ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФГАОУ ВО НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа «Прикладная математика и информатика»

УДК XXXXX

Отчет об исследовательском проекте на тему:

Глобальные градиентные методы оптимизации

Выполнил:

Студент группы БПМИ214
Судаков Илья Александрович _____ _____
(подпись) (дата)

Принял руководитель проекта:

Посыпкин Михаил Анатольевич
Научный сотрудник
Факультета компьютерных наук НИУ ВШЭ

_____ _____
(подпись) (дата)

Москва 2023

Содержание

Аннотация

Многие методы оптимизации (координатный спуск, градиентный спуск, метод сопряженного гадиента и др.) используют так называемый локальный поиск, т.е. поиск минимума одномерной функции вдоль луча в n-мерном евклидовом пространстве. При этом всегда находится локальный минимум. В проекте предлагается исследовать потенциал применения методов глобальной одномерной оптимизации для организации линейного поиска. В каких случаях удастся найти глобальной минимум многомерной функции, используя глобальный линейный поиск в контексте классических градиентных методов.

Задачи проекта

- Программная реализация методов линейного поиска.
  - Метод Золотого сечения
  - Алгоритм Moore-Skelboe
- Реализация алгоритма оптимизации многомерной функции.
  - Координатный спуск
  - Градиентный спуск
- Экспериментальное и теоретическое исследование.

Цели проекта

Написать алгоритм, который будет находить глобальный минимум многомерной непрерывной функции:

- с хорошей скоростью
- с заданной точностью
- не хуже классических методов для унимодальных функций

Ссылка на материалы

Ссылка на репозиторий, с разработанными материалами:
https://github.com/sudakovcom/global-optimization.git

Ключевые слова

Глобальная оптимизация, градиентный спуск, интервальный анализ

1 Метод золотого сечения

Начнем с рассмотрения способов одномерной оптимизации, первый из них классический, второй более сложный, использующий интервальный анализ. Результаты этих двух методов мы и будем сравнивать, тестируя их на функции Растригина и функции Экли.

Метод золотого сечения — это эффективный и простой способ нахождения локального минимума функции на заданном отрезке, за счет того, что на очередной итерации требует вычисления значения функции только в одной точке, что может быть очень важно, для сложновычислимых функций.

Замечание. Метод золотого сечения находит глобальный минимум только в случае унимодальных функций.

Определение. [5] Функция **унимодальная** на отрезке $[a, b]$, если $\exists \alpha, \beta : a \leq \alpha \leq \beta \leq b$ такие, что:

- на отрезке $[a, \alpha]$ функция монотонно убывает
- на отрезке $[\beta, b]$ функция монотонно возрастает
- $\forall x \in [\alpha, \beta] : f(x) = \min\limits_{x \in [a,b]} f(x)$

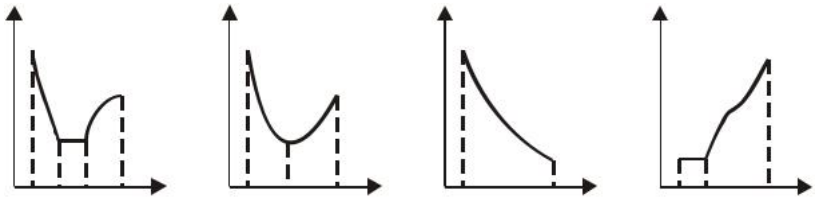


Рис. 1.1: Примеры унимодальных функций

Свойство. Любой локальный минимум будет являться одновременно и глобальным минимумом.

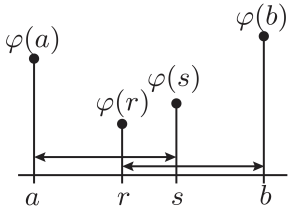


Рис. 1.2: Итерация

Алгоритм. На рисунке 1.2 изображена одна итерация метода. Пусть хотим найти минимум функции φ на отрезке $[a, b]$, разобъем его на три части двумя точками r, s так, чтобы при отсекании одного из крайних подотрезков, одна из точек стала границей подотрезка для следующей итерации, то есть соотношение длин подотрезков оставалось прежним:

$$|[a, r]| = |[s, b]| = l_1$$

$|[r,s]| = l_2$

тогда, если $\varphi(r) \leq \varphi(s)$, то s станет правой границей рассматриваемого отрезка, а r станет правой внутренней точкой:

$$\frac{|[a,s]|}{|[a,b]|} = \frac{|[a,r]|}{|[a,s]|} \Rightarrow \frac{l_1 + l_2}{2l_1 + l_2} = \frac{l_1}{l_1 + l_2} \Rightarrow l_1^2 - l_1 \cdot l_2 - l_2^2 = 0 \Rightarrow \frac{l_1}{l_2} = \frac{1+\sqrt{5}}{2} = \Phi \approx 1.618$$

Замечание. Число Φ называют **золотым сечением**

Несмотря на то, что данный метод гарантированно находит глобальный минимум только для унимодальных функций, мы будем его использовать для сравнения с более надежными методами.

Сложность.

На каждой итерации длина рассматриваемого отрезка умножается на

$$\frac{l_1 + l_2}{l_1 + l_2 + l_2} = \frac{3+\sqrt{5}}{4+2\sqrt{5}} = \Phi^{-1} = \phi$$

Для достижения точности δ потребуется

$$\frac{\ln(\frac{|[a,b]|}{\delta})}{\ln(\Phi)} \approx 2\ln(\frac{|[a,b]|}{\delta})$$

итераций.

Замечание. Так как применяя этот метод к не унимодальной функции, он не гарантирует нахождение глобального минимума, то будем переходить из текущей точки в найденную, только если значение в найденной меньше текущего.

2 Глобальная одномерная оптимизация

Алгоритм поиска глобального минимума функции вдоль одного направления будет основываться на интервальном анализе [2]. Поэтому для начала приведем основные определения и теоремы, которые нам пригодятся для посторения алгоритма.

Определение. Интервалом $[a, b]$ называется следующее множество:

$$[a, b] := \{x \in \mathbb{R} | a \leq x \leq b\}$$

в других обозначениях:

$$\mathbf{x} := [\underline{\mathbf{x}}, \overline{\mathbf{x}}]$$

где $\underline{\mathbf{x}}, \overline{\mathbf{x}}$ - левая и правая граница интервала соответственно.

Арифметические свойства интервалов.

- $\mathbf{x} + \mathbf{y} = [\underline{\mathbf{x}} + \underline{\mathbf{y}}, \overline{\mathbf{x}} + \overline{\mathbf{y}}]$
- $\mathbf{x} - \mathbf{y} = [\underline{\mathbf{x}} - \overline{\mathbf{y}}, \overline{\mathbf{x}} - \underline{\mathbf{y}}]$
- $\mathbf{x} \cdot \mathbf{y} = [min(\underline{\mathbf{x}}\underline{\mathbf{y}}, \underline{\mathbf{x}}\overline{\mathbf{y}}, \overline{\mathbf{x}}\underline{\mathbf{y}}, \overline{\mathbf{x}}\overline{\mathbf{y}}), max(\underline{\mathbf{x}}\underline{\mathbf{y}}, \underline{\mathbf{x}}\overline{\mathbf{y}}, \overline{\mathbf{x}}\underline{\mathbf{y}}, \overline{\mathbf{x}}\overline{\mathbf{y}})]$
- $\mathbf{x}/\mathbf{y} = \mathbf{x} \cdot [1/\overline{\mathbf{y}}, 1/\underline{\mathbf{y}}]$, если $\underline{\mathbf{y}} > 0$

Пример. Пусть известно, что первый гонщик проезжает длину гонки за время от 80 до 100 минут, а второй от 85 до 90 минут, какая может быть разница во времени приезда к финишу? Воспользуемся вторым арифметическим свойством:

$\mathbf{T_1} = [80, 100]$

$\mathbf{T_2} = [85, 90]$, тогда результатом будет интервал $\mathbf{T_1} - \mathbf{T_2} = [80 - 90, 100 - 85] = [-10, 15]$

То есть первый гонщик может как приехать на 10 минут раньше, так и задержаться на 15 минут.

Замечание. Так как интервалы являются множествами, то для них можно определить частичный порядок относительно включения:

$$\mathbf{a} \subseteq \mathbf{b} \iff \underline{\mathbf{b}} \leq \underline{\mathbf{a}} \cap \overline{\mathbf{a}} \leq \underline{\mathbf{b}}$$

Свойство. Интервалы обладают свойством монотонности относительно арифметических операций: пусть $\mathbf{a} \subseteq \mathbf{a}', \mathbf{b} \subseteq \mathbf{b}', \star \subseteq \{+, -, \cdot, /\}$, тогда $\mathbf{a} \star \mathbf{b} \subseteq \mathbf{a}' \star \mathbf{b}'$

Теорема. Основная теорема интервальной арифметики

Пусть $f(x_1, ..., x_n)$ - функция вещественных аргументов $x_1, ..., x_n$, и для нее определен результат $\mathbf{F}(\mathbf{X_1}, ..., \mathbf{X_n})$ подстановки вместо аргументов интервалов которые они пробегают $(\mathbf{X_1}, ..., \mathbf{X_n}) \subset \mathbb{IR}^n$ и для $(\mathbf{X_1}, ..., \mathbf{X_n})$ операциии выполняются по правилам интервальной арифметики. Тогда выполнено следущее:

$$\{f(x_1, ..., x_n) | x_1 \in \mathbf{X_1}, ..., x_n \in \mathbf{X_n}\} \subseteq \mathbf{F}(\mathbf{X_1}, ..., \mathbf{X_n})$$

Предложение. Пусть $f(x_1, ..., x_n)$ - функция вещественных аргументов $x_1, ..., x_n$, и $\mathbf{F}(\mathbf{X_1}, ..., \mathbf{X_n})$ соответствующая ей интервальная функция, тогда выполняется монотонность по включению:

Пусть $\mathbf{X_1}, ..., \mathbf{X_n}$ и $\mathbf{Y_1}, ..., \mathbf{Y_n}$, такие что $\mathbf{X_1} \subseteq \mathbf{Y_1}, ..., \mathbf{X_n} \subseteq \mathbf{Y_n}$, тогда:

$$\mathbf{F}(\mathbf{X_1}, ..., \mathbf{X_n}) \subseteq \mathbf{F}(\mathbf{Y_1}, ..., \mathbf{Y_n})$$

Определение. Пусть $N > 0$ натуральное число, тогда если $\langle S_0, ..., S_{n-1} \rangle$ семейство непустых подмножеств множества S, тогда будем называть его покрытием внутри S. В частности, если обьединение $S_0, ..., S_{n-1}$ равно S, тогда последовательность $\langle S_0, ..., S_{n-1} \rangle$ является покрытием S.

Теорема. Рассмотрим задачу глобальной оптимизации. Пусть $\langle B_0, ..., B_{n-1} \rangle$ семейство множеств, содержащее глобальный минимум, такое что упорядочено по возрастанию нижней границы $\mathbf{F}(B_i)$ для $i = 0, 1, 2, ..., N-1$. Пусть U наименьшее из верхних значений функции для подмножеств $\langle \mathbf{F}(B_0), ..., \mathbf{F}(B_{n-1}) \rangle$. Тогда интервал $[lb(\mathbf{F}(B_0)), U]$ содержит глобальным минимум μ.

Используя данную теорию, напишем алгоритм поиска глобального минимума вдоль направления.

3 Алгоритм Moore-Skelboe

Ниже приведена реализация алгоритма на языке python[4], использующая библиотеки pyinterval[3] и numpy[1].

```
def MooreSkelboe(func_index, index, a, b, p, e_d,
                 e_f):  # func_index(number of function in list of functions), index(index of direction),
    # a(left border), b(right border), p(current point), e_d(error of d), e_f(error of f)
    F = Functions[func_index * 2 + 1]
    interval_d = []
    for i in range(len(p)):
        interval_d.append(interval[p[i], p[i]])
    interval_d[index] = interval[a, b]

    interval_f = F(interval_d)
    set_of_intervals = [[interval_d, interval_f]]
    U = right(interval_f)
    w_f = right(interval_f) - left(interval_f)
    w_d = right(interval_d[index]) - left(interval_d[index])
    best_interval = set_of_intervals[0]
    while (w_d > e_d) | (w_f > e_f):
        set_of_intervals.pop(0)
        mid_p = mid(best_interval[0][index])
        interval_1 = best_interval[0].copy()
        interval_2 = best_interval[0].copy()
        interval_1[index] = interval[left(best_interval[0][index]), mid_p]
        interval_1_f = F(interval_1)
        interval_2[index] = interval[mid_p, right(best_interval[0][index])]
        interval_2_f = F(interval_2)
        U = min(U, right(interval_1_f))
        U = min(U, right(interval_2_f))


        if (len(set_of_intervals)>0) and (U<left(set_of_intervals[-1][1])):
            l = 0
            r = len(set_of_intervals) - 1
            while l < r:
                m = int((l + r) / 2)
                if left(set_of_intervals[m][1]) > U:
                    r = m
                else:
                    l = m + 1
            set_of_intervals = set_of_intervals[:l]

        set_of_intervals.append([interval_1, interval_1_f])
        set_of_intervals.append([interval_2, interval_2_f])
        set_of_intervals.sort(key=lambda item: left(item[1]))


        best_interval = set_of_intervals[0]
        w_f = right(best_interval[1]) - left(best_interval[1])
        w_d = right(best_interval[0][index]) - left(best_interval[0][index])

    best_point = []
    for i in range(len(p)):
        best_point.append(mid(best_interval[0][i]))

    return best_point
```

Замечание. Данный алгоритм применим не только для одномерных функций, но и для функций с произвольным числом измерений, но его сложность будет расти экспоненциально от размерности. Измерения по данному способу оптимизации в таблицах 6.5, 6.6.

4 Координатный спуск

Рассмотрим первый, один из наиболее простых методов многометрной оптимизации - **координатный спуск**. На каждой итерации происходит оптимизация по одной переменной, то есть происходит поиск минимума вдоль заданного направления. На рисунке 4.1 изображена одна итерация метода. Существуют различные порядки перебирания направлений поиска, в простейшем варианте они перебираются по порядку.

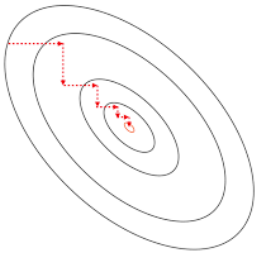


Рис. 4.1: Наглядное представление метода

Замечание. Алгоритм не требует вычисления производной функции в точке.

Алгоритм. [6] Пусть целевая функция имеет вид: $f(x) : \mathbb{X} \to \mathbb{R}$, где $\mathbb{X} \subset \mathbb{R}^n$

Задача оптимизации задана в следующем виде: найти $x_{opt} : f(x_{opt}) = \min\limits_{x \in \mathbb{X}} f(x) = \mu$

- Задать начальное приближение и точность расчёта: X_0, ε
- Рассчитать $x_i^{k+1} = \arg\min\limits_{y \in \mathbb{X}_i} f(x_1^{k+1}, \ldots, x_{i-1}^{k+1}, y, x_{i+1}^k, \ldots, x_n^k)$
- Проверить условие остановки (на усмотрение):
  - $|x^j - x^{j+1}| < \varepsilon$
  - $|F(x^j) - F(x^{j+1})| < \varepsilon$
  - иначе перейти к пункту 2

Таким образом, на каждой итерации будет выполнено $F(x^0) \geqslant F(x^1) \geqslant F(x^2) \geqslant \ldots$.

Замечание. Чтобы найти $x_i^{k+1} = \arg\min\limits_{y \in \mathbb{X}_i} f(x_1^{k+1}, \ldots, x_{i-1}^{k+1}, y, x_{i+1}^k, \ldots, x_n^k)$ будем использовать метод золотого сечения и алгоритм глобальной одномерной оптимизации.

5 Градиентный спуск

Другой, не менее изветсный метод многомерной оптимизации - **градиентный спуск**. На каждой итерации вычисляется градиент функции в текущей точке, вдоль которого происходит поиск минимума вдоль направления. На рисунке 5.1 изображена одна итерация метода.

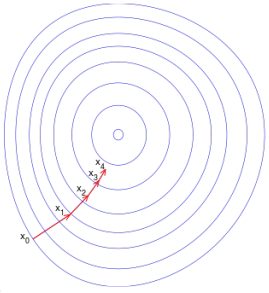


Рис. 5.1: Наглядное представление метода

Замечание. Алгоритм требует вычисления градиента функции в точке, поэтому необходимо, чтобы функция была дифференцируемой.

Алгоритм. [6] Пусть целевая функция имеет вид: $f(x) : \mathbb{X} \to \mathbb{R}$, где $\mathbb{X} \subset \mathbb{R}^n$

Задача оптимизации задана в следующем виде: найти $x_{opt} : f(x_{opt}) = \min\limits_{x \in \mathbb{X}} f(x) = \mu$

- Задать начальное приближение и точность расчёта: X_0, ε
- Рассчитать $x^{j+1} = x^j - \lambda^j \nabla F\left(x^j\right)$
- Проверить условие остановки (на усмотрение):
  - $|x^j - x^{j+1}| < \varepsilon$
  - $|F(x^j) - F(x^{j+1})| < \varepsilon$
  - $\|\nabla F(x^i)\| < \varepsilon$
  - иначе перейти к пункту 2

Замечание. $x^{j+1} = x^j - \lambda^j \nabla F\left(x^j\right)$, где λ^j задает скорость градиентного спуска и может быть выбрана как:

- постоянная, тогда метод может не сходиться
- убывать по какому-то закону
- гарантировать наискорейший спуск

Модификация: метод наискорейшего спуска

В случае наискорейшего спуска λ^j определяется как:

$$\lambda^j = \underset{\lambda}{\operatorname{argmin}} F\left(x^{j+1}\right) = \underset{\lambda}{\operatorname{argmin}} F\left(x^j - \lambda \nabla F\left(x^j\right)\right)$$

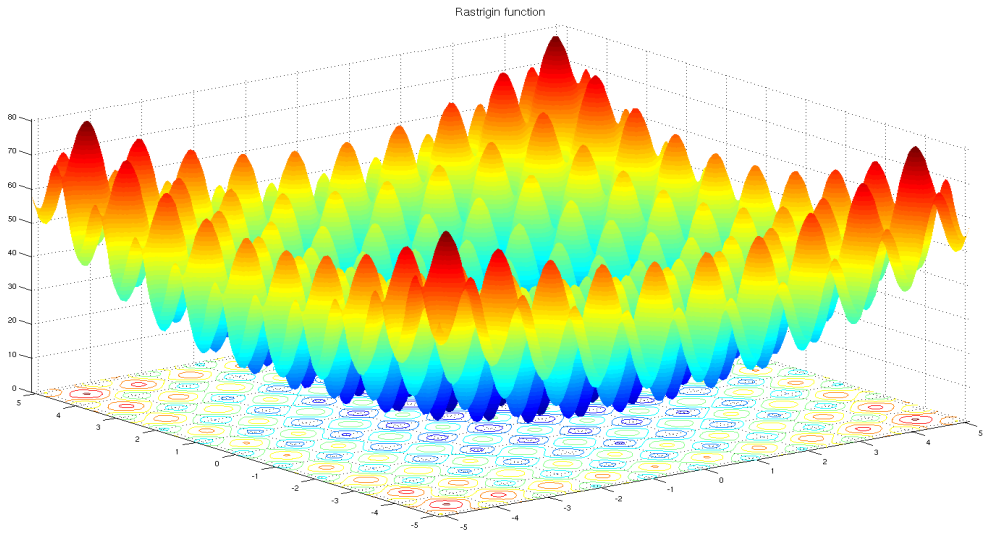
Замечание. Чтобы вычислить λ^j, будем использовать метод золотого сечения и алгоритм Moore-Skelboe.

6 Тестирование

Ниже приведены результаты тестирования двух методов оптимизации на тестовых функциях [7], первый использует метод золотого сечения, для поиска минимума вдоль направления, а другой алгоритм Moore-Skelboe.

Функция Растригина

График функции при $n = 2$



Аналитическое выражение

$$f(x) = 10n + \sum_{i=1}^{n} \left(x_i^2 - 10 \cdot \cos\left(2\pi x_i\right)\right)$$

Глобальный минимум

$$f(0, ..., 0) = 0$$

Таблица 6.1: Результаты тестирования координатного спуска на функции Растригина.

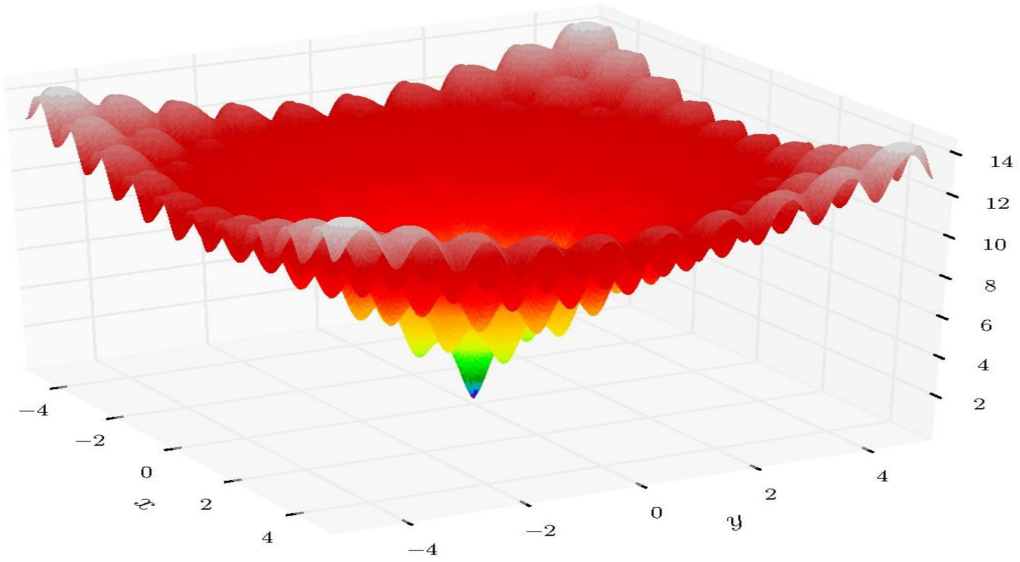
	Golden section			Moore – Skelboe		
n	Prec	Time	Minimum	Prec	Time	Minimum
2	10^-3	0.2s	1.9899	10^-3	0.1s	0.0001
5	10^-3	0.2s	4.9748	10^-3	0.2s	0.0001
10	10^-3	0.3s	9.9496	10^-3	1.2s	0.0002
20	10^-3	0.5s	19.8992	10^-3	3.9s	0.0004
30	10^-3	0.7s	29.8489	10^-3	8.5s	0.0006
40	10^-3	0.8s	39.7985	10^-3	15.2s	0.0007
50	10^-3	0.2s	49.7482	10^-3	23.5s	0.0009
60	10^-3	0.2s	59.6978	10^-3	33.7s	0.0011
70	10^-3	0.2s	69.6474	10^-3	45.9s	0.0013
80	10^-3	0.2s	79.5971	10^-3	35.6s	0.0005
90	10^-3	0.5s	89.5467	10^-3	45.1s	0.0006
100	10^-3	0.3s	99.4964	10^-3	55.6s	0.0006

Таблица 6.2: Результаты тестирования градиентного спуска на функции Растригина.

	Golden section			Moore – Skelboe		
n	Prec	Time	Minimum	Prec	Time	Minimum
2	10^-3	0.4s	1.9899	10^-3	0.2s	0.0001
5	10^-3	0.5s	4.9748	10^-3	0.1s	0.0001
10	10^-3	0.8s	9.9496	10^-3	0.1s	0.0002
20	10^-3	0.7s	19.8991	10^-3	0.3s	0.0001
30	10^-3	0.3s	29.8487	10^-3	0.4s	0.0001
40	10^-3	0.2s	39.7983	10^-3	0.6s	0.0002
50	10^-3	0.8s	49.7479	10^-3	0.7s	0.0002
60	10^-3	0.7s	59.6975	10^-3	0.8s	0.0001
70	10^-3	0.7s	69.6471	10^-3	0.9s	0.0001
80	10^-3	0.6s	79.5967	10^-3	1.1s	0.0001
90	10^-3	0.8s	89.5463	10^-3	1.2s	0.0001
100	10^-3	0.8s	99.4959	10^-3	1.4s	0.0001

Функция Экли

График функции при $n = 2$



Аналитическое выражение

$$f(x) = 20 + e - 20e^{-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} - e^{\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)}$$

Глобальный минимум

$$f(0, ..., 0) = 0$$

Таблица 6.3: Результаты тестирования координатного спуска на функции Экли.

	Golden section			Moore – Skelboe		
n	Prec	Time	Minimum	Prec	Time	Minimum
2	10^-3	0.5s	0.0001	10^-3	0.4s	0.0006
5	10^-3	0.3s	0.0001	10^-3	0.9s	0.0012
10	10^-3	0.4s	3.5744	10^-3	2.7s	0.0012
20	10^-3	0.7s	3.5744	10^-3	5.5s	0.0012
30	10^-3	0.1s	3.5744	10^-3	10.6s	0.0012
40	10^-3	0.6s	3.5744	10^-3	17.1s	0.0012
50	10^-3	0.3s	3.5744	10^-3	24.7s	0.0012
60	10^-3	0.3s	3.5744	10^-3	33.7s	0.0012
70	10^-3	0.5s	3.5744	10^-3	44.8s	0.0012
80	10^-3	0.6s	3.5744	10^-3	57.2s	0.0012
90	10^-3	0.7s	3.5744	10^-3	1m 10.6s	0.0012
100	10^-3	0.9s	3.5744	10^-3	1m 25.7s	0.0001

Таблица 6.4: Результаты тестирования градиентного спуска на функции Экли.

	Golden section			Moore – Skelboe		
n	Prec	Time	Minimum	Prec	Time	Minimum
2	10^-3	0.2s	0.0006	10^-3	0.5s	0.0006
5	10^-3	0.6s	0.0002	10^-3	0.1s	0.0001
10	10^-3	0.3s	0.0002	10^-3	0.1s	0.0002
20	10^-3	0.4s	0.0001	10^-3	0.4s	0.0001
30	10^-3	0.1s	0.0001	10^-3	0.4s	0.0001
40	10^-3	0.6s	0.0001	10^-3	0.5s	0.0002
50	10^-3	0.4s	0.0001	10^-3	0.7s	0.0002
60	10^-3	0.5s	0.0001	10^-3	0.9s	0.0001
70	10^-3	0.8s	0.0001	10^-3	0.9s	0.0001
80	10^-3	0.6s	0.0001	10^-3	1.1s	0.0001
90	10^-3	0.8s	0.0001	10^-3	1.3s	0.0001
100	10^-3	0.8s	0.0001	10^-3	1.4s	0.0001

7 Выводы

Проведя тестирование на приведенных выше функциях, мы получаем вполне неплохой результат. Там, где классические способы поиска находят лишь локальный минимум, который значительно отличается от глобального, нам удалось найти глобальные с хорошей точностью. Также нам удалось сделать это достаточно быстро, во первых, благодаря координатному спуску мы избавились от экспоненциальной сложности поиска, как если бы мы делали это исключительно методом интервального анализа. Во вторых мы смогли значительно ускорить поиск, применяя алгорим Moore-Skelboe уже в градиентном спуске, при n = 100, время выполнения упало с 55.6s до 1.4s на функции Растригина и с 1m 25.7s до 1.4s на функции Экли.

Таблица 6.5: Результаты тестирования многомерноего алгоритма Moore-Skelboe на функции Растригина.

n	Prec	Time	Minimum
1	10^-3	0.7s	0.0001
2	10^-3	0.8s	0.0001
3	10^-3	0.2s	0.0001
4	10^-3	1.3s	0.0001
5	10^-3	3.2s	0.0002
6	10^-3	11.5s	0.0002
7	10^-3	49.2s	0.0003
8	10^-3	3m 42.3s	0.0003

Таблица 6.6: Результаты тестирования многомерноего алгоритма Moore-Skelboe на функции Экли.

n	Prec	Time	Minimum
1	10^-3	0.2s	0.0003
2	10^-3	0.3s	0.0005
3	10^-3	1.1s	0.0004
4	10^-3	2.7s	0.0005
5	10^-3	7.4s	0.0005
6	10^-3	21.4s	0.0005
7	10^-3	1m 10.1s	0.0005
8	10^-3	4m 10.6s	0.0003

Список литературы

[1] Charles R. Harris, K. Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke и Travis E. Oliphant. "Array programming with NumPy". В: *Nature* 585 (2020), с. 357—362. DOI: 10.1038/s41586-020-2649-2.

[2] Helmut Ratschek и Jon Rokne. "Interval methods". В: *Handbook of global optimization* (1995), с. 751—828.

[3] Stefano Taschini. *PyInterval*. URL: https://pyinterval.readthedocs.io/en/latest/ (дата обр. 14.05.2023).

[4] Guido Van Rossum и Fred L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697.

[5] Виктор Анатольевич Гончаров. "Методы оптимизации". В: *М.: Высшее образование* (2009).

[6] Андрей Владимирович Пантелеев и Татьяна Александровна Летова. *Методы оптимизации в примерах и задачах*. Высшая школа, 2008.

[7] Антон Борисович Сергиенко. “Тестовые функции для глобальной оптимизации”. В: *Сибирский государственный аэрокосмический университет имени академика М.Ф. Решетнева* (2015).