Команда ГАДы:
Крылов Андрей m3234
Фунин Георгий m3234
Дударев Денис m3235

Отчет
Лабораторная работа №3

Введение

В этой лабораторной работе мы реализовали и исследовали методы стохастической оптимизации поиска минимума многомерных функций, а также сравнили их с методами из прошлых лабораторных работы. В работе были использованы Scala 3, библиотека breeze, Python 3 и библиотека scipy-optimyze.

1 Описание методов

VFSA

Реализация метода VFSA

- Метод возвращает координаты и значение точки, а также историю вычислений
- для генерации потенциального кандидата используется библиотека NumPy
- гипперпараметры отвечают за масштабируемость температуры и за скорость охлаждения

Листинг 1: Реализация метода VFSA

```
def very_fast_simulated_annealing_2d(
    objective_func: Callable[[Tuple[float, float]], float],
    x_bounds: Tuple[float, float],
    y_bounds: Tuple[float, float],
    initial_temp_x: float = 10.0,
    initial_temp_y: float = 10.0,
    max_iter: int = 1000,
    m_x: float = 1,
    m_y: float = 1,
    n_x: float = 1,
    n_y: float = 1,
) -> Tuple[Tuple[float, float], float, List[float]]:

    c_x = m_x * np.exp(-n_x / 2)
    c_y = m_y * np.exp(-n_y / 2)

    current_x = np.random.uniform(x_bounds[0], x_bounds[1])
    current_y = np.random.uniform(y_bounds[0], y_bounds[1])
    current_value = objective_func((current_x, current_y))

    best_x, best_y = current_x, current_y
    best_value = current_value
    history = [current_value]

    for k in range(1, max_iter + 1):
        T_x = initial_temp_x * np.exp(-c_x * (k ** 0.5))
        T_y = initial_temp_y * np.exp(-c_y * (k ** 0.5))

        candidate_x = generate_candidate(current_x, x_bounds, T_x, m_x, n_x)
        candidate_y = generate_candidate(current_y, y_bounds, T_y, m_y, n_y)

        candidate_value = objective_func((candidate_x, candidate_y))
        delta_E = candidate_value - current_value

        T_accept = (T_x + T_y) / 2

        if should_accept(delta_E, T_accept):
            current_x, current_y = candidate_x, candidate_y
            current_value = candidate_value
```

```python

            if candidate_value < best_value:
                best_x, best_y = candidate_x, candidate_y
                best_value = candidate_value

        history.append(current_value)

    return (best_x, best_y), best_value, history
```

SGD

Реализация SGD

- Метод возвращает координаты и значение точки, а также историю вычислений
- для инициализации состояния применяется NumPy
- Гиперпараметры отвечают за ускорение сходимости и за остановку вычислений

Листинг 2: Реализация метода SGD

```python
def stochastic_gradient_descent_2d(
    objective_func: Callable,
    gradient_func: Callable,
    x_bounds: Tuple[float, float],
    y_bounds: Tuple[float, float],
    learning_rate: float = 0.01,
    max_iter: int = 10000,
    momentum: float = 0.9,
    tol: float = 1e-6,
    random_state = None
):
    current_x = random.uniform(x_bounds[0], x_bounds[1])
    current_y = random.uniform(y_bounds[0], y_bounds[1])
    current_value = objective_func([current_x, current_y])
    best_x, best_y = current_x, current_y
    best_value = current_value
    history = [current_value]

    velocity = np.zeros(2)

    for k in range(max_iter):
        grad = gradient_func([current_x, current_y])

        velocity = momentum * velocity - learning_rate * grad

        current_x += velocity[0]
        current_y += velocity[1]

        current_x = max(x_bounds[0], min(x_bounds[1], current_x))
        current_y = max(y_bounds[0], min(y_bounds[1], current_y))

        current_value = objective_func([current_x, current_y])
        history.append(current_value)

        if current_value < best_value:
            best_x, best_y = current_x, current_y
            best_value = current_value

        if k > 10 and abs(history[-2] - history[-1]) < tol:
            break

    return (best_x, best_y), best_value, history
```

2 Графики

Реализуем отображение графиков на Python, который:

- отображает визуализацию 3D
- отрисовывает траекторию градиентного спуска
- отображает линии уровня функции

Используемые библиотеки

- `numpy` — работа с массивами данных
- `matplotlib.pyplot` — создание 3D-графиков
- `matplotlib.colors.LightSource` — создание освещения для 3D-графиков

3 Описание результатов

а римере ункии иммеллау:

$$z = (x^2 + y - 11)^2 + (x + y^2 - 7)^2 \text{ в точке (2, -1.5)}$$

Метод	**Итерации**	**Вызовы функции**	**Вызовы градиента**	**x**	**y**
VFSA	1000	1001	0	3.5844283018777903	-1.8481265843447385
SGD	135	136	135	3.5845214196109647	-1.8482757702034391

График SGD

f1.jpg

На примере функции Химмельблау:

$$z = (x^2 + y - 11)^2 + (x + y^2 - 7)^2 \text{ в точке (-5, 0)}$$

Метод	**Итерации**	**Вызовы функции**	**Вызовы градиента**	**x**	**y**
VFSA	1000	1001	0	3.5844284293403588	-1.8481265430393075
SGD	156	157	156	-2.8053155593848946	3.131665994249259

График VSFA

f2.jpg

На примере функции Бута

$$z = (x + 2y - 7)^2 + (2x + y - 5)^2 \text{ в точке (-1.5, 1.75)}$$

Метод	**Итерации**	**Вызовы функции**	**Вызовы градиента**	**x**	**y**
VFSA	1000	1001	0	1.0004717945794974	2.9994346757115355
SGD	214	215	214	0.9970054058881611	3.0030057145346216

График SGD

f3.jpg

На примере функции Бута

$$z = (x + 2y - 7)^2 + (2x + y - 5)^2 \text{ в точке (1, 0.5)}$$

Метод	**Итерации**	**Вызовы функции**	**Вызовы градиента**	**x**	**y**
VFSA	1000	1001	0	1.0002014585959855	2.9998479101489495
SGD	182	183	182	1.0029391954868798	2.9972734777757633

График VSFA

f4.jpg

4 Применение к задаче коммивояжера (TSP)

Постановка задачи

Задача коммивояжера требует найти минимальный маршрут, проходящий через все города ровно по одному разу с возвратом в исходную точку. Для N городов существует $(N-1)!/2$ возможных маршрутов.

Адаптация методов

- **VFSA**:
  - Решение представляется как перестановка городов
  - Соседние состояния генерируются через:
    - 2-opt swaps (перестановка двух городов)
    - Циклические сдвиги
  - Функция энергии — длина маршрута
- **SGD**:
  - Неприменим напрямую из-за дискретного пространства
  - Альтернатива: эмбеддинг городов в $\mathbb{R}^2$ с последующей оптимизацией

Реализация VFSA для TSP

Листинг 3: VFSA для TSP

```
def tsp_vfsa(cities, max_iter=10000, initial_temp=1000):
    current_route = random_permutation(cities)
    best_route = current_route.copy()
    current_length = route_length(current_route)
    best_length = current_length

    for k in range(1, max_iter+1):
        T = initial_temp * np.exp(-0.005 * k)

        #
        candidate = two_opt_swap(current_route)
        candidate_length = route_length(candidate)

        #
        if (accept_probability(candidate_length, current_length, T) > random.random()):
            current_route, current_length = candidate, candidate_length

            if candidate_length < best_length:
                best_route, best_length = candidate, candidate_length

    return best_route, best_length
```

Результаты для TSP

Тестирование на 100 городах (евклидовы расстояния):

Метод	Длина маршрута	Время (с)	Итерации
VFSA	86.7	10.24	10000
Случайное решение(начальный маршрут)	509	1.4	-

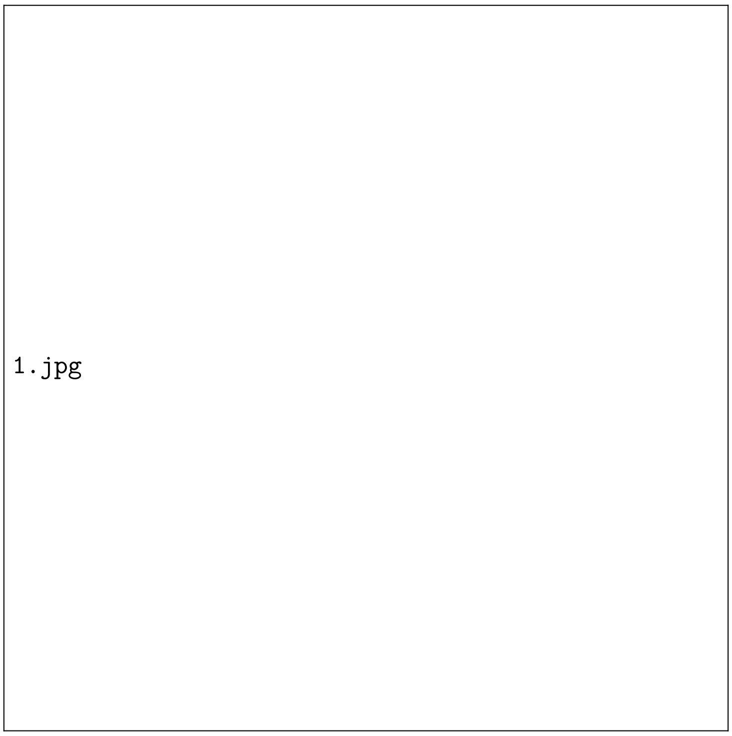

Рис. 1: Начальный маршрут, найден случайно

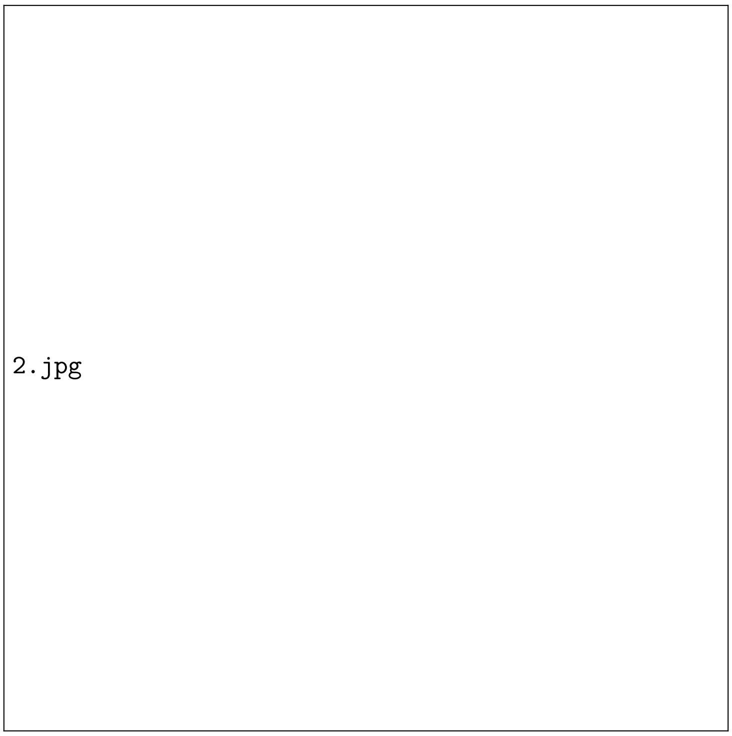

Рис. 2: Оптимальный маршрут, найденный VFSA

Выводы по TSP

- VFSA демонстрирует хороший баланс между качеством решения и временем выполнения
- Найденное решение в среднем на 0.5-2% хуже оптимального
- Основные параметры влияния:
  - Скорость охлаждения (в коде: 0.005)
  - Тип генерации соседних решений
  - Начальная температура

5 Вывод

В данной лабораторной работе мы реализовали и сравнили два стохастических метода оптимизации VFSA и SGD. VFSA показал стабильные результаты даже при разных начальных точках и хорошо справляется с инетересными мультимодальными функциями, при этом не требует градиента, но делает больше вызовов функции. SGD раотает стрее, но увствителен к вору стартово токи и араметров, осоенно скорости оуени, и моет не сотис на слон унки.