

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**"ЛЭТИ" ИМ. В.И.УЛЬЯНОВА(ЛЕНИНА)**  
**Кафедра МО ЭВМ**

**ОТЧЁТ**

**по учебной практике**

**Тема: «Аппроксимация множества точек  
полиномом 3-й степени алгоритмом роя  
частиц»**

Студент гр. 1381

\_\_\_\_\_

Кагарманов Д. И.

Студент гр. 1381

\_\_\_\_\_

Исайкин Г. И.

Преподаватель

\_\_\_\_\_

Жангиров Т. Р.

Санкт-Петербург

2023

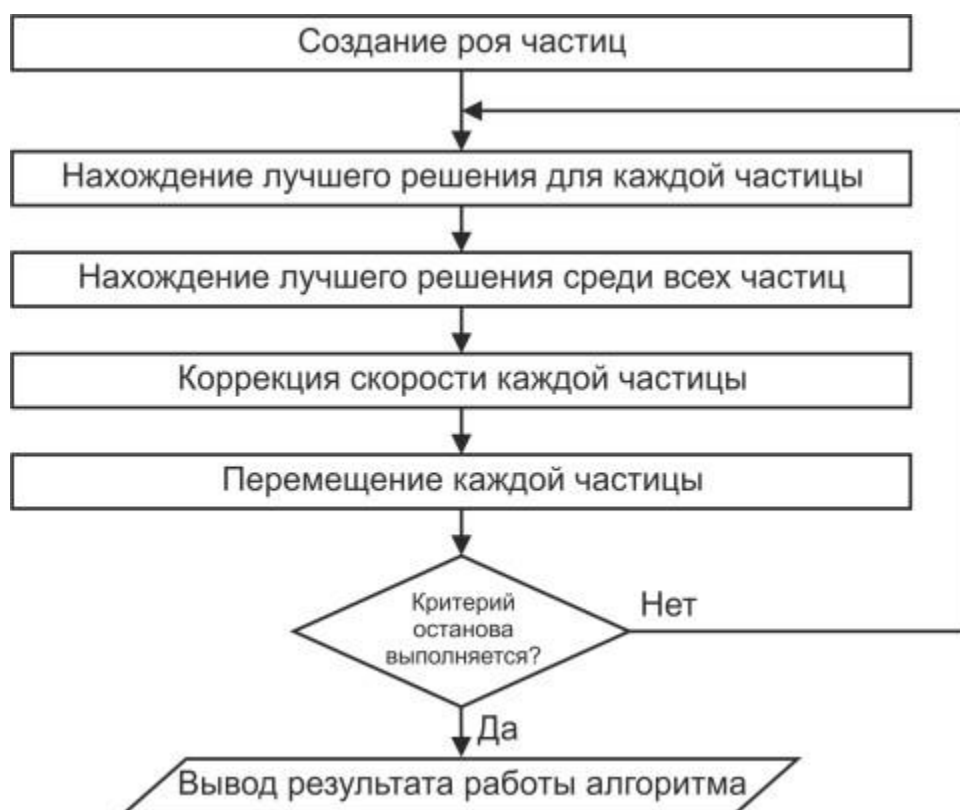
## Задание

Пусть имеется набор точек. Необходимо найти коэффициенты  $a, b, c, d$  полинома вида  $ax^3 + bx^2 + cx + d$ , чтобы полином наилучшим образом аппроксимировал данное множество точек. В качестве метрики необходимо использовать абсолютное отклонение

## Выполнение работы

### Итерация №1

#### 1. Алгоритм роя частиц



##### 1.1 Создание роя частиц.

Изначально мы имеем множество точек  $X$ , которые мы хотим аппроксимировать. Хранятся они будут в списке  $X = [(x_1, y_1), \dots, (x_n, y_n)]$ ,  $n$  – кол-во точек, оно задается пользователем.

Рой частиц в нашем случае будет представлять набор полиномов со случайно сгенерированными начальными коэффициентами  $a, b, c, d$ . Частицы храним в списке

списков:  $S = [ [a_1, b_1, c_1, d_1], \dots, [a_m, b_m, c_m, d_m] ]$ ,  $m$  – кол-во частиц, задается пользователем.

Начальная позиция  $j$ -ой частицы:  $x_j^0 = (a_j^0, b_j^0, c_j^0, d_j^0)$

## 1.2 Нахождение лучшего решения для каждой частицы.

В качестве целевой функции будем использовать сумму модулей отклонений частицы от всех заданных точек:  $\sum_{i=1}^n |y_i - s_j| \rightarrow \min$ ,

где  $n$  – кол-во точек,  $s_j$  –  $j$ -ая частица.

На каждой итерации алгоритма запоминаем лучшее решение (набор коэффициентов) для каждой частицы.

$p_j^k = (a_j^k, b_j^k, c_j^k, d_j^k)$  – лучшее решение для  $j$ -ой частицы на  $k$ -ой итерации алгоритма. Храним в списке, исходно инициализируем начальными (случайными) значениями.

## 1.3 Нахождение лучшего решения для всех частиц.

Среди решений, найденных на предыдущем шаге, находим лучшее.

$$b^k = (a^k, b^k, c^k, d^k)$$

## 1.4 Коррекция скорости каждой частицы.

Изначально у каждой  $j$ -ой частицы имеется вектор скоростей для каждого коэффициента (задается случайно):

$$v_j = (v_{ja}, v_{jb}, v_{jc}, v_{jd})$$

Классическая коррекция скорости для переменной  $a$  частицы  $j$ -ой:

$$v_{ja}^{k+1} = v_{ja}^k + \alpha_p r_p (p_{ja}^k - a_j^k) + \alpha_b r_b (b_a^k - a_j^k)$$

$$r_p, r_b = rand(0, 1)$$

$\alpha_p, \alpha_b$  – весовые коэффициенты, подбираются опытным путем для конкретной задачи. Возможно, будет возможность их выбора для удобства тестирования.

При неудовлетворительных результатах сменим коррекцию скорости на канонический алгоритм, при котором весовые коэффициенты не так сильно влияют на сходимость.

### 1.5 Перемещение частицы

Каждую частицу (то есть коэффициенты каждого полинома) изменяем по формуле:

$$\begin{aligned} a_j^{k+1} &= a_j^k + v_{ja}^{k+1} \\ b_j^{k+1} &= b_j^k + v_{jb}^{k+1} \\ c_j^{k+1} &= c_j^k + v_{jc}^{k+1} \\ d_j^{k+1} &= d_j^k + v_{jd}^{k+1} \end{aligned}$$

### 1.6 Критерий останова.

Превышение некоторого кол-ва итераций, или необходимое значение целевой функции (пределы погрешности). Вектор  $b^k$  будет содержать лучшее решение.

### Распределение ролей:

Исайкин Г. И. – GUI, тестирование

Кагарманов Д. И. – алгоритм

## 2. Реализация алгоритма

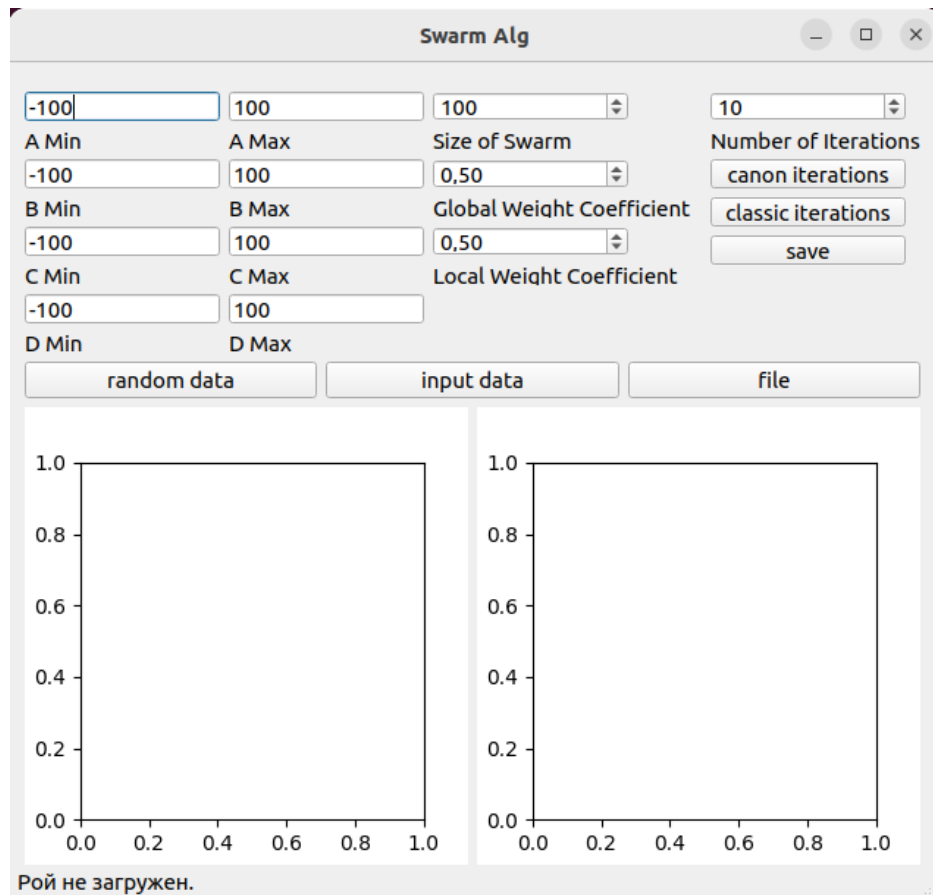
Для алгоритма были созданы два класса: классы *Swarm* и *Particle* и библиотеки *random*, *numpy*, *math*.

*Swarm* является обобщением всего роя, поэтому содержит все необходимые атрибуты, присущие всему рою: матрицу аппроксимируемых точек, размер частиц в рое, границы значений коэффициентов, лучшие значения, весовые коэффициенты, а также методы по созданию роя, запуску алгоритма (классического или канонического). Вдобавок сеттеры и геттеры атрибутов, и другие вспомогательные методы, необходимые для GUI.

Основная же суть алгоритма находится в классе *Particle* – в классе частицы. Атрибуты класса хранят положение частицы (в реалиях нашей задачи это – коэффициента полинома), ее скорости, лучшие положение и состояние целевой функции. Метод *deviation* – вычисление целевой функции, функции отклонения частицы (графика полинома) от заданных точек. Метод *CheckFunc* – проверка, является ли текущее значения функции отклонения лучшим. Метод *NextIteration\_Classic* – реализация итерации классического алгоритма. Вычисляем целевую функцию для частицы, если она является лучшей глобальной или локальной – запоминаем. Далее происходит расчет изменения скоростей по формулам и выше и движение частицы. Подобные действия для всех частиц в рое – это и есть одна итерация алгоритма. Метод *NextIteration\_canon* – реализация итерации канонического алгоритма. Более точный, чем классический. Если введенные пользователем весовые коэффициенты в сумме  $< 4$ , то алгоритм по умолчанию будет использовать значения 1.7 и 2.8 для локального и глобального решения соответственно. И последний метод – *checkLimit*. Он проверяет, не вышла ли позиция частицы за границы введенных значений, если вышла – ей присваивается лучшее найденное на данный момент решение среди всего роя (таким образом перемещаем ее ближе к основному рою).

### 3. GUI

При создании были использованы библиотеки PyQt5, matplotlib и sys и среда Qt Designer.



GUI представляет из себя окно с различными кнопками, полями и графиками.

График слева внизу – график результата работы алгоритма, то есть лучший найденный полином, а также точки, которые мы аппроксимируем.

График справа – график целевой функции, где по оси OX – номер итерации.

Поле A min – нижняя граница коэффициента A полинома  $Ax^3 + Bx^2 + Cx + D$

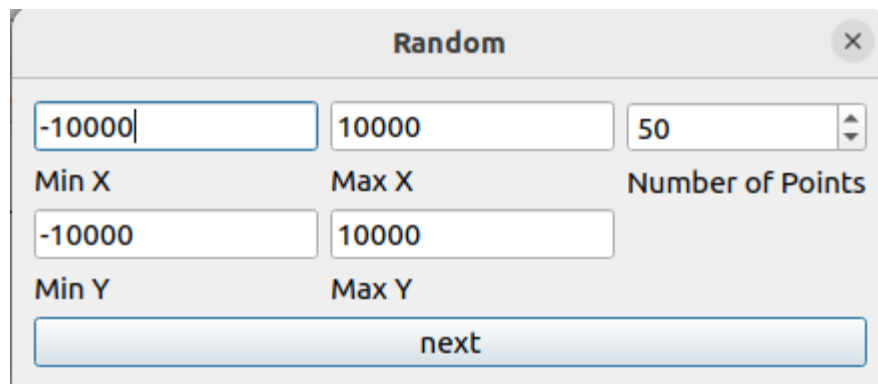
A max – верхняя граница коэффициента A. Аналогично с остальными коэффициентами.

Size of Swarm – кол-во частиц в рое

Global и Local Weight Coefficient – весовые коэффициенты (глобальный и локальный)

Number of Iterations – число итераций, которое совершить алгоритм при нажатии кнопки

ниже (одной из двух – от это зависит, какая из модификаций алгоритма выполнится)  
Кнопка save – сохранить текущие результаты алгоритма в файле одним из доступных видов: settings(без информации об итерациях) и save with iterations(с информацией о каждой итерации).  
Кнопка file – загрузить начальные данные из файла.

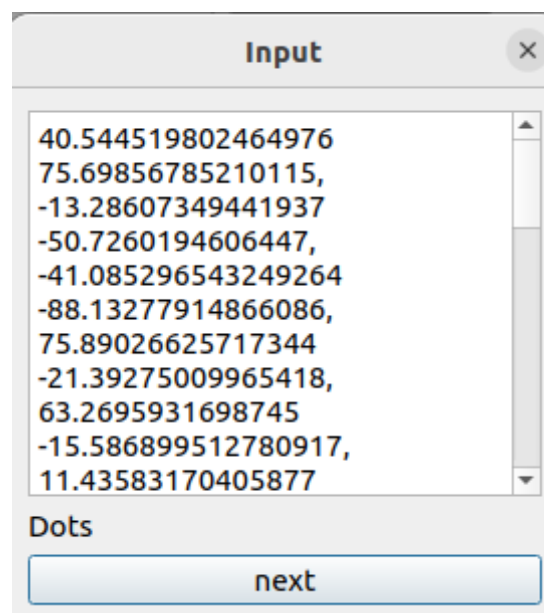


The 'Random' dialog box contains input fields for 'Min X', 'Max X', 'Min Y', 'Max Y', and 'Number of Points'. The 'Number of Points' field is a spinner set to 50. A 'next' button is at the bottom.

Min X	Max X	Number of Points
-10000	10000	50
Min Y	Max Y	
-10000	10000	

next

При нажатии кнопки random появится окно для случайной генерации начальных данных. В окне можно выбрать границы генерации и количество аппроксимируемых точек.



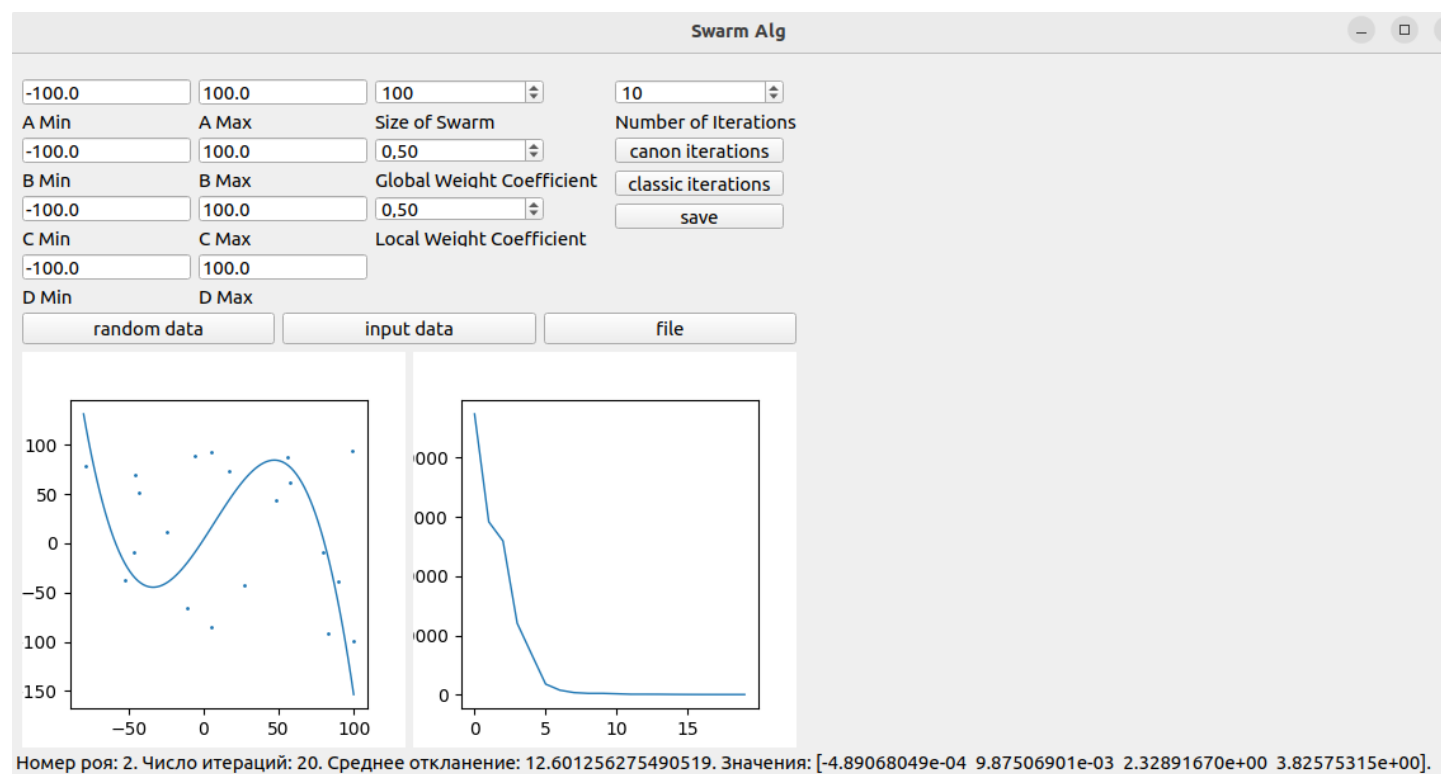
The 'Input' dialog box displays a list of 12 numerical values. Below the list is a 'Dots' label and a 'next' button.

40.544519802464976  
75.69856785210115,  
-13.28607349441937  
-50.7260194606447,  
-41.085296543249264  
-88.13277914866086,  
75.89026625717344  
-21.39275009965418,  
63.2695931698745  
-15.586899512780917,  
11.43583170405877

Dots

next

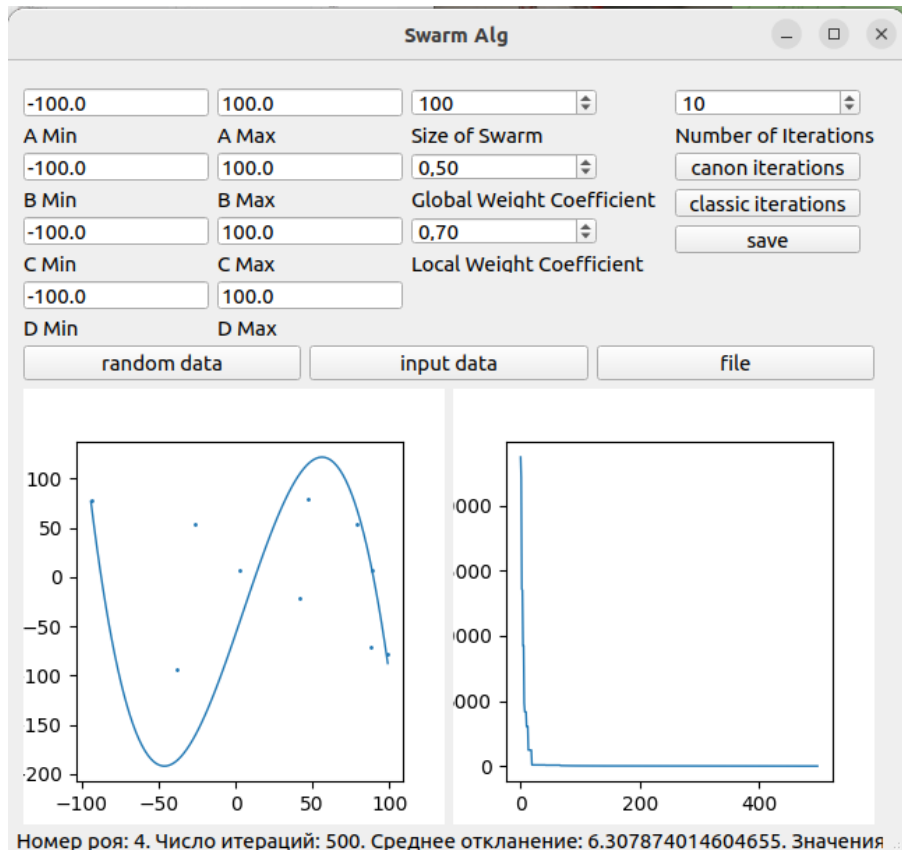
При нажатии кнопки input data появится окно для ручного ввода координат точек, где одна точка записывается в две строчки: первая строчка – координата X, вторая – Y, потом запятая и следующая точка. Если изменить точку или точки когда алгоритм уже выполнил хотя бы 1 итерация – алгоритм начнет работу заново.



Снизу, в строке отображается кол-во итераций, среднее отклонение и коэффициента полинома в списке(Значения).

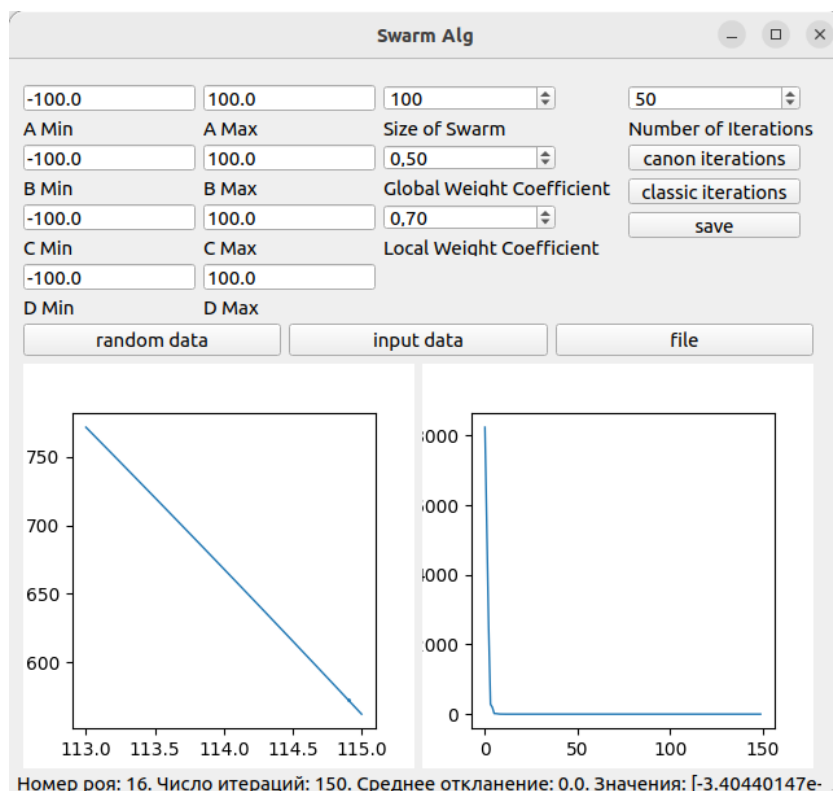


## 4. Тестирование

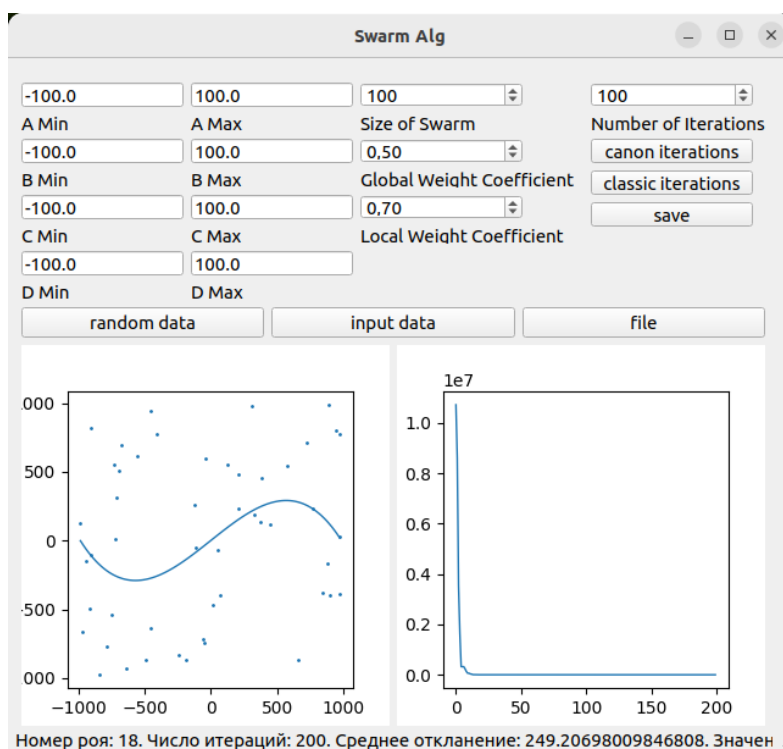


Классический алгоритм роя из 100 частиц. Аппроксимация десяти случайных точек.

График демонстрирует достаточно хорошую аппроксимацию.



Канонический алгоритма для роя из 100 частиц. Аппроксимация одной точки.



Канонический алгоритма для роя из 100 частиц. Аппроксимация 50-ти точек.

**Вывод:**

Была решена задача нахождения коэффициентов многочлена 3-ей степени методом роя частиц. Была написана программа с GUI на языке Python. Для запуска программы необходимо запустить файл Facade.py