            {
                return Function(BestLocalX);
            }

        }
        /// <summary>
        /// Функция от лучшего X
        /// </summary>
        public double BestY
        {
            get
            {
                return Function(BestX);
            }
        }

        /// <summary>
        /// Позиция точки
        /// </summary>
        /// <returns></returns>
        public double[] Position()
        {
            double[] pos = new double[2];
            pos[0] = X;
            pos[1] = Function(X);
            return pos;
        }

    }
```

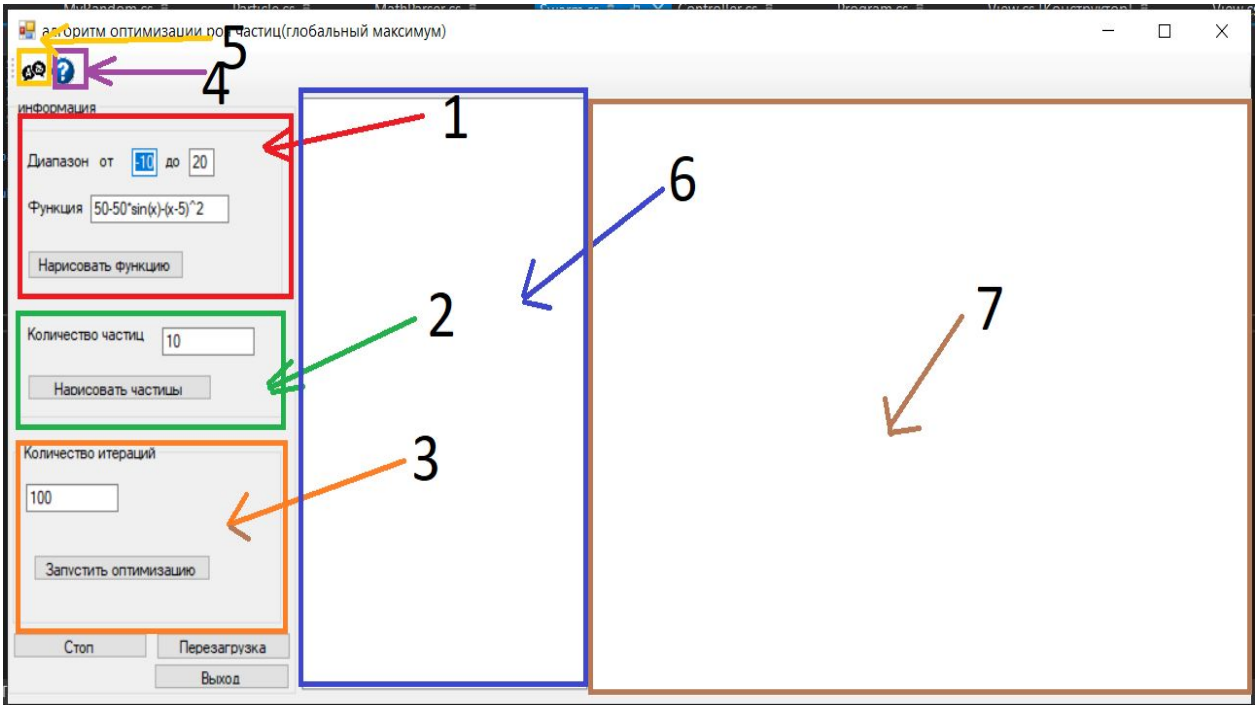
Класс роя

```
    public class Swarm
    {
        private Particle[] swarm;
        private Random rnd;
        private ArrayList listeners = new ArrayList();
        private Timer timer;
        private int k = 0;
        private int iter;
        private int swarmsize;

        public Swarm()
        {
            timer = new Timer();
            timer.Enabled = false;
            timer.AutoReset = true;
            timer.Elapsed += new ElapsedEventHandler(timer_Elapsed);
            timer.Interval = 100;
        }
        public double Best
        {
            get
            {
                return swarm[0].BestX;
            }
        }


           /// <summary>
        /// Обновление памяти лучшей позиции системы у частицы
        /// </summary>
        ///

        public void BestUpdate(double value)
        {
            for (int i = 0; i < swarmsize; i++)
                swarm[i].BestX = value;
        }

        /// <summary>
        /// Создает рой и привязывает его к переменной
        /// </summary>
        /// <param name="swarmsize"></param>
        /// <param name="distance1"></param>
        /// <param name="distance2"></param>
        /// <param name="function"></param>
        /// <param name="rnd"></param>
       public void AttachSwarm(int swarmsize, double distance1, double distance2,
Func<double,double> function, Random rnd)
       {
            this.swarmsize = swarmsize;
            this.rnd = rnd;
            this.swarm = CreateSwarm(swarmsize, distance1, distance2, function, rnd);
        }
        /// <summary>
```

```
        /// создание роя(АЛГОРИТМ ОПТИМИЗАЦИИ РОЯ ЧАСТЬ 1: СЛУЧАЙНОЕ РАСПРЕДЕЛЕНИЕ ЧАСТИЦ ПО МНОЖЕСТВУ РЕШЕНИЙ)
        /// </summary>
        public Particle[] CreateSwarm(int swarmsize, double distance1, double distance2, Func<double, double> function, Random rnd)
        {
            swarm = new Particle[swarmsize];
            for (int i = 0; i < swarmsize; i++)
            {
                swarm[i] = new Particle(function, distance1, distance2, rnd);
            }
            BestUpdate(swarm[0].X);
            for (int i = 0; i < swarmsize; i++)
            {
                if (swarm[i].Y > swarm[i].BestY)
                    BestUpdate(swarm[i].X);
            }
            return swarm;
        }

        /// <summary>
        /// обработчик таймера (АЛГОРИТМ ОПТИМИЗАЦИИ РОЯ ЧАСТЬ 2: ОСНОВНАЯ ЧАСТЬ)
        /// </summary>
        void timer_Elapsed(object sender, ElapsedEventArgs e)
        {
            k++;
            if (k >= iter)
            {
                k = 0;
                timer.Stop();
            }
            //Обновление лучших собственных положений точек
            for (int i = 0; i < swarmsize; i++)
            {
                if (swarm[i].Y > swarm[i].BestLocalY)
                {
                    swarm[i].BestLocalX = swarm[i].X;
                }
            }

            //Обновление лучшего положения системы
            for (int i = 0; i < swarmsize; i++)
            {
                if (swarm[i].Y > swarm[i].BestY)
                    BestUpdate(swarm[i].X);
            }

            //Пересчитывание скорости и положения точки
            for (int i = 0; i < swarmsize; i++)
                {
                    var alpha = rnd.NextDouble();
                    var beta = rnd.NextDouble();
                    var gamma = rnd.NextDouble();
                    swarm[i].V = gamma*(swarm[i].V + alpha * (swarm[i].BestLocalX - swarm[i].X) + beta * (swarm[i].BestX - swarm[i].X));
                    swarm[i].X += swarm[i].V;
                }
                UpdateObservers();
        }

        /// <summary>
```

```
        /// Возвращает все положения точек в данный момент
        /// </summary>
        public List<double[]> Position()
        {
            var condition = new List<double[]>();
            foreach (Particle sw in swarm)
            {
                condition.Add(sw.Position());
            }
            return condition;
        }

        /// <summary>
        /// Запусе таймера
        /// </summary>
        public void StartOptimization(int iter)
        {
            this.iter = iter;
            timer.Enabled = true;
        }
        /// <summary>
        /// Остановить таймер
        /// </summary>
        public void StopOptimization()
        {
            timer.Stop();
        }


        /// <summary>
        /// Методы для подписания/обновления наблюдателей
        /// </summary>
        public void Register(IObserver o)
        {
            listeners.Add(o);
            o.UpdateState();
        }
        public void Deregister(IObserver o)
        {
            listeners.Remove(o);
        }
        public void UpdateObservers()
        {
            foreach (IObserver ob in listeners)
            {
                ob.UpdateState();}}}
```

Графический интерфейс



1 – Задание функции и рассматриваемого диапазона

2 – задания количества частиц в алгоритме

3 – Задание количества итераций и запуск алгоритма

4 – Справка

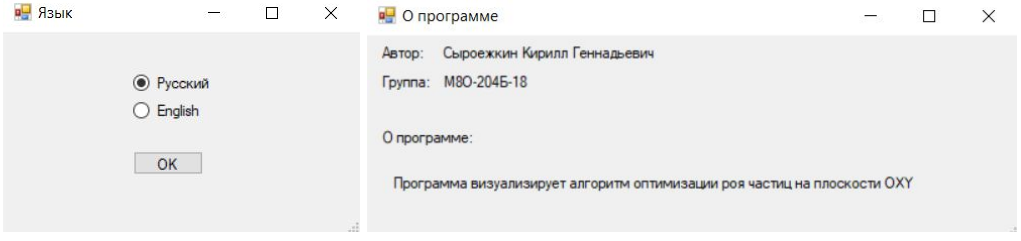
5 - Смена языка

6 – Таблица с положениями точек

7 – Место для рисования графика

Вспомогательные окна:

Смена языка Справка



Как работать с программой

Задать диапазон (вещественные числа) расчетов и функцию(любую математическую функцию), которую нужно оптимизировать.
Задать количество частиц, которые будут производить поиск глобального максимума.

Задать количество итераций, которое программа будет выполнять.

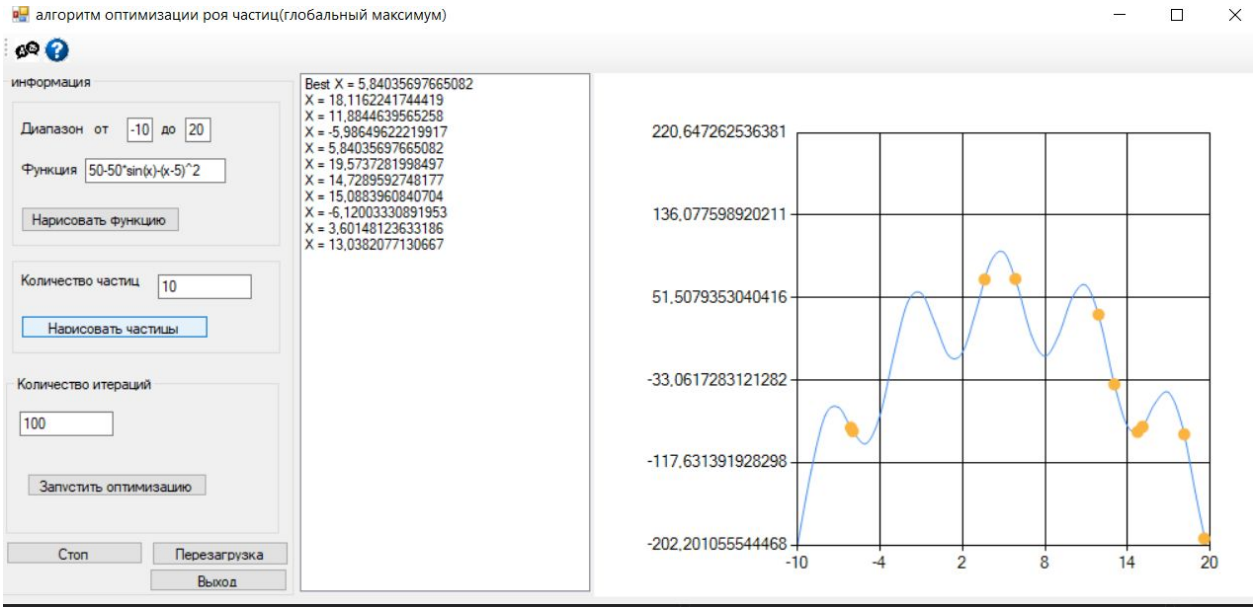
Запустить оптимизацию.

Протокол выполнения

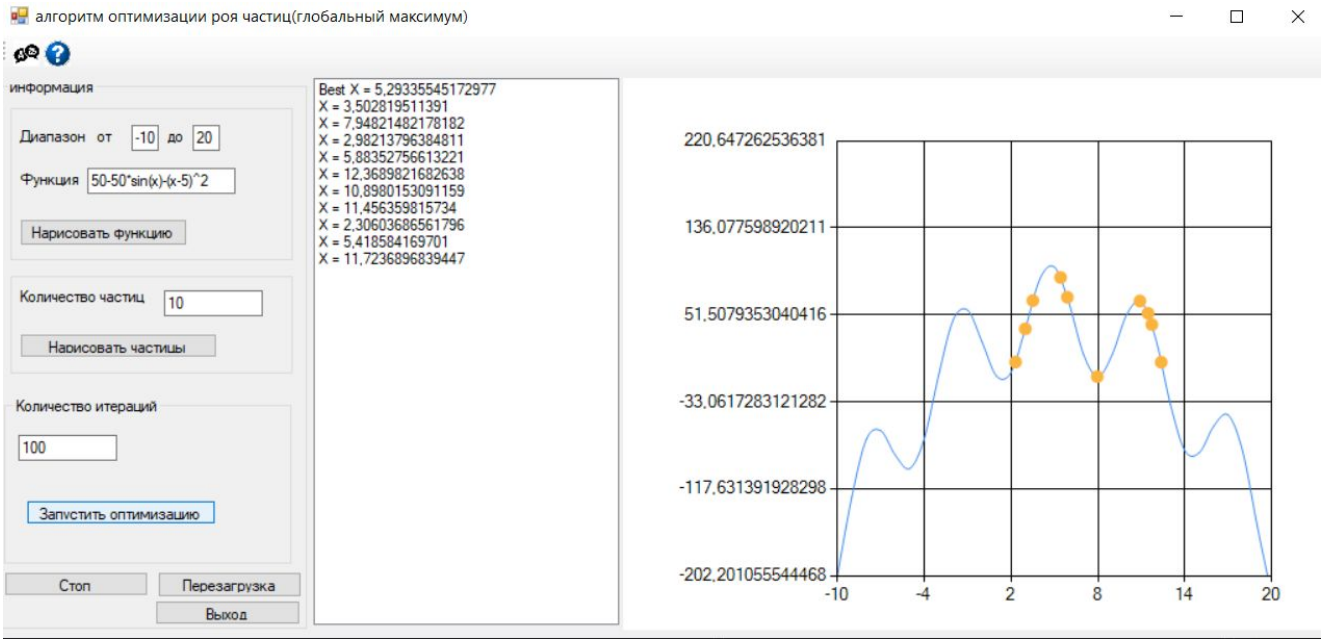
Найдем глобальный максимум функции:

$$f(x) = 50 - 50 * \sin sin\ (x)\ - (x-5)^2$$

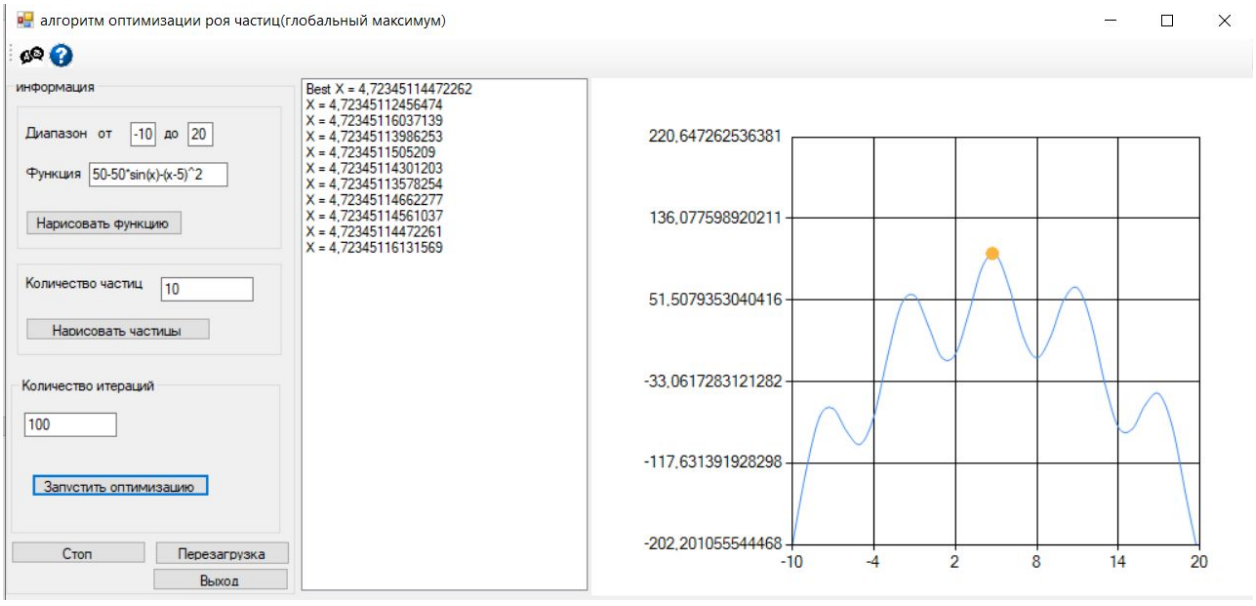
Заносим данные, и рисуем частицы



Далее все точки начали скапливаться в 1 месте



все частицы сошлись в 1

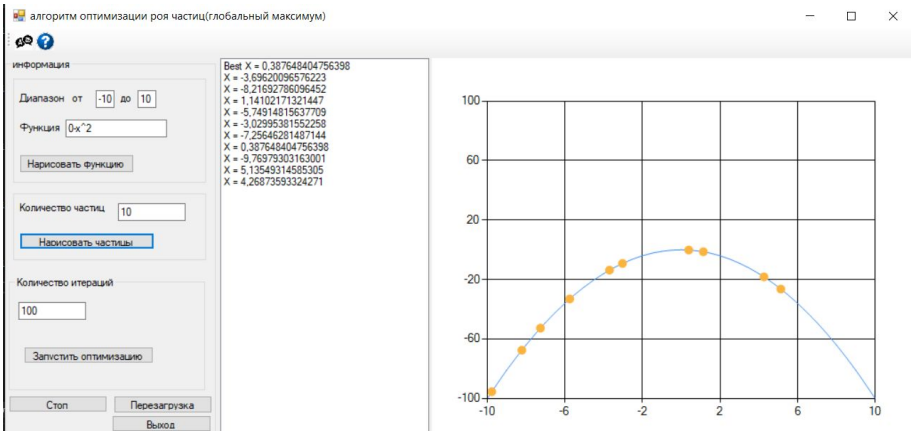


Глобальный максимум функции ~ 4,723

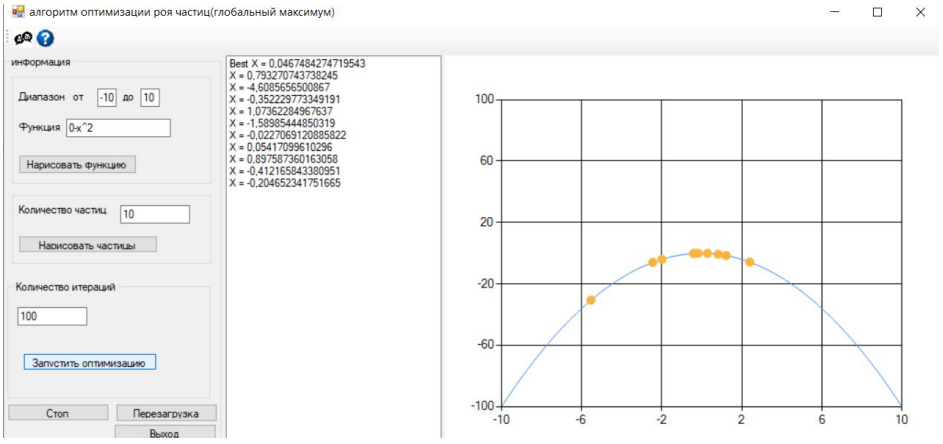
Найдем глобальный максимум функции:

$$f(x) = \quad -x^2$$

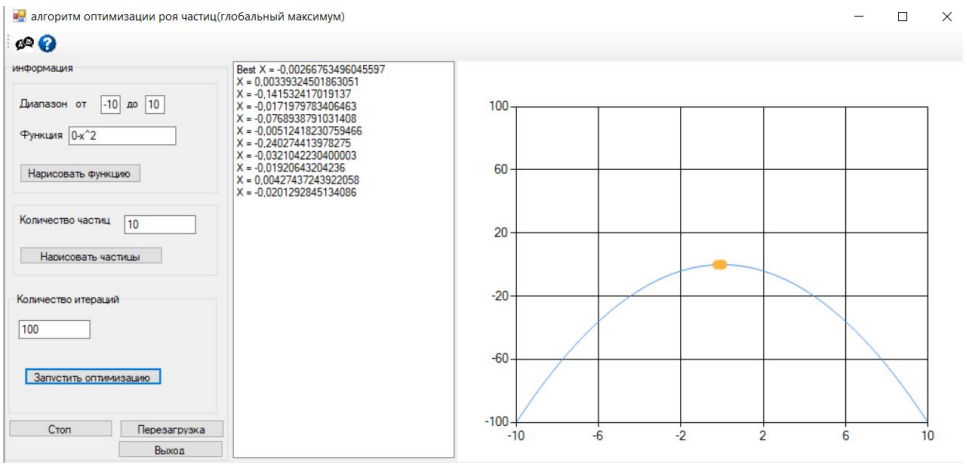
Заносим данные, и рисуем частицы



Далее все точки начали скапливаться в 1 месте



все частицы сошлись в 1 точке

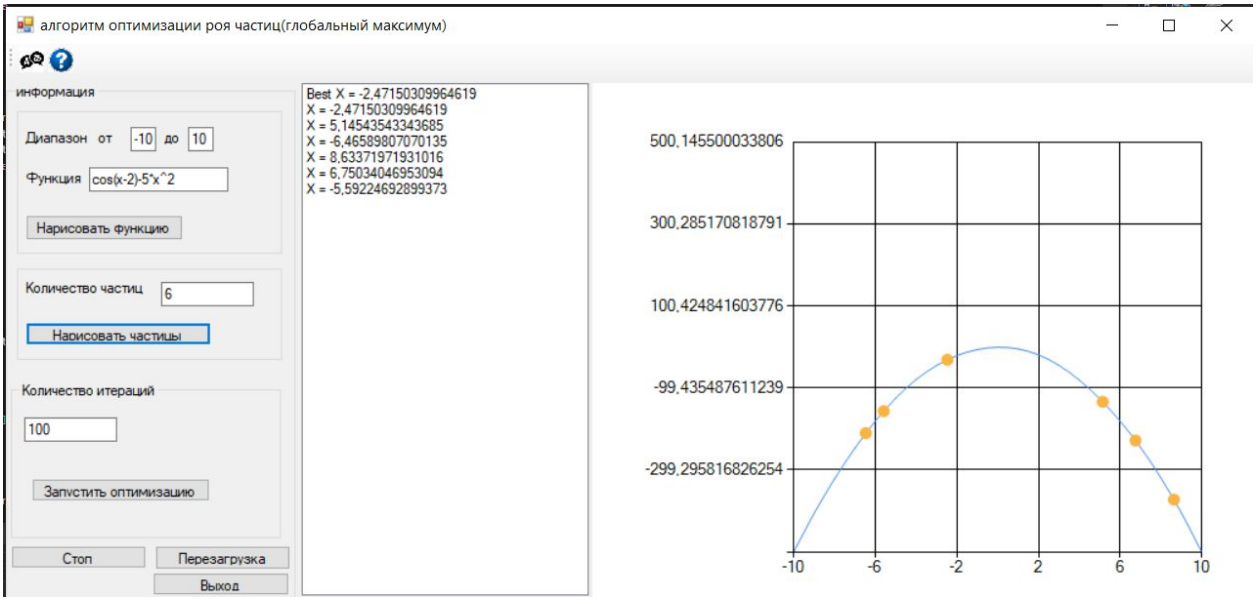


Глобальный максимум функции ~ 0

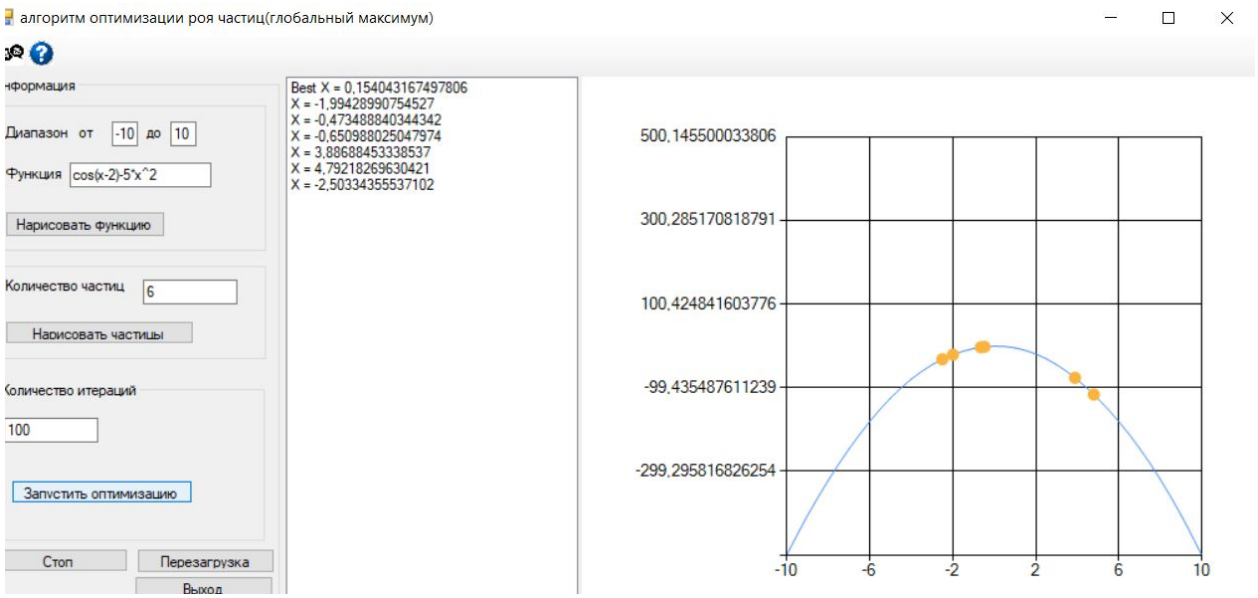
Найдем глобальный максимум функции:

$$f(x) = cos(x-2) - 5 * x^2$$

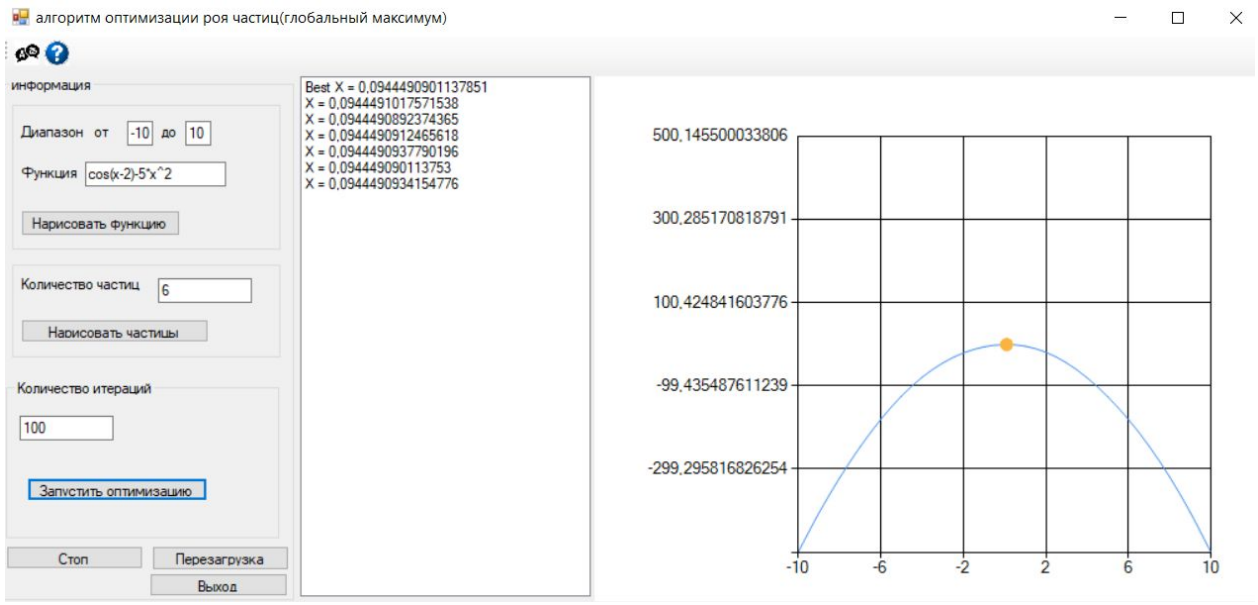
Заносим данные, и рисуем частиц



Далее все точки начали скапливаться в 1 месте



все частицы сошлись в 1 точке



Глобальный максимум функции ~ 0,1

Используемое оборудование:

Вывод

В первой статье, описывающей алгоритм роя частиц, Джеймс Кеннеди и Рассел Эберхарт высказывали

идею использования алгоритм для имитации социального поведения – Кеннеди, как социального психолога, крайне привлекала эта идея . Однако, как это уже было подтверждено, наибольшее распространение алгоритм получил в задачах оптимизации сложных многомерных нелинейных функций.

Алгоритм роя частиц широко применяется, в числе прочих, в задачах машинного обучения (в частности, для обучения нейросетей и распознавания изображений), параметрической и структурной оптимизации (форм, размеров и топологий) в области проектирования, в областях биохимии и биомеханики. По эффективности он может соперничать с другими методами глобальной оптимизации, а низкая алгоритмическая сложность способствует простоте его реализации.

Наиболее перспективными направлениями дальнейших исследований в данном направлении следует считать теоретические исследования причин сходимости алгоритма роя частиц и связанных с этим вопросов из областей роевого интеллекта и теории хаоса, комбинирование различных модификаций алгоритма для решения сложных задач, рассмотрение алгоритма роя частиц как многоагентной вычислительной системы, а также исследование возможностей включения в него аналогов более сложных природных механизмов.

Оборудование

Процессор: Intel core i7 8550u (4 ядра, 8 потоков 1.6-4GHz)

Оперативная память: 16Gb

Видеокарта: GTX 1050 4Gb

Память: 256Gb

Программное обеспечение

Операционная система: windows 10 (версия 1903)

Среда разработки: Visual Studio 2019

Версия .net framework: 4.7.2

Литература

Sean Luke. *Essentials of Metaheuristics. A Set of Undergraduate Lecture Notes by Department of Computer Science George Mason University Second Edition October, 2015 p57.*

S. Achasova, O. Bandman, V. Markova, et al., *Parallel Substitution Algorithm. Theory and Application., Singapore: World Scientific, 1994 p231.*

L. M. Adleman*, Molecular Computation Of Solutions To Combinatorial Problems, Science, 266, 11, pp. 1021–1024, 1994.*

E. Alba, B. Dorronsoro, *Cellular Genetic Algorithms, Springer, 2008*

D. Boneh, C. Dunworth, R. J. Lipton, J. Sgall*, On the Computational Power of DNA, DAMATH: Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science, 71, 1996.*

C. Detrain, J. L. Deneubourg*, Self-Organized Structures in a Superorganism: Do Ants Behave Like Molecules?, Physics of Life Reviews 3, no. 3, pp. 162-187, 2006.*

M. Dorigo, M. Birattari, T. Stutzle*, Ant Colony Optimization, Technical Report No. TR/IRIDIA/2006-023, September 2006.*

S. ***Goss, S. Aron, J.-L. Deneubourg, J. M. Pasteels****, Self-organized shortcuts in the Argentine ant, Naturwissenschaften, vol. 76, pp. 579–581, 1989.*

J. H. Holland*, Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, MI, 1975.*