4.3. Практическое занятие № 2 (по теме 4). РАЗРАБОТКА ГЕНЕТИЧЕСКОГО АЛГОРИТМА ПОИСКА ЭКСТРЕМУМА ФУНКЦИИ. Задание и пример выполнения

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ № 2 (по теме 4)

«РАЗРАБОТКА ГЕНЕТИЧЕСКОГО АЛГОРИТМА ПОИСКА ЭКСТРЕМУМА ФУНКЦИИ»

Задание

Реализовать на языке программирования (по Вашему выбору) программу поиска оптимума (максимума/минимума) функции F(X) на интервале [A,B] с использованием генетического алгоритма.

Требования к программе:

1. Ввод значения интервалов A и B
2. Ввод типа оптимума: минимум или максимум
3. Ввод количества популяций генетического алгоритма.
4. Ввод параметра d-отображения графика (см.ниже).
5. По желанию: ввод/изменение параметров генетического алгоритма:
    a. Размер популяции.
    b. Вероятность cut-point.
    c. Коэффициент скрещивания.
    d. Коэффициент мутации.
    e. Что-то еще – на усмотрение разработчика.
6. Отображение графика функции на интервале [A–d, B+d], где d-задается (см.выше).
7. Отображение на графике функции популяций на каждом шаге, зеленым цветом выделять наиболее жизнеспособные, красным – наименее.
8. Исследовать работу алгоритма на различных функциях (по варианту, см ниже).

Пример выполнения работы

Разработку будем вести на языке Python. Нам понадобятся следующие Фреймворки (устанавливаем в терминалеPyCharm по команде pip):

1. PyQt5 – Пользовательский интерфейс(GUI): форма для ввода параметров: `pip install PyQt5`.
2. Sympy – библиотека, для обработки вводимых пользователем функций: `pip install sympy`.
3. Numpy – математическая библиотека: `pip install numpy`
4. Matplotlib – библиотека для создания графиков: `pip install matplotlib`

Разработаем класс для «особи» (точки на плоскости), назовем этот класс Agent. Агент может посчитать свое Y-значение (фактически вычислить фитнесс-функцию), а также провести операцию мутации (сделать случайных «отскок» по оси X). В функцию мутации передаются параметры – сила мутации (strange) и частота (chast). Сила мутации в данном примере представляет собой величину вариации по оси X, т.е. на какое значение интервала от исходной «особи» (точки) будет создана новая «мутированная особь» (новая точка). Сила мутации может быть также задана случайным образом.

```python
class agent():
    def __init__(self, x):
        self.X = x
        self.Y = None
    def mutate(self, strange, chast):
        if np.random.rand() <= chast:
            if np.random.rand() > 0.5:
                self.X += strange
            else:
                self.X -= strange
    def calculate(self, fun):
        self.Y = fun(self.X)

    def print(self):
        print(self.X, self.Y)

    def rX(self):
        return self.X

    def rY(self):
        return self.Y
```

Разработаем класс самого приложения – mywindow.

1. Конструктор _init__ отвечает за инициализацию стартовых значений (здесь связь их с полями интерфейса)
2. Функция Start – главная функция приложения. В этой функции происходит следующее:
    - Инициализация значений.
    - Обработка введенной функции.
    - Создание первого поколения агентов.
    - Цикл где ищутся худшие особи, чтобы их впоследствии отсеять (оставляем 50 процентов).
    - Добавляются точки на график.
    - Создаются новые особи (агенты) и происходит их мутация.

Оставшиеся функции вспомогательные, комментарии к ним приведены в листинге.

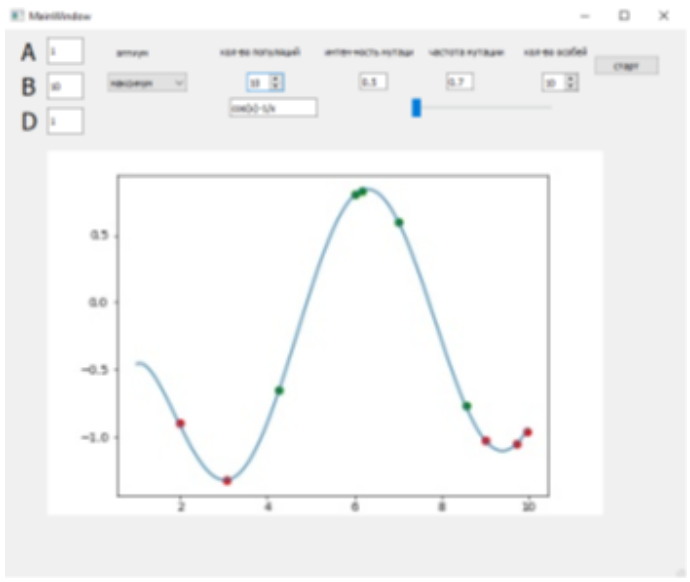
```python
class mywindow(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self):
        super().__init__()
        self.setupUi(self)
        self.StartBtn.clicked.connect(self.start)
        self.Slider.valueChanged.connect(self.slide)
        self.StopBtn.setVisible(False)
        self.count = 0
    def start(self): #создаем UI (пользовательский интерфейс)
        A = int(self.AEdit.text())
        B = int(self.BEdit.text())
        D = int(self.DEdit.text())
        tp = self.TypeBox.currentText()
        countpop = self.CountPopSB.value()
        strengMut = float(self.MutStrange.text())
        chastMut = float(self.MutChast.text())
        countos = self.CountOsSB.value()
        self.Slider.setMaximum(countpop-1)
        self.agents = []        #INIT FUNC
        x = sympy.Symbol('x')
        fun = sympy.lambdify(x, self.FunEdit.text())
        ar = np.array(self.gen_x(A, B))
        xx, y = self.get_y(fun, ar)
        #создаем точки, особи (агенты)
        for i in range(countos):
            par = (B - A) * np.random.rand() + A
            self.agents.append(agent(par))
            self.agents[-1].calculate(fun)
        for i in range(countpop):
```

```
            self.drawplot(xx, y) #далее ищем  "плохие" точки
            for k in range(int(countos / 2)):
                minim = 1000
                maxim = -1000
                index = 0
                ind = 0
                if tp == 'максимум':
                    for d in self.agents:
                        if d.rY() < minim:
                            minim = d.rY()
                            index = ind
                        ind += 1
                elif tp == 'минимум':
                    for d in self.agents:
                        if d.rY() > maxim:
                            maxim = d.rY()
                            index = ind
                        ind += 1
                self.drawpoint(self.agents[index].rX(), self.agents[index].rY(), 'r')
                del self.agents[index]
            for f in self.agents:
                self.drawpoint(f.rX(), f.rY(), 'g')
            self.saveplt(i)
            #Генерируем следующее поколение
            secund = 0
            for dl in range(int(countos / 2)):
                self.agents.append(agent(self.agents[secund].rX()))
                secund += 1
            for ag in range(len(self.agents)):
                self.agents[ag].mutate(strengMut, chastMut)
                self.agents[ag].calculate(fun)
        self.show_graph(0)
    def slide(self): #slider`s event-function
        self.show_graph(self.Slider.value())

    def show_graph(self, name):  # функция для вывода графика (открывается в папке - tmp )
        self.mypix = QPixmap("./tmp/" + str(name) + ".jpg").scaled(640, 480)
        self.Graph.setPixmap(self.mypix)
    def gen_x(self, A, B): #генерируем массив между  A и B.
        shag = 0.01
        ret = []
        while A < B:
            ret.append(A)
            A += shag
        return ret
    def get_y(self, fun, ar): #функция преобразования массива Y в формат numpy
        ret = np.zeros(len(ar))
        for ind, i in enumerate(ar):
            ret[ind] = fun(i)
        return ar, ret
    def drawplot(self, xx, y): #функция прорисовки графика
        fig = plt.figure()
        self.ax = fig.add_subplot(1, 1, 1)
        self.ax.plot(xx, y)
    def drawpoint(self, x, y, color): #функция для добавления точки к графику
        self.ax.scatter(x, y, c=color)
    def saveplt(self, name): #функция для сохранения графика
        plt.savefig('./tmp/' + str(name) + '.jpg', dpi=50)
        plt.close()


app = QtWidgets.QApplication([])
application = mywindow()
application.show()
sys.exit(app.exec())
```

Скриншоты выполнения программы



A – задаем левую границу, B – задаем правую границу, D – плюс к отображению.

Оптимум – ищем максимумы или минимумы.

Кол-во популяций – сколько будет создано популяций.

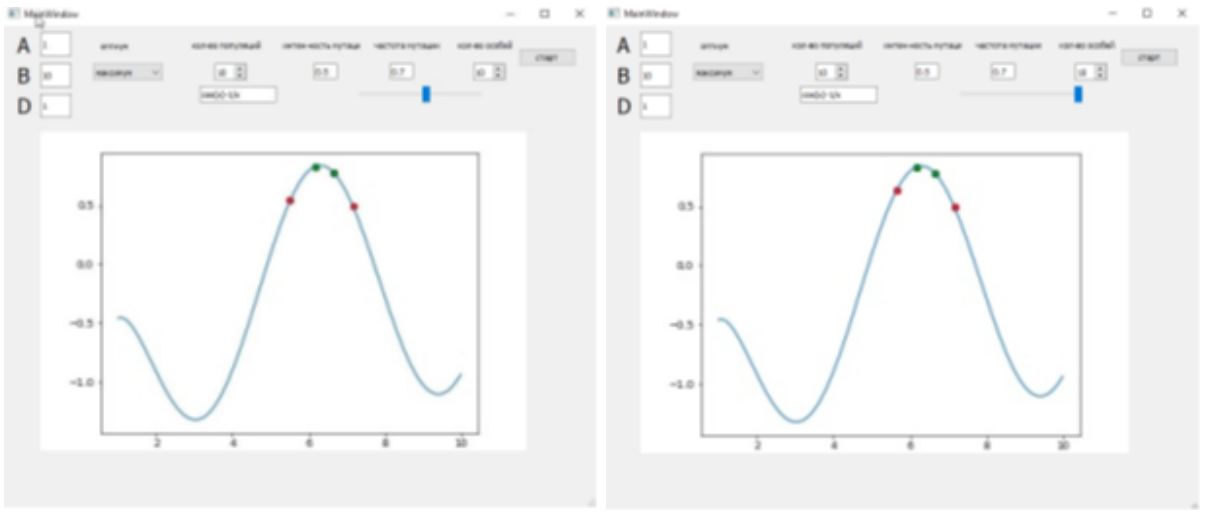
Интенсивность мутации – на какое значение в + или – мутирует агент.

Частота мутации – вероятность мутации.

Количество особей – количество агентов в поколении.

Кнопка старт – начало работы.

Слайдер – после окончания работы, позволит перемещаться по поколениям.



Варианты заданий

В таблице ниже указан номер варианта и звёздочкой – номер функции для исследования. В каждом варианте 3 функции.

Список функций для исследования.

1. F(x) = sin(x)-x/3
2. F(x) = sin(x)+x/3
3. F(x) = cos(x)-x/3
4. F(x) = cos(x)+x/3
5. F(x) = 2*x*sin(x)-x/5
6. F(x) = 3* sin(x)-x/3
7. F(x) = 4* x*sin(x)+x/4

8. F(x) = 2*sin(x)+x/5
9. F(x) = 3* x*sin(x)+x/3
10. F(x) = 4*sin(x)+x/4
11. F(x) = 2* cos (x)-x/5
12. F(x) = 3* x*cos (x)-x/3
13. F(x) = 4* cos (x)+x/4
14. F(x) = 2* x*cos (x)+x/5
15. F(x) = 3* cos (x)+x/3
16. F(x) = 4* x*cos (x)+x/4
17. F(x) = 2* x*sin(x)+x*cos(x)
18. F(x) = 3*sin(x)-x*cos(x)/3
19. F(x) = 4* x*sin(x)+x*cos(x)
20. F(x) = 5*cos(x)-x*cos(x)/2
21. F(x) = 2* x*cos(x)+x* sin (x)
22. F(x) = 3*cos(x)-x* sin (x)/3
23. F(x) = 4* x*cos(x)+x* sin (x)
24. F(x) = 5*cos(x)-x* sin (x)/2
25. F(x) = cos(x)*x* sin (x)/3
26. F(x) = 5*cos(x)*x^2* sin (x)
27. F(x) = cos(x)*x^2/5-sin (x)

№ функции	Вариант 1	Вариант 2	Вариант 3	Вариант 4	Вариант 5	Вариант 6	Вариант 7	Вариант 8	Вариант 9	Вариант 10	Вариант 11	Вариант 12	Вариант 13	Вариант 14	Вариант 15	Вариант 16	Вариант 17	Вариант 18	Вариант 19	Вариант 20
1.	*							*				*								
2.		*							*				*							
3.			*					*		*										
4.				*					*		*									
5.	*				*							*								
6.		*				*							*							
7.			*				*							*						
8.				*				*							*					
9.					*				*							*				
10.	*					*				*										
11.		*					*				*									
12.			*									*		*						
13.				*									*		*					
14.					*									*		*				
15.						*				*					*					
16.							*				*					*				
17.																	*			
18.																		*		
19.																			*	
20.																				*
21.																	*			
22.																		*		
23.																			*	
24.																	*			*
25.																		*		
26.																			*	
27.																				*

Последнее изменение: Вторник, 10 мая 2022, 15:43

◄ 4.2. Видеолекция - Эволюционное моделирование в интеллектуальных системах

Перейти на...

4.4. Отчет по практическому занятию № 2 ►