Санкт-Петербургский государственный университет

Хэ Пин

Выпускная квалификационная работа

Поиск глобального максимума. случай многих равных эстремумов

Уровень образования: магистратура

Направление 01.04.02 «Прикладная математика и информатика»

Основная образовательная программа ВМ.5751.2020 «Математическое моделирование, программирование и искусственный интеллект»

Научный руководитель:

Профессор, кафедра статистического моделирования

д. ф.-м. н., профессор С. М. Ермаков

Рецензент:

Директор, ООО «Аптик СПб»

к. ф.-м. н., К. А. Тимофеев

Санкт-Петербург

2022

Saint Petersburg State University
Applied Mathematics and Computer Science
Statistical Modelling

He Ping

Graduation Project

Global extremum search. the case of many equal extrema

Scientific Supervisor:
Professor, Department of Statistical
Modelling S. M. Ermakov

Reviewer:
General manager, Ltd «Spb Uptick»
K. A. Timofeev

Saint Petersburg
2022

Оглавление

Введение

Задача поиска наибольшего (наименьшего) значения функции многих переменных является составной частью многих важных прикладных задач и ей посвящена обширная литература. Различают задачи поиска с ограничениями и без, классические методы решения задачи и стохастические, задачи с одним наибольшим значением в заданной области и многими равными значениями. Последний случай получил развитие сравнительно недавно [1] и задачей данной работы является его обобщение и дальнейшее развитие. В процессе работы были найдены интересные приложения к задаче решения систем уравнений и получены новые результаты в этой области. Работа содержит реферативную часть, которая на простых примерах показывает недостатки существующих классических методов решения экстремальных задач, описывает известные связи между экстремальными задачами и задачами решения систем нелинейных уравнений и описывает наиболее известный и употребительный стохастический метод имитации отжига для решения экстремальных задач. Новыми результатами является Лемма 2, являющаяся уточнением результатов работы [1] и основанный на ней метод отделения и приближенного вычисления корней систем уравнений. Приводятся результаты вычислений для некоторых характерных примеров.

Глава 1

Простейшие методы решения экстремальных задач. Связь с методами решения уравнений

1.1. Простейшие методы решения экстремальных задач. Связь с методами решения уравнений

1.1.1. Градиентный спуск

Градиентный спуск (GD)[2] — самый простой и старый из методов оптимизации. Это итерационный метод, процедура оптимизации выполняется итерациями, каждая из которых улучшает целевое значение. Интуитивно понятный выбор направления спуска d — это направление самого крутого спуска. Следование направлению самого крутого спуска гарантированно приведет к улучшению, при условии, что целевая функция является плавной, размер шага достаточно мал, и мы еще не находимся в точке, где градиент равен нулю. Направление самого крутого спуска — это направление, противоположное градиенту ∇f , отсюда и название градиентный спуск.

Алгоритм 1: Градиентный спуск[3]

Input: f, T, начальная точка $x_1 \in K$, последовательность размеров шагов $\{\eta_t\}$
for $t = 1$ *to* T **do**
  Let $y_{t+1} = x_t - \eta_t \nabla f(x_t), x_{t+1} = \Pi_{\mathcal{K}}(y_{t+1})$
end
return x_{T+1}

Хотя выбор μ_t может иметь значение на практике, теоретически сходимость алгоритма градиентного спуска хорошо понятна и приведена в следующей теореме. Ниже мы предполагаем, что функция ограничена таким образом, что $|f(x)| \leq M$.

Теорема 1. [3] Для неограниченной минимизации β-гладких функций и $\mu_t = \frac{1}{\beta}$, градиентный спуск сходится как :

$$\frac{1}{T}\sum_t \|\nabla_t\|^2 \leq \frac{4M\beta}{T} \tag{1.1}$$

1.1.2. Стохастический градиентный спуск

Добавление стохастичности к градиентному спуску может быть полезным в больших задачах нелинейной оптимизации. Седловые точки, где градиент очень близок к нулю, могут привести к тому, что методы спуска будут выбирать размеры шага, которые слишком малы, чтобы быть полезными.

Алгоритм 2: Стохастический градиентный спуск [3]

Input: f, T, начальная точка $x_1 \in K$, последовательность размеров шагов $\{\eta_t\}$
for $t = 1$ *to* T **do**
  Let $y_{t+1} = x_t - \eta_t \hat{\nabla} f(x_t), x_{t+1} = \Pi_{\mathcal{K}}(y_{t+1})$
end
return x_{T+1}

Теорема 2. [4] При безусловной минимизации β-гладких функций и $\eta_t = \eta = \sqrt{\frac{M}{\beta\sigma^2 T}}$, алгоритм стохастического градиентного спуска сходится как

$$\mathbb{E}\left[\frac{1}{T}\sum_t \|\nabla_t\|^2\right] \leq 2\sqrt{\frac{M\beta\sigma^2}{T}} \tag{1.2}$$

1.2. Системы нелинейных уравнений. Метод Ньютона и секущие. Недостатки

1.2.1. Метод Ньютона

Метод Ньютона является наиболее эффективным методом поиска корней методом итерации.

Метод Ньютона-Рафсона имеет следующий вид:

Теорема 3. [6]Предположим, что f дважды непрерывно дифференцируемо на открытом интервале (a, b) и что существует$f(x^*) \in (a, b)$ с $f'(x^*) \neq 0$. Метод Ньютона определяется с помощью последовательности

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, k = 1, 2, \ldots. \tag{1.3}$$

Предположим также, что x_k сходится к x^* при $k \to \infty$. Тогда для k достаточно большого,

$$|x_{k+1} - x^*| \leq M|x_k - x^*|^2 \text{ if } M > \frac{|f''(x^*)|}{2|f'(x^*)|}. \tag{1.4}$$

Алгоритм 3: (Ньютона-Рафсона)[5]Находит нули $g : \mathbb{R}^n \to \mathbb{R}^n$, при условии $(\partial g(z)/\partial z)^{-l}$ существует и является непрерывным.

1 Выберите $z_0 \in \mathbb{R}^n$.

2 Установите $i = 0$.

3 Вычислите $g(z_i)$.

4 Если $g(z(z_i) = 0$, остановитесь; в противном случае вычислите z_{i+l} в соответствии с (8), установите $i = i + 1$ и перейдите к шагу 2

Таким образом, x_k сходится к x^* квадратично

Доказательство. Пусть $e_k = x_k - x^*$, так что $x_k - e_k = x^*$, установив $x = x_k$ и $h = -e_k$, имеем

$$f(x_k - e_k) = f(x_k) - e_k f'(x_k) + \frac{(e_k)^2}{2} f''(\xi_k). \tag{1.5}$$

для некоторых ξ_k между x_k и x^* . Поскольку $x_k - e_k = x^*$ и $f(x^*) = 0$, мы имеем

$$0 = f(x_k) - (x_k - x^*) f'(x_k) + \frac{(e_k)^2 f''(\xi_k)}{2f'(x_k)}, \tag{1.6}$$

Поскольку производная f непрерывна $f'(x^*) \neq 0$, мы имеем $f'(x_k) \neq 0$ при x_k достаточно близком к x^* . Таким образом, мы можем разделить на $f'(x_k)$, чтобы получить

$$0 = \frac{f(x_k)}{f'(x_k)} - (x_k - x^*) + \frac{(e_k)^2 f''(\xi_k)}{2f'(x_k)}. \tag{1.7}$$

который, по определению метода Ньютона, дает

$$x_{k+1} - x^* = \frac{(e_k)^2 f''(\xi_k)}{2f'(x_k)} \tag{1.8}$$

Итак

$$|x_{k+1} - x^*| \leq \frac{|f''(\xi_k)|}{2|f'(x_k)|} |x_k - x^*|^2. \tag{1.9}$$

Таким образом, $f'(x_k)$ сходится к $f'(x^*)$, а поскольку ξ_k между x_k и x^*, ξ_k сходится к x^*, и, следовательно, $f''(\xi_k)$ сходится к $f''(x^*)$, Поэтому для достаточно больших k,

$$|x_{k+1} - x^*| \leq M|x_{k+1} - x^*|^2 \text{ if } M > \frac{|f''(\xi_k)|}{2|f'(\xi_k)|} \tag{1.10}$$

■

1.2.2. Дискретизация метода Ньютона, Метод секущих

Метод Ньютона может быть дискретизирован [5], чтобы избежать явного вычисления производных. Самый прямой подход к тому, чтобы избежать вычисления $\mathrm{F}'(x)$, — это просто аппроксимировать частные производные $\partial_j f_i(x)$ разностными коэффициентами.ents. Двумя типичными и часто используемыми разностными приближениями являются

$$\partial_j f_i(x) \doteq (1/h_{ij})\Big[f_i(x + \sum_{k=1}^{j} h_{ik}e^k) - f_i(x + \sum_{k=1}^{j-1} h_{ik}e^k)\Big], \tag{1.11}$$

и

$$\partial_j f_i(x) \doteq (1/h_{ij})[f_i(x + h_{ij}e^j) - f_i(x)], \tag{1.12}$$

где h_{ij} заданы параметры дискретизации, а e^j - вектор координат j-й координатный вектор. В более общем смысле, пусть $h \in R^\nu$ представляют собой вектор параметров и пусть $\Delta_{ij}(x, h)$ указывают разностная аппроксимация $\partial_j f_i(x)$ с имущества, когда $\partial j f_i(x)$ существует, тогда

$$\lim_{h\to 0} \Delta_{ij}(x, h) = \partial_j f_i(x), i, j = 1, \ldots, n. \tag{1.13}$$

Затем с помощью разностной матрицы

$$j(x, h) = (\Delta_{ij}(x, h)), \tag{1.14}$$

Имеем

$$x_{k+1} = x_k - J(x_k, h_k)^{-1}F(x_k), k = 0, 1, \ldots, \tag{1.15}$$

называется дискретизированной итерацией Ньютона. Обратите внимание, что векторы параметров $h_k \in \mathbb{R}^\nu$ могут изменяться в зависимости от индекса итерации.

Дискретизированные методы Ньютона представляют собой n-мерные обобщения одномерных дискретизированных методов Ньютона:

$$x^{k+1} = x^k - \Big[\frac{f(x^k + h^k) - f(x^k)}{h^k}\Big]^{-1} f(x^k), k = 0, 1, \ldots. \tag{1.16}$$

Метод секущих [7] является частным случаем (1.16)

$$x^{k+1} = x^k - \Big[\frac{f(x^{k-1}) - f(x^k)}{x^{k-1} - x^k}\Big]^{-1} f(x^k), k = 0, 1, \ldots, \tag{1.17}$$

где $h^k = x^{k-1} - x^k$.

Можно использовать аналогичные варианты выбора h^k в n-мерных дискретизированных методах Ньютона. Однако, чтобы обсудить полученные методы в подходящей общности, желательно начать с несколько иного подхода к одномерным методам (1.16). Следующая итерация x^{k+1} из (1.16) является решением линеаризованного уравнени

$$l(x) = x^k - \Big[\frac{f(x^k + h^k) - f(x^k)}{h^k}\Big](x - x^k) + f(x^k) = 0. \tag{1.18}$$

Важным моментом сейчас является то, что l можно рассматривать двумя различными способами: либо это рассматривается как приближение касательной линии $l_{\mathrm{T}}(x) = f'(x^k)(x \in x^k) + f(x^k)$, или как линейная интерполяция f между точками x^k и $x^k + h^k$. В случае дискретизированного Методами Ньютона была использована первая интерпретация, и мы заменили производную $F'(x^k)$ на аппроксимирующую ее матрицу $J(x^k, h^k)$ разностных коэффициентов.

Чтобы расширить вторую точку зрения до n измерений, мы заменяем каждую "компонентную поверхность" $f_i, i = 1, \ldots, n, \in R^{n+l}$ гиперплоскостью , которая интерполирует f_i в $n + 1$ заданных точках $x^{k,j}, j = 0, \ldots, n$ в районе x^k. То есть, вектор a^i и a_i скалярное должны быть нашли такого, что аффинное отображение $L_i x = a_i + x^{\mathrm{T}} a^i$ удовлетворяет

$$L_i x^{k,j} = f_i(x^{k,j}), j = 0, 1, \ldots, n. \tag{1.19}$$

Следующая итерация x^{k+l} затем получается как пересечение этих n гиперплоскостей в R^{n+1} с гиперплоскостью $x = 0$; то есть x^{k+1} является решением линейной системы $L_i X = 0, i = 1, \ldots, n$. Это описывает общий метод секущей в n измерениях. В зависимости от выбора точек интерполяции $x^{k,j}, j = 0, \ldots, n$, существует множество возможных различных конкретных методов, но, прежде чем дать какой-либо из них, мы разработаем некоторые результаты по n-мерной линейной интерполяции, чтобы увидеть, какой на самом деле может быть следующая итерация рассчитано.

Метод Ньютона и метод секущих являются известными методами решения нелинейных уравнений. Использование метода секущих позволяет избежать нахождения производной, но оба метода имеют общие недостатки:

1. Зависит от первоначального приближения

2. Может зацикливаться бесконечно

3. Результат может быть слишком далеко от локального корня

4. Самое важное: невозможно автоматически отделить корни

В следующем разделе мы обсудим возможность использования метода глобальной оптимизации для автоматического разделения корней.

1.3. Сведение к задаче поиска экстремума

Мы можем преобразовать задачу решения уравнений в задачу нахождения глобального минимума.

$$H(x) = \sum_{l=1}^{s} p_l g_l^2(X) = \min, \tag{1.20}$$

Или нахождение глобального максимума:

$$H(X) = \frac{1}{1 + \sum_{l=1}^{s} p_l g_l^2(X)} = \max, \tag{1.21}$$

1.4. Моделирование распределений

Разработанные в работе алгоритмы основаны на моделировании распределений. Хорошо известна формула обращения, дающая принципиальное решение задачи вычисления реализации ξ случайной величины с заданной непрерывной функцией распределения F. Реализация ξ выражается через реализацию равномерно распределенной на $[0,1]$ случайной величины α. Справедлива

Теорема 4. [8] Пусть F — непрерывная функция распределения на $\mathbf{R}^1$ и пусть обратная к ней определена формулой

$$F^{-1}(x) = \inf\{t : F(t) = x; 0 < x < 1\}, \tag{1.22}$$

тогда если α — равномерно распределенная на $[0,1]$ случайная величина, то $F^{-1}(\alpha)$ имеет своей функцией распределения F.

Доказательство. Пусть $\xi = F^{-1}(\alpha)$. Поскольку $F_\xi(x) = P(\xi \le x)$, то имеем

$$P(\xi \le x) = P(F^{-1}(\alpha) \le x) = P(\inf\{t : F(t) = \alpha\} \le x) = P(\alpha \le F(x)). \tag{1.23}$$

Далее, так как $0 \le F(x) \le 1$, а α равномерно распределена на $[0,1]$, то $P(\alpha \le F(x)) = F(x)$, то есть $F_\xi(x) = F(x)$, что и доказывает теорему ■

Далее будем рассматривать задачу моделирования случайного вектора $\Xi = (\xi_1, \dots \xi_s)$, предполагая существование плотности $f(x_1, \dots x_s)$ совместного распределения его компонент на $\mathbf{R}^s$. Отметим, что это предположение делается скорее для упрощения обозначений, чем по существу.

Компоненты реализации случайного вектора Ξ могут быть получены путем последовательного моделирования следующей последовательности плотностей:

$$\varphi_1(x_1) = \int_{-\infty}^{+\infty} dx_2 \dots \int_{-\infty}^{+\infty} dx_s f(x_1, \dots, x_s) \text{ для } \xi_1,$$

$$\varphi_2(x_2/\xi_1) = \frac{\int_{-\infty}^{+\infty} dx_3 \dots \int_{-\infty}^{+\infty} dx_s f(\xi_1, , x_2, \dots, x_s)}{\int_{-\infty}^{+\infty} dx_2 \dots \int_{-\infty}^{+\infty} dx_s f(\xi_1, x_2, \dots, x_s)} \text{ для } \xi_2, \tag{1.24}$$

$$\varphi_s(x_s/, \dots, \xi_s - 1) = \frac{f(\xi_1, \dots, \xi_{s-1}, x_s)}{\int_{-\infty}^{+\infty} f(\xi_1, \dots, \xi_{s-1}, x_s) dx_s} \text{ для } \xi_3,$$

то есть xi_1 находится путем моделирования $\varphi_1(x)$ — частной (одномерной) плотности ее распределения; и далее, если найдены выборочные значения $\xi 1, \dots, \xi_k (k < s)$, строится условная плотность распределения ξ_{k+1} и моделируется соответствующее выборочное значение ξ_{k+1}.

Процедура требует s независимых реализаций равномерно распределенных случайных величин $\alpha_1, \dots, \alpha_s$ и принципиально решает задачу моделирования случайного вектора Ξ. Необходимость вычисления интегралов делает ее еще более сложной, чем в одномерном случае, однако возможность разного рода замен переменных позволяет в отдельных случаях получать от нее практическую пользу.

Разумеется также, что нумерацию переменных x_i можно изменить произвольным образом.

Если непосредственное практическое применение формул (1.24) приводит к сложным алгоритмам, то теоретическое использование их или их модификаций может быть весьма плодотворным.

1.5. Марковский процесс

Марковский процесс [9] (или цепь Маркова) в x-пространстве представляет собой случайную величину, зависящую от дискретного времени, для которой вероятность состояния x_n в определенное время $t_n = n\Delta t$ обозначаемая как $P_n(x_n)$, эволюционирует

как функция дискретного времени t_n в соответствии с основным уравнением (1.25) вида:

$$P_{n+1(x_{n+1})} = \sum_{x_n} W(x_{n+1}, x_n) P_n(x_n), \tag{1.25}$$

где $W(x_{n+1}, x_n)$ является условной вероятностью (или оператором перехода) для перехода от x_n к x_{n+1} за данный временной шаг Δt. Условная вероятность, которая не зависит от n из-за однородности по времени (1.26), представляет собой неотрицательную и нормированную по столбцам (обычно несимметричную) матрицу, т.е.,

$$W(x', x) \geq 0, \quad \sum_{x'} W(x', x) = 1. \tag{1.26}$$

тличительной особенностью марковского процесса является то, что P_{n+1} зависит только от P_n, а не от предыдущих времен.

Основное уравнение позволяет нам, в принципе, итеративно вычислять временную эволюцию вероятности P_n, как только задано начальное условие, например, на итерации $n = 0$. Марковский процесс является сходящимся,$P_n(x) \to p_{eq}(x)$ для $n \to \infty$ при довольно общих обстоятельствах к единственному равновесному распределению. Достаточным условием для этого является то, что оператор перехода W является эргодическим и удовлетворяет уравнению детального баланса (1.27):

$$W(x', x)\bar{P}(x) = W(x', x)\bar{P}(x'), \tag{1.27}$$

С некоторыми $\bar{P}(x) \geq 0$, $\bar{P}(x)$ появляющееся в уравнении — это именно то единственное стационарное решение, к которому имеет место сходимость. Основное уравнение для $n \to \infty, p_n(x) \to p_{eq}(x) = \bar{P}(x)$.

1.6. Алгоритм метода Метрополиса

Метод Метрополиса[9] — это метод моделирования цепи Маркова для получения реализаций случайной величины с заданной плотностью распределения $\bar{P}(x) = p_{eq}(x)$: например, это может быть распределение Больцмана $P_{Boltzmann}$. В этом случае должно выполняться уравнение детального баланса

$$W(x', x)P_{eq}(x) = W(x', x)P_{eq}(x'). \tag{1.28}$$

Существует довольно большая свобода в выборе W, и поэтому конструкция представлена во многих вариантах (Metropolis MC, Heat Bath MC и т.д.). Например, Metropolis

и сотрудники представили следующую очень простую схему. Они предложили выбирать вероятность перехода $T(x', x)$, определяющую вероятность перехода от x к x', которую можно выбирать с большой свободой, без каких-либо требований детального баланса. В принципе, пробное распределение T должно быть: 1) простым для реализации на компьютере, 2) эргодичным (любая конфигурация должна быть достижима за конечное число шагов) и 3) неотрицательным ($T(x', x) \geq 0$) и нормализованный к единице ($\sum_{x'} T(x', x) = 1$). Возможно множество простых примеров: например , для задачи континуума можно взять плоское распределение в компактной области (Вставка), многомерное гауссово или лоренцево (Коши) распределение. Чтобы определить $W(x', x)$, удовлетворяющее подробному условию баланса в (1.29), новая конфигурация x' генерируемая выбранной пробной вероятностью перехода $T(x', x)$ должна приниматься только с вероятностью:

$$A(x', x) = \min\left\{1, \frac{T(x, x')P_{eq}(x')}{T(x', x)P_{eq}(x)}\right\}, \tag{1.29}$$

так что результирующая условная вероятность $W(x', x)$ задается формулой:

$$W(x', x) = A(x', x)T(x', x) \text{ for } x' \neq x, \tag{1.30}$$

Глава 2

Модификации

2.1. Имитация отжига и его модификация

2.1.1. История имитационного отжига

[4] Метод Имитации отжига получил свое название от физического процесса отжига в твердых телах. При отжиге твердого тела твердое тело нагревается, а затем медленно охлаждается. Чем медленнее охлаждение, тем меньше дефектов решетки появляется, и в конечном итоге получается кристалл с совершенной структурой. Метод имитации отжига связывает этот термодинамический процесс с глобальной задачей дискретной оптимизации. При применении методов имитации отжига решения принимаются путем сравнения двух решений, так как температурный параметр низкий, способность подниматься вверх уменьшается, в конечном итоге сходясь к глобальному оптимальному решению.

2.1.2. Определение терминов

[4] Для описания специфических особенностей алгоритма имитации отжига [10] для задач дискретной оптимизации необходимо несколько определений. Пусть - пространство решений (т.е. множество всех возможных решений). Пусть $f : \Omega \to \Re$ - целевая функция, определенная в пространстве решений. Цель состоит в том, чтобы найти глобальный минимум ω^* (т.е., $\omega^* \in \Omega$, такой, что $f(\omega) \geq f(\omega^*)$) для всех $\omega \in \Omega$). Целевая функция должна быть ограничена, чтобы гарантировать, существование ω^*. Определить $\mathsf{N}(\omega)$ как функцию окрестности для $\omega \in \Omega$, следовательно, связанная с каждым решением $\omega \in Omega$, которое является соседними решениями $\mathsf{N}(\omega)$, которые могут быть достигнуты за одну итерацию локального поиска.

Моделирование отжига начинается с начального решения $\omega \in \Omega$, соседнего решения $\omega' \in \mathsf{N}(\omega)$ Затем генерируется (либо случайным образом, либо с использованием заранее заданного правила). Имитация отжига основана на критерии приемлемости Метрополиса (Метрополис и др., 1953), который моделирует, как термодинамическая система переходит от текущего решения (состояния) $\omega \in \Omega$ к кандидату решение $\omega' \in \mathsf{N}(\omega)$,

в котором минимизируется энергосодержание. Вариант решения, ω', принимается как текущее решение на основе вероятности принятия

$$P\{\text{Accept } \omega' \text{ as next solution}\} = \begin{cases} \exp[-(f(\omega') - f(\omega))/t_k] & \text{если } f(\omega') - f(\omega) > 0 \\ 1 & \text{если } f(\omega') - f(\omega) \leq 0 \end{cases} \tag{2.1}$$

определить t_k как параметр температуры на итерации (внешнего цикла) k, такой, что

$$t_k > 0 \text{ для всех } k \text{ и } \lim_{k \to +\infty} t_k = 0 \tag{2.2}$$

Эта вероятность принятия является основным элементом механизма поиска при моделировании отжига. Если температура снижается достаточно медленно, то система может достичь равновесия (устойчивого состояния) на каждой итерации k. Пусть $f(\omega)$ и $f(\omega')$ обозначают энергии (значения функции), связанные с решениями $\omega \in \Omega$ и $\omega' \in \mathsf{N}(\omega)$ соответственно. Это равновесие следует распределению Больцмана, которое можно описать как вероятность того, что система находится в состоянии $\omega \in \Omega$ с энергией $f(\omega)$ при температуре T такой, что

$$\mathbf{P}\{\text{Система находится в состоянии } \omega \text{ при температуре } T\} = \frac{\exp(-f(\omega)/t_k)}{\sum_{\omega'' \in \Omega} \exp(-f(\omega'')/t_k)} \tag{2.3}$$

Если вероятность генерации решения-кандидата ω' из соседей решения $\omega \in \Omega$ равна $g_k(\omega, \omega')$, где

$$\sum_{\omega' \in \mathsf{N}(\omega)} g_k(\omega, \omega') = 1, \text{ для всех } \omega \in \Omega, k = 1, 2, \ldots, \tag{2.4}$$

тогда неотрицательная квадратная стохастическая матрица может быть определена с вероятностями перехода

$$\mathbf{P}_k(\omega, \omega') = \begin{cases} g_k(\omega, \omega') \exp(-\Delta_{\omega,\omega'}/t_k) & \omega' \in \mathsf{N}(\omega), \omega' \neq \omega \\ 0 & \omega' \notin \mathsf{N}(\omega), \omega' \neq \omega \\ 1 - \sum_{\substack{\omega'' \in \mathsf{N}(\omega) \\ \omega'' \neq \omega}} \mathbf{P}_k(\omega, \omega'') & \omega' = \omega \end{cases}, \tag{2.5}$$

для всех решений $i \in \Omega$ и всех итераций $k = 1, 2, \ldots$ и $\Delta_{\omega,\omega'} \equiv f(\omega') - f\omega$. Эти вероятности перехода определяют последовательность решений, генерируемых из неоднородной цепи Маркова (Ромео и Санджованни-Винсентелли, 1991). Обратите внимание, что жирный шрифт указывает на матричное (векторное) обозначение, а все векторы являются векторами строк.

Алгоритм 4: Имитация отжига (Эглезе, 1990) [4]

Выберите начальное решение $\omega \in \Omega$

Выберите счетчик изменения температуры $k = 0$

Выберите график охлаждения по температуре,t_k

Выберите начальную температуру $T = t_0 \geq 0$

Выберите расписание повторений, $\mathbf{M}_k$ которое определяет количество итераций, выполняемых в каждой температуре, t_k

Повторить

Установите счетчик повторений $m = 0$

    Повторить

    Сгенерируйте решение $\omega' \in \mathsf{N}(\omega)$

    Вычислить $\Delta_{\omega,\omega'} = f(\omega') - f(\omega)$

    Если $\Delta_{\omega,\omega'} \leq 0$, то $\omega \to \omega'$

    Если $\Delta_{\omega,\omega'} > 0$, то $\omega \to \omega'$ с вероятностью $\exp(-\Delta_{\omega,\omega'}/t_k)$

    $m \leftarrow m + 1$

    До $m = \mathbf{M}_k$

$k \leftarrow k + 1$

Пока критерием остановки является

Эта имитируемая формулировка отжига приводит к $\mathbf{M}_0 + \mathbf{M}_1 + \ldots + \mathbf{M}_k$ выполняемых итераций, где k соответствует значению, при котором выполняются критерии остановки. Кроме того, если $\mathbf{M}_k = 1$ для всех k, то температура меняется на каждой итерации.

2.1.3. Модификация

[1] Метод имитации отжига широко применяется для решения прикладных задач по поиску абсолютного экстремума. Его математическая сторона кратко может быть изложена следующим образом. Пусть $f(X)$, $X \in D$, $D \in R^s$, ограниченная функция, $|f(X)| \leq M$. Как правило, D также ограничена в R^s. Случай неограниченной D требует отдельного рассмотрения. Пусть также T — вещественный параметр,$T \in R1$. Можно показать, что функция (плотность распределения)

$$\overline{\mathfrak{M}}(T, X) = C(T) \exp\left(-\frac{f(X)}{T}\right), \tag{2.6}$$

где C — константа нормировки, зависящая от T , при $T \to 0$ слабо сходится к δ-функции, сосредоточенной в точке глобального минимума $f(X)$.

Алгоритм состоит в моделировании методом Метрополиса [11] последовательности плотностей (2.6) с убывающими значениями $T_1 > T_2 > \ldots > T_n$. Реализации соответствующих случайных величин, полученные при T_k, служат исходными для алгоритма Метрополиса при $T_k + 1, k = 1, 2, \ldots, n - 1$.

В [12] предлагалось для нахождения глобального максимума использовать другую функцию типа $\overline{\mathfrak{M}}(T, X)$, а именно

$$\mathfrak{M}(X, n) = \frac{f^n(X)}{\int\limits_D f^n(X) dX}, \tag{2.7}$$

При $n \to \infty$ также сходится к функции глобального максимума. Легко показать, что применение алгоритма Метрополиса приводит к модификации метода имитации отжига, где параметр n играет роль T . Далее мы будем рассматривать алгоритмы, основанные на моделировании $\mathfrak{M}(X, n)$ при $n \to \infty$. При этом в работе специально рассматривается случай двух и более одинаковых максимумов (минимумов) исследуемой функции.

Пусть $(\mathfrak{A}, D, \mu)$ — пространство с σ-алгеброй подмножеств D и конечной мерой μ, $f(X) \in L(D, \mu)$, f является μ-интегрируемой, ограниченной, неотрицательной функцией в D, имеющей некоторое множество M глобальных максимумов.

Рассмотрим последовательность распределений, заданных следующим образом:

$$\mathscr{P}_n(\mathscr{A}) = \frac{\int_{\mathscr{A}} f^n(x)\mu(dx)}{\int_D f^n(x)\mu(dx)}, \quad \forall \mathscr{A} \in \mathfrak{A}, \quad n = 1, 2, \ldots \tag{2.8}$$

Вопрос о характере предельного распределения в общем случае не является простым. Мы остановимся на двух специальных случаях, которые необходимы для конструирования алгоритма.

Пусть μ — дискретная мера, сосредоточенная в точках $x_1, x_2, \ldots$ Предварительно докажем следующую лемму.

Лемма 1. [1] Пусть q_i — вещественные числа, $0 \leq q_i < 1$ и

$$\sum_{i=1}^{\infty} q_i < \infty, \tag{2.9}$$

тогда справедливо равенство

$$\lim_{k \to \infty} \sum_{i=1}^{\infty} q_i^k = 0 \tag{2.10}$$

Доказательство. Для $\forall > 0$ в силу условия (2.9) найдется $\mathcal{N} \in \mathbb{N}$, такое что $\sum_{i=\mathcal{N}+1}^{\infty} q_i < \varepsilon/3$ и, поскольку $q_i < 1$, такое $k \in \mathbb{N}$, что для любого $qi < \mathcal{N}$ верно $q_i^k < 2^{-i}\varepsilon/3$.

Тогда для выбранных $\mathcal{N}$ и k будем иметь

$$\sum_{i=1}^{\infty} q_i^k = \sum_{i=1}^{\mathcal{N}} q_i^k + \sum_{i=\mathcal{N}+1}^{\infty} q_i^k \leq \frac{\varepsilon}{3} \sum_{i=1}^{\mathcal{N}} 2^{-i} = \frac{\varepsilon}{3} < \varepsilon, \quad \text{since} \quad \sum_{i=\mathcal{N}+1}^{\infty} q_i^k < \sum_{i=\mathcal{N}+1}^{\infty} q_i. \tag{2.11}$$

■

Пусть теперь Q — дискретное мультимодальное распределение с m одинаковыми модами

$$Q_k = \begin{pmatrix} x_1 & x_2 & \ldots \\ c_k q_1^k & c_k q_2^k & \ldots \end{pmatrix}, \quad k = 1, 2, \ldots, \tag{2.12}$$

причем $x_{i_1}, x_{i_2}, \ldots, x_{i_m}$ — одинаковые моды, $q_{i_1} = q_{i_2} = \ldots = q^*$, а c_k — константа нормировки, $c_k = (\sum_{i=1}^{\infty} q_i^k)^{-1}$. Обозначим

$$Q_0 = \begin{pmatrix} x_{i_1} & x_{i_2} & \ldots & x_{i_m} \\ 1/m & 1/m & \ldots & 1/m \end{pmatrix}. \tag{2.13}$$

Теорема 5. [1] Q_k при $k \to \infty$ слабо сходится к Q_0.

Доказательство. Покажем, что если $q_j \notin \{q_{i_1}, \ldots, q_{i_m}\}$, то $c_k q_j^k \to 0$ при $k \to \infty$:

$$\frac{q_j^k}{\sum_{i=1}^{\infty} q_i^k} = \frac{(q_j/q^*)^k}{\sum_{j=1}^{\infty}(q_j/q^*)^j} = \frac{(q_j/q^*)^k}{m + \sum_{j\notin(i_1,\ldots,i_m)}(q_j/q^*)^k} \tag{2.14}$$

При $k \to \infty$ второе слагаемое знаменателя стремится к нулю по лемме и очевидно вся дробь стремится к нулю также. ■

Таким образом, предельным распределением является равномерное распределение на множестве равных экстремумов.

Вернемся к общему случаю. Обозначим $G = \max(f(X)), X \in D)$, и положим

$$M = \{x \in D : f(x) = G\} \tag{2.15}$$

Не умаляя общности, будем считать, $G = 1$.

Справедлива следующая теорема.

Теорема 6. [1] Если $\mu(M) > 0$, то справедливо равенство

$$\lim_{n\to\infty} \frac{\int_{\mathscr{A}} f^n(x)\mu(dx)}{\int_D f^n(x)\mu(dx)} = \frac{\mu(A \cap M)}{\mu(M)} \tag{2.16}$$

Доказательство. Как и в дискретном случае имеем

$$\lim_{n\to\infty} \frac{\int_{\mathscr{A}} f^n(x)\mu(dx)}{\int_D f^n(x)\mu(dx)} = \lim_{n\to\infty} \frac{\int_{\mathscr{A}\cap M} f^n(x)\mu(dx) + \int_{\mathscr{A}\setminus M} f^n(x)\mu(dx)}{\int_M f^n(x)\mu(dx) + \int_{D\setminus M} f^n(x)\mu(dx)}$$
$$= \lim_{n\to\infty} \frac{\mu(\mathscr{A} \cap M) + \int_{\mathscr{A}\setminus M} f^n(x)\mu(dx)}{\mu(M) + \int_{D\setminus M} f^n(x)\mu(dx)}, \tag{2.17}$$

поскольку интеграл на $D\setminus M$ стремится к нулю. Напомним, что на множествах $\mathscr{A}\setminus M$ и $D\setminus M$ выполнено $f(x) < 1$ для всех $x(\mod \mu)$. ■

для построения прикладных алгоритмов, результат теоремы 6 может быть весьма полезным. Введем в рассмотрение функцию $(0 < \varepsilon < 1)$

$$G_\varepsilon = \begin{cases} f(x), & f(x) \le 1 - \varepsilon \\ 1 - \varepsilon, & f(x) > 1 - \varepsilon \end{cases} \tag{2.18}$$

Если $\mu(\{X : F(X) = 1\}) = 0$, мы будем предполагать, что при каждом $\varepsilon > 0$ справедливо неравенство $\mu(\{X : G(X) = 1 - \varepsilon\}) > 0$.

Во всяком случае так будет в случае меры Лебега μ для непрерывной F. При этом множество $\{X : G_\varepsilon(X) = 1 - \varepsilon\}$ может состоять из набора изолированных, при достаточно малых ε, множеств $M_i, i = 1, 2, \ldots$, на каждом из которых $G_\varepsilon(X) = \text{const}$, но $\mu(M_i)$ не обязано, вообще говоря, совпадать с $\mu(M_j)$ при $i \neq j$.

В соответствии с теоремой 6 абсолютные экстремумы $G_\varepsilon(X)$ будут распределены равномерно на множестве $M_\varepsilon = X : G_\varepsilon = 1 - \varepsilon$.

Алгоритм, являющийся аналогом метода имитации отжига, предполагает задание последовательности $(\varepsilon_k, n_k), k = 1, 2, \ldots$ При каждом ε_k и n_k метод Метрополиса используется для моделирования $(G_{\epsilon_k}(X))^{n_k} C_k$. Здесь ε_k — убывающая, а n_k — возрастающая последовательности и C_k — константа нормировки. При каждом k заполняется массив m реализаций моделируемых случайных величин.

2.1.4. Новые результаты

Новые результаты работы состоит в Лемме 2, являющаяся уточнением результатов работы [1] и основанный на ней метод отделения и приближенного вычисления корней систем уравнений. Приводятся результаты вычислений для некоторых характерных примеров

Лемма 2. При приведенных предположениях вероятность $P\{\Xi_m \in Q_1\}$ стремится к 1 при $m \to \infty$.

Доказательство. Пусть $Q_2 = D - Q_1$, то $D = Q_2 + Q_1$, $Q_2 = X : f(X) < M - \varepsilon$. Если

$$f_1(X) = \begin{cases} M - \varepsilon, X \in Q_1 \\ 0, X \in Q_2, \end{cases} \tag{2.19}$$

$$f_2(X) = \begin{cases} f(X), X \in Q_2 \\ 0, X \in Q_1, \end{cases} \tag{2.20}$$

То

$$\begin{aligned} f^m(X) = (f_1(X) + f_2(X))^m \\ f_1^m(X) + f_2^m(X) \\ (M - \varepsilon)^m + f_2^m(X) \\ (M - \varepsilon)^m [1 + (\frac{f_2(x)}{M - \varepsilon})^m], \end{aligned} \tag{2.21}$$

и

$$\frac{f_2(x)}{M-\varepsilon} < 1. \tag{2.22}$$

$$P\{\Xi_m \in Q_1\} = \int\limits_{Q_1} F_m(X)dx = \frac{(M-\varepsilon)^m}{\int\limits_D f_m(X)dX} = \frac{1}{1+\int\limits_D (\frac{f_2(X)}{M-\varepsilon})^m dX} \tag{2.23}$$

но

$$\int\limits_D (\frac{f_2(X)}{M-\varepsilon})^m dX \to 0, \tag{2.24}$$

что доказывает лемму. ■

2.2. Масштабирование функции

Возможно, потребуется масштабировать уравнение, чтобы иметь надлежащий масштаб для применения нашего алгоритма поиска корней. Вы можете масштабировать или уменьшать масштаб по определенным осям в соответствии с определенным соотношением.

$$H(X) \to H(k_i X), \tag{2.25}$$

где k_i — коэффициент масштабирования каждой оси.

Он также может быть масштабирован в соответствии с некоторыми функциями, такими как экспоненциальная функция и логарифмическая функция.

2.2.1. Алгоритм кластеризации DBSCAN

В результате имитации отжига получаются идеально разделенные равномерные распределения. На самом деле, для хорошей работы достаточно любого метода кластеризации. Для автоматического выбора количества кластеров мы используем кластеризацию на основе плотности.

DBSCAN [13] (пространственная кластеризация приложений с шумом на основе плотности) — это алгоритм, который может обнаруживать кластеры произвольной формы. DBSCAN обычно считает кластерами плотные области объектов в пространстве данных, которые разделены низкой плотностью.

Глава 3

Моделирование

3.1. Решение простой нелинейной системы по сравнению с классическим моделируемым отжигом

Основным преимуществом улучшенного метода имитации отжига является то, что все корни могут быть найдены одновременно, что может быть затруднительно для классического метода имитации отжига. Мы решим нелинейную систему в качестве примера: Используя классический имитированный отжиг для решения системы, давайте перейдем к глобальной задаче поиска минимума:

$$H(x) = \sum_{l=1}^{s} p_l g_l^2(X) = \min, \tag{3.1}$$

$$\begin{cases} f_1(x, y) = y - \sin(x^2) \\ f_2(x, y) = y - 0.5x - 2. \end{cases} \tag{3.2}$$

Проблема превращается в:

$$(y - \sin(x^2))^2 + (y - 0.5x - 2)^2 = \min, \tag{3.3}$$

Точечный график классического имитируемого отжига Рис. 3.1 выглядит следующим образом. Очевидно, что он не может найти все корни, и корни плохо отделились. Результат, кроме финальной точки, практически бесполезен.

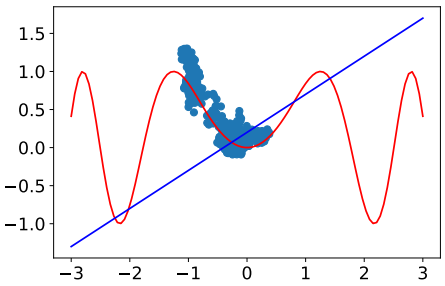


Рис. 3.1. Точечная диаграмма классического имитированного результата отжига

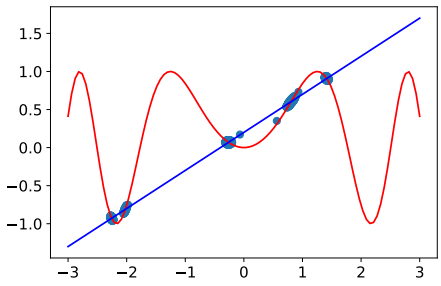


Рис. 3.2. Диаграмма рассеяния результата классического смоделированного отжига

Затем мы обсудим решение улучшенного моделирования отжига. Проблема трансформируется в:

$$H(x,y) = \frac{1}{1 + (y - \sin(x^2))^2 + (y - 0.5x - 2)^2} = \max, \tag{3.4}$$

Построим функцию согласно (2.18), выбрав $\varepsilon = 0.005$

$$H_\varepsilon(x,y) = \begin{cases} H(x,y), & H(x,y) \leq 0.995 \\ 0.995, & H(x,y) > 0.995 \end{cases}. \tag{3.5}$$

Рассмотрим класс функций $F(n, H(x,y)))$, где n — параметр, а $H(x,y))$ это функция, глобальные экстремумы которой должны быть найдены

$$F(n, H(x,y)) = \frac{H_\varepsilon^n(x,y)}{\int_D H_\varepsilon^n(x,y)dx} \tag{3.6}$$

Как было доказано выше, функция (3.6) при $n \to \infty$ сходится к равномерному распределению на множество точек равных глобальных экстремумов. Графики для ряда значений n, представленные в Рис. 3.3 и 3.4 наглядно это демонстрируют.

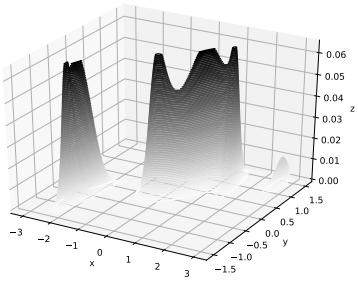


Рис. 3.3. График функции $F(10, H(x,y))$

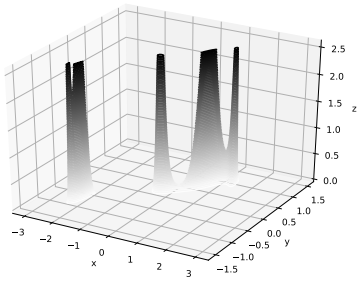


Рис. 3.4. График функции $F(100, H(x,y))$

Согласно теореме 6 абсолютные экстремумы будут распределены на множестве $M_\varepsilon = x : f_\varepsilon(x) = 1 - \varepsilon$. В этом случае $M_\varepsilon = M_1 \cup M_2, \mu(M_1) = \mu(M_2)$.

Используя улучшенный имитационный отжиг, мы можем найти все корни сразу. Потому что корни идеально разделены, можно использовать практически любой метод кластеризации. Поскольку распределение равномерное, DBSCAN больше подходит для текущей ситуации и может легко отфильтровать шум в начале имитации отжига.

3.1.1. Решение гиперболической системы, масштабирование модели

Мы используем гиперболическую функцию в качестве примера модели масштабирования. Когда корни будут очень близко, использование имитации отжига напрямую часто не позволяет разделить корни

$$\begin{cases} f_1(x,y) = x^2 + 5y^2 - 2.1x - y - 4.9 \\ f_2(x,y) = 2x^2 - y^2 - 4.2x + 0.2y + 3.4, \end{cases} \tag{3.7}$$

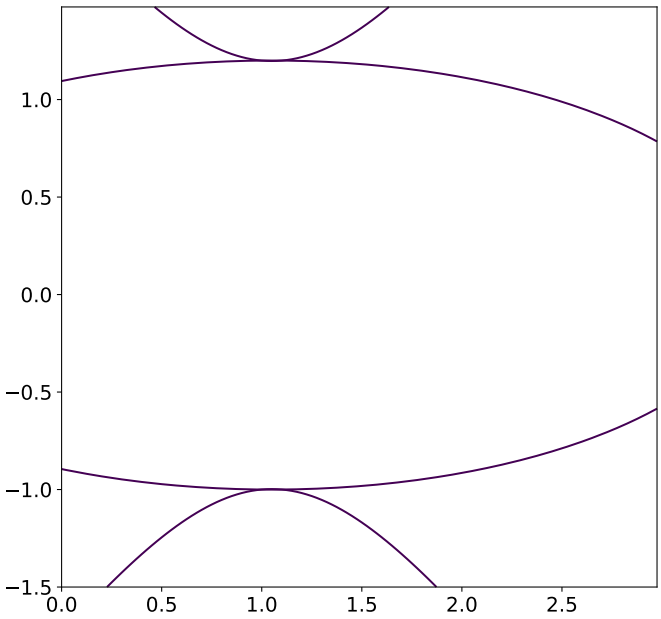


Рис. 3.5. смоделированный результат отжига гиперболической системы

Мы видим, что расположение корня узкое и длинное, это может затруднить для разделения корней. Мы можем решить систему $f(x/10, y)$, чтобы получить более широкую модель. Чтобы получить правильное решение, мы можем легко разделить x конечного результата на 10.

В этой системе мы используем параметры ниже, чтобы разделить все корни.

$$B = 10, a_j = 100, \varepsilon = 0.00000001, \tag{3.8}$$

Результат имитации отжига следующий. Мы видим, что масштабированную функ-

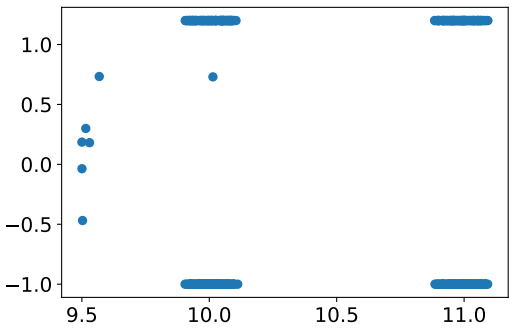


Рис. 3.6. Смоделированный результат отжига масштабированной гиперболической системы

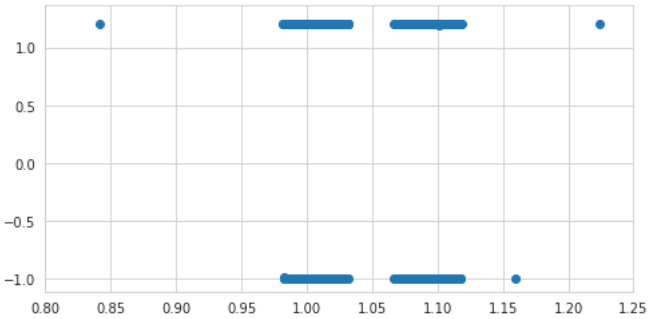


Рис. 3.7. смоделированный результат отжига гиперболической системы

цию легче отделить из-за больших расстояний от корней. Если мы не масштабируем функцию, результат может быть следующим:

Хотя окончательный корень все же можно отделить, используя параметры DBSCAN по умолчанию. может получить только два кластера, потому что точки расположены слишком близко.

Корени	масштабированный x	настоящий x	y
Корень №1	10.004614	1.000461	1.191932
Корень №2	10.005746	1.000575	-1.000014
Корень №3	10.986438	1.098644	1.199998
Корень №4	10.991736	1.099174	-1.000017

Таблица 3.1. Результат кластеризации DBSCAN гиперболической системы

3.1.2. Решение многомерной системы, анализ точности

Будем решать нелинейную систему и произведем анализ точности.

$$f_i(X) = \sum_{j=1}^{n} a_{i,j} x_i^2 + \mathbf{b}_i, i = 1, 2, \dots n, \tag{3.9}$$

$$a_{i,j} = \begin{pmatrix} 0.1 & 0.2 & 0.3 & 0.4 & 0.5 & 0.6 \\ 0.2 & 0.3 & 0.4 & 0.5 & 0.6 & 0.1 \\ 0.3 & 0.4 & 0.5 & 0.6 & 0.1 & 0.2 \\ 0.4 & 0.5 & 0.6 & 0.1 & 0.2 & 0.3 \\ 0.5 & 0.6 & 0.1 & 0.2 & 0.3 & 0.4 \\ 0.6 & 0.1 & 0.2 & 0.3 & 0.4 & 0.5 \end{pmatrix}, \tag{3.10}$$

$$\mathbf{b}_i = \begin{pmatrix} 1.8 & 2.05 & 2.0 & 1.95 & 1.9 & 1.85. \end{pmatrix} \tag{3.11}$$

Решение системы должно быть $\pm 1, \pm 1, \pm 1, \pm 1, \pm 1, \pm\sqrt{0.5}$, $2^6 = 64$ всего 64 корня.

Поскольку каждое решение симметрично, рассмотрим корень с положительными координатами и вычислим среднеквадратичную ошибку всех координат

$$\text{MSE} = \frac{1}{6} \sum_{i=1}^{n} (x_i - \hat{x}_i)^2 \tag{3.12}$$

В этой системе мы используем параметры ниже, чтобы разделить все корни.

$$B = 1, a_j = 1, \varepsilon = 0.03, m = 1, 5, 25, 125, \tag{3.13}$$

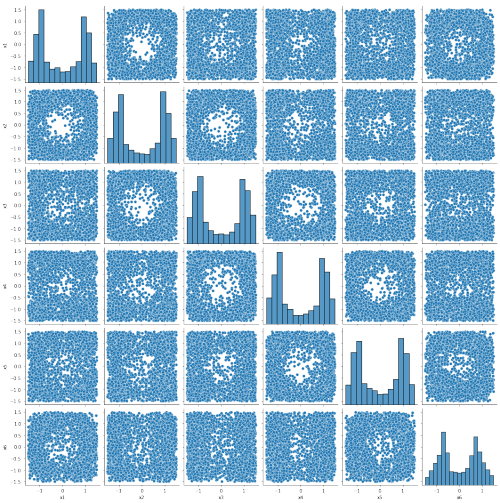
Результат имитации отжига следующий.

Мы видим, что корни расходятся по мере роста m. Когда m приближается к бесконечности, корни идеально разделены, но кажется, что точность не меняется со значением m. В виде увеличивается количество итераций, значительно возрастает точность.

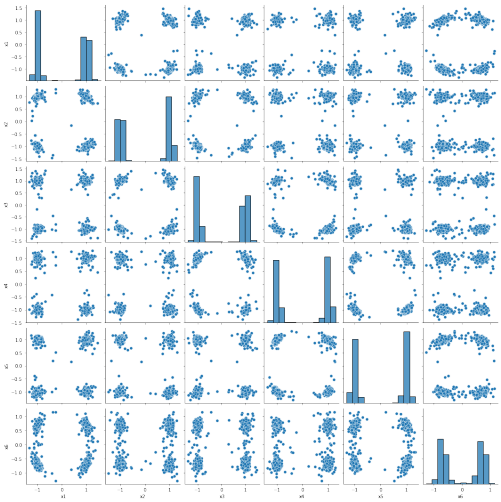
Далее мы обсудим влияние ε на разделяющие корни. ε для улучшения скорости попадания метод Монте-Карло, малый ε может уменьшить ошибки решения, но частота попаданий тоже меньше и требуется большее число итераций, большой ε может значительно улучшить показатель попаданий Монте-Карло, но слишком большой ε может привести к невозможности разделения корней. В качестве примера используем параметр

$$B = 1, a_j = 1, 125, \varepsilon = 0.01, 0.02, 0.05, 0.10, \tag{3.14}$$

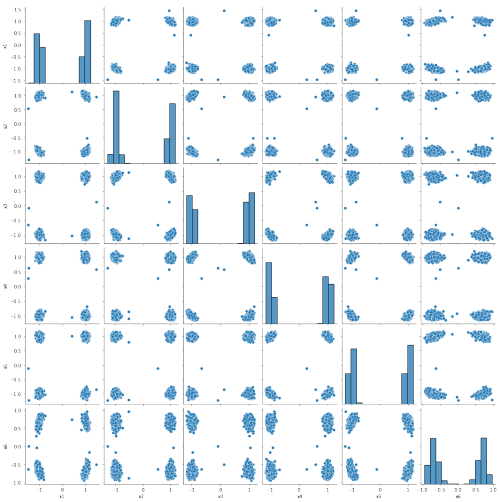
(*а*) $m = 1$



(*б*) $m = 5$



(*в*) $m = 25$



(*г*) $m = 125$

Рис. 3.8. График пары рассеяния для $\varepsilon = 0.03, m = 1, 5, 25, 125$

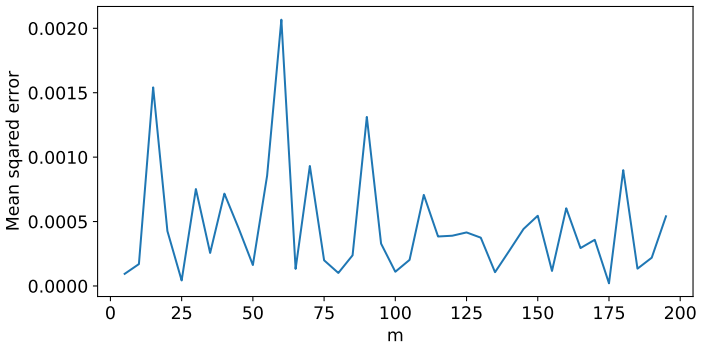
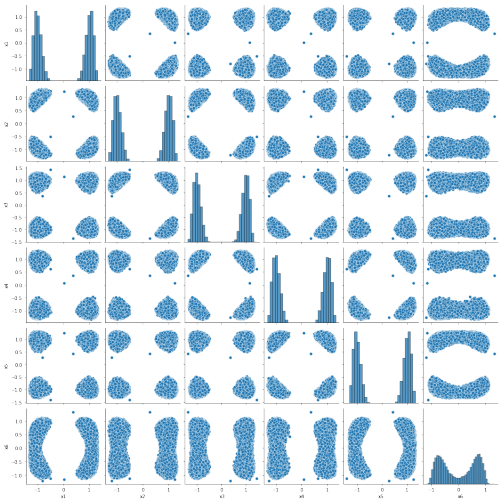
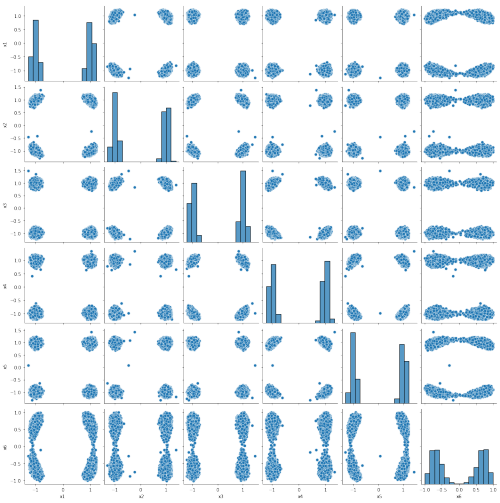


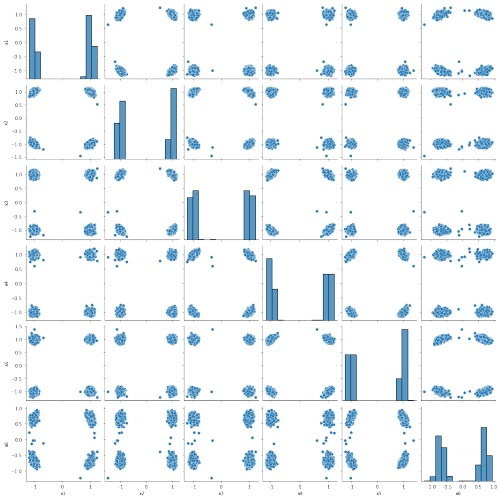
Рис. 3.9. Среднеквадратические ошибки при различных m



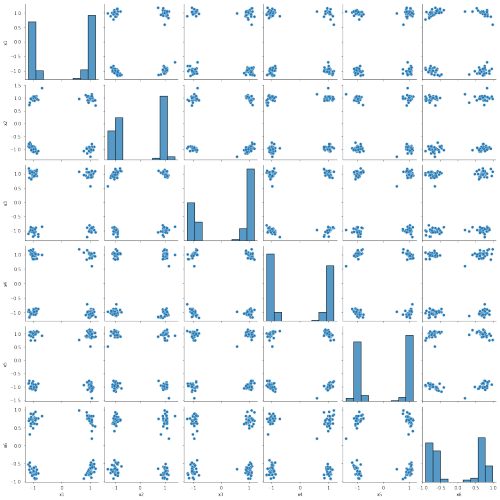
(*а*) $\varepsilon = 0.1$



(*б*) $\varepsilon = 0.05$



(*в*) $\varepsilon = 0.03$



(*г*) $\varepsilon = 0.01$

Рис. 3.10. График пары рассеяния для $m = 125, \varepsilon = 0.1, 0.05, 0.03, 0.02$

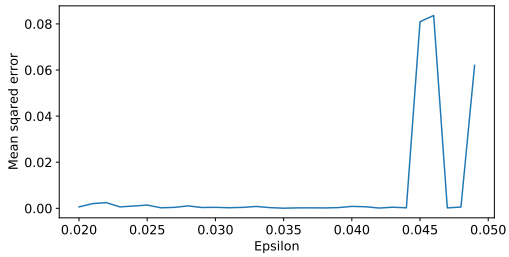


Рис. 3.11. MSE при различных ε

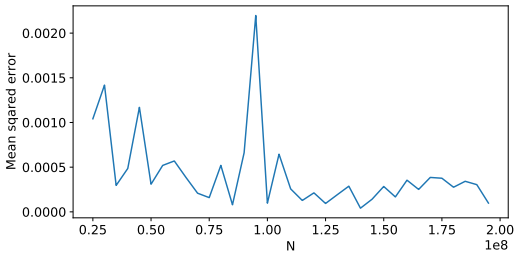


Рис. 3.12. MSE при различных N

Заключение

Таким образом, в работе было показано, что обобщение метода имитации отжига, полученное в работе [1](Ермаков, Куликов, Леора), может быть использовано в задаче автоматического разделения корней системы нелинейных уравнений . Также могут быть указаны начальные приближения для применения методов Ньютона и его модификаций. Теоретической основой указанных методов является лемма 2 главы 3 (новый результат), а также методы изменения масштаба и кластеризации. Приведенные численные примеры свидетельствуют о перспективности разработанных статистических методов.

Список литературы

1. С.М. Ермаков Д.В. Куликов С. Н. Л. К анализу метода имитации отжига в многоэкстремальном случае // Вестник Санкт-Петербургского университета. Математика. Механика. Астрономия. — 2017. — Vol. 4(62), no. 2. — P. 220–226. — Access mode: https://dspace.spbu.ru/bitstream/11701/6944/1/05-Ermakov.pdf.
2. Hazan E. Introduction to Online Convex Optimization. — 2019. — Sep. — 1909.05207.
3. Hazan E. Lecture Notes: Optimization for Machine Learning. — 2019. — Access mode: https://arxiv.org/pdf/1909.03550v1.
4. Mykel J. Kochenderfer T. A. W. Algorithms for Optimization. — The MIT Press, 2019. — Access mode: https://mitpress.mit.edu/books/algorithms-optimization.
5. Polak E. Computational Methods in Optimization: A Unified Approach. — Elsevier, 1971. — Access mode: https://www.sciencedirect.com/bookseries/mathematics-in-science-and-engineering/vol/77/suppl/C.
6. Overton M. Quadratic Convergence of Newton's Method. — 2017. — Access mode: https://cs.nyu.edu/~overton/NumericalComputing/newton.pdf.
7. Ortega J. M., Rheinboldt W. C. Iterative Solution of Nonlinear Equations in Several Variables. — Society for Industrial and Applied Mathematics, 2000. — jan. — Access mode: https://epubs.siam.org/doi/book/10.1137/1.9780898719468.
8. Ермаков С. Метод Монте-Карло в вычислительной математике. Вводный курс. — Российская Федерация : Невский Диалект, 2009. — ISBN: 978-5-7940-0140-2.
9. Stella L. Studies of Classical and Quantum Annealing : Ph.D. thesis ; SISSA. — 2005. — Access mode: https://web.archive.org/web/20060516151710/http://www.sissa.it/cm/thesis/2005/stella.pdf.
10. Henderson D., Jacobson S. H., Johnson A. W. The Theory and Practice of Simulated Annealing // Handbook of Metaheuristics / ed. by Glover F., Kochenberger G. A. — Boston, MA : Springer US, 2003. — P. 287–319. — ISBN: 978-0-306-48056-0. — Access mode: https://www.researchgate.net/publication/225260290_The_Theory_and_Practice_of_Simulated_Annealing.
11. Equation of State Calculations by Fast Computing Machines / Metropolis N., Rosenbluth A. W., Rosenbluth M. N., Teller A. H., and Teller E. // The Journal of Chemical Physics. — 1953. — June. — Vol. 21, no. 6. — P. 1087–1092. — Access

mode: https://www.researchgate.net/publication/239113707_Perspective_on_Equation_of_state_calculations_by_fast_computing_machines.

12. Ермаков С. Метод Монте-Карло и смежные вопросы. — МОСКВА : НАУКА, 1975.
13. El-Sonbaty Y., Ismail M., Farouk M. An efficient density based clustering algorithm for large databases // 16th IEEE International Conference on Tools with Artificial Intelligence. — IEEE Comput. Soc. — Access mode: https://ieeexplore.ieee.org/document/1374253.

Приложение А

Решение простой нелинейной системы по сравнению с классическим моделируемым отжигом

Модуль “code_simple.py”

```
#Классический имитация отжига

m=10000
epsilon=0.01
n_iterations = 1000000
import numpy as np
from numpy import asarray
from numpy import exp
from numpy.random import randn
from numpy.random import rand
from numpy.random import seed
import pandas as pd
from numba import njit

@njit
def ff(x1,x2):
    return (x2-np.sin(x1**2))**2+(x2-0.5*x1-0.2)**2


@njit
def ff2(x1,x2):
  return np.where(ff(x1,x2) <= 1-epsilon, ff(x1,x2), 1-epsilon)

@njit
def integral_approximation(ff, a, b,c,d):
  return (b-a)*(d-c)*np.mean(ff)

# Define bounds of integral
a = -3
b = 3
c = -3
```

```
d =3

# Generate function values
a_range = np.arange(a,b+0.0001,.0001)
b_range = np.arange(c,d+0.0001,.0001)

fx3 = ff2(a_range,b_range)**m
approx3 = integral_approximation(fx3,a,b,c,d)

@njit
def ff3(x1,x2):
    return ff2(x1,x2)**m/approx3
@njit
def simulated_annealing(ff3, n_iterations, step_size,temp):
    x1bests=list()
    x2bests=list()
    x1best =-2+rand()*4
    x2best= -3+rand()*6
    best_eval = ff3(x1best,x2best)
    x1curr,x2curr,curr_eval = x1best,x2best, best_eval
    for i in range(n_iterations):
        x1candidate = x1curr + (rand()-0.5) * step_size
        x2candidate = x2curr + (rand()-0.5) * step_size

        candidate_eval = ff3(x1candidate,x2candidate)
        if candidate_eval <=  best_eval:
            x1best,x2best ,best_eval = x1candidate,x2candidate, candidate_eval
            x1bests.append(x1candidate)
            x2bests.append(x2candidate)
        diff = candidate_eval - curr_eval
        t = temp / float(i + 1)
        metropolis = exp(-diff / t)
        if candidate_eval >  best_eval and rand() <metropolis:
            x1curr,x2curr,curr_eval = x1candidate,x2candidate, candidate_eval
            x1bests.append(x1candidate)
            x2bests.append(x2candidate)
    return x1bests,x2bests


```

```
71
72 seed(1)
73
74
75
76 step_size = 0.2
77
78
79
80 x1bests,x2bests=simulated_annealing(ff, n_iterations, step_size,10)
81
82 dfx1=pd.DataFrame(x1bests)
83 dfx2=pd.DataFrame(x2bests)
84
85 dfx1.rename( columns={0 :'x1'}, inplace=True )
86 dfx2.rename( columns={0 :'x2'}, inplace=True )
87
88 bests=pd.concat([dfx1,dfx2],axis=1)
89
90 print(bests)
91
92
93
94
95
96 # Модифицированный имитация отжига
97 m=100
98 epsilon=0.002
99 n_iterations = 1000000
100 import numpy as np
101 from numpy import asarray
102 from numpy import exp
103 from numpy.random import randn
104 from numpy.random import rand
105 from numpy.random import seed
106 import pandas as pd
107 from numba import njit
108
109 @njit
```

```
110 def ff(x1,x2):
111     return 1/(1+(x2-np.sin(x1**2))**2+(x2-0.5*x1-0.2)**2)
112 @njit
113 def ff2(x1,x2):
114   return np.where(ff(x1,x2) <= 1-epsilon, ff(x1,x2), 1-epsilon)
115
116 @njit
117 def integral_approximation(ff, a, b,c,d):
118   return (b-a)*(d-c)*np.mean(ff)
119
120 # Define bounds of integral
121 a = -10
122 b = 10
123 c = -10
124 d =10
125
126 # Generate function values
127 a_range = np.arange(a,b+0.0001,.0001)
128 b_range = np.arange(c,d+0.0001,.0001)
129
130 fx3 = ff2(a_range,b_range)**m
131 approx3 = integral_approximation(fx3,a,b,c,d)
132
133 @njit
134 def ff3(x1,x2):
135     return ff2(x1,x2)**m/approx3
136 @njit
137 def simulated_annealing(ff3, n_iterations, step_size):
138     x1bests=list()
139     x2bests=list()
140     x1best =-3+rand()*5
141     x2best= -2+rand()*5
142     best_eval = ff3(x1best,x2best)
143     x1curr,x2curr,curr_eval = x1best,x2best, best_eval
144     for i in range(n_iterations):
145         x1candidate = -3+rand()*5
146         x2candidate = -2+rand()*5
147
148         candidate_eval = ff3(x1candidate,x2candidate)
```

```
149         if candidate_eval >=  best_eval:
150             x1best,x2best ,best_eval = x1candidate,x2candidate, candidate_eval
151             x1bests.append(x1candidate)
152             x2bests.append(x2candidate)
153         diff = candidate_eval - curr_eval
154         metropolis = ff3(x1candidate,x2candidate)/approx3
155         if candidate_eval <  best_eval and rand() <metropolis:
156             x1curr,x2curr,curr_eval = x1candidate,x2candidate, candidate_eval
157             x1bests.append(x1candidate)
158             x2bests.append(x2candidate)
159     return x1bests,x2bests
160
161
162
163 seed(1)
164
165
166
167 step_size = 5
168
169
170
171 x1bests,x2bests=simulated_annealing(ff3, n_iterations, step_size)
172
173 dfx1=pd.DataFrame(x1bests)
174 dfx2=pd.DataFrame(x2bests)
175
176 dfx1.rename( columns={0 :'x1'}, inplace=True )
177 dfx2.rename( columns={0 :'x2'}, inplace=True )
178
179 bests=pd.concat([dfx1,dfx2],axis=1)
180
181 print(bests)
```

Приложение Б

Решение гиперболической системы, масштабирование модели

Модуль “code_hyper.py”

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 from numpy import asarray
4 from numpy import exp
5 from numpy.random import randn
6 from numpy.random import rand
7 import pandas as pd
8 from numpy.random import seed
9 from sklearn.cluster import KMeans
10 import scipy.integrate as integrate
11 import seaborn as sns
12 from numba import njit  #a simple library for cuda acceleration
13
14 #система поиска глобальных экстремумов
15 b=10
16 @njit
17 def ff0(x,y):
18 return 1/(b**2+(100*(x**2+5*y**2-2.1*x-y-4.9)**2+\
19 100*(2*x**2-y**2-4.2*x+0.2*y+3.4)**2))
20 @njit
21 def ff(x,y):
22 return 1/(b**2+(100*(x**2+5*y**2-2.1*x-y-4.9)**2+\
23 100*(2*x**2-y**2-4.2*x+0.2*y+3.4)**2))
24 epsilon=0.00000001
25
26
27 @njit
28 def ff2(x,y):
29 return np.where(ff(x/10,y) <= (1/b**2)-epsilon, ff(x,y), (1/b**2)-epsilon)
30 a1 = np.linspace(-4, 4, 300)
31 a2 = np.linspace(-4, 4, 300)
```

```
x,y = np.meshgrid(a1, a2)
vv2 = ff2(x, y)
qwe=15
@njit
def integral_approximation(f, a, b,c,d):
return (b-a)*(d-c)*np.mean(f)
# Определить границы интеграла
a = -qwe
b = qwe
c = -qwe
d =qwe
# Generate function values
x_range = np.arange(a,b+0.00001,.00001)
y_range = np.arange(c,d+0.00001,.00001)
fx3 = ff2(x_range,y_range)**100
approx3 = integral_approximation(fx3,a,b,c,d)
@njit
def ff3(x,y):
return ff2(x,y)**100/approx3

vv3 = ff3(x, y)


@njit
def simulated_annealing(ff3, n_iterations):
xbests=list()
ybests=list()
xbest =8+rand()*4
ybest= -1.5+rand()*3
best_eval = ff3(xbest,ybest)
xcurr,ycuur ,curr_eval = xbest,ybest, best_eval
for i in range(n_iterations):
xcandidate = 9.5+rand()*2
ycandidate = -1.5+rand()*3
candidate_eval = ff3(xcandidate,ycandidate)
if candidate_eval >= best_eval:
xbest,ybest, best_eval = xcandidate,ycandidate, candidate_eval
xbests.append(xcandidate)
ybests.append(ycandidate)
```

```
71 diff = candidate_eval - curr_eval
72 metropolis = ff2(xcandidate,ycandidate)**100/approx3
73 if candidate_eval <  best_eval and rand() <metropolis:
74 xcurr,ycurr ,curr_eval = xcandidate,ycandidate, candidate_eval
75 xbests.append(xcandidate)
76 ybests.append(ycandidate)
77 return xbests,ybests
78
79 seed(1)
80 n_iterations = 10000000
81
82 x1bests,x2bests= simulated_annealing(ff2, n_iterations)
83
84 # Кластеризация DBSCAN
85 def MyDBSCAN(D, eps, MinPts):
86 labels = [0]*len(D)
87 C = 0
88 for P in range(0, len(D)):
89 if not (labels[P] == 0):
90 continue
91 NeighborPts = regionQuery(D, P, eps)
92 if len(NeighborPts) < MinPts:
93 labels[P] = -1
94 else:
95 C += 1
96 growCluster(D, labels, P, NeighborPts, C, eps, MinPts)
97 return labels
98 def growCluster(D, labels, P, NeighborPts, C, eps, MinPts):
99 labels[P] = C
100 i = 0
101 while i < len(NeighborPts):
102 Pn = NeighborPts[i]
103 if labels[Pn] == -1:
104 labels[Pn] = C
105 elif labels[Pn] == 0:
106 labels[Pn] = C
107 # Найти всех соседей Pn
108 PnNeighborPts = regionQuery(D, Pn, eps)
109 if len(PnNeighborPts) >= MinPts:
```

```
NeighborPts = NeighborPts + PnNeighborPts
i += 1
def regionQuery(D, P, eps):
neighbors = []
for Pn in range(0, len(D)):

if numpy.linalg.norm(D[P] - D[Pn]) < eps:
neighbors.append(Pn)

return neighbors

eps=0.5
MinPts=20
cluster=MyDBSCAN(bests.values, eps, MinPts)
# Кластеризация DBSCAN
cluster_type=pd.DataFrame(cluster)
cluster_type.rename( columns={0 :'type'}, inplace=True )
solution=pd.concat([bests,cluster_type],axis=1)
result_every_root=pd.DataFrame()
for i in range(1,max(cluster)+1):
mean_of_root=solution.loc[solution['type'] == i].mean()
mean_of_rootT=pd.DataFrame(mean_of_root).T
result_every_root=pd.concat([result_every_root,mean_of_rootT],axis=0)

#восстановить решение поиска глобального экстремума
x1_true=result_every_root.iloc[:,0]/10
x1_true=pd.DataFrame(x1_true)
x1_true.rename( columns={'x1' :'x1_true'}, inplace=True )
pd.concat([result_every_root.iloc[:,0],x1_true,result_every_root.iloc[:,1],\
result_every_root.iloc[:,2]],axis=1)
print(result_every_root)
```

Приложение В

Решение многомерной системы, анализ точности

Модуль “code_analysis.py”

```
#Библиотеки используемые в программе
import numpy as np
from numpy import asarray
from numpy import exp
from numpy.random import randn
from numpy.random import rand
from numpy.random import seed
import pandas as pd
from numba import njit
n_test=range(5000000,200000000,5000000)
m_test=range(5,200,5)
float_epsilon = np.arange(0.02,0.05,0.001)
epsilon_test = list(float_epsilon)
mse_m=np.empty(0)
#for n_iterations in list(n_test):
#for m in list(m_test):
for epsilon in list(epsilon_test):
  #m=125
  #epsilon=0.03
  n_iterations = 100000000

  #Определите систему для нахождения глобальных экстремумов
  @njit
  def ff(x1,x2,x3,x4,x5,x6):
      return 1/((1+
                (0.1*x1**2+0.2*x2**2+0.3*x3**2+0.4*x4**2
                +0.5*x5**2+0.6*x6**2-1.8)**2+
                (0.2*x1**2+0.3*x2**2+0.4*x3**2+0.5*x4**2
                +0.6*x5**2+0.1*x6**2-2.05)**2+
                (0.3*x1**2+0.4*x2**2+0.5*x3**2+0.6*x4**2
                +0.1*x5**2+0.2*x6**2-2.0)**2+
                (0.4*x1**2+0.5*x2**2+0.6*x3**2+0.1*x4**2
                +0.2*x5**2+0.3*x6**2-1.95)**2+
```

```
                (0.5*x1**2+0.6*x2**2+0.1*x3**2+0.2*x4**2
                +0.3*x5**2+0.4*x6**2-1.9)**2+
                (0.6*x1**2+0.1*x2**2+0.2*x3**2+0.3*x4**2
                +0.4*x5**2+0.5*x6**2-1.85)**2
                ))

@njit
def ff2(x1,x2,x3,x4,x5,x6):
    return np.where(ff(x1,x2,x3,x4,x5,x6) <= 1-epsilon,
    ff(x1,x2,x3,x4,x5,x6), 1-epsilon)

@njit
def integral_approximation(ff, a, b,c,d,e,f,g,h,i,j,k,l):
    return (b-a)*(d-c)*(f-e)*(h-g)*(j-i)*(l-k)*np.mean(ff)

# Define bounds of integral
a = -3
b = 3
c = -3
d =3
e = -3
f = 3
g = -3
h =3
i = -3
j = 3
k = -3
l =3
# Generate function values
a_range = np.arange(a,b+0.0001,.0001)
b_range = np.arange(c,d+0.0001,.0001)
c_range = np.arange(e,f+0.0001,.0001)
d_range = np.arange(g,h+0.0001,.0001)
e_range = np.arange(i,j+0.0001,.0001)
f_range = np.arange(k,l+0.0001,.0001)
fx3 = ff2(a_range,b_range,c_range,d_range,e_range,f_range)**m
approx3 = integral_approximation(fx3,a,b,c,d,e,f,g,h,i,j,k,l)

@njit
```

```
73  def ff3(x1,x2,x3,x4,x5,x6):
74      return ff2(x1,x2,x3,x4,x5,x6)**m/approx3
75  #Модифицированный алгоритм имитации отжига
76  @njit
77  def simulated_annealing(ff3, n_iterations, step_size):
78      x1bests=list()
79      x2bests=list()
80      x3bests=list()
81      x4bests=list()
82      x5bests=list()
83      x6bests=list()
84      x1best =-1.5+rand()*3
85      x2best= -1.5+rand()*3
86      x3best =1.5+rand()*3
87      x4best= -1.5+rand()*3
88      x5best =-1.5+rand()*3
89      x6best= -1.5+rand()*3
90      best_eval = ff3(x1best,x2best,x3best,x4best,x5best,x6best)
91      x1curr,x2curr,x3curr,x4curr,x5curr,x6curr,curr_eval =\
92      x1best,x2best,x3best,x4best,x5best,x6best, best_eval
93      for i in range(n_iterations):
94          x1candidate = -1.5+rand()*3
95          x2candidate = -1.5+rand()*3
96          x3candidate = -1.5+rand()*3
97          x4candidate = -1.5+rand()*3
98          x5candidate = -1.5+rand()*3
99          x6candidate = -1.5+rand()*3
100         candidate_eval = ff3(x1candidate,x2candidate,x3candidate,\
101         x4candidate,x5candidate,x6candidate)
102         if candidate_eval >=  best_eval:
103             x1best,x2best,x3best,x4best,x5best,x6best, best_eval=\
104             x1candidate,x2candidate,x3candidate,x4candidate,\
105             x5candidate,x6candidate, candidate_eval
106             x1bests.append(x1candidate)
107             x2bests.append(x2candidate)
108             x3bests.append(x3candidate)
109             x4bests.append(x4candidate)
110             x5bests.append(x5candidate)
111             x6bests.append(x6candidate)
```

```
112           diff = candidate_eval - curr_eval
113           metropolis = ff(x1candidate,x2candidate,x3candidate,\
114           x4candidate,x5candidate,x6candidate)**m/approx3
115           if candidate_eval <  best_eval and rand() <metropolis:
116               x1curr,x2curr,x3curr,x4curr,x5curr,x6curr ,curr_eval =\
117               x1candidate,x2candidate,x3candidate,x4candidate,\
118               x5candidate,x6candidate, candidate_eval
119               x1bests.append(x1candidate)
120               x2bests.append(x2candidate)
121               x3bests.append(x3candidate)
122               x4bests.append(x4candidate)
123               x5bests.append(x5candidate)
124               x6bests.append(x6candidate)
125       return  x1bests,x2bests,x3bests,x4bests,x5bests,x6bests
126
127   #Кластеризация DBSCAN
128   def MyDBSCAN(D, eps, MinPts):
129       labels = [0]*len(D)
130       C = 0
131       for P in range(0, len(D)):
132           if not (labels[P] == 0):
133             continue
134           NeighborPts = regionQuery(D, P, eps)
135           if len(NeighborPts) < MinPts:
136               labels[P] = -1
137           else:
138             C += 1
139             growCluster(D, labels, P, NeighborPts, C, eps, MinPts)
140       return labels
141   def growCluster(D, labels, P, NeighborPts, C, eps, MinPts):
142       labels[P] = C
143       i = 0
144       while i < len(NeighborPts):
145           Pn = NeighborPts[i]
146           if labels[Pn] == -1:
147             labels[Pn] = C
148           elif labels[Pn] == 0:
149               labels[Pn] = C
150               # Find all the neighbors of Pn
```

```
151                 PnNeighborPts = regionQuery(D, Pn, eps)
152                 if len(PnNeighborPts) >= MinPts:
153                     NeighborPts = NeighborPts + PnNeighborPts
154             i += 1
155     def regionQuery(D, P, eps):
156         neighbors = []
157         for Pn in range(0, len(D)):
158 
159             if numpy.linalg.norm(D[P] - D[Pn]) < eps:
160               neighbors.append(Pn)
161 
162         return neighbors
163     seed(1)
164 
165 
166 
167     step_size = 5
168 
169 
170 
171     x1bests,x2bests,x3bests,x4bests,x5bests,x6bests=\
172     simulated_annealing(ff3, n_iterations, step_size)
173     #Преобразование результата в фрейм данных
174     dfx1=pd.DataFrame(x1bests)
175     dfx2=pd.DataFrame(x2bests)
176     dfx3=pd.DataFrame(x3bests)
177     dfx4=pd.DataFrame(x4bests)
178     dfx5=pd.DataFrame(x5bests)
179     dfx6=pd.DataFrame(x6bests)
180 
181     dfx1.rename( columns={0 :'x1'}, inplace=True )
182     dfx2.rename( columns={0 :'x2'}, inplace=True )
183     dfx3.rename( columns={0 :'x3'}, inplace=True )
184     dfx4.rename( columns={0 :'x4'}, inplace=True )
185     dfx5.rename( columns={0 :'x5'}, inplace=True )
186     dfx6.rename( columns={0 :'x6'}, inplace=True )
187     bests=pd.concat([dfx1,dfx2,dfx3,dfx4,dfx5,dfx6],axis=1)
188 
189     cluster=MyDBSCAN(bests.values, 0.4, 3)
```

```
190  cluster_type=pd.DataFrame(cluster)
191  cluster_type.rename( columns={0 :'type'}, inplace=True )
192  solution=pd.concat([bests,cluster_type],axis=1)
193  #решение глобального экстремума
194  result_every_root=pd.DataFrame()
195  for i in range(1,max(cluster)+1):
196    mean_of_root=solution.loc[solution['type'] == i].mean()
197    rootdf=pd.DataFrame(mean_of_root)
198    rootdf.rename( columns={0 :i}, inplace=True )
199    mean_of_rootT=rootdf.T
200    result_every_root=pd.concat([result_every_root,mean_of_rootT],axis=0)
201  df=result_every_root.iloc[:,:6]
202  greatest=df.loc[df.sum(1).idxmax()].values
203  ap=np.array([1,1,1,1,1,np.sqrt(0.5)])
204  #Вычисление среднеквадратичную ошибку
205  mse=mean_squared_error(ap,greatest)
206  mse_m=np.append(mse_m,mse)
```