**Санкт-Петербургский государственный университет**

**Факультет прикладной математики и процессов управления**

**Лабораторная работа №2**

**Исследование минимума функции с помощью**

**генетического алгоритма**

**6 вариант**

|  |  |  |
| --- | --- | --- |
| Студентка гр. Б15-ПУ |  | Гладкая М.В. |
| Преподаватель |  | Дик А.Г. |

Санкт-Петербург

2023 г.

**Оглавление**

1. **Цель работы ………………………………………………………………3**
2. **Задача………………………………………………………………………3**
3. **Теоритическая часть……………………………………………………..3**
4. **Описание программы ……………………………………………………4**
   1. **Список используемых библиотек……...………………………...…4**
   2. **Описание функций и классов..…………...…………………………4**
   3. **Общий ход программы………………………………………………6**
5. **Рекомендации программиста……………………………………………9**
6. **Контрольный пример…………………………………………………….9**
7. **Анализ полученных данных…...…………………………..…………..12**
8. **График исследуемой функции…………………………………………18**
9. **Вывод…………………………………………..………………………….18**
10. **Приложения……………………………………………………………...19**
11. **Цель работы**

Исследование кодирования генотипа хромосом в генетическом алгоритме и их проверка его эффективности для нахождения минимума функции.

1. **Задачи**

* Изучить особенности кодирования генетических алгоритмов
* Написать программу, находящую минимум заданной функции с помощью генетического алгоритма
* Реализовать визуализацию результатов
* Проанализировать полученные результаты

1. **Теоритическая часть**

Описание алгоритма: генетический алгоритм является одним из эволюционных алгоритмов, то есть в его основе лежат процессы отбора, мутации и комбинирования (скрещивания).

Более подробно рассмотрим каждый шаг генетического алгоритма:

Каждая особь популяции обладает тремя основными параметрами: положением по оси Х, положением по оси У и значением целевой функции (именно это значение выступает в роли основного параметра при селекции).

На первом этапе происходит создание начальной популяции, где каждая особь случайным образом получает свои координаты по Х и У. Далее идет процесс размножения. Случайным образом выбирается пара родителей, которые производят потомков. Потомки должны наследовать черты родителей. Для того, чтобы не произошло доминирования со стороны одной особи, в размножении участвуют гены обоих родителей.

После размножения отбирается часть особей, которая подвергается случайным мутациям в генах. В случае с поиском экстремума будет меняться положение индивида по координатам Х и У.

В самом конце должен произойти отбор наиболее приспособленных особей. Особи с наименьшим показателем значения целевой функции проходят дальше. Для того, чтобы популяция не росла беспрерывно, её численность регулируется определенным заданным параметром.

Шаги “размножение”, “мутация” и “отбор” повторяются до тех пор, пока не будет достигнута точка Останова. В качестве конца алгоритма зачастую выбирают достижение максимального числа итераций алгоритма, которое задается во входных данных.

1. **Описание программы**

Программный код написан на языке Python.С кодом можно ознакомиться по ссылке <https://github.com/9Neechan/GeneticAlgorithmMinFunc> .

* 1. **Список используемых библиотек**

Список используемых библиотек представлен в Таблице 1.

*Таблица 1. Список используемых библиотек*

|  |  |
| --- | --- |
| **Название** | **Описание** |
| random | Генерация рандомных значений |
| tkinter | Реализация UI |
| PIL | Работа с изображениями графиков внутри UI |
| pandas | Работа с массивами данных |
| matplotlib | Отрисовка графиков |

* 1. **Описание функций и классов**

В таблице 2 представлено описание функций файла main.py.

*Таблица 2. Функции main.py*

|  |  |  |
| --- | --- | --- |
| **Имя функции** | **Входные данные** | **Описание функции** |
| func() | x, y - переменные | Возвращает значение заданной функции от переменных x, y |
| generate\_initial\_population() | size – размер популяции, start – нижняя граница диапазона поиска минимума,  end – верхняя граница | Генерирует популяцию размера size с рандомными значениями из промежутка (start, end) и записывает ее в массив population |
| crossover() | parent1, parent2 - родители | Создает двух детей путем скрещивания генов родителей: func(x1, y2) и func(x2, y1) |
| mutate() | child - ребенок | Осуществляет мутацию одного из генов ребенка: рандомно выбирается какой ген подвергнется мутации – x или y, далее к значению гена прибавляется рандомное значение из диапазона (-0.5, 0.5) |
| genetic\_  algorithm() | num\_generations – количество поколений,  population\_size – размер популяции,  start – нижняя граница диапазона поиска минимума,  end – верхняя граница | Осуществляет поиск минимума функции с заданными параметрами с помощью генетического алгоритма и подготавливает данные для визуализации.  Данные для таблицы записывает в дата-фрейм data. |
| checkButton  State() | args\* | Отвечает за активацию кнопок «Рассчитать» и «Очистить» в зависимости от ввода аргументов |
| clickButton() | - | Отвечает за нажатие кнопки «Рассчитать». По нажатию кнопки вызывается функция genetic\_algorithm, производятся расчеты, удаляются результаты предыдущих расчетов.  Создается объект класса ShowResult – выводятся графики и таблица.  Выводятся окончательные результаты работы программы – (x, y) и f(x, y) = min |
| cleanButton() | - | Отвечает за нажатие кнопки «Очистить». По нажатию кнопки удаляются результаты предыдущих расчетов. |

В таблице 3 представлено описание класса ShowResult.

*Таблица 3. Класс ShowResult*

|  |  |  |
| --- | --- | --- |
| **Имя функции** | **Входные данные** | **Описание функции** |
| \_\_init\_\_() | self,  data – дата-фрейм для составления таблицы | Инициализирует и отрисовывает части UI по визуализации полученных результатов |
| next\_img() | self | Реализует логику работы кнопки «Далее» |
| previous\_img() | self | Реализует логику работы кнопки «Назад» |
| show\_img() | self | Отрисовывает графики |
| show\_table() | self | Отрисовывает таблицу, заполняет ее значениями и отрисовывает скролл-бар |

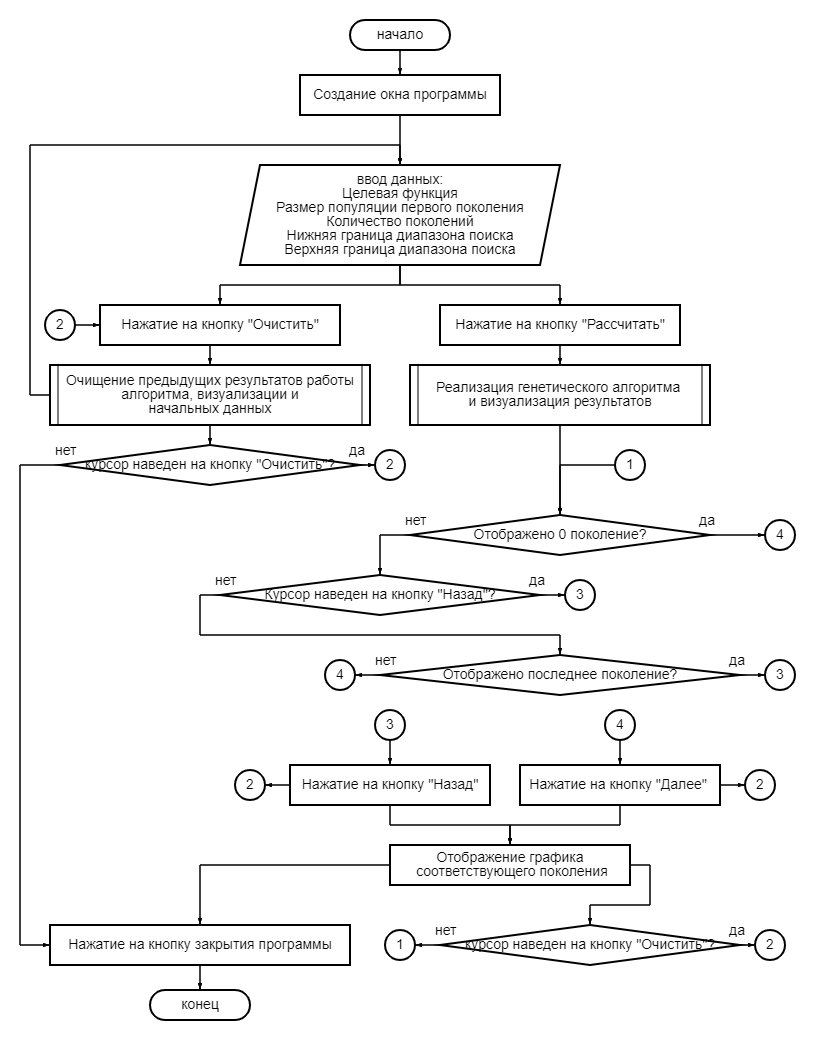
В таблице 4 представлено описание класса UserInput.

*Таблица 4. Класс UserInput*

|  |  |  |
| --- | --- | --- |
| **Имя функции** | **Входные данные** | **Описание функции** |
| \_\_init\_\_() | self  text – текст, отображаемый над кнопкой  from\_, to – нижняя и верхняя границы ввода  increment – величина пролистывания значений  initial\_value – изначально отображаемое значение в ячейке ввода | Определяет шаблон для атрибутов ввода |

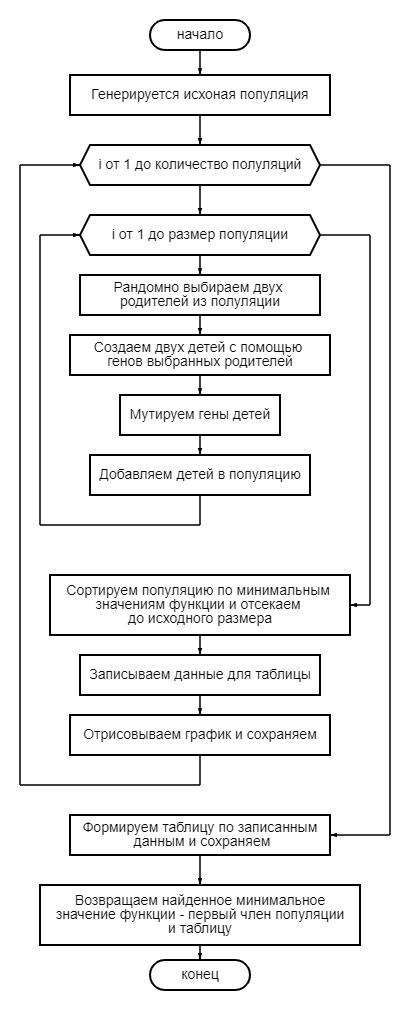
* 1. **Общий ход программы**

На рисунке 1 представлена блок-схема общего хода программы.



*Рисунок 1. Блок-схема общего хода программы*

На рисунке 2 изображен ход подпрограммы, реализующей работу генетического алгоритма.



*Рисунок 2. Блок-схема подпрограммы, реализующей работу генетического алгоритма*

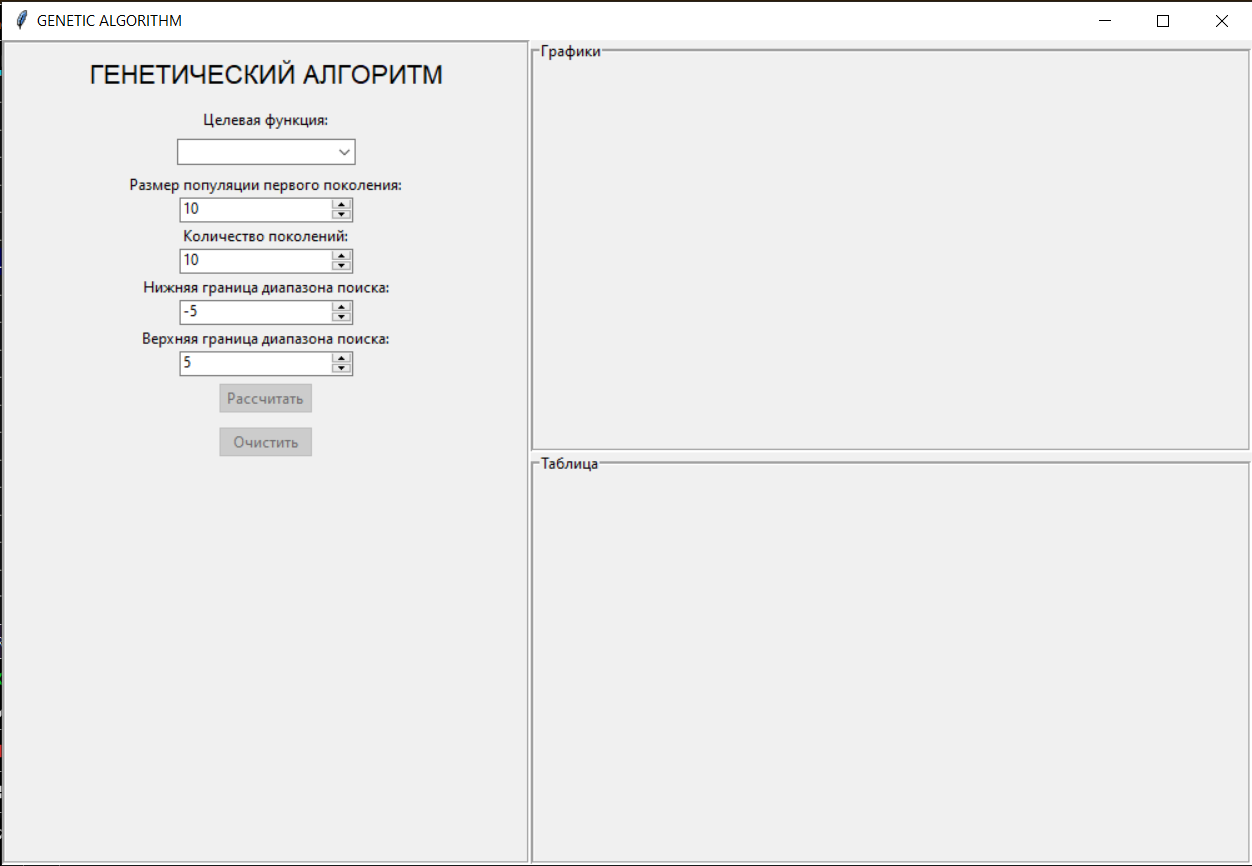
1. **Рекомендации программиста**

Для запуска программы необходима 64-битная операционная система Windows и python не ниже 3.9. Для работы с кодом необходима IDE PyCharm или другая любая среда разработки для python.

Импортируйте библиотеки из пункта 4.1.

1. **Контрольный пример**

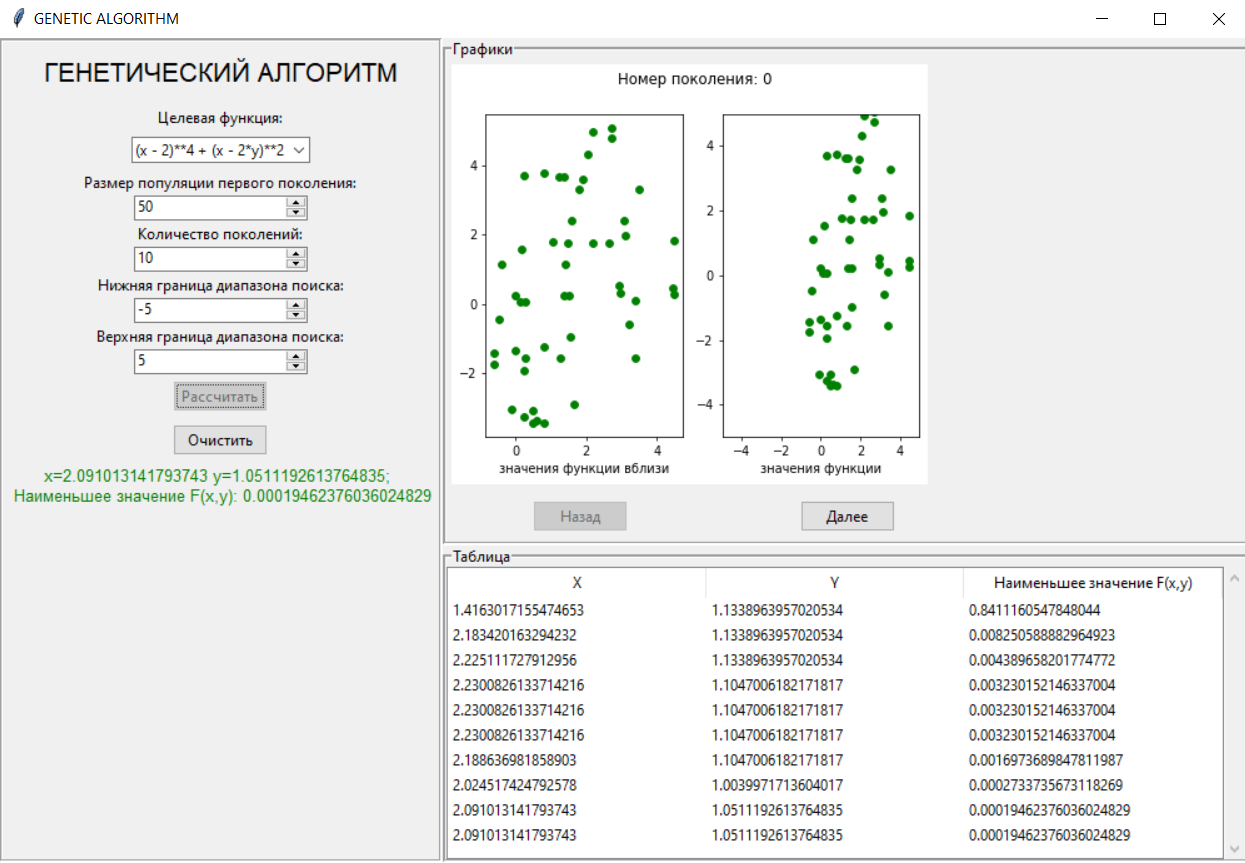
Запустите программу. Перед вами откроется начальное окно программы с уже заполненными начальными значениями, при желании их можно поменять (Рис. 3). Выберите целевую функцию из предложенного списка. После того, как все поля входных данных будут заполнены, активируется кнопка «Рассчитать». Нажмите на нее для расчета минимума выбранной целевой функции с помощью генетического алгоритма.



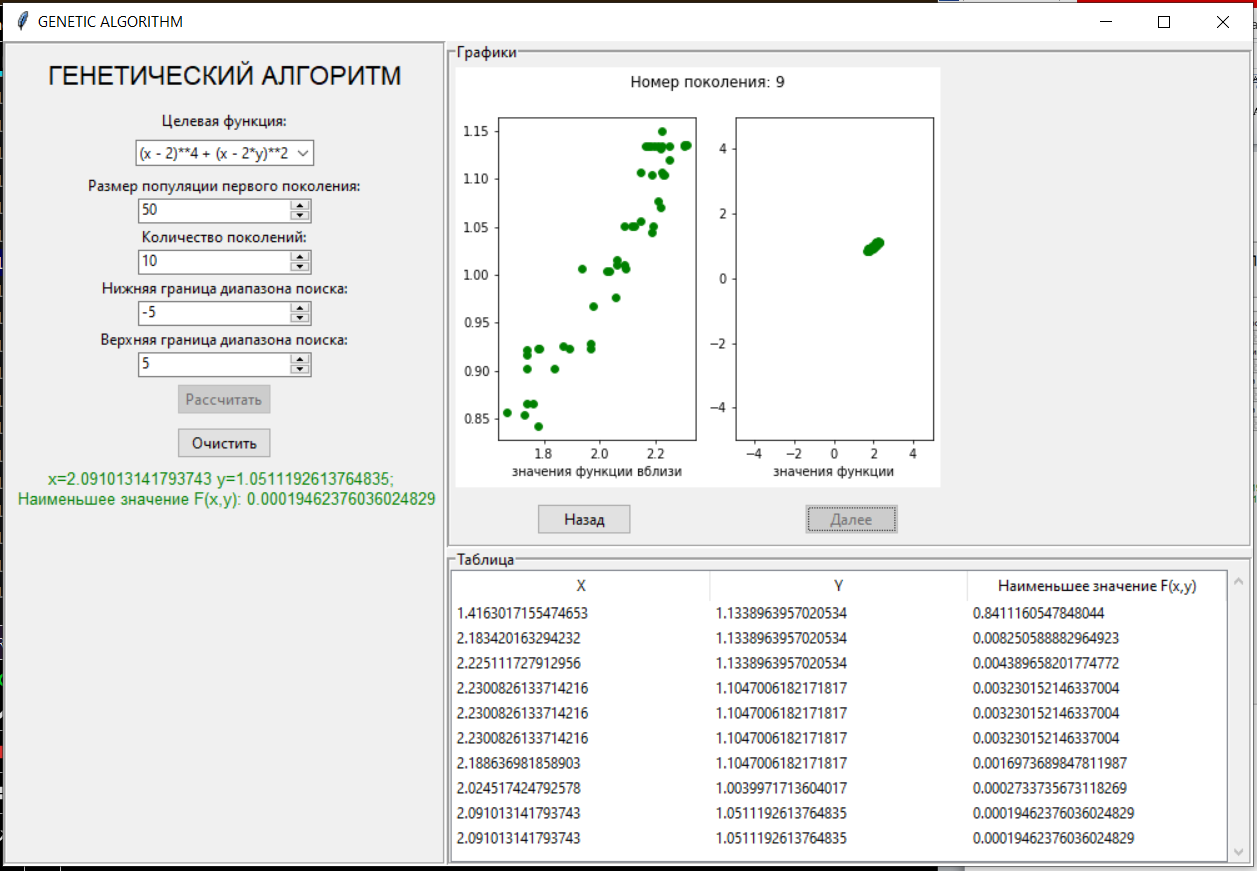
*Рисунок 3. Начальное окно*

По окончании расчетов зеленым шрифтом будет выведен минимум функции и значения x и y, при которых оно получено. В окне «Графики» будут выведены графики для каждого поколения, где будут отражены значения минимума функции для каждой особи в популяции. В окне «Таблица» будет выведена таблица со значениями x, y и минимальным значением функции для каждого поколения.

Чтобы пролистывать между графиками поколений нажимайте на кнопки «Далее» и «Назад». На рисунках 4 и 5 изображены результаты работы программы для 0 и 9 поколения соответственно при введенных значениях.

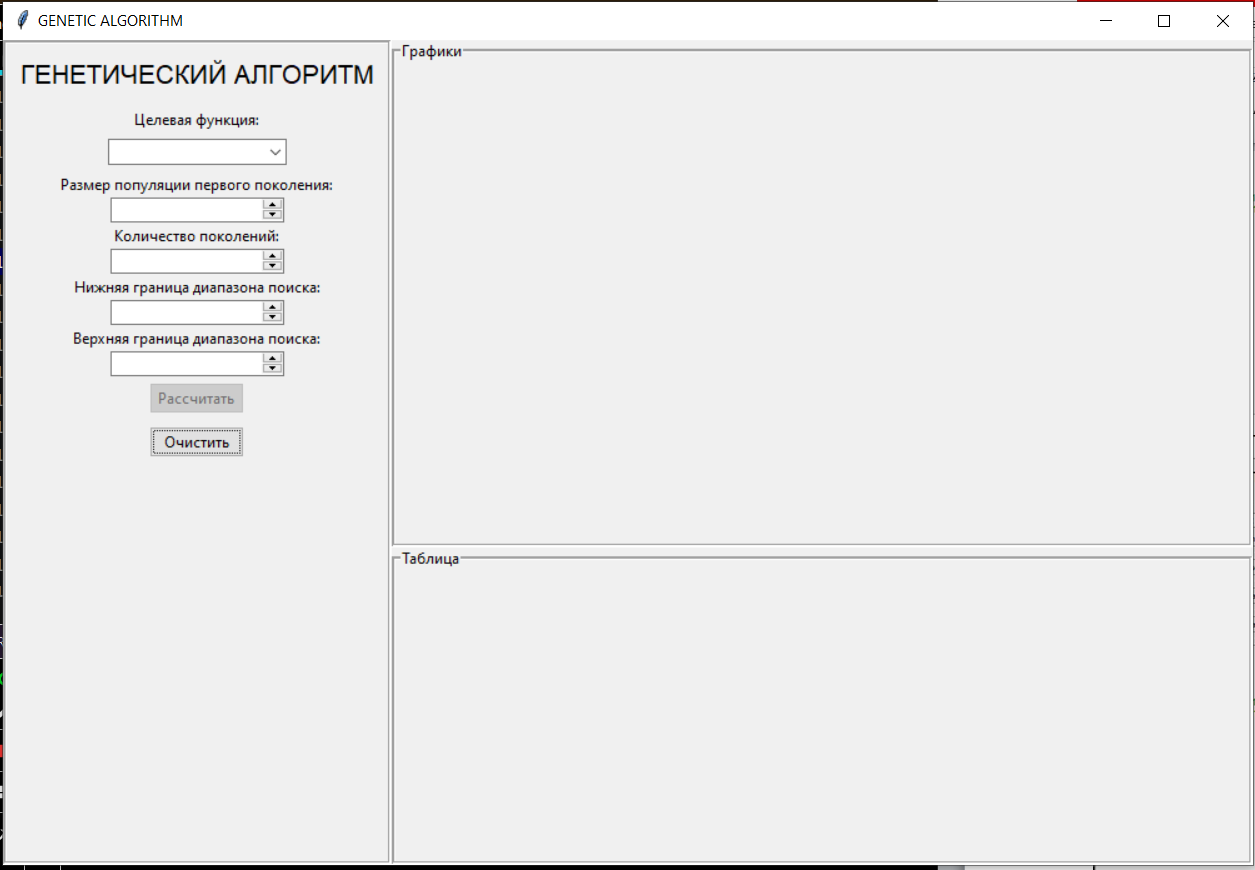


*Рисунок 4. Окно программы после нажатия кнопки «Рассчитать»*



*Рисунок 5. Окно программы с изображением графика последнего поколения*

Чтобы очистить введенные параметры и (или) результаты работы программы нажмите на кнопку «Очистить». Окно программы после нажатия на кнопку «Очистить» изображено на рисунке 6.

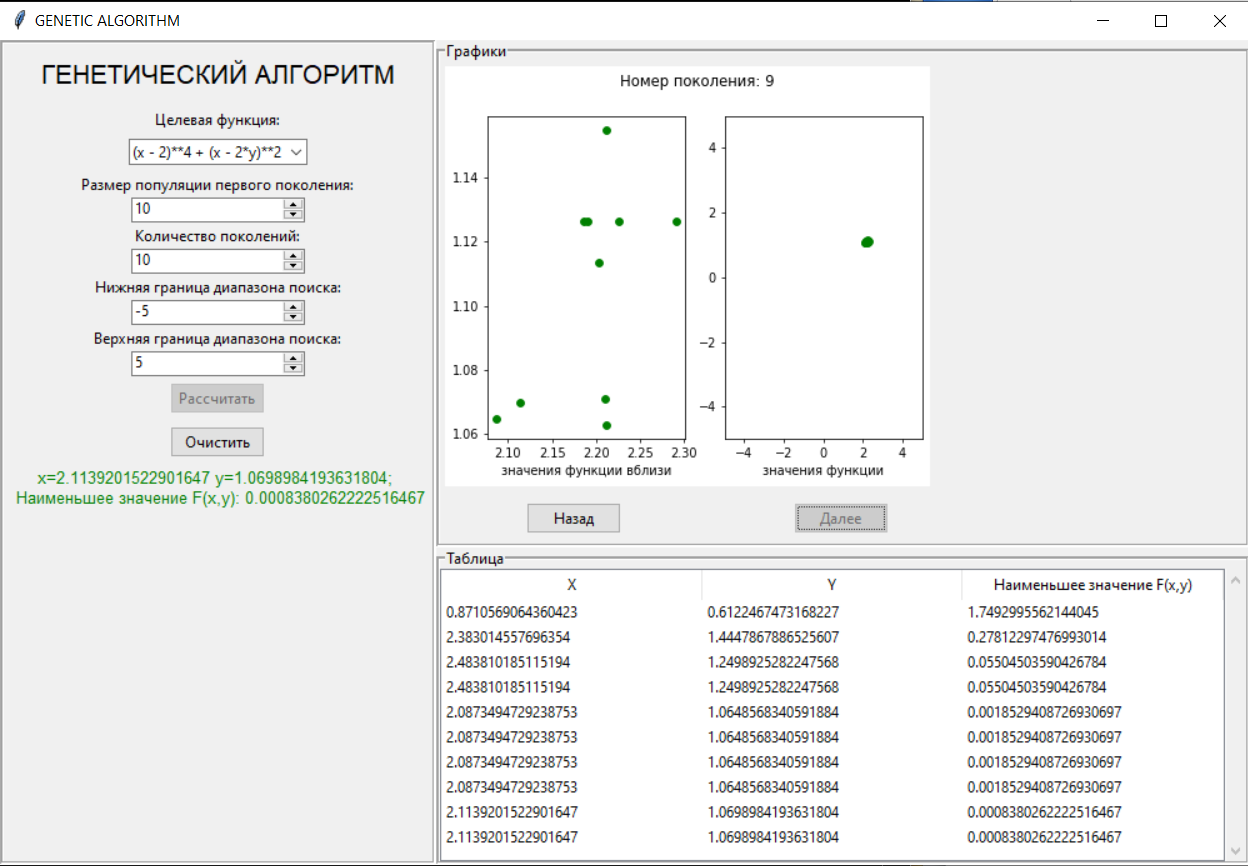


*Рисунок 6. Окно программы после нажатия на кнопку «Очистить»*

1. **Анализ результатов работы алгоритма**

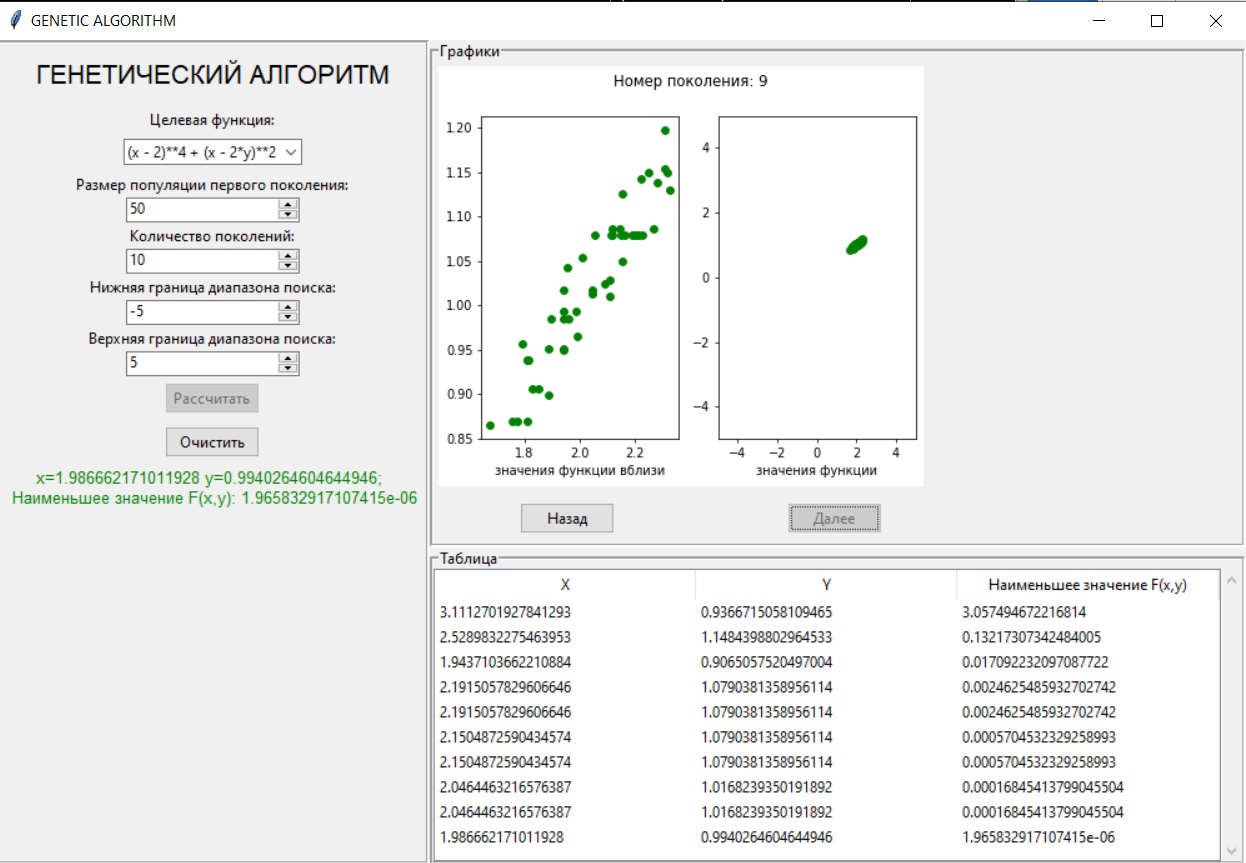
У заданной функции  точкой минимума является точка (2, 1).

При начальных данных минимум равен (2.114, 1.100) – рисунок 7.

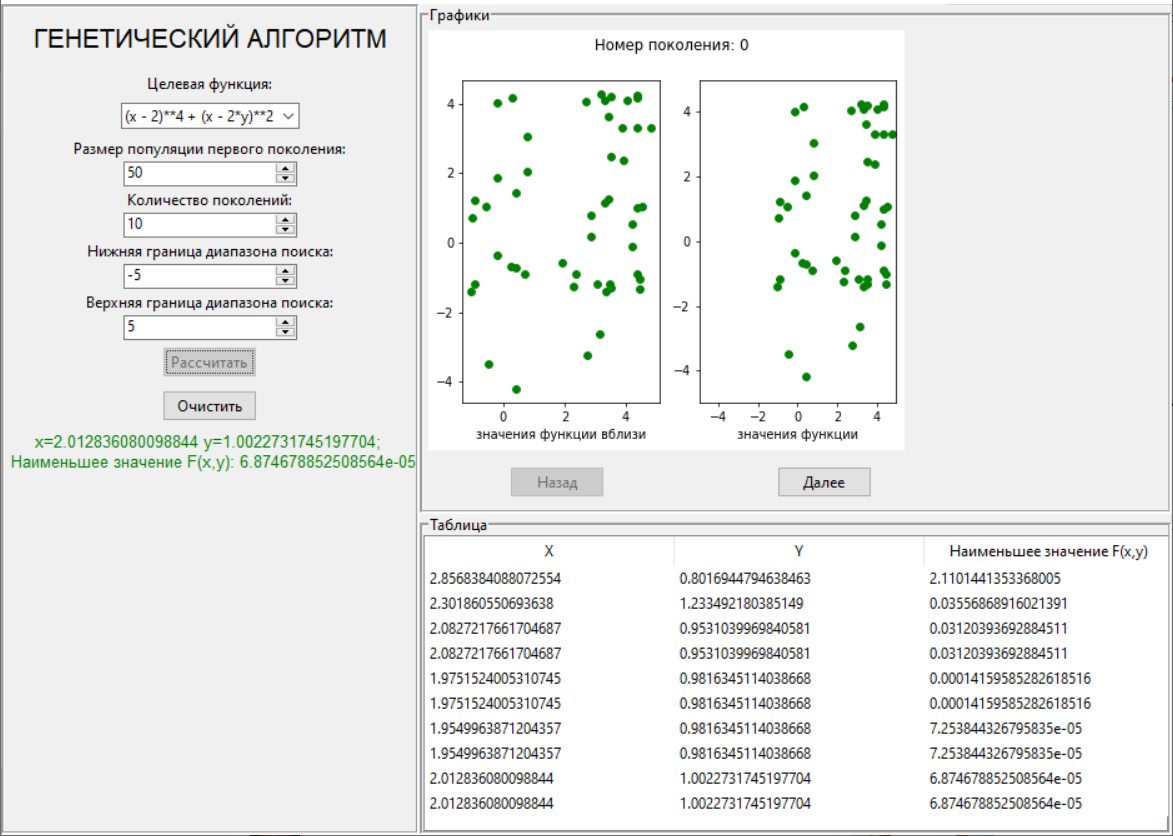


*Рисунок 7. Результат работы программы при начальных данных*

Увеличим размер популяции первого поколения до 50, тогда минимум станет равен (1.987, 0.994) и (2.013, 1.002). Это оказало положительное влияние на точность нахождения минимума – погрешность стала меньше (рисунки 8, 9), поэтому оставим это число для дальнейшей настройки точности алгорима.

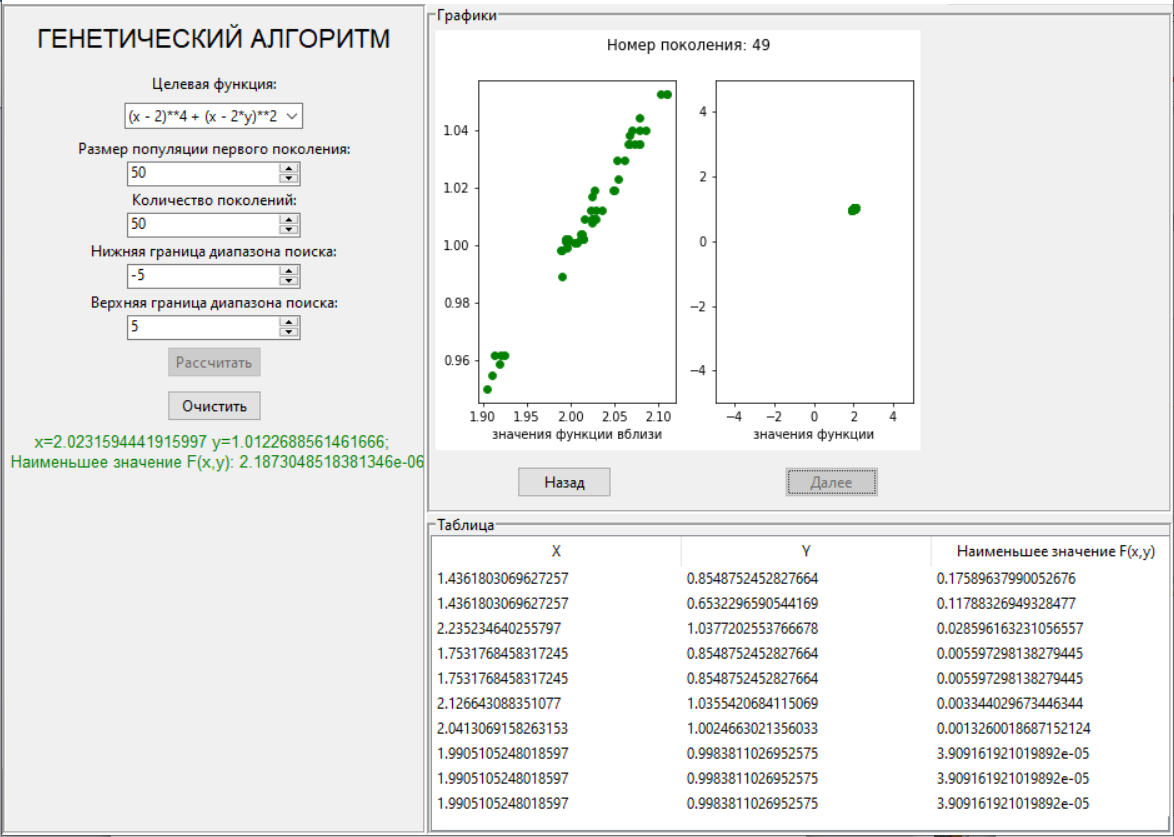


*Рисунок 8. Результат работы программы*



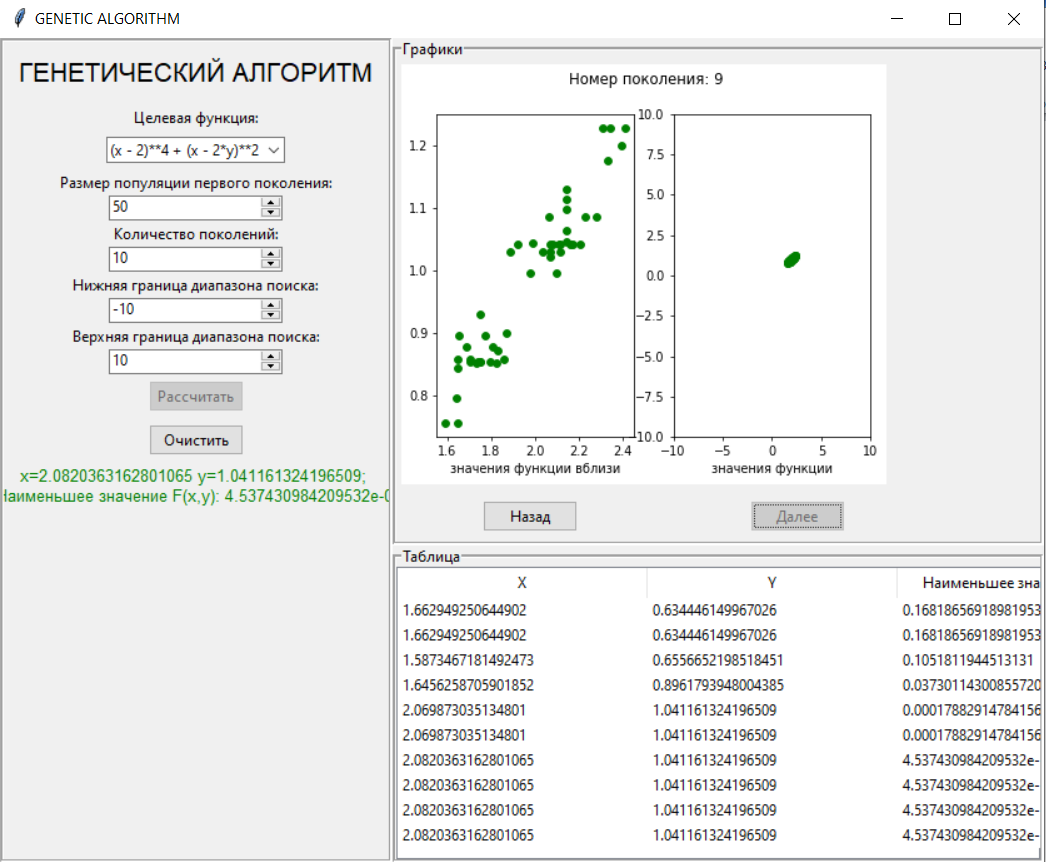
*Рисунок 9. Результат работы программы*

Увеличим количество поколений до 50, это не улучшило результаты (рисунок 10) – (2.023, 1.012), поэтому вернем это значение обратно на 10 пунктов.



*Рисунок 10. Результат работы программы*

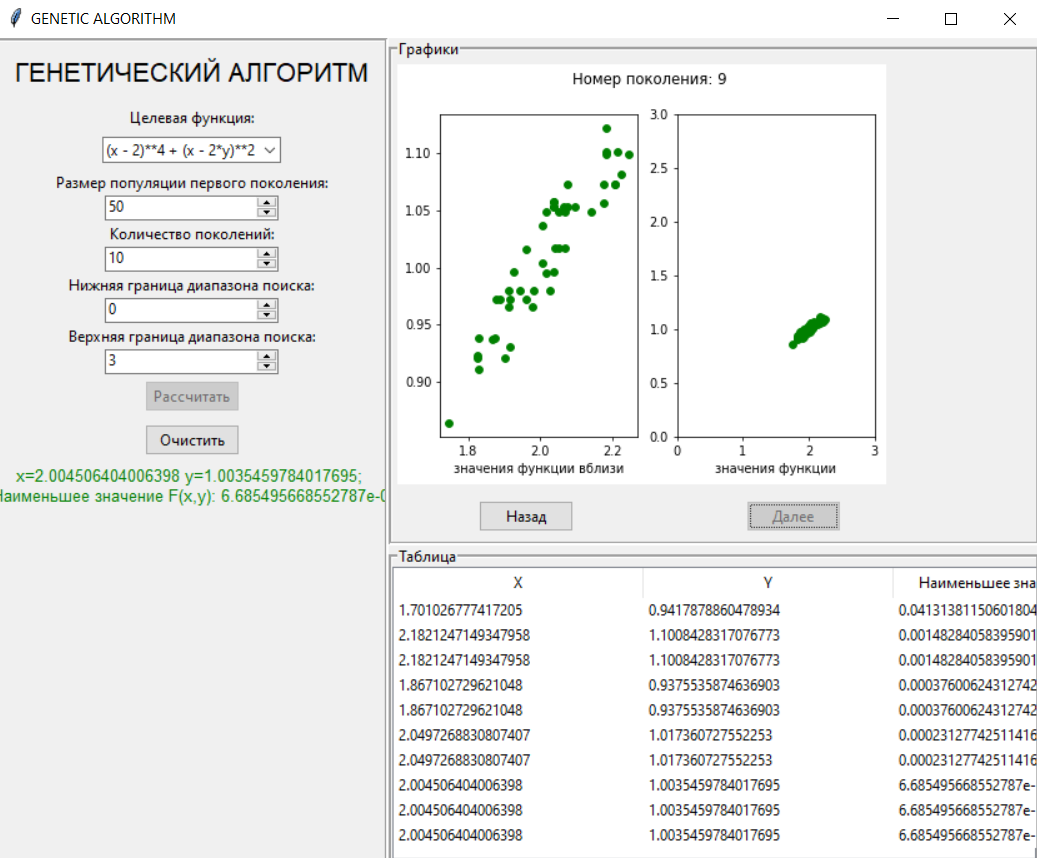
Увеличим диапазон поиска минимума на 10 – на 5 поднимем верхнюю границу и на 5 опустим нижнюю границу. На точность работы алгоритма это повлияло отрицательно (рисунок 11).



*Рисунок 11. Результат работы программы*

В связи с ухудшением точности работы алгоритма при расширении диапазона поиска, наоборот, уменьшим его – рисунок 12. Как видно, чем меньше диапазон поиска, тем точнее результаты работы.

При диапазоне поиска минимума от 0 до 3 с учтенными ранее наилучшими параметрами была достигнута погрешность в <0,005 – найденный минимум составил (2.0045, 1.003).

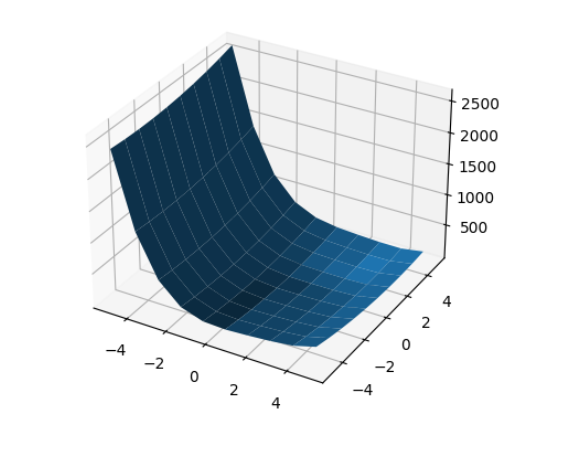


*Рисунок 12. Результат работы алгоритма*

Далее, при изменении размера популяции первого поколения и количества поколений при диапазоне поиска от 0 до 3 показатели погрешности не улучшались. Отсюда сделаем вывод, что минимальная погрешность точности нахождения минимума заданной функции – 0,005. Она достигается при входных данных, введенных на рисунке 12.

1. **График исследуемой функции**

На рисунке 13 представлен график исследуемой функции. График был потроен с помощью библиотеки matplotlib, язык программирования Python.



*Рисунок 13. График функции* 

1. **Вывод**

В ходе данной работы были изучены особенности кодирования генетических алгоритмов, написана соответствующая программа, находящая минимум заданной функции по двум точкам, реализована визуализация результатов работы программы и проанализированы полученные результаты.

**Приложения**

*Приложение 1. main.py*

1. **import** random
2. **from** tkinter **import** **\***
3. **from** tkinter **import** ttk
4. **from** PIL **import** Image, ImageTk
5. **import** pandas as pd
6. **import** matplotlib.pyplot as plt
8. INPUT\_FUNS **=** ["(x - 2)\*\*4 + (x - 2\*y)\*\*2"]

11. # ------------------------------------algorithm-realisation-------------------------------------------------------------
12. **def** func(x, y):
13. **return** eval(INPUT\_FUNS[0])  # (x - 2)\*\*4 + (x - 2\*y)\*\*2

16. **def** generate\_initial\_population(size, start, end):
17. population **=** []
18. **for** i **in** range(size):
19. x **=** random.uniform(start, end)
20. y **=** random.uniform(start, end)
21. it **=** [x, y, func(x, y)]
22. population.append(it)
23. **return** population

26. **def** crossover(parent1, parent2):
27. child1 **=** [parent1[0], parent2[1], func(parent1[0], parent2[1])]
28. child2 **=** [parent2[0], parent1[1], func(parent2[0], parent1[1])]
29. **return** child1, child2

32. **def** mutate(child):
33. x **=** child[0]
34. y **=** child[1]
35. **if** random.random() > 0.5:
36. x **+=** random.uniform(**-**0.5, 0.5)
37. **else**:
38. y **+=** random.uniform(**-**0.5, 0.5)
39. **return** [x, y, func(x, y)]

42. **def** genetic\_algorithm(num\_generations, population\_size, start, end):
43. dataForTable **=** []
44. snapshot\_arr **=** []
45. # Генерируем исходную популяцию
46. population **=** generate\_initial\_population(population\_size, start, end)
48. **for** i **in** range(num\_generations):
49. **for** \_ **in** range(population\_size):
50. # Производим новых потомков с мутирующими генами
51. parent1 **=** random.choice(population)
52. parent2 **=** random.choice(population)
53. child1, child2 **=** crossover(parent1, parent2)
54. child1 **=** mutate(child1)
55. child2 **=** mutate(child2)
56. population.append(child1)
57. population.append(child2)
59. # Отбираем лучших из поколения
60. population **=** sorted(population, key**=lambda** z: z[2])
61. population **=** population[:population\_size]
62. # записываем данные для изуализации таблицы
63. dataForTable.append(population[0])
65. # Отрисовка и сохранение графика
66. x **=** list(map(**lambda** x: x[0], population))
67. y **=** list(map(**lambda** x: x[1], population))
68. snapshot\_name **=** f"pictures/g{i}.png"
69. fig, (ax1, ax2) **=** plt.subplots(1, 2)
70. fig.suptitle(f"Номер поколения: {i}")
71. ax2.set\_xlabel("значения функции")
72. ax2.plot(x, y, 'go')
73. ax2.set\_xlim(start, end)
74. ax2.set\_ylim(start, end)
75. ax1.set\_xlabel("значения функции вблизи")
76. ax1.plot(x, y, 'go')
77. plt.savefig(snapshot\_name, dpi**=**70, bbox\_inches**=**'tight')
78. plt.close()
79. snapshot\_arr.append(snapshot\_name)
81. # Сохраняем таблицу
82. data **=** pd.DataFrame(dataForTable)
83. data.columns **=** ['X', 'Y', 'Наименьшее значение F(x,y)']
85. **return** population[0], data

88. #-------------------------------img-and-table-frames-logic--------------------------------------------------------------
89. **class** ShowResult:
90. **def** \_\_init\_\_(self, data):
91. # Счётчик для изображений
92. self.i **=** 0
93. self.max\_i **=** int(num\_generations.item.get()) **-** 1
94. # Рамка для наших изображений
95. self.canvas **=** Canvas(frame\_btn, width**=**420, height**=**345, borderwidth**=**2)
97. # Добавим 1-ое изображение
98. self.photo **=** None
99. self.show\_img()
101. # Кнопки для перелистывания изображений
102. self.button\_next **=** ttk.Button(frame\_btn, text**=**"Далее", command**=**self.next\_img, state**=**NORMAL)
103. self.button\_next.grid(row**=**1, column**=**1)
104. self.button\_back **=** ttk.Button(frame\_btn, text**=**"Назад", command**=**self.previous\_img, state**=**DISABLED)
105. self.button\_back.grid(row**=**1, column**=**0)
107. # Выведем таблицу
108. self.data **=** data
109. self.show\_table()
111. **def** next\_img(self) **-**> None:
112. self.i **+=** 1
113. self.button\_back['state'] **=** NORMAL
114. self.show\_img()
115. **if** self.i **==** self.max\_i:
116. self.button\_next['state'] **=** DISABLED
118. **def** previous\_img(self) **-**> None:
119. self.i **-=** 1
120. self.button\_next['state'] **=** NORMAL
121. self.show\_img()
122. **if** self.i **==** 0:
123. self.button\_back['state'] **=** DISABLED
125. **def** show\_img(self):
126. self.photo **=** ImageTk.PhotoImage(Image.open("pictures/g{0}.png".format(self.i)))
127. self.canvas.create\_image(3, 3, anchor**=**'nw', image**=**self.photo)
128. self.canvas.grid(column**=**0, row**=**0, columnspan**=**2)
130. **def** show\_table(self) **-**> None:
131. # Создаем виджет таблицы
132. treeview **=** ttk.Treeview(frame\_scroll, columns**=**list(self.data.columns), show**=**'headings')
133. treeview.pack(side**=**LEFT, fill**=**BOTH, expand**=**1)
134. # Добавляем колонки в таблицу
135. **for** col **in** self.data.columns:
136. treeview.heading(col, text**=**col)
137. # Добавляем строки в таблицу и заполняем значения ячеек данными из дата-фрейма
138. **for** i, row **in** self.data.iterrows():
139. treeview.insert('', END, values**=**list(row))
140. # Создаем scrollbar widget и добавляем его в таблицу
141. scrollbar **=** ttk.Scrollbar(frame\_scroll, orient**=**VERTICAL, command**=**treeview.yview)
142. scrollbar.pack(side**=**RIGHT, fill**=**Y)
143. treeview.configure(yscrollcommand**=**scrollbar.set)

146. #------------------------------------------------buttons-logic----------------------------------------------------------
148. **def** checkButtonState(**\***args) **-**> None:
149. """ Кнопка активируется когда все атрибуты input введены """
150. **if** num\_generations.item.get() **and** population\_size.item.get() **and** start.item.get() **and** end.item.get() **and** combobox:
151. btn\_start['state'] **=** NORMAL
152. **else**:
153. btn\_start['state'] **=** DISABLED
154. **if** num\_generations.item.get() **or** population\_size.item.get() **or** start.item.get() **or** end.item.get() **or** combobox:
155. btn\_clean['state'] **=** NORMAL

158. **def** clickButton() **-**> None:
159. """ По нажатию кнопки производятся расчёты """
160. btn\_start['state'] **=** DISABLED
161. xy\_bestScore, data **=** genetic\_algorithm(population\_size**=**int(population\_size.item.get()),
162. num\_generations**=**int(num\_generations.item.get()),
163. start**=**min(int(start.item.get()), int(end.item.get())),
164. end**=**max(int(start.item.get()), int(end.item.get())))
166. # Перед тем как сгенерировать новые виджеты, удалим старые
167. **for** widget **in** frame\_input.winfo\_children():
168. **if** widget.winfo\_name() **==** "result\_label":
169. widget.destroy()
170. **for** widget **in** frame\_btn.winfo\_children():
171. widget.destroy()
172. **for** widget **in** frame\_scroll.winfo\_children():
173. widget.destroy()
175. # Создаём новые
176. ShowResult(data)
177. text **=** "x={0} y={1}; \n Наименьшее значение F(x,y): {2}".format(xy\_bestScore[0], xy\_bestScore[1], xy\_bestScore[2])
178. Label(frame\_input, text**=**text, fg**=**'green', font**=**('Arial', 10), name**=**"result\_label").pack()

181. **def** cleanButton() **-**> None:
182. """ По нажатию кнопки удаляем старые данные """
183. population\_size.item.delete(0, END)
184. num\_generations.item.delete(0, END)
185. start.item.delete(0, END)
186. end.item.delete(0, END)
187. combobox.set('')
188. **for** widget **in** frame\_input.winfo\_children():
189. **if** widget.winfo\_name() **==** "result\_label":
190. widget.destroy()
191. **for** widget **in** frame\_btn.winfo\_children():
192. widget.destroy()
193. **for** widget **in** frame\_scroll.winfo\_children():
194. widget.destroy()

197. #---------------------------------------user-input----------------------------------------------------------------------
199. **class** UserInput:
200. """ Класс определяющий шаблон для атрибутов ввода tkinter """
202. **def** \_\_init\_\_(self, text, from\_, to, increment, initial\_value):
203. # Текстовые переменные tkinter
204. self.var **=** StringVar()  # текст вводимый пользователем
205. self.var.set(initial\_value)
207. # Определяем структуру объекта ввода
208. Label(frame\_input, text**=**text).pack()
209. self.item **=** ttk.Spinbox(frame\_input, validate**=**'key', textvariable**=**self.var,
210. from\_**=**from\_, to**=**to, increment**=**increment)
211. self.item.pack()
212. self.var.trace('w', checkButtonState)
214. #-------------------------------------------------UI--------------------------------------------------------------------
216. # Создаётся окно пользователя
217. root\_window **=** Tk()
218. root\_window.title("GENETIC ALGORITHM")
219. root\_window.geometry('1000x660')
221. # Создание фреймов и настройка их свойств
222. frame\_input **=** Frame(root\_window, bd**=**3, relief**=**GROOVE)
223. Label(frame\_input, text**=**"ГЕНЕТИЧЕСКИЙ АЛГОРИТМ", font**=**('Arial', 16)).pack(pady**=**10)
224. frame\_input.grid(column**=**0, row**=**0, rowspan**=**2, sticky**=**'NSEW')
226. frame\_img **=** LabelFrame(root\_window, bd**=**3, relief**=**GROOVE, text**=**"Графики")
227. frame\_img.grid(column**=**1, row**=**0, sticky**=**'NSEW')
229. frame\_table **=** LabelFrame(root\_window, bd**=**3, relief**=**GROOVE, text**=**"Таблица")
230. frame\_table.grid(column**=**1, row**=**1, sticky**=**'NSEW')
231. # Дополнительные мини-фреймы
232. frame\_btn **=** Frame(frame\_img)
233. frame\_btn.pack(fill**=**BOTH, expand**=**1)
234. frame\_scroll **=** Frame(frame\_table)
235. frame\_scroll.pack(fill**=**BOTH, expand**=**1)
237. # Задание параметров для столбцов и строк
238. root\_window.columnconfigure(0, weight**=**1)
239. root\_window.columnconfigure(1, weight**=**4)
240. root\_window.rowconfigure(0, weight**=**1)
241. root\_window.rowconfigure(1, weight**=**1)
243. # Ввод функции которую мы хотим исследовать
244. Label(frame\_input, text**=**"Целевая функция:").pack()
245. var\_txt **=** StringVar()
246. combobox **=** ttk.Combobox(frame\_input, textvariable**=**var\_txt)
247. combobox['values'] **=** INPUT\_FUNS
248. combobox['state'] **=** 'readonly'
249. combobox.pack(padx**=**5, pady**=**5)
250. var\_txt.trace('w', checkButtonState)
252. # Ввод данных
253. population\_size **=** UserInput(text**=**"Размер популяции первого поколения:", initial\_value**=**"10", from\_**=**2, to**=**999, increment**=**1)
254. num\_generations **=** UserInput(text**=**"Количество поколений:", initial\_value**=**"10", from\_**=**1, to**=**999, increment**=**1)
255. start **=** UserInput(text**=**"Нижняя граница диапазона поиска:", initial\_value**=**"-5", from\_**=-**99999, to**=**99999, increment**=**1)
256. end **=** UserInput(text**=**"Верхняя граница диапазона поиска:", initial\_value**=**"5", from\_**=-**99999, to**=**99999, increment**=**1)
258. # Запуск программы
259. btn\_start **=** ttk.Button(frame\_input, text**=**"Рассчитать", command**=**clickButton, state**=**DISABLED)
260. btn\_start.pack(padx**=**5, pady**=**5)
262. # Очистить старые значения
263. btn\_clean **=** ttk.Button(frame\_input, text**=**"Очистить", command**=**cleanButton, state**=**DISABLED)
264. btn\_clean.pack(padx**=**5, pady**=**5)
266. root\_window.mainloop()