Отчет по лабораторной работе №4

**Методы стохастической оптимизации**

*Аксенова Валерия, Шустров Андрей*

***Описание методов:***

*Имитация отжига (simulated annealing)* - метод реализован в классическом виде с оптимизацией досрочной остановки, если результат перестает обновляться. В качестве гиперпараметров принимаются следующие аргументы – начальная температура, температура остановки, коэффициент охлаждения и количество итераций, сделанных на каждом значении температуры.

*Эволюционная стратегия с адаптацией матрицы ковариации (CMA-ES) –* стандартная реализация. При инициализации алгоритма задается размер шага, размер популяции для обучения, а также ограничение на количество итераций. Ключевым преимуществом алгоритма является его способность адаптировать форму распределения поиска к ландшафту целевой функции путем обновления матрицы ковариации, что позволяет эффективно оптимизировать сложные мультимодальные функции.

***Исследование:***

Для исследования из предыдущих лабораторных были взяты наиболее интересные функции: Розенброка, Химмельблау, Букина и Растригина, а также датасет *California\_housing* из *scikit-learn.*

***Имитация отжига:***

Для сравнения эффективности были выбраны одни из наиболее интересных, по предыдущим лабораторным работам, методов: BFGS, SR1 и алгоритм Нельдера-Мида. Реализации взяты из библиотеки *SciPy*. Метриками для сравнения являются: количество итераций алгоритма, количество вычислений исходной функции и ее производных и абсолютное отклонение от оптимума.

| **ROSENBROCK** | *Итераций* | *Оценок функции* | *Ошибка* |
| --- | --- | --- | --- |
| *Simulated Annealing* | 224 000 | 448 001 | 225.928039 |
| *BFGS* | 458 | 1 800 | 0.000000 |
| *SR1* | 269 | 807 | 0.000000 |
| *Nelder–Mead* | 376 | 699 | 0.000000 |

| ***HIMMEBLAU*** | Итераций | Оценок функции | Ошибка |
| --- | --- | --- | --- |
| *Simulated Annealing* | 213 000 | 426 001 | 0.000001 |
| *BFGS* | 48 | 162 | 0.000000 |
| *SR1* | 57 | 138 | 0.000000 |
| *Nelder–Mead* | 64 | 125 | 0.000000 |

| **RASTRIGIN** | *Итераций* | *Оценок функции* | *Ошибка* |
| --- | --- | --- | --- |
| *Simulated Annealing* | 213 000 | 426 001 | 0.000001 |
| *BFGS* | 12 | 51 | 3.800814 |
| *SR1* | 31 | 90 | 1.902843 |
| *Nelder–Mead* | 59 | 117 | 0.951422 |

| **CALIFORNIA HOUSING** | *Итераций* | *Оценок функции* | *Ошибка* |
| --- | --- | --- | --- |
| *Simulated Annealing* | 199 000 | 398 001 | 0.002516 |
| *BFGS* | 16 | 180 | 0.000321 |
| *SR1* | 21 | 210 | 0.000321 |
| *Nelder–Mead* | 1 000 | 1 435 | 1.065544 |

Во всех задачах отжиг оказался существенно медленнее и менее точным, чем градиентные и квазиньютоновские методы. Для функции Розенброка метод отжига не сошелся вовсе. В гладких непрерывных задачах с небольшим числом переменных отжиг уступает по эффективности и точности методам, использующим градиентную информацию.

***Эволюционная стратегия с адаптацией матрицы ковариации:***

***Rosenbrock***

| **σ** | **Размер популяции** | **Ошибка** | **Вычислений функции** | **Итераций** |
| --- | --- | --- | --- | --- |
| 0.1 | default | 0.072746 | 6 000 | 1 000 |
| 0.1 | 10 | 0.000000 | 6 160 | 616 |
| 0.1 | 20 | 0.000000 | 8 640 | 432 |
| 0.3 | default | 0.518334 | 6 000 | 1 000 |
| 0.3 | 10 | 0.000000 | 7 500 | 750 |
| 0.3 | 20 | 0.000000 | 4 720 | 236 |
| 0.5 | default | 0.000000 | 4 722 | 787 |
| 0.5 | 10 | 0.000000 | 6 890 | 689 |
| 0.5 | 20 | 0.000000 | 9 800 | 490 |

***Himmelblau***

| **σ** | **Размер популяции** | **Ошибка** | **Вычислений функции** | **Итераций** |
| --- | --- | --- | --- | --- |
| 0.1 | default | 0.000000 | 918 | 153 |
| 0.1 | 10 | 0.000000 | 1 330 | 133 |
| 0.1 | 20 | 0.000000 | 1 980 | 99 |
| 0.3 | default | 0.000000 | 870 | 145 |
| 0.3 | 10 | 0.000000 | 1 190 | 119 |
| 0.3 | 20 | 0.000000 | 1 940 | 97 |
| 0.5 | default | 0.000000 | 930 | 155 |
| 0.5 | 10 | 0.000000 | 1 250 | 125 |
| 0.5 | 20 | 0.000000 | 1 960 | 98 |

***Rastrigin***

| **σ** | **Размер популяции** | **Ошибка** | **Вычислений функции** | **Итераций** |
| --- | --- | --- | --- | --- |
| 0.1 | default | 0.000000 | 924 | 154 |
| 0.1 | 10 | 0.000000 | 1 180 | 118 |
| 0.1 | 20 | 0.000000 | 1 940 | 97 |
| 0.3 | default | 0.000000 | 870 | 145 |
| 0.3 | 10 | 0.000000 | 1 160 | 116 |
| 0.3 | 20 | 0.000000 | 1 920 | 96 |
| 0.5 | default | 0.000000 | 876 | 146 |
| 0.5 | 10 | 0.000000 | 1 230 | 123 |
| 0.5 | 20 | 0.000000 | 1 880 | 94 |

***California housing***

| **σ** | **Размер популяции** | **Ошибка** | **Вычислений функции** | **Итераций** |
| --- | --- | --- | --- | --- |
| 0.1 | default | 0.000337 | 10 000 | 1 000 |
| 0.1 | 10 | 0.000339 | 10 000 | 1 000 |
| 0.1 | 20 | 0.000321 | 3 100 | 155 |
| 0.3 | default | 0.001034 | 8 950 | 895 |
| 0.3 | 10 | 0.000443 | 7 510 | 751 |
| 0.3 | 20 | 0.000321 | 3 080 | 154 |
| 0.5 | default | 0.000334 | 9 900 | 990 |
| 0.5 | 10 | 0.000321 | 2 440 | 244 |
| 0.5 | 20 | 0.000321 | 2 780 | 139 |

Для функции Розенброка оказались наиболее эффективны маленькие популяции: они обеспечивают надежный выход в оптимум быстрее, несмотря на то что на каждом шаге выполняется чуть больше проверок. Для Химмельблау и Растригина любая комбинация параметров всё же сходится к идеальному решению, и здесь главный выигрыш даёт увеличение размера популяции — оно сокращает число шагов, хотя и требует больше функций за итерацию. В задаче California housing наилучший баланс между скоростью и точностью достигается при умеренных настройках смещения и относительно небольшой популяции: это позволяет быстро приблизиться к минимуму, не тратя лишних вычислительных ресурсов.

***Задача OneMax с помощью Генетического алгоритма***

***Постановка задачи:***

Задача *OneMax* — это классическая задача в области стохастической оптимизации, которая часто используется для иллюстрации работы генетических алгоритмов (*ГА*). Суть задачи заключается *в максимизации количества единиц в бинарной строке фиксированной длины.*

***Задача оптимизации:***

Пусть дана бинарная строка. Наша *цель* – найти решение, которое бы давало максимальную сумму цифр этого списка:



Здесь N – длина списка. Очевидно, лучший, наиболее приспособленный индивид, должен состоять из всех единиц.

***Описание методов:***



*1.* ***Оценка приспособленности (фитнес-функция)*** *oneMaxFitness(individual):*

Возвращает количество единиц в бинарной строке — это и есть мера "приспособленности". Чем больше единиц, тем лучше решение.

*2.* ***Инициализация популяции****:*

*individualCreator()* — создает одного индивидуума с бинарной строкой длины ONE\_MAX\_LENGTH.

*populationCreator(n)* — формирует начальную популяцию из n таких индивидуумов.

*3.* ***Отбор (селекция)*** *selTournament(population, p\_len)*:

Применяется турнирный отбор: из трех случайно выбранных особей побеждает та, у которой максимальная приспособленность. *.* Это позволяет сохранить давление отбора и увеличить шансы лучших решений на размножение.

*4.* ***Скрещивание (кроссовер)*** *def cxOnePoint(child1, child2):*

Используется одноточечное скрещивание: случайно выбирается точка разреза, после которой потомки обмениваются хвостами хромосом..

Скрещивание выполняется с заданной вероятностью *P\_CROSSOVER*.

*5.* ***Мутация*** *mutFlipBit(mutant, indpb=0.01):*

Мутация инвертирует каждый бит хромосомы с небольшой вероятностью (обычно обратно пропорциональной длине хромосомы).

Это обеспечивает генетическое разнообразие и позволяет алгоритму выходить из локальных минимумов.

*6.* ***Клонирование:***

Перед изменением отобранных особей выполняется их глубокое копирование (clone), чтобы избежать побочных эффектов при скрещивании или мутации:

*7.* ***Цикл эволюции:***

Главный цикл работает до тех пор, пока не найдено оптимальное решение (все единицы) или не достигнуто максимальное число поколений. На каждой итерации:

* Отбираются родители
* Проводится скрещивание и мутация
* Пересчитываются значения приспособленности
* Обновляется популяция
* Сохраняется статистика (макс. и среднее значение фитнеса в поколении)

***Результаты:***

*Поколение* 50: *Макс приспособ.* = 100, *Средняя приспособ*. = 98.525

*Лучший индивидуум* = 1 1 1 1 1 1 1 …

Задача *OneMax* наглядно показывает, как работает генетический алгоритм. Несмотря на простую фитнес-функцию, алгоритм демонстрирует *основные принципы эволюционного подхода*:

* *отбор лучших решений*
* *сочетание хороших признаков через скрещивание*
* *внесение случайности через мутацию*

*ГА* нашел оптимальное решение за ограниченное число поколений, что подтверждает его эффективность даже при случайной инициализации.

