ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Университет ИТМО

Отчёт по лабораторной работе № 3

«Численные методы оптимизации: нахождение минимума функции Розенброка»

Выполнил работу:

Демьянов Фёдор Александрович

Академическая группа: № J3114

Санкт-Петербург 2025

Цель лабораторной работы

Цель данной лабораторной работы — освоить численные методы оптимизации на примере задачи минимизации функции двух переменных, продемонстрировать преимущества и особенности различных подходов и оценить их эффективность на практике. В качестве тестовой функции выбрана функция Розенброка.

Задачи

1. Изучить теоретические основы и алгоритмы следующих методов оптимизации:
    - метод Ньютона;
    - метод SR1;
    - метод тяжёлого шарика.
2. Реализовать указанные методы в программной среде.
3. Для каждого метода реализовать подбор шага с помощью метода золотого сечения.
4. Визуализировать траектории сходимости и поведение значений целевой функции по итерациям.
5. Сравнить методы по скорости сходимости, количеству итераций и точности решения.

Теоретическая часть

Функция Розенброка

Функция Розенброка — это классическая функция двух переменных, которая часто используется в тестировании численных методов оптимизации:

$$f(x, y) = 100(y - x^2)^2 + (1 - x)^2$$

Функция имеет единственный минимум в точке $\left(f(1{,}1)\right)$, где значение функции равно нулю. Её особенность заключается в узкой вытянутой долине, что затрудняет сходимость градиентных методов и делает её хорошим тестом для сравнительного анализа.

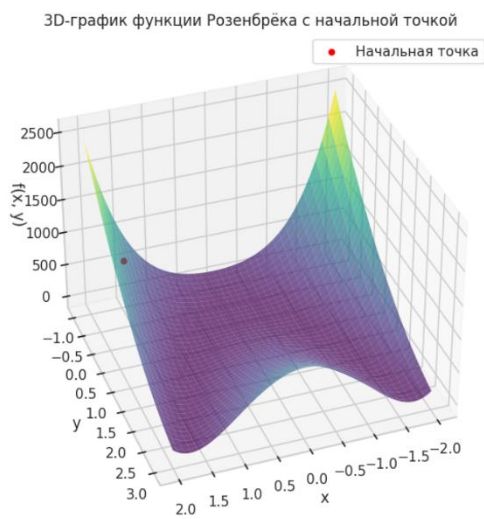

График 1. Поверхность функции Розенброка

На рисунке изображена 3D-поверхность функции Розенброка с отмеченной начальной точкой.

Градиент — вектор, содержащий частные производные функции по каждой переменной. Он показывает направление наискорейшего роста функции. В оптимизации движение производится в направлении, противоположном градиенту.

$$\nabla f(x, y) = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]^T$$

Градиент функции Розенброка:

$$\nabla f(x, y) = \begin{bmatrix} -400x(y - x^2) - 2(1 - x) \\ 200(y - x^2) \end{bmatrix}$$

Гессиан — матрица вторых производных функции. Она описывает локальную кривизну функции и необходима в методах второго порядка:

$$\nabla^2 f(x,y) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix}$$

Гессиан функции Розенброка:

$$\nabla^2 f(x,y) = \begin{bmatrix} 1200x^2 - 400y + 2 & -400x \\ -400x & 200 \end{bmatrix}$$

Критерии остановки

$$|\nabla f(x_k)| < \varepsilon \quad \text{или} \quad |f(x_k) - f(x^*)| < 0.001$$

Используемые методы

Метод Ньютона

Метод Ньютона относится к методам второго порядка и использует как градиент, так и гессиан функции. Алгоритм позволяет достичь квадратичной сходимости в окрестности минимума.

Итерационная формула: $x_{k+1} = x_k - [\nabla^2 f(x_k)]^{-1} \nabla f(x_k)$

Алгорим:

1. Задать начальную точку (x_0), точность (ε), максимум итераций (N).

2. Для ($k = 0, 1, 2, \dots$):

- Вычислить градиент: ($\nabla f(x_k)$)
- Вычислить гессиан: ($H_k = \nabla^2 f(x_k)$)
- Решить систему: ($H_k d_k = \nabla f(x_k)$)
- Найти оптимальный шаг (α_k) с помощью метода золотого сечения:

$$\phi(\alpha) = f(x_k - \alpha d_k), \quad \alpha_k = \arg\min \phi(\alpha)$$

- Обновить приближение:

$$x_{k+1} = x_k - \alpha_k d_k$$

- Если ($|\nabla f(x_{k+1})| < \varepsilon$), остановиться.

Код:

```python
def newton_method_rosenbrock(x0, y0, eps=1e-6, max_iter=1000):
    # Начальные координаты
    x, y = x0, y0
    # Список точек траектории
    path = [(x, y)]
    # Счетчик количества итераций
    it = 0

    for _ in range(max_iter):
        it += 1
        # Вычисляем градиент и гессиан в текущей точке
        grad = rosen_grad(x, y)
        H = rosen_hess(x, y)

        try:
            # Решаем систему H·d = grad, получаем направление d
            d = np.linalg.solve(H, grad)
        except np.linalg.LinAlgError:
            # Если Гессиан вырожден (невозможно обратить) — останавливаем
            print("Гессиан не обратим, шаг пропущен.")
            break

        # Определяем функцию от шага alpha
        phi = lambda a: rosenbrock(*(np.array([x, y]) - a * d))

        # Используем метод золотого сечения для поиска оптимального шага
        alpha = golden_section_search(phi, a = 0, b = 0.1)

        # Делаем шаг
        x, y = np.array([x, y]) - alpha * d

        # Добавляем новую точку в путь
        path.append((x, y))

        # Проверка условия сходимости: малый градиент
        if np.linalg.norm(grad) < eps:
            break

    # Возвращаем массив всех точек пути и финальные координаты
    return np.array(path), (x, y), it
```

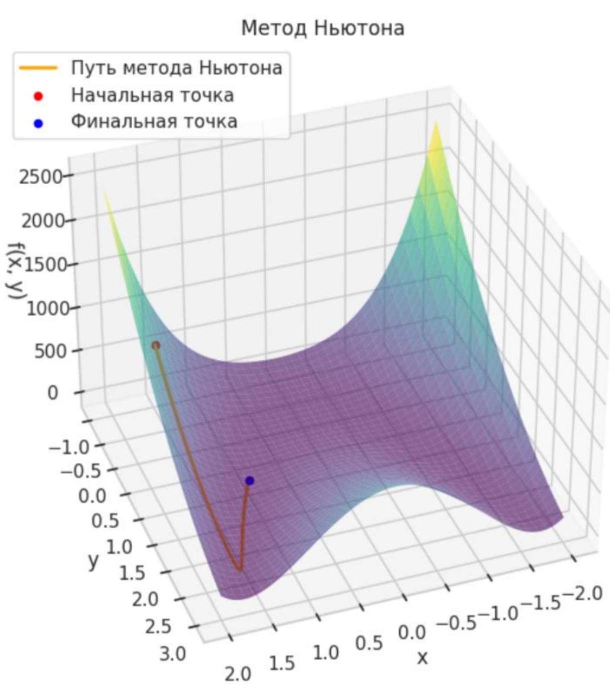

График 2. Траектория метода Ньютона.

Анализ:

Благодаря использованию гессиана метод быстро и точно направляется к минимуму. Уже за малое количество итераций он входит в долину и доходит до точки (1, 1).

Метод SR1 (Symmetric Rank-1)

Метод SR1 — один из квазиньютоновских методов. Он использует приближение обратной матрицы Гессе и обновляет его по информации о шагах и изменениях градиента:

$$B_{k+1} = B_k + \frac{(s_k - B_k y_k)(s_k - B_k y_k)^T}{(s_k - B_k y_k)^T y_k}$$

где $(s_k = x_{k+1} - x_k), \big(y_k = f(x_{k+1}) - f(x_k)\big)$.

Преимуществом метода является отсутствие необходимости прямого расчёта гессиана и относительная устойчивость к шуму в данных.

Алгоритм:

1. Задать начальную точку (x_0), начальное приближение $(B_0 = I)$, точность (ε), максимум итераций (N).

2. Для ($k \ = \ 0, 1, 2, \dots$):

- Вычислить градиент: $(g_k = \nabla f(x_k))$

- Найти направление: $(d_k = -B_k g_k)$

- Найти шаг (α_k) методом золотого сечения:

$$\phi(\alpha) = f(x_k + \alpha d_k), \quad \alpha_k = \arg\min \phi\,(\alpha)$$

- Обновить точку

$$\mathrm{x_{k+1}\ =\ x_k\ +\ \alpha_k\, d_k}$$

- Вычислить:

$$s_k = x_{k+1} - x_k, \quad y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$$

- Обновить матрицу (B_k):

$$B_{k+1} = B_k + \frac{(s_k - B_k y_k)(s_k - B_k y_k)^T}{(s_k - B_k y_k)^T y_k}$$

- Если$(|\nabla f(x_{k+1})| < \varepsilon)$, остановиться.

Код:

```
def sr1_method_rosenbrock(x0, y0, eps=1e-6, max_iter=200000):

    # Счетчик количества итераций
    it = 0
    # Начальные координаты
    x, y = x0, y0
    # Начальное приближение для обратного Гессиана (единичная матрица)
    B_inv = np.eye(2)
    # Список точек траектории
    path = [(x, y)]
    for _ in range(max_iter):
        it += 1
        # Вычисляем градиент в текущей точке
        grad = rosen_grad(x, y)
        # Направление спуска
        d = -B_inv @ grad
        # Сохраняем текущие значения для обновления
        x_prev, y_prev = x, y
        grad_prev = grad.copy()
        # Определяем функцию от шага alpha
        phi = lambda a: rosenbrock(*(np.array([x, y]) + a * d))
        # Используем метод золотого сечения для поиска оптимального шага
        alpha = golden_section_search(phi, a = 0, b = 0.1)
        # Делаем шаг
        x, y = np.array([x, y]) + alpha * d
        # Вычисляем новый градиент
        grad_new = rosen_grad(x, y)
        # Разница градиентов
        y_k = grad_new - grad_prev
        # Разница точек
        s_k = np.array([x - x_prev, y - y_prev])
        # Вспомогательный вектор
        v_k = s_k - B_inv @ y_k
        # Обновление B_inv по формуле SR1, если условие выполняется
        if np.abs(v_k @ y_k) > eps * np.linalg.norm(v_k) * np.linalg.norm(y_k):
            B_inv += np.outer(v_k, v_k) / (v_k @ y_k)
        # Добавляем новую точку в путь
        path.append((x, y))
        # Проверка условия сходимост
        if np.linalg.norm(grad_new) < eps:
            break
    # Возвращаем массив всех точек пути и финальные координаты
    return np.array(path), (x, y), it
```

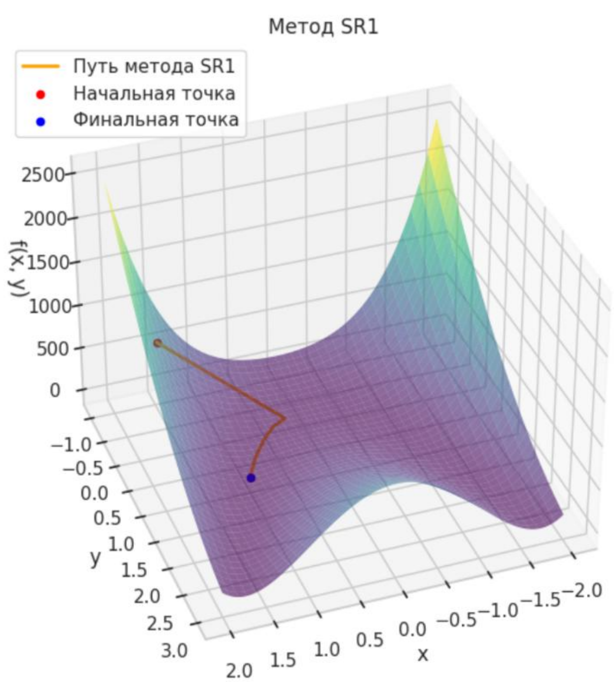

График 3. Траектория метода SR1

Анализ: Метод SR1 сходится медленнее, поскольку использует приближённый гессиан. Однако траектория стабильна и последовательна, метод не делает резких скачков.

Метод тяжёлого шарика (Heavy Ball)

Метод моделирует движение точки как движение тела с массой, учитывая ускорение, вызванное градиентом, и инерцию предыдущего движения:

$$\begin{cases} v_{k+1} = \beta v_k - \alpha \nabla f(x_k) \\ x_{k+1} = x_k + v_{k+1} \end{cases}$$

Метод обладает высокой скоростью сходимости, но чувствителен к выбору параметров. Для стабилизации используется подбор шага методом золотого сечения.

Алгоритм:

1. Задать начальную точку (x_0), начальную скорость $(v_0 = 0)$, параметры $\alpha,\ \beta$, точность (ε), максимум итераций (N).

2. Для $(k = 0, 1, 2, \dots)$:

- Вычислить градиент: $(g_k = \nabla f(x_k))$
- Найти оптимальный шаг (α_k) методом золотого сечения:

$$\phi(\alpha) = f(x_k - \alpha g_k), \quad \alpha_k = \arg\min \phi(\alpha)$$

- Обновить скорость:

$$v_{k+1} = \beta v_k - \alpha_k g_k$$

- Обновить позицию

$$x_{k+1} = x_k + v_{k+1}$$

- Если $(|g_{k+1}| < \varepsilon)$, остановиться.

Код:

```
def heavy_ball_method(x0, y0, beta=0.7, max_iter=1000, eps=1e-6):
    # Начальная инициализация координат
    x, y = x0, y0
    # Начальная скорость
    v = np.array([0.0, 0.0])
    # Сохраняем путь
    path = [(x, y)]
    # Счётчик итераций
    it = 0
    # Основной цикл
    for _ in range(max_iter):
        # Увеличиваем счётчик итераций
        it += 1
        # Вычисляем градиент функции в текущей точке
        grad = rosen_grad(x, y)
        # Определяем функцию от шага alpha вдоль направления -grad
        phi = lambda a: rosenbrock(*(np.array([x, y]) - a * grad))
        # Используем метод золотого сечения для поиска оптимального шага
        alpha = golden_section_search(phi, a = 0, b = 0.1)
        # Обновляем скорость с учётом ускорения
        v = beta * v - alpha * grad
        # Обновляем координаты с учётом текущей скорости
        x, y = x + v
        # Добавляем новую точку в путь
        path.append((x, y))
        # Проверка на сходимость по норме градиента
        if np.linalg.norm(grad) < eps:
            break
    # Финальные значения
    xf, yf = path[-1]  # последняя точка траектории
    zf = rosenbrock(xf, yf)  # значение функции в ней

    # Возвращаем:
    # - массив точек траектории
    # - финальную координату (xf, yf)
    # - значение функции в финальной точке
    # - количество итераций
    return np.array(path), (xf, yf), zf, it
```

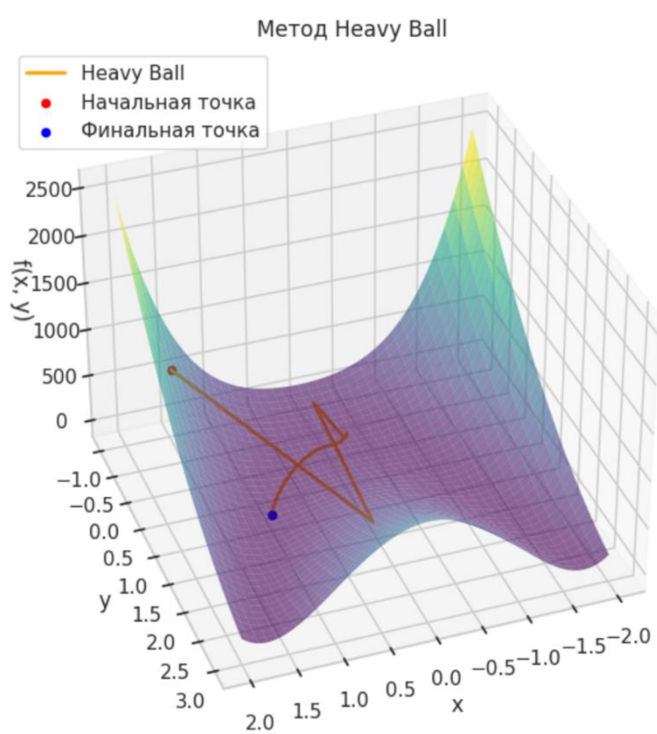

График 4. Траектория метода тяжёлого шарика

Анализ: Метод демонстрирует быструю сходимость, особенно на начальных этапах. Благодаря инерции он быстро движется по направлению к долине, но оставляет резкий и не самый оптимальный путь. Метод использует слишком большую скорость на первых шагах.

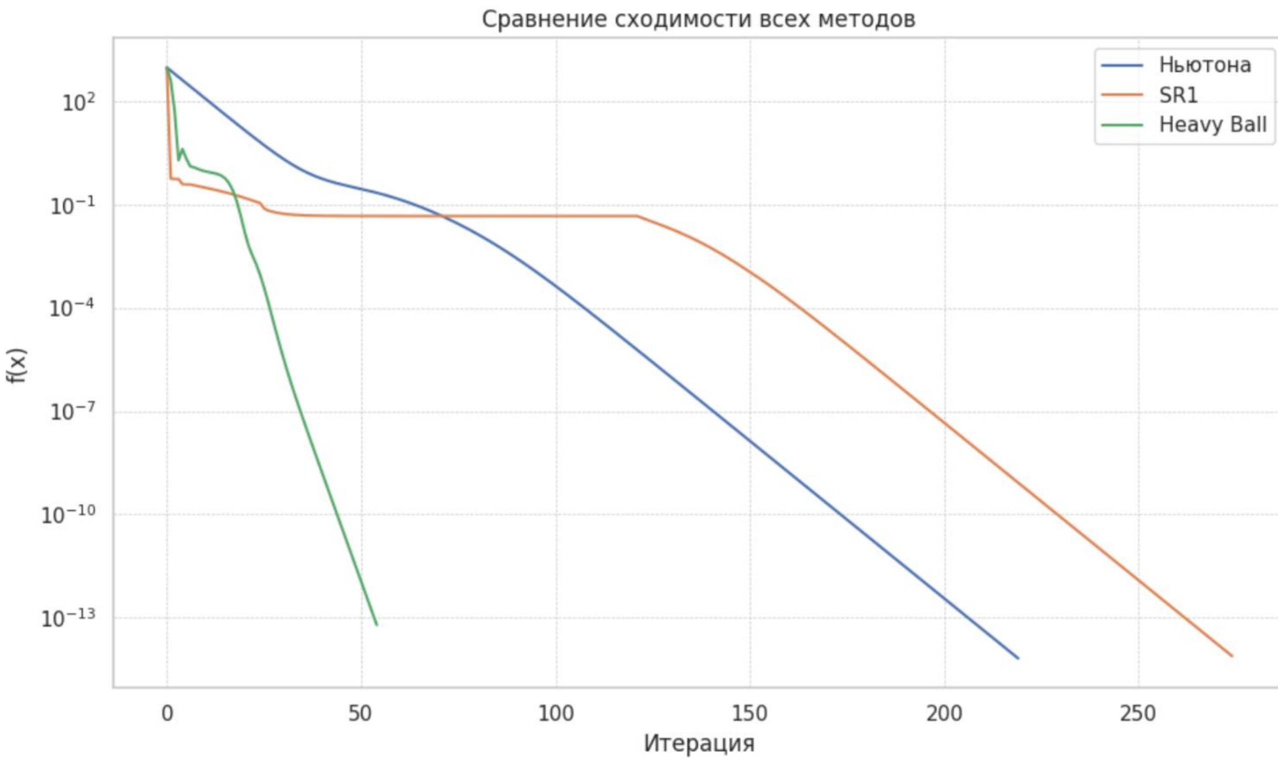

График 5. Логарифмическая сходимость значений функции

Анализ:

На графике видно, что метод шарика достигает экстремально малых значений функции за минимальное количество итераций. Метод Ньютона сходится медленнее, но стабильно. SR1 после 125 итераций начинает стабильно приближаться к минимальным значениям.

	Метод	Итерации	f(x*, y*)	Условие точности (<0.001)
0	SR1	274	7.693286e-15	True
1	Ньютона	219	6.548590e-15	True
2	Heavy Ball	54	6.129567e-14	True

График 6. Таблица сравнения методов

Анализ: Сравнение по количеству итераций и финальному значению функции показывает, что:

- Метод Ньютона — быстрый и точный;
- Heavy Ball — самый быстрый по итерациям, но менее точный;
- SR1 — надёжный и стабильный, хотя и требует больше шагов.

Выводы

В ходе лабораторной работы были изучены и реализованы три численных метода оптимизации: метод Ньютона, квазиньютоновский метод SR1 и метод тяжёлого шарика. Все методы были протестированы на функции Розенброка — классическом примере задачи оптимизации с узкой долиной и одним минимумом.

Метод Ньютона продемонстрировал высокую скорость и точность сходимости, благодаря использованию информации о кривизне функции через гессиан. Однако он требует вычисления и обращения матрицы Гессе, что увеличивает вычислительную сложность.

Метод SR1, как квазиньютоновский подход, позволил отказаться от прямого вычисления гессиана, сохранив при этом разумную сходимость. Он показал стабильную и надёжную работу, хотя и уступал другим методам по скорости.

Метод тяжёлого шарика оказался наиболее быстрым по количеству итераций, особенно в начале оптимизации, но потребовал аккуратной настройки параметров для предотвращения колебаний и излишнего «перескакивания» минимума.

Применение метода золотого сечения для подбора шага в каждом методе улучшило их устойчивость и позволило избежать необходимости ручного задания параметров.

В результате все три метода успешно сошлись к минимуму функции с заданной точностью.

Приложения

Код в colab: NM_lab_3