**Санкт-Петербургский государственный университет**

**Факультет прикладной математики и процессов управления**

**Лабораторная работа №3**

**Разработка и реализация алгоритма роя частиц для решения**

**задач глобальной оптимизации**

**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…………………………………………16**
9. **Вывод…………………………………………..………………………….16**
10. **Приложения………………………………………………………………17**
11. **Цель работы**

Исследование особенностей алгоритмов роевого интеллекта для решения задач глобальной оптимизации и сравнение с генетическим алгоритмом.

1. **Задачи**

* Изучить особенности кодирования роевого интеллекта
* Написать программу поиска минимума функции с помощью алгоритма роя частиц
* Протестировать программу на заданной функции
* Реализовать GUI
* Проанализировать полученные результаты и сравнить их с генетическим алгоритмом

1. **Теоритическая часть**

Алгоритм роя частиц (PSO) является эффективным методом поиска минимума функции от двух переменных. Он использует идею имитации поведения стаи птиц или роя насекомых.

Метод PSO направлен на итеративную оптимизацию задачи, начиная с набора или популяции возможных решений, называемых в данном контексте роем частиц, в котором каждая частица знает глобальное наилучшее положение в рое, а также его индивидуальное наилучшее положение, найденное до сих пор в процессе поиска в пространстве решения задачи.

На каждой итерации на скорость и положение каждой частицы в рое, представленные D-мерными векторами, влияют индивидуальные и коллективные знания, направляющие повторяющиеся полеты частиц над пространством возможных решений задачи в поисках оптимума, пока не будет удовлетворен подходящий критерий остановки.

В PSO производится несколько основных этапов, включая инициализацию роя, вычисление фитнесс-функции для каждой частицы, стратегию обновления и оценку приспособленности.

Чтобы понять алгоритм роя частиц, представьте себе n-мерное пространство (область поиска), в котором рыщут частицы (агенты алгоритма). В начале частицы разбросаны случайным образом по всей области поиска, и каждая частица имеет случайный вектор скорости. В каждой точке, где побывала частица, рассчитывается значение целевой функции. При этом каждая частица запоминает, какое (и где) лучшее значение целевой функции она лично нашла, а также каждая частица знает, где расположена точка, являющаяся лучшей среди всех точек, которые разведали частицы. На каждой итерации частицы корректируют свою скорость (модуль и направление), чтобы с одной стороны быть поближе к лучшей точке, которую частица нашла сама (авторы алгоритма назвали этот аспект поведения "ностальгией"), и, в то же время, приблизиться к точке, которая в данный момент является глобально лучшей. Через некоторое количество итераций частицы должны собраться вблизи наиболее хорошей точки, хотя возможно, что часть частиц останется где-то в относительно неплохом локальном экстремуме, но главное, чтобы хотя бы одна частица оказалась вблизи глобального экстремума.

Самое интересное в алгоритме - это коррекция скорости, именно от этого шага зависит сходимость алгоритма. В первоначальном виде алгоритма коррекция скорости выглядела следующим образом:

*vi*,*t*+1 = *vi*,*t* + *φ*p *r*p (*p*i - *xi*,*t*) + *φ*g *r*g (*g*i - *xi*,*t*)

Здесь:

* *vi*,*t* – *i*-я компонента скорости при *t*-ой итерации алгоритма
* *xi*,*t* – *i*-я координата частицы при *t*-ой итерации алгоритма
* *pi* – *i*-я координата лучшего решения, найденного частицей
* *gi* – *i*-я координата лучшего решения, найденного всеми частицами
* *rp*, *rg* – случайные числа в интервале (0, 1)
* *φp*, *φg* – весовые коэффициенты, которые надо подбирать под конкретную задачу.

Затем корректируем текущую координату каждой частицы:

*xi*,*t*+ 1 = *xi*,*t* + *vi*,*t*+1

После этого рассчитываем значение целевой функции в каждой новой точке, каждая частица проверяет, не стала ли новая координата лучшей среди всех точек, где она побывала. Затем среди всех новых точек проверяем, не нашли ли мы новую глобально лучшую точку, и, если нашли, запоминаем ее координаты и значение целевой функции в ней.

Один из самых распространенных вариантов алгоритма вводит нормировку коэффициентов φ*p* и φ*g*, чтобы сходимость не так сильно зависела от их выбора.

*vi*,*t*+1 = *χ* [*vi*,*t* + *φp* *rp* (*p*i - *xi*,*t*) + *φg* *rg* (*g*i - *xi*,*t*)],

https://jenyay.net/uploads/Programming/ParticleSwarm/eqn_1.gif, *φ* = *φp* + *φg* (*φ* > 4, k∈(0, 1))

1. **Описание программы**

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

* 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 |
| math | Использование математических функций |

* 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 |
| swarm\_  algorithm () | num\_generations – количество поколений,  population\_size – размер популяции,  start – нижняя граница диапазона поиска минимума,  end – верхняя граница  fi\_p, fi\_g - весовые коэффициенты  k – коэффициент нормировки fi\_p, fi\_g | Осуществляет поиск минимума функции с заданными параметрами с помощью алгоритма роя частиц 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 |

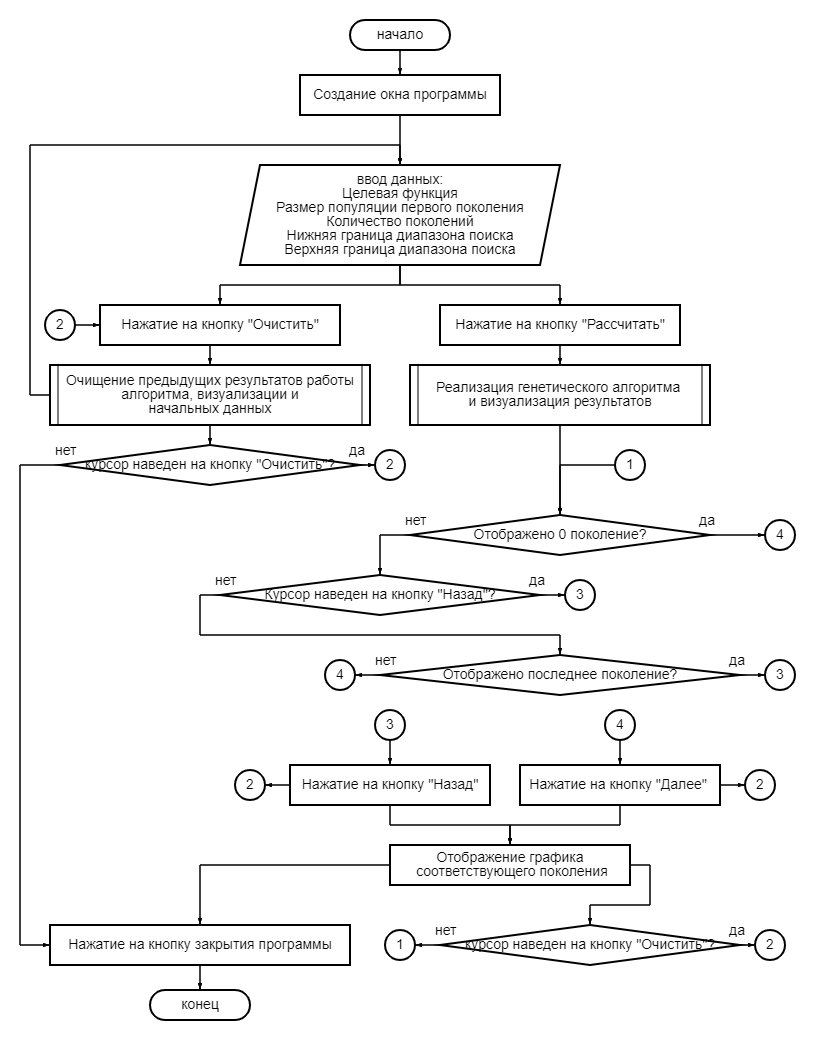
В таблице 4 представлено описание класса UserInput.

*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 |

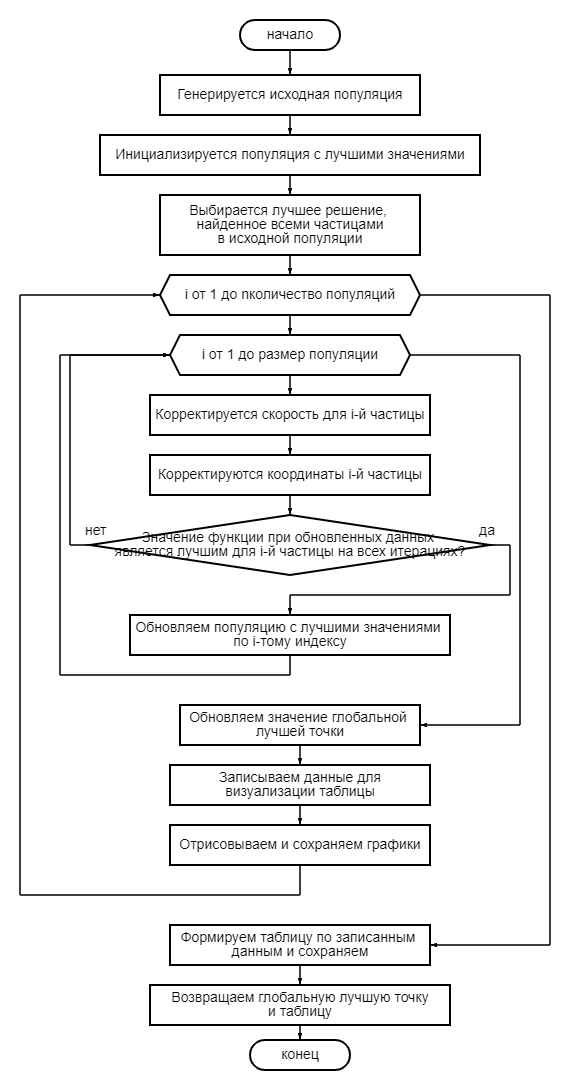
* 1. **Общий ход программы**

На рисунке 1 представлена блок-схема общего хода программы.



*Рисунок 1. Блок-схема общего хода программы*

На рисунке 2 изображен ход подпрограммы, реализующей работу алгоритма роя частиц.



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

* 1. **Оптимизация алгоритма**

Для сокращения количества лишних расчетов, был добавлен дополнительный критерий останова – если значения (x, y), округленные до трех знаков после запятой, повторяются 3 и более раз, то программа останавливает вычисления и за результаты работы считаются (x, y), на которых были остановлены вычисления.

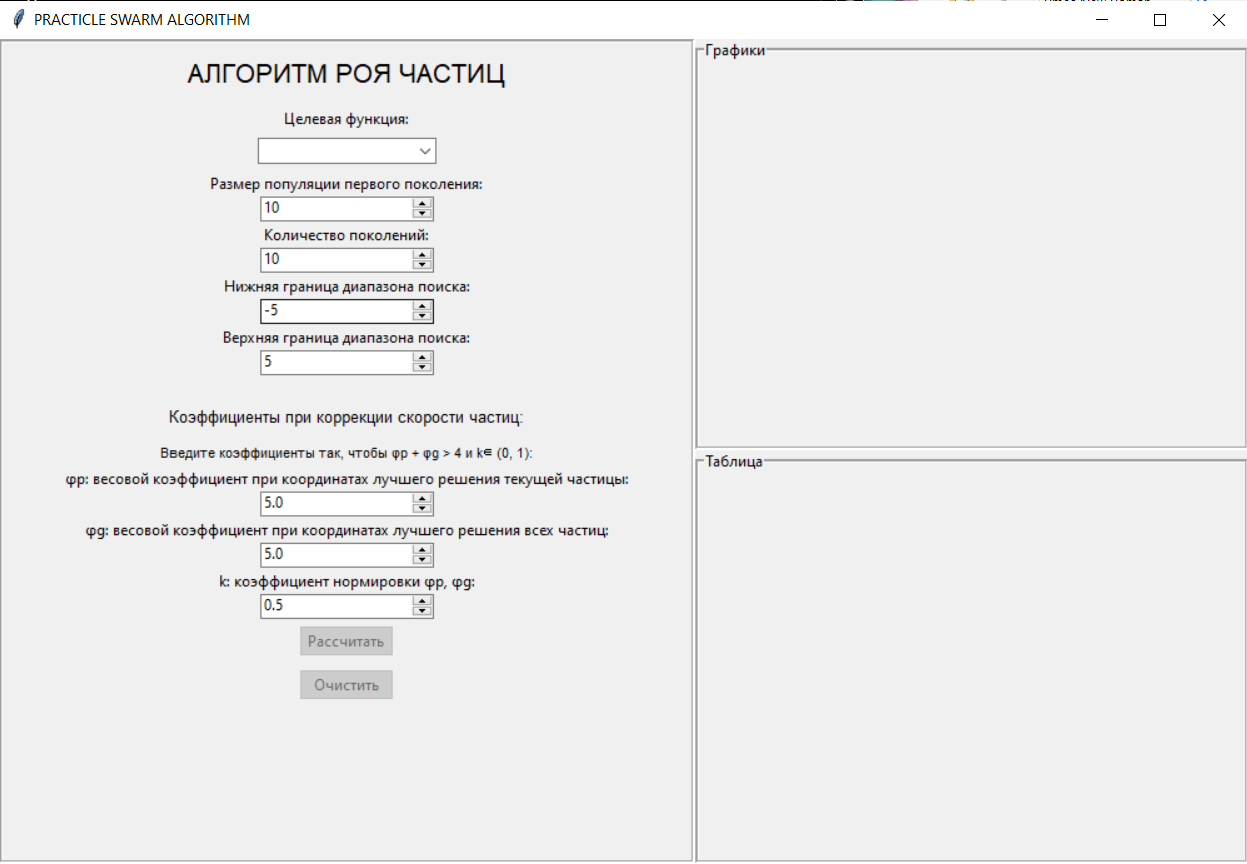
1. **Рекомендации программиста**

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**

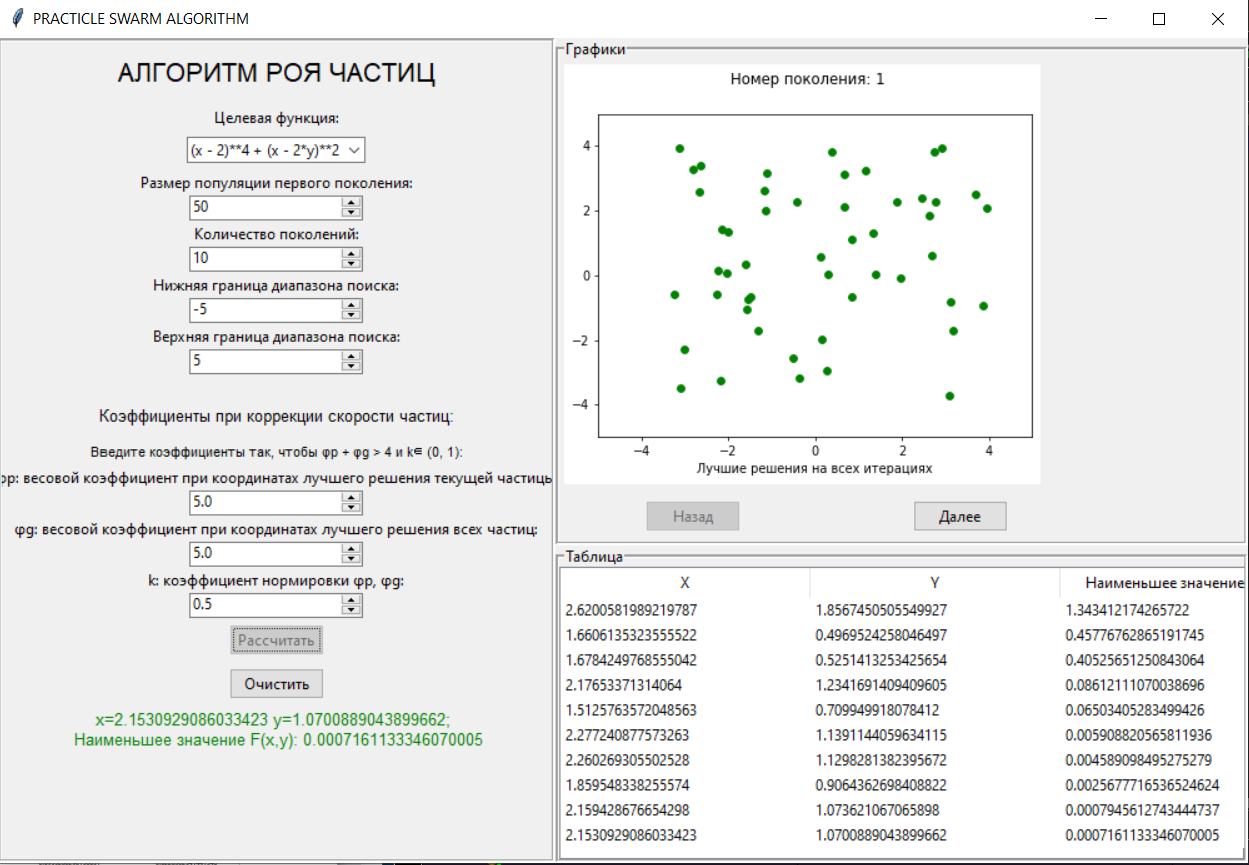
Запустите программу. Перед вами откроется начальное окно программы с уже заполненными начальными значениями, при желании их можно поменять (Рис. 3). Выберите целевую функцию из предложенного списка. После того, как все поля входных данных будут заполнены, активируется кнопка «Рассчитать». Нажмите на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



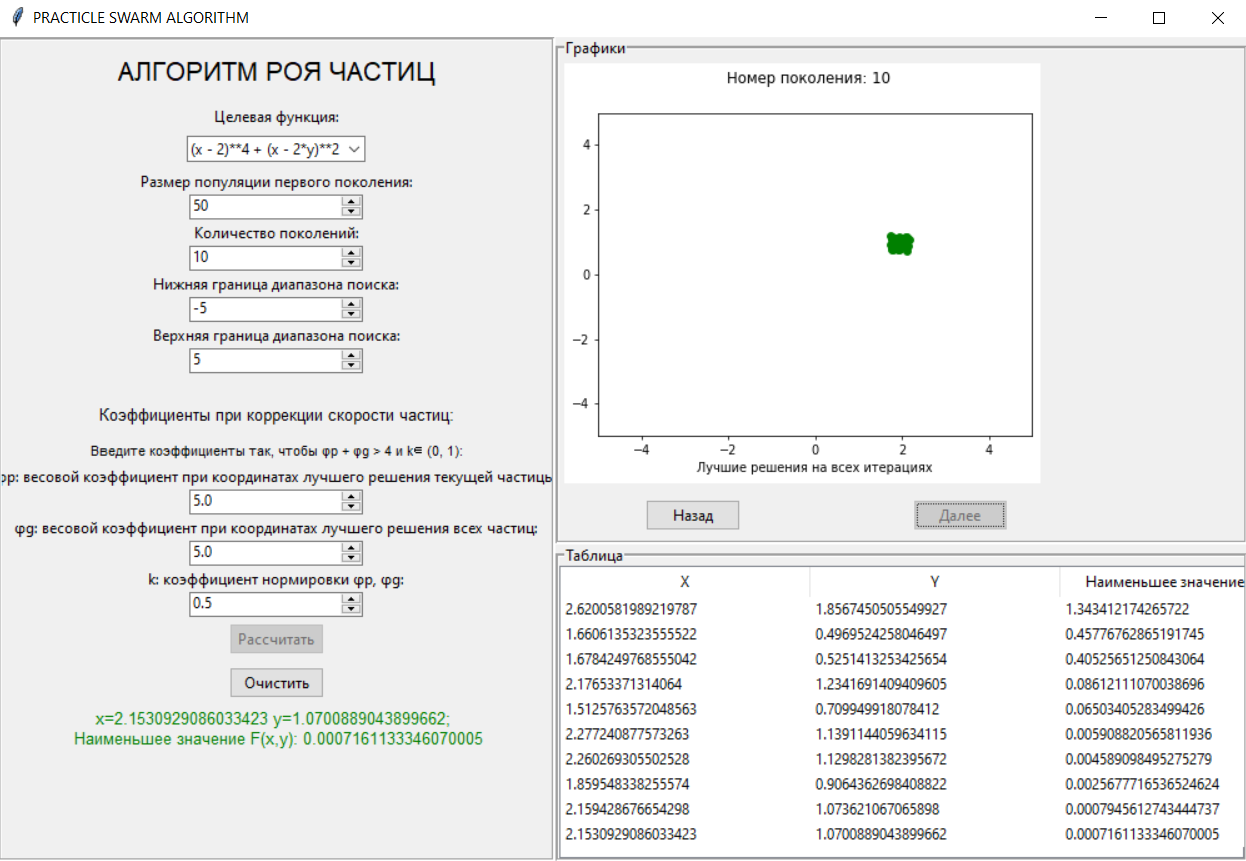
*Рисунок 3. Начальное окно*

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.

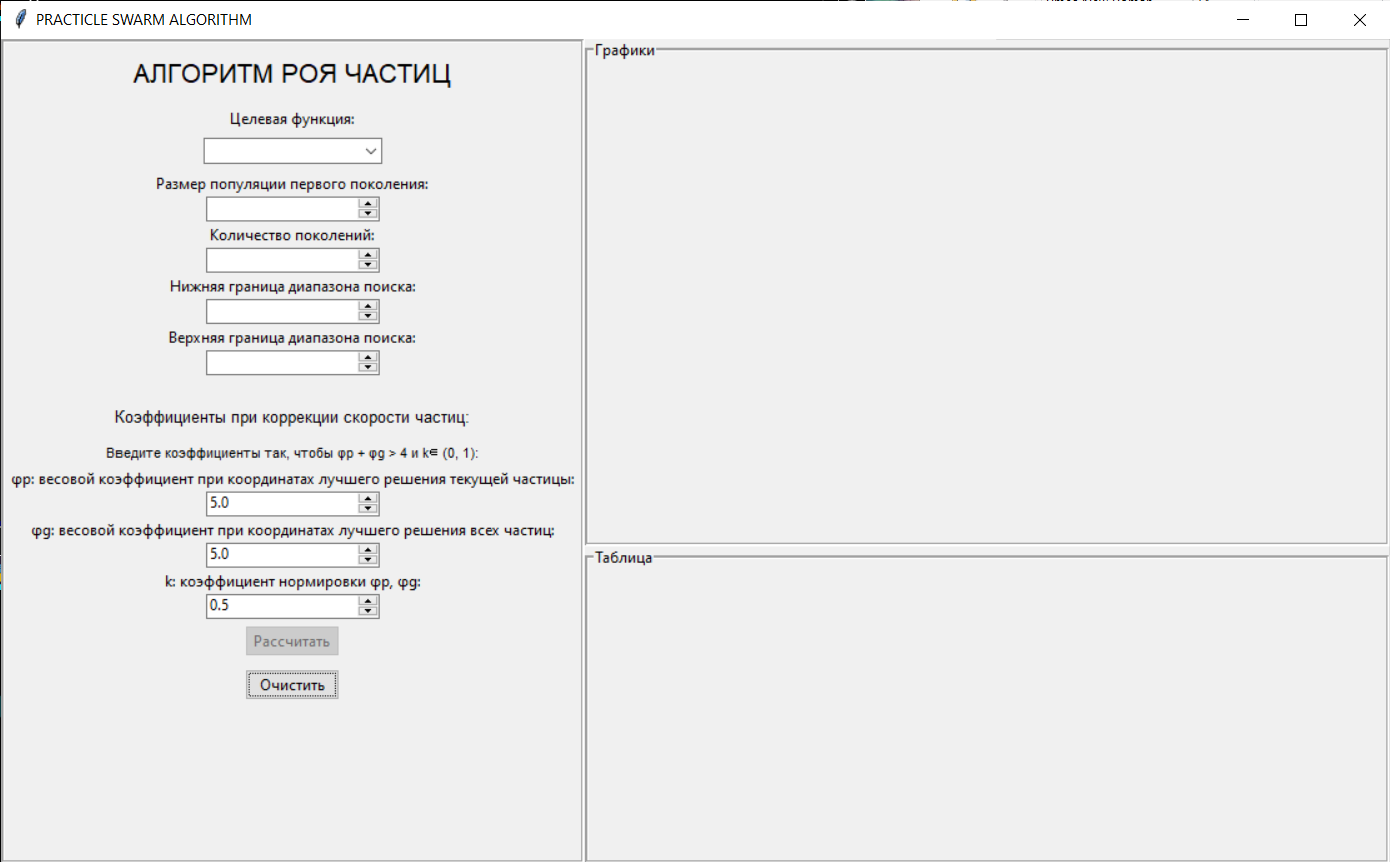


*Рисунок 4. Окно программы после нажатия кнопки «Рассчитать»*



*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.

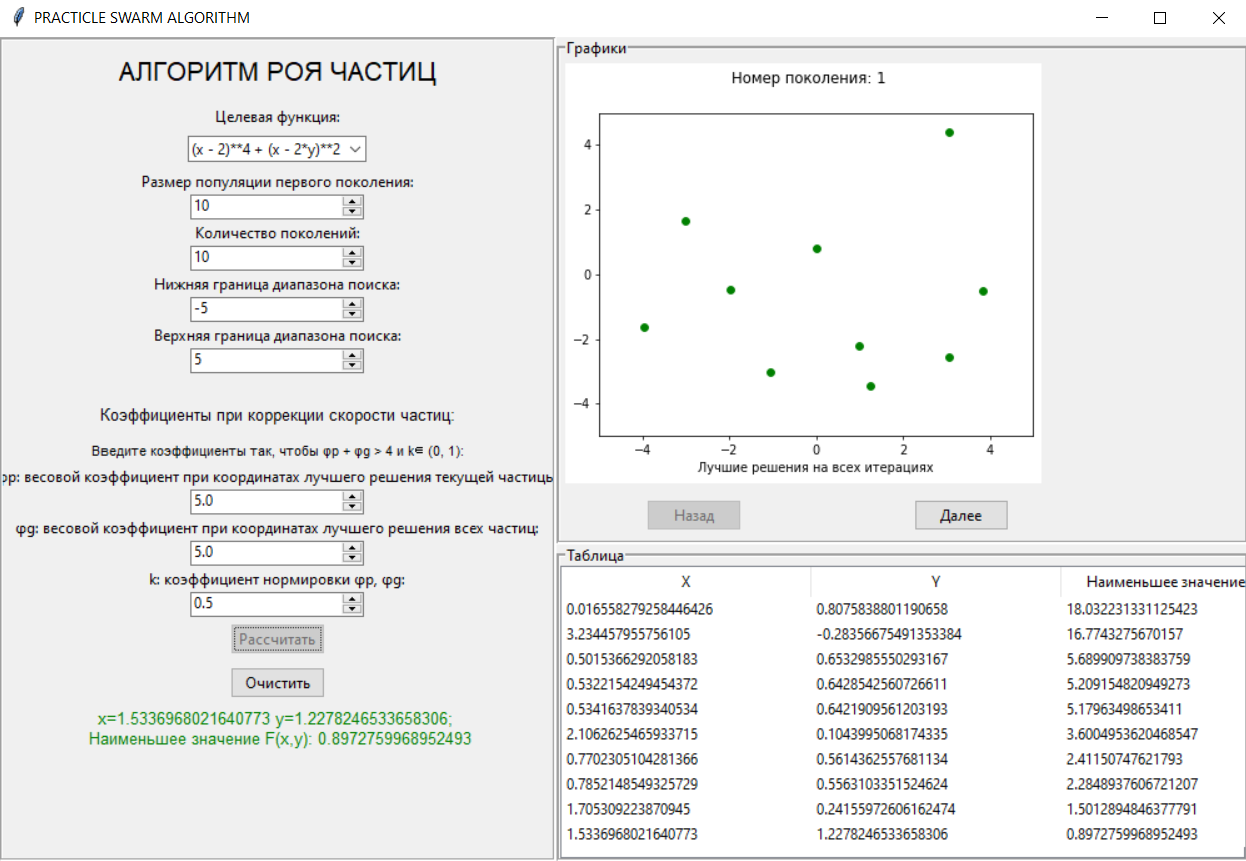


*Рисунок 6. Окно программы после нажатия на кнопку «Очистить»*

1. **Анализ результатов работы алгоритма**

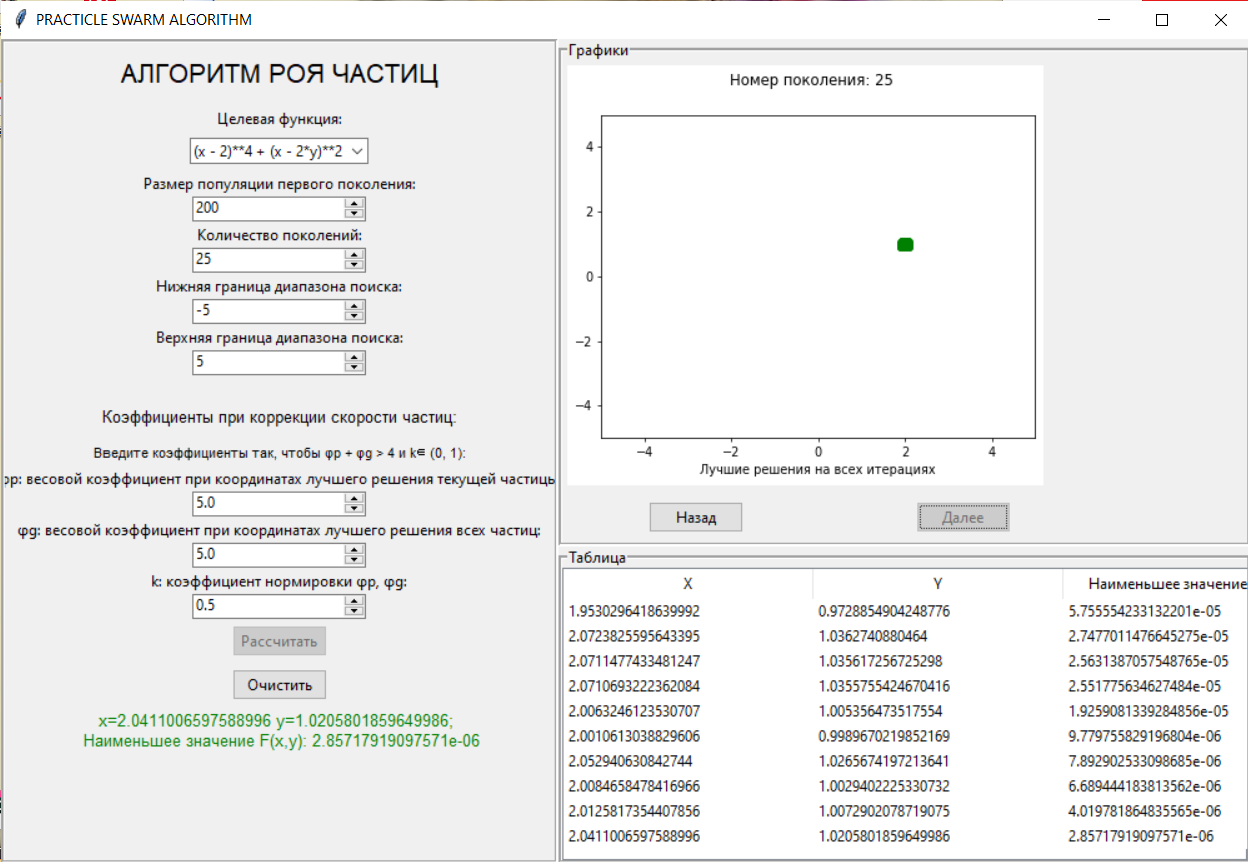
У заданной функции  точкой минимума является точка (2, 1).

При начальных данных минимум равен (1.534, 1.228) – рисунок 7. Так же я запустила программу еще несколько раз с этими данными и получила следующие результаты: (2.413, 0.951), (3.519, 1.259), (2.141, 0.091). Как видно из этих данных у алгоритма низкая сходимость и большая погрешность.



*Рисунок 7. Результат работы программы при начальных данных*

Для увеличения точности выставим размер популяции – 200 частиц и количество поколений до 25. Уже с этими данными результаты запуска программы следующие: (2.256, 1.203), (1.983, 0.993), (2. 006, 1.162), (2.041, 1.021). Как видно, показатели сходимости улучшились, и точность расчетов увеличилась (рис. 8).



*Рисунок 8. Результат работы программы*

Но все-же главное влияние на поведение частиц алгоритма, а следовательно и на его точность, оказывают коэффициенты φp, φg и k. Это обусловоено тем, что они влияют на коррекцию скорости частиц, а именно от этого шага зависит сходимость алгоритма.

Увеличим коэффициент φp до 25, а φg оставим прежним. Результаты при таких входных данных: (2.287, 1.202), (2.491, 1.193), (1.621, 0.647). При этом частицы быстро достигали ответа – за 4-8 итераций, далее значения повторялись. Это оказалось не лучшим расставлением коэффициентов. Действительно, если ориентироваться на весовой коэффициент при координатах локального лучшего решения текущей частицы, то, вероятность того, что ни одна частица не попадет в абсолютный минимум очень велика.

Тогда вернем φp = 5, а φg наоборот поднимем до25. Результаты при таких входных данных: (1.987, 0.994), (1.953, 0.977), (2.015, 1.007), (1.980, 0.990). Алгоритм выдает стабильный результат, так что оставим значения коэффициентов φp, φg в текущем положении.

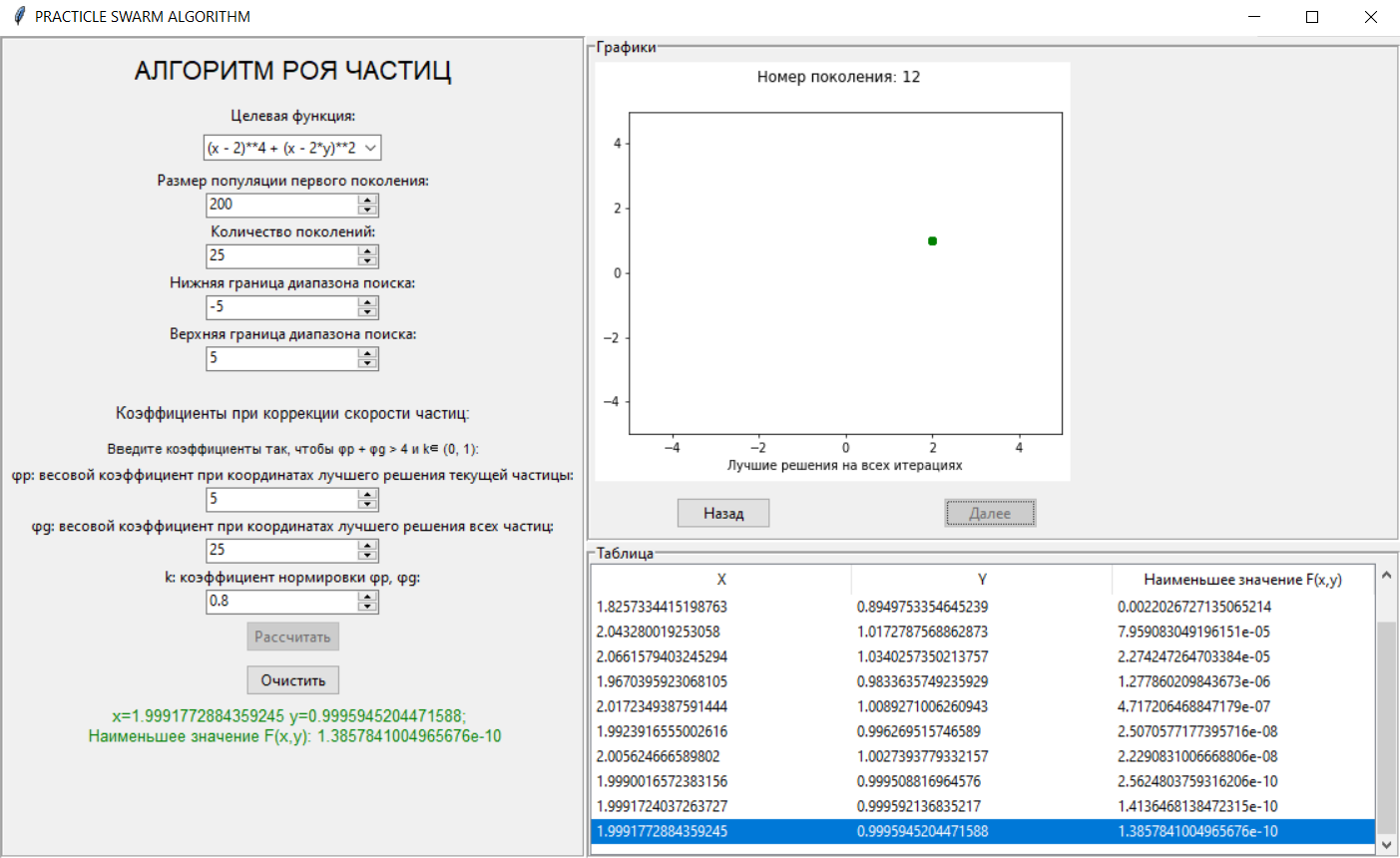
Коффициент k отвечает за нормировку коэффициентов φp, φg, т.е. определяет на сколько сходимость алгоритма будет зависеть от них. Проведем эксперимент, как изменение коэффициента k влияет на точность алгоритма (Таблица 5) при текущих входных данных.

*Таблица 5. Зависимость результатов работы алгоритма от коэффициента k*

|  |  |
| --- | --- |
| Значение коэффициента k | Результат работы алгоритма |
| 0,1 | (2.656, 1.309), (1.431, 0.854), (1.724, 0.886) |
| 0,2 | (2.264, 1.246), (2.105, 1.045), (1.732, 0.761) |
| 0,3 | (1.962, 0.981), (1.933, 0.970), (1.950, 0.975) |
| 0,4 | (2.027, 1.014), (1.933, 0.997), (1.933, 0.997) |
| 0,5 | (1.987, 0.994), (1.953, 0.977), (2.015, 1.007) |
| 0,6 | (1.964, 0.982), (1.979, 0.990), (1.969, 0.985) |
| 0,7 | (1.985, 0.995), (1.923, 0.965), (2.029, 1.015) |
| 0,8 | (2.014, 1.006), (2.011, 1.005), (1.999, 0.999) |
| 0,9 | (2.044, 1.022), (2.078, 1.039), (2.033, 1.110) |

При увеличении k возрастает точность алгоритма. Самые точные значения при k = 0.8.

Таким образом, что минимальная погрешность точности нахождения минимума заданной функции – 0,001. Она достигается при входных данных, введенных на рисунке 9.



*Рисунок 9. Результат работы программы*

В среднем при этих входных данных погрешность равна 0.015.

Сравнение генетического алгоритма и алгоритма роя частиц:

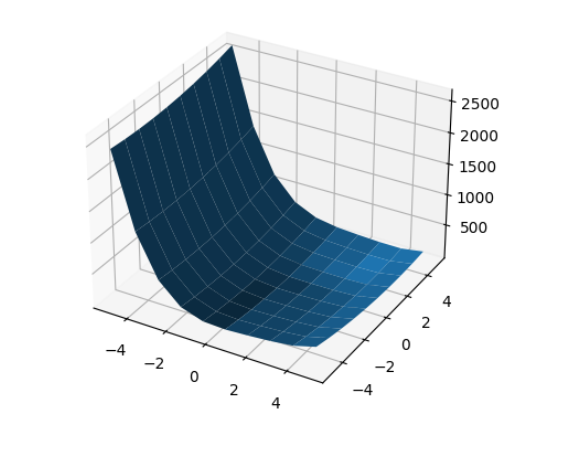
В генетическом алгоритме особи распределяются по первым найденным точкам минимума. Так как он подразумевает случайные изменения, то при каждом новом запуске программы в одной из этих точек может начать доминировать ген, и все поколения будут сходиться в этой точке, и если она не правильная, то результат алгоритма будет неверным.

В алгоритме роя частиц все частицы плавно перемещаются в одну точку – ближе к найденному минимуму.

Генетический алгоритм хуже сходится в одну точку, так как каждую новую итерацию в случайной позиции (между родителями) появляется новая особь, а также случайные мутации, которые не всегда могут попадать в нужную точку. Рой частиц плавно приближался к найденному минимуму.

1. **График исследуемой функции**

На рисунке 10 представлен график исследуемой функции. График был потроен с помощью библиотеки matplotlib, язык программирования Python.



*Рисунок 10. 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 и проанализированы результаты работы программы.

**Приложения**

Приложение А. 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
7. **import** math
9. INPUT\_FUNS **=** ["(x - 2)\*\*4 + (x - 2\*y)\*\*2"]

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

17. **def** generate\_initial\_population(size, start, end):
18. population **=** []
19. **for** i **in** range(size):
20. x **=** random.uniform(start, end)
21. y **=** random.uniform(start, end)
22. v **=** [random.uniform(**-**5, 5), random.uniform(**-**5, 5)]
23. it **=** [x, y, v, func(x, y)]
24. population.append(it)
25. **return** population

28. **def** swarm\_algorithm(num\_generations, population\_size, start, end, fi\_p, fi\_g, k):
29. """
30. :param num\_generations: критерий останова - кол-во итераций алгоритма
31. :param population\_size: размер исходной популяции
32. :param start: нижняя граница поиска
33. :param end: верхняя граница поиска
34. :param fi\_p + fi\_g: > 4
35. :param k: в интервале (0, 1)
36. """
37. stop\_ind **=** **-**1
39. dataForTable **=** []
40. snapshot\_arr **=** []
42. r\_p **=** random.random()
43. r\_g **=** random.random()
44. fi **=** fi\_p **+** fi\_g
45. K **=** 2 **\*** k **/** (abs(2 **-** fi **-** math.sqrt(pow(fi, 2) **-** 4 **\*** fi)))
47. # генерируем исходную популяцию
48. now\_population **=** generate\_initial\_population(population\_size, start, end)
49. best\_population **=** now\_population
50. best\_val **=** sorted(best\_population, key**=lambda** z: z[3])[0]
52. **for** i **in** range(num\_generations):
53. **for** \_ **in** range(population\_size):
54. # корректируем скорость для кажой частицы
55. new\_v\_x **=** K **\*** (best\_population[\_][2][0] **+** fi\_p **\*** r\_p **\***
56. (best\_population[\_][0] **-** now\_population[\_][0]) **+**
57. fi\_g **\*** r\_g **\*** (best\_val[0] **-** now\_population[\_][0]))
58. new\_v\_y **=** K **\*** (best\_population[\_][2][1] **+** fi\_p **\*** r\_p **\***
59. (best\_population[\_][1] **-** now\_population[\_][1]) **+**
60. fi\_g **\*** r\_g **\*** (best\_val[1] **-** now\_population[\_][1]))
61. new\_v **=** [new\_v\_x, new\_v\_y]
62. now\_population[\_][2] **=** new\_v
64. # корректируем координаты каждой частицы
65. now\_population[\_][0] **+=** new\_v\_x
66. now\_population[\_][1] **+=** new\_v\_y
67. now\_population[\_][3] **=** func(now\_population[\_][0], now\_population[\_][1])
69. # ищем лучшее решение для каждой частицы
70. **if** now\_population[\_][3] < best\_population[\_][3]:
71. best\_population[\_] **=** now\_population[\_]
73. # проверяем не нашли ли глобальную лучшую точку
74. best\_val **=** sorted(best\_population, key**=lambda** z: z[3])[0]
76. # записываем данные для изуализации таблицы
77. dataForTable.append([best\_val[0], best\_val[1], best\_val[3]])
79. # Отрисовка и сохранение графика
81. x\_best **=** list(map(**lambda** x: x[0], best\_population))
82. y\_best **=** list(map(**lambda** x: x[1], best\_population))
83. snapshot\_name **=** f"pictures/g{i}.png"
84. fig, ax2 **=** plt.subplots(1, 1)
85. fig.suptitle(f"Номер поколения: {i + 1}")
86. ax2.set\_xlabel("Лучшие решения на всех итерациях")
87. ax2.plot(x\_best, y\_best, 'go')
88. ax2.set\_xlim(start, end)
89. ax2.set\_ylim(start, end)
90. plt.savefig(snapshot\_name, dpi**=**70, bbox\_inches**=**'tight')
91. plt.close()
92. snapshot\_arr.append(snapshot\_name)
94. **if** i > 3:
95. **if** round(dataForTable[i][1], 3) **==** round(dataForTable[i **-** 1][1], 3) **==** \
96. round(dataForTable[i **-** 2][1], 3) **and** \
97. round(dataForTable[i][0], 3) **==** round(dataForTable[i **-** 1][0], 3) **==** \
98. round(dataForTable[i **-** 2][0], 3):
99. stop\_ind **=** i
100. **break**
102. # Сохраняем таблицу
103. data **=** pd.DataFrame(dataForTable)
104. data.columns **=** ['X', 'Y', 'Наименьшее значение F(x,y)']
106. print("Минимум функции в точке(", best\_val[0], best\_val[1], "), значение функции: ",
107. best\_val[3])
109. **return** best\_val, data, stop\_ind

112. # -------------------------------img-and-table-frames-logic--------------------------------------------------------------
113. **class** ShowResult:
114. **def** \_\_init\_\_(self, data, stop\_ind):
115. # Счётчик для изображений
116. self.i **=** 0
117. **if** stop\_ind **==** **-**1:
118. self.max\_i **=** int(num\_generations.item.get()) **-** 1
119. **else**:
120. self.max\_i **=** stop\_ind
121. # Рамка для наших изображений
122. self.canvas **=** Canvas(frame\_btn, width**=**420, height**=**345, borderwidth**=**2)
124. # Добавим 1-ое изображение
125. self.photo **=** None
126. self.show\_img()
128. # Кнопки для перелистывания изображений
129. self.button\_next **=** ttk.Button(frame\_btn, text**=**"Далее", command**=**self.next\_img, state**=**NORMAL)
130. self.button\_next.grid(row**=**1, column**=**1)
131. self.button\_back **=** ttk.Button(frame\_btn, text**=**"Назад", command**=**self.previous\_img, state**=**DISABLED)
132. self.button\_back.grid(row**=**1, column**=**0)
134. # Выведем таблицу
135. self.data **=** data
136. self.show\_table()
138. **def** next\_img(self) **-**> None:
139. self.i **+=** 1
140. self.button\_back['state'] **=** NORMAL
141. self.show\_img()
142. **if** self.i **==** self.max\_i:
143. self.button\_next['state'] **=** DISABLED
145. **def** previous\_img(self) **-**> None:
146. self.i **-=** 1
147. self.button\_next['state'] **=** NORMAL
148. self.show\_img()
149. **if** self.i **==** 0:
150. self.button\_back['state'] **=** DISABLED
152. **def** show\_img(self):
153. self.photo **=** ImageTk.PhotoImage(Image.open("pictures/g{0}.png".format(self.i)))
154. self.canvas.create\_image(3, 3, anchor**=**'nw', image**=**self.photo)
155. self.canvas.grid(column**=**0, row**=**0, columnspan**=**2)
157. **def** show\_table(self) **-**> None:
158. # Создаем виджет таблицы
159. treeview **=** ttk.Treeview(frame\_scroll, columns**=**list(self.data.columns), show**=**'headings')
160. treeview.pack(side**=**LEFT, fill**=**BOTH, expand**=**1)
161. # Добавляем колонки в таблицу
162. **for** col **in** self.data.columns:
163. treeview.heading(col, text**=**col)
164. # Добавляем строки в таблицу и заполняем значения ячеек данными из дата-фрейма
165. **for** i, row **in** self.data.iterrows():
166. treeview.insert('', END, values**=**list(row))
167. # Создаем scrollbar widget и добавляем его в таблицу
168. scrollbar **=** ttk.Scrollbar(frame\_scroll, orient**=**VERTICAL, command**=**treeview.yview)
169. scrollbar.pack(side**=**RIGHT, fill**=**Y)
170. treeview.configure(yscrollcommand**=**scrollbar.set)

173. # ------------------------------------------------buttons-logic----------------------------------------------------------
175. **def** checkButtonState(**\***args) **-**> None:
176. """ Кнопка активируется когда все атрибуты input введены """
177. **if** num\_generations.item.get() **and** population\_size.item.get() **and** \
178. start.item.get() **and** end.item.get() **and** combobox:
179. btn\_start['state'] **=** NORMAL
180. **else**:
181. btn\_start['state'] **=** DISABLED
182. **if** num\_generations.item.get() **or** population\_size.item.get() **or** \
183. start.item.get() **or** end.item.get() **or** combobox:
184. btn\_clean['state'] **=** NORMAL

187. **def** clickButton() **-**> None:
188. """ По нажатию кнопки производятся расчёты """
189. btn\_start['state'] **=** DISABLED
190. xy\_bestScore, data, stop\_ind **=** \
191. swarm\_algorithm(population\_size**=**int(population\_size.item.get()),
192. num\_generations**=**int(num\_generations.item.get()),
193. start**=**min(int(start.item.get()), int(end.item.get())),
194. end**=**max(int(start.item.get()), int(end.item.get())),
195. fi\_p**=**float(fi\_p.item.get()),
196. fi\_g**=**float(fi\_g.item.get()),
197. k**=**float(k.item.get()))
199. # Перед тем как сгенерировать новые виджеты, удалим старые
200. **for** widget **in** frame\_input.winfo\_children():
201. **if** widget.winfo\_name() **==** "result\_label":
202. widget.destroy()
203. **for** widget **in** frame\_btn.winfo\_children():
204. widget.destroy()
205. **for** widget **in** frame\_scroll.winfo\_children():
206. widget.destroy()
208. # Создаём новые
209. ShowResult(data, stop\_ind)
210. text **=** "x={0} y={1}; \n Наименьшее значение F(x,y): {2}".format(xy\_bestScore[0],
211. xy\_bestScore[1],
212. xy\_bestScore[3])
213. Label(frame\_input, text**=**text, fg**=**'green', font**=**('Arial', 10), name**=**"result\_label").pack()

216. **def** cleanButton() **-**> None:
217. """ По нажатию кнопки удаляем старые данные """
218. population\_size.item.delete(0, END)
219. num\_generations.item.delete(0, END)
220. start.item.delete(0, END)
221. end.item.delete(0, END)
222. combobox.set('')
223. **for** widget **in** frame\_input.winfo\_children():
224. **if** widget.winfo\_name() **==** "result\_label":
225. widget.destroy()
226. **for** widget **in** frame\_btn.winfo\_children():
227. widget.destroy()
228. **for** widget **in** frame\_scroll.winfo\_children():
229. widget.destroy()

232. # ---------------------------------------user-input----------------------------------------------------------------------
234. **class** UserInput:
235. """ Класс определяющий шаблон для атрибутов ввода tkinter """
237. **def** \_\_init\_\_(self, text, from\_, to, increment, initial\_value):
238. # Текстовые переменные tkinter
239. self.var **=** StringVar()  # текст вводимый пользователем
240. self.var.set(initial\_value)
242. # Определяем структуру объекта ввода
243. Label(frame\_input, text**=**text).pack()
244. self.item **=** ttk.Spinbox(frame\_input, validate**=**'key', textvariable**=**self.var,
245. from\_**=**from\_, to**=**to, increment**=**increment)
246. self.item.pack()
247. self.var.trace('w', checkButtonState)

250. # -------------------------------------------------UI--------------------------------------------------------------------
252. # Создаётся окно пользователя
253. root\_window **=** Tk()
254. root\_window.title("PRACTICLE SWARM ALGORITHM")
255. root\_window.geometry('1000x660')
257. # Создание фреймов и настройка их свойств
258. frame\_input **=** Frame(root\_window, bd**=**3, relief**=**GROOVE)
259. Label(frame\_input, text**=**"АЛГОРИТМ РОЯ ЧАСТИЦ", font**=**('Arial', 16)).pack(pady**=**10)
260. frame\_input.grid(column**=**0, row**=**0, rowspan**=**2, sticky**=**'NSEW')
262. frame\_img **=** LabelFrame(root\_window, bd**=**3, relief**=**GROOVE, text**=**"Графики")
263. frame\_img.grid(column**=**1, row**=**0, sticky**=**'NSEW')
264. frame\_table **=** LabelFrame(root\_window, bd**=**3, relief**=**GROOVE, text**=**"Таблица")
265. frame\_table.grid(column**=**1, row**=**1, sticky**=**'NSEW')
267. # Дополнительные мини-фреймы
268. frame\_btn **=** Frame(frame\_img)
269. frame\_btn.pack(fill**=**BOTH, expand**=**1)
270. frame\_scroll **=** Frame(frame\_table)
271. frame\_scroll.pack(fill**=**BOTH, expand**=**1)
273. # Задание параметров для столбцов и строк
274. root\_window.columnconfigure(0, weight**=**1)
275. root\_window.columnconfigure(1, weight**=**4)
276. root\_window.rowconfigure(0, weight**=**1)
277. root\_window.rowconfigure(1, weight**=**1)
279. # Ввод функции которую мы хотим исследовать
280. Label(frame\_input, text**=**"Целевая функция:").pack()
281. var\_txt **=** StringVar()
282. combobox **=** ttk.Combobox(frame\_input, textvariable**=**var\_txt)
283. combobox['values'] **=** INPUT\_FUNS
284. combobox['state'] **=** 'readonly'
285. combobox.pack(padx**=**5, pady**=**5)
286. var\_txt.trace('w', checkButtonState)
288. # Ввод данных
289. population\_size **=** UserInput(text**=**"Размер популяции первого поколения:", initial\_value**=**"10", from\_**=**2, to**=**999,
290. increment**=**1)
291. num\_generations **=** UserInput(text**=**"Количество поколений:", initial\_value**=**"10", from\_**=**1, to**=**999, increment**=**1)
292. start **=** UserInput(text**=**"Нижняя граница диапазона поиска:", initial\_value**=**"-5", from\_**=-**99999, to**=**99999, increment**=**1)
293. end **=** UserInput(text**=**"Верхняя граница диапазона поиска:", initial\_value**=**"5", from\_**=-**99999, to**=**99999, increment**=**1)
295. Label(frame\_input, text**=**"", font**=**('Arial', 10)).pack()
296. Label(frame\_input, text**=**"Коэффициенты при коррекции скорости частиц:", font**=**('Arial', 10)).pack()
297. Label(frame\_input, text**=**"", font**=**('Arial', 1)).pack()
298. Label(frame\_input, text**=**"Введите коэффициенты так, чтобы φp + φg > 4 и k∈(0, 1):", font**=**('Arial', 8)).pack()
299. fi\_p **=** UserInput(text**=**"φp: весовой коэффициент при координатах лучшего решения текущей частицы:", initial\_value**=**"5.0",
300. from\_**=-**99999.0, to**=**99999.0, increment**=**1.0)
301. fi\_g **=** UserInput(text**=**"φg: весовой коэффициент при координатах лучшего решения всех частиц:", initial\_value**=**"5.0",
302. from\_**=-**99999.0, to**=**99999.0, increment**=**1.0)
303. k **=** UserInput(text**=**"k: коэффициент нормировки φp, φg:", from\_**=**0.001, to**=**0.999, increment**=**0.01, initial\_value**=**"0.5")
305. # Запуск программы
306. btn\_start **=** ttk.Button(frame\_input, text**=**"Рассчитать", command**=**clickButton, state**=**DISABLED)
307. btn\_start.pack(padx**=**5, pady**=**5)
309. # Очистить старые значения
310. btn\_clean **=** ttk.Button(frame\_input, text**=**"Очистить", command**=**cleanButton, state**=**DISABLED)
311. btn\_clean.pack(padx**=**5, pady**=**5)
313. root\_window.mainloop()