Федеральное государственное автономное образовательное учреждение
высшего образования
**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**
Факультет систем управления и робототехники

Лабораторная работа №1
Градиентный спуск

Студенты: Бахтаиров Р.А.,
Сайфуллин Д.Р.
Группа: R3243
Преподаватель: Попов А.М.

Санкт-Петербург,
2025 г.

1 Введение

В данной лабораторной работе реализованы и исследованы методы оптимизации: методы нулевого порядка (золотое сечение, параболы, Брент), градиентный спуск с различными стратегиями выбора шага, а также построение траектории с использованием потенциальной функции. Все методы протестированы на заданных функциях, включая мультимодальные, с построением графиков сходимости и анализом результатов.

Примечание: весь код для выполнения работы находится в файле `main.ipynb`.

2 Одномерная оптимизация нулевого порядка

2.1 Реализация методов

Реализованы следующие методы:

- **Метод золотого сечения**: сужение интервала с использованием $\phi = \frac{1+\sqrt{5}}{2}$.
- **Метод парабол (явная формула)**:

$$u = x_2 - \frac{(x_2 - x_1)^2(f(x_2) - f(x_3)) - (x_2 - x_3)^2(f(x_2) - f(x_1))}{2\left((x_2 - x_1)(f(x_2) - f(x_3)) - (x_2 - x_3)(f(x_2) - f(x_1))\right)}$$

- **Метод парабол (солвер)**: аппроксимация параболы через три точки с решением системы уравнений $ax^2 + bx + c = f(x)$.

2.2 Тестовые функции

В первом задании необходимо протестировать реализованные методы на следующих функциях:

1. $f(x) = -5x^5 + 4x^4 - 12x^3 + 11x^2 - 2x + 1$ на $[-0.5, 0.5]$
2. $f(x) = -\ln^2(x-2) + \ln^2(10-x) - x^{0.2}$ на $[6, 9.9]$
3. $f(x) = -3x\sin(0.75x) + e^{-2x}$ на $[0, 2\pi]$
4. $f(x) = e^{3x} + 5e^{-2x}$ на $[0, 1]$
5. $f(x) = 0.2x\ln x + (x-2.3)^2$ на $[0.5, 2.5]$

2.3 Анализ

Построим графики сходимости методов для каждой из функций и сравним результаты.

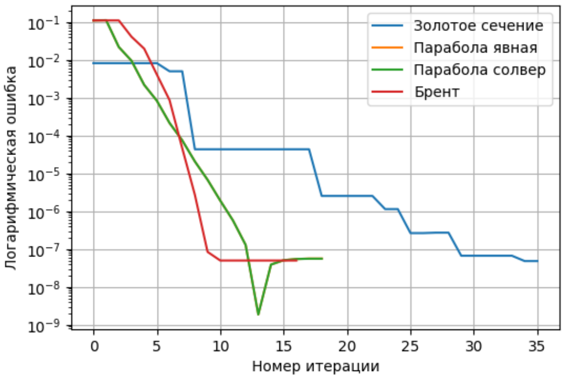

Рис. 1: Сходимость методов для функции 1

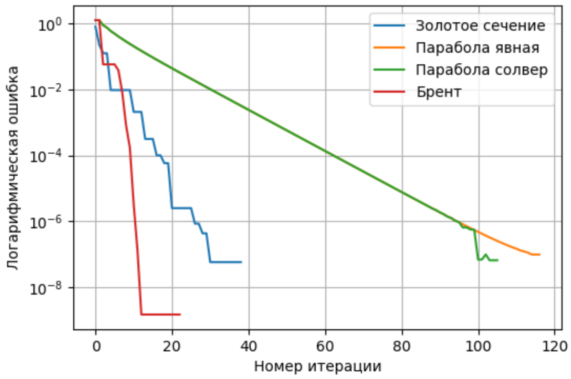

Рис. 2: Сходимость методов для функции 2

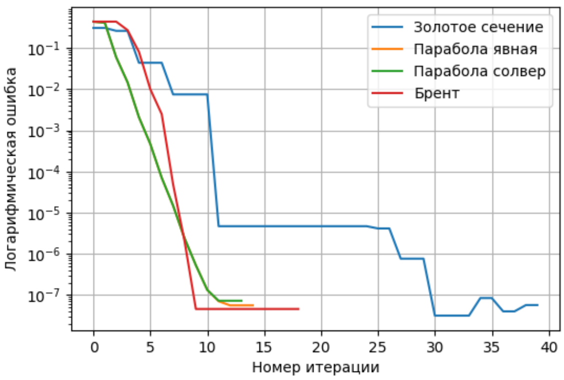

Рис. 3: Сходимость методов для функции 3

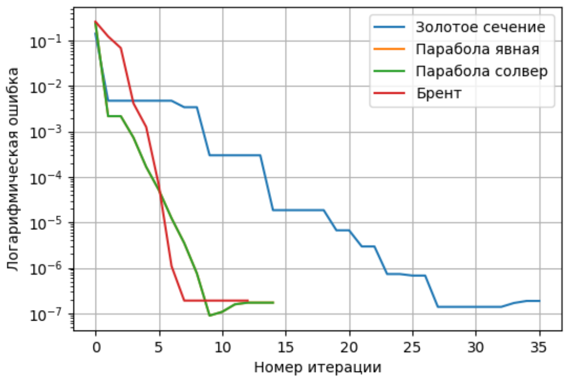

Рис. 4: Сходимость методов для функции 4

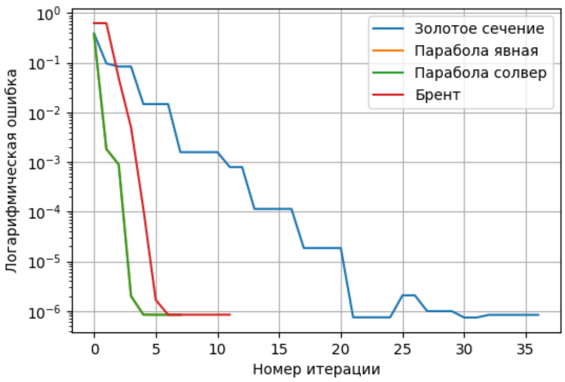

Рис. 5: Сходимость методов для функции 5

Анализируем результаты:

1. На графике видно, что метод золотого сечения сходится медленнее всех: ошибка убывает ступенчато и лишь к 15–20 итерации достигает порядка 10^{-7}. Методы парабол (как явная формула, так и через solve) и Брента выходят на точность $10^{-7} - 10^{-8}$ гораздо быстрее (примерно к 7–10 итерации). В итоге наименьшее число итераций показывает метод Брента.

2. Здесь параболический метод через solve (зелёная кривая) убывает по логарифмической шкале почти линейно, то есть довольно медленно достигает нужной точности (лишь к 100+ итерациям ошибка около 10^{-8}). Зато метод Брента (красная кривая) очень быстро (к 15–20 итерации) падает до 10^{-8}. Золотое сечение (синяя кривая) тоже быстрее, чем «парабола solve», но чуть отстаёт от Брента. Парабола (явная формула) (оранжевая кривая) в итоге добирается к точности 10^{-8} чуть раньше 110 итераций.

3. Методы парабол (зелёная и оранжевая) и Брента (красная) снова сходятся заметно быстрее золотого сечения. При этом Брент и «парабола solve» достигают порядка 10^{-7} уже на 5–7 итерации, тогда как золотое сечение выходит на ту же точность ближе к 15 итерации. Разница между параболическими методами здесь не столь велика: оба быстро уменьшают ошибку.

4. Аналогично, золотое сечение идёт «ступеньками» и лишь к 25–30 итерации выходит на 10^{-7}. Параболические методы и Брент делают это существенно быстрее (уже около 7–10 итерации). При этом метод Брента чуть обгоняет «параболу solve», а явная парабола примерно посередине.

5. На данном графике видно, что уже к 5–7 итерации методы Брента (красная) и парабол (зелёная) достигают уровня 10^{-7}, в то время как золотое сечение (синяя кривая) лишь к 15–20 итерации подходит к той же точности. Парабола (явная формула) (оранжевая) ведёт себя близко к «параболе solve», но иногда чуть медленнее на начальных шагах.

По приведённым графикам можно сделать следующие наблюдения:

- Методы, использующие параболическую интерполяцию (парабола и Брент), как правило, сходятся быстрее, чем «чистое» золотое сечение.

- Метод Брента почти во всех примерах демонстрирует наивысшую скорость сходимости, достигая заданной точности за наименьшее число итераций.

- «Парабола solve» иногда даёт чуть медленнее сходимость, если при построении системы уравнений возникают неблагоприятные численные условия, но всё равно обгоняет золотое сечение.

- «Парабола заданная явной формулой» обычно ведёт себя похоже на «параболу solve», иногда сходится быстрее, иногда медленнее — это зависит от свойств целевой функции и точности вычислений.

Теперь попробуем использовать встроенный оптимизатор `scipy.optimize.minimize` для каждого метода и сравним с результатом работы нашего алгоритма. Построим графики:

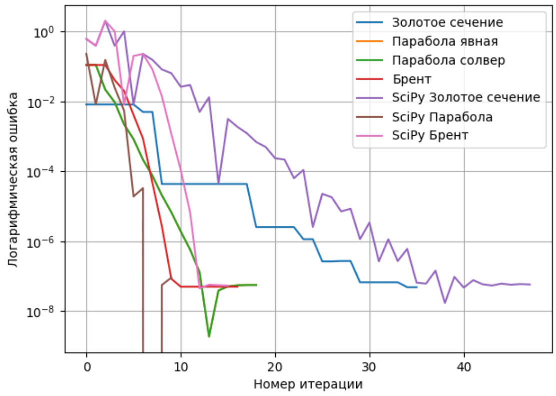

Рис. 6: Сходимость методов для функции 1

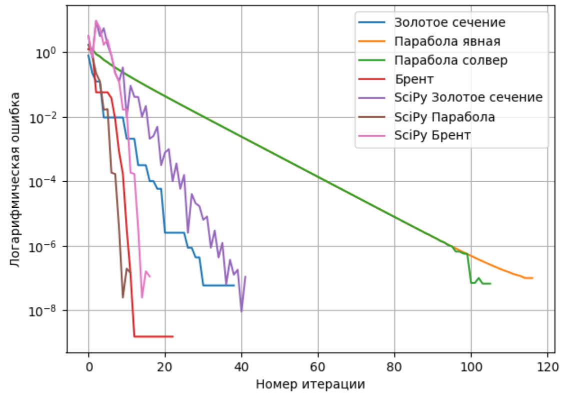

Рис. 7: Сходимость методов для функции 2

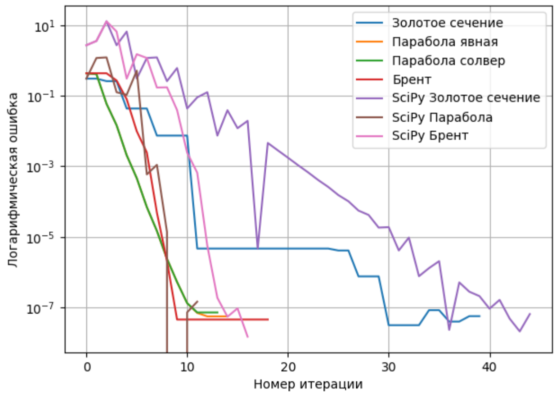

Рис. 8: Сходимость методов для функции 3

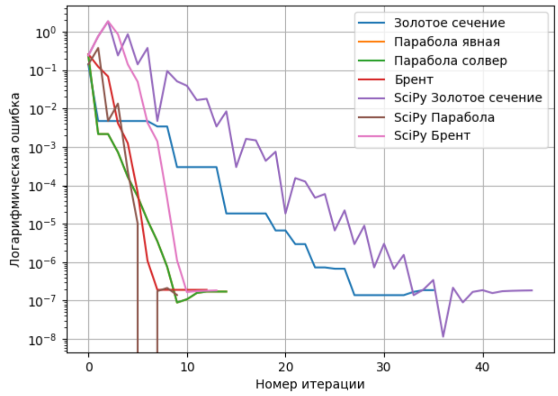

Рис. 9: Сходимость методов для функции 4

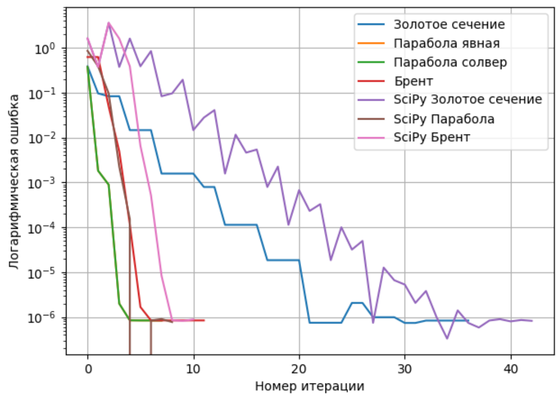

Рис. 10: Сходимость методов для функции 5

По приведённым графикам можно сделать следующие наблюдения:

1. На графике видно, что методы Брента (красная) и парабол (зелёная и оранжевая) достигают уровня 10^{-7} существенно быстрее золотого сечения (синяя кривая). При этом «SciPy Брент» и «SciPy Парабола» тоже показывают быструю сходимость, в то время как «SciPy Золотое сечение» идёт более ступенчато.

2. Параболический метод через solve (зелёная кривая) здесь убывает почти линейно в логарифмическом масштабе и достигает точности порядка 10^{-8} только ближе к 100-й итерации. Зато метод Брента (наш и из SciPy) выходит на ту же точность гораздо раньше (примерно к 15–20 итерации). Золотое сечение (наше и SciPy) идёт быстрее «параболы solve», но отстаёт от Брента.

3. Методы парабол (оранжевая и зелёная) и Брента (красная) снова сходятся заметно быстрее золотого сечения (синяя). При этом «SciPy Брент» (розовая) и «SciPy Парабола» (чёрная) также показывают быстрый спад ошибки, в отличие от «SciPy Золотого сечения» (фиолетовая), которое идёт более плавно и достигает 10^{-7} позже.

4. Аналогично, золотое сечение идёт «ступеньками» и лишь к 25–30 итерации достигает ошибки порядка 10^{-7}. Параболические методы (оба варианта) и Брент делают это гораздо быстрее, уже к 10–12 итерации. SciPy-реализации ведут себя сходным образом: «SciPy Брент» и «SciPy Парабола» выходят на малую ошибку быстрее, чем «SciPy Золотое сечение».

5. На данном графике видно, что уже к 5–7 итерации методы Брента (красная и розовая) и парабол (зелёная, оранжевая, чёрная) выходят на уровень 10^{-7}, тогда как золотое сечение (синяя и фиолетовая) лишь к 15–20 итерации. При этом «Парабола solve» иногда чуть запаздывает, если система уравнений даёт неблагоприятные числа, но всё равно опережает золотое сечение.

В пятом пункте требуется рассмотреть 2 мультимодальные функции, построить их графики и найти точки минимума каждым из методов. Будем исследовать следующие функции:

1. $f_1(x) = (x^2-3)^2+\sin(5x)$ на отрезке $[-2,3]$. Данная функция сочетает квадратичный компонент $(x^2-3)^2$, имеющий минимум при $x = \pm\sqrt{3}$, и колебательную часть $\sin(5x)$, которая добавляет несколько локальных минимумов.

2. $f_2(x) = 0.1\,x^2 + \cos(3x) + 2\sin(5x)$ на отрезке $[-2,2]$. Здесь квадратичный член $0.1\,x^2$ даёт «параболическое» возрастание на больших $|x|$, а сумма $\cos(3x) + 2\sin(5x)$ порождает множественные «волны» и локальные впадины.

Для каждой из этих функций запустим реализованные методы: золотое сечение, парабола (явная формула), парабола (через solve) и Брент. На рисунках ниже показаны графики каждой функции и отмечены точки, куда сошёлся тот или иной метод, а также приведена логарифмическая ошибка на каждом шаге (при наличии логов итераций).

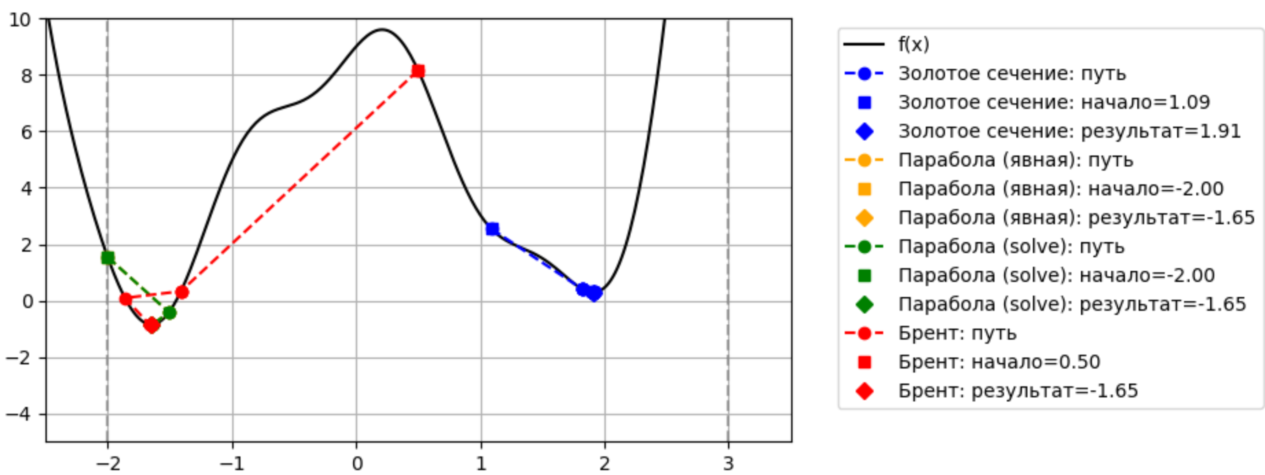

Рис. 11: График функции $f_1(x)$ и результаты методов на отрезке $[-2,3]$.

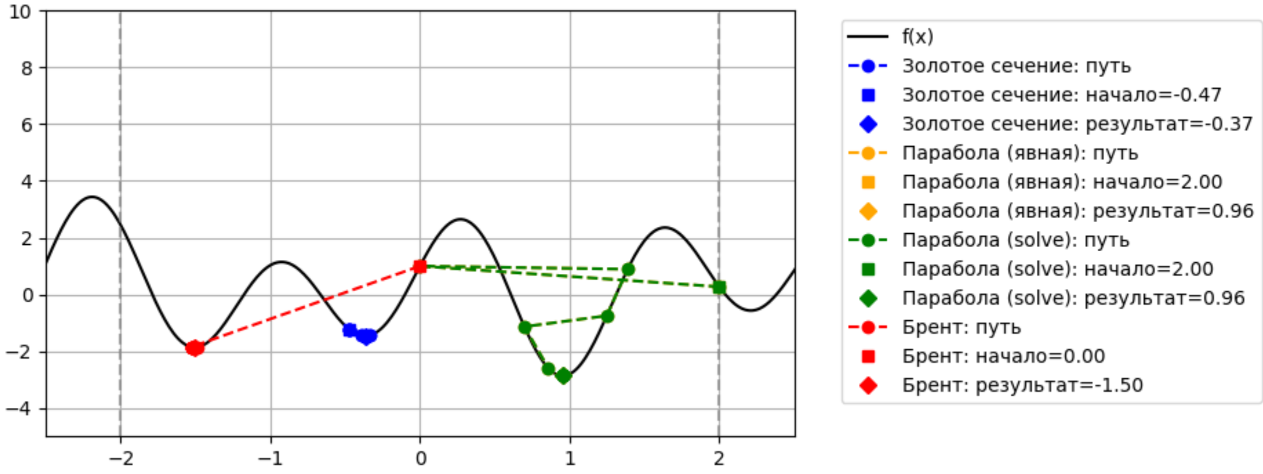

Рис. 12: График функции $f_2(x)$ и результаты методов на отрезке $[-2, 2]$.

Анализируем графики и делаем следующие выводы:

1. На первом графике видно, что при начальных приближениях в диапазоне от $x = -2.0$ до $x = 0.5$ методы «Парабола» (явная и solve) и Брент смещаются в локальный минимум около $x \approx -1.65$, тогда как «Золотое сечение» при старте с $x \approx 1.09$ уходит к другому локальному минимуму вблизи $x \approx 1.91$. Это показывает, что разные начальные условия (или интервалы) приводят к поиску разных локальных впадин при мультимодальной форме $f(x)$.

2. На втором графике функции, имеющей несколько волн на отрезке $[-2, 2]$, видно, что метод Брента (красная кривая), начав с $x = 0.00$, уходит к минимуму около $x = -1.50$, «Золотое сечение» (синий) при старте с $x \approx -0.47$ попадает в точку $x \approx -0.37$, а «Парабола» (явная и solve) при $x = 2.00$ сходится к минимуму около $x = 0.96$. Таким образом, каждая пара «старт + метод» нашла свою локальную впадину, что ещё раз подчёркивает, что унимодальные алгоритмы без многократного запуска не гарантируют нахождение глобального минимума при наличии нескольких локальных.

2.4 Выводы

1. На тестовых унимодальных функциях все методы успешно сходятся к точке минимума, однако методы с параболической аппроксимацией и метод Брента достигают заданной точности быстрее, чем золотое сечение; это подтверждается графиками логарифмической ошибки: золотое сечение даёт более ступенчатое уменьшение, а параболы и Брент часто выходят на значения 10^{-7}–10^{-8} за меньшее число итераций.

2. При сравнении с методами SciPy наблюдается схожая тенденция: реализация golden в `minimize_scalar` работает похожим образом на наше золотое сечение, а brent и bounded сходятся существенно быстрее; разница в точном числе итераций объясняется внутренними эвристиками SciPy и разными критериями остановки, однако общее преимущество параболических методов и Брента остаётся.

3. При тестировании на мультимодальных функциях все унимодальные алгоритмы (золотое сечение, парабола, Брент) могут находить разные локальные минимумы в зависимости от начального приближения или заданного отрезка, то есть ни один из рассмотренных методов не гарантирует поиск глобального минимума, если функция

имеет несколько впадин на заданном интервале; это подтверждают графики, где каждый метод останавливается в своей локальной точке.

4. Таким образом, для унимодальных задач методы парабол и Брента обеспечивают более быструю сходимость по сравнению с золотым сечением, а для мультимодальных функций без дополнительных приёмов (например, многократных запусков или глобальных алгоритмов) нет гарантии найти глобальный минимум.

3 Градиентный спуск

По заданию требуется реализовать следующие стратегии:

- Постоянный шаг: $\alpha_k = \text{const}$.
- Поиск β: минимизация вдоль направления с использованием методов нулевого порядка, $\alpha_k = \frac{\beta}{\|\nabla f(x_k)\|}$.
- Адаптивный шаг: $\alpha_k = \frac{1}{L_k}$ (константа Липшица).
- Армихо-Вульф: backtracking для выбора шага.

Для визуализации результатов работы стратегий рассмотрим задачу минимизации квадратичной функции $f(x) = x^\top Ax + b^\top x$, где $A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$, $b = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$.

Аналитически решение задаётся $\nabla f(x^*) = 0$, то есть $2A\,x^* + b = 0$, откуда $x^* = -\frac{1}{2}\,A^{-1}b$. Константа Липшица: $L = 2\lambda_{\max}(A)$.

Мы запускаем градиентный спуск с четырьмя стратегиями выбора шага и сравниваем их поведение на нескольких начальных приближениях.

3.1 Анализ

Протестируем алгоритм с каждой их стратегий, построим 2D- 3D-графики и отметим на них результат работы траекторий.

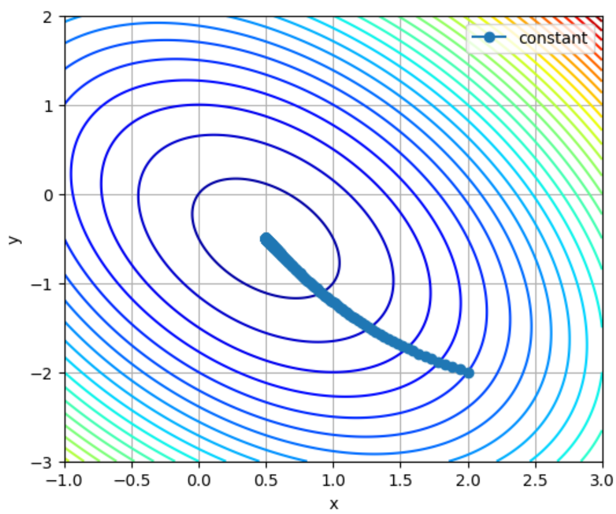

Рис. 13: Линии уровня, constant.

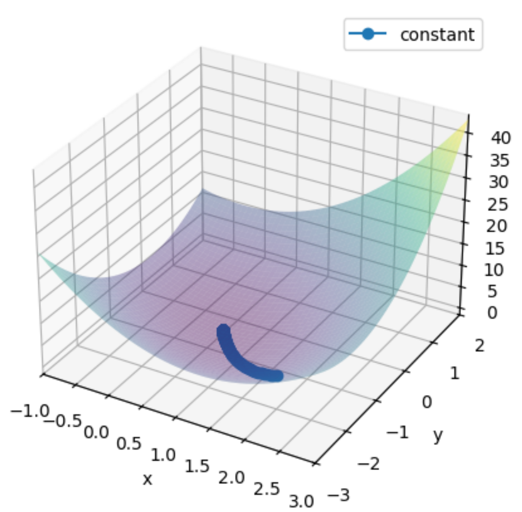

Рис. 14: 3D-график, constant.

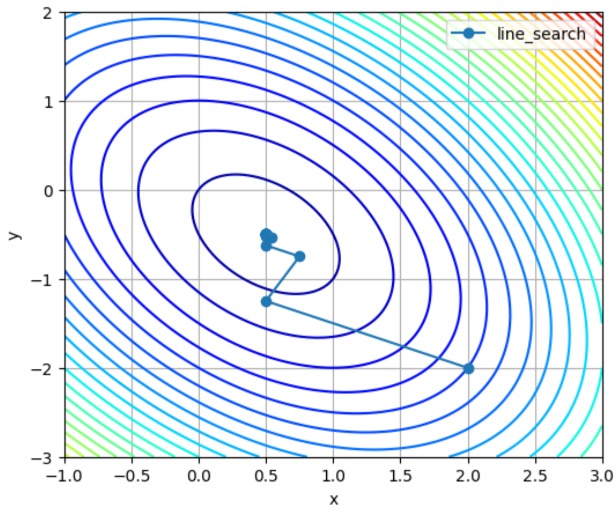

Рис. 15: Линии уровня, line_search.

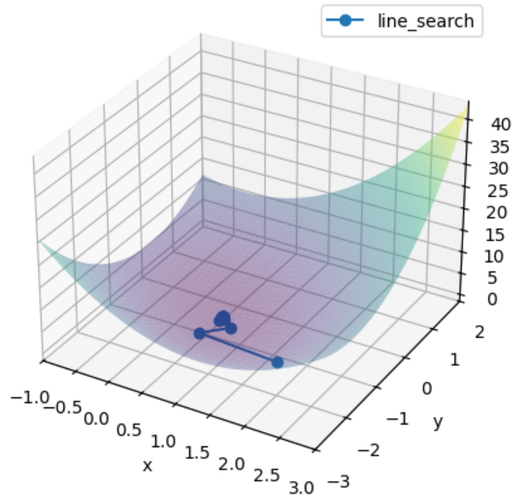

Рис. 16: 3D-график, line_searc.

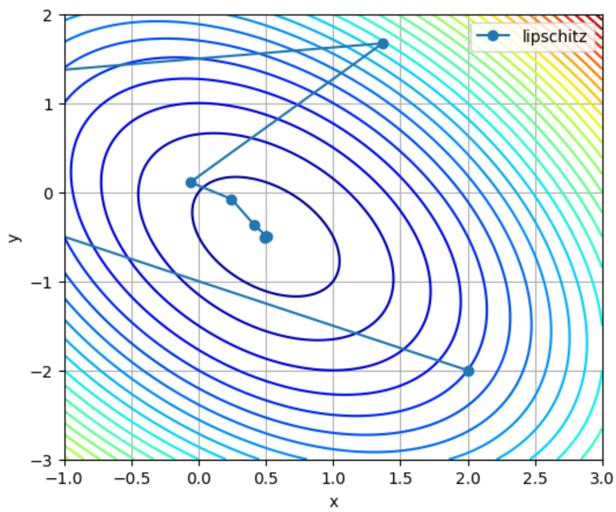

Рис. 17: Линии уровня, lipschitz.

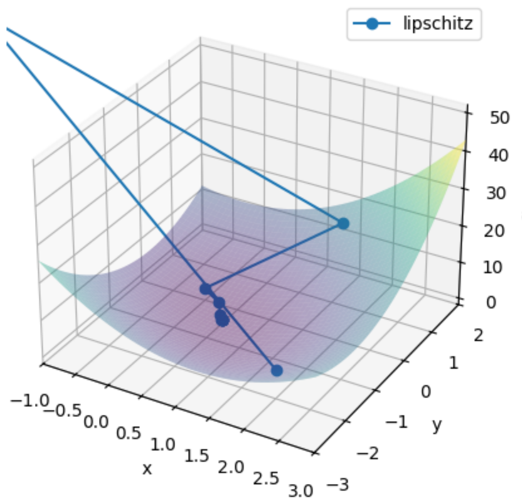

Рис. 18: 3D-график, lipschitz.

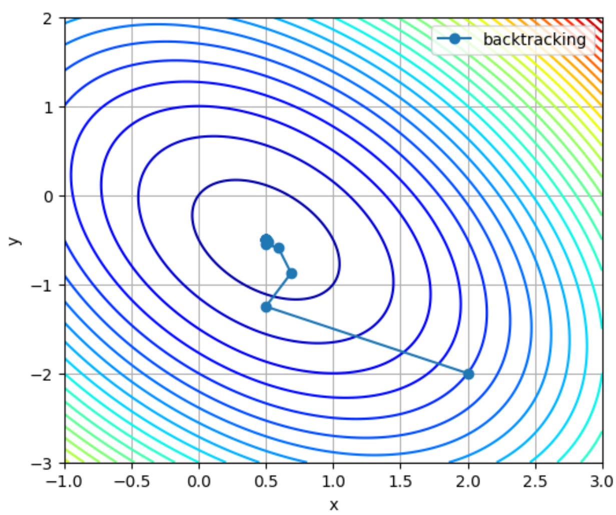

Рис. 19: Линии уровня, backtracking.

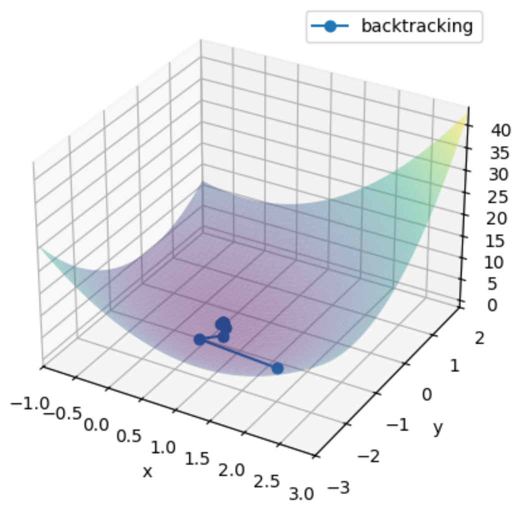

Рис. 20: 3D-график, backtracking.

Видно, что:

- constant идёт плавной дугой к минимуму,
- lipschitz может «выстрелить» вверх, если локальная оценка L_k мала.
- line_search и backtracking делают более прямолинейные шаги,

Для наглядности построим график логарифмической ошибки $\|x_k - x^*\|$ от номера итерации для всех четырёх стратегий (constant, line_search, lipschitz, backtracking).

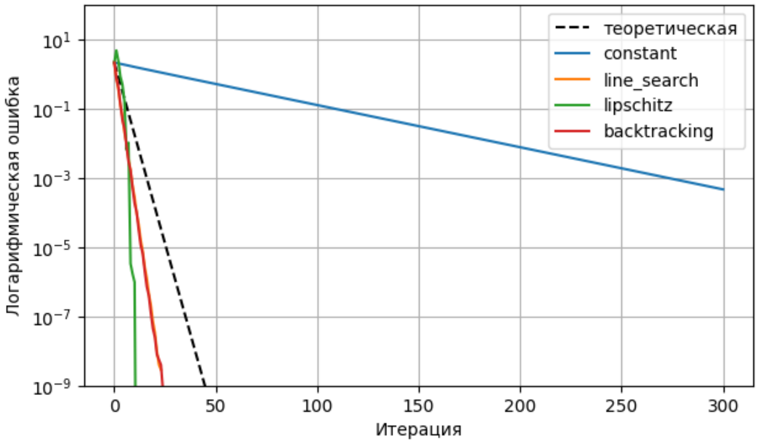

Рис. 21: Сходимость для квадратичной задачи, все стратегии.

Видно, что:

- constant — метод убывает примерно линейно, причём медленнее, чем теоретическая кривая $(1-\mu/L)^k$, если шаг не оптимален;
- line_search и lipschitz показывают более резкий спад ошибки на первых шагах;
- backtracking (Армихо) часто выходит на минимальную ошибку быстрее всех.

Теперь рассмотрим функцию Розенброка $f(x,y) = (1-x)^2 + 100\left(y-x^2\right)^2$, которая является классическим примером «неудобной» функции с узкой «долиной» вдоль параболы $y = x^2$. Мы вновь используем метод градиентного спуска в четырёх вариантах: constant, line_search, lipschitz, backtracking, и сравниваем траектории.

Далее приведены 2D-графики для каждой стратегии и та же функция в виде 3D-поверхности $z = f(x,y)$ и траектории $\{x_k\}$.

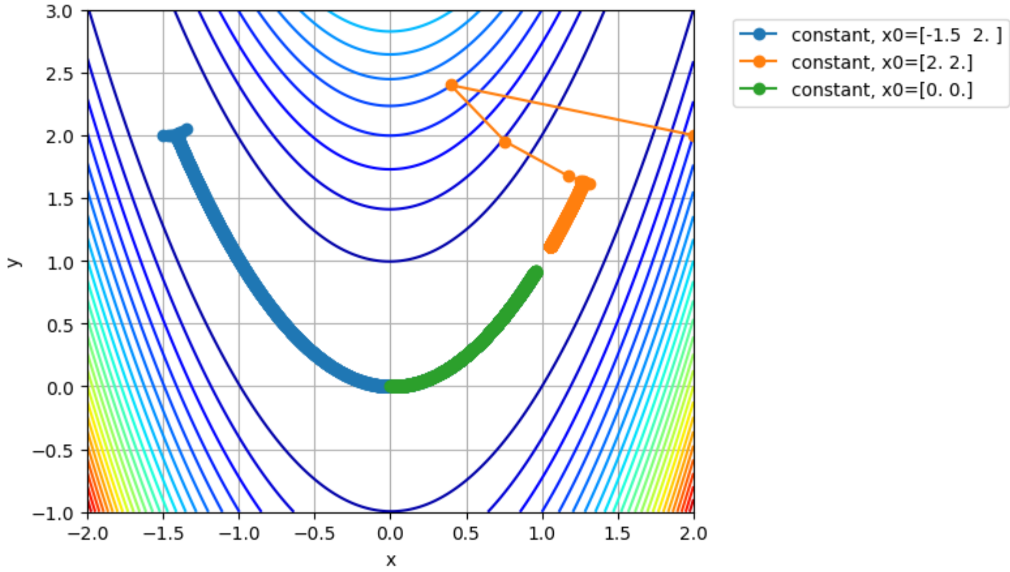

Рис. 22: Линии уровня, constant.

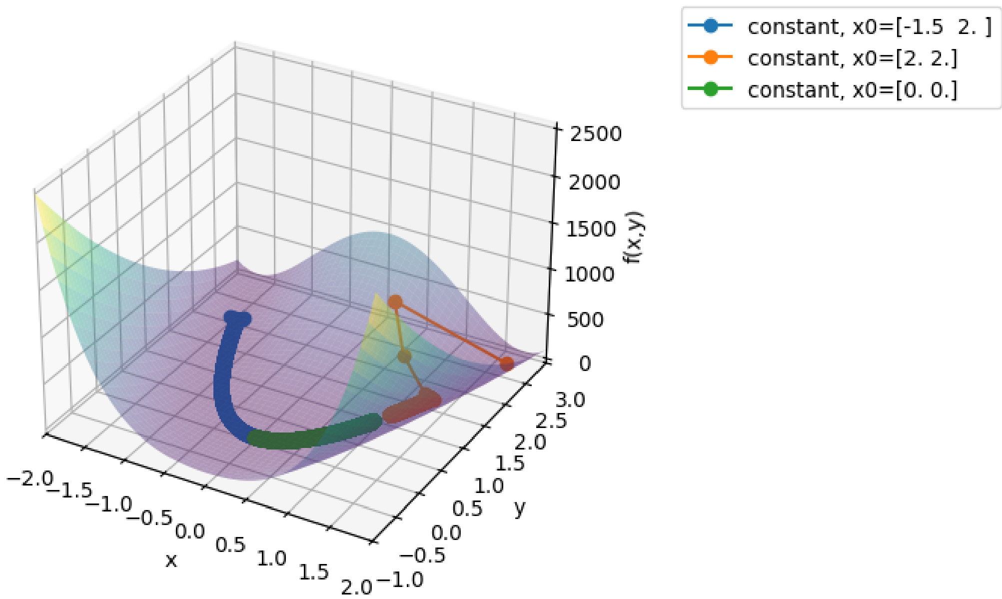

Рис. 23: 3D-график, constant.

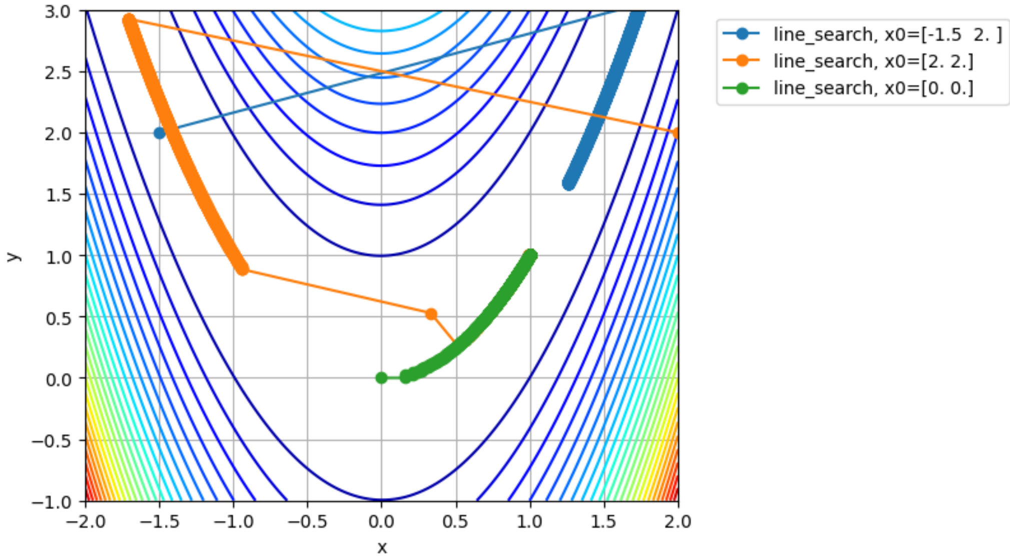

Рис. 24: Линии уровня, line_search.

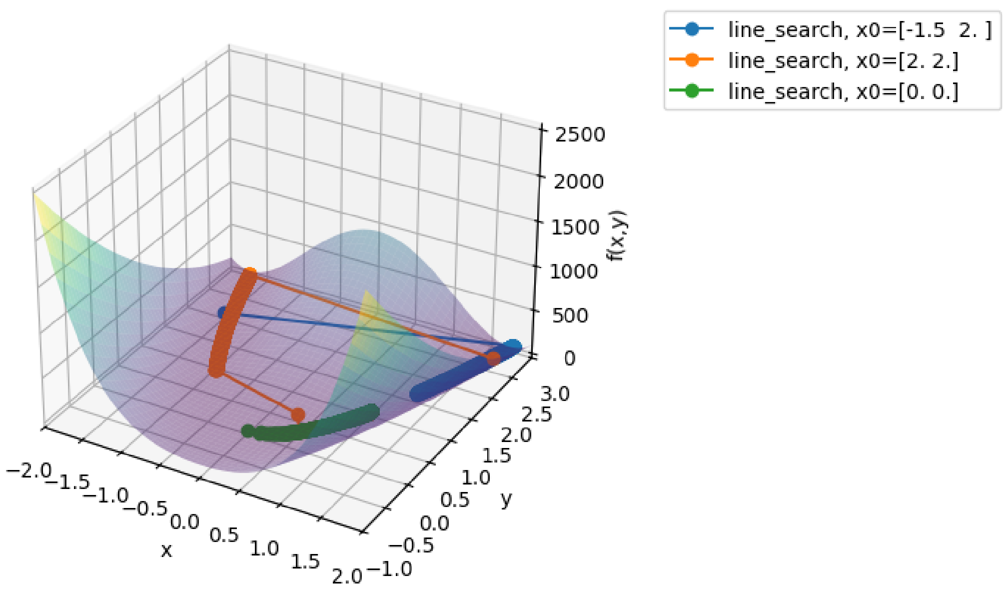

Рис. 25: 3D-график, line_search.

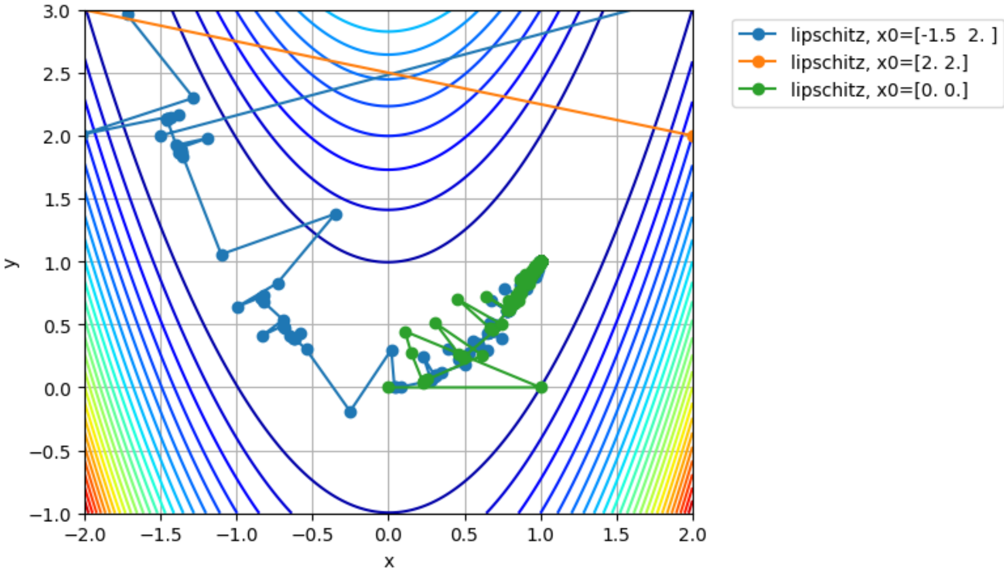

Рис. 26: Линии уровня, lipschitz.

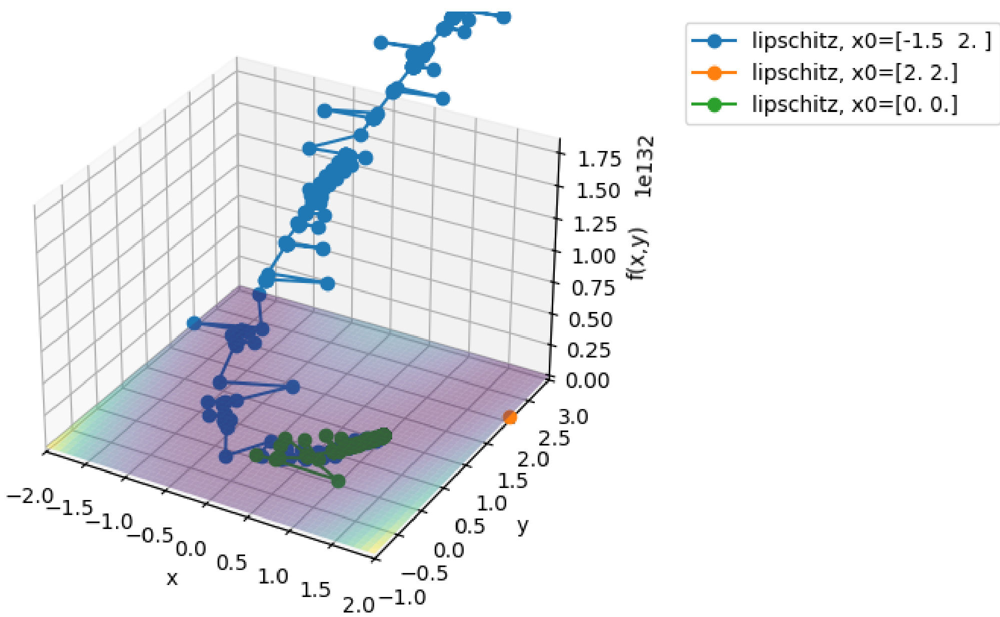

Рис. 27: 3D-график, lipschitz.

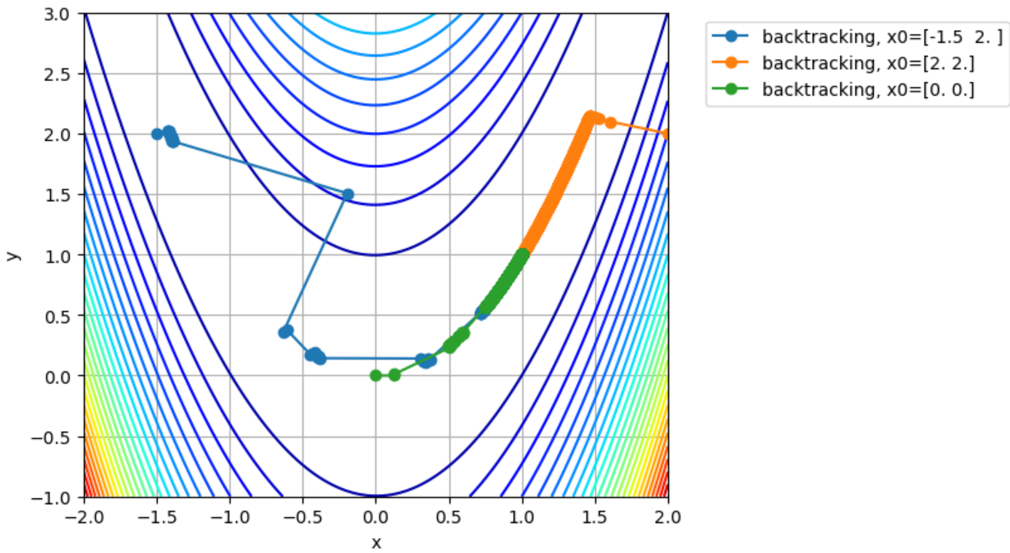

Рис. 28: Линии уровня, backtracking.

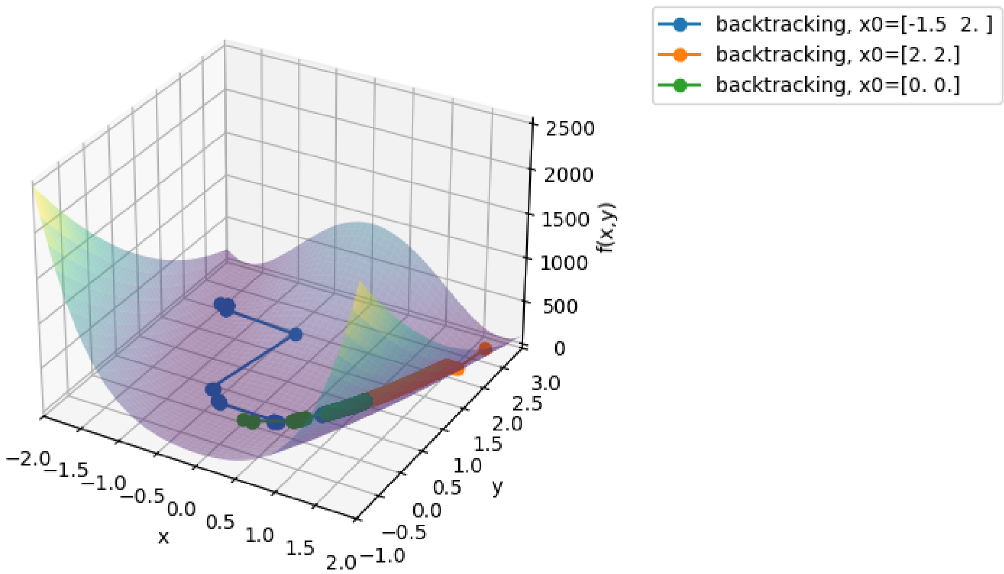

Рис. 29: 3D-график, backtracking.

Как видно из 2D-графиков, функция Розенброка имеет минимум в $(1, 1)$, но «долина» довольно узкая и «крутая».

- constant рисует плавную (иногда изогнутую) траекторию, поскольку шаг зафиксирован и не учитывает геометрию долины.
- line_search делает более прямолинейные отрезки, подбирая шаг через одномерную оптимизацию.
- lipschitz может «выстрелить» наверх, если локальная оценка L_k занижена, что приводит к очень большим значениям f на какой-то итерации.
- backtracking даёт «осторожные» шаги, сохраняя условие Армихо $f(x_{k+1}) \leq f(x_k) - c\,\alpha_k \|\nabla f(x_k)\|^2$, и в итоге часто даёт самую ровную сходимость.

Чтобы наглядно сравнить скорость сходимости различных стратегий градиентного спуска на функции Розенброка, построим графики зависимости $\|x_k - (1,1)\|$ от номера итерации k при трёх разных начальных приближениях.

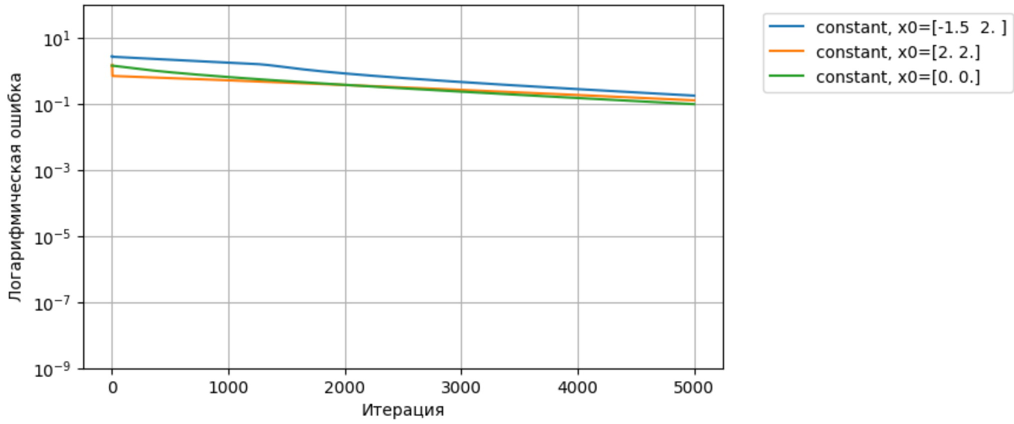

Рис. 30: Логарифмическая ошибка по x для constant, несколько x_0.

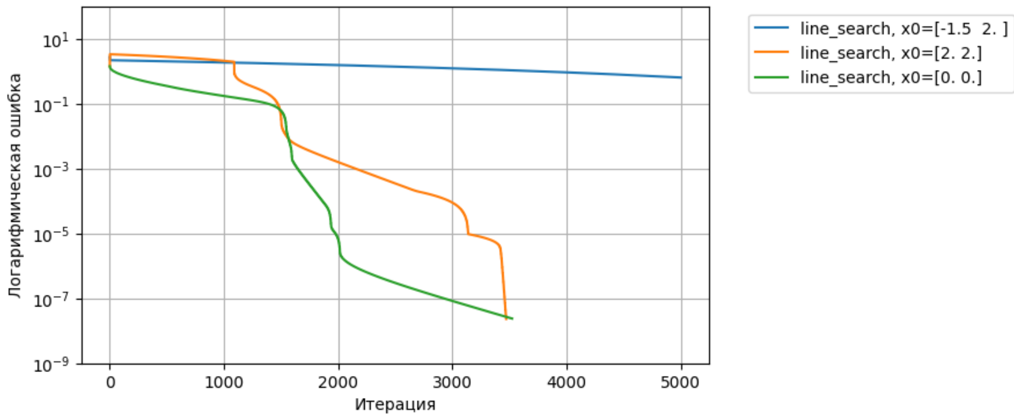

Рис. 31: Логарифмическая ошибка по x для line_search, несколько x_0.

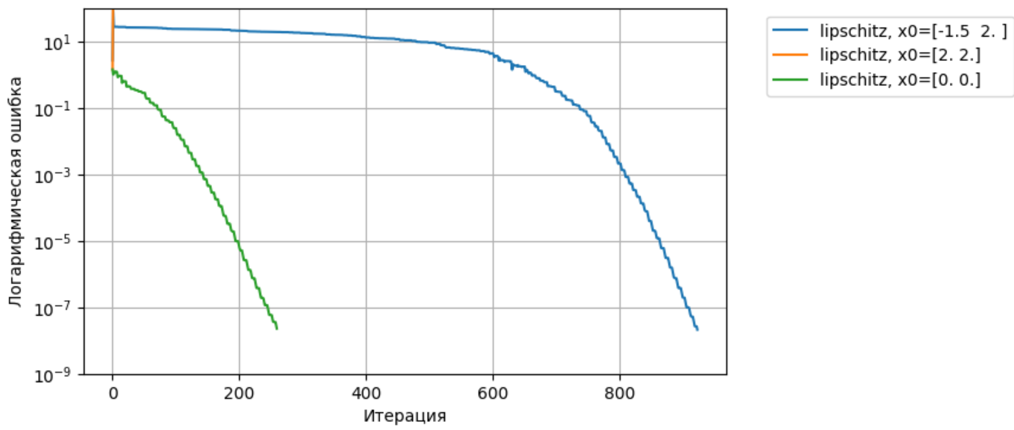

Рис. 32: Логарифмическая ошибка по x для lipschitz, несколько x_0.

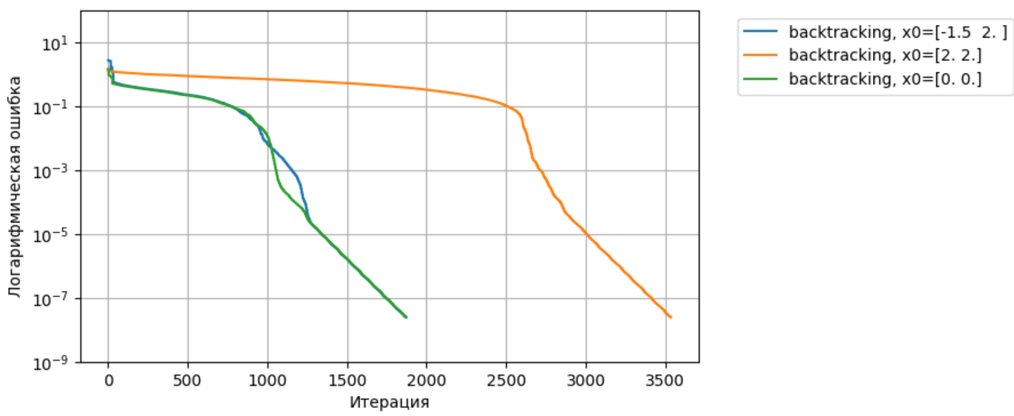

Рис. 33: Логарифмическая ошибка по x для backtracking, несколько x_0.

На всех четырёх графиках видно, что:

- Backtracking показывает довольно «плавную» убывание ошибки, однако при стартовой точке $(2, 2)$ метод расходует гораздо больше итераций (около 3000–3500), прежде чем достичь ошибки порядка 10^{-8}. Для ближних стартов $(0, 0)$ или $(-1.5, 2)$ кривая убывает быстрее.

- Constant сходится равномерно, но достаточно медленно (кривая примерно линейна по оси итераций); за 5000 итераций ошибка падает к 10^{-8}. При разных x_0 кривая смещается, но форма остаётся похожей.

- Line_search даёт «ступенчатое» поведение: например, при $(2, 2)$ ошибка долго «застывает» на уровне 10^{-1}, а затем быстро «проваливается» к 10^{-7}. При $(0, 0)$ сходится ещё быстрее.

- Lipschitz при удачном старте (зелёная кривая) даёт стремительное падение ошибки всего за 200–300 итераций; при неудачном (синяя кривая) метод сначала может «улетать» на огромные значения, но в итоге всё же «возвращается» и убывает до 10^{-9}.

3.2 Вывод

- Для квадратичной функции $f(x) = x^\top Ax + b^\top x$ все четыре стратегии (constant, line_search, lipschitz, backtracking) гарантированно находят минимум, причём line search и backtracking часто выходят на заданную точность заметно быстрее, чем constant. Lipschitz тоже может быть эффективен, если локальные оценки L_k не «врут» слишком сильно.

- Для функции Розенброка также все методы в итоге сходятся к глобальному минимуму $(1, 1)$, но:

  – line_search и backtracking в большинстве случаев дают стабильную сходимость без резких скачков;

  – lipschitz может «выпрыгивать» на большие значения, если L_k оказывается заниженным, особенно при далёком x_0;

  – constant требует подбора α, иначе метод «ползёт» очень долго или начинает «болтаться».

- В обоих примерах backtracking часто даёт наиболее «гладкую» кривую убывания ошибки, хотя иногда требует дополнительных итераций, если стартовая точка далека.

Таким образом, все четыре стратегии работоспособны, но скорость и стабильность зависят от свойств задачи (выпуклость, глобальная/локальная липшицевость) и от начального приближения.

4 Построение траектории

4.1 Постановка задачи и её разрешение

В данном задании мы попробуем применить метод градиентного спуска для решения практической задачи. Нам необходимо, в лице робота, пройти из точки A в B, учитывая препятствия.

Для построения нашей "полосы препятствий"мы восползуемся потенциальной функцией вида:

$$F(x,y) = a((x-x_b)^2 + (y-y_b)^2) + \sum_1^n b_i exp(-\frac{(x-x_{p_i})^2}{2\sigma_{x_i}} - \frac{(y-y_{p_i})^2}{2\sigma_{y_i}})$$

Данная функция, по своей сути, строит квадратичную функцию с точкой минимума в (x_b, y_b) и добавляет гауссовы колокола вокруг. В общем то, из-за того, что эти колокола вне своих "радиусов"очень быстро приближаются к нулю, прибавления таких функций глобально не повлияет на расположение минимума.
Рассмотрим наши параметры для этой функции:

- $(x_b, y_b) = (6, 4)$
- $a = 0.1$ -коэффициент квадратичной функции. Взяли его таким маленьким, потому что при больших значениях он очень быстро перекрывает колокол
- Высота колоколов, за которые отвечают коэффициенты b_i варьируется в интервале от 20 до 60
- "Радиусы"колоколов все равны 0.8.
- И, конечно, точки - центры колоколов:(3,2),(-3,-2),(3,8),(3,-2)

Для данной функции воспользуемся довольно простым методом - будем подбирать коэффициент перед градиентом с помощью оптимизации функции одной переменной.

Посмотрим на то, какой получился траектория(для начальной точки(-6,-6)):

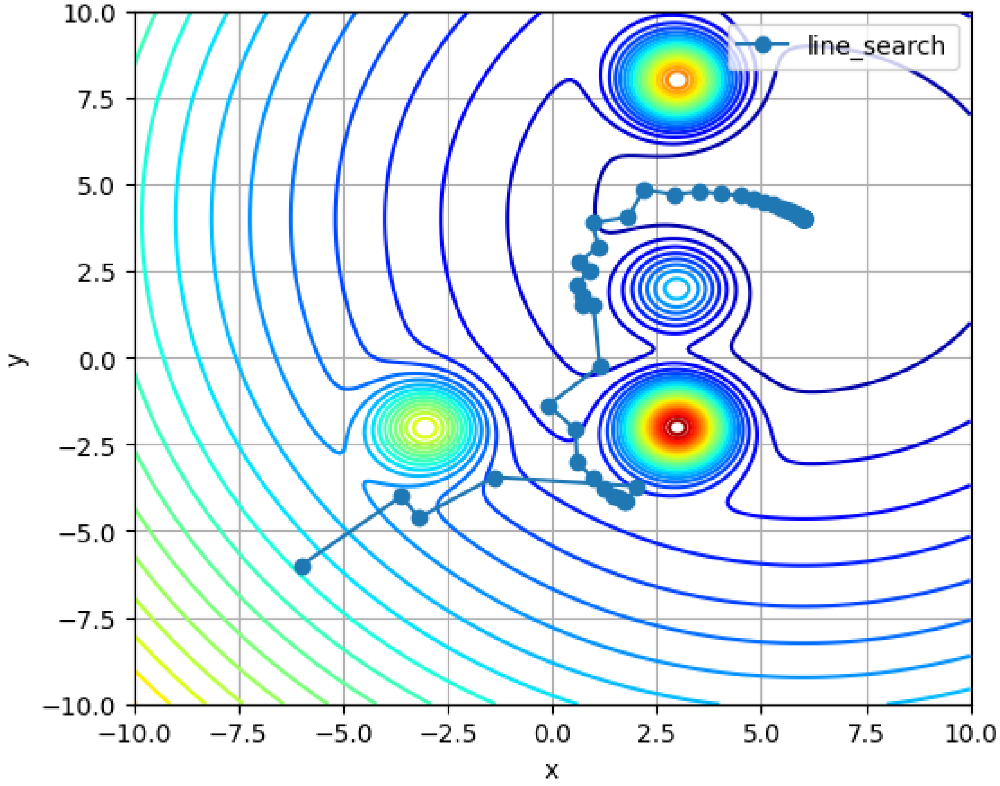

Рис. 34: 2D-график, line_search.

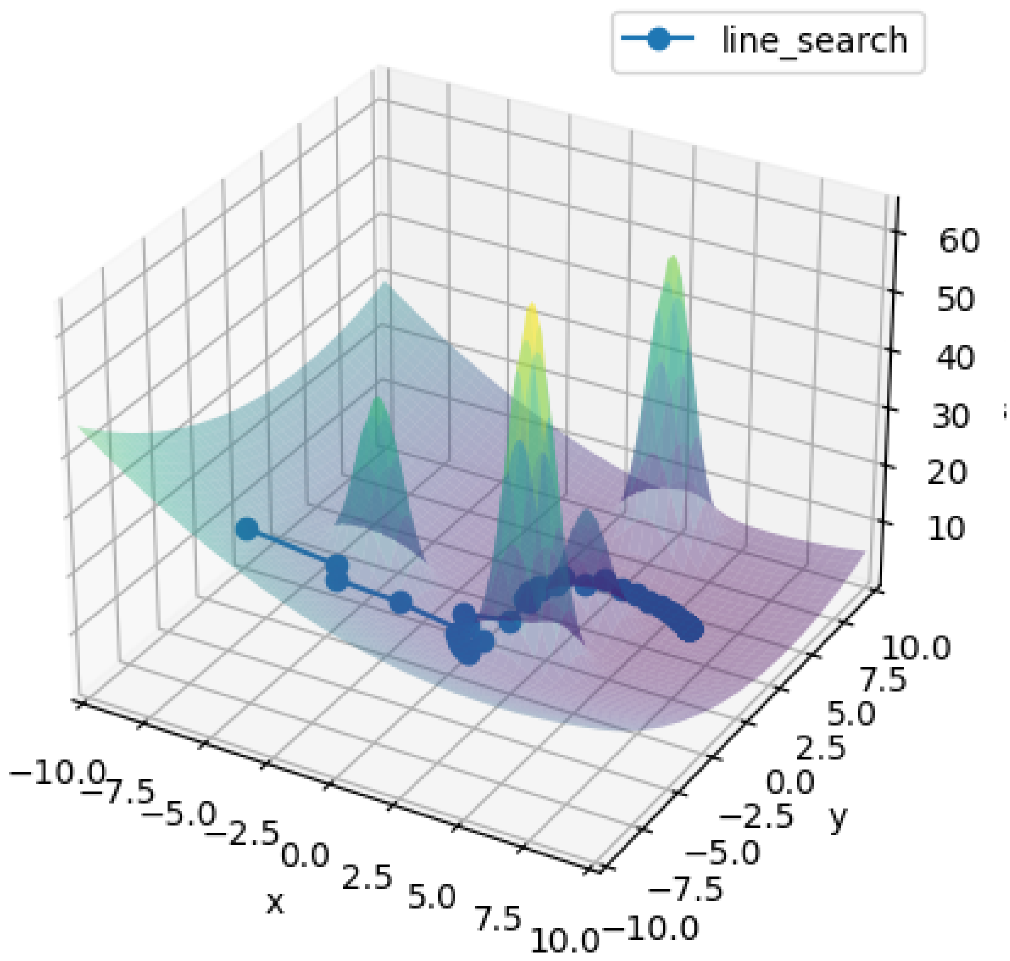

Рис. 35: 3D-график, line_search.

Как можно наблюдать - метод справился! Полученная точка $\approx (6.01, 4.01)$. Небольшая ошибка возникла по двум причинам: ошибка численного вычисления градиента и очень маленькое значение градиента вблизи точки минимума. А ведь критерий для остановки нашего алгоритма завязан на модуле градиента... В общем, причина ошибки достаточно понятна.

Давайте также обратим внимание на траекторию - она получилась достаточно ровной, без сильных перелётов. Возможно, настоящий робот даже сможет по ней проехать.

Стоит сделать небольшое признание. Хотя для этой конкретной конфигурации наш метод линейного поиска и справился, но стоит понимать, что, к примеру, если даже немного сдвинуть колокол в точке (3,8) ближе к центру координат, то наш метод застрянет между двух колоколов и не спустится в точку минимума.

Немного про полученную траекторию. В данном случае можно наблюдать, что в дали от точки минимума(когда градиент значительный) наш метод довольно хорошо проскакивает к условно минимальным точкам, может сделать достаточно значительнео перемешение за 2-3 шага. Но вот приближившись к минимуму происходит быстрое уменьшение шага. Также достаточно интересным является то, что наш метод в какой то момент пытается практически упереться в вершину холма. Такое поведения могло бы оказаться фатальным для сходимости метода, тем не менее он довольно хорошо вышел из ситуации.

4.2 Вывод

В данном разделе нам удалось применить полученные знания и методы для решения практической задачи. Как оказалось, метод градиентного спуска можно применять довольно креативно - не только искать минимумы разлиных функций, но и использовать данные об траектории метода, чтобы самим строить траекторию робота. Также выяснилось, что для решения этой задачи может подойти достаточна простая стратегия для выбора коэффициента в градиентном методе, хотя и условия для использования должны быть в каком-то смыле "стерильными".