МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

Національний технічний університет
«Харківський політехнічний інститут»
Кафедра КМАД

ЗВІТ З ЛАБОРАТОРНОЇ РОБОТИ №10

Виконав студент:

Омельніцький Андрій Миколайович
Група: КН-120А

Перевірила:

Ольга Василівна Костюк

Харків, 2023 г.

Зміст

Глава 1

Мета роботи

Визначення екстремуму функції багатьох змінних. Застосування методу найшвидшого спуску.

Порядок виконання:

1. Виконати оптимізацію побудованої моделі за двома змінними, використовуючи в якості першої оптимізаційної змінної обмежену передісторію (змінну кількість тижнів) для побудови шаблонів моделі, в якості другої оптимізаційної змінної обмежену передісторію (змінну кількість тижнів) для адаптивного обчислення коефіцієнта. Графічно представити значення критерію якості моделі, що залежить від двох змінних.
2. Отримати прогнозні значення за оптимізованою моделлю на останній тиждень даних, порівняти з фактичними даними; зробити висновки.
3. Вивчити та реалізувати метод найшвидшого спуску. Навести у звіті алгоритм методу.
4. Знайти за допомогою цього методу екстремум квадратичної функції.
5. Дослідити швидкість збіжності методу найшвидшого спуску для квадратичної функції з різною орієнтацією осей і еліптичністю ліній рівня.
6. Навести графік (для кожної функції), що демонструє роботу методу за допомогою лаб. роботи No0.
7. Навести графік залежності кількості ітерацій від кута повороту квадратичної функцій відносно осі x для різних значень еліптичності ε.
8. Використовуючи програму методу, виконати постановку оптимізаційної задачі (взяти функцію двох змінних $G(\tau, \sigma)$, врахувати її економічний сенс, розглядаючи модель (1-21) в стаціонарному режимі), здійснити розв'язання оптимізаційної задачі. За результатами розв'язання зробити висновки щодо оптимальних значень змінних.
9. Код програми, усі результати, отримані в ході виконання роботи, занести до звіту. Зробити висновки за роботою.

Глава 2

Основна частина

2.1. Пункти 1-2

Виконаємо оптимізацію побудованої моделі за двома змінними: передісторію для побудови шаблонів моделі та передісторію для адаптивного обчислення коефіцієнта. Графічно зобразимо функцію втрат яка залежить від x - передісторію для побудови шаблонів моделі та y - передісторію для адаптивного обчислення коефіцієнта. Отримаємо що при збільшенні передісторію для побудови шаблонів та зменьшенні передісторію реальних данних для адаптивного обчислення коефіцієнта функція втрат здобуває мінімальних значень. Тобто реальні данні для обчислення коефіцієнта треба обмежити лише поточним днем, а передісторію данних для побудови шаблона треба максимізувати.

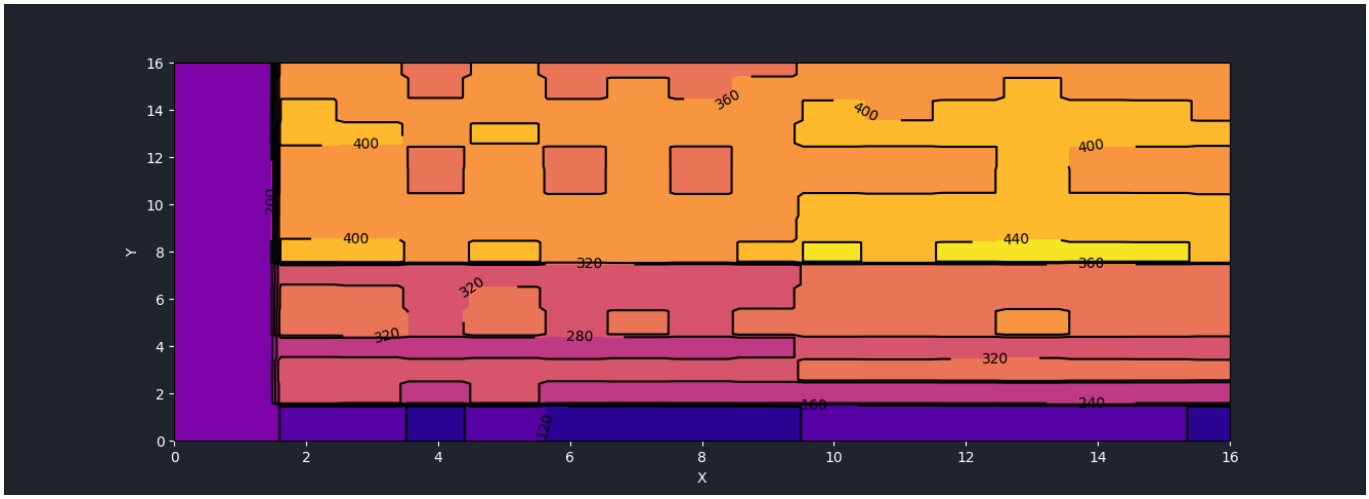

Рис. 2.1. Приклад

Результат роботи на оптимальних данних для понеділка останнього тижня. Де функція втрат здобуває значення $lost = 107.2695001282108$

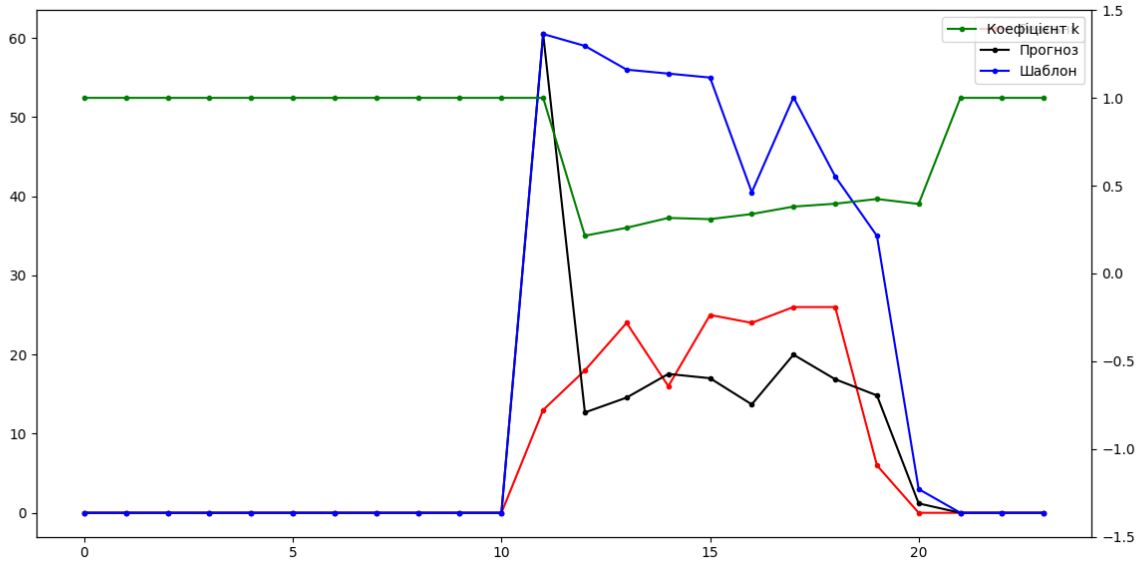

Рис. 2.2. Приклад

2.2. Пункт 3

Критерій зупинки:

$$|x_i - x_{i+1}| < \varepsilon \tag{1}$$

.
Алгоритм:

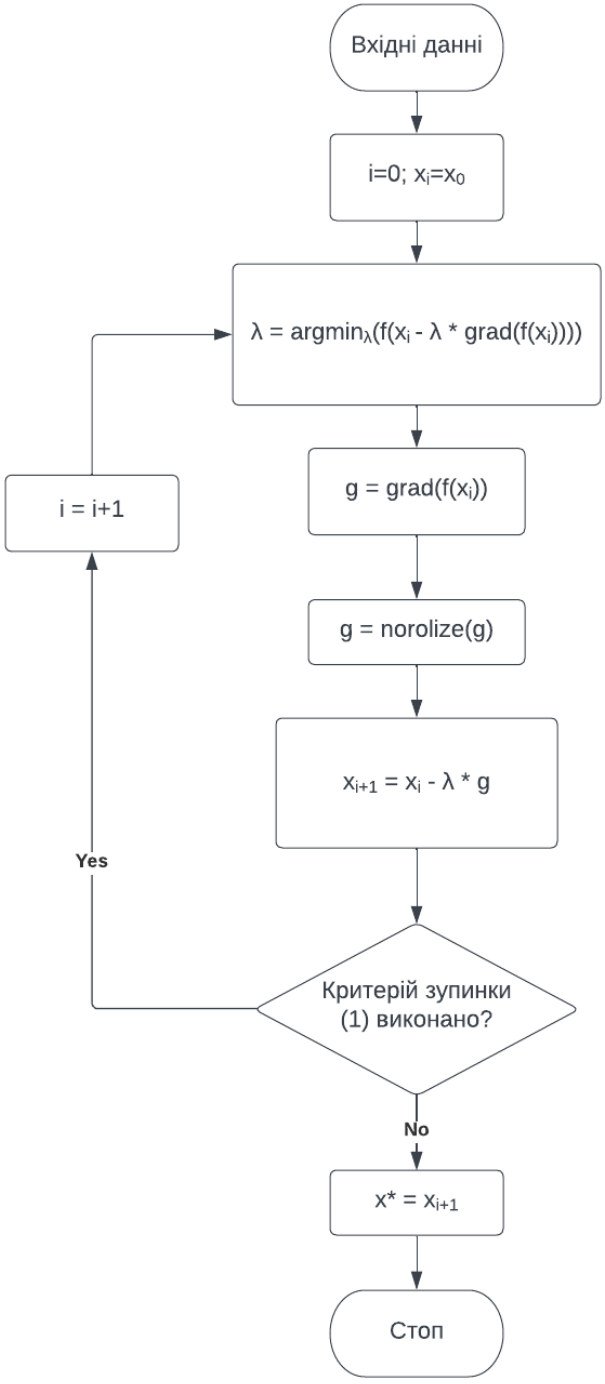

Рис. 2.3. Алгоритм

2.3. Пункт 4

Знайдемо екстремум квадратичної функції $f(x) = (Ax, x) + (b, x)$ за таких параметрів.

$$A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}; b = \begin{pmatrix} 0 \\ 0 \end{pmatrix}; x_0 = \begin{pmatrix} 1 \\ 1 \end{pmatrix};$$

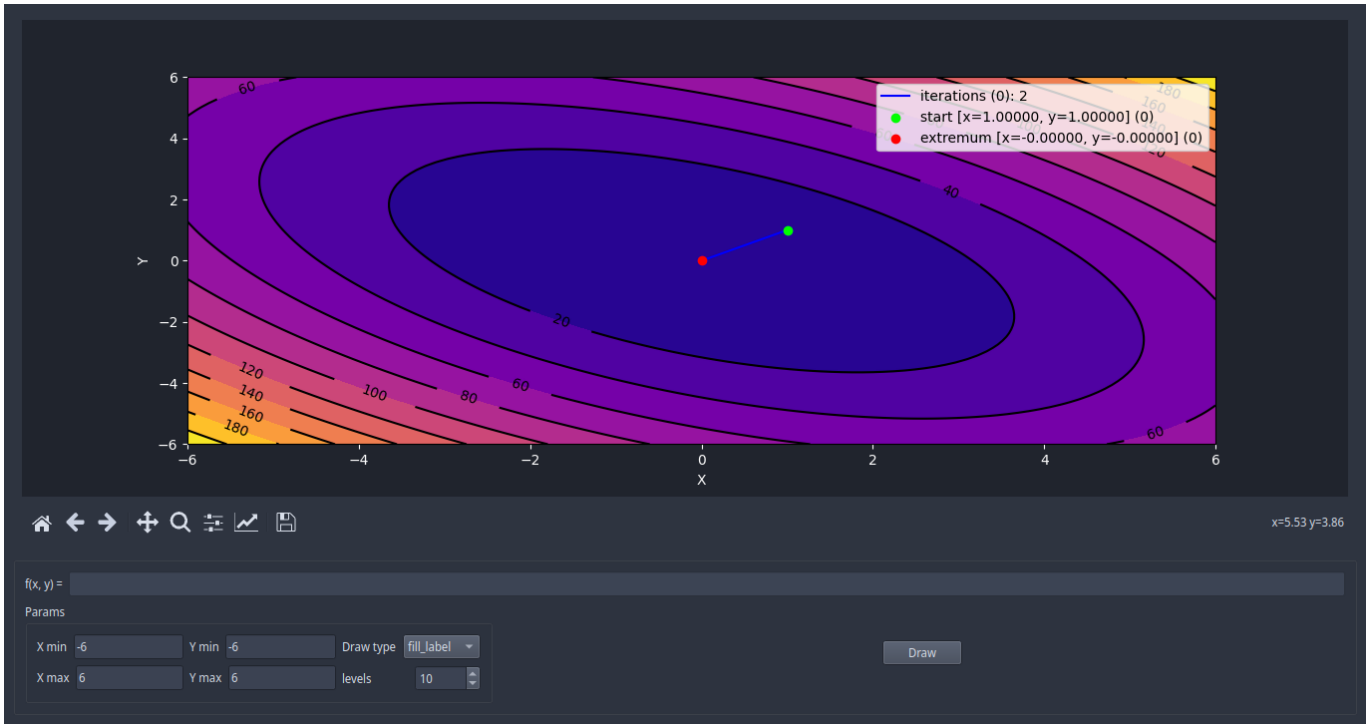

Рис. 2.4. Алгоритм

Отримаємо точку екстремуму $x = 0; y = 0$. Подальше будемо зображати лише графік без інтерфейсу програми.

2.4. Пункти 5-7

Дослідемо залежності кількості ітерацій методу від кута повороту квадратичної функцій відносно осі x_1 для різних значень еліптичності ε. Розглянемо за таких значень еліптичності $\varepsilon = 1, 10, 100$. На рис. 2.5 зображено кути $\alpha \in [0; 2\pi]$ й видно, що мінімальна кількість ітерацій досягається за таких значеннях кута $\alpha = n\pi + \frac{3\pi}{4}$. А максимальна за таких значеннях кута $\alpha = n\pi + \frac{\pi}{4}$.

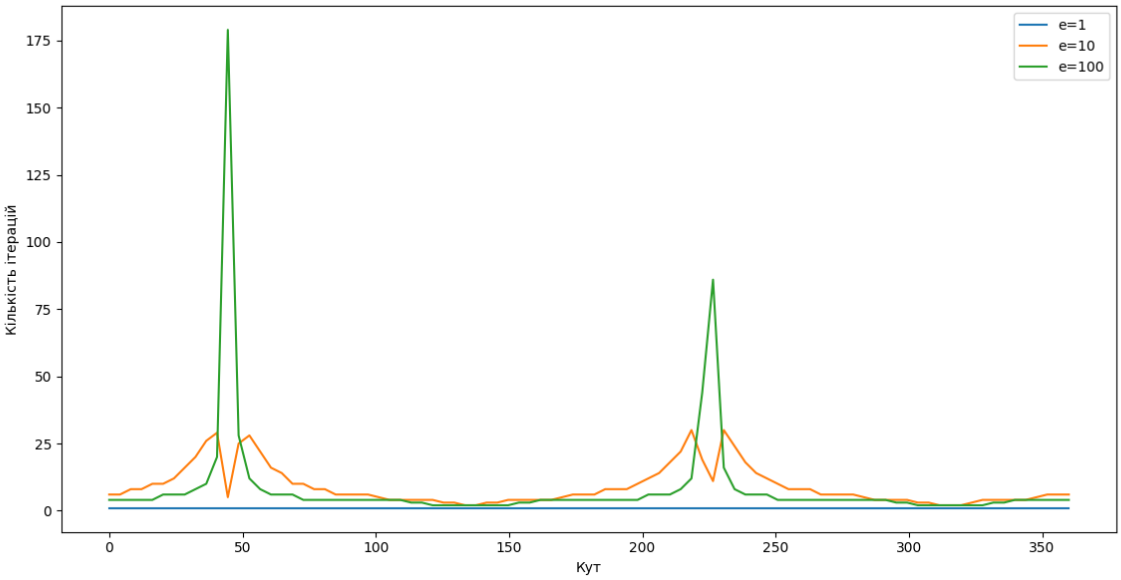

Рис. 2.5. Алгоритм

Розглянемо приклади роботи для усіх параметрів еліптичності за фіксованого куту $\alpha = \pi$ та початкової точки $x_0 = (1, 1)$.

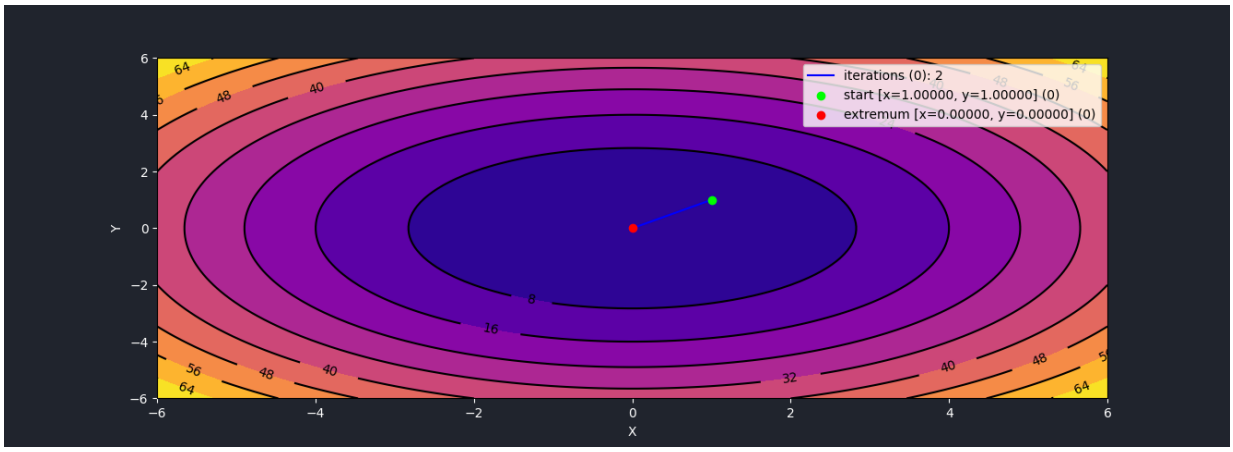

Рис. 2.6. Приклад для $\varepsilon = 1$

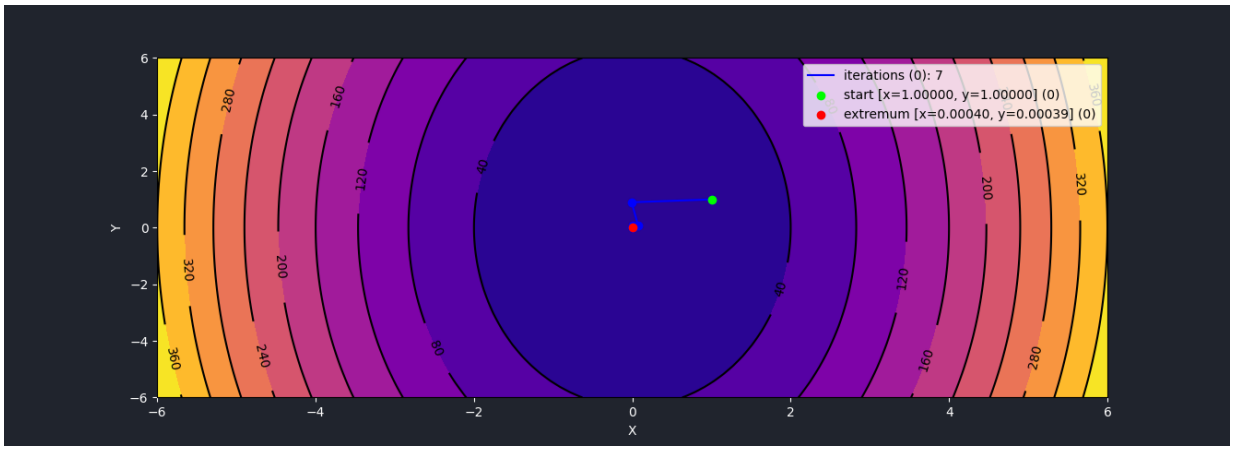

Рис. 2.7. Приклад для $\varepsilon = 10$

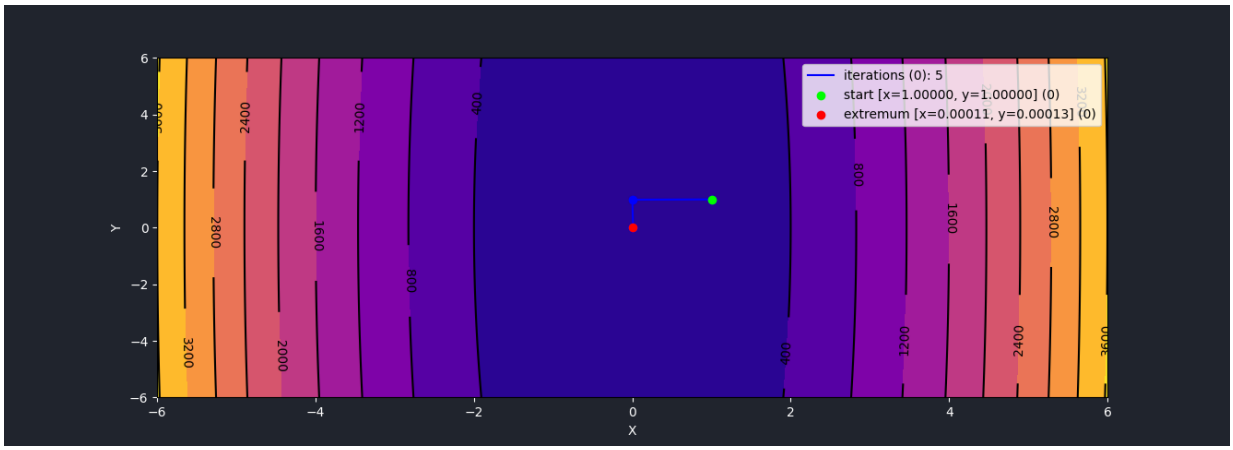

Рис. 2.8. Приклад для $\varepsilon = 100$

2.5. Пункт 8

Оптимізуемо функцію прибутку держави $G(\tau, \sigma)$. Де τ - норма оподаткування, а σ - норма відрахування на пригнічення тіньового сектора. Візуалізуємо процес максимізації та функцію $G(\tau, \sigma)$. В результаті оптимізації отримаємо $\tau = 0.6454$, а $\sigma = 0$. Це сівдчить про те, що виділення на пригнічення тіньового сектора не мають сенсу, для збільшення прибутку держави, а оптимальною нормою оподаткування є $\tau = 0.6454$.

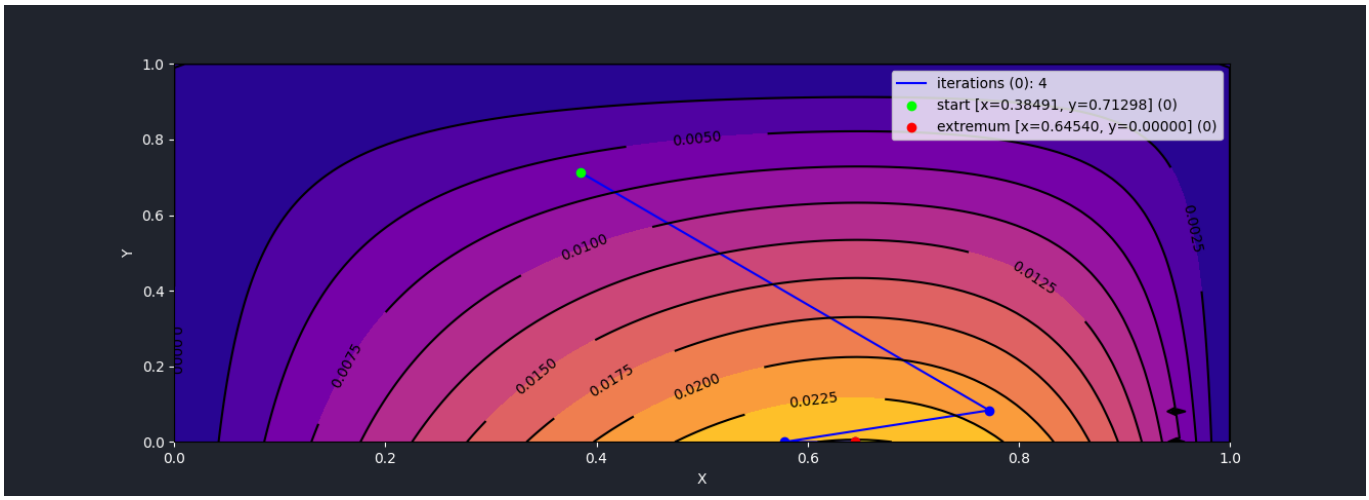

Рис. 2.9. Приклад для $\varepsilon = 100$

Глава 3

Висновок

У ході лабораторної роботи було вивченно метод найшвидшого спуску. А також застосовано його для знаходження екстремуму різніх функцій багатьох змінних.

Глава 4

Код програми

4.1. 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
from matplotlib.backend_bases import MouseButton
import matplotlib.pyplot as plt
import sympy
from numpy import pi
from collections.abc import Callable
from scipy.integrate import odeint, solve_ivp
from scipy.optimize import fsolve


# 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

from services.model import EconomicModel
from method import test_func, gradient_descent,
  get_rotated_matrix
from model import get_target_func


```

```
29
30 def plot_levels(fig, ax, params, data, levels=5, *,
     contour_type='line_label'):
31     '''
32     contour_type = 'line' | 'fill' | 'line_label' | '
         fill_label'
33     '''
34
35     x, y = params[0]['value'], params[1]['value']
36
37     # fig, ax = plt.subplots()
38
39     match contour_type:
40         case 'line':
41             ax.contour(x, y, data, levels=levels, cmap='
                 plasma')
42         case 'line_label':
43             cs = ax.contour(x, y, data, levels=levels,
                 cmap='plasma')
44             ax.clabel(cs)
45         case 'fill':
46             ax.contourf(x, y, data, levels=levels, cmap='
                 plasma')
47         case 'fill_label':
48             color_line = np.zeros((levels, 3))
49             c = np.linspace(1, 0.2, levels)
50             # color_line[:, 0] = c
51             # color_line[:, 1] = c
52             # color_line[:, 2] = c
53             color_line[:, 0] = 0
54             color_line[:, 1] = 0
55             color_line[:, 2] = 0
56             ax.contourf(x, y, data, levels=levels, cmap='
                 plasma')
57             cs = ax.contour(x, y, data, levels=levels,
                 colors=color_line)
58             ax.clabel(cs)
59
60     fig.set_figwidth(5)
61     fig.set_figheight(5)
62
```

```
63
64  def plot_path(ax, points, label, color='r'):
65      x, y = points[0], points[1]
66      ax.plot(x, y, color=color, label=label)
67
68
69
70
71  def f_by_tau_sigma(
72      model: EconomicModel,
73      target_func: Callable,
74      tau: float, sigma: float,
75      init: list[float] | None = None) -> float:
76      prepared_model_call = lambda x, *args: model(x,
          changed_params={"tau": args[0], "sigma": args[1]})
77      # print(tau, sigma)
78      #
79      if init is None:
80          init = [0, 0.5, 0.25, 0.1, 0, 0.5, 0.25, 0.1]
81
82      new_x = fsolve(prepared_model_call, x0=init, args=(tau
          , sigma))
83
84      #
                                                fsolve.    .   .
                                                                .
85      # t = np.arange(t_start, t_end, 0.1)
86      # new_x = odeint(lambda t, x, *args:
          prepared_model_call(x, *args), init, t, tfirst=True,
           args=(tau, sigma))[-1]
87
88      model.tau = tau
89      model.sigma = sigma
90      result = target_func(new_x)
91
92      #
93      model.set_default()
94      return result
95
```

```
96
97
98
99  # A = [[2, 1], [1, 2]]
100 # b = [0, 0]
101 # target_func = lambda p: test_func(*p, A, b)
102
103
104 # model = EconomicModel()
105 # target_func = lambda p: f_by_tau_sigma(model, model.G, *
        p)
106
107
108 # es = [1, 10, 100]
109 # e = es[2]
110 # angle = math.pi/2
111 # A = [[1, 0], [0, e]]
112 # b = [0, 0]
113 # func = get_rotated_matrix(angle)(test_func)
114 # target_func = lambda p: func(*p, A, b)
115
116 # 0 16
117 func = get_target_func()
118 target_func = lambda p: func(*p)
119
120
121 class GraphicWidget(QWidget):
122     def __init__(self, parent=None):
123         QWidget.__init__(self, parent)
124
125         self.plot_layout = QVBoxLayout(self)
126
127         self._figure = Figure(facecolor='#20242d')
128         self._canvas = FigureCanvas(self._figure)
129         self._axis = self._figure.add_subplot(111)
130         self._init_axis()
131         self._canvas.draw()
132         self.toolbar = NavigationToolbar(self._canvas,
              self)
133
134         self.plot_layout.addWidget(self._canvas)
```

```
135         self.plot_layout.addWidget(self.toolbar)
136
137         self.levels = 10
138         self.contour_type = 'line'
139         self.max_count_init = [0, 0]
140
141         self.is_legend = True
142         self.number_of_max_points = 0
143
144         self._canvas.mpl_connect('button_press_event',
              self.mpl_on_click)
145
146         self.func = target_func
147         # self.is_max = False
148         self.is_max = True
149
150     def mpl_on_click(self, event):
151         if event.dblclick:
152             x, y = event.xdata, event.ydata
153             if event.button is MouseButton.RIGHT:
154                 x, y = 1, 1
155             points = self.count_max(x, y)
156             self.plot_path(points)
157
158     def _init_axis(self):
159         color = '#ffffff'
160
161         self._axis.tick_params(axis='x', colors=color)
162         self._axis.tick_params(axis='y', colors=color)
163
164         self._axis.xaxis.label.set_color(color)
165         self._axis.yaxis.label.set_color(color)
166
167     def _clear_plot(self):
168         self.number_of_max_points = 0
169         self._axis.clear()
170
171     def update_plot(self):
172         self._clear_plot()
173         self._axis.set_xlabel('X')
174         self._axis.set_ylabel('Y')
```

```
175
176         params = self._get_params()
177         data = self._count_data()
178         plot_levels(self._figure, self._axis, params, data
                , self.levels, contour_type=self.contour_type)
179
180         self._canvas.draw()
181         self._canvas.flush_events()
182
183     def set_grid(self, grid):
184         self.x, self.y = grid
185
186     def set_exp(self, expression):
187         ...
188         # self.expression = expression
189
190     def plot_path(self, points):
191         self._axis.plot(
192             points[0], points[1],
193             color="blue",
194             zorder=1,
195             label=f'iterations ({self.number_of_max_points
                  }): {points.shape[1]}'
196         )
197         self._axis.scatter(
198             points[0], points[1],
199             zorder=2,
200             color="blue",
201         )
202         self._axis.scatter(
203             points[0][0], points[1][0],
204             color="#00ff00",
205             zorder=2,
206             label=f"start [x={points[0][0]:.5f}, y={points
                  [1][0]:.5f}] ({self.number_of_max_points})"
207         )
208         self._axis.scatter(
209             points[0][-1], points[1][-1],
210             color="#ff0000",
211             zorder=2,
```

```
212             label=f"extremum [x={points[0][-1]:.5f}, y={
                    points[1][-1]:.5f}] ({self.
                    number_of_max_points})"
213         )
214 
215         if self.is_legend:
216             self._axis.legend()
217 
218         self._canvas.draw()
219         self._canvas.flush_events()
220 
221         self.number_of_max_points += 1
222 
223     def _count_max(self):
224         eps = 0.001
225         bounds = [[self.x[0][0], self.x[-1][0]], [self.y
                    [0][0], self.y[0][-1]]]
226 
227         grad_iters = []
228         func = self.func
229         if self.is_max == True:
230             func = lambda *args, **kwargs: -self.func(*
                    args, **kwargs)
231 
232         x, y = gradient_descent(func, self.max_count_init,
                    eps, bounds=bounds, iterations=grad_iters)
233         print(f'Extremum: x={x} y={y} f(x)={self.func([x, 
                    y])}')
234 
235         grad_iters = np.array(grad_iters)
236         return np.array([grad_iters[:, 0], grad_iters[:,
                    1]])
237 
238     def count_max(self, x, y):
239         self.max_count_init = [x, y]
240         return self._count_max()
241 
242     def _count_data(self):
243         data = []
244         for x in self.x[:, 0]:
245             data.append([])
```

```
246             for y in self.y[0]:
247                 data[-1].append(self.func([x, y]))
248 
249         return np.array(data)
250 
251     def _get_params(self):
252         return {
253             0: {
254                 'name': 'x',
255                 'value': self.x,
256             },
257             1: {
258                 'name': 'y',
259                 'value': self.y,
260             }
261         }
262 
263 
264 class MyUi_MainWindow(Ui_MainWindow):
265     def setupUi(self, MainWindow, *args, **kwargs):
266         super().setupUi(MainWindow, *args, **kwargs)
267 
268         self.graphic = GraphicWidget(MainWindow)
269         self.main_layout.insertWidget(0, self.graphic)
270         self.graphic.show()
271 
272         self._init_defaults()
273         self._init_events()
274 
275 
276     def _count_grid(self):
277         xmin = float(self.x_min_edit.text())
278         xmax = float(self.x_max_edit.text())
279 
280         ymin = float(self.y_min_edit.text())
281         ymax = float(self.y_max_edit.text())
282 
283         return np.mgrid[xmin:xmax:100j, ymin:ymax:100j]
284 
285     def _set_grid(self):
286         self.graphic.set_grid(self._count_grid())
```

```
287
288     def _set_levels(self):
289         self.graphic.levels = self.levels_input.value()
290
291     def _set_exp(self):
292         self.graphic.set_exp(self.exp_input.text())
293
294     def _set_draw_type(self):
295         self.graphic.contour_type = self.draw_type_input.
              currentText()
296
297     def _init_events(self):
298         self.draw_btn.clicked.connect(self.graphic.
              update_plot)
299
300         self.x_min_edit.editingFinished.connect(self.
              _set_grid)
301         self.x_max_edit.editingFinished.connect(self.
              _set_grid)
302         self.y_min_edit.editingFinished.connect(self.
              _set_grid)
303         self.y_max_edit.editingFinished.connect(self.
              _set_grid)
304
305         self.levels_input.editingFinished.connect(self.
              _set_levels)
306
307         self.exp_input.editingFinished.connect(self.
              _set_exp)
308
309         self.draw_type_input.currentTextChanged.connect(
              self._set_draw_type)
310
311     def _init_defaults(self):
312         self.x_min_edit.setText(str(-6))
313         self.x_max_edit.setText(str(6))
314         self.y_min_edit.setText(str(-6))
315         self.y_max_edit.setText(str(6))
316         self._set_grid()
317
318         self.levels_input.setValue(10)
```

```
319             self._set_levels()
320
321             # test_exp = '-1.1*(x**2) - 1.5*(y**2) + 2*x*y + x
                    + 5'
322             # self.exp_input.setText(test_exp)
323             self.exp_input.setEnabled(False)
324             # self._set_exp()
325
326             self._set_draw_type()
327
328
329 def main():
330     import sys
331     app = QApplication(sys.argv)
332
333     mainWindow = QMainWindow()
334
335     ui = MyUi_MainWindow()
336     ui.setupUi(mainWindow)
337
338     mainWindow.show()
339
340     app.exec()
341
342
343 if __name__ == '__main__':
344     main()
```

4.2. model.py

```
1  import numpy as np
2  import pandas as pd
3  import matplotlib.pyplot as plt
4  from statsmodels.graphics import tsaplots
5  from datetime import datetime, timedelta
6  import math
7  import functools
8
9
10 class PredictionModel:
```

```
11      def __init__(self):
12          self._template = np.zeros((7, 24))
13          self._shop_id = -1
14          self._data = None
15
16          self.week_day_by_int = {
17              0: "                    ",
18              1: "                  ",
19              2: "              ",
20              3: "              ",
21              4: "   ’             ",
22              5: "              ",
23              6: "              ",
24          }
25
26      def count_by_day_and_hour(self):
27          ...
28
29      def _count_average_template(self):
30          average = lambda sales: sales.mean()
31          self._count_template_by_sales(average)
32
33      def _count_median_template(self):
34          average = lambda sales: sales.median()
35          self._count_template_by_sales(average)
36
37      def _count_template_by_sales(self, func):
38          for i, day in enumerate(range(0, 7)):
39              all_week_day_data = self._data[self._data['
                  week_day'] == day]
40              hours = all_week_day_data['time'].unique()
41
42              for j, hour in enumerate(hours):
43                  all_data_in_hour = all_week_day_data[
                      all_week_day_data['time'] == hour]
44                  sales = all_data_in_hour['All']
45
46                  self._template[i][j] = func(sales)
47
48      def _preparing_data_frame(self):
```

```
49          self._data["All"] = self._data["All"].replace(np.
              nan, 0)
50          weeks_day = []
51          weeks_id = []
52          week_id = 0
53          last_week_day = 0
54          for date in self._data['date']:
55              date = datetime.strptime(date, "%Y-%m-%d").
                  date()
56              weeks_day.append(date.weekday())
57              if date.weekday() == 0 and last_week_day == 6:
58                  week_id += 1
59              weeks_id.append(week_id)
60              last_week_day = date.weekday()
61
62          self._data['week_day'] = weeks_day
63          self._data['week_id'] = weeks_id
64
65      def _training_by_type(self, training_type):
66          if training_type == 'median':
67              self._count_median_template()
68          elif training_type == 'average':
69              self._count_average_template()
70
71      def training_model(self, data_frame, training_type='
          median'):
72          self._data = data_frame
73          self._preparing_data_frame()
74          self._training_by_type(training_type)
75
76      def _get_week_by_date(self, date):
77          week = [date]
78
79          date_now = date
80          while True:
81              # print(date_now, date_now.weekday())
82              date_now = date_now + timedelta(days=1)
83              if date_now.weekday() == 0:
84                  break
85
86              week.append(date_now)
```

```
87
88          date_now = date
89          while True:
90              # print(date_now, date_now.weekday())
91              date_now = date_now - timedelta(days=1)
92              if date_now.weekday() == 6:
93                  break
94
95              week.insert(0, date_now)
96
97          # print(date)
98          # print(week)
99          return week
100
101     def plot_statistic(self, ploted_days='__all__'):
102         if ploted_days == '__all__':
103             ploted_days = range(0, 7)
104
105         w = math.ceil(math.sqrt(len(ploted_days)))
106         h = math.ceil(len(ploted_days) / w)
107
108         plt.subplots_adjust(wspace=0.3, hspace=0.6)
109         axes = []
110
111         for i, day in enumerate(ploted_days):
112             ax = plt.subplot(h, w, i+1)
113             axes.append(ax)
114             ax.set_title(self.week_day_by_int[day])
115             ax.set_xlabel('             ')
116             ax.set_ylabel('               ')
117
118         for i, day in enumerate(ploted_days):
119             all_week_day_data = self._data[self._data['
                week_day'] == day]
120             weeks_dates = all_week_day_data['date'].unique
                ()
121             for week_date in weeks_dates:
122                 week_data = all_week_day_data[self._data['
                    date'] == week_date]
123
124                 sales = week_data['All']
```

```
125                 hours = week_data['time']
126                 axes[i].plot(hours, sales, color='black')
127 
128         reserve_template = self._template.copy()
129 
130         self._training_by_type('median')
131         for i, day in enumerate(ploted_days):
132             sales = self._template[day]
133             hours = range(0, 24)
134 
135             axes[i].plot(hours, sales, color='g', label='
                median')
136 
137         self._training_by_type('average')
138         for i, day in enumerate(ploted_days):
139             sales = self._template[day]
140             hours = range(0, 24)
141 
142             axes[i].plot(hours, sales, color='r', label='
                average')
143 
144         self._template = reserve_template
145 
146         plt.legend()
147         plt.show()
148 
149     def _count_k(self, week_day, data):
150         if len(data) == 0:
151             return 1
152         if data.iloc[-1]['All'] == 0:
153             return 1
154 
155         n = 0
156         result = 0
157         for hour_data_id in range(len(data)):
158             hour_data = data.iloc[hour_data_id]
159             if self._template[week_day][hour_data['time']]
                    == 0:
160                 continue
161 
162             n += 1
```

```
163             result += hour_data['All'] / self._template[
                    week_day][hour_data['time']]
164
165         if n == 0:
166             return 1
167         return result / n
168
169     def predict(self, week_day, hour, history, sub_history
            =None, is_adapt_template=False, template_id=None):
170         if hour == 0:
171             if template_id is not None:
172                 template_id.append(week_day)
173             return 0
174         if sub_history is None:
175             k = self._count_k(week_day, history)
176         else:
177             k = self._count_k(week_day, pd.concat([
                    sub_history, history]))
178
179         if is_adapt_template:
180
181             sales = []
182             for j in range(len(history)):
183                 t_h = history.iloc[:j]
184                 sales.append(self.predict(week_day, j, t_h
                        , sub_history))
185
186             min_val = self.count_lost(list(history['All'])
                    , sales)
187             t_id = week_day
188             for i in range(self._template.shape[0]):
189                 sales = []
190                 for j in range(len(history)):
191                     t_h = history.iloc[:j]
192                     sales.append(self.predict(i, j, t_h,
                            sub_history))
193
194                 res = self.count_lost(list(history['All'])
                        , sales)
195                 # if hour < 12:
196                 #     print(f'{history["All"]=}')
```

```
197                     #     print(f'{sales=}')
198                     #     print(f'{res=}')
199                     #     print(f'{min_val=}')
200                     #     print(f'{t_id=}')
201
202                     # print('='*20)
203                     # print(f'{i=}')
204                     # print(f'{res=}')
205                     # print(f'{min_val=}')
206                     # print(f'{t_id=}')
207                     # print('='*20)
208                     if min_val > res:
209                         min_val = res
210                         t_id = i
211
212                 return self.predict(t_id, hour, history,
                      sub_history, template_id=template_id)
213             else:
214                 if template_id is not None:
215                     template_id.append(week_day)
216                 template = self._template[week_day][hour]
217                 return k * template
218
219     def count_lost(self, real, predicted):
220         assert len(real) == len(predicted)
221
222         lost = 0
223         for i in range(len(real)):
224             lost += abs(real[i] - predicted[i])
225
226         return lost
227
228
229
230 def test_predict_for_template(df):
231     df = df.copy()
232     model = PredictionModel()
233     # model.training_model(df, 'average')
234     model.training_model(df)
235
236
```

```
237     last_week_id = (df['week_id'].unique()[-2])
238     losts = []
239     for id_temp in range(last_week_id):
240         t_data = df[df['week_id'] <= id_temp]
241         r_data = df[df['week_id'] == last_week_id]
242         # print(t_data)
243         # print(r_data)
244 
245 
246         model.training_model(t_data)
247 
248 
249         day = 2
250         # print(r_data)
251         target_date = r_data[r_data['week_day'] == day].
              iloc[-1]['date']
252         target_data = r_data[r_data['date'] == target_date
              ]
253 
254         # print(f'{target_date=}')
255         # print(target_data)
256 
257         y = []
258         k = []
259         sales = target_data
260         hours = target_data['time']
261         for t in range(len(sales)):
262             y.append(model.predict(day, t, sales.iloc[:t])
                  )
263             k.append(model._count_k(day, sales.iloc[:t]))
264 
265         lost = model.count_lost(list(sales['All']), list(y
              ))
266         losts.append(lost)
267 
268     plt.xlabel('                    ␣          ␣      ␣
                        ')
269     plt.ylabel('             ')
270     plt.plot(range(1, last_week_id+1), losts)
271     plt.show()
272 
```

```
273
274 def test_predict_for_k(df):
275     df = df.copy()
276     model = PredictionModel()
277     # model.training_model(df, 'average')
278     model.training_model(df)
279
280
281     last_week_id = (df['week_id'].unique()[-1])
282     t_data = df[df['week_id'] != last_week_id]
283     r_data = df[df['week_id'] == last_week_id]
284     # print(t_data)
285     # print(r_data)
286
287
288     model.training_model(t_data)
289
290
291     day = 0
292     # print(r_data)
293     target_date = r_data[r_data['week_day'] == day].iloc
          [-1]['date']
294     target_data = r_data[r_data['date'] == target_date]
295
296
297     losts = []
298     for id_temp in range(last_week_id+1):
299         y = []
300         k = []
301         # history_added = df[(df['week_id'] <= id_temp) &
              (df['week_day'] == day)]
302         history_added = df[(df['week_id'] < id_temp) & (df
              ['week_day'] == day)]
303         print(history_added)
304         sales_traget = target_data['All']
305         hours = target_data['time']
306         for t, value in enumerate(sales_traget):
307             sales = target_data.iloc[:t]
308             # print(f'{history_added=}')
309             # print(f'{sales=}')
310             # print(f'{target_data.iloc[:t+1]=}')
```

```
311             y.append(model.predict(day, t, sales,
                    history_added))
312             k.append(model._count_k(day, sales))
313
314         lost = model.count_lost(list(sales_traget), list(y
                ))
315         losts.append(lost)
316
317     plt.xlabel('                    ␣          ␣       ␣
                    ')
318     plt.ylabel('            ')
319     plt.plot(range(last_week_id+1), losts)
320     plt.show()
321
322
323
324 def get_target_func():
325     df = pd.read_csv('data.csv', sep=';')
326     df = df.set_index('index')
327     # df_res = df.copy()
328
329     # df = df.copy()
330     model = PredictionModel()
331     model.training_model(df)
332
333
334     # last_week_id = (df['week_id'].unique()[-1])
335     # t_data = df[df['week_id'] != last_week_id]
336     # r_data = df[df['week_id'] == last_week_id]
337
338
339     # model.training_model(t_data)
340
341
342     day = 0
343     last_week_id = (df['week_id'].unique()[-1])
344     # print(r_data)
345     # target_date = r_data[r_data['week_day'] == day].iloc
            [-1]['date']
346     # target_data = r_data[r_data['date'] == target_date]
347
```

```
348
349
350     # losts = []
351     # for id_temp in range(last_week_id):
352     #     t_data = df[df['week_id'] <= id_temp]
353     #     r_data = df[df['week_id'] == last_week_id]
354     #     # print(t_data)
355     #     # print(r_data)
356
357
358     #     model.training_model(t_data)
359
360
361     #     day = 0
362     #     # print(r_data)
363     #     target_date = r_data[r_data['week_day'] == day].
        iloc[-1]['date']
364     #     target_data = r_data[r_data['date'] ==
        target_date]
365
366     #     # print(f'{target_date=}')
367     #     # print(target_data)
368
369     #     y = []
370     #     k = []
371     #     sales = target_data
372     #     hours = target_data['time']
373     #     for t in range(len(sales)):
374     #         y.append(model.predict(day, t, sales.iloc[:t
        ]))
375     #         k.append(model._count_k(day, sales.iloc[:t])
        )
376
377     #     lost = model.count_lost(list(sales['All']), list
        (y))
378     #     losts.append(lost)
379
380
381     # losts = []
382     # for week_id in range(last_week_id+1):
383     #     y = []
```

```
384         #     k = []
385         #     # history_added = df[(df['week_id'] <= id_temp)
                & (df['week_day'] == day)]
386         #     history_added = df[(df['week_id'] < week_id) & (
                df['week_day'] == day)]
387         #     print(history_added)
388         #     sales_traget = target_data['All']
389         #     hours = target_data['time']
390         #     for t, value in enumerate(sales_traget):
391         #         sales = target_data.iloc[:t]
392         #         y.append(model.predict(day, t, sales,
                history_added))
393
394         #     lost = model.count_lost(list(sales_traget), list
                (y))
395         #     losts.append(lost)
396
397         # lost = model.count_lost(list(sales_traget), list(y))
398
399         @functools.cache
400         def func(id_temp, week_id):
401             t_data = df[df['week_id'] < id_temp]
402             r_data = df[df['week_id'] == last_week_id]
403
404             model = PredictionModel()
405             model.training_model(t_data)
406
407             target_date = r_data[r_data['week_day'] == day].
                  iloc[-1]['date']
408             target_data = r_data[r_data['date'] == target_date
                  ]
409
410             y = []
411             history_added = df[(df['week_id'] < week_id) & (df
                  ['week_day'] == day)]
412             sales_traget = target_data['All']
413             print(model._template)
414             print(t_data)
415             print(history_added)
416             for t, value in enumerate(sales_traget):
417                 sales = target_data.iloc[:t]
```

```
418                 y.append(model.predict(day, t, sales,
                        history_added))
419
420             lost = model.count_lost(list(sales_traget), list(y
                    ))
421             return lost
422
423         def target_func(template_n, k_n):
424             print(template_n, k_n)
425             template_n = round(template_n)
426             k_n = round(k_n)
427             return func(template_n, k_n)
428
429         return target_func
430
431
432 func = get_target_func()
433 lost = []
434 print(func(7, 0))
435 # for i in range(0, 17):
436 #     lost.append(func(i, 0))
437
438 # print(min(lost), lost.index(min(lost)))
439 # print(lost[lost.index(min(lost))])
440 # 7
```

4.3. method.py

```
1  import numpy as np
2  import math
3  import matplotlib.pyplot as plt
4  from scipy.optimize import fsolve
5  from PyQt5 import QtCore, QtWidgets# , QtWebEngineWidgets
6  from PyQt5.QtGui import QFont
7  import numpy as np
8  import plotly.graph_objects as go
9  from numpy import linalg as LA
10 from typing import List, Tuple
11 import math
12
```

```
13 from services.opt import *
14
15
16 def part_diff(function, point, diff_id, eps=1e-9):
17     d_point = point.copy()
18     d_point[diff_id] = point[diff_id] + eps
19     return (function(d_point) - function(point)) / eps
20
21
22 def grad_in_point(function, point):
23     grad = []
24     for i in range(len(point)):
25         grad.append(part_diff(function, point, i))
26
27     return np.array(grad)
28
29
30 def gradient_descent(func, init, eps, bounds = None,
     iterations = None, min_cls=DichotomyMethod):
31     minimization_method = min_cls()
32     minimization_method.is_plot = False
33     minimization_method.is_log = False
34
35     point = np.array(init, dtype=float)
36
37     while True:
38         if iterations is not None:
39             iterations.append(point.copy())
40
41         n_grad = grad_in_point(func, point)
42         # grad = n_grad
43         grad = n_grad / np.linalg.norm(n_grad, ord=1)
44
45         if bounds is not None:
46             for i, (start, end) in enumerate(bounds):
47                 if abs(point[i] - start) < eps:
48                     grad[i] = min(grad[i], 0)
49                     n_grad[i] = min(n_grad[i], 0)
50                 elif abs(point[i] - end) < eps:
51                     grad[i] = max(grad[i], 0)
52                     n_grad[i] = max(n_grad[i], 0)
```

```
53
54          f = lambda alpha: func(point - (alpha * grad))
55
56          sub_eps = 1e-6
57          search_bounds = minimization_method.get_interval(f
              , 0, min(sub_eps / np.max(np.abs(n_grad)),
              sub_eps))
58          alpha, *_ = minimization_method(f, search_bounds,
              sub_eps)
59          step = grad * (-alpha)
60
61          if np.sum(step**2)**0.5 < eps:
62              return point
63
64          point += step
65
66          if bounds is not None:
67              for i, (start, end) in enumerate(bounds):
68                  if point[i] < start:
69                      point[i] = start
70                  elif point[i] > end:
71                      point[i] = end
72
73
74  def draw_path(func, debug_list, bounds, number_of_lines,
      xtitle='', ytitle=''):
75      l, r = bounds
76      x = np.linspace(l, r, 30)
77      y = np.linspace(l, r, 20)
78      X, Y = np.meshgrid(x, y)
79      Z = []
80      print(len(X[0]), len(Y[:, 0]))
81      print(X[0], Y[:, 0])
82      for y_t in Y[:, 0]:
83          Z.append([])
84          for x_t in X[0]:
85              Z[-1].append(func([x_t, y_t]))
86      Z = np.array(Z)
87      minv = np.min(Z)
88      maxv = np.max(Z)
89      fig = go.Figure(data=
```

```
90          go.Contour(
91              z=Z,
92              x=x,
93              y=y,
94              contours=dict(
95                  showlabels=True
96              ),
97              contours_start=minv,
98              contours_end=maxv,
99              contours_size=(maxv - minv) / number_of_lines,
100         )
101     )
102     x_values = [debug_list[i][0] for i in range(len(
            debug_list))]
103     y_values = [debug_list[i][1] for i in range(len(
            debug_list))]
104     fig.add_trace(go.Scatter(x=x_values, y=y_values,
            showlegend=False))
105     fig.add_trace(go.Scatter(x=[x_values[0]], y=[y_values
            [0]],
106                              name='start', marker=dict(
                                    color="Green", size=6),
                                    showlegend=True))
107     fig.add_trace(go.Scatter(x=[x_values[-1]], y=[y_values
            [-1]],
108                              name=f'extremum␣({len(
                                    debug_list)-1}␣iterations)'
                                    ,
109     marker=dict(color="Blue", size=6), showlegend=True))
110     fig.update_layout(
111         legend=dict(
112             yanchor="top",
113             xanchor="left",
114             x=0.01,
115             y=0.99,
116         ),
117         xaxis_title=xtitle,
118         yaxis_title=ytitle,
119     )
120     fig.show()
121
```

```
122
123 def rotate(coords, angle):
124     rot = [[math.cos(angle), -math.sin(angle)],[math.sin(
        angle), math.cos(angle)]]
125     return np.matmul(coords, rot)
126
127
128 def get_rotated_matrix(angle):
129     def rot_decorator(func):
130         def wrap(x, y, *args, **kwargs):
131             x, y = rotate(np.array([x, y]), angle)
132             return func(x, y, *args, **kwargs)
133         return wrap
134     return rot_decorator
135
136
137 def test_func(x, y, A, b):
138     return A[0][0] * x**2 + (A[0][1] + A[1][0]) * x * y +
        A[1][1] * y**2 + b[0] * x + b[1] * y
139
140
141
142
143
144
145
146
147 # aph = 0.5
148 # bt = 1.5
149 # gm = 1.5
150 # dt = 0.1
151 # nu = 5.0
152 # mu = 20.0
153 # lmd = 20.0
154 # ro = 10.0
155 # A0 = 1.0
156 # L0 = 1.0
157 # D0 = 1.0
158 # theta = (1 + aph * (bt - 1)) ** (-1)
159 # def L1(data):
```

```
160 # return data[3] * ((1 - aph) * A0 * data[1] / data[2]) **
        (1 / aph)
161 # def Q1(data):
162 # return A0 * data[3] ** aph * L1(data) ** (1 - aph)
163 # def D1(data):
164 # return D0 * math.exp(-bt * data[1]) * data[5] / (data[1]
        + data[5])
165 # def S1(data):
166 # return L0 * (1 - math.exp(-gm * data[2])) * data[2] / (
       data[2] + data[6]);
167 # def I1(data, tau):
168 # # print(tau)
169 # return (1 - tau) * (1 - theta) * data[0]
170 # def G1(data, tau):
171 # return (1 - tau) * theta * data[0]
172 # def L2(data):
173 # return data[7] * ((1 - aph) * A0 * data[5] / data[6]) **
        (1 / aph)
174 # def Q2(data):
175 # return A0 * data[7] ** aph * L2(data) ** (1 - aph)
176 # def D2(data):
177 # return D0 * math.exp(-bt * data[5]) * data[1] / (data[1]
        + data[5])
178 # def S2(data):
179 # return L0 * (1 - math.exp(-gm * data[6])) * data[6] / (
       data[2] + data[6])
180 # def I2(data):
181 # return (1 - theta) * data[4]
182 # def G2(data):
183 # return theta * data[4]
184 # #
185 # def T(data, tau):
186 # return tau * data[0]
187 # def G(data, tau, sigma):
188 # return (1 - sigma) * tau * data[0]
189 # def calculate(data, *args):
190 # result = [0, 0, 0, 0, 0, 0, 0, 0]
191 # result[0] = (data[1] * min(Q1(data), D1(data)) - data[2]
        * min(L1(data), S1(data)) - data[0]) /
192 # nu
193 # result[1] = (D1(data) - Q1(data)) / mu
```

```
194 # result[2] = (L1(data) - S1(data)) / lmd
195 # result[3] = -dt * data[3] + I1(data, args[0])
196 # result[4] = (math.exp(-ro * args[1] * T(data, args[0]))
    * data[5] * min(Q2(data), D2(data)) -
197 # data[6] * min(L2(data), S2(data)) - data[4]) / nu
198 # result[5] = (D2(data) - Q2(data)) / mu
199 # result[6] = (L2(data) - S2(data)) / lmd
200 # result[7] = -dt * data[7] + I2(data)
201 # return result
202 # def profit(tau, sigma):
203 # initial = [0.0, 0.5, 0.25, 0.1,
204 # 0.0, 0.5, 0.25, 0.1]
205 # stationary = fsolve(calculate, x0=initial, args=(tau,
    sigma))
206 # return -G(stationary, tau, sigma)
207
208
209
210
211
212
213
214
215
216 #
217 # x0, y0 = 3, 3
218 # eps = 0.001
219 # e = 100
220 # A = [[2, 1], [1, 4]]
221 # b = [0, 0]
222
223 # A = [[1, 0], [0, e]]
224 # height = 10
225 # # x0 = get_start_position(height, A, b)
226
227 # bounds = [[-6, 6], [-6, 6]]
228
229
230 # target_func = lambda p: test_func(*p, A, b)
231 # init_p = np.array([x0, y0])
232
```

```
233
234 # grad_iters = []
235
236 # x, y = gradient_descent(target_func, init_p, eps,
      iterations=grad_iters)
237
238 # print("Extremum: ", x, y)
239 # draw_path(target_func, grad_iters, bounds[0], 10)
240 # print("f(x,y) = ", target_func([x, y]))
241
242 # axis_titles = ["x", "y"]
243
244 # sq = 5
245 # bounds = (-sq, -sq), (sq, sq)
246
247 # resolution = 50
248 # ncontours = 15
249
250 # method_names = ["Gradient Descent"]
251
252 # draw(to_minimise, axis_titles, bounds, resolution,
      ncontours, [debug_list], method_names)
253
254 def test_8():
255     x0 = 1
256     y0 = 1
257     init_p = np.array([x0, y0])
258     eps = 0.001
259     angles = np.linspace(0, math.pi*2, 90)
260     # angles = np.linspace(-math.pi/4, math.pi*2 + math.pi
          /4, 90)
261     # angles = np.linspace(-math.pi*2, math.pi*2, 90)
262
263     # for report
264     mins = lambda n: math.pi * n + math.pi/4*3
265     maxs = lambda n: math.pi * n + math.pi/4
266
267
268     es = [1, 10, 100]
269     for e in es:
270         A = [[1, 0], [0, e]]
```

```
271             b = [0, 0]
272
273             y = []
274             for angle in angles:
275                 func = get_rotated_matrix(angle)(test_func)
276                 target_func = lambda p: func(*p, A, b)
277
278                 grad_iters = []
279                 gradient_descent(target_func, init_p, eps,
                      iterations=grad_iters)
280
281                 y.append(len(grad_iters) - 1)
282
283             plt.plot(angles/math.pi * 180, y, label=f'e={e}')
284
285         plt.xlabel('      ')
286         plt.ylabel('                    ␣                ')
287         plt.legend()
288         plt.show()
289
290
291 # test_8()
292 # fig = go.Figure()
293 # for epsilon in epsilons:
294 #     A = [[1, 0], [0, epsilon]]
295 #     b = [0, 0]
296 #     angles = np.linspace(start, end, steps)
297 #     height = epsilon
298
299 #     stats = []
300 #     for theta in angles:
301 #         rotA, rotb = rotate(A, theta), b
302 #         debug_list = []
303
304 #         to_minimise = get_target_function(rotA, rotb)
305 #         x0 = get_start_position(height, rotA, rotb)
306
307 #         x, y = gradient_descent(to_minimise, (x0, 0),
      eps, None, debug_list)
308
309 #     stats.append(len(debug_list) - 1)
```

```
310 #     theta = angles[len(angles) // 2 + 1]
311 #     rotA, rotb = rotate(A, theta), b
312 #     debug_list = []
313 #     to_minimise = get_target_function(rotA, rotb)
314 #     x0 = get_start_position(height, rotA, rotb)
315 #     x, y = gradient_descent(to_minimise, (x0, 0), eps,
    None, debug_list)
316 #     axis_titles = ["x","y"]
317 #     sq = 10
318 #     bounds = (-sq, -sq), (sq, sq)
319 #     res = 50
320 #     draw_path(to_minimise, debug_list, (-4, 4), 10)
321
322
323 #     # fig.add_trace(go.Scatter(x=angles, y=stats,
324 #     # mode='lines',
325 #     # name=f'e = {epsilon}'))
326 #     #
327 #     # fig.update_layout(
328 #     # xaxis_title="Angle",
329 #     # yaxis_title="Number of iterations",
330 #     # )
331 #     #
332
333 # fig.show()
334 #
335
336
337
338
339
340
341
342 # to_minimise = profit
343 # x0, y0 = 0.5, 0.5
344 # eps = 0.01
345 # bounds_vars = ((0.0, 1.0), (0.0, 1.0))
346 # debug_list = []
347 # x, y = gradient_descent(to_minimise, (x0, y0), eps,
    bounds_vars, debug_list)
348 # print("x = ", x, "y = ", y)
```

```
# print("f(x,y) = ", to_minimise(x, y))
# axis_titles = ["tau", "sigma"]
# sq = 1
# bounds = (0.0, 0.0), (1.0, 1.0)
# resolution = 50
# ncontours = 15
# method_names = ["Gradient Descent"]
# print(debug_list)
# f = np.vectorize(to_minimise)
# draw_path(f, debug_list, (0, 1), 10, '$\\tau$', '$\
  sigma$')
```

4.4. services.opt.py

```
import copy
import numpy as np
import math
import matplotlib.pyplot as plt
from colorama import init as colorama_init
colorama_init()
from colorama import Fore, Back, Style


class ABSOptimizationMethod(object):
    method_name = 'optimization␣method'

    def __init__(self):
        self.function_text = 'f(x)'
        self.is_plot = False
        self.is_log = True
        self.is_minimization = True
        self.log_iteration_like_table = True

    def __call__(self, func, bounds, *args, function_text=
      None, **kwargs):
        """                 func

                                        numpy"""
        self._log_start(function_text or self.
          function_text, *bounds)
```

```
23
24          # Decorate func to count number of calls
25          optimiation_func = copy.copy(func)
26          optimiation_func = self.
              optimization_type_function_decorator(
              optimiation_func)
27          optimiation_func = self.function_calls_count(
              optimiation_func)
28
29          result, iterations, *other = self.
              optimization_method(optimiation_func, bounds, *
              args, **kwargs)
30
31          self._log_result(result, func(result))
32          self._log_number_of_iterations(len(iterations))
33
34          if self.is_plot:
35              self._plot_it(func, iterations, *bounds)
36
37          return result, iterations, *other
38
39      def get_interval(self, func, x0, step=0.5):
40          """Sven method"""
41
42          f_l = func(x0 - step)
43          f_r = func(x0 + step)
44          f = func(x0)
45
46          if f_l >= f and f < f_r:
47              return (x0 - step, x0 + step)
48
49          if f_l < f < f_r:
50              step = -step
51
52          p = 1
53          while True:
54              f_new = func(x0 + 2**p * step)
55
56              if f_new >= f:
57                  a = x0 + 2**(p - 2) * step
58                  b = x0 + 2**p * step
```

```
59
60                  return tuple(sorted([a, b]))
61
62              f = f_new
63              p += 1
64
65      def function_calls_count(self, func):
66          self.function_calls = 0
67
68          def wrap(*args, ignore_call=False, **kwargs):
69              if not ignore_call:
70                  self.function_calls += 1
71              return func(*args, **kwargs)
72
73          return wrap
74
75      def optimization_type_function_decorator(self, func):
76          def wrap(*args, ignore_call=False, **kwargs):
77              result = func(*args, **kwargs)
78              if not self.is_minimization:
79                  return -result
80
81              return result
82
83          return wrap
84
85      @staticmethod
86      def _log_decorator(func):
87          def wrap(self, *args, **kwargs):
88              if not self.is_log:
89                  return
90
91              func(self, *args, **kwargs)
92
93          return wrap
94
95      @_log_decorator
96      def _log_iteration(self, idx, x, y, L):
97          if self.log_iteration_like_table:
98              print(Fore.GREEN + f'iteration␣{idx}:' + Style
                .RESET_ALL, end='')
```

```
99              print(f'\tL = {L:.5f}', end='')
100             print(f'\tx = {x:.5f}', end='')
101             print(f'\tf(x) = {y:.5f}', end='')
102             print()
103             return
104
105         print(Fore.GREEN + f'iteration {idx}:' + Style.
              RESET_ALL)
106         print(f'\tL = {L:.5f}')
107         print(f'\tx = {x:.5f}')
108         print(f'\tf(x) = {y:.5f}')
109
110     @_log_decorator
111     def _log_start(self, func_text, l, r):
112         print(Fore.RED + 'Optimization method: ' + self.
              method_name + Style.RESET_ALL)
113         print(Fore.GREEN + 'Initial parameters:' + Style.
              RESET_ALL)
114         print(f'\t{func_text}')
115         print(f'\tl = {l:.5f}, r = {r:.5f}')
116
117     @_log_decorator
118     def _log_result(self, x, y):
119         print(Fore.GREEN + 'Result:' + Style.RESET_ALL)
120         print(f'\tx = {x:.5f}')
121         print(f'\tf(x) = {y:.5f}')
122         print(Fore.GREEN + 'Function calls:' + Style.
              RESET_ALL + str(self.function_calls))
123
124     @_log_decorator
125     def _log_number_of_iterations(self, iters_num):
126         print(Fore.GREEN + 'Number of iterations:' + Style
              .RESET_ALL + str(iters_num))
127
128     def _plot_it(self, func, iterations, l, r, steps =
          1000):
129         color_function = '#1050A8'
130         color_iteration = '#FFF702'
131         color_extremum = '#F30223'
132
133         x = np.linspace(l, r, steps)
```

```
134             # y = list(map(func, x))
135             y = func(x)
136             iterations_y = list(map(func, iterations))
137 
138             plt.xlabel('x')
139             plt.ylabel('f(x)')
140 
141             #
142             plt.plot(x, y, color=color_function, zorder=0)
143 
144             #
145             plt.scatter(
146                 iterations, iterations_y,
147                 label=f'iteration␣[n={len(iterations)}]',
148                 color=color_iteration,
149                 zorder=1,
150                 s=30,
151             )
152 
153             #
154             plt.scatter(
155                 [iterations[-1]], [iterations_y[-1]],
156                 label=f'extremum␣[x={iterations[-1]:.5f},␣y={
                      iterations_y[-1]:.5f}]',
157                 color=color_extremum,
158                 zorder=1,
159                 s=40,
160             )
161 
162             #
163             for i in range(len(iterations)):
164                 plt.annotate(str(i + 1), (iterations[i],
                      iterations_y[i]), ha='center', va='bottom')
165 
166             plt.legend()
167             plt.show()
168 
169         def optimization_method(self):
170             ...
```

```
171
172
173 class DichotomyMethod(ABSOptimizationMethod):
174     method_name = 'Dichotomy method'
175
176     def optimization_method(self, func, bounds, eps):
177         l, r = bounds
178
179         iterations = []
180         idx = 0
181
182         while r - l >= eps:
183             temp_x = (l + r) / 2
184             x1 = temp_x - eps / 2
185             x2 = temp_x + eps / 2
186
187             if func(x1) > func(x2):
188                 l = temp_x
189             else:
190                 r = temp_x
191
192             self._log_iteration(idx, temp_x, func(temp_x,
                  ignore_call=True), (r - l))
193
194             iterations.append(temp_x)
195             idx += 1
196
197         result = (l + r) / 2
198         iterations.append(result)
199
200         #
201         self._log_iteration(idx, result, func(result,
              ignore_call=True), (r - l))
202
203         return result, iterations
204
205
206 phi = golden_ratio = (1 + math.sqrt(5)) / 2
207
208 class GoldenSectionMethod(ABSOptimizationMethod):
```

```
209     method_name = 'Golden section method'
210
211     def optimization_method(self, func, bounds, eps,
          max_iter=100):
212         l, r = bounds
213         iterations = []
214         d = (r - l)
215         ind = 0
216         interval_changes = 0
217
218         while (r - l) >= eps:
219             d = d / phi
220             x1 = r - d
221             x2 = l + d
222
223             temp_x = (l + r) / 2
224             self._log_iteration(ind, temp_x, func(temp_x,
                  ignore_call=True), (r - l))
225
226             if func(x1) <= func(x2):
227                 r = x2
228             else:
229                 l = x1
230
231             iterations.append(temp_x)
232             interval_changes += 1
233             ind += 1
234
235         result = (l + r) / 2
236
237         return result, iterations, interval_changes, self.
            function_calls
238
239
240 class FibonacciMethod(ABSOptimizationMethod):
241     method_name = 'Fibonacci method'
242
243     @staticmethod
244     def fib(n):
245         a = 0
246         b = 1
```

```
247
248         if n == 0:
249             return a
250
251         if n == 1:
252             return b
253
254         for i in range(1, n):
255             a, b = b, a + b
256
257         return b
258
259     def optimization_method(self, func, bounds, eps):
260         fib = self.fib
261         l, r = bounds
262         iterations = []
263         ind = 0
264         interval_changes = 0
265
266         n = 0
267         while fib(n) <= (r - l) / (eps):
268             n += 1
269
270         n = max(n, 3)
271
272         for k in range(n - 2):
273             p = (fib(n - k - 1) / fib(n - k))
274             x1 = l + (r - l) * (1 - p)
275             x2 = l + (r - l) * p
276
277             temp_x = (l + r) / 2
278             self._log_iteration(ind, temp_x, func(temp_x,
                  ignore_call=True), (r - l))
279
280             if func(x1) <= func(x2):
281                 r = x2
282             else:
283                 l = x1
284
285             iterations.append(temp_x)
286             interval_changes += 1
```

```
287                 ind += 1
288
289             result = (l + r) / 2
290
291             return result, iterations, interval_changes, self.
                  function_calls
```

4.5. services.model.py

```
1   import math
2   from scipy.optimize import fsolve
3   from collections.abc import Iterable
4
5
6   class EconomicModel(object):
7       def __init__(self):
8           self.set_default()
9
10      def set_default(self):
11          self.alpha = 0.5
12          self.beta = 1.5
13          self.gamma = 1.5
14          self.delta = 0.1
15          self.nu = 5
16          self.mu = 20
17          self.lambda_ = 20
18          self.rho = 10
19          self.A0 = 1
20          self.L0 = 1
21          self.D0 = 1
22          self.tau = 0.6
23          self.sigma = 0.5
24          self.theta = (1 + self.alpha * (self.beta - 1)) **
                (-1)
25
26      def L1(self, x):
27          return x[3] * ((1 - self.alpha) * self.A0 * x[1] /
                x[2]) ** (1 / self.alpha)
28
29      def Q1(self, x):
```

```
30          return self.A0 * x[3] ** self.alpha * self.L1(x)
              ** (1 - self.alpha)
31
32      def D1(self, x):
33          return self.D0 * math.exp(-self.beta * x[1]) * x
              [5] / (x[1] + x[5])
34
35      def S1(self, x):
36          return self.L0 * (1 - math.exp(-self.gamma * x[2])
              ) * x[2] / (x[2] + x[6])
37
38      def I1(self, x):
39          return (1 - self.tau) * (1 - self.theta) * x[0]
40
41      def L2(self, x):
42          return x[7] * ((1 - self.alpha) * self.A0 * x[5] /
               x[6]) ** (1 / self.alpha)
43
44      def Q2(self, x):
45          return self.A0 * x[7] ** self.alpha * self.L2(x)
              ** (1 - self.alpha)
46
47      def D2(self, x):
48          return self.D0 * math.exp(-self.beta * x[5]) * x
              [1] / (x[1] + x[5])
49
50      def S2(self, x):
51          return self.L0 * (1 - math.exp(-self.gamma * x[6])
              ) * x[6] / (x[2] + x[6])
52
53      def I2(self, x):
54          return (1 - self.theta) * x[4]
55
56      def T(self, x):
57          return self.tau * x[0]
58
59      def G(self, x):
60          """                                          """
61          return (1 - self.sigma) * self.tau * x[0]
62
63      def G1(self, x):
```

```
64          """
               """
65          return (1 - self.tau) * self.theta * x[0]
66
67      def G2(self, x):
68          """                                                    """
69          return self.theta * x[4]
70
71      def count_model(self, x):
72          P1 = (x[1] * min(self.Q1(x), self.D1(x))\
73              - x[2] * min(self.L1(x), self.S1(x)) - x[0]) /
                 self.nu
74
75          p1 = (self.D1(x) - self.Q1(x)) / self.mu
76
77          w1 = (self.L1(x) - self.S1(x)) / self.lambda_
78
79          K1 = -self.delta * x[3] + self.I1(x)
80
81          P2 = (math.exp(-self.rho * self.sigma * self.T(x))
               \
82              * x[5]\
83              * min(self.Q2(x), self.D2(x)) - x[6]\
84              * min(self.L2(x), self.S2(x)) - x[4]) / self.
                 nu
85
86          p2 = (self.D2(x) - self.Q2(x)) / self.mu
87
88          w2 = (self.L2(x) - self.S2(x)) / self.lambda_
89
90          K2 = -self.delta * x[7] + self.I2(x)
91
92          result = [P1, p1, w1, K1, P2, p2, w2, K2]
93          return result
94
95      def __call__(self, x, changed_params={}):
96          for param_name in changed_params:
97              if not hasattr(self, param_name):
98                  raise ValueError(param_name)
99              setattr(self, param_name, changed_params[
                     param_name])
```

```
100
101         result = self.count_model(x)
102         self.set_default()
103         return result
104
105
106 def G_by_x(model: EconomicModel, tau: float, init: list[
      float] | None = None) -> float:
107     if isinstance(tau, Iterable):
108         return list(map(lambda x: G_by_x(model, x), list(
              tau)))
109
110     prepared_model_call = lambda x, *args: model(x,
          changed_params={"tau": args[0]})
111
112     #
113     if init is None:
114         init = [0, 0.5, 0.25, 0.1, 0, 0.5, 0.25, 0.1]
115     new_x = fsolve(prepared_model_call, x0=init, args=(tau
          ,))
116
117     model.tau = tau
118     result = model.G(new_x)
119
120     #
121     model.set_default()
122
123     return result
```