Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Российский химико-технологический университет имени Д.И. Менделеева»
Кафедра информационных компьютерных технологий

ОТЧЕТ ПО КУРСОВОЙ РАБОТЕ

Выполнил студент группы .....КС-30 Тимофеев Владислав Дмитриевича

Приняли: .....Кращенников Роман

Дата сдачи: 18.05.2023

Оглавление

Описание задачи.

Найти минимум функции Egg Holder, используя метод стаи серых волков

Теоретическая часть.

Описание метода стаи серых волков

Оптимизатор серых волков (Grey Wolf Optimizer, GWO) - это метаэволюционный алгоритм, вдохновленный социальным поведением серых волков в природе. Он был предложен в 2014 году и применяется для решения задач оптимизации.

Алгоритм GWO имитирует иерархию и взаимодействие волков в стае, чтобы находить оптимальное решение задачи. Волки в стае делятся на три категории: альфа-волк (наилучшее решение), бета-волк и омега-волк (младшие волки). На каждой итерации алгоритма, волки обновляют свои положения в пространстве поиска, чтобы приближаться к оптимальному решению.

Процесс оптимизации в GWO включает в себя следующие шаги:

1. Инициализация положений волков случайным образом в пространстве поиска.
2. $$\bar{X} = random(lb, ub)$$
3. Оценка функции приспособленности (целевой функции) для каждого волка.

$$fitness = egg\_holder(\bar{X})$$

4. Определение альфа-волка (лучшего решения) и бета-волка на основе их значений функции приспособленности.
5. Обновление положений волков, используя формулы, основанные на их иерархии в стае.

$$\bar{D}_a = |\bar{C}_1 * \bar{X}_a - \bar{X}|$$
$$\bar{D}_\beta = |\bar{C}_2 * \bar{X}_\beta - \bar{X}|$$
$$\bar{D}_\delta = |\bar{C}_3 * \bar{X}_\delta - \bar{X}|$$

$$\bar{X}_1 = \bar{X}_a - \bar{A}_1 * (\overline{D_a}$$
$$\bar{X}_1 = \bar{X}_a - \bar{A}_1 * (\overline{D_a})$$
$$\bar{X}_2 = \bar{X}_\beta - \bar{A}_2 * (\overline{D_\beta})$$
$$\bar{X}_3 = \bar{X}_\delta - \bar{A}_3 * (\overline{D_\delta})$$
$$\bar{X}(t+1) = \frac{\bar{X}_1 + \bar{X}_2 + \bar{X}_3}{3}$$
$$\bar{A} = 2\bar{a} * \bar{r}_1 - \bar{a}$$

$$\bar{C} = 2 * \bar{r}_2$$
$$a = 2 * (1 - \frac{t}{T})$$

6. Повторение шагов 2-4 до достижения заданного критерия остановки (например, достаточной точности решения или максимального числа итераций).

Преимущества GWO включают его простоту реализации, эффективность в решении различных задач

оптимизации и возможность работы с непрерывными и дискретными пространствами параметров.
Однако, как и другие метаэволюционные алгоритмы, GWO не гарантирует нахождение глобального оптимума и может требовать настройки параметров для достижения оптимальной производительности в конкретной задаче.

Описание функции Egg Holder

Функция "Egg Holder" (Рис. 1) (иногда называемая "Egg Crate") является одной из множества тестовых функций, которые используются для оценки производительности алгоритмов оптимизации. Функция "Egg Holder" определяется следующим образом:

$$f(\mathbf{x}) = -(x_2 + 47)\sin\left(\sqrt{\left|x_2 + \frac{x_1}{2} + 47\right|}\right) - x_1 \sin\left(\sqrt{|x_1 - (x_2 + 47)|}\right)$$

$$x_i \in [-512, 512]$$

Функция "Egg Holder" представляет собой поверхность с несколькими яйцами, расположенными в углублениях. Цель состоит в том, чтобы найти минимальное значение этой функции, что означает нахождение оптимальной позиции яиц.

Одна из причин, по которой функция "Egg Holder" может быть не самым подходящим выбором для тестирования метода GWO, связана с ее особенностями. Эта функция имеет множество локальных минимумов и сильные неоднородности в ландшафте поиска, что может затруднить работу алгоритмов оптимизации.

Также, большая исследуемая поверхность может представлять препятствие для метода оптимизации, включая метод серых волков (GWO), если ресурсы (время и вычислительная мощность) ограничены.
Функции с большой поверхностью могут потребовать больше итераций и ресурсов для обнаружения глобального оптимума. Это может привести к более длительному времени выполнения алгоритма и увеличению вероятности застревания в локальных оптимумах. Кроме того, большая поверхность может усложнить адаптацию параметров алгоритма для достижения хороших результатов.

Метод GWO является метаэволюционным алгоритмом, который ищет глобальный оптимум в пространстве поиска. Однако, функция "Egg Holder" представляет собой задачу с множеством локальных минимумов, и вероятность застревания в одном из них велика. Это может привести к тому, что GWO будет сходиться к локальным оптимумам, не достигая глобального оптимума. При использовании такой функции для тестирования GWO может быть сложно оценить его способность находить глобальные оптимумы.

Вместо этого, для тестирования метода GWO рекомендуется выбирать функции, которые имеют хорошую сбалансированность между локальными и глобальными оптимумами, а также меньшее количество локальных экстремумов. Такие функции могут представлять собой сглаженные и выпуклые поверхности, где вероятность достижения глобального оптимума выше.

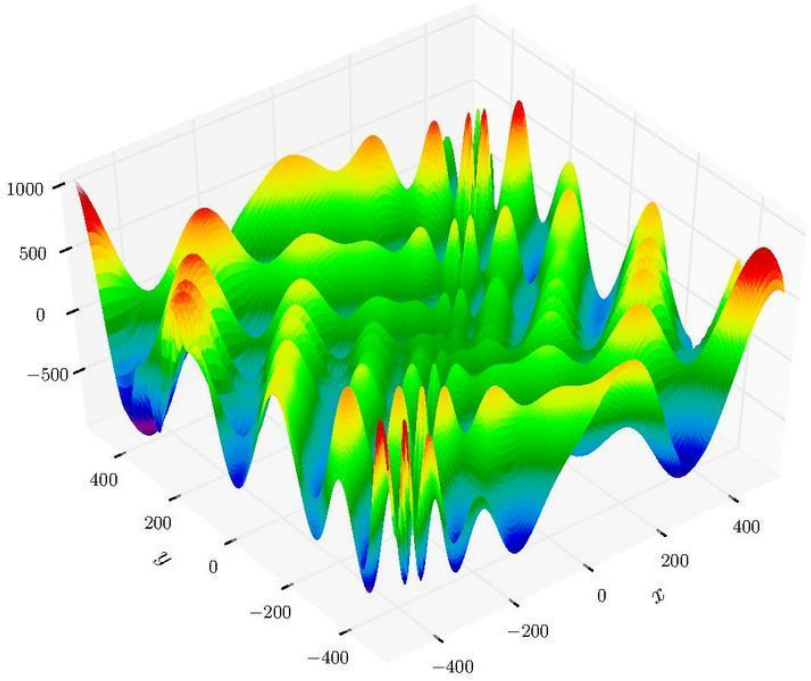


Рис. 1 График функции Egg Holder

Выполнение задачи.

Метод был реализован на Python.

```
[1]
import numpy as np
import matplotlib.pyplot as plt

[2]
# Глобальная переменная для подсчета вызовов функции egg_holder
egg_holder_calls = 0

# The function has multiple local minima and one global minimum at f(512, 404.2319) =
-959.6407 for n = 2 dimension.
def egg_holder(x):
    global egg_holder_calls
    egg_holder_calls += 1
    R = 0
    for i in range(x.shape[0]-1):
        R -= x[i]*np.sin(np.sqrt(abs(x[i]-x[i+1]-47))) +
(x[i+1]+47)*np.sin(np.sqrt(abs(x[i+1]+x[i]/2+47)))
    return R

[3]
# Оригинальный алгоритм

def GWO(obj_func, lb, ub, dim, search_agent_no, max_iter):
    global egg_holder_calls
    egg_holder_calls = 0

    alpha_pos = np.zeros(dim)
```

```
    beta_pos = np.zeros(dim)
    delta_pos = np.zeros(dim)
    alpha_score = float("inf")
    beta_score = float("inf")
    delta_score = float("inf")
    pos = np.zeros((search_agent_no, dim))
    for i in range(search_agent_no):
        pos[i, :] = np.random.uniform(lb, ub, dim)
    for l in range(max_iter):
        for i in range(search_agent_no):
            fitness = obj_func(pos[i, :])
            if fitness < alpha_score:
                delta_score = beta_score
                delta_pos = beta_pos.copy()
                beta_score = alpha_score
                beta_pos = alpha_pos.copy()
                alpha_score = fitness
                alpha_pos = pos[i, :].copy()
            if fitness > alpha_score and fitness < beta_score:
                delta_score = beta_score
                delta_pos = beta_pos.copy()
                beta_score = fitness
                beta_pos = pos[i, :].copy()
        a = 2 * (1 - l / max_iter)
        #a = 2 * (1 - l / (max_iter - 1))
        for i in range(search_agent_no):
            r1 = np.random.rand(dim)  # random weight factor
            r2 = np.random.rand(dim)  # random weight factor
            A1 = 2 * a * r1 - a
            C1 = 2 * r2
            D_alpha = abs(C1 * alpha_pos - pos[i, :])
            X1 = alpha_pos - A1 * D_alpha
            r1 = np.random.rand(dim)  # random weight factor
            r2 = np.random.rand(dim)  # random weight factor
            A2 = 2 * a * r1 - a
            C2 = 2 * r2
            D_beta = abs(C2 * beta_pos - pos[i, :])
            X2 = beta_pos - A2 * D_beta
            r1 = np.random.rand(dim)
            r2 = np.random.rand(dim)
            A3 = 2 * a * r1 - a
            C3 = 2 * r2
            D_delta = abs(C3 * delta_pos - pos[i, :])
            X3 = delta_pos - A3 * D_delta
            pos[i, :] = (X1 + X2 + X3) / 3
            # check if position is within bounds
            pos[i, :] = np.clip(pos[i, :], lb, ub)
        # record the best fitness value for each iteration
        best_fitness = alpha_score
        # Check if the best fitness value is below a certain threshold
        if best_fitness < -3719.7248363 * 0.95: #даем право на погрешность в 5 %
            break
        #print("Iteration: {}  x: {}  Fitness: {:.4f}".format(l+1, alpha_pos,
best_fitness))
    return alpha_pos, alpha_score, l + 1, egg_holder_calls
```

В данном случае показана реализация классического метода. Также доступны модификации в виде:

1. Убрать delta вожака, оставив только alpha и beta
2. На каждом шаге выбирать случайное значение для позиции delta вожака
3. Изменить коэффициент сходимости a на a = 2 * (1 - l**2 / max_iter**2), используя оригинальный метод
4. Изменить коэффициент сходимости a на a = 2 * (1 - l**2 / max_iter**2), используя первую модификацию
5. Изменить коэффициент сходимости a на a = 2 * (1 - l**2 / max_iter**2), используя вторую модификацию

Тестирование

Для начала, проводим тестирование для второго измерения, а именно:

1. Как влияет количество волков на сходимость методов
2. Количество итераций до сходимости
3. Количество вызовов целевой функции

Для второго измерения мы берем максимальное количество итераций 100, если метод не сходится, мы не считаем итерации в нем, вызовы целевой функции считаем в любом случае.

Делаем 100 тестов для каждого набора параметров.

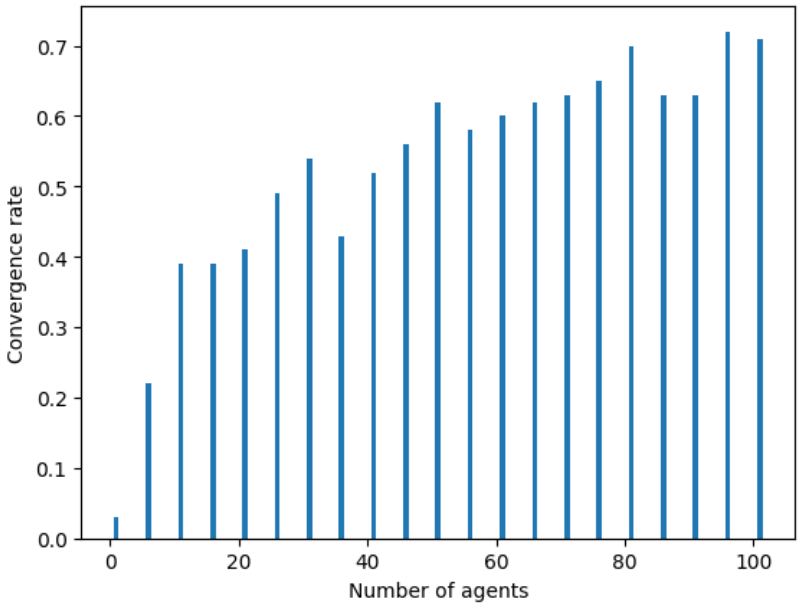
Функция для тестирования:

```
def run_GWO_search_agent(search_agent_no):
    dim = 5
    max_iter = 600
    lb = np.full(dim, -512)
    ub = np.full(dim, 512)

    best_pos, best_score, iterations, egg_holder_calls = GWO(egg_holder, lb, ub, dim,
search_agent_no, max_iter)
    error = abs(best_score + 3719.7248363)
    converged = error <= 3719.7248363 * 0.05

    return converged, iterations, egg_holder_calls

search_agent_nos = range(500, 2001, 50)
num_trials = 100
convergence_rates = []
average_iterations = []
average_egg_holder_calls = []
for search_agent_no in search_agent_nos:
    converged_count = 0
    iterations_count = 0
    egg_holder_call_count = 0
    for i in range(num_trials):
        converged, iterations, egg_holder_calls =
run_GWO_search_agent(search_agent_no)
```

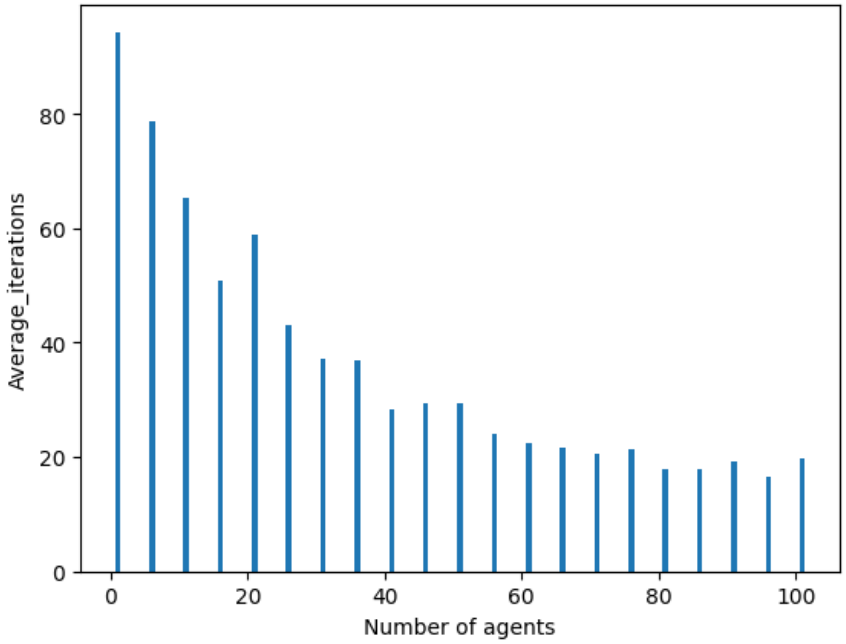
```
        if converged:
            converged_count += 1
            iterations_count += iterations
        egg_holder_call_count += egg_holder_calls

    convergence_rate = converged_count / num_trials
    average_iteration = iterations_count / (converged_count + 0.000001)
    average_egg_holder_call = egg_holder_call_count / num_trials
    convergence_rates.append(convergence_rate)
    average_iterations.append(average_iteration)
    average_egg_holder_calls.append(average_egg_holder_call)
    print(f"search_agent_no: {search_agent_no}, convergence rate: {convergence_rate},
average iterations: {average_iteration}, average egg holder calls:
{average_egg_holder_call}")

plt.bar(search_agent_nos, convergence_rates)
plt.xlabel("Number of agents")
plt.ylabel("Convergence rate")
plt.show()

plt.bar(search_agent_nos, average_iterations)
plt.xlabel("Number of agents")
plt.ylabel("Average_iterations")
plt.show()

plt.bar(search_agent_nos, average_egg_holder_calls)
plt.xlabel("Number of agents")
plt.ylabel("Average_egg_holder_calls")
plt.show()
```

Оригинальный метод

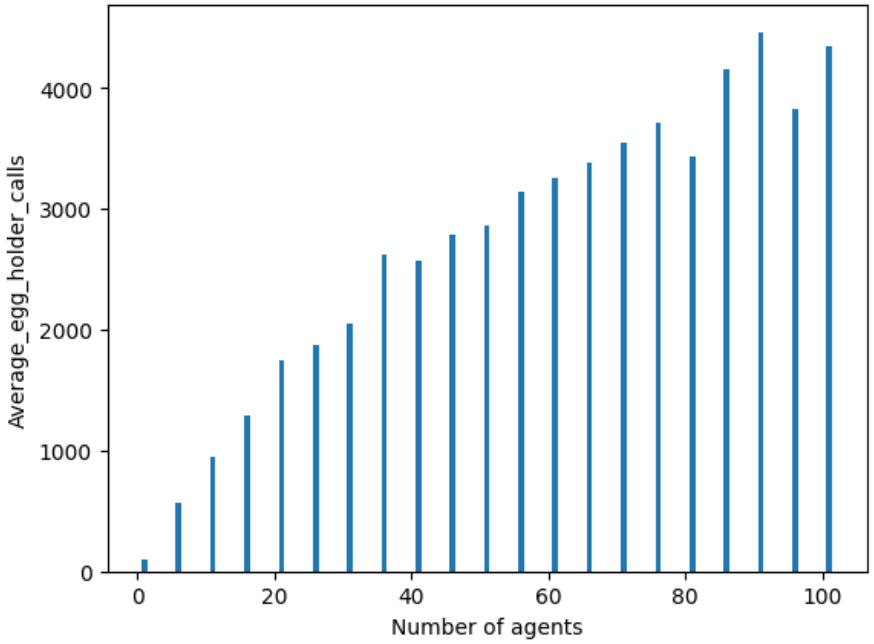
Тестирование сходимости показало, что при достаточно большом количестве агентов, сходимость достигается в районе 70%



При этом количество итераций для сходимости уменьшается от возрастания популяции, и становится около 24 при большом количестве агентов



В свою очередь, количество вызовов целей функции растет от увеличения количества агентов



Можно заметить, что количество вызовов функции равно количеству итераций помноженных на количество агентов, следовательно, чем быстрее метод сойдется, тем меньше будет вызовов

Сравнение оригинального метода и модификаций

Ввиду большого количества модификаций, приведу таблицу сравнения, при оптимальных выявленных параметрах

	Оригинальный	Мод 1	Мод 2	Мод 3	Мод 4	Мод 5
Сходимость	70%	79%	89%	70%	82%	97%
Итерации	24	19	38	20	20	42
Вызовы	4100	3400	4100	4200	3200	4400

Как видно из таблицы, модификация со случайным delta вожаком и измененным коэффициентом a, имеет наивысший процент сходимости, а также требует больше всего ресурсов. Лучше оригинала справляется метод с двумя вожаками, при этом, требуя меньше всего ресурсов.

Тестирование для 5 измерения

Ввиду того что функция требует очень больших расчетов, получилось протестировать, только для 5 измерения и при определенных параметрах (не оптимальных), таблицы результатов представлена ниже. Количество тестов пришлось снизить до 20, что понизит точность статистики.

	Оригинальный	Мод 1	Мод 2	Мод 3	Мод 4	Мод 5
Кол-во волков	2000	2000	2000	2000	2000	2000
Сходимость	20%	15%	0%		15%	
Итерации	152	300	0		252	
Вызовы	541100	600000	600000		585700	

Заключение.

В заключение, использование метода серых волков (GWO) и его модификаций в целом может быть эффективным для решения задач оптимизации. GWO основан на социальном поведении волков в стае и обладает простой реализацией и небольшим количеством настраиваемых параметров. Он может успешно справляться с различными типами задач оптимизации и иметь хорошую производительность в нахождении глобальных оптимумов в некоторых случаях.

Однако, при использовании GWO на функции "Egg Holder" возникают некоторые проблемы. Функция "Egg Holder" характеризуется множеством локальных минимумов и неоднородным ландшафтом поиска, а также большой площадью исследования, что может затруднять работу алгоритма оптимизации. В связи с этим, GWO может столкнуться с трудностями при нахождении глобального оптимума на этой функции и вероятность застревания в локальных минимумах значительно возрастает.

Рекомендуется использовать функции, которые представляют сбалансированное соотношение между глобальными и локальными оптимумами, а также имеют более гладкую и выпуклую поверхность для тестирования GWO и его модификаций. Такие функции позволяют алгоритму более эффективно искать глобальные оптимумы и предоставляют более надежные результаты.

В конечном счете, выбор функции для тестирования GWO должен быть основан на требованиях и характеристиках конкретной задачи оптимизации. Важно учитывать сложность поверхности, наличие локальных и глобальных оптимумов, а также ограничения алгоритма и ресурсы, доступные для вычислений.

Так, тесты показали, что тяжело добиться большого процента сходимости для 5 измерения, не затрачивая огромные ресурсы, также тестирование показали, что модификации, которые лучше справлялись при 2 измерении, не справляются с 5 измерением.