МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

Національний технічний університет
«Харківський політехнічний інститут»
Кафедра КМАД

ЗВІТ З ЛАБОРАТОРНОЇ РОБОТИ №0

Виконав студент:

Омельніцький Андрій Миколайович
Група: КН-120А

Перевірила:

Ольга Василівна Костюк

Харків, 2023 г.

Зміст

Глава 1

Мета роботи

Вивчення методів побудови ліній рівня функції двох змінних, розробка програмного модуля для візуалізації оптимізаційної задачі.

Порядок виконання:

1. Розробити власний програмний модуль, який дозволяє графічно зобразити задану кількість ліній рівня довільної функції двох змінних при заданих діапазонах обмежень.
2. Доповнити модуль можливістю візуалізації траєкторії методу - ламаної, що задана набором точок.
3. Використовуючи розроблений модуль, навести графіки ліній рівня функцій, що наведені нижче, відмітити точку екстремуму цієї функції. Передбачити можливість побудови графіку для довільних значень кількості ліній рівня та діапазонах обмежень.
4. Отримані результати занести до звіту, зробити висновки.

Глава 2

Програмна реалізація

2.1. Технології

Програму реалізовано за допомогою мови програмування Python та її модулів. Модулі: PyQt5(Для ітрерфейсу), matplotlib(Для відображення ліній рівня). Для знаходження точки екстремуму був обраний алгоритм градіентрого спуску.

2.2. Інтерфейс

Інтерфейс програми представлений на рис 2.1 містить: область для зображення ліній рівня, поле для вводу вункції, поля для завдання обмежень, поле для завдання кількості ліній рівня, поле для вибору типа відображення ліній рівля, кнопки для відображення ліній рівня після завдяння данніих. Також програма має можливість будувати шлях від завданної точки шляхом подвійного натискання лівої кнопки миші на графику до точки екстремуму.

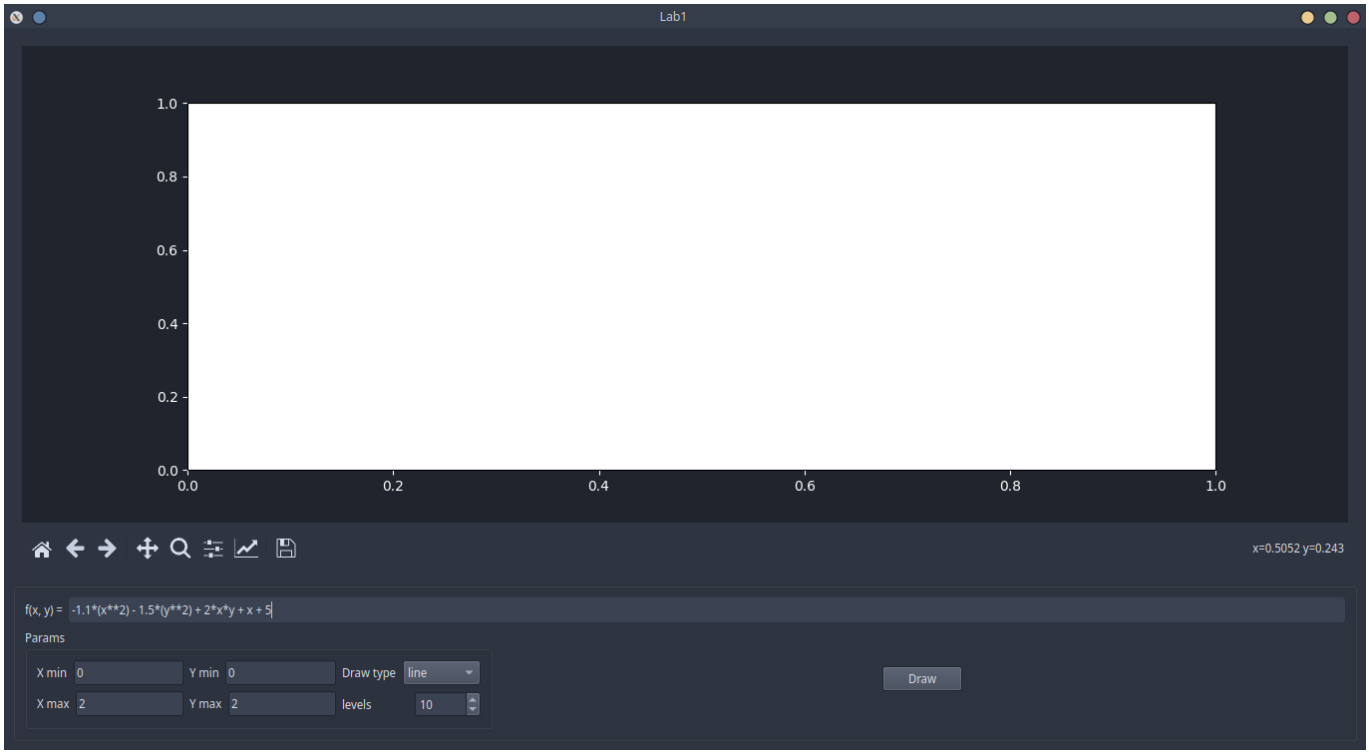

Рис. 2.1. Інтерфейс програми

Глава 3

Результати роботи програми

3.1. Приклад роботи програми

Застосуемо програму для функції $f(x, y) = -1.1x^2 - 1.5y^2 + 2xy + x + 5$ та прорахуему екстремум функциі із точки $x_0 = 0.23624, y_0 = 1.54007$. Отримаемо зображення ліній рівня рис 3.1 та результату роботи алгоритву пошуку екстремуму т. єкстремуму: $x = 1.15385, y = 0.76923$.

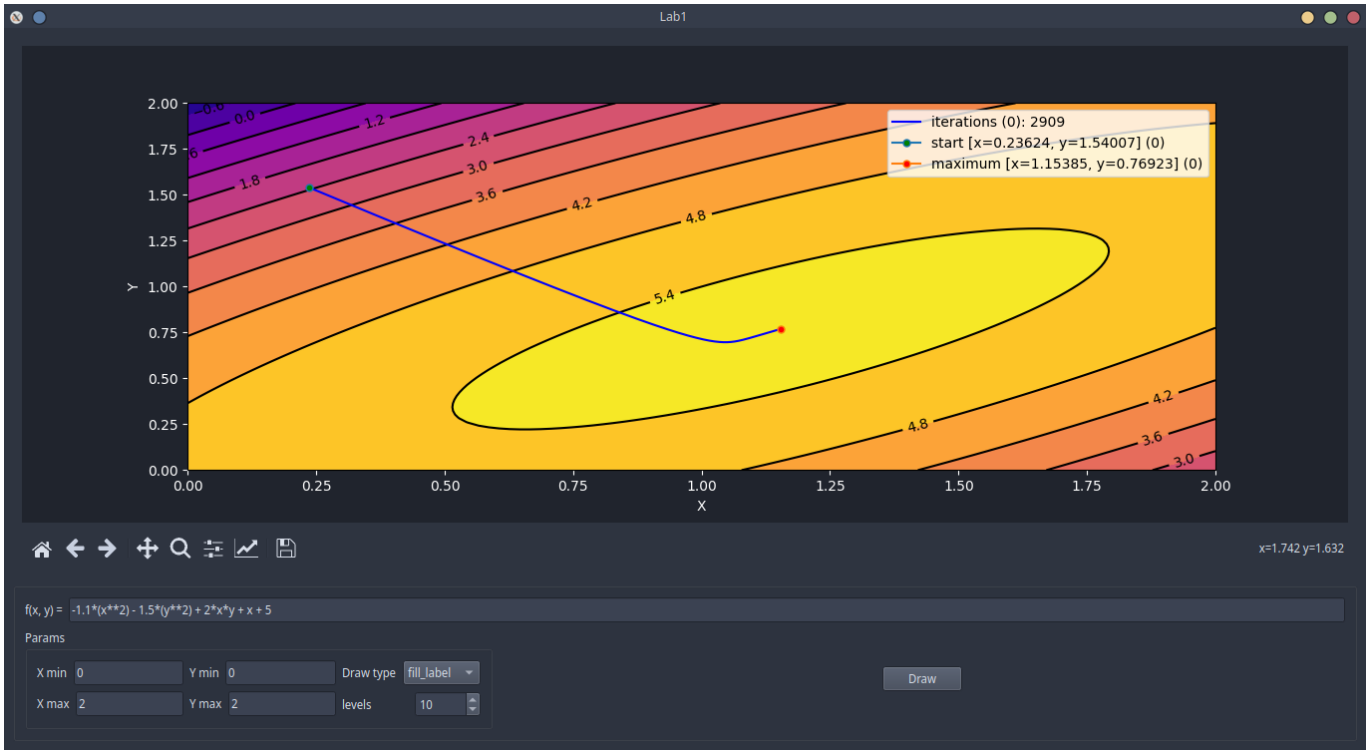

Рис. 3.1. Приклад 1

3.2. Ілюстрації роботи програми

Тепер застосуємо програму для функції $f(x, y) = x\sin(4\pi x)+y\sin(4\pi y)$, різних значень обмежень x та y та різною кількістю ліній рівня.

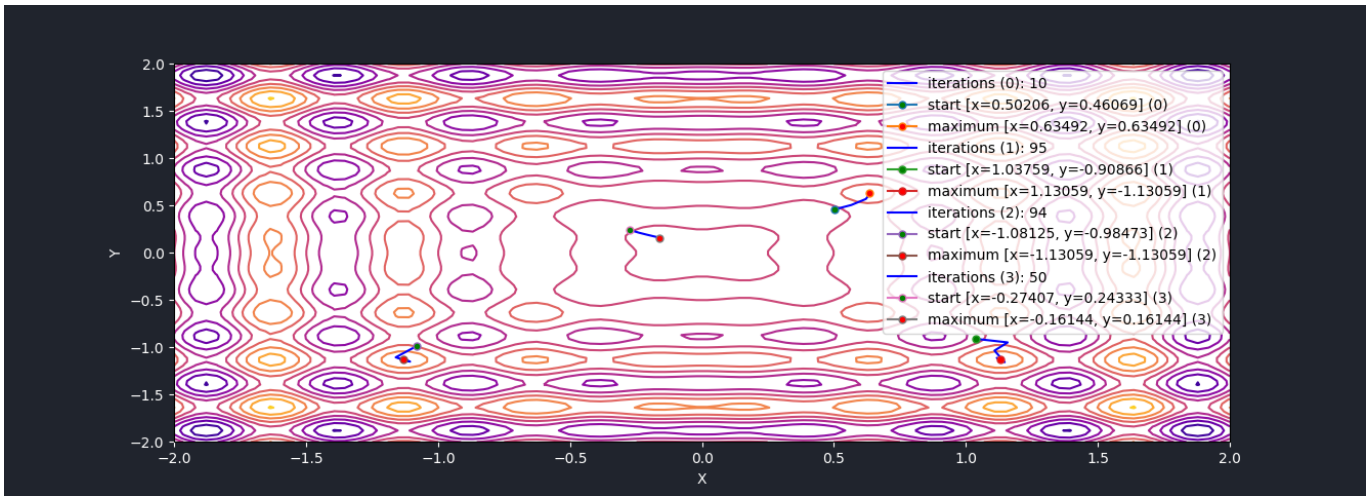

Рис. 3.2. $x, y \in [-2; 2]$ lines $= 10$

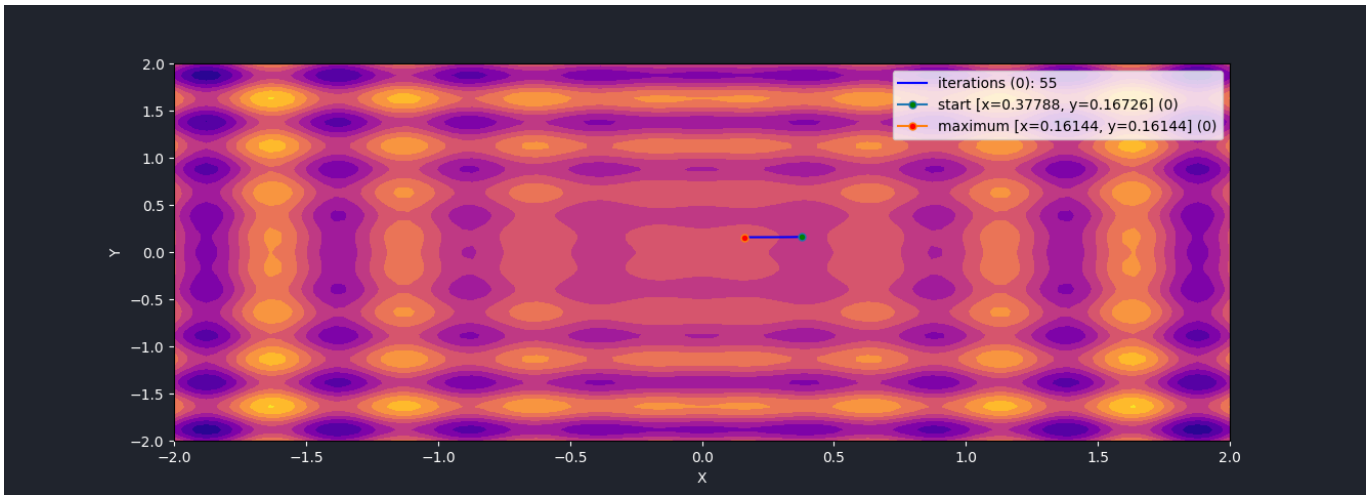

Рис. 3.3. $x, y \in [-2; 2]$ lines $= 10$

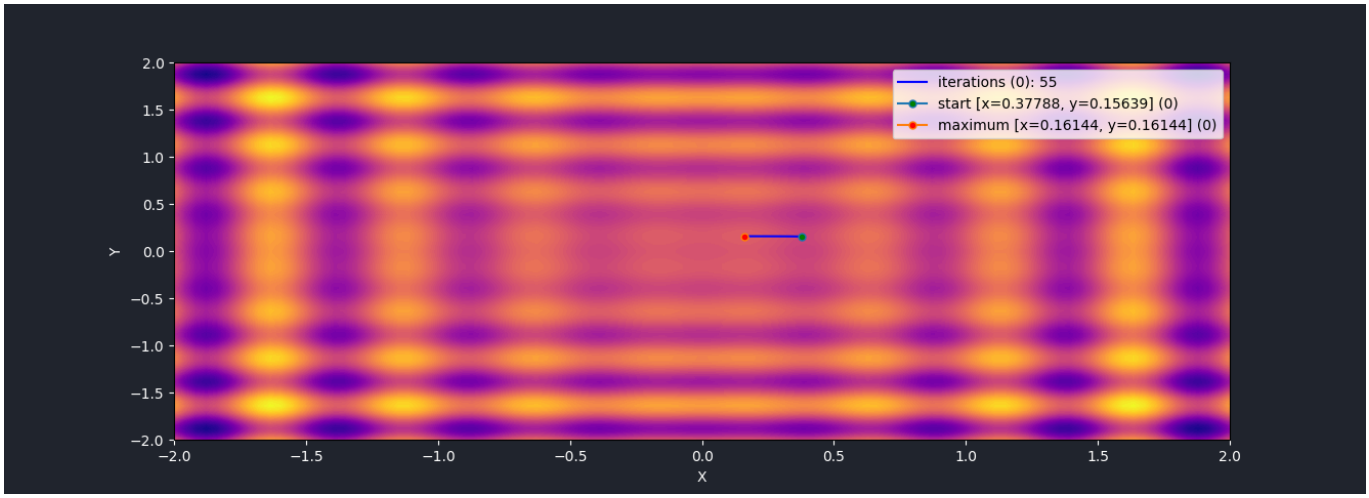

Рис. 3.4. $x, y \in [-2; 2]$ lines $= 30$

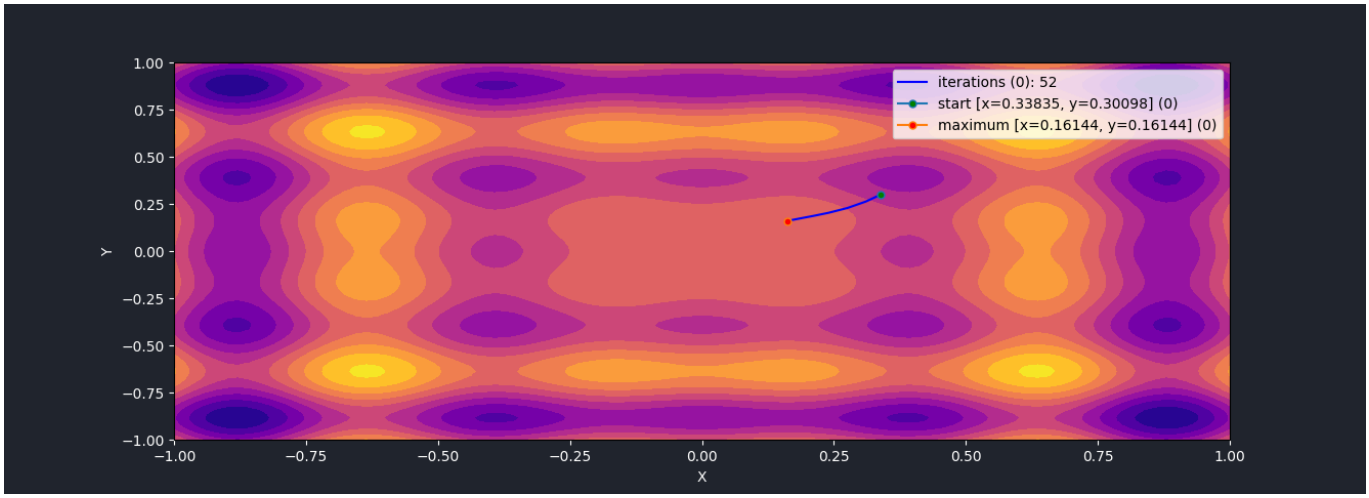

Рис. 3.5. $x, y \in [-1; 1]$ lines $= 10$

3.3. Аналіз роботи програми

Отже можна помітити, що чим краще обмежити зачення x та y, то тим кращім буде зображення локальних екстремумів. Як приклад можна подивитися рис. 3.3 та рис. 3.5, де й видно що при обмеженні $x, y \in [-1; 1]$ на рис. 3.5 краще зображено положення локальних екстремумів ніж при обмеженні $x, y \in [-2; 2]$ на рис. 3.3. Також можна помітити в порівнянні рис. 3.3 з рис. 3.4 та рис. 3.5 з рис. 3.6, що чим біша кількість ліній рівня, то ти краще зображено положення локальних екстремумів.

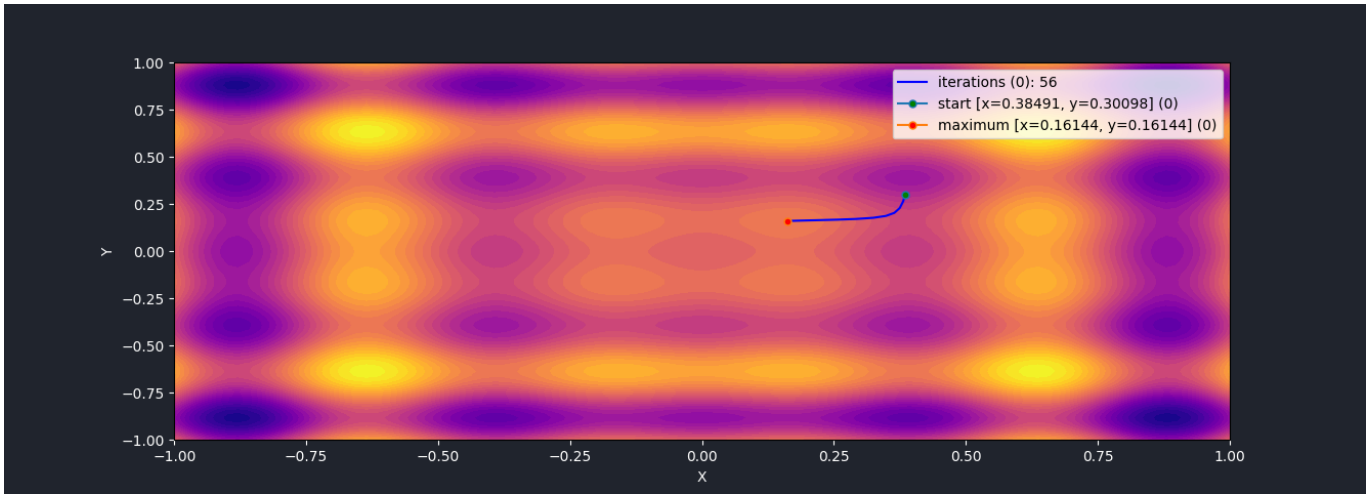

Рис. 3.6. $x, y \in [-1; 1]$ lines $= 30$

Глава 4

Висновок

У ході лабораторної работи було розроблено програму для побудови ліній рівня функції двох змінних та візуалізації оптимізаційної задачі. Як приклад роботи програми було розглянуто функції $f(x, y) = -1.1x^2 - 1.5y^2 + 2xy + x + 5$ і $f(x, y) = x\sin(4\pi x) + y\sin(4\pi y)$, та знайдено локальні екстремуми цих функції за допомогую методу градієнтного спуску. Також було проведено порівння результатів програми за різних занченнях обмежень x та y та різніх значень кількості ліній рівня.

Глава 5

Код програми

5.1. main.py

```
import matplotlib; matplotlib.use('Qt5Agg')
import numpy as np
import math
from PyQt5.QtWidgets import QApplication, QMainWindow,
  QWidget, QVBoxLayout
import sip
from matplotlib.backends.backend_qt5agg import
  FigureCanvasQTAgg as FigureCanvas
from matplotlib.backends.backend_qt5agg import
  NavigationToolbar2QT as NavigationToolbar
from matplotlib.figure import Figure
import matplotlib.pyplot as plt
import sympy
from numpy import pi


# import UI
from py_UI.app_ui import Ui_MainWindow

from count_expression import count_expression
from count_diff import count_function_diff, get_grad_func



def input_expression(help_str='Input␣expression:␣'):
    data_str = input(help_str)
    return data_str


def input_params():
    from numpy import pi
    xmin, xmax = map(eval, input('Input␣xmin,␣xmax:␣').
      split(',␣'))
```

```
30      ymin, ymax = map(eval, input('Input ymin, ymax: ').
          split(', '))
31      x, y = np.mgrid[xmin:xmax:100j, ymin:ymax:100j]
32      params = {}
33      params[0] = {}
34      params[0]['name'] = 'x'
35      params[0]['value'] = x
36      params[1] = {}
37      params[1]['name'] = 'y'
38      params[1]['value'] = y
39      return params
40
41
42  def plot_levels(fig, ax, params, data, levels=5, *,
      contour_type='line_label'):
43      '''
44      contour_type = 'line' | 'fill' | 'line_label' | '
          fill_label'
45      '''
46
47      x, y = params[0]['value'], params[1]['value']
48
49      # fig, ax = plt.subplots()
50
51      match contour_type:
52          case 'line':
53              ax.contour(x, y, data, levels=levels, cmap='
                  plasma')
54          case 'line_label':
55              cs = ax.contour(x, y, data, levels=levels,
                  cmap='plasma')
56              ax.clabel(cs)
57          case 'fill':
58              ax.contourf(x, y, data, levels=levels, cmap='
                  plasma')
59          case 'fill_label':
60              color_line = np.zeros((levels, 3))
61              c = np.linspace(1, 0.2, levels)
62              # color_line[:, 0] = c
63              # color_line[:, 1] = c
64              # color_line[:, 2] = c
```

```
65              color_line[:, 0] = 0
66              color_line[:, 1] = 0
67              color_line[:, 2] = 0
68              ax.contourf(x, y, data, levels=levels, cmap='
                  plasma')
69              cs = ax.contour(x, y, data, levels=levels,
                  colors=color_line)
70              ax.clabel(cs)
71
72      fig.set_figwidth(5)
73      fig.set_figheight(5)
74
75
76  def find_extremum(grad_fun, init, n_iterations, eta, b, *,
        eps=1e-6):
77      points = [np.array(init)]
78      v = 0
79      for k in range(n_iterations):
80          # print(k)
81          x = points[-1]
82          # print(x)
83          x = x + v * b - eta * grad_fun(x[0] + b * v, x[1]
                + b * v)
84
85          dist = math.hypot(points[-1][0] - x[0], points
                [-1][1] - x[1])
86          points.append(x)
87          if dist < eps:
88              break
89
90      return np.array(points).T
91
92
93  def plot_path(ax, points, label, color='r'):
94      x, y = points[0], points[1]
95      ax.plot(x, y, color=color, label=label)
96
97
98  class GraphicWidget(QWidget):
99      def __init__(self, parent=None):
100         QWidget.__init__(self, parent)
```

```
101
102         self.plot_layout = QVBoxLayout(self)
103
104         self._figure = Figure(facecolor='#20242d')
105         self._canvas = FigureCanvas(self._figure)
106         self._axis = self._figure.add_subplot(111)
107         self._init_axis()
108         self._canvas.draw()
109         self.toolbar = NavigationToolbar(self._canvas,
              self)
110
111         self.plot_layout.addWidget(self._canvas)
112         self.plot_layout.addWidget(self.toolbar)
113
114         self.levels = 10
115         self.contour_type = 'line'
116         self.max_count_init = [0, 0]
117
118         self.is_legend = True
119         self.number_of_max_points = 0
120
121         self._canvas.mpl_connect('button_press_event',
              self.mpl_on_click)
122
123     def mpl_on_click(self, event):
124         if event.dblclick:
125             x, y = event.xdata, event.ydata
126             points = self.count_max(x, y)
127             self.plot_path(points)
128
129     def _init_axis(self):
130         color = '#ffffff'
131
132         self._axis.tick_params(axis='x', colors=color)
133         self._axis.tick_params(axis='y', colors=color)
134
135         self._axis.xaxis.label.set_color(color)
136         self._axis.yaxis.label.set_color(color)
137
138     def _clear_plot(self):
139         self.number_of_max_points = 0
```

```
140         self._axis.clear()
141
142     def update_plot(self):
143         self._clear_plot()
144         self._axis.set_xlabel('X')
145         self._axis.set_ylabel('Y')
146
147         params = self._get_params()
148         data = self._count_data()
149         plot_levels(self._figure, self._axis, params, data
                , self.levels, contour_type=self.contour_type)
150
151         self._canvas.draw()
152         self._canvas.flush_events()
153
154     def set_grid(self, grid):
155         self.x, self.y = grid
156
157     def set_exp(self, expression):
158         self.expression = expression
159
160     def plot_path(self, points):
161         self._axis.plot(
162             points[0], points[1],
163             color="blue",
164             label=f'iterations ({self.number_of_max_points
                }): {points.shape[1]}'
165         )
166         self._axis.plot(
167             points[0][0], points[1][0],
168             marker="o",
169             markersize=5,
170             markerfacecolor="green",
171             label=f"start [x={points[0][0]:.5f}, y={points
                [1][0]:.5f}] ({self.number_of_max_points})"
172         )
173         self._axis.plot(
174             points[0][-1], points[1][-1],
175             marker="o",
176             markersize=5,
177             markerfacecolor="red",
```

```
178             label=f"maximum [x={points[0][-1]:.5f}, y={points[1][-1]:.5f}] ({self.number_of_max_points})"
179         )
180 
181         if self.is_legend:
182             self._axis.legend()
183 
184         self._canvas.draw()
185         self._canvas.flush_events()
186 
187         self.number_of_max_points += 1
188 
189     def _count_max(self):
190         print("_count_max")
191         grad = get_grad_func(self.expression)
192 
193         points = find_extremum(grad, self.max_count_init, n_iterations=10000, eta=0.01, b = 0.5, eps=1e-10)
194         return points
195 
196     def count_max(self, x, y):
197         self.max_count_init = [x, y]
198         return self._count_max()
199 
200     def _count_data(self):
201         return count_expression(self._get_params(), self.expression)
202 
203     def _get_params(self):
204         return {
205             0: {
206                 'name': 'x',
207                 'value': self.x,
208             },
209             1: {
210                 'name': 'y',
211                 'value': self.y,
212             }
213         }
```

```
214
215
216 class MyUi_MainWindow(Ui_MainWindow):
217     def setupUi(self, MainWindow, *args, **kwargs):
218         super().setupUi(MainWindow, *args, **kwargs)
219
220         self.graphic = GraphicWidget(MainWindow)
221         self.main_layout.insertWidget(0, self.graphic)
222         self.graphic.show()
223
224         self._init_defaults()
225         self._init_events()
226
227
228     def _count_grid(self):
229         xmin = float(self.x_min_edit.text())
230         xmax = float(self.x_max_edit.text())
231
232         ymin = float(self.y_min_edit.text())
233         ymax = float(self.y_max_edit.text())
234
235         return np.mgrid[xmin:xmax:100j, ymin:ymax:100j]
236
237     def _set_grid(self):
238         self.graphic.set_grid(self._count_grid())
239
240     def _set_levels(self):
241         self.graphic.levels = self.levels_input.value()
242
243     def _set_exp(self):
244         self.graphic.set_exp(self.exp_input.text())
245
246     def _set_draw_type(self):
247         self.graphic.contour_type = self.draw_type_input.
              currentText()
248
249     def _init_events(self):
250         self.draw_btn.clicked.connect(self.graphic.
              update_plot)
251
```

```
252             self.x_min_edit.editingFinished.connect(self.
                    _set_grid)
253             self.x_max_edit.editingFinished.connect(self.
                    _set_grid)
254             self.y_min_edit.editingFinished.connect(self.
                    _set_grid)
255             self.y_max_edit.editingFinished.connect(self.
                    _set_grid)
256 
257             self.levels_input.editingFinished.connect(self.
                    _set_levels)
258 
259             self.exp_input.editingFinished.connect(self.
                    _set_exp)
260 
261             self.draw_type_input.currentTextChanged.connect(
                    self._set_draw_type)
262 
263         def _init_defaults(self):
264             self.x_min_edit.setText(str(0))
265             self.x_max_edit.setText(str(2))
266             self.y_min_edit.setText(str(0))
267             self.y_max_edit.setText(str(2))
268             self._set_grid()
269 
270             self.levels_input.setValue(10)
271             self._set_levels()
272 
273             test_exp = '-1.1*(x**2) -1.5*(y**2) +2*x*y +x+
                    5'
274             self.exp_input.setText(test_exp)
275             self._set_exp()
276 
277             self._set_draw_type()
278 
279 
280 def main():
281     import sys
282     app = QApplication(sys.argv)
283 
284     mainWindow = QMainWindow()
```

```
285
286     ui = MyUi_MainWindow()
287     ui.setupUi(mainWindow)
288
289     mainWindow.show()
290
291     app.exec()
292
293
294 if __name__ == '__main__':
295     main()
```

5.2. count_diff.py

```
1  import sympy
2  from sympy import *
3  import numpy as np
4  from count_expression import count_grad_func
5
6  def count_function_diff(expression):
7      x, y = sympy.symbols('x␣y')
8      exp = eval(expression)
9      x_d = str(sympy.diff(exp, x))
10     y_d = str(sympy.diff(exp, y))
11
12     return (x_d, y_d)
13
14 def get_grad_func(expression):
15     x_d, y_d = count_function_diff(expression)
16     # print(x_d, y_d)
17     return lambda x, y: count_grad_func(x, y, x_d, y_d)
```

5.3. count_expression.py

```
1  import numpy as np
2  from numpy import *
3
4  def count_expression(params: dict, expression: str):
5      from numpy import pi
```

```
6       for ind in params:
7           exec(f"{params[ind]['name']}=params[ind]['value']"
              )
8
9       return eval(expression)
10
11  def count_grad_func(x, y, x_d, y_d):
12      return np.array([-eval(x_d), -eval(y_d)])
```