

Численные методы

1. Решение уравнений
2. Вычисление площади
(интеграла)
3. Вычисление длины кривой
4. Оптимизация

Численные методы

Тема 1. Решение уравнений

Основные понятия

Задача: решить уравнение

$$x^2 = 5 \cos x \iff x^2 - 5 \cos x = 0$$

$$f(x) = 0$$

Типы решения:

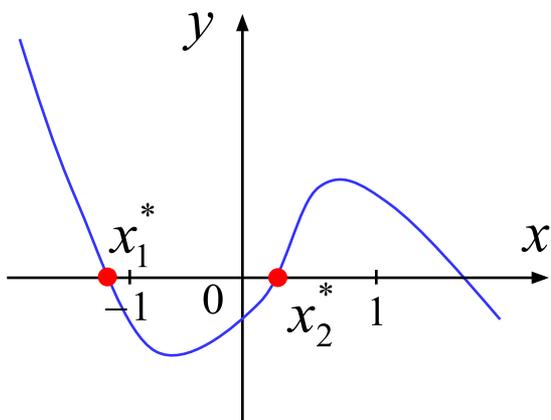
- **аналитическое** (точное, в виде формулы)

$$x^* = \dots$$



- **приближенное** (неточное)

графический метод



численные методы

$x_0 = -1$ начальное приближение

$x_1 = -1,102$

$x_2 = -1,215$



при $N \rightarrow \infty$

$$x^* \approx -1,252\dots$$

Численные методы

Идея: последовательное уточнение решения с помощью некоторого алгоритма.

Область применения: когда найти точное решение невозможно или крайне сложно.

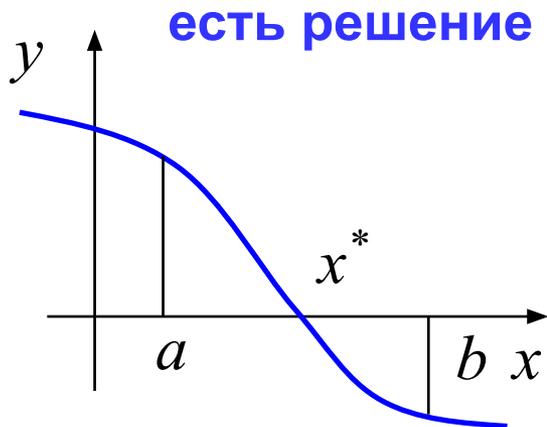
- ⊕ 1) можно найти хоть какое-то решение
- 2) во многих случаях можно оценить ошибку (то есть можно найти решение **с заданной точностью**)

- ⊖ 1) нельзя найти *точное* решение

$$\sqrt{x+1} - 4\sin(x-1) = 0 \quad \longrightarrow \quad x = \cancel{1,2974} \quad x \approx 1,3974$$

- 2) невозможно исследовать решение при изменении параметров
- 3) большой объем вычислений
- 4) иногда сложно оценить ошибку
- 5) нет универсальных методов

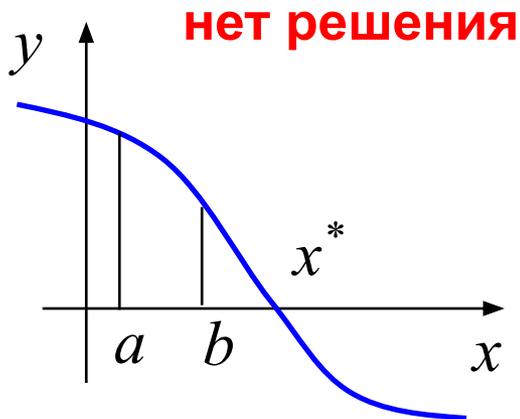
Есть ли решение на $[a, b]$?



$$f(a) > 0$$

$$f(b) < 0$$

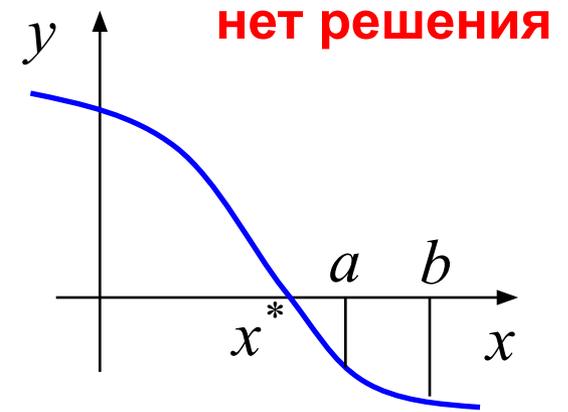
$$f(a)f(b) < 0$$



$$f(a) > 0$$

$$f(b) > 0$$

$$f(a)f(b) > 0$$



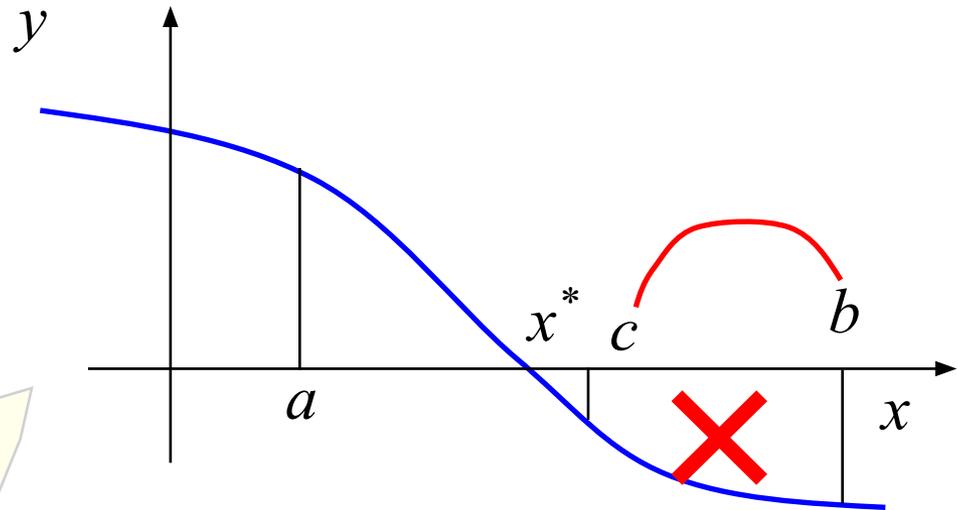
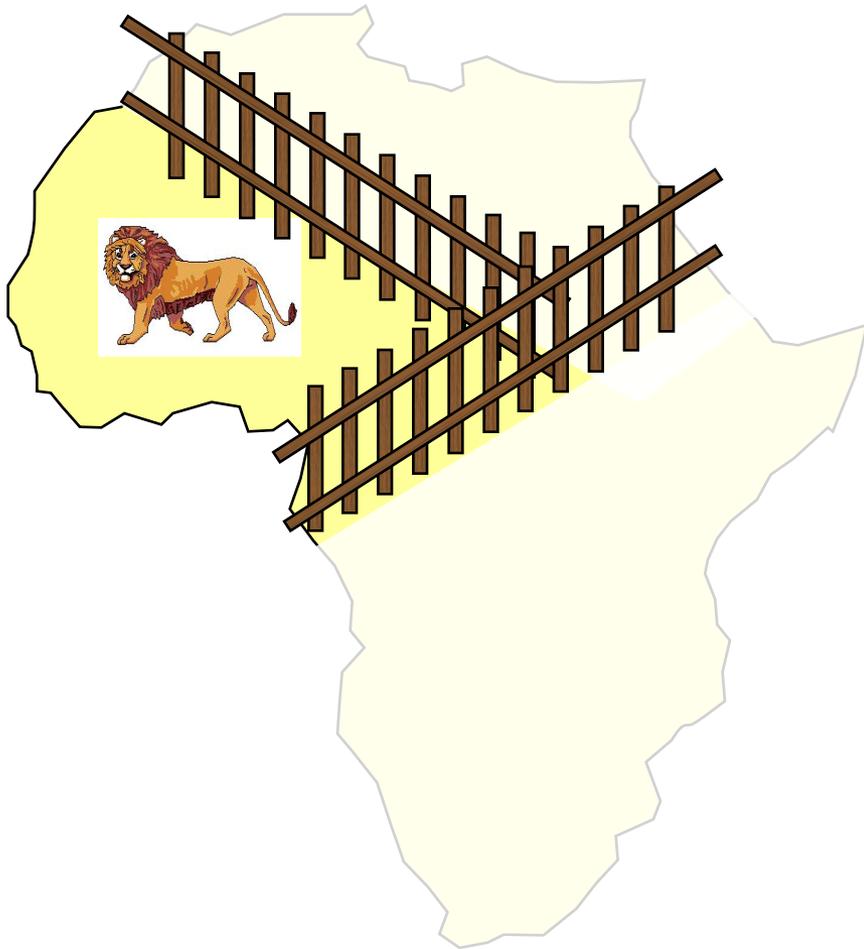
$$f(a) < 0$$

$$f(b) < 0$$



Если **непрерывная** функция $f(x)$ имеет разные знаки на концах интервала $[a, b]$, то в некоторой точке внутри $[a, b]$ имеем $f(x) = 0$!

Метод дихотомии (деление пополам)



1. Найти середину отрезка $[a, b]$:
$$c = (a + b) / 2;$$
2. Если $f(c) * f(a) < 0$, сдвинуть правую границу интервала
$$b = c;$$
3. Если $f(c) * f(a) \geq 0$, сдвинуть левую границу интервала
$$a = c;$$
4. Повторять шаги 1-3, пока не будет $b - a \leq \epsilon$.

Метод дихотомии (деления пополам)

-  • простота
 - можно получить решение с заданной **точностью** (в пределах точности машинных вычислений)
-  • нужно знать **интервал** $[a, b]$
 - на интервале $[a, b]$ должно быть только **одно** решение
 - **большое число шагов** для достижения высокой точности
 - только для функций **одной** переменной

Метод деления отрезка пополам

```
//-----  
// BinSolve находит решение на [a,b]  
//          методом деления отрезка пополам  
// Вход:   a, b - границы интервала,  a < b  
//          eps - точность решения  
// Выход:  x - решение уравнения f(x)=0  
//-----  
float BinSolve ( float a, float b, float eps )  
{  
    float c;  
    while ( b - a > eps )  
    {  
        c = (a + b) / 2;  
        if ( f(a)*f(c) < 0 )  
            b = c;  
        else a = c;  
    }  
    return (a + b) / 2;  
}
```

```
float f ( float x )  
{  
    return x*x - 5;  
}
```

Как подсчитать число шагов?

```
float BinSolve ( float a, float b,  
                float eps, int &n )  
{  
    float c;  
    n = 0;  
    while ( b - a > eps )  
    {  
        c = (a + b) / 2;  
        if ( f  
            b  
            else a  
            n ++;  
        }  
    return (
```

значение переменной
меняется внутри функции

Вызов в основной программе:

```
float x;  
int N;  
...  
x = BinSolve ( 2, 3, 0.0001, N );  
printf("Ответ: x = %7.3f", x);  
printf("Число шагов: %d", N);
```

Метод итераций (повторений)

Задача: $f(x) = 0$ $x = ?$

Эквивалентные преобразования:

$b \cdot f(x) = 0$ имеет те же решения при $b \neq 0$

$$x + b \cdot f(x) = x$$

$$x = \varphi(x), \quad \varphi(x) = x + b \cdot f(x)$$

Идея решения:

x_0 – начальное приближение (например, с графика)

$$x_k = \varphi(x_{k-1}) = x_{k-1} + b \cdot f(x_{k-1}), \quad k = 1, 2, \dots$$

Проблемы:

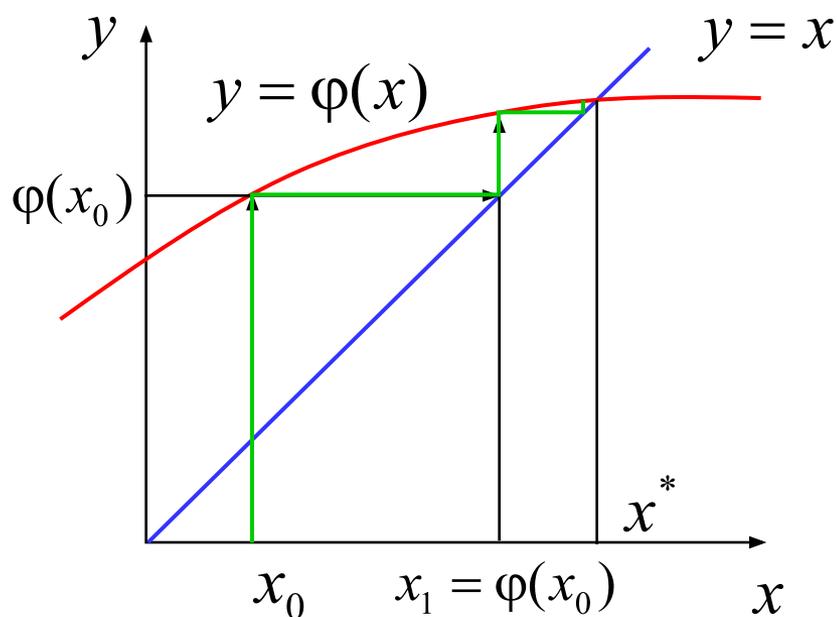
- 1) как лучше выбрать b ?
- 2) всегда ли так можно найти решение?

Сходимость итераций

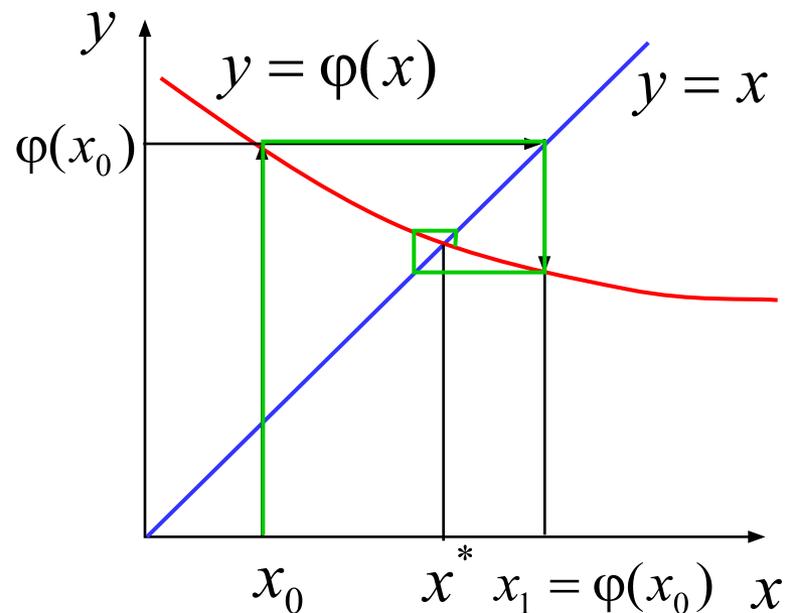
Сходящийся итерационный процесс:

последовательность x_0, x_1, \dots приближается (сходится) к точному решению.

$$x^* = \varphi(x^*) \quad x_0, x_1, x_2, \dots \rightarrow x^*$$



односторонняя сходимость

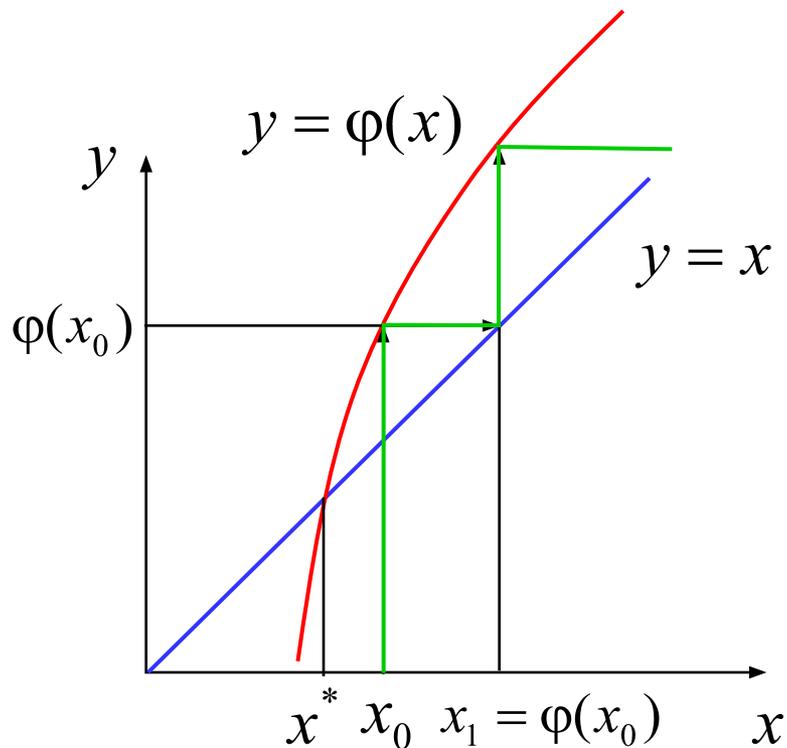


двусторонняя сходимость

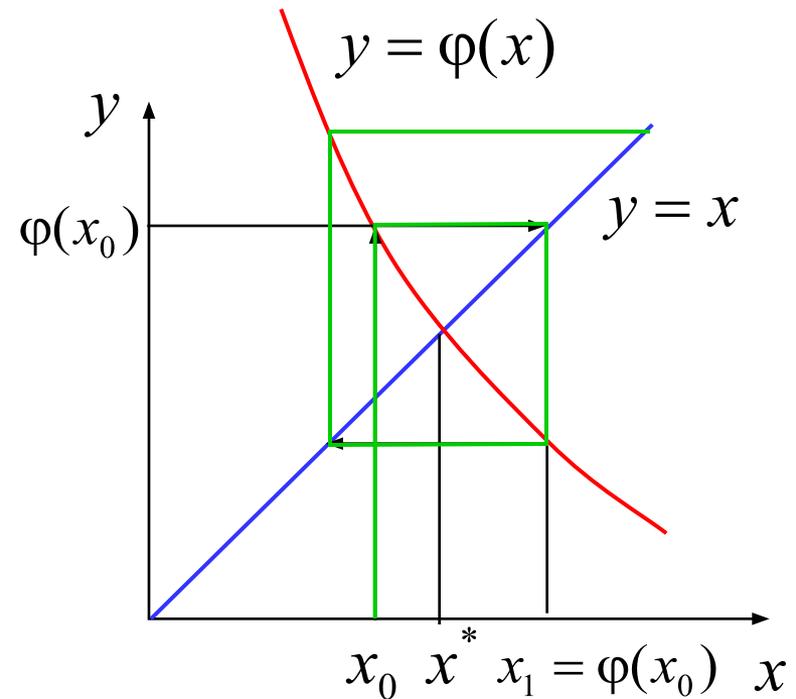
Расходимость итераций

Расходящийся итерационный процесс:

последовательность x_0, x_1, \dots неограниченно возрастает или убывает, не приближается к решению.



односторонняя расходимость

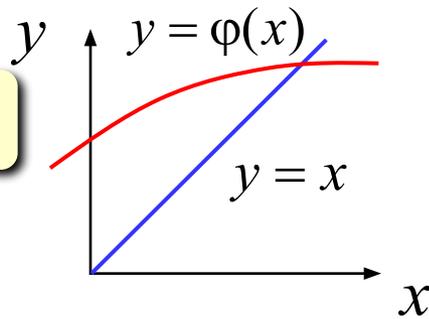


двусторонняя расходимость

От чего зависит сходимость?

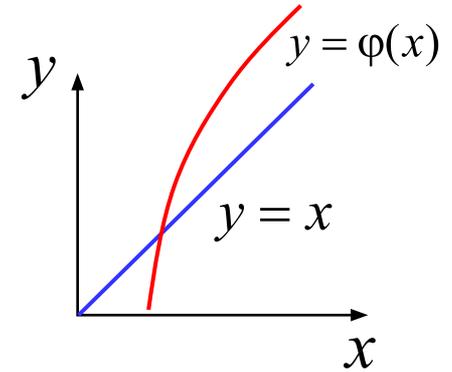
сходится

$$0 < \varphi'(x) < 1$$

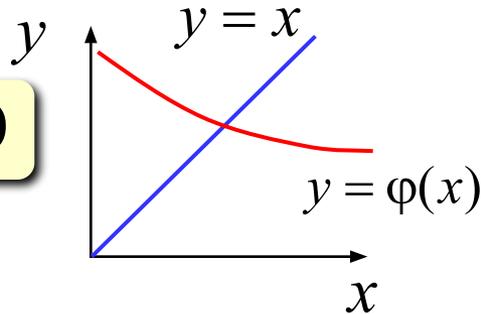


расходится

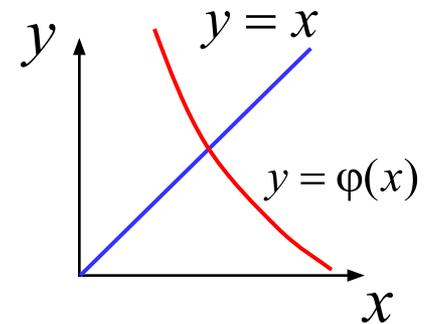
$$\varphi'(x) > 1$$



$$-1 < \varphi'(x) < 0$$



$$\varphi'(x) < -1$$



Выводы:

- сходимость итераций зависит от производной $\varphi'(x)$
- итерации сходятся при $|\varphi'(x)| < 1$ и расходятся при $|\varphi'(x)| > 1$
- сходимость определяется выбором параметра b

$$\varphi(x) = x + b \cdot f(x) \Rightarrow \varphi'(x) = 1 + b \cdot f'(x)$$

Как выбрать b ?

- наугад, попробовать разные варианты
- для начального приближения x_0

$$-1 < 1 + b \cdot f'(x_0) < 1 \quad \Rightarrow \quad -2 < b \cdot f'(x_0) < 0$$

$$f'(x_0) > 0 \quad \Rightarrow \quad -\frac{2}{f'(x_0)} < b < 0$$

$$f'(x_0) < 0 \quad \Rightarrow \quad 0 < b < -\frac{2}{f'(x_0)}$$

- пересчитывать на каждом шаге, например:

$$1 + b \cdot f'(x_k) = 0 \quad \Rightarrow \quad b = -\frac{1}{f'(x_k)}$$



Какие могут быть проблемы?

Метод итераций (программа)

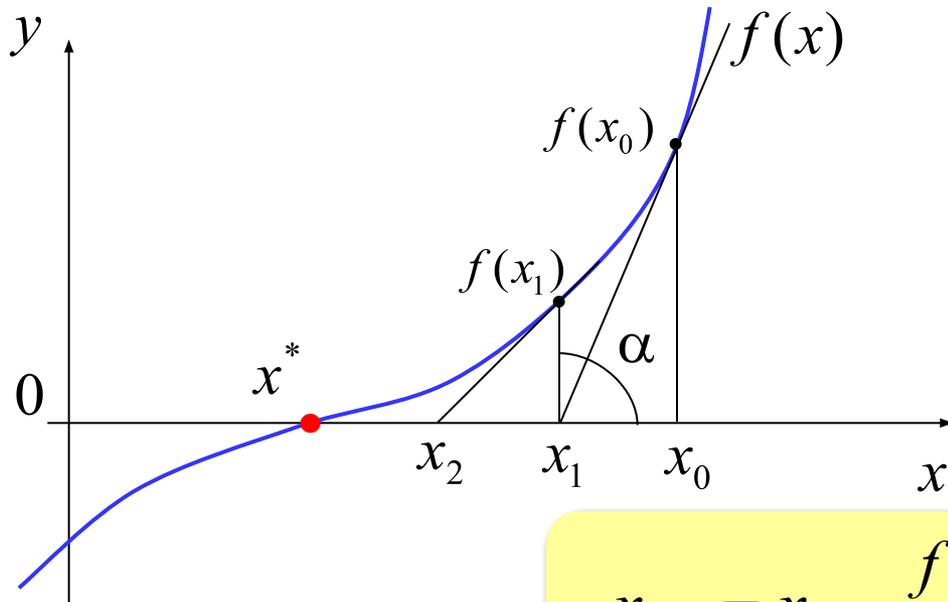
```
//-----
// Iter решение уравнения методом итераций
// Вход:  x - начальное приближение
//        b - параметр
//        eps - точность решения
// Выход: решение уравнения f(x)=0
//        n - число шагов
////-----
float Iter ( float x, float b, float eps, int &n)
{
    int n = 0;
    float dx;
    while ( 1 ) {
        dx = b*f(x);
        x = x + dx;
        if ( fabs(dx) < eps ) break;
        n ++;
        if ( n > 100 ) break;
    }
    return x;
}
```

$$x = x + b \cdot f(x)$$

нормальный
ВЫХОД

аварийный
ВЫХОД

Метод Ньютона (метод касательных)



$$\operatorname{tg} \alpha = \frac{f(x_0)}{x_0 - x_1}$$

$$\operatorname{tg} \alpha = f'(x_0)$$

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$



Какая связь с методом итераций?

$$x_k = x_{k-1} + b \cdot f(x_{k-1}) \quad \Rightarrow \quad b = -\frac{1}{f'(x_{k-1})}$$

Метод Ньютона (программа)

```
//-----  
// Newton решение уравнения методом Ньютона  
// Вход:  x - начальное приближение  
//        eps - точность решения  
// Выход: решение уравнения  $f(x)=0$   
//        n - число шагов  
/////-----  
float Newton ( float x, float eps, int &n)  
{  
    int n = 0;  
    float dx;  
    while ( 1 ) {  
        dx = f(x) / df(x);  
        x = x - dx;  
        if ( fabs(dx) < eps ) break;  
        n ++;  
        if ( n > 100 ) break;  
    }  
    return x;  
}
```

```
float f ( float x ) {  
    return 3*x*x*x+2*x+5;  
}  
float df ( float x ) {  
    return 9*x*x + 2;  
}
```

Метод Ньютона



- быстрая (квадратичная) сходимость – ошибка на k -ом шаге обратно пропорциональна k^2
- не нужно знать интервал, только начальное приближение
- применим для функция нескольких переменных



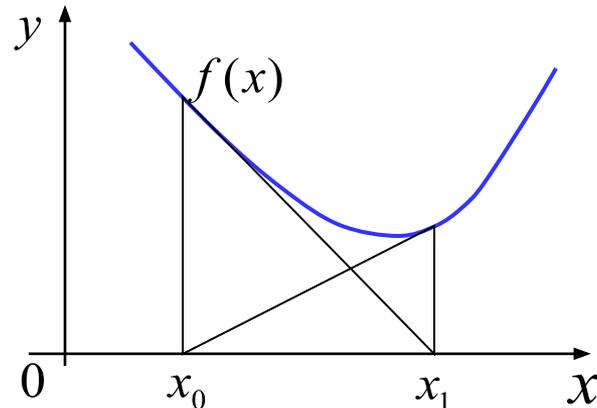
- нужно уметь вычислять производную (по формуле или численно)
- производная не должна быть равна нулю

$$x^3 = 0 \Rightarrow f'(x) = 3x^2$$

- может зацикливаться

$$f(x) = x^3 - 2x + 2$$

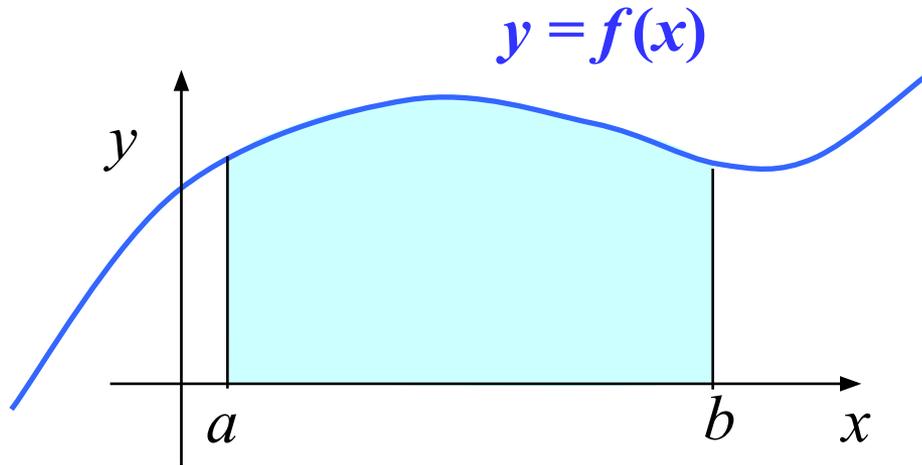
$$x_0 = 0$$



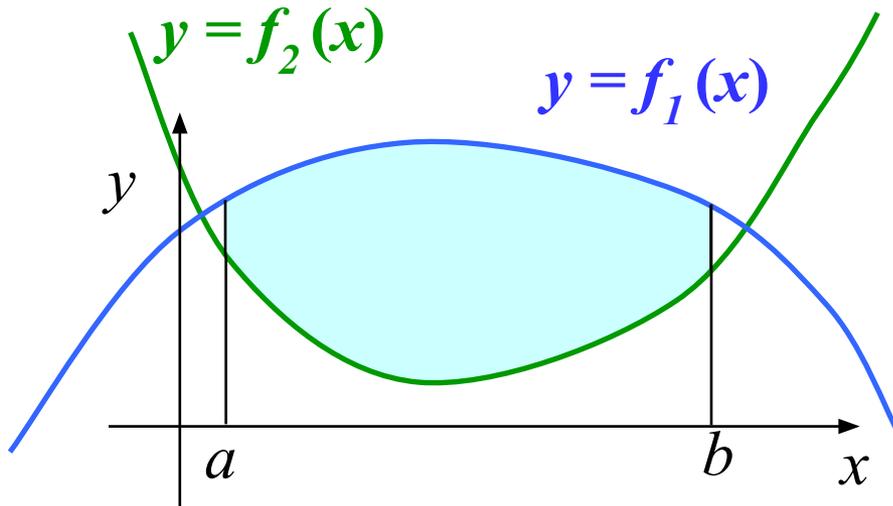
Численные методы

Тема 2. Вычисление площади (интеграла)

Площадь криволинейной трапеции

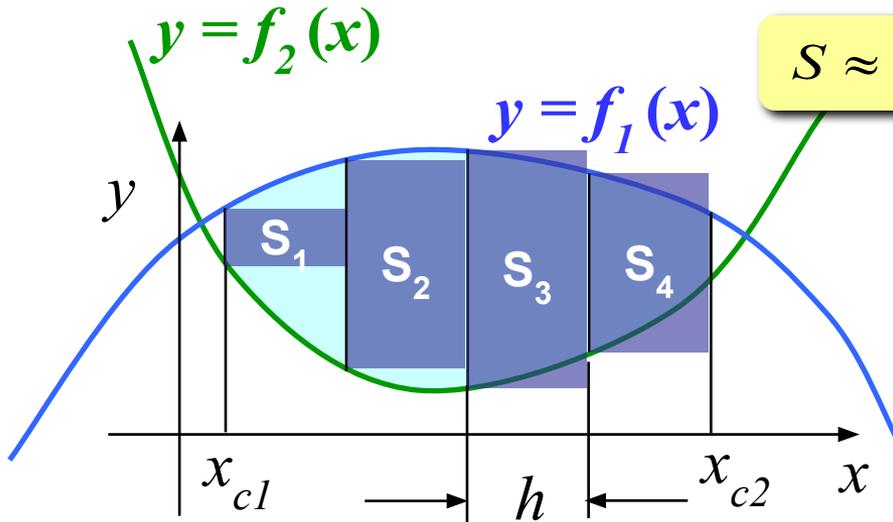


$$S = \int_a^b f(x) dx$$

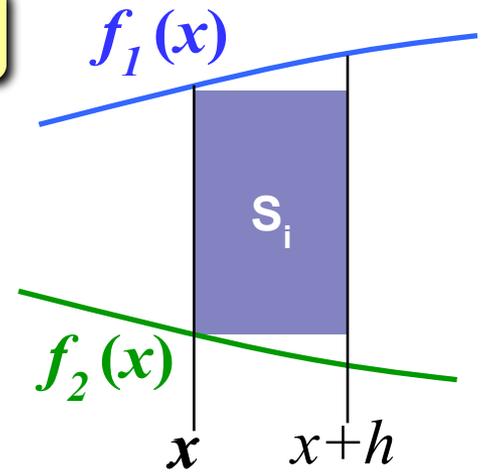


$$S = \int_a^b f_1(x) dx - \int_a^b f_2(x) dx$$

Метод (левых) прямоугольников



$$S \approx S_1 + S_2 + S_3 + S_4$$



```
float Area()
```

```
{
```

```
    float x, S = 0, h=0.001;
```

```
    for (x = xc1; x < xc2; x += h)
```

```
        for (x = xc1; x < xc2; x += h)
```

```
            re
```

```
                S += f1(x) - f2(x);
```

```
        S *= h;
```

```
}
```

$$S_i = (f_1(x) - f_2(x)) \cdot h$$

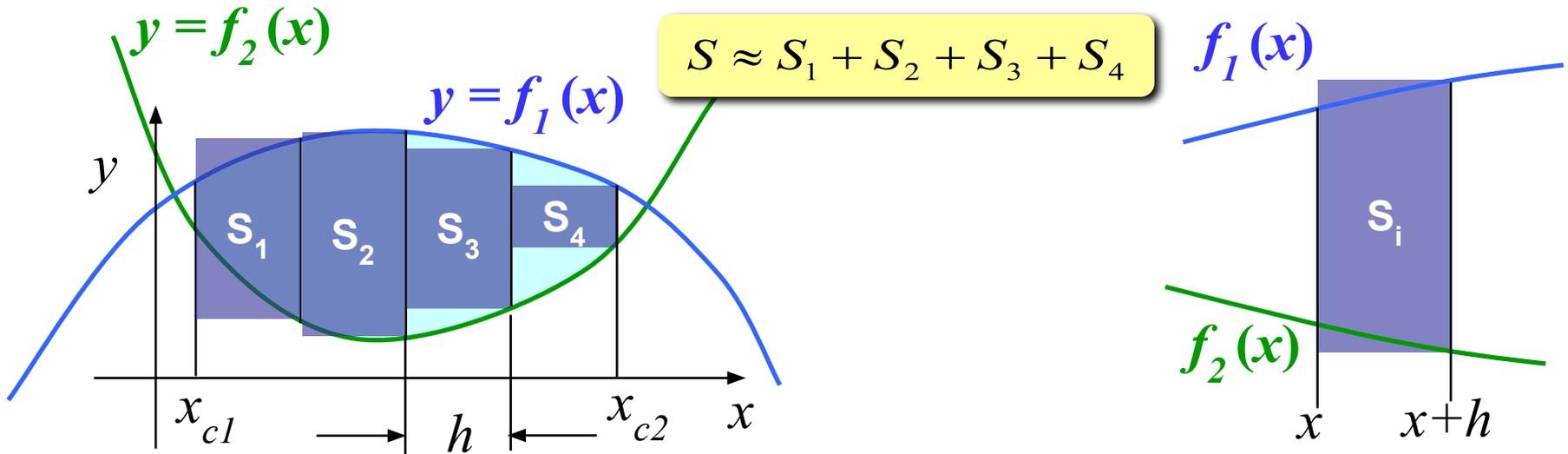


Почему не
 $x \leq x_{c2}$?



Как улучшить
решение?

Метод (правых) прямоугольников



```
float Area()
```

```
{
```

```
    float x, S = 0, h=0.001;
```

```
    for (x = xc1; x < xc2; x += h)
```

```
        S += f1(x+h) - f2(x+h);
```

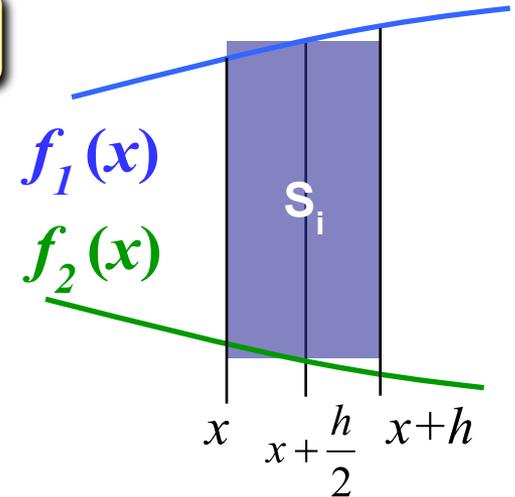
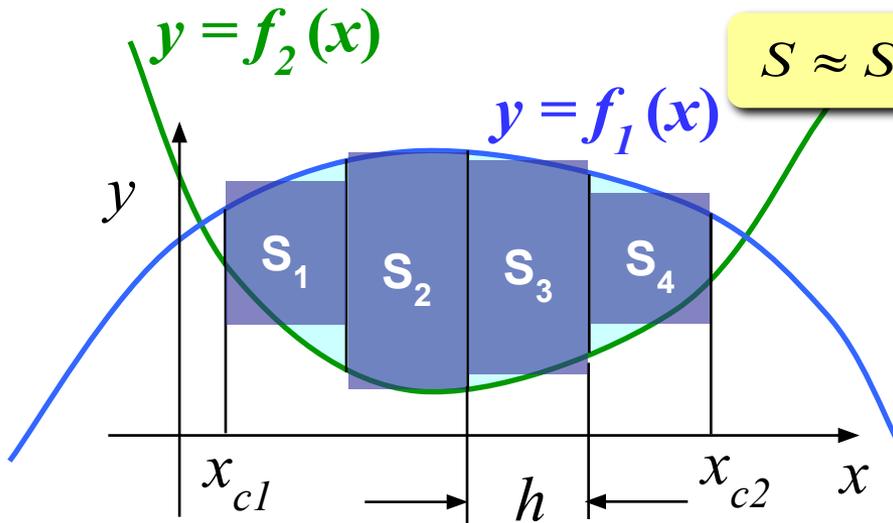
```
    S *= h;
```

```
    return S;
```

```
}
```

$$S_i = (f_1(x+h) - f_2(x+h)) \cdot h$$

Метод (средних) прямоугольников



$$S_i = \left[f_1\left(x + \frac{h}{2}\right) - f_2\left(x + \frac{h}{2}\right) \right] \cdot h$$

```
float Area()
```

```
{
```

```
    float x, S = 0, h=0.001;
```

```
    for (x = xc1; x < xc2; x += h)
```

```
        S += f1(x+h/2) - f2(x+h/2);
```

```
    S *= h;
```

```
    return S;
```

```
}
```



Какой метод точнее?

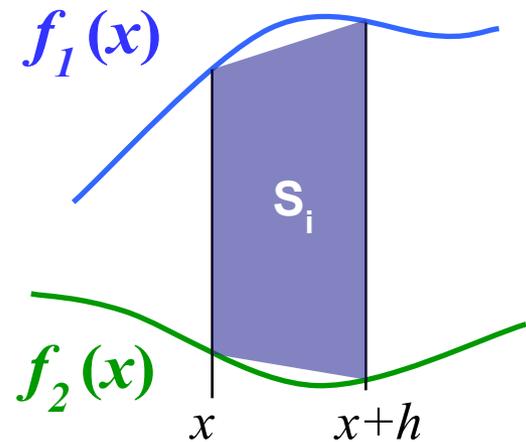
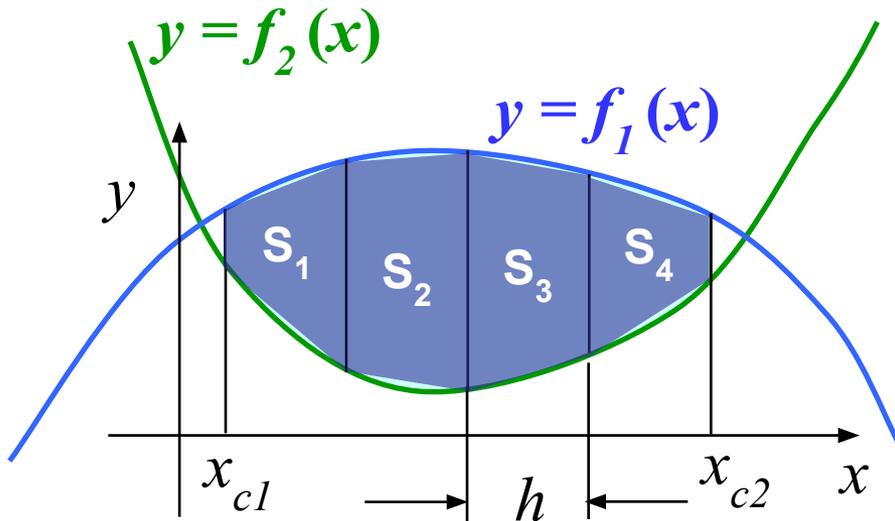
левые (правые):

$$\varepsilon = O(h)$$

средние

$$\varepsilon = O(h^2)$$

Метод трапеций



$$S_i = \frac{f_1(x) - f_2(x) + f_1(x+h) - f_2(x+h)}{2} \cdot h$$

```
for ( x = xc1; x < xc2; x += h )
```

```
S = ( f1(xc1) - f2(xc1)
      + f1(xc2) - f2(xc2) ) / 2.;
```

```
S
for ( x = xc1+h; x < xc2; x += h )
```

```
  S += f1(x) - f2(x);
```

```
S *= h;
```



Как улучшить?

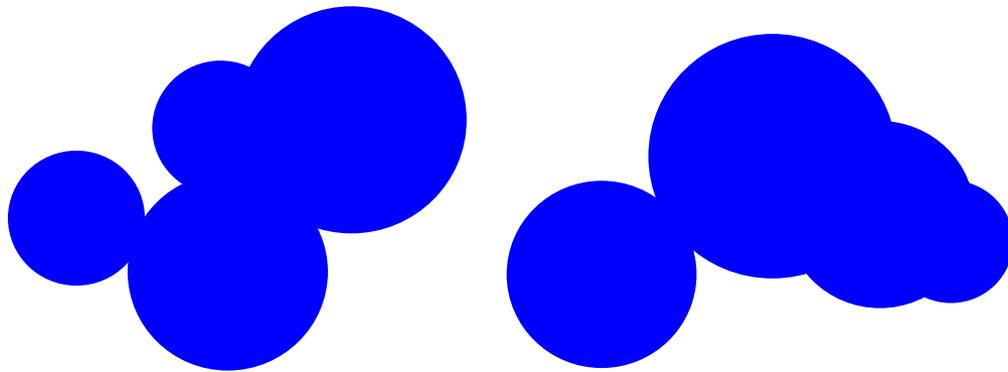
Ошибка $\varepsilon = O(h^2)$

Метод Монте-Карло

Применение: вычисление площадей сложных фигур (трудно применить другие методы).

Требования: необходимо уметь достаточно просто определять, попала ли точка (x, y) внутрь фигуры.

Пример: заданы 100 кругов (координаты центра, радиусы), которые могут пересекаться. Найти площадь области, перекрытой кругами.

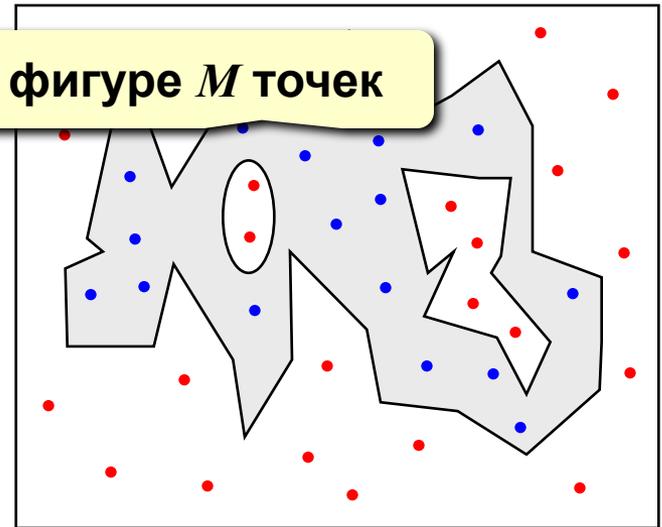


Как найти S ?

Метод Монте-Карло

1. Вписываем сложную фигуру в другую фигуру, для которой легко вычислить площадь (**прямоугольник**, круг, ...).
2. **Равномерно** N точек со случайными координатами внутри прямоугольника.
3. Подсчитываем количество точек, **попавших на фигуру**: M .
4. Вычисляем **площадь**: $\frac{S}{S_0} \approx \frac{M}{N} \Rightarrow S \approx S_0 \cdot \frac{M}{N}$

На фигуре M точек



Всего N точек

$$S \approx S_0 \cdot \frac{M}{N}$$

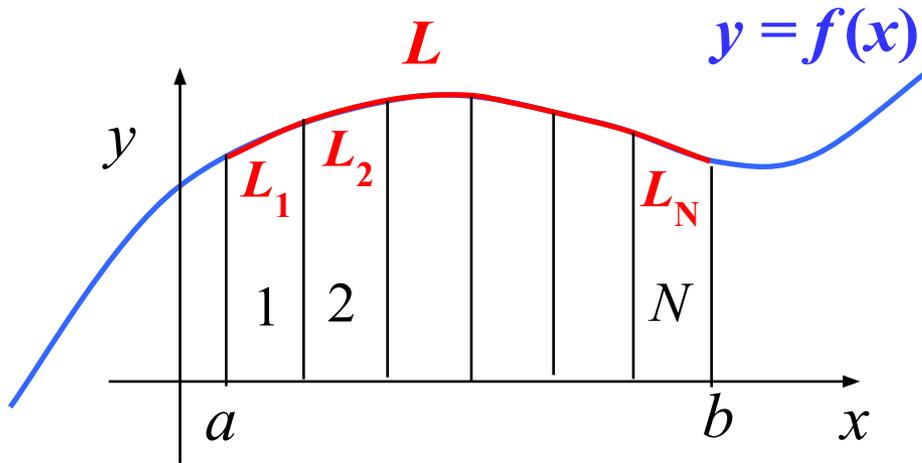


1. Метод приближенный.
2. Распределение должно быть равномерным.
3. Чем больше точек, тем точнее.
4. Точность ограничена датчиком случайных чисел.

Численные методы

Тема 3. Вычисление длины кривой

Длина кривой



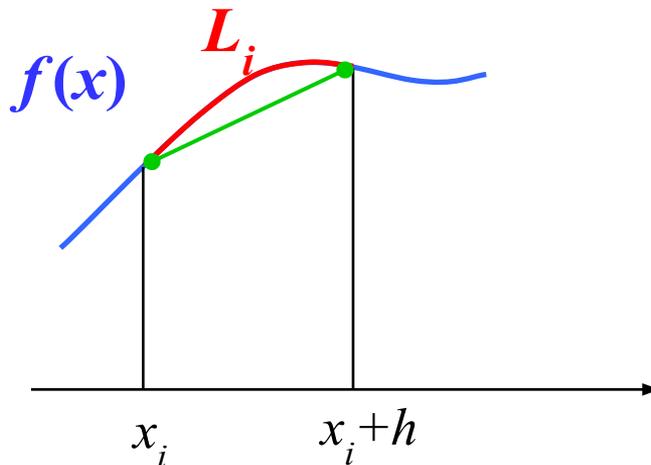
Точное решение:

$$L = \int_a^b \sqrt{1 + [f'(x)]^2} dx$$



- нужна формула для производной
- сложно взять интеграл

Приближенное решение:



$$L = L_1 + L_2 + \dots + L_N = \sum_{i=1}^N L_i$$

$$L_i \approx \sqrt{h^2 + [f(x_i + h) - f(x_i)]^2}$$

Длина кривой

```
//-----  
// CurveLen вычисление длины кривой  
// Вход:  a, b - границы интервала  
// Выход: длина кривой  $y = f(x)$  на интервале [a,b]  
//-----  
float CurveLen ( float a, float b )  
{  
    float x, dy, h=0.0001, h2 = h*h, L = 0;  
    for ( x=a; x<b; x+=h ) {  
        dy = f(x+h) - f(x);  
        L += sqrt(h2 + dy*dy);  
    }  
    return L;  
}
```

Численные методы

Тема 4. Оптимизация

Основные понятия

Оптимизация – поиск оптимального (наилучшего в некотором смысле) решения.

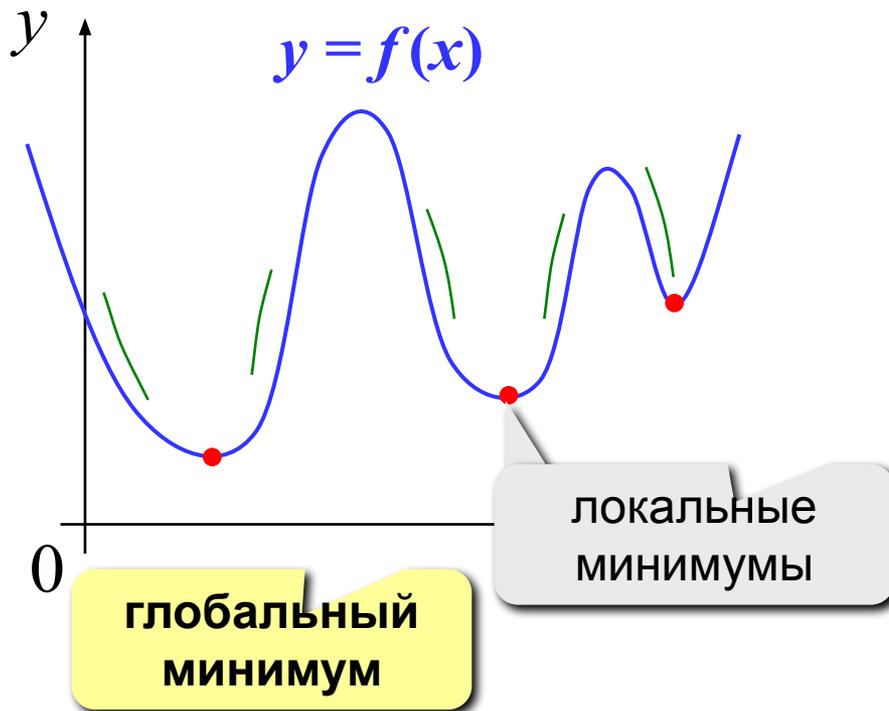
Цель: определить значения неизвестных параметров, при которых заданная функция достигает **минимума** (затраты) или **максимума** (доходы).

$$f(x) \rightarrow \min \quad \text{или} \quad f(x) \rightarrow \max$$

Ограничения – условия, которые делают задачу осмысленной.

Найти x , при котором $f(x) \rightarrow \min$ или $f(x) \rightarrow \max$ при заданных ограничениях.

Локальные и глобальные минимумы



Задача: найти глобальный минимум.

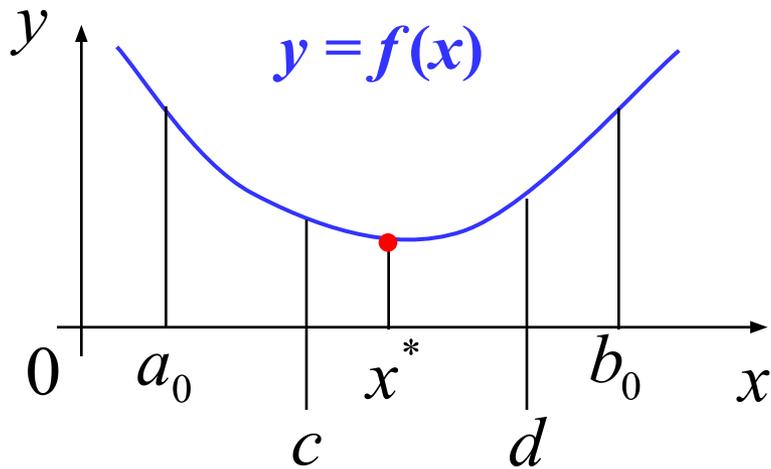
Реальность:

- большинство известных алгоритмов находят только *локальный* минимум вблизи начальной точки
- алгоритмы поиска *глобального* минимума в общем случае неизвестны

Что делать:

- для функций **одной переменной** начальная точка определяется по **графику**
- случайный выбор начальной точки
- запуск алгоритма поиска с нескольких разных точек и выбор наилучшего результата

Минимум функции одной переменной



Дано: на интервале $[a, b]$ функция непрерывна и имеет единственный минимум.

Найти: x^*

Принцип сжатия интервала:

$$[a_0, b_0] \rightarrow [a_1, b_1] \rightarrow \dots \rightarrow [a_n, b_n]$$

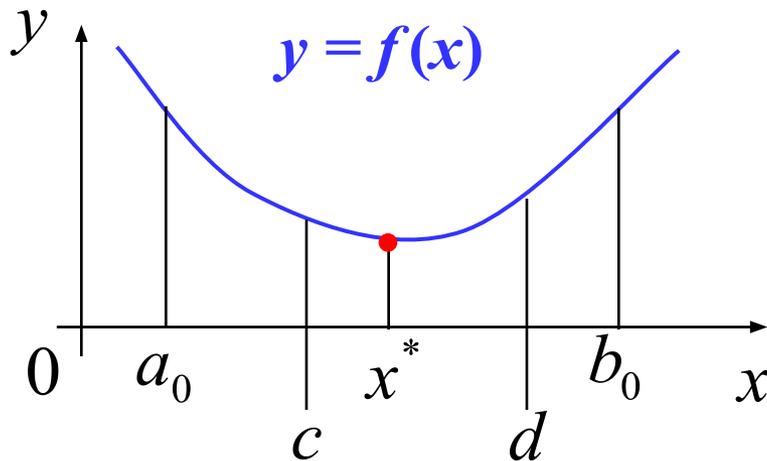
$$c < d \quad f(c) < f(d) \quad \Rightarrow [a_0, d]$$

$$f(c) > f(d) \quad \Rightarrow [c, b_0]$$



Как выбрать c и d наилучшим образом?

Минимум функции одной переменной



Постоянное сжатие в обоих случаях:

$$d - a_0 = b_0 - c$$

Коэффициент сжатия:

$$\mu = \frac{d - a_0}{a_0 - b_0} = \frac{b_0 - c}{a_0 - b_0} \rightarrow \min$$

Самое быстрое сжатие:

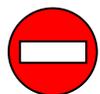
$$\mu = 0,5 \quad \text{при} \quad c = d = \frac{a_0 + b_0}{2}$$

ДОЛЖНО БЫТЬ $c \neq d$

Метод «почти половинного» деления:

$$c = \frac{a_0 + b_0}{2} - \varepsilon, \quad d = \frac{a_0 + b_0}{2} + \varepsilon$$

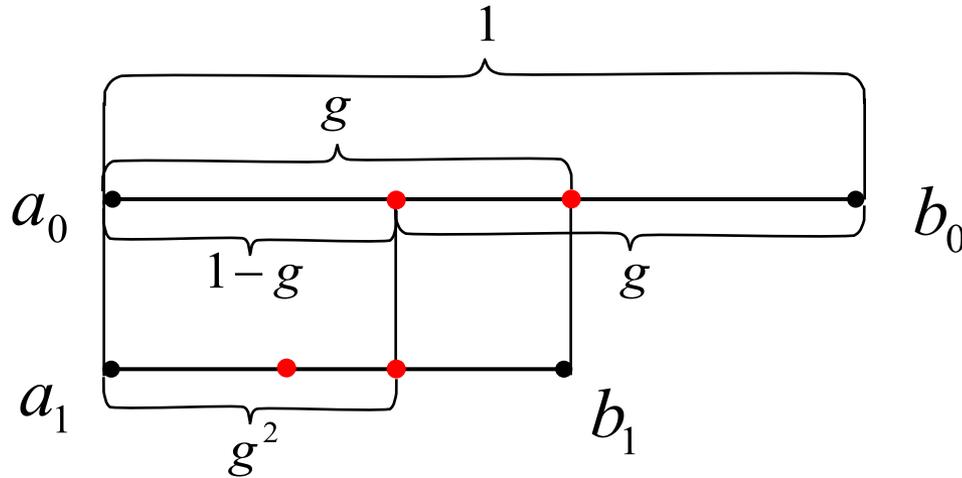
ε – малое число



нужно искать два значения функции на каждом шаге

Отношение «золотого сечения»

Идея: выбрать c и d так, чтобы на каждом шаге вычислять только одно новое значение функции.



Уравнение для определения g :

$$1 - g = g^2 \Rightarrow g = \frac{1 + \sqrt{5}}{2} \approx 0,618$$

Отношение «золотого сечения»:

$$g \approx 0,618$$



Метод «золотого сечения»

```
//-----  
// Gold поиск минимума функции («золотое сечение»)  
// Вход:  a, b – границы интервала  
//        eps – точность  
// Выход: x, при котором f(x) имеет минимум  
//        на интервале [a,b]  
//-----  
float Gold (float a, float b, float eps )  
{  
float x1, x2, g=0.618034, R=g*(b - a);  
while ( fabs(b-a) > eps ) {  
    x1 = b - R; x2 = a + R;  
    if ( f(x1) > f(x2) ) a = x1;  
    else                b = x2;  
    R *= g;  
}  
return (a + b) /2.;  
}
```



Как вычислять только одно значение на каждом шаге?

Функции нескольких переменных

Найти $\{x_1, x_2, \dots, x_n\}$, для которых $f(x_1, x_2, \dots, x_n) \rightarrow \min$ при заданных ограничениях.

Проблемы:

- нет универсальных алгоритмов поиска глобального минимума
- неясно, как выбрать начальное приближение (зависит от задачи и интуиции)

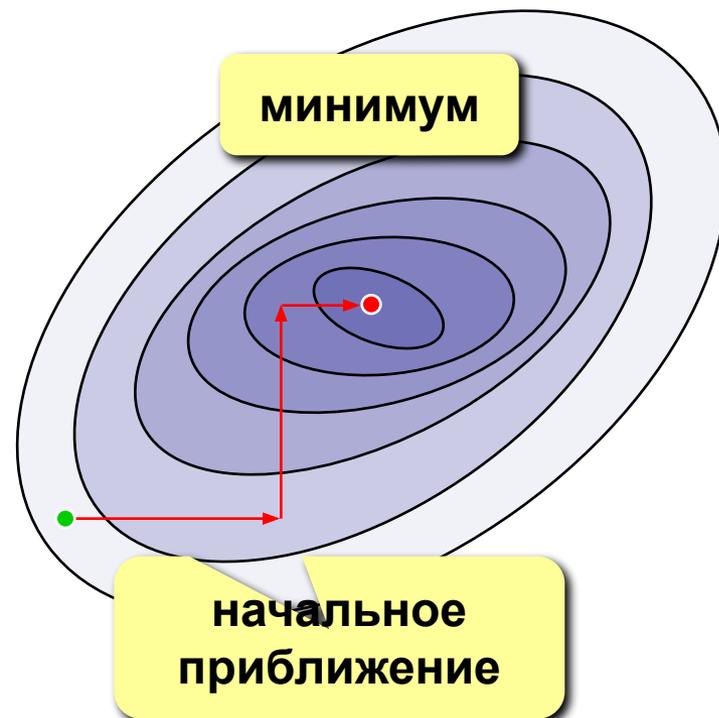
Подходы:

- методы локальной оптимизации (результат зависит от выбора начального приближения)
- случайный поиск (без гарантии)
- методы глобальной оптимизации (для особых классов функций)

Метод покоординатного спуска

Идея:

- выбираем начальную точку
- будем менять только x_1 , а остальные переменные «заморозим», находим минимум по x_1
- теперь будем менять только x_2 , а остальные переменные «заморозим», ...



- ⊕ • простота, сводится к нескольким задачам с одной переменной
- ⊖ • можно двигаться к минимуму быстрее
- большой объем вычислений
- может не найти решение для сложных функций

Градиентные методы

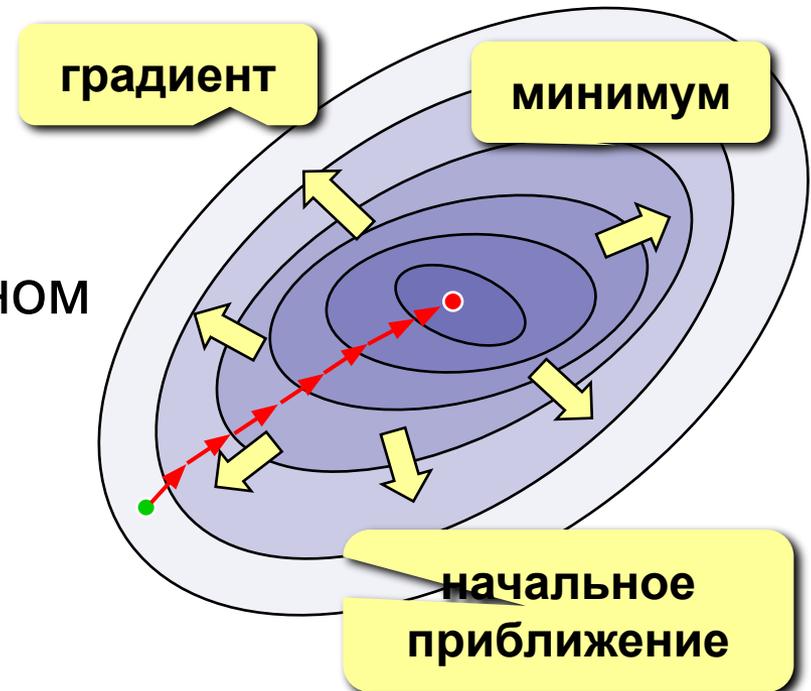
Градиент – это вектор, показывающий направление наискорейшего возрастания функции.

Идея:

- выбираем начальную точку
- на каждом шаге движемся в направлении, противоположном градиенту

 • быстрая сходимость

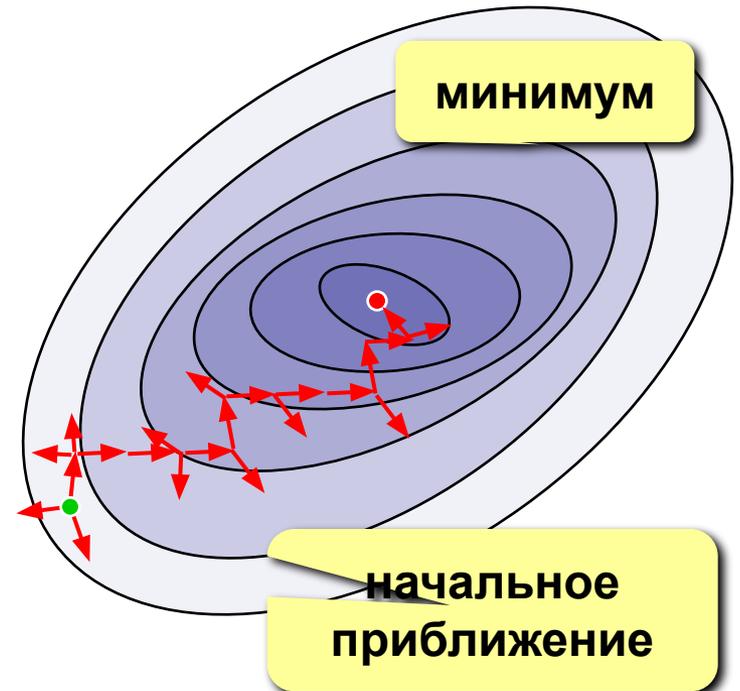
-  • необходимо считать производные (по формуле или численно)
- плохо работает для быстро меняющихся функций



Метод случайного поиска

Идея:

- выбираем начальную точку
- пробуем сделать шаг в случайном направлении
- если значение функции уменьшилось, шаг удачный (запоминается)



- простота реализации
- не требует вычисления производных
- много вариантов с самообучением
- хорошо работает для функций с многими локальными минимумами



- очень большой объем вычислений

Конец фильма
