

# Матрично-векторное дифференцирование. Линейный поиск

Даня Меркулов

## 1 Матрично-векторное дифференцирование

### 1.1 Градиент

Пусть  $f(x) : \mathbb{R}^n \rightarrow \mathbb{R}$ , тогда вектор, который содержит все первые частные производные:

$$\nabla f(x) = \frac{df}{dx} = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}$$

называется градиентом функции  $f(x)$ . Этот вектор указывает направление наискорейшего возрастания. Таким образом, вектор  $-\nabla f(x)$  указывает направление наискорейшего убывания функции в точке. Кроме того, вектор градиента всегда ортогонален линии уровня в точке.

#### Example

Для функции  $f(x, y) = x^2 + y^2$  градиент равен:

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

Он указывает направление наискорейшего возрастания функции.

#### Question

Как связана норма градиента с крутизной функции?

### 1.2 Гессиан

Пусть  $f(x) : \mathbb{R}^n \rightarrow \mathbb{R}$ , тогда матрица, содержащая все вторые частные производные:

$$f''(x) = \nabla^2 f(x) = \frac{\partial^2 f}{\partial x_i \partial x_j} = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_n} \end{pmatrix}$$

Гессиан может быть тензором:  $(f(x) : \mathbb{R}^n \rightarrow \mathbb{R}^m)$  Таким образом, это просто трехмерный тензор, каждый срез которого это гессиан соответствующей скалярной функции  $(\nabla^2 f_1(x), \dots, \nabla^2 f_m(x))$ .

### i Example

Для функции  $f(x, y) = x^2 + y^2$  гессиан равен:

$$H_f(x, y) = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

Эта матрица содержит информацию о кривизне функции в разных направлениях.

### i Question

Как можно использовать гессиан для определения выпуклости или вогнутости функции?

## 1.3 Теорема Шварца

Пусть  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  - функция. Если смешанные частные производные  $\frac{\partial^2 f}{\partial x_i \partial x_j}$  и  $\frac{\partial^2 f}{\partial x_j \partial x_i}$  непрерывны на открытом множестве, содержащем точку  $a$ , то они равны в точке  $a$ . То есть,

$$\frac{\partial^2 f}{\partial x_i \partial x_j}(a) = \frac{\partial^2 f}{\partial x_j \partial x_i}(a)$$

Согласно данной теореме, если смешанные частные производные непрерывны на открытом множестве, то гессиан симметричен. То есть,

$$\frac{\partial^2 f}{\partial x_i \partial x_j} = \frac{\partial^2 f}{\partial x_j \partial x_i} \quad \nabla^2 f(x) = (\nabla^2 f(x))^T$$

Эта симметричность упрощает вычисления и анализ, связанные с гессианом в различных приложениях, особенно в оптимизации.

### i Контрпример Шварца

$$f(x, y) = \begin{cases} \frac{xy(x^2 - y^2)}{x^2 + y^2} & \text{для } (x, y) \neq (0, 0), \\ 0 & \text{для } (x, y) = (0, 0). \end{cases}$$



Можно проверить, что  $\frac{\partial^2 f}{\partial x \partial y}(0,0) \neq \frac{\partial^2 f}{\partial y \partial x}(0,0)$ , хотя смешанные частные производные существуют, и в каждой другой точке симметричность выполняется.

## 1.4 Якобиан

Обобщением понятия градиента на случай многомерной функции  $f(x) : \mathbb{R}^n \rightarrow \mathbb{R}^m$  является следующая матрица:

$$J_f = f'(x) = \frac{df}{dx^T} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_2}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_1} \\ \frac{\partial f_1}{\partial x_2} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_1}{\partial x_n} & \frac{\partial f_2}{\partial x_n} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix}$$

Она содержит информацию о скорости изменения функции по отношению к ее входу.

### i Question

Можно ли связать эти три определения выше (градиент, якобиан, и гессиан) с помощью одного утверждения?

### i Example

Для функции

$$f(x, y) = \begin{bmatrix} x + y \\ x - y \end{bmatrix},$$

Якобиан равен:

$$J_f(x, y) = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

### Question

Как матрица Якоби связана с градиентом для скалярных функций?

## 1.5 Итог

$$f(x) : X \rightarrow Y; \quad \frac{\partial f(x)}{\partial x} \in G$$

| X                         | Y              | G                         | Name  |
|---------------------------|----------------|---------------------------|---|
| $\mathbb{R}$              | $\mathbb{R}$   | $\mathbb{R}$              | $f'(x)$ (производная)                         |
| $\mathbb{R}^n$            | $\mathbb{R}$   | $\mathbb{R}^n$            | $\frac{\partial f}{\partial x_i}$ (градиент)  |
| $\mathbb{R}^n$            | $\mathbb{R}^m$ | $\mathbb{R}^{n \times m}$ | $\frac{\partial f_i}{\partial x_j}$ (якобиан) |
| $\mathbb{R}^{m \times n}$ | $\mathbb{R}$   | $\mathbb{R}^{m \times n}$ | $\frac{\partial f}{\partial x_{ij}}$          |

## 1.6 Аппроксимация Тейлора первого порядка

Аппроксимация Тейлора первого порядка, также известная как линейное приближение, строится вблизи некоторой точки  $x_0$ . Если  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  - дифференцируемая функция, то ее аппроксимация первого порядка задается следующим образом:

$$f_{x_0}^I(x) = f(x_0) + \nabla f(x_0)^T(x - x_0)$$

где:

- $f(x_0)$  - значение функции в точке  $x_0$ .
- $\nabla f(x_0)$  - градиент функции в точке  $x_0$ .

Часто для упрощения теоретического анализа в некоторых методах заменяют функцию вблизи некоторой точки на её аппроксимацию

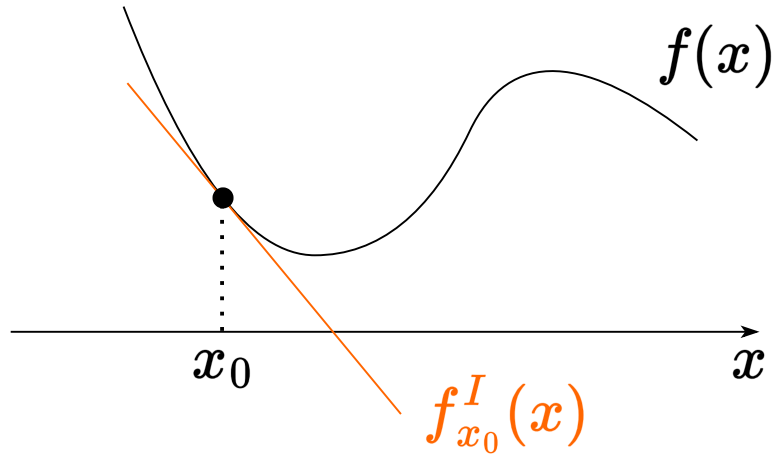


Рисунок 1: Аппроксимация Тейлора первого порядка в окрестности точки  $x_0$

## 1.7 Аппроксимация Тейлора второго порядка

Аппроксимация Тейлора второго порядка, также известная как квадратичное приближение, использует информацию о кривизне функции. Для дважды дифференцируемой функции  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ , ее аппроксимация второго порядка, строящаяся вблизи некоторой точки  $x_0$ , задается следующим образом:

$$f_{x_0}^{II}(x) = f(x_0) + \nabla f(x_0)^T(x - x_0) + \frac{1}{2}(x - x_0)^T \nabla^2 f(x_0)(x - x_0)$$

Где  $\nabla^2 f(x_0)$  - гессиан функции  $f$  в точке  $x_0$ .

Когда линейного приближения функции не достаточно, можно рассмотреть замену  $f(x)$  на  $f_{x_0}^{II}(x)$  в окрестности точки  $x_0$ . В общем, приближения Тейлора дают нам способ локально аппроксимировать функции. Аппроксимация первого порядка определяется градиентом функции в точке, т.е. нормалью к касательной гиперплоскости. А аппроксимация второго порядка представляет из себя параболу. Эти приближения особенно полезны в оптимизации и численных методах, потому что они предоставляют простой способ работы со сложными функциями.

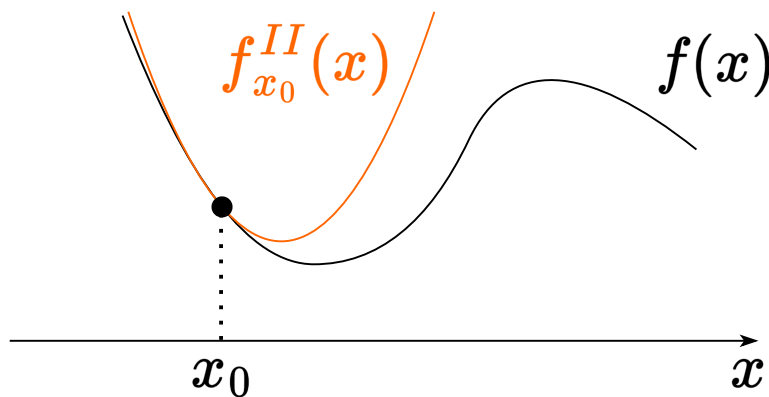


Рисунок 2: Аппроксимация Тейлора второго порядка в окрестности точки  $x_0$

## 2 Дифференциалы

### i Theorem

Пусть  $x \in S$  - внутренняя точка множества  $S$ , и пусть  $D : U \rightarrow V$  - линейный оператор. Мы говорим, что функция  $f$  дифференцируема в точке  $x$  с производной  $D$ , если для всех достаточно малых  $h \in U$  выполняется следующее разложение:

$$f(x + h) = f(x) + D[h] + o(\|h\|)$$

Если для любого линейного оператора  $D : U \rightarrow V$  функция  $f$  не дифференцируема в точке  $x$  с производной  $D$ , то мы говорим, что  $f$  не дифференцируема в точке  $x$ .

После получения дифференциальной записи  $df$  мы можем получить градиент, используя следующую формулу:

$$df(x) = \langle \nabla f(x), dx \rangle$$

Далее, если у нас есть дифференциал в такой форме и мы хотим вычислить вторую производную матричной/векторной функции, мы рассматриваем "старый"  $dx$  как константу  $dx_1$ , затем вычисляем  $d(df) = d^2 f(x)$

$$d^2 f(x) = \langle \nabla^2 f(x) dx_1, dx \rangle = \langle H_f(x) dx_1, dx \rangle$$

### 2.1 Свойства дифференциалов

Пусть  $A$  и  $B$  - постоянные матрицы, а  $X$  и  $Y$  - переменные (или матричные функции).

- $dA = 0$
- $d(\alpha X) = \alpha(dX)$

- $d(AXB) = A(dX)B$
- $d(X + Y) = dX + dY$
- $d(X^T) = (dX)^T$
- $d(XY) = (dX)Y + X(dY)$
- $d\langle X, Y \rangle = \langle dX, Y \rangle + \langle X, dY \rangle$

- $d\left(\frac{X}{\phi}\right) = \frac{\phi dX - (d\phi)X}{\phi^2}$
- $d(\det X) = \det X \langle X^{-T}, dX \rangle$
- $d(\operatorname{tr} X) = \langle I, dX \rangle$
- $df(g(x)) = \frac{df}{dg} \cdot dg(x)$
- $H = (J(\nabla f))^T$
- $d(X^{-1}) = -X^{-1}(dX)X^{-1}$

#### Example

Найти  $df, \nabla f(x)$ , если  $f(x) = \langle x, Ax \rangle - b^T x + c$ .

#### Example

Найти  $df, \nabla f(x)$ , если  $f(x) = \ln \langle x, Ax \rangle$ .

1. Заметим, что  $A$  должна быть положительно определенной, потому что  $\langle x, Ax \rangle$  аргумент логарифма и для любого  $x$  формула должна быть положительной. Таким образом,  $A \in \mathbb{S}_{++}^n$ . Давайте сначала найдем дифференциал:

$$\begin{aligned} df &= d(\ln \langle x, Ax \rangle) = \frac{d(\langle x, Ax \rangle)}{\langle x, Ax \rangle} = \frac{\langle dx, Ax \rangle + \langle x, d(Ax) \rangle}{\langle x, Ax \rangle} = \\ &= \frac{\langle Ax, dx \rangle + \langle x, Adx \rangle}{\langle x, Ax \rangle} = \frac{\langle Ax, dx \rangle + \langle A^T x, dx \rangle}{\langle x, Ax \rangle} = \frac{\langle (A + A^T)x, dx \rangle}{\langle x, Ax \rangle} \end{aligned}$$

2. Наша основная цель - получить форму  $df = \langle \cdot, dx \rangle$

$$df = \left\langle \frac{2Ax}{\langle x, Ax \rangle}, dx \right\rangle$$

Таким образом, градиент равен  $\nabla f(x) = \frac{2Ax}{\langle x, Ax \rangle}$

#### Example

Найти  $df, \nabla f(X)$ , если  $f(X) = \langle S, X \rangle - \log \det X$ .

## 3 Линейный поиск

### 3.1 Задача

Предположим, у нас есть задача минимизации функции  $f(x) : \mathbb{R} \rightarrow \mathbb{R}$  скалярной переменной:

$$f(x) \rightarrow \min_{x \in \mathbb{R}}$$

Иногда мы рассматриваем похожую задачу поиска минимума функции на отрезке  $[a, b]$ :

$$f(x) \rightarrow \min_{x \in [a, b]}$$

### i Example

Типичным примером задачи линейного поиска является выбор подходящего шага для алгоритма градиентного спуска:

$$\begin{aligned} x_{k+1} &= x_k - \alpha \nabla f(x_k) \\ \alpha &= \operatorname{argmin} f(x_{k+1}) \end{aligned}$$

Линейный поиск является фундаментальной задачей оптимизации, использующийся для решения сложных задач. Для упрощения предположим, что  $f(x)$  *унимодальна*, то есть имеет единственный пик или впадину.

## 3.2 Унимодальная функция

### i Definition

Функция  $f(x)$  называется **унимодальной** на отрезке  $[a, b]$ , если существует  $x_* \in [a, b]$ , что  $f(x_1) > f(x_2) \quad \forall a \leq x_1 < x_2 < x_*$  и  $f(x_1) < f(x_2) \quad \forall x_* < x_1 < x_2 \leq b$

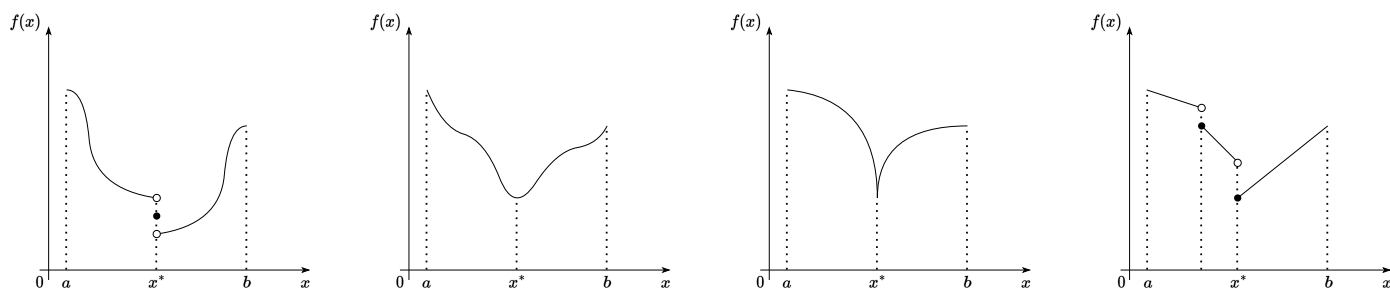


Рисунок 3: Примеры унимодальных функций

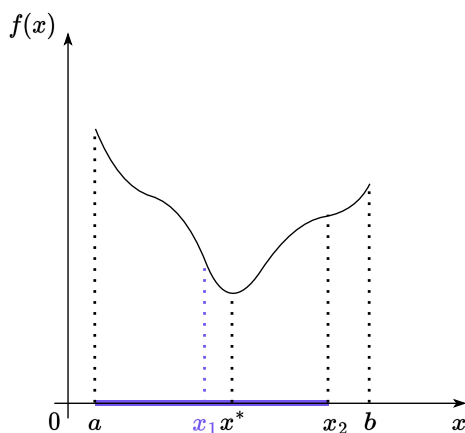
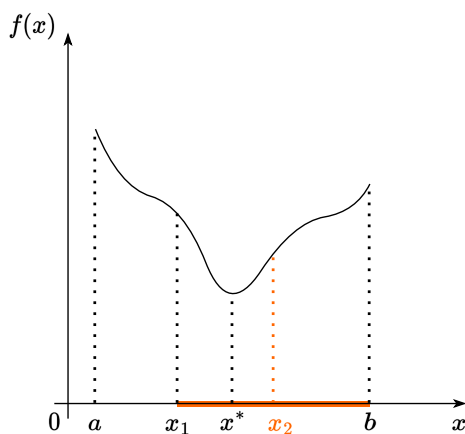
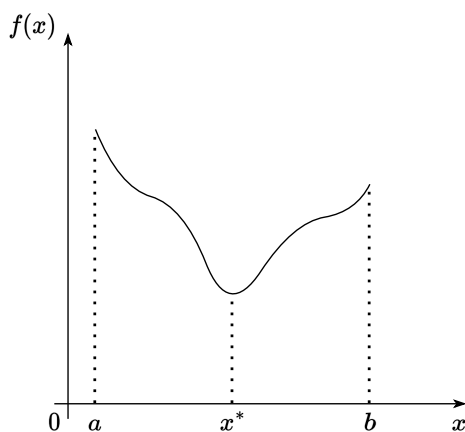
## 3.3 Ключевое свойство унимодальных функций

Пусть  $f(x)$  является унимодальной функцией на отрезке  $[a, b]$ . Тогда если  $x_1 < x_2 \in [a, b]$ , то:

- if  $f(x_1) \leq f(x_2) \rightarrow x_* \in [a, x_2]$
- if  $f(x_1) \geq f(x_2) \rightarrow x_* \in [x_1, b]$



**Доказательство** Докажем первое утверждение. Предположим, что  $f(x_1) \leq f(x_2)$ , но  $x^* > x_2$ . Тогда, поскольку  $x_1 < x_2 < x^*$ , из определения унимодальности функции  $f(x)$  следует, что должно выполняться неравенство  $f(x_1) > f(x_2)$ . Мы получили противоречие.



### 3.4 Метод дихотомии

Мы хотим решить следующую задачу:

$$f(x) \rightarrow \min_{x \in [a,b]}$$

Делим отрезок на две равные части и выбираем, основываясь на ключевом свойстве, описанном выше, ту, которая содержит решение задачи. Наша цель после одной итерации метода - локализовать решение в отрезке в два раза меньшей длины.

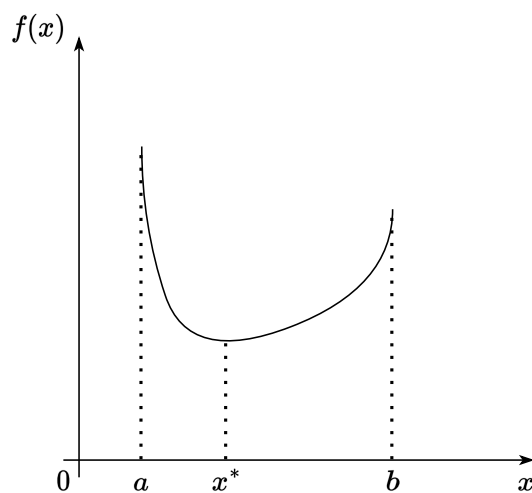


Рисунок 4: Метод дихотомии для унимодальной функции

Мы измеряем значение функции в середине отрезка

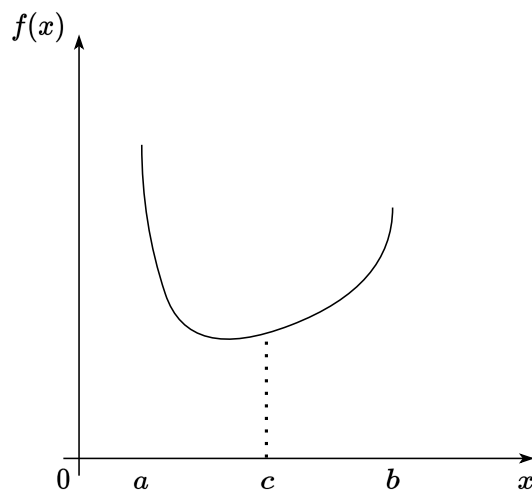


Рисунок 5: Метод дихотомии для унимодальной функции

Чтобы применить ключевое свойство, мы выполняем еще одно измерение.

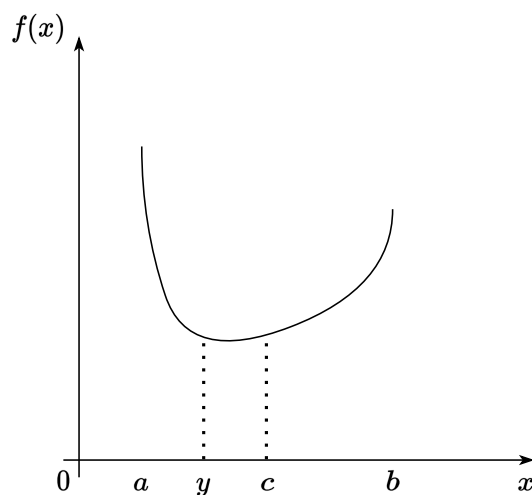


Рисунок 6: Метод дихотомии для унимодальной функции

Выбираем целевой отрезок. В этом случае нас все устраивает, потому что уже разделили решение на две равные части. Но это не всегда так.

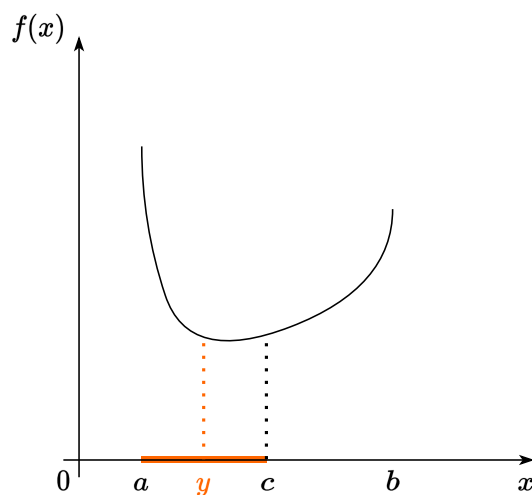


Рисунок 7: Метод дихотомии для унимодальной функции

Рассмотрим другую унимодальную функцию.



Рисунок 8: Метод дихотомии для унимодальной функции

Измеряем значение функции в середине отрезка.

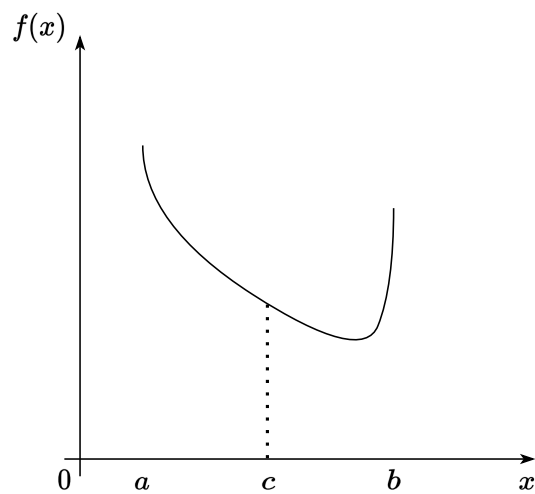


Рисунок 9: Метод дихотомии для унимодальной функции

Делаем еще одно измерение.

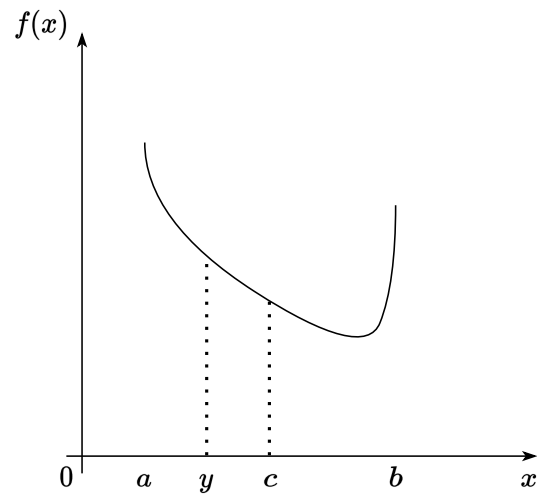


Рисунок 10: Метод дихотомии для унимодальной функции

Выбираем целевой отрезок. Мы можем видеть, что полученный отрезок не является половиной исходного. Он равен  $\frac{3}{4}(b - a)$ . Чтобы исправить это, нам нужен еще один шаг алгоритма.

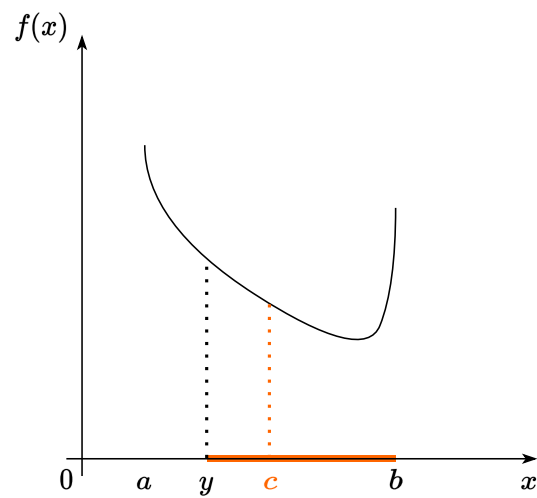


Рисунок 11: Метод дихотомии для унимодальной функции

После еще одного дополнительного измерения мы точно получим  $\frac{2}{3} \frac{3}{4}(b - a) = \frac{1}{2}(b - a)$



Рисунок 12: Метод дихотомии для унимодальной функции

В итоге, каждая последующая итерация будет требовать не более двух измерений значений функции.



Рисунок 13: Метод дихотомии для унимодальной функции

### 3.5 Метод дихотомии. Алгоритм

```
def binary_search(f, a, b, epsilon):  
    c = (a + b) / 2  
    while abs(b - a) > epsilon:  
        y = (a + c) / 2.0  
        if f(y) <= f(c):  
            b = c
```

```
    c = y
else:
    z = (b + c) / 2.0
if f(c) <= f(z):
    a = y
    b = z
else:
    a = c
    c = z
return c
```

### 3.6 Метод дихотомии. Оценка

Длина отрезка на  $k$ -й итерации:

$$\Delta_k = b_k - a_k = \frac{1}{2^k}(b - a)$$

Для унимодальных функций это верно, если мы выбираем середину отрезка в качестве выхода итерации  $x_k$ :

$$|x_k - x_*| \leq \frac{\Delta_k}{2} \leq \frac{1}{2^{k+1}}(b - a) \leq (0.5)^k \cdot \frac{b - a}{2}$$

Заметим, что на каждой итерации мы спрашиваем оракул не более двух раз, поэтому количество вызовов функции равно  $N = 2 \cdot k$ , что означает:

$$|x_k - x_*| \leq (0.5)^{\frac{N}{2}} \cdot \frac{b - a}{2} \leq (0.707)^N \frac{b - a}{2}$$

Помечая правую часть последнего неравенства за  $\varepsilon$ , мы получаем количество итераций метода, необходимое для достижения точности  $\varepsilon$ :

$$K = \left\lceil \log_2 \frac{b - a}{\varepsilon} - 1 \right\rceil$$

### 3.7 Метод золотого сечения

Идея очень похожа на метод дихотомии. На отрезке есть две точки - левая и правая точки золотого сечения и интуитивно понятно, что на следующей итерации одна из точек останется точкой золотого сечения.

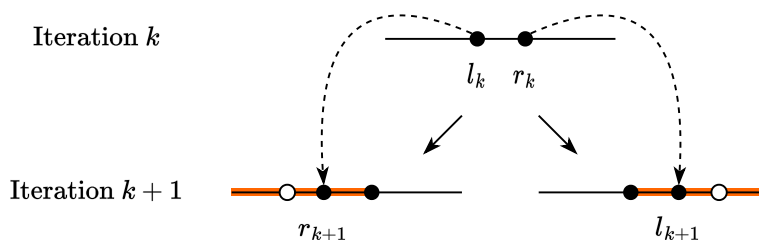


Рисунок 14: Идея, позволяющая уменьшить количество вызовов функции

### 3.8 Метод золотого сечения. Алгоритм

```
def golden_search(f, a, b, epsilon):  
    tau = (sqrt(5) + 1) / 2  
    y = a + (b - a) / tau**2  
    z = a + (b - a) / tau  
    while b - a > epsilon:  
        if f(y) <= f(z):  
            b = z  
            z = y  
            y = a + (b - a) / tau**2  
        else:  
            a = y  
            y = z  
            z = a + (b - a) / tau  
    return (a + b) / 2
```

### 3.9 Метод золотого сечения. Оценка

$$|x_k - x_*| \leq \frac{b_k - a_k}{2} = \left(\frac{1}{\tau}\right)^N \frac{b - a}{2} \approx 0.618^k \frac{b - a}{2}$$

$$\text{где } \tau = \frac{\sqrt{5}+1}{2}.$$

- Знаменатель геометрической прогрессии для метода золотого сечения **больше**, чем для метода дихотомии: 0.618 больше, чем 0.5.
- Количество вызовов функции **меньше** для метода золотого сечения, чем для метода дихотомии: 0.707 больше (значит медленнее), чем 0.618. Для каждой итерации метода дихотомии (кроме первой), функция вызывается не более двух раз, в то время как для метода золотого сечения, она вызывается не более одного раза за итерацию.

### 3.10 Метод параболической интерполяции

Три точки, не лежащие на одной прямой, однозначно определяют параболу, проходящую через них. Идея метода — аппроксимировать функцию такой параболой и в качестве следующего приближения



взять точку её минимума. Предположим, у нас есть 3 точки  $x_1 < x_2 < x_3$  такие, что отрезок  $[x_1, x_3]$  содержит минимум функции  $f(x)$ . Тогда мы должны решить следующую систему уравнений:

$$ax_i^2 + bx_i + c = f_i = f(x_i), i = 1, 2, 3$$

Заметим, что эта система линейна, мы должны решить ее относительно  $a, b, c$ . Минимум этой параболы вычисляется по формуле:

$$u = -\frac{b}{2a} = x_2 - \frac{(x_2 - x_1)^2(f_2 - f_3) - (x_2 - x_3)^2(f_2 - f_1)}{2[(x_2 - x_1)(f_2 - f_3) - (x_2 - x_3)(f_2 - f_1)]}$$

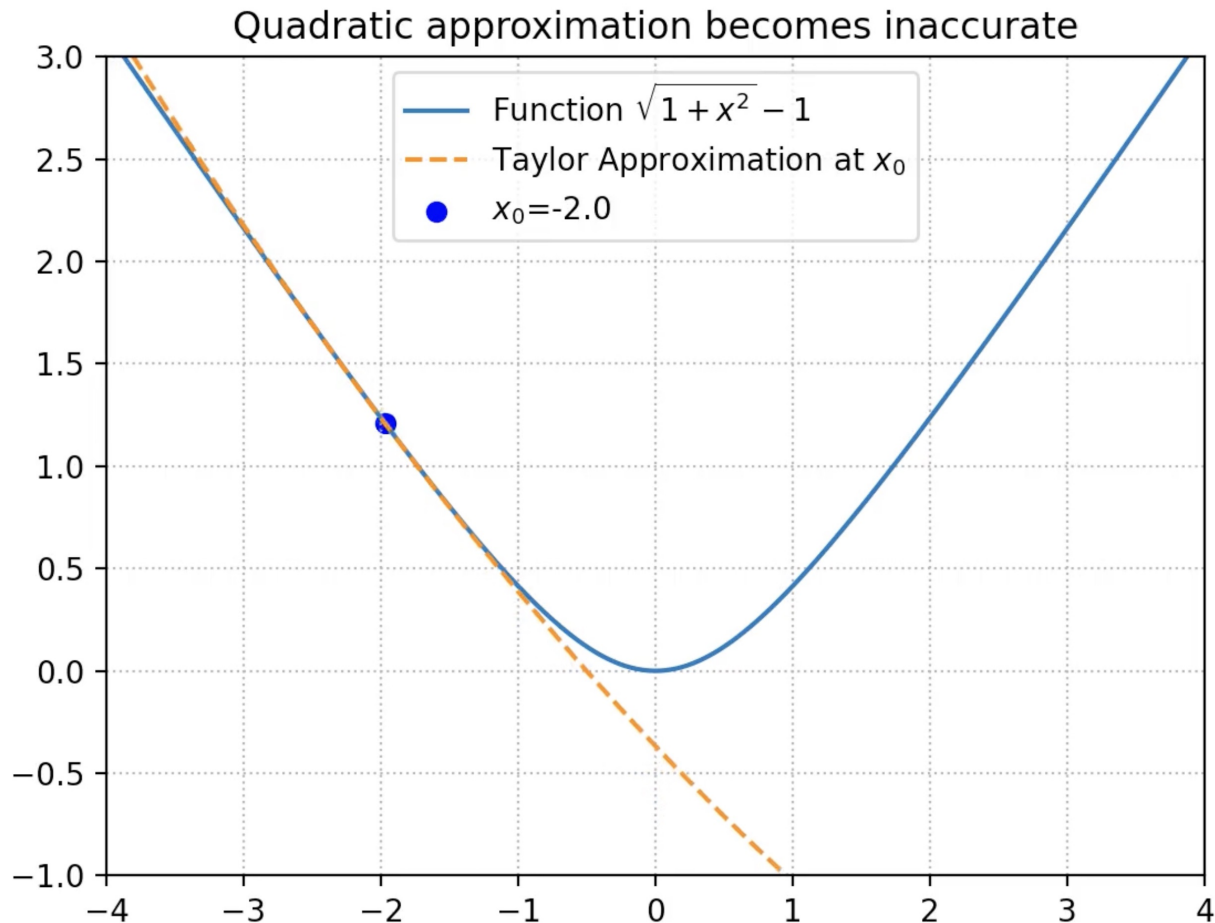
Заметим, что если  $f_2 < f_1, f_2 < f_3$ , то  $u$  будет лежать в  $[x_1, x_3]$

### 3.11 Метод параболической интерполяции. Алгоритм <sup>1</sup>

```
def parabola_search(f, x1, x2, x3, epsilon):
    f1, f2, f3 = f(x1), f(x2), f(x3)
    while x3 - x1 > epsilon:
        u = x2 - ((x2 - x1)**2*(f2 - f3) - (x2 - x3)**2*(f2 - f1))/(2*((x2 - x1)*(f2 - f3) - (x2 - x3)*(f2 - f1)))
        fu = f(u)

        if x2 <= u:
            if f2 <= fu:
                x1, x2, x3 = x1, x2, u
                f1, f2, f3 = f1, f2, fu
            else:
                x1, x2, x3 = x2, u, x3
                f1, f2, f3 = f2, fu, f3
        else:
            if fu <= f2:
                x1, x2, x3 = x1, u, x2
                f1, f2, f3 = f1, fu, f2
            else:
                x1, x2, x3 = u, x2, x3
                f1, f2, f3 = fu, f2, f3
    return (x1 + x3)/2
```

<sup>1</sup>Скорость сходимости этого метода суперлинейна, но локальна, что означает, что мы можем получить выгоду от использования этого метода только вблизи некоторой окрестности оптимума. [Здесь](#) доказательство суперлинейной сходимости порядка 1.32.



### 3.12 Неточный линейный поиск

Нам не всегда нужно точно решать задачу минимизации. Иногда, достаточно найти приближенное решение. Такое часто встречается в задаче выбора шага метода оптимизации

$$\begin{aligned}x_{k+1} &= x_k - \alpha \nabla f(x_k) \\ \alpha &= \operatorname{argmin} f(x_{k+1})\end{aligned}$$

Рассмотрим скалярную функцию  $\phi(\alpha)$  в точке  $x_k$ :

$$\phi(\alpha) = f(x_k - \alpha \nabla f(x_k)), \alpha \geq 0$$

Первое приближение  $\phi(\alpha)$  в окрестности  $\alpha = 0$  равно:

$$\phi(\alpha) \approx f(x_k) - \alpha \nabla f(x_k)^T \nabla f(x_k)$$

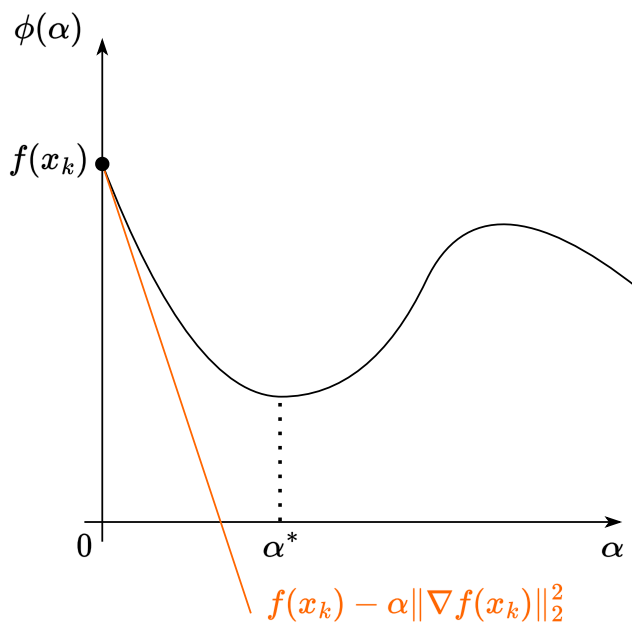


Рисунок 15: Иллюстрация аппроксимации Тейлора  $\phi_0^I(\alpha)$

### 3.13 Неточный линейный поиск. Условие достаточного убывания

Условие неточного линейного поиска, известное как условие Армихо, утверждает, что  $\alpha$  должно обеспечить достаточное убывание функции  $f$ , удовлетворяющее:

$$f(x_k - \alpha \nabla f(x_k)) \leq f(x_k) - c_1 \cdot \alpha \nabla f(x_k)^T \nabla f(x_k)$$

для некоторой постоянной  $c_1 \in (0, 1)$ . Заметим, что установка  $c_1 = 1$  соответствует первому приближению Тейлора  $\phi(\alpha)$ . Однако это условие может принимать очень малые значения  $\alpha$ , потенциально замедляя процесс решения. Обычно на практике используется  $c_1 \approx 10^{-4}$ .

#### Example

Если  $f(x)$  представляет собой функцию стоимости в задаче оптимизации, важен выбор подходящего значения  $c_1$ . Например, при обучении моделей ML неправильное значение  $c_1$  может привести к очень медленной сходимости или пропуску минимума.



Рисунок 16: Иллюстрация условия достаточного убывания с коэффициентом  $c_1$

### 3.14 Неточный линейный поиск. Условия Гольдштейна

Рассмотрим две линейные скалярные функции  $\phi_1(\alpha)$  и  $\phi_2(\alpha)$ :

$$\phi_1(\alpha) = f(x_k) - c_1 \alpha \|\nabla f(x_k)\|^2$$

$$\phi_2(\alpha) = f(x_k) - c_2 \alpha \|\nabla f(x_k)\|^2$$

Условия Гольдштейна-Армихо находят функцию  $\phi(\alpha)$  между  $\phi_1(\alpha)$  и  $\phi_2(\alpha)$ . Обычно  $c_1 = \rho$  и  $c_2 = 1 - \rho$ , с  $\rho \in (0, 0.5)$ .

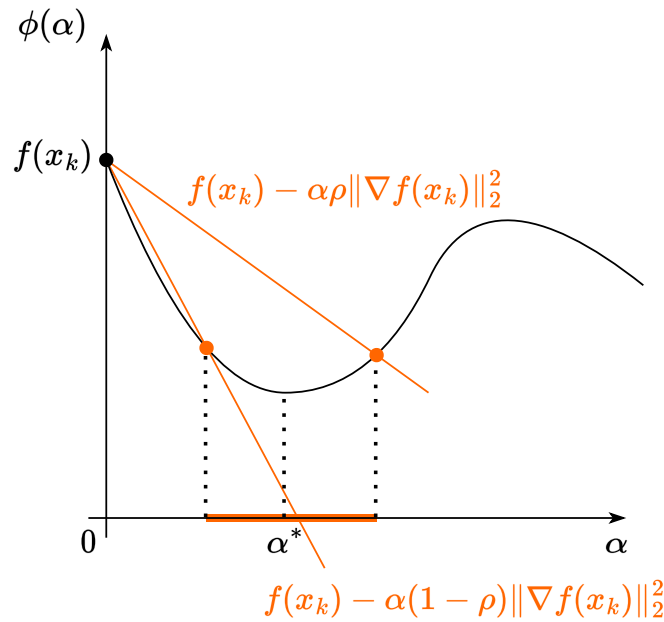


Рисунок 17: Иллюстрация условий Гольдштейна

### 3.15 Неточный линейный поиск. Условие ограничения на кривизну

Чтобы избежать слишком коротких шагов, мы вводим второй критерий:

$$-\nabla f(x_k - \alpha \nabla f(x_k))^T \nabla f(x_k) \geq c_2 \nabla f(x_k)^T (-\nabla f(x_k))$$

для некоторого  $c_2 \in (c_1, 1)$ . Здесь  $c_1$  из условия Армихо.

Левая часть является производной  $\nabla_\alpha \phi(\alpha)$ , гарантирующей, что наклон  $\phi(\alpha)$  в целевой точке не менее чем в  $c_2$  раз больше начального наклона  $\nabla_\alpha \phi(\alpha)(0)$ .

Обычно для методов Ньютона и квазиньютоновских методов используется  $c_2 \approx 0.9$ . В объединении условие достаточного убывания и ограничение на кривизну образуют условия Вульфа.



Рисунок 18: Иллюстрация условия ограничения на кривизну

### 3.16 Неточный линейный поиск. Условия Вульфа

$$-\nabla f(x_k - \alpha \nabla f(x_k))^T \nabla f(x_k) \geq c_2 \nabla f(x_k)^T (-\nabla f(x_k))$$

Вместе, условие достаточного убывания и ограничение на кривизну образуют условия Вульфа.

#### Theorem

Пусть  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  непрерывно дифференцируема, и пусть  $\phi(\alpha) = f(x_k - \alpha \nabla f(x_k))$ . Предположим, что  $\nabla f(x_k)^T p_k < 0$ , где  $p_k = -\nabla f(x_k)$ , деля  $p_k$  направлением спуска. Также предположим, что  $f$  ограничена снизу вдоль луча  $\{x_k + \alpha p_k \mid \alpha > 0\}$ . Мы хотим показать, что для  $0 < c_1 < c_2 < 1$ , существуют интервалы шагов, удовлетворяющие условиям Вульфа.



Рисунок 19: Иллюстрация условий Вульфа

### 3.17 Неточный линейный поиск. Условия Вульфа. Доказательство

1. Поскольку  $\phi(\alpha) = f(x_k + \alpha p_k)$  ограничена снизу и  $l(\alpha) = f(x_k) + \alpha c_1 \nabla f(x_k)^T p_k$  неограничена снизу (как  $\nabla f(x_k)^T p_k < 0$ ), график  $l(\alpha)$  должен пересекать график  $\phi(\alpha)$  по крайней мере один раз. Пусть  $\alpha' > 0$  будет наименьшим таким значением, удовлетворяющим:

$$f(x_k + \alpha' p_k) \leq f(x_k) + \alpha' c_1 \nabla f(x_k)^T p_k. \quad (1)$$

Это гарантирует выполнение условия достаточного убывания.

2. По теореме о среднем значении, существует  $\alpha'' \in (0, \alpha')$  такое, что:

$$f(x_k + \alpha' p_k) - f(x_k) = \alpha' \nabla f(x_k + \alpha'' p_k)^T p_k. \quad (2)$$

Подставляя  $f(x_k + \alpha' p_k)$  из (1) в (2), мы получаем:

$$\alpha' \nabla f(x_k + \alpha'' p_k)^T p_k \leq \alpha' c_1 \nabla f(x_k)^T p_k.$$

Делим на  $\alpha' > 0$ , получаем:

$$\nabla f(x_k + \alpha'' p_k)^T p_k \leq c_1 \nabla f(x_k)^T p_k. \quad (3)$$

3. Поскольку  $c_1 < c_2$  и  $\nabla f(x_k)^T p_k < 0$ , неравенство  $c_1 \nabla f(x_k)^T p_k < c_2 \nabla f(x_k)^T p_k$  выполняется. Это означает, что существует  $\alpha''$  такое, что:

$$\nabla f(x_k + \alpha'' p_k)^T p_k \leq c_2 \nabla f(x_k)^T p_k. \quad (4)$$

Неравенства (3) и (4) вместе гарантируют выполнение условий Вульфа.

4. Для сильных условий Вульфа, условие ограничения на кривизну:

$$|\nabla f(x_k + \alpha p_k)^T p_k| \leq c_2 |\nabla f(x_k)^T p_k| \quad (5)$$

выполняется, потому что  $\nabla f(x_k + \alpha p_k)^T p_k$  отрицательно и ограничено снизу  $c_2 \nabla f(x_k)^T p_k$ .

5. Из-за гладкости  $f$ , существует интервал вокруг  $\alpha''$ , где выполняются условия Вульфа (и, следовательно, сильные условия Вульфа). Таким образом, доказательство завершено.

### 3.18 Бэктрекинг

Бэктрекинг - это техника для нахождения шага, удовлетворяющего условию Армихо, условиям Гольдштейна или другим критериям неточного линейного поиска. Она начинается с относительно большого шага и итеративно уменьшает его до тех пор, пока не будет выполнено условие.

#### 3.18.1 Алгоритм:

1. Выберите начальный шаг,  $\alpha_0$ , и параметры  $\beta \in (0, 1)$  и  $c_1 \in (0, 1)$ .
2. Проверьте, удовлетворяет ли выбранный шаг выбранному условию (например, условию Армихо).
3. Если условие выполнено, остановитесь; в противном случае, установите  $\alpha := \beta \alpha$  и повторите шаг 2.

Шаг  $\alpha$  обновляется как

$$\alpha_{k+1} := \beta \alpha_k$$

в каждой итерации до тех пор, пока выбранное условие не будет выполнено.

### Example

В обучении моделей машинного обучения линейный поиск с возвратом может использоваться для регулировки скорости обучения. Если потеря не уменьшается достаточно, скорость обучения уменьшается мультипликативно до тех пор, пока не будет выполнено условие Армихо.

### 3.19 Численная иллюстрация

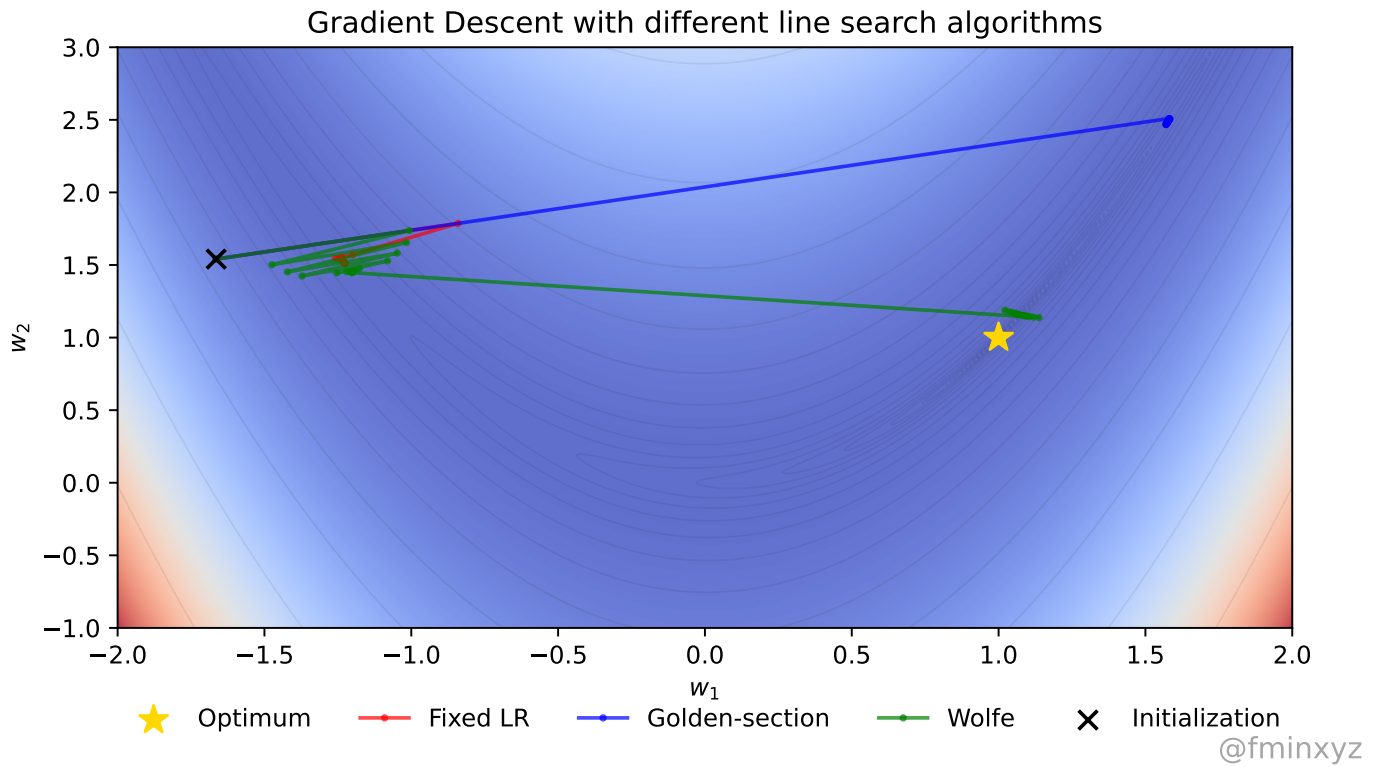


Рисунок 20: Сравнение различных алгоритмов линейного поиска

[Открыть в Colab](#)



### 3.20 Градиентный спуск с линейным поиском



## 4 Задачи

### 4.1 Задача 1

#### i Example

Найдите  $\nabla f(x)$ , если  $f(x) = \frac{1}{2}x^T A x + b^T x + c$ .

### 4.2 Задача 2

#### i Example

Найдите  $\nabla f(X)$ , если  $f(X) = \text{tr}(A X^{-1} B)$

### 4.3 Задача 3

#### Example

Найдите градиент  $\nabla f(x)$  и гессиан  $\nabla^2 f(x)$ , если  $f(x) = \frac{1}{3}\|x\|_2^3$

## 5 Задачи на дом

### 5.1 Линейный поиск

1. [10 баллов] Рассмотрим строго выпуклую квадратичную функцию  $f: \mathbb{R}^n \rightarrow \mathbb{R}$ , и пусть мы начинаем из точки  $x_k \in \mathbb{R}^n$  двигаться в направлении антиградиента  $-\nabla f(x_k)$ , при этом  $\nabla f(x_k) \neq 0$ . Покажите, что минимум  $f$  вдоль этого направления как функция шага  $\alpha$ , для убывающей функции в точке  $x_k$ , удовлетворяет условию Армихо для любого  $c_1$  в диапазоне  $0 \leq c_1 \leq \frac{1}{2}$ . В частности, покажите, что следующее неравенство выполняется в оптимальном  $\alpha^*$ :

$$\varphi(\alpha) = f(x_{k+1}) = f(x_k - \alpha \nabla f(x_k)) \leq f(x_k) - c_1 \alpha \|\nabla f(x_k)\|_2^2$$

2. Реализация и тестирование условий линейного поиска в градиентном спуске [36 баллов]

$$x_{k+1} = x_k - \alpha \nabla f(x_k)$$

В этом задании мы будем модифицировать существующий Python код для градиентного спуска, чтобы включить различные условия линейного поиска. Мы протестируем эти модификации на двух функциях: квадратичной функции и функции Розенброка. Основные цели - понять, как различные стратегии линейного поиска влияют на сходимость алгоритма градиентного спуска и сравнить их эффективность на основе количества вызовов функции.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize_scalar
np.random.seed(214)

# Define the quadratic function and its gradient
def quadratic_function(x, A, b):
    return 0.5 * np.dot(x.T, np.dot(A, x)) - np.dot(b.T, x)

def grad_quadratic(x, A, b):
    return np.dot(A, x) - b

# Generate a 2D quadratic problem with a specified condition number
def generate_quadratic_problem(cond_number):
    # Random symmetric matrix
    M = np.random.randn(2, 2)
    M = np.dot(M, M.T)

    # Ensure the matrix has the desired condition number
    U, s, V = np.linalg.svd(M)
```

```
s = np.linspace(cond_number, 1, len(s)) # Spread the singular values
A = np.dot(U, np.dot(np.diag(s), V))

# Random b
b = np.random.randn(2)

return A, b

# Gradient descent function
def gradient_descent(start_point, A, b, stepsize_func, max_iter=100):
    x = start_point.copy()
    trajectory = [x.copy()]

    for i in range(max_iter):
        grad = grad_quadratic(x, A, b)
        step_size = stepsize_func(x, grad)
        x -= step_size * grad
        trajectory.append(x.copy())

    return np.array(trajectory)

# Backtracking line search strategy using scipy
def backtracking_line_search(x, grad, A, b, alpha=0.3, beta=0.8):
    def objective(t):
        return quadratic_function(x - t * grad, A, b)
    res = minimize_scalar(objective, method='golden')
    return res.x

# Generate ill-posed problem
cond_number = 30
A, b = generate_quadratic_problem(cond_number)

# Starting point
start_point = np.array([1.0, 1.8])

# Perform gradient descent with both strategies
trajectory_fixed = gradient_descent(start_point, A, b, lambda x, g: 5e-2)
trajectory_backtracking = gradient_descent(start_point, A, b, lambda x, g: backtracking_line_se

# Plot the trajectories on a contour plot
x1, x2 = np.meshgrid(np.linspace(-2, 2, 400), np.linspace(-2, 2, 400))
Z = np.array([quadratic_function(np.array([x, y]), A, b) for x, y in zip(x1.flatten(), x2.flatt

plt.figure(figsize=(10, 8))
plt.contour(x1, x2, Z, levels=50, cmap='viridis')
plt.plot(trajectory_fixed[:, 0], trajectory_fixed[:, 1], 'o-', label='Fixed Step Size')
plt.plot(trajectory_backtracking[:, 0], trajectory_backtracking[:, 1], 'o-', label='Backtrackin

# Add markers for start and optimal points
```

```
plt.plot(start_point[0], start_point[1], 'ro', label='Start Point')
optimal_point = np.linalg.solve(A, b)
plt.plot(optimal_point[0], optimal_point[1], 'y*', markersize=15, label='Optimal Point')

plt.legend()
plt.title('Gradient Descent Trajectories on Quadratic Function')
plt.xlabel('x1')
plt.ylabel('x2')
plt.savefig("linesearch.svg")
plt.show()
```

Начните с ознакомления с предоставленным Python кодом. Этот код реализует градиентный спуск с фиксированным шагом и бэктрекингом на квадратичной функции. Ознакомьтесь с тем, как реализованы функции градиентного спуска и стратегии размера шага.

1. [10/36 баллов] Измените функцию градиентного спуска, чтобы включить следующие условия линейного поиска:

1. Дихотомия
2. Условие достаточного убывания
3. Условия Вольфа
4. Шаг Поляка

$$\alpha_k = \frac{f(x_k) - f^*}{\|\nabla f(x_k)\|_2^2},$$

где  $f^*$  - оптимальное значение функции. Кажется странным использовать оптимальное значение функции в размере шага, но есть варианты оценить его даже без знания оптимального значения.

5. Метод знака градиента:

$$\alpha_k = \frac{1}{\|\nabla f(x_k)\|_2},$$

Протестируйте модифицированный алгоритм градиентного спуска с реализованными стратегиями поиска размера шага на предоставленной квадратичной функции. Постройте траектории по итерациям для каждого условия. Выберите и укажите гиперпараметры для неточных условий линейного поиска. Выберите и укажите **критерий останова**. Начните с точки  $x_0 = (-1, 2)^T$ .

2. [8/36 баллов] Сравните эти 7 методов с точки зрения бюджета. Постройте график значения функции от количества вызовов функции для каждого метода на одном графике.
3. [10/36 баллов] Постройте траекторию для другой функции с тем же набором методов

$$f(x_1, x_2) = 10(x_2 - x_1^2)^2 + (x_1 - 1)^2$$

с  $x_0 = (-1, 2)^T$ . Возможно, вам придется подстроить гиперпараметры.

4. [8/36 баллов] Постройте тот же график значения функции от количества вызовов функции для этого эксперимента.

## 5.2 Матрично-векторное дифференцирование

1. [6 баллов] Найдите градиент  $\nabla f(x)$  и гессиан  $f''(x)$ , если  $f(x) = \frac{1}{2}\|A - xx^T\|_F^2$ ,  $A \in \mathbb{S}^n$
2. [6 баллов] Найдите градиент  $\nabla f(x)$  и гессиан  $f''(x)$ , если  $f(x) = \frac{1}{2}\|Ax - b\|_2^2$ .
3. [8 баллов] Найдите градиент  $\nabla f(x)$  и гессиан  $f''(x)$ , если

$$f(x) = \frac{1}{m} \sum_{i=1}^m \log(1 + \exp(a_i^T x)) + \frac{\mu}{2}\|x\|_2^2, \quad a_i, x \in \mathbb{R}^n, \quad \mu > 0$$

4. [8 баллов] Найдите градиент  $\nabla_A f(A)$  от следа функции матричной экспоненты  $f(A) = \text{tr}(e^A)$  с точки зрения  $A$ .  
Подсказка: Используйте определение матричной экспоненты. Используйте определение дифференциала  $df = f(A + dA) - f(A) + o(\|dA\|)$  с пределом  $\|dA\| \rightarrow 0$ .
5. [20 баллов] **Главные компоненты через вычисление градиента.** Пусть есть набор данных  $\{x_i\}_{i=1}^N, x_i \in \mathbb{R}^D$ , который мы хотим преобразовать в набор данных меньшей размерности  $d$  с помощью проекции на линейное подпространство, определяемое матрицей  $P \in \mathbb{R}^{D \times d}$ . Ортогональная проекция вектора  $x$  на это подпространство может быть вычислена как  $P(P^T P)^{-1} P^T x$ . Чтобы найти оптимальную матрицу  $P$ , рассмотрим следующую задачу оптимизации:

$$F(P) = \sum_{i=1}^N \|x_i - P(P^T P)^{-1} P^T x_i\|^2 = N \cdot \text{tr}((I - P(P^T P)^{-1} P^T)^2 S) \rightarrow \min_{P \in \mathbb{R}^{D \times d}},$$

где  $S = \frac{1}{N} \sum_{i=1}^N x_i x_i^T$  - выборочная ковариационная матрица для нормализованного набора данных.

1. Найдите градиент  $\nabla_P F(P)$ , рассчитанный для произвольной матрицы  $P$  с ортогональными столбцами, т.е.  $P : P^T P = I$ .

Подсказка: При вычислении дифференциала  $dF(P)$ , сначала рассмотрите  $P$  как произвольную матрицу, а затем используйте ортогональность столбцов  $P$  в полученном выражении.

2. Рассмотрите разложение матрицы  $S$  на собственные значения:

$$S = Q \Lambda Q^T,$$

где  $\Lambda$  - диагональная матрица с собственными значениями на диагонали, и  $Q = [q_1 | q_2 | \dots | q_D] \in \mathbb{R}^{D \times D}$  - ортогональная матрица, состоящая из собственных векторов  $q_i$  в качестве столбцов. Докажите следующее:

1. Градиент  $\nabla_P F(P)$  равен нулю для любой матрицы  $P$ , состоящей из  $d$  различных собственных векторов  $q_i$  в качестве ее столбцов.
2. Минимальное значение  $F(P)$  достигается для матрицы  $P$ , состоящей из собственных векторов  $q_i$ , соответствующих наибольшим собственным значениям  $S$ .