



# ЧИСЛЕННЫЕ МЕТОДЫ

к.ф.-м.н., доц. Чернышенко Всеволод Сергеевич  
[VSChernyshenko@fa.ru](mailto:VSChernyshenko@fa.ru)  
- 2025 -



1. Теория погрешностей.
2. Методы решения нелинейных уравнений.
3. Методы интерполяции и экстраполяции функций.
4. Численное интегрирование функций.
5. Определение собственных чисел и собственных векторов матрицы.
6. Приближенное решение обыкновенных дифференциальных уравнений.
7. Ряды Фурье

1. В. И. Киреев, А. В. Пантелейев. Численные методы в примерах и задачах: Учебное пособие. – 4-е изд., испр. — СПб.: Издательство «Лань», 2022. – 448 с.
2. А. В. Зенков. Численные методы: учебное пособие для вузов – Москва : Юрайт, 2023. – 122 с. – (Высшее образование). – ЭБС Юрайт. – URL: <https://urait.ru/bcode/513646>
3. П.Н. Вабищевич. Численные методы: Вычислительный практикум. Практическое применение численных методов при использовании алгоритмического языка PYTHON URSS. 2022. 320 с.



Модуль	Посещение лекций и семинаров	Домашние задания	Контрольная работа	Бонусные баллы
I	max 5	max 5	max 10	$\rightarrow \infty$
II	max 5	max 15		$\rightarrow \infty$

Экзамен

Онлайн-консультации – как только сформируете запрос.



- **Первый этап математического анализа** – создание математической модели (постановка задачи). Для физического процесса модель обычно состоит из уравнений, описывающих процесс.
- **Второй этап** – математическое исследование. В зависимости от сложности модели применяются различные математические методы. Часто применяются численные методы.
- **Третий этап** – осмысление математического решения и сопоставление его с экспериментальными данными. Проверка адекватности математической модели.



- Одной из важнейших задач в процессе математического моделирования является **вычисление значений функций**. Иными словами, помимо разработки математической модели, требуется еще разработка алгоритма, сводящего все вычисления к последовательности арифметических и логических действий.
- Выбирать модель и алгоритм требуется с учетом скорости и объема памяти компьютера.

Область применения численных методов – решение тех задач математического анализа, для которых аналитическое (точное) решение затруднено или невозможно

Примеры:

- «неберущиеся» интегралы (нет первообразных функций)
- Математические задачи, требующие больших затрат времени и другие

## АНАЛИТИЧЕСКИЕ

Теоретические рассуждения и выводы. Рассматриваются в курсе математики, физики и иных наук.

Конечный результат: Формулы, системы уравнений.

### ПРЕИМУЩЕСТВА

- Вычисления по конечным формулам
- Можно строить графики
- Решить доп. теоретические задачи

### НЕДОСТАТКИ

- Приближения при выводе формул
- Отсутствие методов решения систем уравнений некоторого вида
- Трудности проведения вычислений по формулам



## ГРАФИЧЕСКИЕ

Построение графиков, диаграмм, запись измерений с помощью датчиков.

Конечный результат: Графики и точки на графиках.

## ПРЕИМУЩЕСТВА

- Наглядное представление о поведении исследуемой величины
- Позволяет оценить приближенное значение некоторой величины
- Можно составить таблицу значений

## НЕДОСТАТКИ

- Трудности проведения дополнительных теоретических исследований

## ЧИСЛЕННЫЕ

Решение задачи сводится к вычислению в определенной последовательности.

Конечный результат: Число или числа.

### ПРЕИМУЩЕСТВА

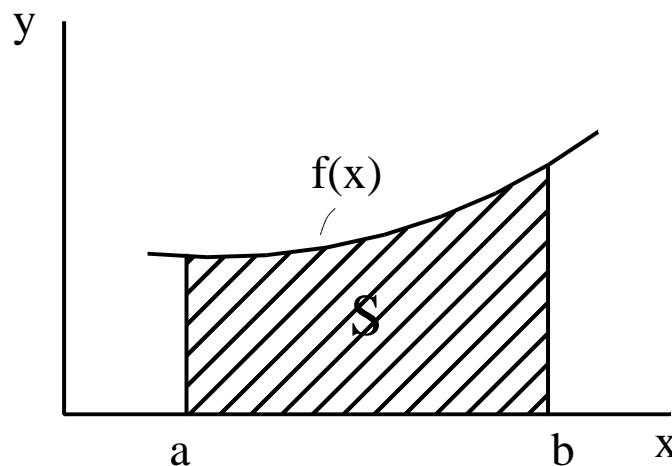
- Решение задач, для которых нет аналитических методов

### НЕДОСТАТКИ

- Вычисления содержат погрешности
- Не для всех задач есть численные методы
- Вычисления могут занимать много времени



- Все методы численного интегрирования основаны на геометрической интерпретации интеграла как площади фигуры, ограниченной осью абсцисс, подынтегральной кривой и прямыми  $x = a$ ,  $x = b$ , где  $a$  и  $b$  – пределы интегрирования



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



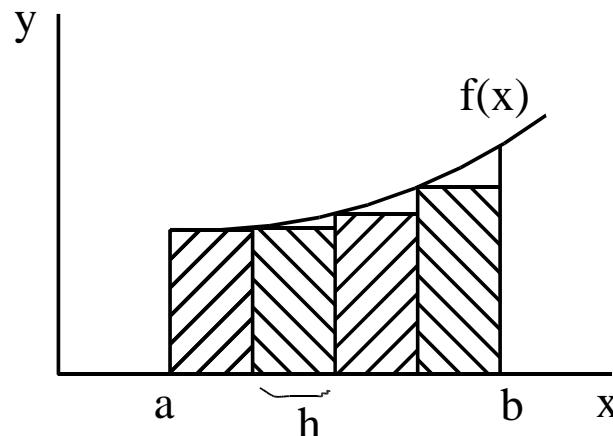
## Формула прямоугольников

- Даны пределы интегрирования. Зададим число элементарных фигур  $N$ , на которое будет разбита вся площадь. Находим шаг интегрирования:

$$h = (b - a) / N$$

- Найдем определенный интеграл как сумму площадей элементарных фигур (прямоугольников), на которые разбита площадь под подынтегральной кривой

$$S = \sum_{i=1}^N S_i = f(a) * h + f(a + h) * h + \dots + f(b - h) * h$$



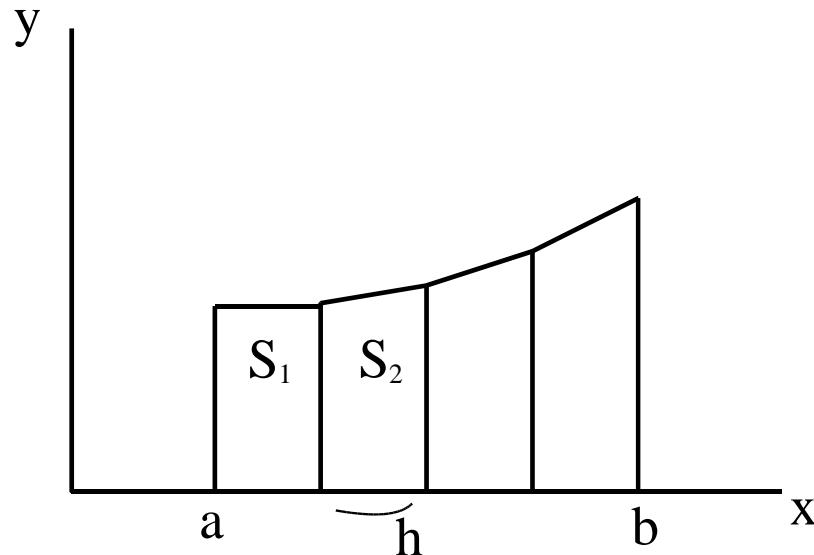
$$S = h * \sum_{x=a}^{b-h} f(x)$$



## Формула трапеций

- Найдем определенный интеграл как сумму площадей элементарных фигур (трапеций), на которые разбита площадь под подынтегральной кривой:

$$S = \sum_{i=1}^N S_i \quad S_1 = \frac{h}{2} [f(a) + f(a+h)] \quad S_2 = \frac{h}{2} [f(a+h) + f(a+2h)]$$



$$S = \left[ \frac{f(a) + f(b)}{2} + \sum_{x=a+h}^{b-h} f(x_i) \right] \times h$$

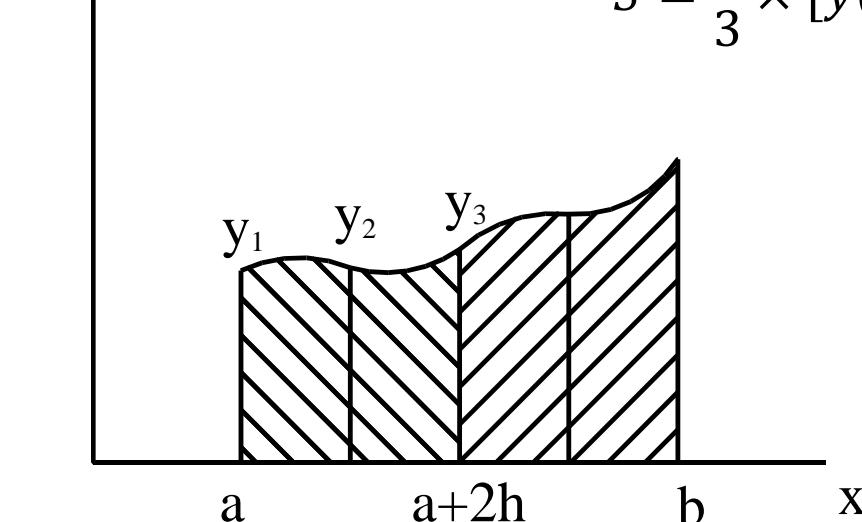


## Формула Симпсона

- Найдем определенный интеграл как сумму площадей элементарных фигур (криволинейных трапеций), на которые разбита площадь под подынтегральной кривой. Площадь криволинейной параболы определяется по формуле:

$$S_i = \frac{h}{3} (y_i + 4y_{i+1} + y_{i+2}) \quad S = \sum_{i=1}^N S_i$$

$$S = \frac{h}{3} \times [y(a) + y(b) + \sum_{x=a+h}^{b-h,2h} 4f(x) + \sum_{x=a+2h}^{b-2h,2h} 2f(x)]$$





1. Вычисления с помощью ручки и бумаги можно проводить с любой степенью точности.
2. В компьютере числа хранятся в ячейках памяти с фиксированной разрядностью не более 15 цифр. Ограничен диапазон представления чисел:  $10^{-307} < |x| < 10^{307}$ .
3. Компьютерные вычисления могут содержать миллионы операций, что приводит к накоплению ошибки.
4. Компьютерная арифметика связана с представлением чисел в CPU/GPU и отличается от обычной.



- Постановка задачи (исходные данные и определение конечного результата исследования).
- Построение модели (модель должна адекватно описывать законы физического явления).
- Разработка численного метода (нахождение метода, позволяющего свести задачу к вычислительному алгоритму).

Все численные методы являются ПРИБЛИЖЕННЫМИ, т.е. решение всегда находится с некоторой погрешностью  $\varepsilon$ .



# ТЕОРИЯ ПРИБЛИЖЁННЫХ ВЫЧИСЛЕНИЙ

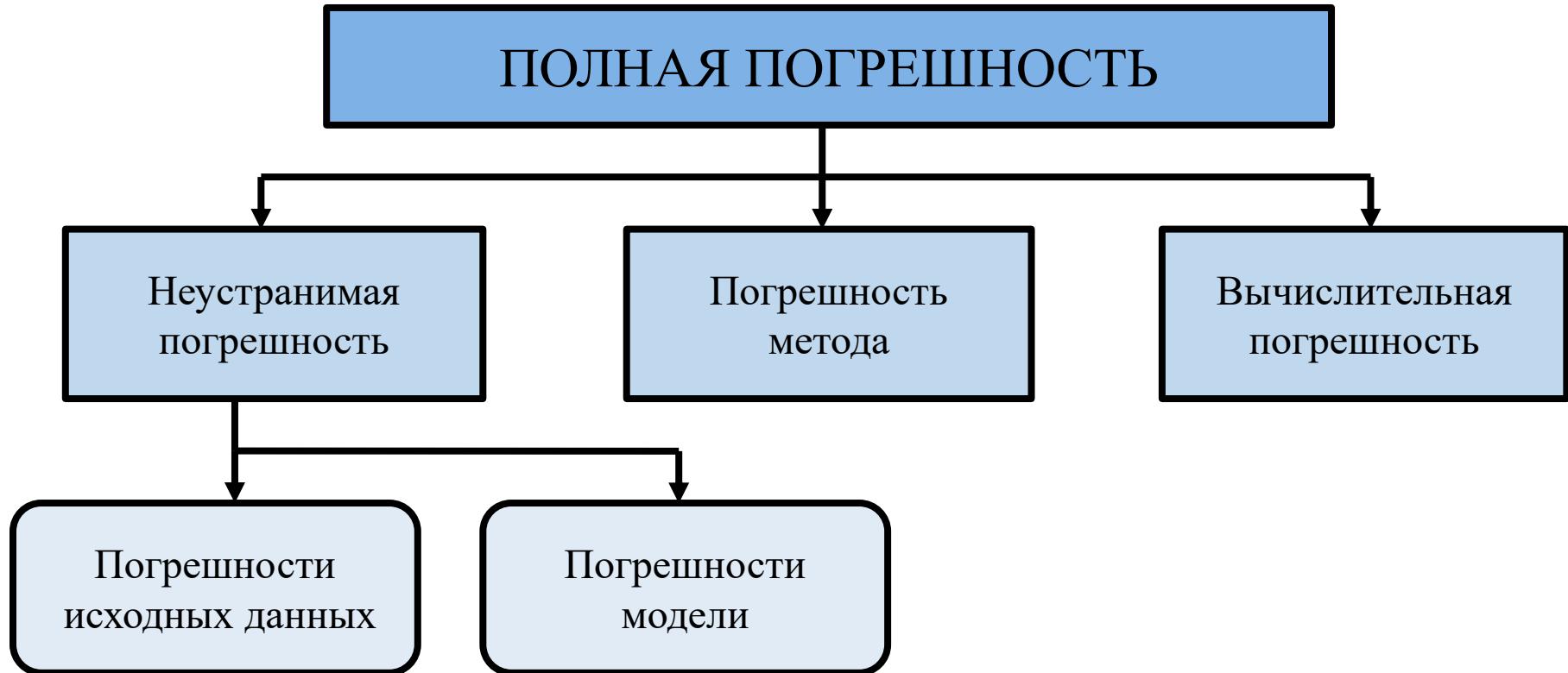


Погрешность решения задачи обуславливается следующими причинами:

1. Математическое описание задачи является неточным, в частности, неточно заданы исходные данные описания.
2. Применяемый для решения метод часто не является точным: получение точного решения требует неограниченного или неприемлемо большого числа арифметических операций, и поэтому вместо получения точного решения задачи приходится прибегать к приближенному.
3. При вводе данных в машину, при выполнении арифметических операций и при выводе данных производятся округления.

Погрешности, соответствующие этим причинам, называют:

- a) неустранимой погрешностью;
- b) погрешностью метода;
- c) вычислительной погрешностью.



Часто **неустранимую погрешность** подразделяют на две части:

- 1.1. Погрешности, являющиеся следствиями **неточности задания числовых данных**, входящих в математическое описание задачи;
- 1.2. Погрешности, являющуюся следствиями **несоответствия математического описания задачи реальности**, т.е. погрешностями математической модели.



**Приближенным значением** некоторой величины  $a$  называется число  $a_p$ , которое незначительно отличается от точного значения этой величины.

**Абсолютной погрешностью**  $\Delta$  приближенного значения называется модуль разности между точным и приближенным значениями этой величины  $\Delta = |a - a_p|$ .

**Относительной погрешностью** приближенной величины  $a_p$  называется отношение абсолютной погрешности приближенной величины к абсолютной величине ее точного значения:

$$\delta = \frac{|a - a_p|}{|a|} = \frac{\Delta}{|a|}.$$



Под предельной абсолютной погрешностью приближенного числа понимается всякое число не меньшее абсолютной погрешности этого числа  $\Delta_a$

$$\Delta = |a - a_P| \leq \Delta_a$$

$a$	$a_P$	$\Delta$	$\Delta_a$
2,718281828459045	2,72	0,001718172	$2,720 - 2,718 = 0,002$ $2,7\underline{2} - 2,718 < 0,01$

Так как  $X$  и  $\Delta$ , как правило, неизвестны, но  $X \approx x$ , и в качестве  $\delta x$  обычно берут отношение  $\Delta x / |x|$ . Отсюда получаем формулы связи  $\delta x$  и  $\Delta x$ :

$$\delta = \frac{\Delta x}{|x|}; \quad \Delta x = |x| \cdot \delta x$$



Так как точное значение  $X$  обычно бывает неизвестно, то и  $\Delta$  - тоже неизвестно. Вместо теоретических понятий абсолютной и относительной погрешностей используют практические понятия предельной абсолютной погрешности и предельной относительной погрешности.

Для краткости их называют просто "относительными погрешностями". Относительную погрешность допускается выражать в процентах.

Процесс вычисления абсолютной и относительной погрешностей обязательно следует завершать их округлением, в соответствии с правилами округления погрешностей.



**Неустранимая погрешность** возникает при погрешности исходных данных, она неконтролируема в процессе численного решения задачи и может уменьшиться только за счет более точного описания задачи и более точного определения параметров.

**Погрешность метода** связана с тем, что исходные данные заменяются приближенными. Например, заменяют интеграл суммой, производную – разностью, функцию – многочленом или строят бесконечный итерационный процесс и обрывают его после конечного числа итераций. Эту погрешность можно регулировать. Погрешность метода целесообразно выбирать так, чтобы она была в 2-5 раз меньше неустранимой погрешности. Большая погрешность метода снижает точность ответа, а заметно меньшая - невыгодна, т.к. обычно требует значительного увеличения объема вычислений.

Вычисления выполняют с определенным числом значащих цифр. Это вносит в ответ погрешность округления, которая накапливается в ходе вычислений (**вычислительная погрешность**).



Значащими цифрами в записи приближенного числа называются:

- все ненулевые цифры;
- нули, содержащиеся между ненулевыми цифрами;
- нули, являющиеся представителями сохраненных десятичных разрядов при округлении.

*Пример: 2,305; 0,0357; 0,001123; 0,035299879  $\approx$  0,035300*



Чтобы округлить число до  $n$  значащих цифр, отбрасывают все цифры, стоящие справа от  $n$ -й значащей цифры, или, если это нужно для сохранения разрядов, заменяют их нулями. При этом:

1. если первая отброшенная цифра меньше 5, то оставшиеся десятичные знаки сохраняют без изменения;
2. если первая отброшенная цифра больше 5, то к последней оставшейся цифре прибавляют единицу;
3. если первая отброшенная цифра равна 5 и среди остальных отброшенных цифр есть ненулевые, то к последней оставшейся цифре прибавляют единицу;
4. если первая из отброшенных цифр равна 5 и все отброшенные цифры являются нулями, то последняя оставшаяся цифра оставляется неизменной, если она четная, и увеличивается на единицу, если нет (правило четной цифры). Это правило гарантирует, что сохраненные значащие цифры числа являются верными в узком смысле, т. е. погрешность округления не превосходит половины разряда, соответствующего последней оставленной значащей цифре. Правило четной цифры должно обеспечить компенсацию знаков ошибок.



Если положительное приближенное число имеет  $n$  верных значащих цифр, то его относительная погрешность  $\delta$  не превосходит величины  $10^{1-n}$ , деленной на первую значащую цифру  $a_H$ .

$$\delta \leq 10^{1-n} / a_H$$

**Пример:** Найти относительную и абсолютную погрешности приближенных чисел:  
a) 3,142 ; b)  $2,997925 \cdot 10^8$  .

Решение.

a) Здесь  $n = 4$ ,  $a_H = 3$ . Подсчитаем относительную погрешность:

$$\delta_a = 10^{1-n} / a_H = 0,001/3 \approx 0,00033.$$

Для определения абсолютной погрешности:

$$\Delta_a \approx |a_p| \delta_a = 3,142 \cdot 0.00033 \approx 0,001.$$

b) Аналогично вычислим:  $n = 7$ ,  $a_H = 2$ ,

$$\delta_a = 10^{1-n} / a_H = 0,000001/2 = 0,0000005;$$

$$\Delta_a \approx |a_p| \delta_a = 2,997925 \cdot 10^8 - 0,0000005 \approx 150.$$



Цифра числа называется **верной в строгом смысле** (в узком смысле), если абсолютная погрешность этого числа не превосходит половины единицы разряда, в котором стоит эта цифра. Цифра числа называется **верной в широком смысле**, если абсолютная погрешность этого числа не превышает единицы разряда, в котором стоит эта цифра.

**Пример.** Исследовать, какие цифры числа  $46,852 \pm 0,007$  являются верными в строгом (узком) смысле.

Будем последовательно брать цифры числа 46,852.

4: число стоит в разряде десятков, следовательно, по определению необходимо взять половину этого разряда, то есть  $10/2 = 5$  и результат сравнить с 0,007, так как  $5 > 0,007$ , то цифра 4 является верной.



**Пример.** Исследовать, какие цифры числа  $46,852 \pm 0,007$  являются верными в строгом (узком) смысле.

6: повторяем аналогичные действия, учитывая, что цифра 6 находится в разряде единиц  $1/2 = 0.5 > 0.007$ , то есть цифра 6 является верной:

8:  $0.1/2 = 0.05 > 0.007$ , то есть цифра 8 является верной:

5:  $0,01/2 = 0.005 < 0.007$ , то есть цифра 5 не является верной, а, следовательно, и следующая цифра 2 тоже не верная.

Таким образом, от первоначальной записи числа  $46,852 \pm 0.007$ , возможно перейти к использованию значения 46.8, отметив, что все цифры этого числа верны в узком смысле.



**Пример.** Пусть  $a = 2,91385$ ,  $\Delta a = 0.0097$ . В числе  $a$  верны в широком смысле цифры 2, 9, 1.

**Пример.** Если в числе  $a = 6,92$  все цифры верны в строгом смысле, то  $\Delta a = 0.005$ . Запись  $b = 4,1$  при условии, что все цифры верны в широком смысле, подразумевает, что  $\Delta b = 0,1$ . но по записи  $c = 4,100$  мы можем заключить, что  $\Delta c = 0.001$ . Таким образом, записи 4,1 и 4,100 в теории приближенных вычислений означают не одно и то же.

Говорят, что приближенное данное записано **правильно**, если в его записи все цифры верные. Если число записано правильно, то по одной его записи в виде десятичной дроби можно судить о точности этого числа.



Цифры в записи числа, о которых неизвестно, верны они или нет, называются **сомнительными**. В промежуточных вычислениях сохраняют одну-две сомнительные цифры, а в окончательном результате сомнительные цифры отбрасывают (иногда оставляют одну).

## Правила округления чисел и погрешностей

Если в старшем из отбрасываемых разрядов стоит цифра меньше пяти, то содержимое сохраняемых разрядов числа не изменяется. В противном случае в младший сохраняемый разряд добавляется единица с тем же знаком, что и у самого числа.

При округлении числа возникает **погрешность округления**, равная модулю разности неокругленного и округленного значений:

$$\Delta_{\text{окр}} = |x - x_{\text{окр}}|.$$

Тогда абсолютная погрешность  $x_{\text{окр}}$  складывается из абсолютной погрешности числа  $x$ , являющегося приближением его точного значения, и погрешности округления.



**Пример.** Пусть в приближенном значении  $a = 16,395$  все цифры верны в широком смысле. Округлим  $a$  до сотых:  $a_{окр} = 16,40$ . Тогда  $\Delta_{окр} = 0,005$ , а полная погрешность  $\Delta a_{окр} = \Delta a + \Delta_{окр} = 0,001 + 0,005 = 0,006$ . Значит в  $a_{окр} = 16,40$  цифра 0 не верна в строгом смысле.

Предельная абсолютная погрешность суммы приближенных чисел равна сумме предельных абсолютных погрешностей слагаемых:

$$\Delta_u = \Delta_x + \Delta_y$$

*Из формулы следует, что предельная абсолютная погрешность суммы не может быть меньше предельной абсолютной погрешности наименее точного из слагаемых, т.е. если в состав суммы входят приближенные слагаемые с разными абсолютными погрешностями, то сохранять лишние значащие цифры в более точных не имеет смысла.*

$$\Delta_u = 0,001 + 0,01 + 0,0001 = 0,0111;$$

$$u = 0,259 + 45,12 + 1,0012 \approx 0,26 + 45,12 + 1,00 = 46,38 \pm 0,01.$$

Если все слагаемые в сумме имеют один и тот же знак, то предельная относительная погрешность суммы не превышает наибольшей из предельных относительных погрешностей слагаемых:

$$\delta_u \leq \max(\delta_{x1}, \delta_{x2}, \dots, \delta_{xn})$$

При вычислении разности двух приближенных чисел  $u = x - y$ , её абсолютная погрешность равна сумме абсолютных погрешностей уменьшаемого и вычитаемого:

$$\Delta_u = \Delta_x + \Delta_y$$

Предельная относительная погрешность:

$$\Delta_u = \frac{\Delta_x - \Delta_y}{|x - y|}$$

Из формулы следует, что если приближенные значения  $x$  и  $y$  близки, то предельная относительная погрешность будет очень большой.

Предельная относительная погрешность произведения  $u = x \cdot y$  приближенных чисел, отличных от нуля, равна сумме предельных относительных погрешностей сомножителей

$$\delta_u = \delta_x + \delta_y.$$

Предельная относительная погрешность частного равна сумме предельных относительных погрешностей делимого и делителя.

**Пример:** Вычислить функцию  $u = 2 \sin(3x + 4y)$ , если

$$x = \frac{\pi}{24} \pm 0,002 \text{ и } y = \frac{\pi}{24} \pm 0,005$$

Найдите предельные абсолютную и относительную погрешности результата и определите число верных значащих цифр.

**Решение:**

$$\begin{aligned}\Delta_u &= \left| \frac{\partial u}{\partial x} \right| \Delta_x + \left| \frac{\partial u}{\partial y} \right| \Delta_y = |6\cos(3x + 4y)| \cdot 0,002 + \\&\quad + |8\cos(3x + 4y)| \cdot 0,005 = \\&= |2\cos(3x + 4y)| \cdot (3 \cdot 0,002 + 4 \cdot 0,005) = \\&= \left| 2\cos \frac{\pi}{4} \right| \cdot 0,026 = \sqrt{2} \cdot 0,026 \approx 0,037\end{aligned}$$

Для функции  $u$  находим  $u = \sqrt{2} \approx 1,1414214$ . Учитывая предельную абсолютную погрешность  $\Delta_u \approx 0,04$ , получаем результат, который имеет две верных значащих цифры в узком смысле. Ответ можно записать в виде  $u = 1,4 \pm 0,04$ .



Если положительное приближенное число имеет  $n$  верных значащих цифр, то его относительная погрешность  $\delta$  не превосходит величины  $10^{1-n}$ , деленной на первую значащую цифру  $a_H$ .

$$\delta \leq 10^{1-n} / a_H$$

**Пример:** Найти относительную и абсолютную погрешности приближенных чисел:  
a) 3,142 ; b)  $2,997925 \cdot 10^8$  .

Решение.

a) Здесь  $n = 4$ ,  $a_H = 3$ . Подсчитаем относительную погрешность:

$$\delta_a = 10^{1-n} / a_H = 0,001/3 \approx 0,00033.$$

Для определения абсолютной погрешности:

$$\Delta_a \approx |a_p| \delta_a = 3,142 \cdot 0.00033 \approx 0,001.$$

b) Аналогично вычислим:  $n = 7$ ,  $a_H = 2$ ,

$$\delta_a = 10^{1-n} / a_H = 0,000001/2 = 0,0000005;$$

$$\Delta_a \approx |a_p| \delta_a = 2,997925 \cdot 10^8 - 0,0000005 \approx 150.$$



# СЛОЖНОСТЬ И НОТАЦИЯ BIG-O



- **Дискретность** – алгоритм представляется как последовательность инструкций исполнителя. Каждая инструкция выполняется только после того, как закончилось выполнение предыдущего шага.
- **Конечность** (результативность, финитность) – алгоритм должен заканчиваться после выполнения конечного числа инструкций.
- **Массовость** – алгоритм решения задачи должен быть применим для некоторого класса задач, различающихся лишь значениями входных данных.
- **Детерминированность (определенность)** – каждый шаг алгоритма должен быть точно определен (записан на формальном языке исполнителя). Детерминированность обеспечивает одинаковость результата, получаемого при многократном выполнении алгоритма, на одном и том же наборе входных данных.

1. **Время выполнения** (execution time) – временная эффективность
2. **Объем потребляемой памяти** (memory consumption) – пространственная эффективность
3. **Объем потребляемой электроэнергии** (energy consumption)

## Что влияет на время выполнения алгоритма (программы)?

1. Размер входных данных
2. Качество реализации алгоритма на языке программирования
3. Качество скомпилированного кода
4. Производительность вычислительной машины





- **Сложность** функции – это зависимость между размером входных данных и сложностью выполнения этой функции до завершения. Размер входных данных обычно обозначается как  $n$ . Однако  $n$  чаще всего представляет собой некую конкретную характеристику, например, длину массива.
- Сложность задачи можно измерять разными способами. Один из удобных способов – учитывать **базовые операции**: сложение, вычитание, умножение, деление, присваивание и вызовы функций. Хотя выполнение каждой из этих операций занимает разное время, общее число базовых операций, необходимых для завершения работы функции, достаточно хорошо коррелирует с фактическим временем выполнения. При этом подсчет базовых операций гораздо проще.

У каждого алгоритма есть параметры, определяющие размеры его входных данных.

- Поиск минимума в массиве:  $n$  – количество элементов в массиве.
- Умножения матриц:  $n, m$  – количество строки и столбцов в матрице.
- Сравнения двух строк:  $s_1, s_2$  – длина первой и второй строк.
- Поиск кратчайшего пути в графе между двумя вершинами:  $n, m$  – количество вершин и ребер в графе.

- Время выполнения алгоритма можно оценить путем подсчета количества "базовых операций" выполняемых им.
- Количество операций алгоритма можно выразить как функцию от размера входных данных:  $T(n)$ ,  $T(s_1, s_2)$ ,  $T(n, m)$ .
- При подсчете количества операций в алгоритме принимаются следующие допущения о гипотетической вычислительной машине:
  - для выполнения "простой" операции (+, -, \*, /, =, if) требуется один временной шаг (такт вычислительной машины).
  - обращение к памяти (чтение, записи) выполняется за один временной шаг.
  - циклы и подпрограммы состоят из последовательности простых операций.

Вычислим количество  $T(n)$  операций, выполняемых алгоритмом поиска максимального элемента в массиве из  $n$  элементов.

```
1 algorithm MaxValue(h, n)
2 hmax = 0
3 for i = 0 to n - 1 do
4     if h[i] > hmax then
5         hmax = h[i]
6         k = i
7     end if
8 end for
9 return k
```

- Результат условного ветвления заранее неизвестен
- Рассматриваем худший случай (worst case) - условие выполняется всегда
- Ветвление if - 1 операция
- Сравнение  $h[i] > hmax$  - 1 операция

Количество  $T(n)$  операций, выполняемых алгоритмом в худшем случае:

$$T(n) = 1 + (2 + 2)n + 1 = 2 + 4n$$

- Можно было также учесть 2 операции для цикла for (сравнение и переход)



- Вычислительная сложность алгоритма (computational complexity) - это оценка количества операций выполняемых алгоритмом в зависимости от размера его входных данных.
- При асимптотическом анализе вычислительной сложности алгоритма (asymptotic computational complexity) рассматривается поведение алгоритма при  $n \rightarrow \infty$ .

### Причины

1. Точное значение временной сложности зависит от определения элементарных операций исполнителя (например, сложность можно измерять в количестве арифметических операций, битовых операций или операций на машине Тьюринга).
2. При увеличении размера входных данных ( $n \rightarrow \infty$ ) вклад постоянных множителей и слагаемых низших порядков становится незначительным.



- При асимптотическом анализе вычислительной сложности алгоритма оценивают количество операций для следующих случаев:
  - *худший случай* (worst case) – максимальное количество операций, требуемых для обработки набора входных данных.
  - *средний случай* (average case) – среднее количество операций, требуемых для обработки набора входных данных.
  - *наилучший случай* (best case) – минимальное количество операций требуемых для обработки набора входных данных.
- Для записи асимптотической оценки вычислительной сложности алгоритмов используются асимптотические обозначения – О-нотацию (Big O).



**Big-O** описывает, как изменяется количество базовых операций в зависимости от размера входных данных при их росте.

Так как различное оборудование выполняет вычисления с разной скоростью, невозможно точно определить время работы алгоритма без учета характеристик конкретного компьютера. Однако нас больше интересует не абсолютное время работы функции, а то, как оно изменяется при увеличении  $n$ , поскольку это позволяет оценить производительность независимо от аппаратного обеспечения.

При больших значениях  $n$  наибольшее влияние оказывает член с наивысшей степенью, поэтому в нотации **Big-O** оставляют только его. Кроме того, константные коэффициенты “не имеют значения” и также отбрасываются.



**Пример:** Подсчитайте количество базовых операций в зависимости от  $n$ , необходимых для завершения следующей функции.

```
def f(n):
    out = 0
    for i in range(n):
        for j in range(n):
            out += i*j

    return out
```

Количество базовых операций в зависимости от  $n$ , необходимых для завершения функции:

Сложений:  $n^2$

Вычитаний: 0

Умножений:  $n^2$

Делений: 0

Присваиваний:  $2n^2 + n + 1$

Вызовов функций: 0

Итого:  $4n^2 + 1$

В нотации Big-O сложность записывается как:  $O(n^2)$ .

*Любой алгоритм со сложностью  $O(n^c)$ , где  $c$  – некоторая константа, называют полиномиальным.*



**Пример:** Подсчитайте количество базовых операций в зависимости от  $n$ , необходимых для завершения следующей функции.

```
def my_fib_iter(n):  
  
    out = [1, 1]  
  
    for i in range(2, n):  
        out.append(out[i - 1] + out[i - 2])  
  
    return out
```

Единственная часть кода, время выполнения которой увеличивается с ростом  $n$  – это цикл for. Код внутри цикла выполняется фиксированное количество раз и не зависит от  $n$ , поэтому количество базовых операций можно записать как  $Cn$ , где  $C$  – некоторая константа.

В нотации Big-O сложность записывается как:  $O(n)$ .



Оценка точной сложности функции может быть затруднительной. В таких случаях достаточно дать верхнюю границу или приблизительную оценку сложности.

**Пример:** Определите верхнюю границу сложности рекурсивного алгоритма вычисления чисел Фибоначчи:

```
def my_fib_rec(n):

    if n < 2:
        out = 1
    else:
        out = my_fib_rec(n-1) + my_fib_rec(n-2)

    return out
```

При больших значениях  $n$  большинство вызовов функции делает два рекурсивных вызова: одно сложение и одно присваивание. Операции сложения и присваивания не зависят от  $n$  и поэтому могут быть проигнорированы. Однако число вызовов функций растет примерно как  $2^n$ , а значит, сложность рекурсивного алгоритма оценивается сверху как:  $O(2^n)$ .

Так как число вызовов растет экспоненциально, рекурсивный алгоритм вычисления чисел Фибоначчи не может быть полиномиальным. То есть, для любого  $c$  существует  $n$ , для которого  $my\_fib\_rec$  выполняет больше операций, чем  $O(n^c)$ . Любая функция со сложностью  $O(c^n)$  (где  $c$  – константа) называется **экспоненциальной**.



**Пример:** Определите сложность следующей функции в нотации Big-O.

```
def divide_by_two(n):
    out = 0
    while n > 1:
        n /= 2
        out += 1
    return out
```

В данном случае только цикл `while` зависит от  $n$ , поэтому сосредоточимся на нем.

Внутри цикла выполняются две операции: деление и сложение, которые являются константными (относительно  $n$ ) по времени выполнения. Следовательно, сложность зависит только от количества итераций цикла.

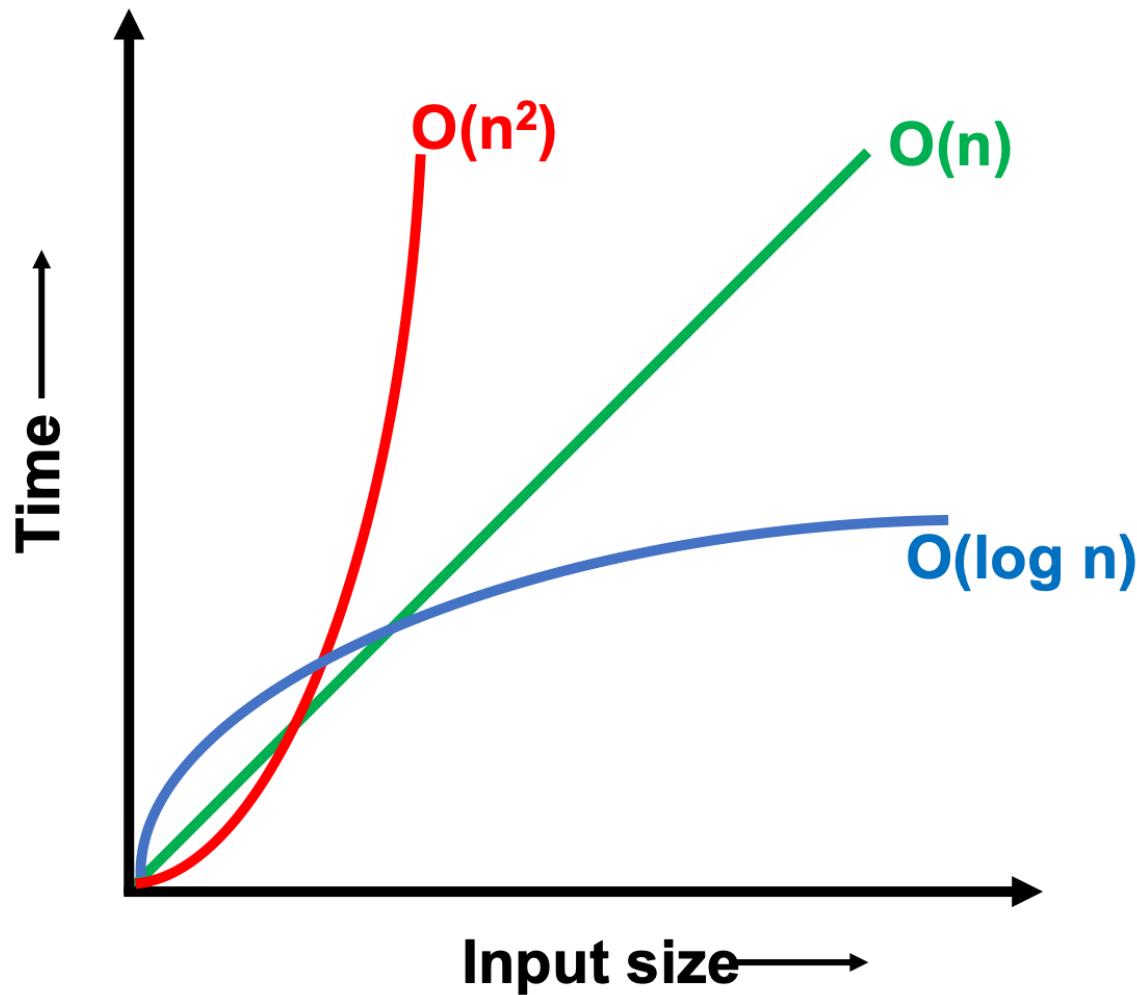
Цикл сокращает  $n$  вдвое на каждой итерации, пока  $n$  не станет меньше 1. Число итераций  $I$  можно найти из уравнения  $\frac{n}{2^I} = 1$ . Решив его, получаем:  $I = \log n$ .

Таким образом, сложность функции `divide_by_two` составляет:  $O(\log n)$ .

База логарифма не играет роли, так как все логарифмы отличаются лишь постоянным множителем. Функция со сложностью  $O(\log n)$  называется **логарифмической**.



Различные алгоритмы требуют разного времени на выполнение задачи. График ниже иллюстрирует, как изменяется время выполнения в зависимости от размера входных данных для алгоритмов со сложностями  $\log(n)$ ,  $n$ ,  $n^2$ .





Класс сложности	Название
$O(1)$	Константная сложность
$O(\log(n))$	Логарифмическая сложность
$O(n)$	Линейная сложность
$O(n \log n)$	$n\text{-log-}n$ сложность
$O(n^2)$	Квадратичная сложность
$O(n^3)$	Кубическая сложность
$O(2^n)$	Экспоненциальная сложность
$O(n!)$	Факториальная сложность



Допустим, у нас есть алгоритм с экспоненциальной сложностью, например  $O(2^n)$ , и максимальный размер задачи, которую мы можем решить с имеющимися вычислительными ресурсами  $R$ , составляет  $N$ .

Если производительность компьютера удваивается, какой размер задачи мы сможем решить теперь?

Пусть:  $R = 2N$ . С новым компьютером у нас  $2R$  вычислительных ресурсов. Тогда новый максимальный размер задачи  $N'$  определяется уравнением:  $2R = 2N'$ .

Подставляя  $R = 2N$ , получаем:

$$2 \times 2N = 2N' \rightarrow 2N + 1 = 2N' \rightarrow N' = N + 1.$$

То есть удвоение мощности компьютера позволяет решить задачу лишь на одну единицу больше. При больших  $N$  этот прирост становится незначительным.

При полиномиальной сложности ситуация значительно лучше. Если  $R = N^c$ , где  $c$  – константа, со значением большим 1, то удвоение ресурсов дает:  $2R = N'^c$ .

Решая уравнение, получаем:  $N' = 2^{1/c} N$ .

При небольших значениях  $c$  (например,  $c < 5$ ) прирост решаемой задачи намного больше, чем при экспоненциальной сложности.



Рассмотрим теперь алгоритм, работающий за логарифмическое время. Пусть  $R=\log N$ . Тогда при удвоении вычислительных ресурсов получаем  $2R=\log N'$ .

Проведя математические преобразования, получаем, что  $N'=N^2$ . Это означает, что удвоение вычислительных мощностей позволяет нам решить задачу, размер которой увеличен в квадрат!

Главный вывод: алгоритмы с экспоненциальной сложностью крайне плохо масштабируются. По мере увеличения размера входных данных время выполнения функции возрастает настолько быстро, что выполнение становится непрактичным.

Рассмотрим последний пример. Вызов функции `my_fib_rec(100)` требует порядка  $2^{100}$  базовых операций.

Даже если, предположим, компьютер способен выполнять 100 триллионов операций в секунду (что значительно быстрее любого современного суперкомпьютера), выполнение займет около **400 миллионов лет**. В то же время итеративный вариант `my_fib_iter(100)` выполнится за **менее чем одну наносекунду**.



# ЧИСЛЕННОЕ РЕШЕНИЕ НЕЛИНЕЙНЫХ УРАВНЕНИЙ



Нелинейные уравнения можно поделить на два типа: алгебраические и трансцендентные. Уравнение называется алгебраическим, если его можно представить в виде  $\sum_{i=0}^n a_i x^i = 0$ .

Это – каноническая форма записи алгебраического уравнения. Пример алгебраического уравнения:  $2.5x^5 - 4x^4 - 3x^2 - 8x + 12 = 0$ .

Если левая часть уравнения  $f(x)=0$  содержит другие функции (тригонометрические, показательные, логарифмические и др.), то уравнение называется трансцендентным. Пример трансцендентного уравнения:  $x \ln x + x^2 \cos x - \frac{x}{\lg x + 3} - 8 = 0$ .



Пусть имеется уравнение вида

$$F(x)=0,$$

где  $F(x)$  – алгебраическая или трансцендентная функция.

Решить такое уравнение – значит установить, имеет ли оно корни, сколько корней, и найти значения корней с заданной точностью.

Известно, что алгебраическое уравнение имеет ровно  $n$  корней – вещественных или комплексных. Если  $n = 1, 2$  (при определённых условиях 3, 4), то существуют точные методы решения алгебраических уравнений. Если же  $n > 4$  или уравнение трансцендентное, то таких методов, как правило, не существует, и решение уравнения находят приближенными методами.

Далее будем предполагать, что  $f(x)$  – непрерывная на интервале поиска корня функция.

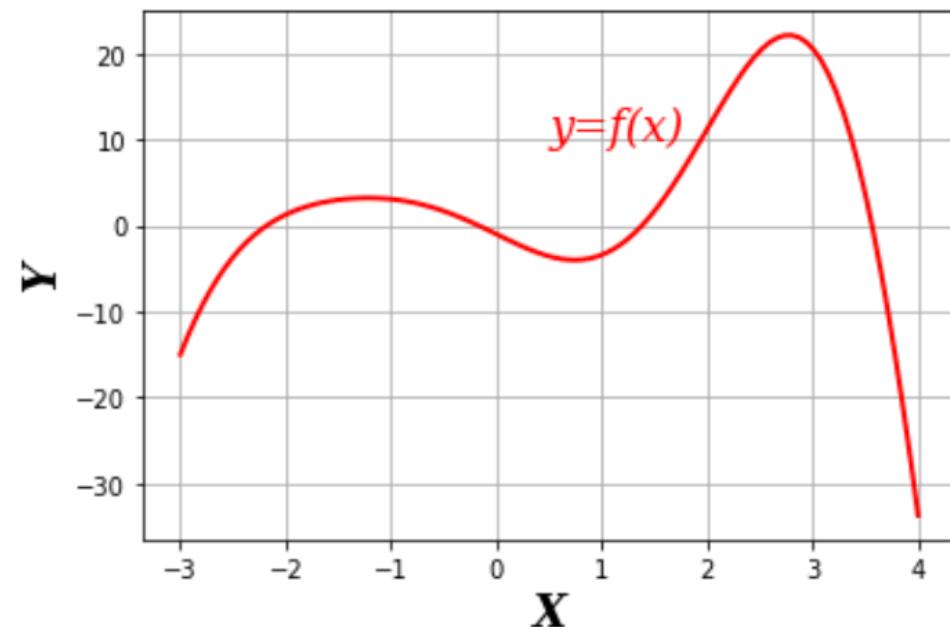


- Методы решения делятся на прямые и численные (итерационные).
- *Прямой метод* – существует формула для определения значения  $x$ , например, нахождение корней квадратного уравнения.
- Точное решение удается получить только в исключительных случаях, и обычно для нахождения корней уравнения применяются численные методы.

Решение уравнения численными методами состоит из двух этапов:

- первый этап – отделение корня, отыскание приближенного значения корня или содержащего его отрезка.
- второй – уточнение приближенных корней - доведение их до заданной степени точности.

Отделить корень – значит указать такой отрезок  $[a, b]$ , на котором содержится ровно один корень уравнения  $f(x) = 0$ .



Корень можно отделить аналитически и графически.

Не существует алгоритмов отделения корня, пригодных для любых функций  $f(x)$  – невозможно для всего многообразия таких уравнений построить прямой метод.

Итерационный процесс состоит в последовательном уточнении начального приближения  $x_0$ . Каждый такой шаг называется итерацией.

В результате итераций находится последовательность приближенных значений корня

$$x_1, x_2, \dots, x_n .$$

Если

$$\lim_{n \rightarrow \infty} \{x_n\} = x_{\text{точное}},$$

то говорят, что итерационный процесс сходится.

В целом, задача приближенного определение местоположения и вида интересующего нас корня – *этапа отделения корней* (нахождение грубых корней) может быть решена:

1. на заданном отрезке  $[a, b]$  вычисляется таблица значений функции с некоторым шагом  $h$  и определяются интервалы  $(\alpha_i, \beta_i)$  длиной  $h$ , на которых функция меняет знак (график функции пересекает ось X), т.е. где находятся корни;
2. *графическим методом*: по построенной таблице строится график и аналогично определяются интервалы, на которых находятся корни.



Отделить корни уравнения:  $f(x) \equiv -6x + 2 = 0$ .

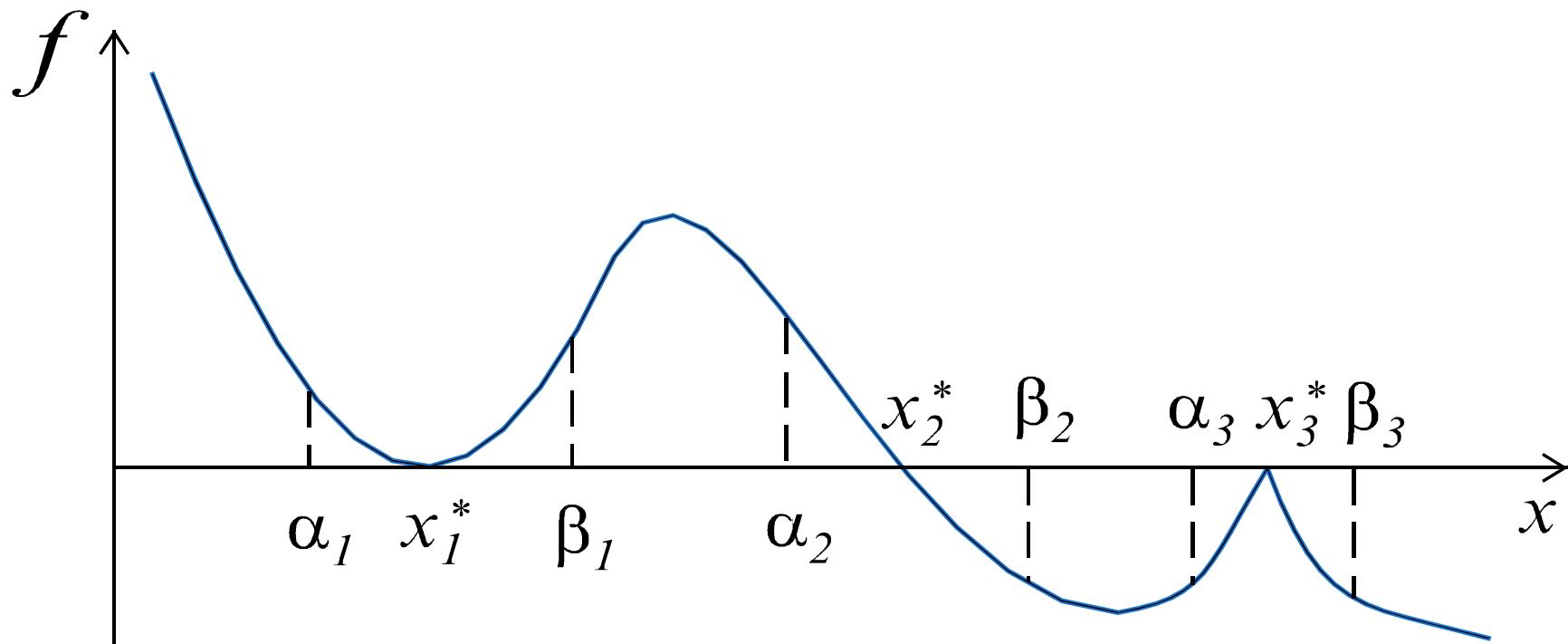
Составим приблизительную схему:

x	$-\infty$	-3	-1	0	1	3	$+\infty$
$f(x)$	-	-	+	+	-	+	+

Следовательно, уравнение имеет три действительных корня, лежащих в интервалах  $[-3, -1]$ ,  $[0, 1]$  и  $[1, 3]$ .

Приближенные значения корней (*начальные приближения*) могут быть также известны из физического смысла задачи, из решения аналогичной задачи при других исходных данных, или могут быть найдены графическим способом.

В инженерной практике распространен *графический способ* определения приближенных корней.



Виды корней:

- a.  $x_1^*$  – кратный корень;
- b.  $x_2^*$  – простой корень;
- c.  $x_3^*$  – вырожденный корень.

Для кратного корня:

$$f'(x_1^*) = 0, \quad f(\alpha_1) \cdot f(\beta_1) > 0;$$

Для простого корня:

$$f'(x_2^*) \neq 0, \quad f(\alpha_2) \cdot f(\beta_2) < 0;$$

Для вырожденного корня:

$f'(x_3^*)$  не существует,

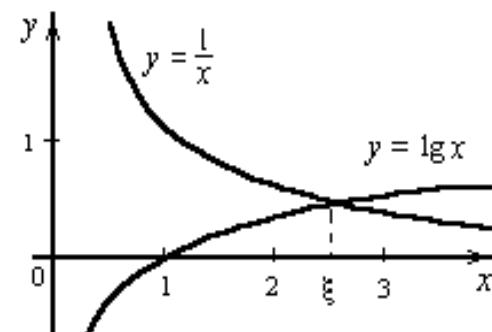
$$f(\alpha_3) \cdot f(\beta_3) > 0$$

Как видно из рисунка, в случаях а. и с. значение корня совпадает с точкой экстремума функции и для нахождения таких корней, назовем их особыми, рекомендуется использовать методы поиска минимума (максимума) функции.

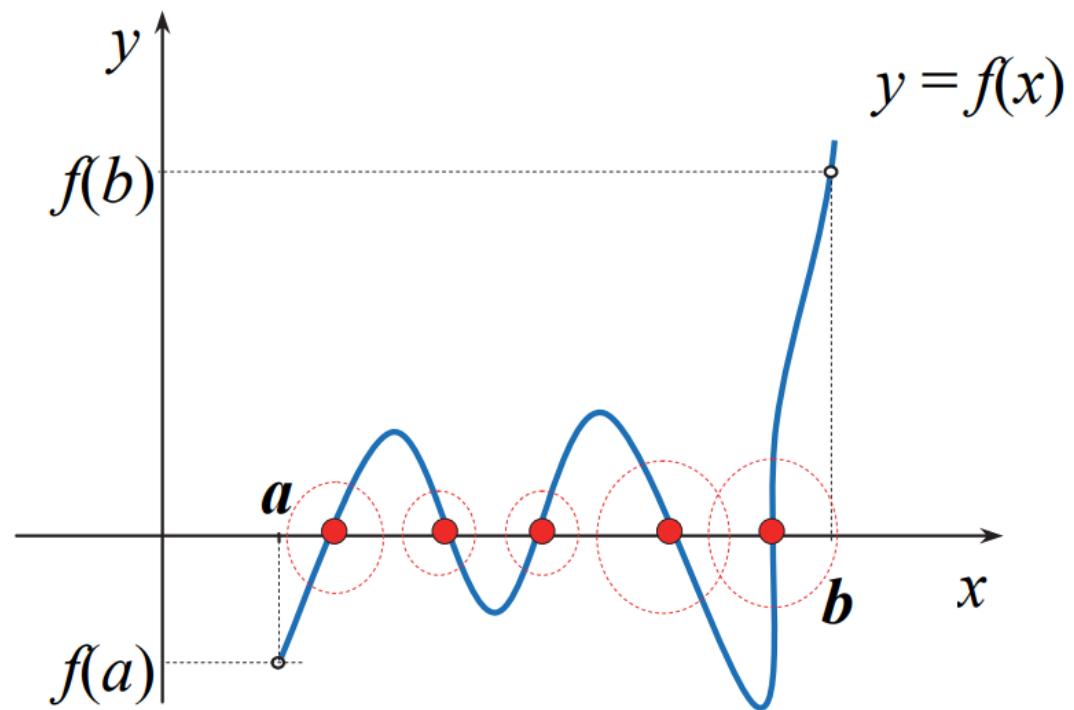
Вычисление значения простого корня с заданной точностью осуществляется одним из итерационных методов.

Принимая во внимание, что действительные корни уравнения – это точки пересечения графика функции  $f(x)$  с осью абсцисс, достаточно построить график функции  $f(x)$  и отметить точки пересечения  $f(x)$  с осью  $Ox$ , или отметить на оси  $Ox$  отрезки, содержащие по одному корню.

Построение графиков часто удается сильно упростить, заменив уравнение *равносильным* ему уравнением:  $f_1(x) = f_2(x)$ , где функции  $f_1(x)$  и  $f_2(x)$  – более простые, чем функция  $f(x)$ . Тогда, построив графики функций  $y = f_1(x)$  и  $y = f_2(x)$ , искомые корни получим как абсциссы точек пересечения этих графиков.



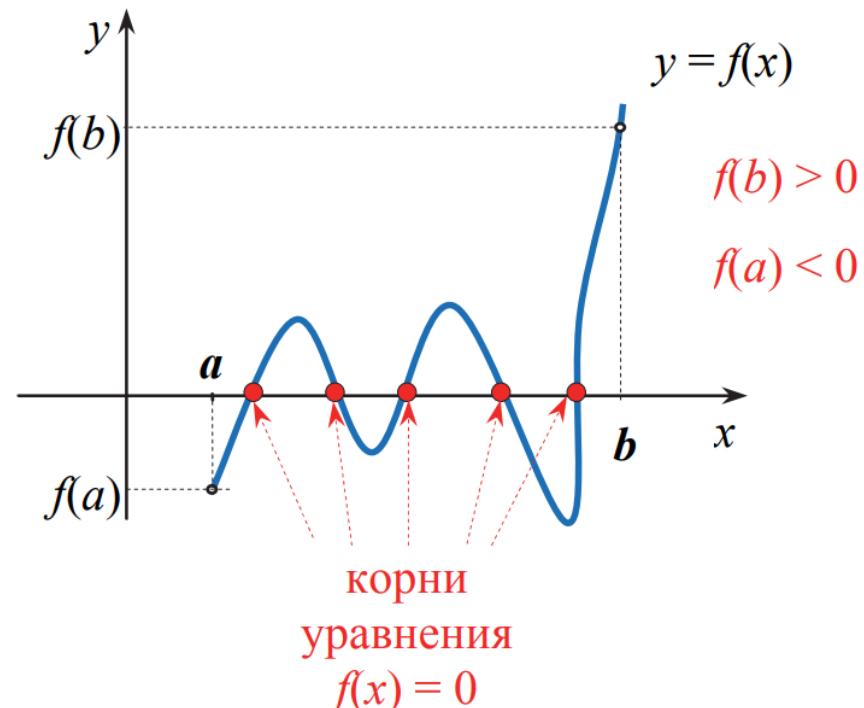
Будем считать, что уравнение имеет лишь изолированные корни, т.е. для каждого корня уравнения существует окрестность, не содержащая других корней этого уравнения.



Приближенное нахождение изолированных действительных корней уравнения складывается из двух обязательных этапов:

- 1) отделение корней, т.е. установление возможно тесных отрезков  $[\alpha, \beta]$ , в которых содержится один и только один корень уравнения;
- 2) уточнение приближенных корней, т.е. доведение их до заданной степени точности.

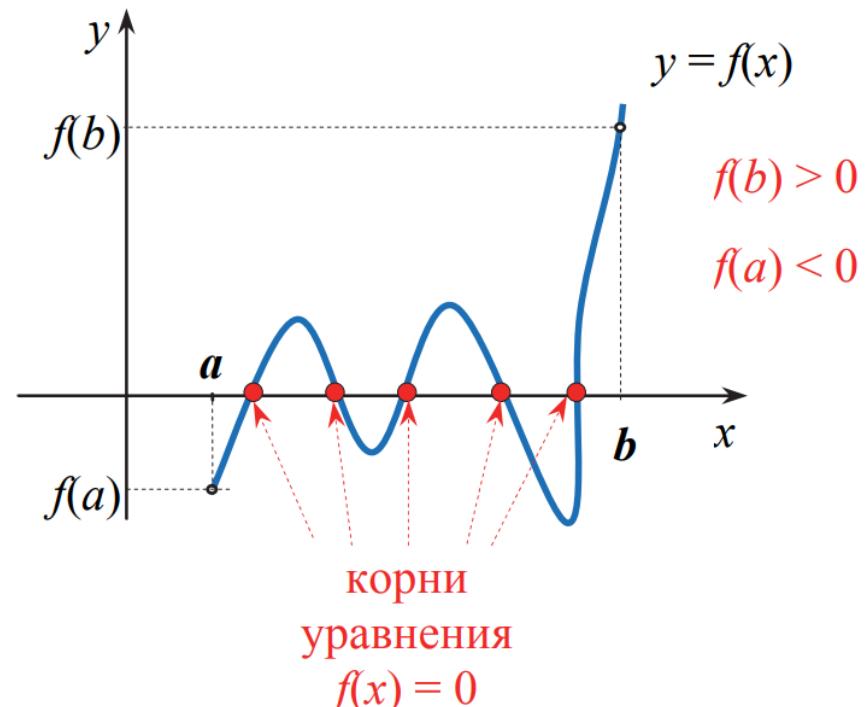
**Теорема:** Если непрерывная функция  $f(x)$  принимает значения разных знаков на концах отрезка  $[\alpha, \beta]$ , т.е.  $f(\alpha) \cdot f(\beta) < 0$ , то внутри этого отрезка содержится по меньшей мере один корень уравнения  $f(x) = 0$ , т.е. найдется хотя бы одно число  $\xi \in (\alpha, \beta)$  такое, что  $f(\xi) = 0$ .



Корень  $\xi$  заведомо будет единственным, если производная  $f'(x)$  существует и сохраняет постоянный знак внутри интервала  $(\alpha, \beta)$ , т.е. если  $f'(x) > 0$  (или  $f'(x) < 0$ ) при  $\alpha < x < \beta$ .

Процесс определения корней начинается с установления знаков функции  $f(x)$  в граничных точках  $x = a$  и  $x = b$  области существования решения.

Определить отрезок  $[\alpha, \beta]$ , в котором содержится один и только один корень уравнения, можно графически.



## Обобщим:

Если удастся подобрать такие  $a$  и  $b$ , что

1.  $f(a) \cdot f(b) < 0$ ;
2.  $f(x)$  – непрерывная на  $[a, b]$  функция;
3.  $f(x)$  – монотонная на  $[a, b]$  функция;

то можно утверждать, что на отрезке  $[a, b]$  корень отделен.

Условия 1–3 – достаточные условия отделения корня, т. е. если эти условия выполняются, то корень отделен, но невыполнение, например, условий 2 или 3 не всегда означает, что корень не отделен. Предложенный метод поиска отрезка  $[a, b]$  – аналитический способ отделения корня.

Корень можно отделить и графически.

Второй этап решения уравнения – уточнение корня. Уточнить корень – значит найти его приближенное значение с заданной погрешностью  $\varepsilon$ .

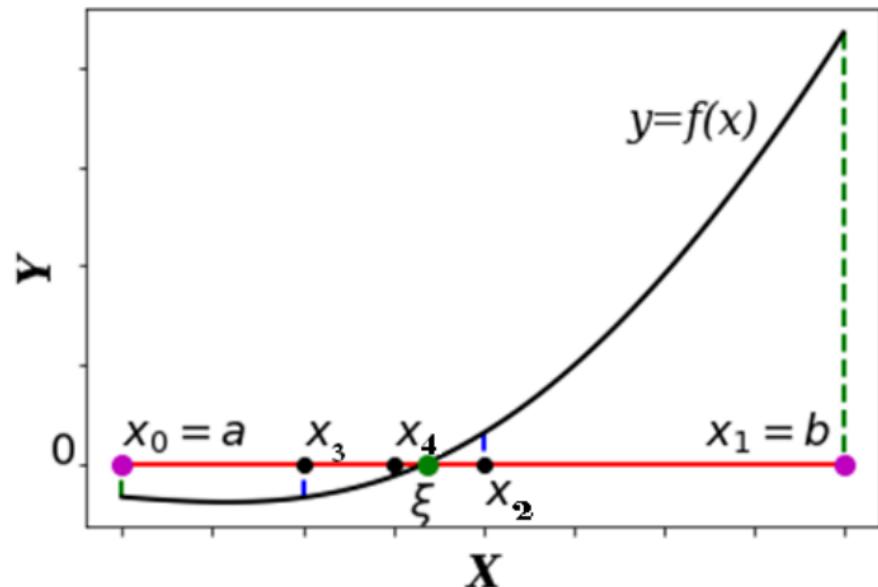
Существует множество методов численного решения уравнений.

Это обусловлено отсутствием универсального метода, пригодного для решения любых уравнений с устраивающей всех скоростью сходимости к корню. У каждого метода есть свои достоинства и недостатки. У многих методов необходимо проверять условия сходимости, т. е. условия, гарантирующие, что за конечное число шагов применения данного метода мы получим корень с любой наперед заданной погрешностью  $\varepsilon$ .



Самый простой метод, пригодный для поиска корней любых непрерывных функций, – метод деления отрезка пополам.

Предположим, что отрезок  $[a, b]$ , на котором отделен корень уравнения, уже найден. Пусть, например,  $f(a) < 0, f(b) > 0$ . Буквой  $\xi$  обозначен корень уравнения.





Пусть уравнение:

- имеет на отрезке  $[a, b]$  единственный корень;
- функция  $F(x)$  на этом отрезке непрерывна.

Разделим отрезок  $[a,b]$  пополам точкой  $c = (a+b)/2$ .

Если  $F(c) \neq 0$ , то возможны два случая:

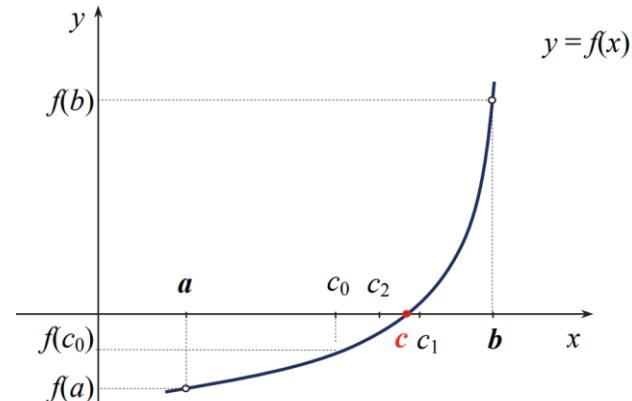
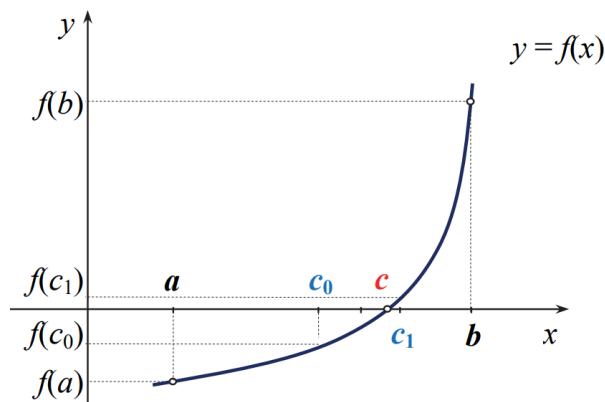
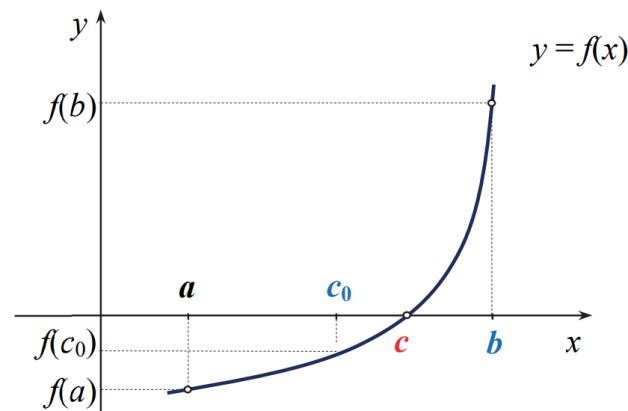
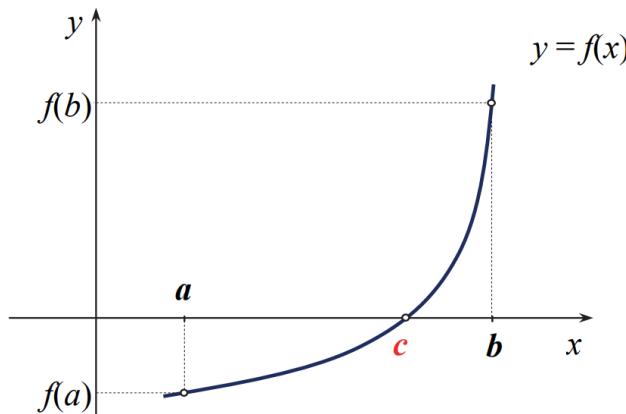
- $F(x)$  меняет знак на отрезке  $[a,c]$ ;
- $F(x)$  меняет знак на отрезке  $[c,b]$ .

Выбирая каждый раз тот из отрезков, на котором функция меняет знак, и продолжая процесс половинного деления дальше, можно дойти до сколь угодно малого отрезка, содержащего корень.



В основе этого метода лежит свойство непрерывных функций, заключающееся в том, что если функция  $f(x)$  на концах отрезка  $[a, b]$  принимает значения разных знаков, т.е.  $f(a) \cdot f(b) < 0$ , то внутри этого отрезка содержится по меньшей мере один корень уравнения  $f(x) = 0$ .

Если  $F(x_i) = 0$ , то корень найден.

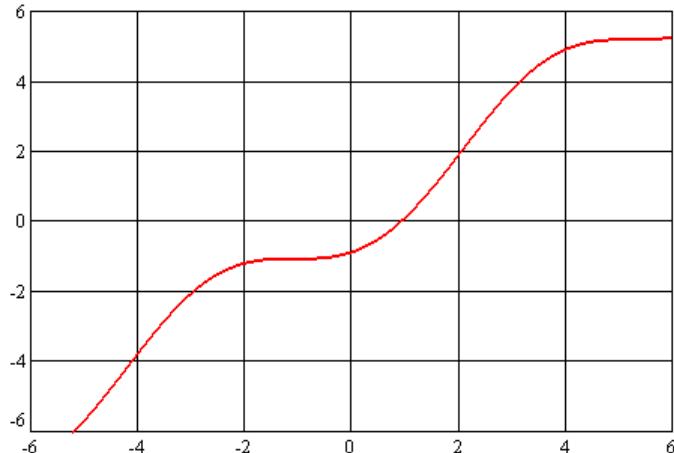




Пример:

$$\sin\left(x + \frac{\pi}{3}\right) = x$$

$$\sin\left(x + \frac{\pi}{3}\right) - x = 0$$



$$a := 0 \qquad b := 2$$

$$f(a) = 0.866 \qquad f(b) = -1.906$$

$$x := \frac{a + b}{2} = 1$$

$$f(x) = -0.111 \qquad b := x$$

$$x := \frac{a + b}{2} = 0.5$$

$$f(x) = 0.5 \qquad a := x$$

$$x := \frac{a + b}{2} = 0.75$$

$$f(x) = 0.224 \qquad a := x$$

$$x := \frac{a + b}{2} = 0.875$$

$$f(x) = 0.064 \qquad a := x$$

$$x := \frac{a + b}{2} = 0.938$$

$$f(x) = -0.022 \qquad b := x$$

$$x := \frac{a + b}{2} = 0.906$$

$$f(x) = 0.021 \qquad a := x$$

$$x := \frac{a + b}{2} = 0.922$$

$$f(x) = -0.00014 \boxed{x = 0.922}$$

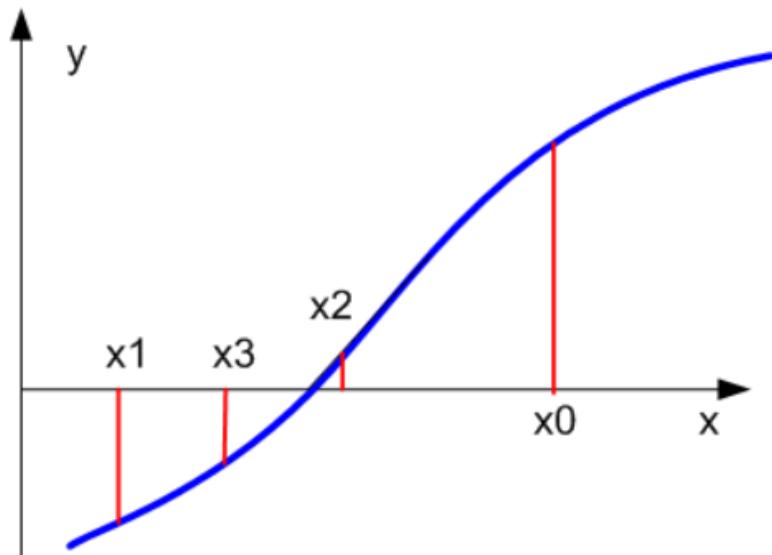


Преимущества:

сходится для любых непрерывных функций.

Недостатки:

- невелика скорость сходимости;
- неприменим для отыскания кратных корней четного порядка.





Уравнение  $f(x) = 0$  (1) записывают в разрешенном относительно  $x$  виде:

$$x = \varphi(x). \quad (2)$$

Переход от (1) к эквивалентной записи (2) можно сделать многими способами, например,

$$\varphi(x) = x + \psi(x) \cdot f(x), \quad (3)$$

где  $\psi(x)$  – произвольная, непрерывная, знакопостоянная функция (часто достаточно выбрать  $\psi = \text{const}$  из диапазона  $\pm 0.1 - 0.9$ ).

В этом случае корни уравнения (2) являются также корнями (1), и наоборот.



Для сходимости итерационного процесса на функцию  $\varphi(x)$  накладываются следующие условия:

- 1)  $\varphi(x)$  должна быть определена и дифференцируема на отрезке  $[a, b]$ , содержащем корень;
- 2) значения функции  $\varphi(x)$  должны принадлежать отрезку  $[a, b]$  для любых значений аргумента  $x \in [a, b]$ ;
- 3)  $|\varphi'(x)| < 1$  для всех  $x \in [a, b]$ .



Исходя из (2) члены рекуррентной последовательности вычисляются по формуле

$$\mathbf{x}_k = \varphi (\mathbf{x}_{k-1}), \quad k = 1, 2, \dots \quad (4)$$

Вычисления  $\mathbf{x}_k$  продолжаем до тех пор, пока

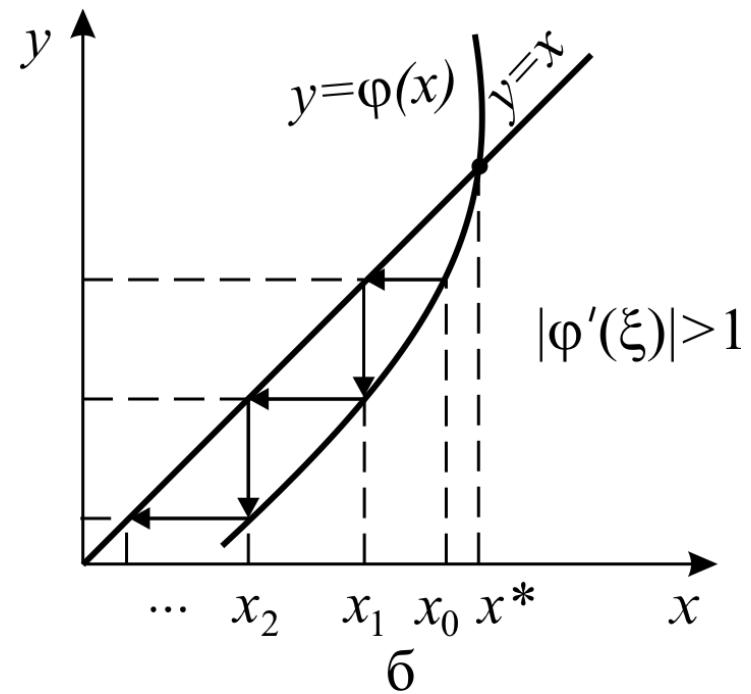
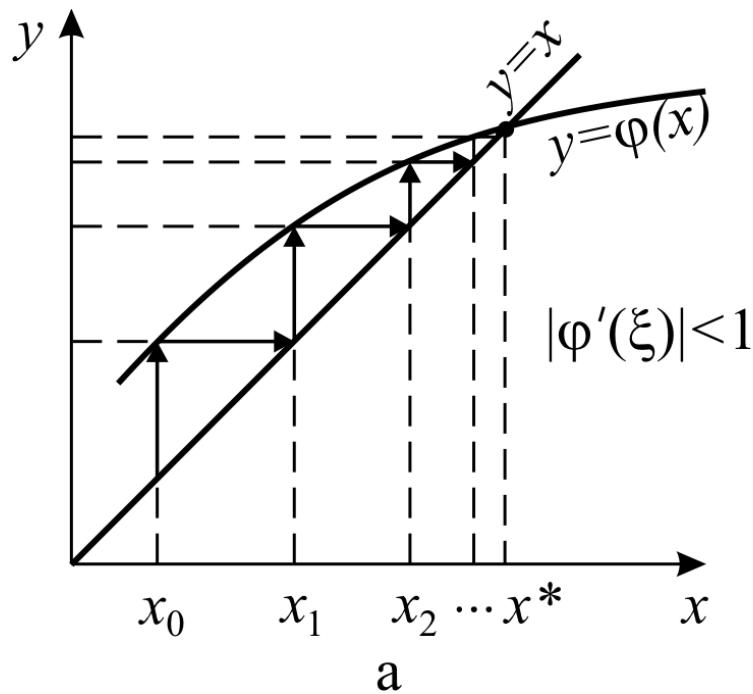
$$|\mathbf{x}_k - \mathbf{x}_{k-1}| > \varepsilon;$$

где  $\varepsilon$  – заданная точность решения.

Метод одиношаговый, т.к. последовательность  $\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{k-1}$  имеет первый порядок ( $m = 1$ ) и для начала вычислений достаточно знать одно начальное приближение:  $\mathbf{x}_0 = \alpha$ , или  $\mathbf{x}_0 = \beta$ , или  $\mathbf{x}_0 = (\alpha + \beta) / 2$ .



Геометрическая иллюстрация сходимости и расходимости метода простой итерации представлена на рисунке, из которого видно, что метод не всегда сходится к точному решению.





**Условия сходимости** метода простой итерации:

1.  $\varphi(x)$  – дифференцируема;
2. выполняется неравенство

$$|\varphi'(\xi)| < 1,$$

для любого  $\xi \in (\alpha, \beta)$ ;  $x^* \in (\alpha, \beta)$ . (5)

Максимальный интервал  $(\alpha, \beta)$ , для которого выполняется (5), называется **областью сходимости**.

При выполнении условия (5) метод сходится, если начальное приближение  $x_0$  выбрано из области сходимости.



Для сходимости итерационного процесса полезно знать следующие специальные приемы:

1. Уравнение  $f(x) = 0$  умножаем на  $m$  и к обеим частям прибавляем  $x$ :

$$x + m f(x) = x \text{ или } x = x - m f(x),$$

где  $m$  – отличная от нуля константа. Тогда можно обозначить:

$$\varphi(x) = x - m f(x).$$

Продифференцируем полученное выражение:

$$\varphi'(x) = 1 - m f'(x).$$

Тогда для выполнения третьего условия достаточно подобрать  $m$  так, чтобы для всех  $x$  отрезка  $[a, b]$  выполнялось условие:

$$0 < |m f'(x)| < 1$$



2. Пусть уравнение  $f(x) = 0$  удалось привести какими-либо эквивалентными преобразованиями к виду  $x = \varphi(x)$ , однако оказалось, что для всех  $x \in [a, b]$   $|\varphi'(x)| > 1$ . Тогда вместо функции  $y = \varphi(x)$  рассмотрим функцию  $x = \psi(y)$ , симметричную ей относительно прямой  $y = x$ .

$$x + m f(x) = x \text{ или } x = x - m f(x),$$

По свойству производных обратных функций на отрезке  $[a, b]$  имеет место соотношение:

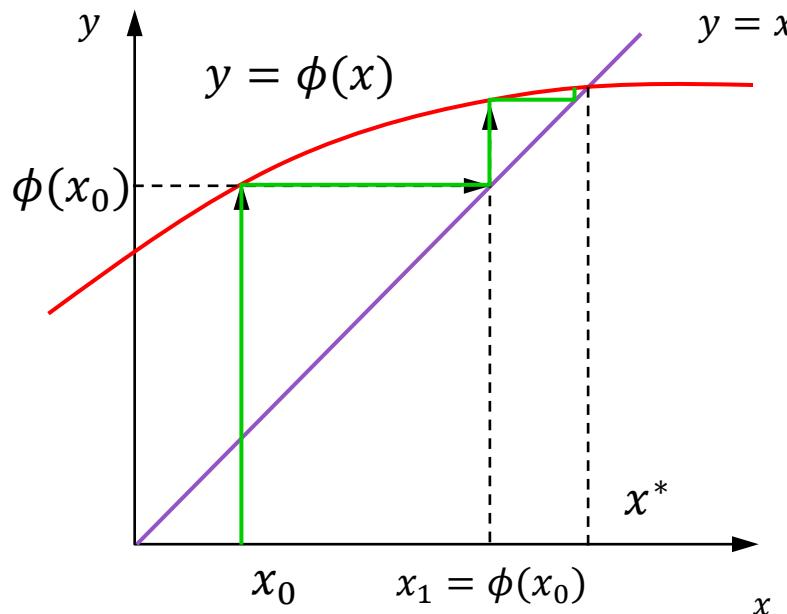
$$|\psi'(x)| = \frac{1}{|\varphi'(x)|} < 1$$

Причем уравнение  $x = \psi(x)$  имеет тот же корень, что и  $x = \varphi(x)$ .

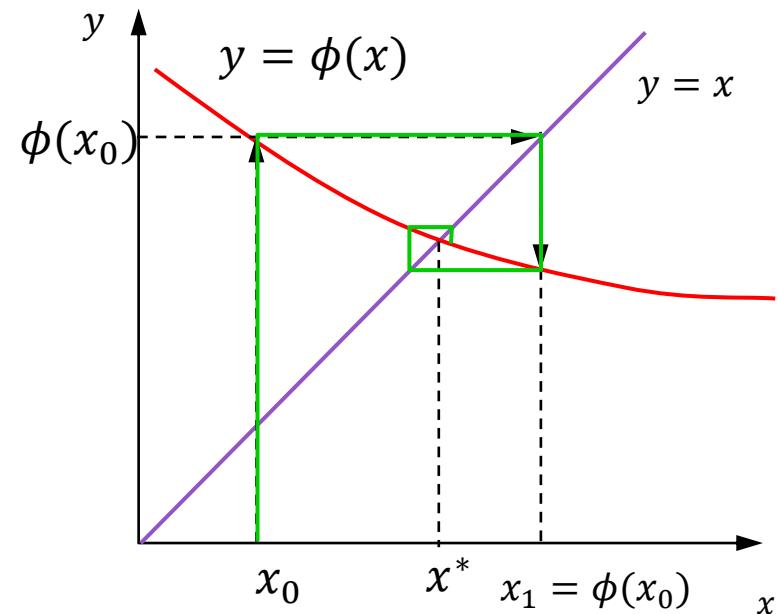


Сходящийся итерационный процесс:

Последовательность  $x_0, x_1, \dots$  приближается (сходится) к точному решению  $x^* = \phi(x^*)$ ,  $x_0, x_1, x_2, \dots \rightarrow x^*$ .



односторонняя сходимость

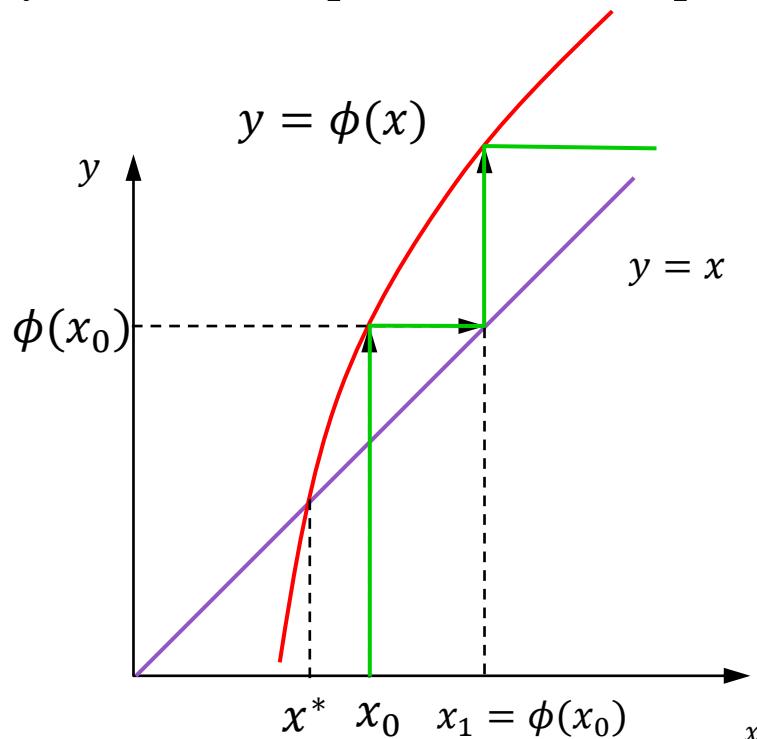


двусторонняя сходимость

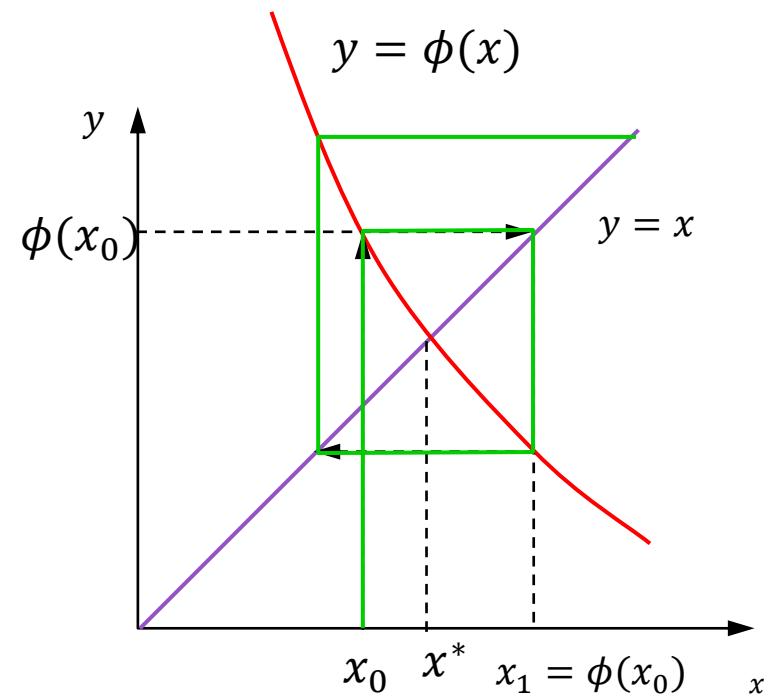


Расходящийся итерационный процесс:

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



односторонняя расходимость

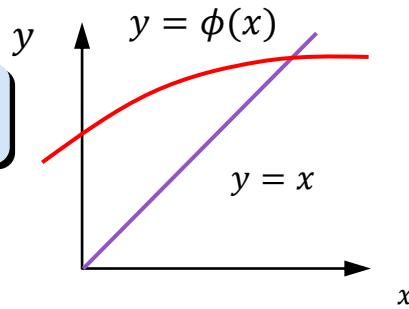


двусторонняя расходимость



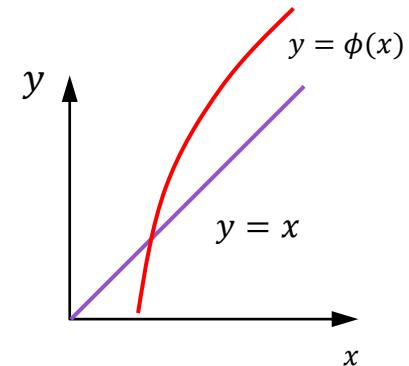
сходится

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

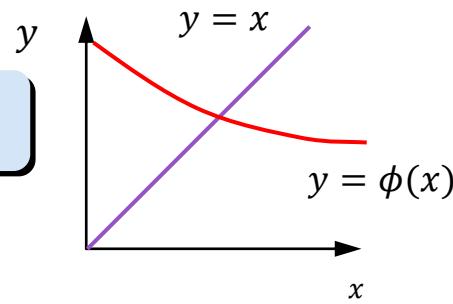


расходится

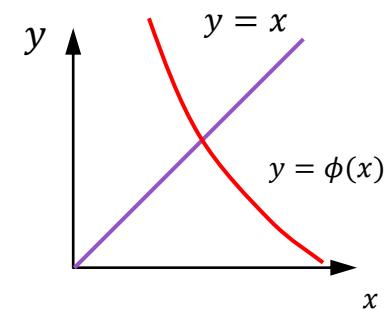
$$\phi'(x) > 1$$



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



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



Выводы:

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

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



- наугад, пробовать разные варианты
- для начального приближения  $x_0$

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

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

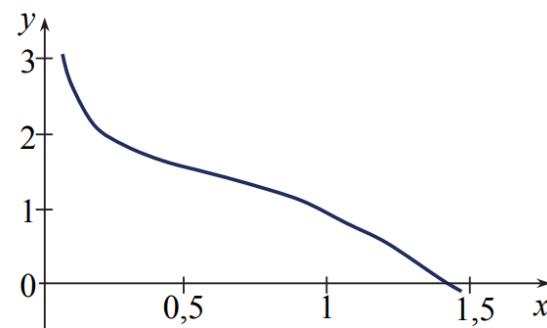
$$f'(x_0) < 0 \Rightarrow 0 < b < -\frac{2}{f'(x_0)}$$

пересчитывать на каждом шаге, например:

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



**Пример.** Решить уравнение  $\sin(2x) - \ln x = 0$  методом простых итераций с точностью до 0,001.



Приведем заданное уравнение к виду  $x = \varphi(x)$ . Для этого воспользуемся первым способом преобразования.

$$x = x - m(\sin(2x) - \ln x), \text{ или } \varphi(x) = x - m(\sin(2x) - \ln x)$$

Подберем число  $m$ , так чтобы на отрезке  $[1,3; 1,5]$  выполнялось условие

$$|\varphi'(x)| = |1 - m(2\cos(2x) - 1/x)| < 1.$$



Например, при  $m = -1/3$  это неравенство верно. Тогда

$$\varphi(x) = x + 1/3(\sin 2x - \ln x),$$

а схема итераций будет иметь вид

$$x_k = x_{k-1} + 1/3(\sin 2x_{k-1} - \ln x_{k-1})$$

где  $x_k$  –  $k$ -е приближение корня,  $k = 1, 2, 3, \dots$



Номер итерации, $k$	Приближение корня, $x^{(k)}$	Проверка условия: $ x_k - x_{k-1}  < \varepsilon = 0,001$
0	1,3	
1	1,384379	не выполняется
2	1,397381	не выполняется
3	1,399154	не выполняется
4	1,399392	не выполняется
5	1,399424	<b>выполняется</b>



Пусть дано уравнение  $f(x) = 0$ , где  $f(x)$  – непрерывная функция, имеющая в интервале  $(a, b)$  производные первого и второго порядка.

**Метод хорд (метод секущих)** также как и метод деления отрезков пополам, предназначен для уточнения корня на интервале  $[a, b]$ , на концах которого функция  $f(x)$  принимает разные знаки –  $f(a) \cdot f(b) < 0$ . Очередное приближение теперь в отличие от метода бисекции берем не в середине отрезка, а в точке  $c_0$  пересечения хорды, проведенной через точки  $(a, f(a))$  и  $(b, f(b))$  с осью абсцисс.



**Метод хорд (секущих)** – это модификация метода Ньютона, позволяющая избавиться от вычисления производной путём её замены приближенной формулой, т.е. вместо касательной проводится секущая.

Тогда вместо  $x_k = x_{k-1} - \frac{f(x_{k-1})}{f'(x_{k-1})} = \phi(x_{k-1})$  получаем:

$$x_k = x_{k-1} - \frac{f(x_{k-1})h}{f(x_{k-1}) - f(x_{k-1} - h)} = \phi(x_{k-1}).$$

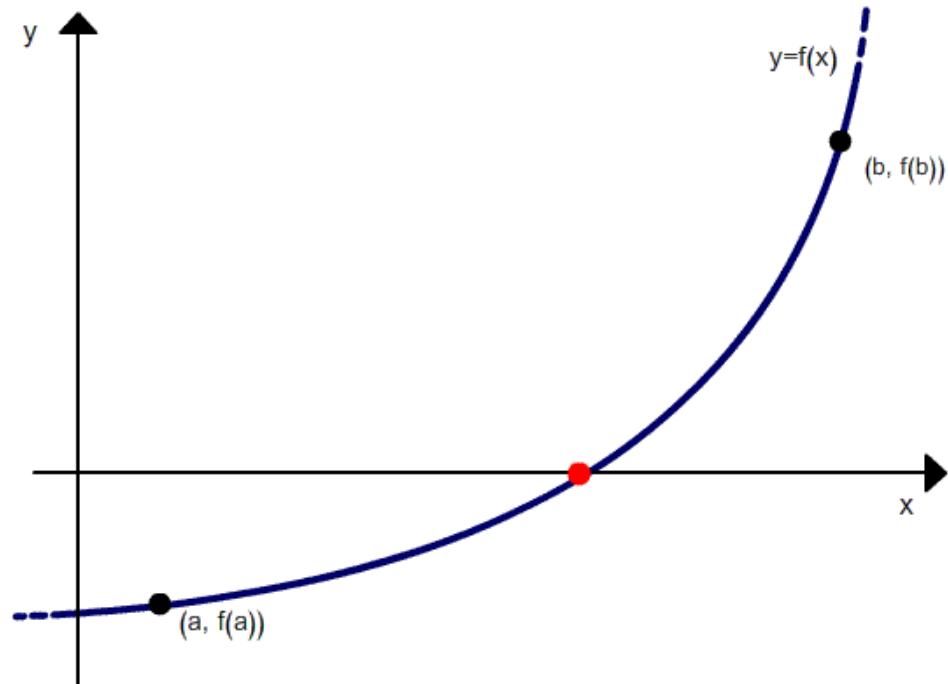
Здесь  $h$  – некоторый малый параметр метода, который подбирается из условия наиболее точного вычисления производной.



Формула для расчетов:

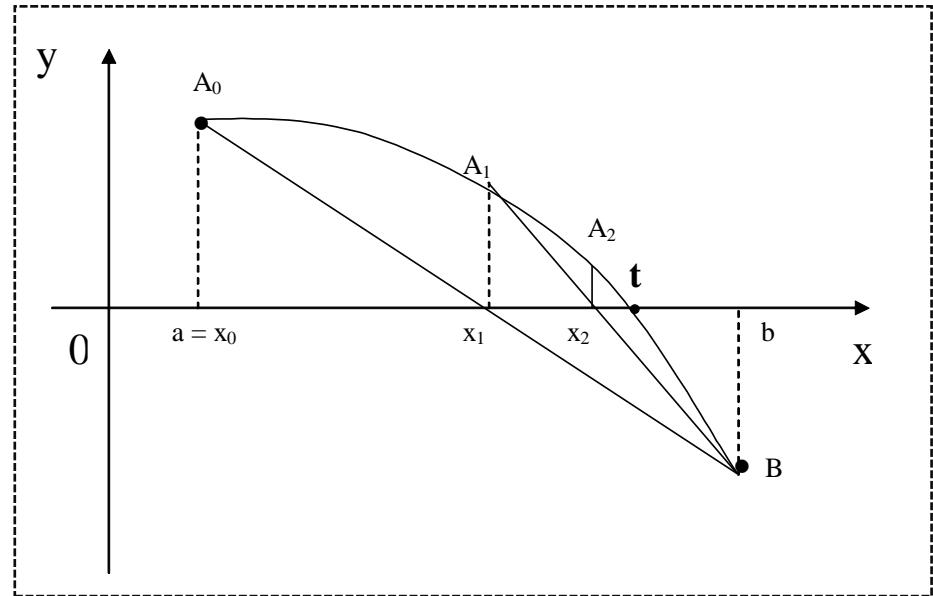
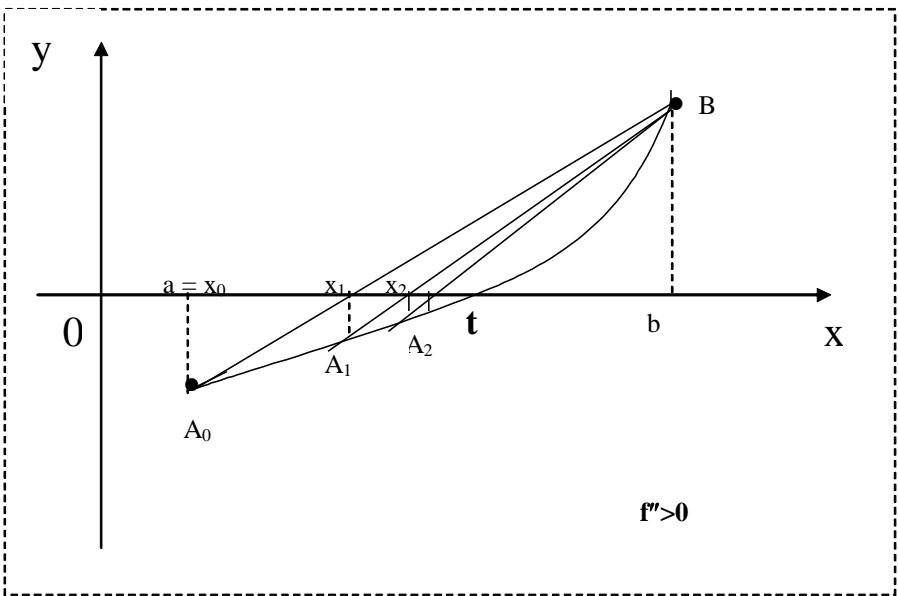
$$x_{n+1} = x_n - \frac{x_n - x_0}{f(x_n) - f(x_0)} f(x_n)$$

Для вычислений  $x_2$  по этой формуле надо указать два начальных приближения  $x_0$  и  $x_1$ . Геометрически в методе хорд касательная к кривой заменяется хордой, проходящей через точки кривой  $y = f(x)$  с абсциссами  $x_n(b)$  и  $x_0(a)$ .





$$f'(x) \cdot f''(x) > 0$$





Уравнение хорды  $A_0B$ :  $\frac{y - f(a)}{f(b) - f(a)} = \frac{x - a}{b - a}$

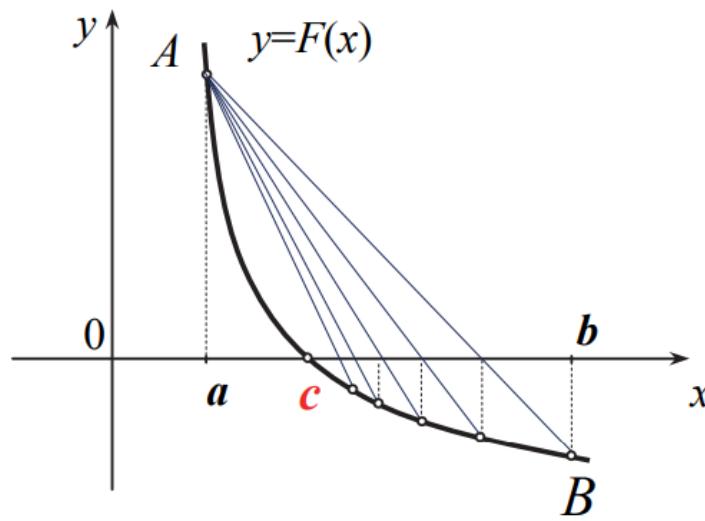
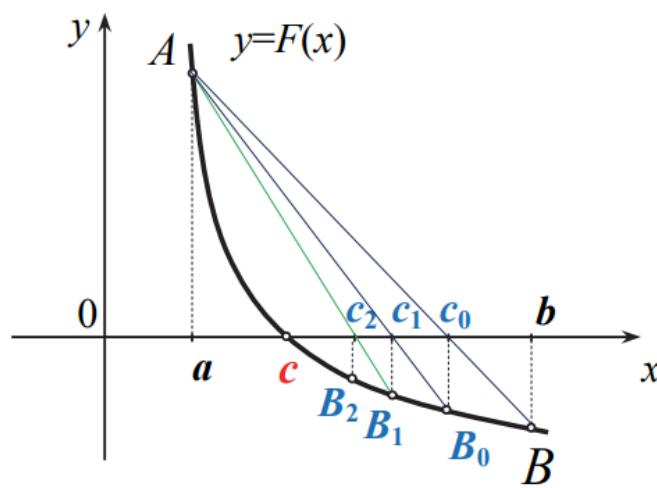
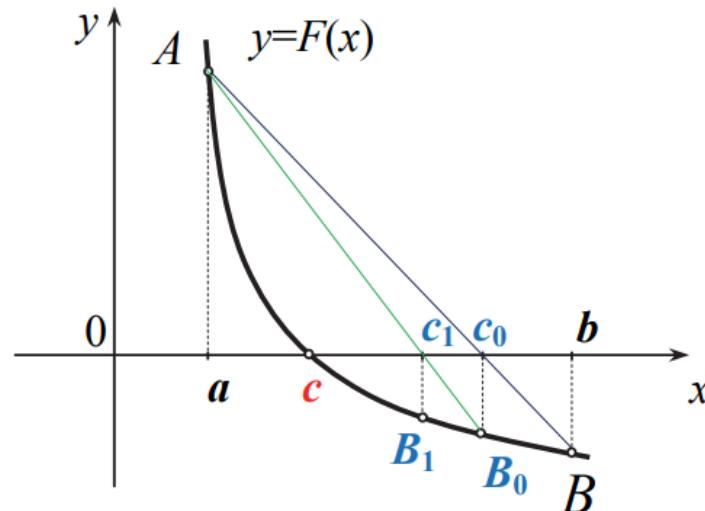
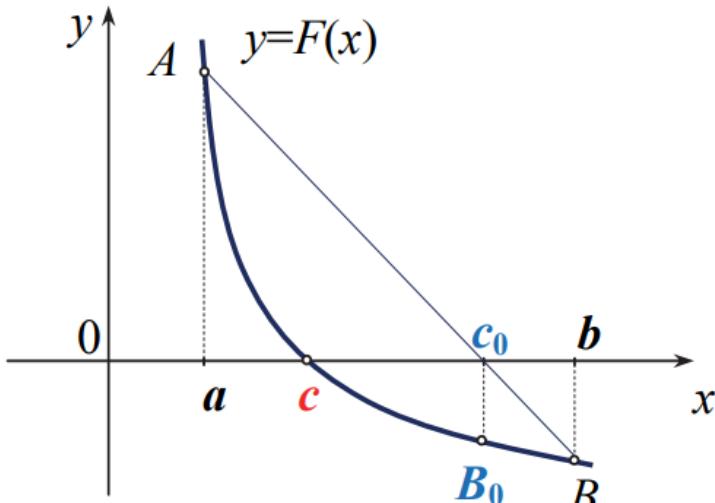
из уравнения прямой, проходящей через две точки:

$$\frac{y - y_1}{y_2 - y_1} = \frac{x - x_1}{x_2 - x_1}$$

Найдем значение  $x = x_1$  для которого  $y = 0$ .

$$x_1 = a - \frac{f(a)(b - a)}{f(b) - f(a)}$$
 или 
$$x_1 = x_0 - \frac{f(x_0)(b - x_0)}{f(b) - f(x_0)}$$

Это формула метода хорд для первого шага.





Сейчас корень внутри отрезка  $[x_1, b]$ .

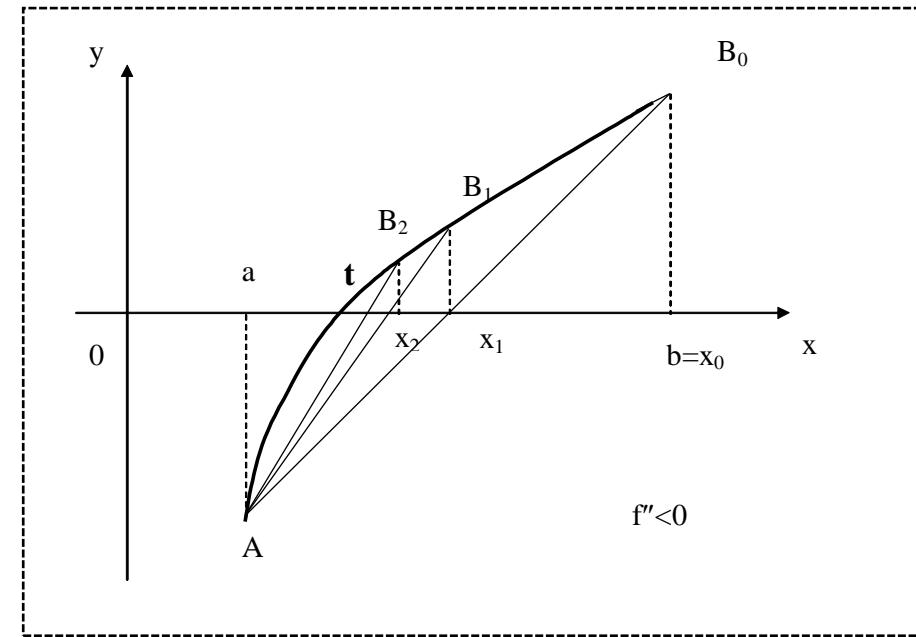
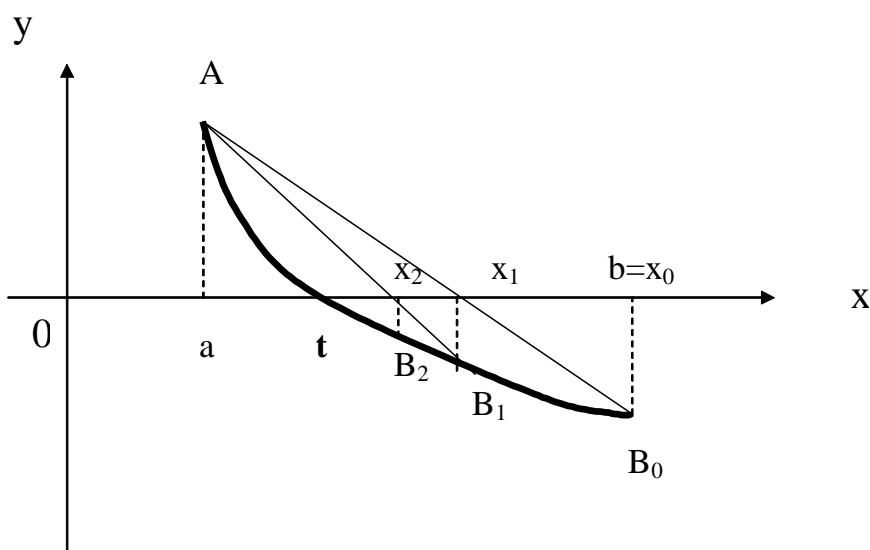
Далее:

$$x_{n+1} = x_n - \frac{f(x_n)(b - x_n)}{f(b) - f(x_n)}$$

Здесь В – неподвижный конец хорды.



$$f'(x) \cdot f''(x) < 0$$





Уравнение хорды  $AB_0$ :

$$\frac{y - f(a)}{f(b) - f(a)} = \frac{x - b}{b - a}$$

Найдем точку  $x_1$  при  $y=0$ .

$$x_1 = b - \frac{f(b)(b - a)}{f(b) - f(a)}; \quad x_2 = x_1 - \frac{f(x_1)(x_1 - a)}{f(x_1) - f(a)}$$

Итак до  $(n+1)$ -го шага:

$$x_{n+1} = x_n - \frac{f(x_n)(x_n - a)}{f(x_n) - f(a)}$$

Здесь А-неподвижный конец хорды.



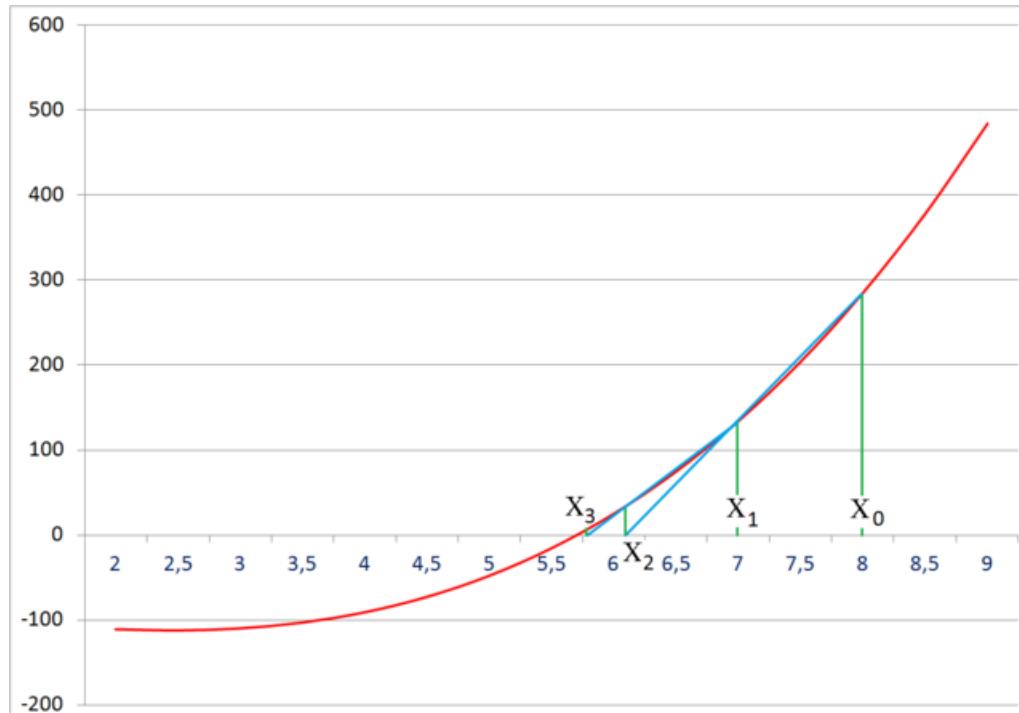
Условие неподвижной точки для метода хорд  $f(x) f''(x) > 0$ . В принципе, оба конца отрезка могут быть подвижными. Метод хорд, как правило, сходится медленнее, чем следующий метод (касательных).

Метод работает и в случае, если начальные точки выбраны по одну и ту же сторону от корня (то есть, корня нет на отрезке между начальными приближениями), но при этом возможны случаи, когда метод не сходится.

В качестве критерия остановки берут один из следующих:

$f(x_k) < \varepsilon$  – значение функции на данной итерации стало меньше заданного  $\varepsilon$ .

$|x_k - x_{k-1}| < \varepsilon$  – изменение  $x_k$  в результате итерации стало меньше заданного  $\varepsilon$ .





Этот метод – модификация метода хорд (секущих). В нём при расчете приближенного значения производной используется вместо точки  $x_{k-1} - h$  в  $x_k = x_{k-1} - \frac{f(x_{k-1})h}{f(x_{k-1}) - f(x_{k-1}-h)} = \phi(x_{k-1})$  точка  $x_{k-2}$ , полученная на предыдущей итерации.

Расчетная формула метода Вегстейна:

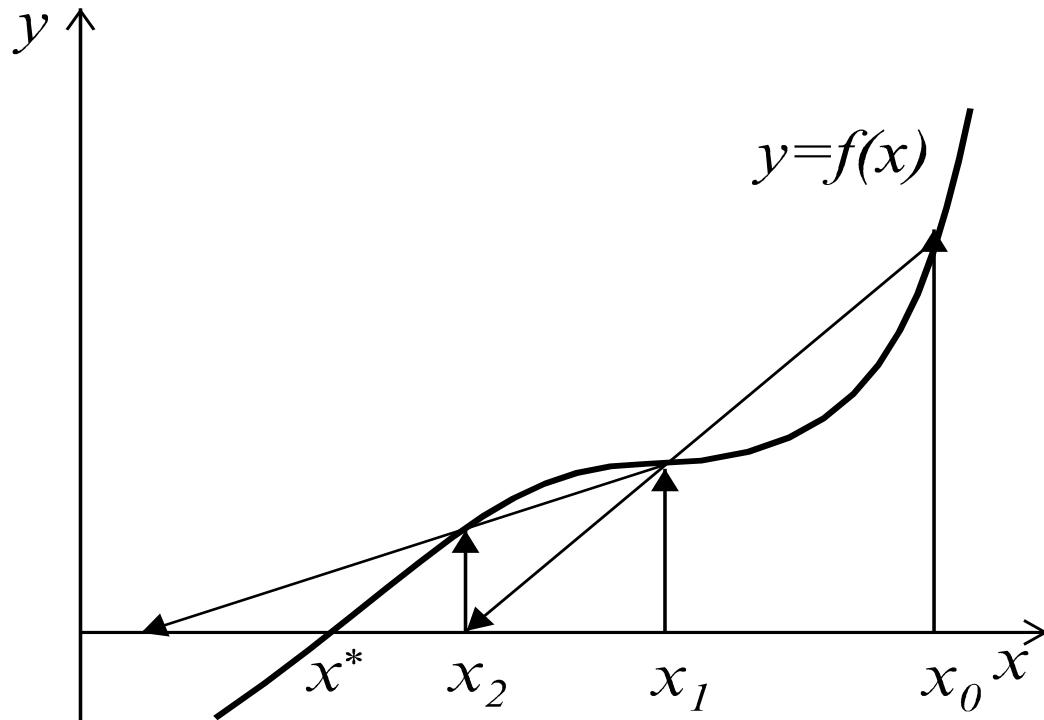
$$x_k = x_{k-1} - \frac{f(x_{k-1})(x_{k-1} - x_{k-2})}{f(x_{k-1}) - f(x_{k-2})} = \phi(x_{k-1}, x_{k-2}).$$



Метод двухшаговый ( $m = 2$ ), т. к. для начала вычислений требуется задать 2 начальных приближения.

Обычно выбирают:  $x_0 = \alpha$ ,  $x_1 = \beta$ .

Метод Вегстейна сходится медленнее метода хорд (секущих), однако, требует в 2 раза меньше вычислений  $f(x)$  и за счет этого оказывается более эффективным.





Этот метод – также модификация метода простой итерации. Идея аналогична той, которая реализована в методе хорд, только в качестве прямой берется касательная, проводимая в текущей точке.

В отличие от метода деления отрезка пополам перед применением метода касательных нужно проверить условия сходимости:

- функция  $f(x)$  должна быть дважды непрерывно дифференцируема на отрезке  $[a, b]$ , на котором отделен корень;
- первая и вторая производные функции  $f'(x)$  и  $f''(x)$  должны сохранять постоянные знаки на отрезке  $[a, b]$ .

Выполнение этих условий гарантирует, что за определенное число шагов мы уточним корень с любой наперед заданной погрешностью  $\epsilon$ . Иногда удается вычислить корень уравнения и в том случае, когда условия сходимости не выполняются.



Метод одношаговый ( $m = 1$ ), т.к. для начала вычислений требуется задать одно начальное приближение  $x_0$  из области сходимости  $x_0 = \alpha$  при  $f(\alpha)f''(\alpha) > 0$ , или  $x_0 = \beta$  при  $f(\beta)f''(\beta) > 0$ .

Метод Ньютона получил название «**метод касательных**» благодаря геометрической иллюстрации его сходимости.

Основной его недостаток – малая область сходимости и необходимость вычисления производной  $f'(x)$ .

Какая связь с методом итераций?

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



Итак, метод применим к выпуклым и монотонным функциям.

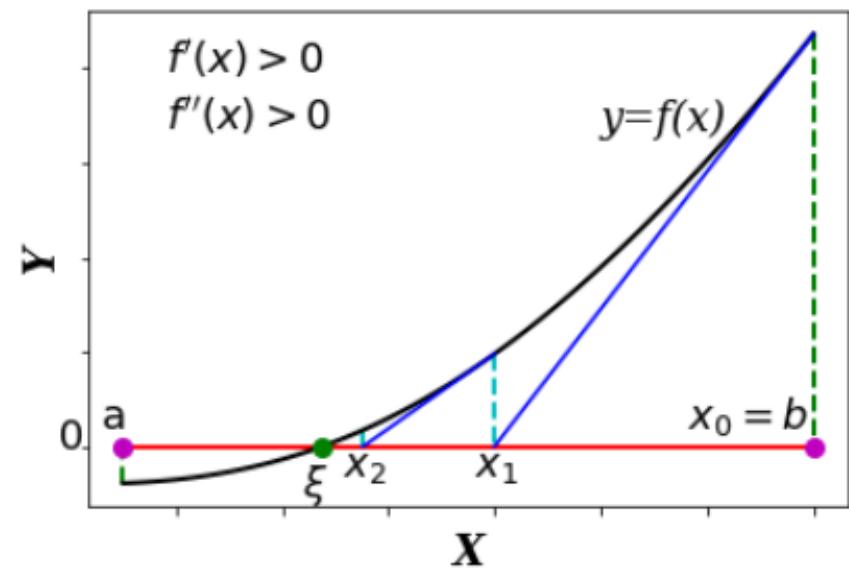
Выбор начальной точки зависит от свойств функции:

- если  $f(a)f''(a) > 0$ , то  $x_0 = a$
- если  $f(b)f''(b) > 0$ , то  $x_0 = b$

Очередное приближение  $x_{i+1}$  вычисляется по формуле:

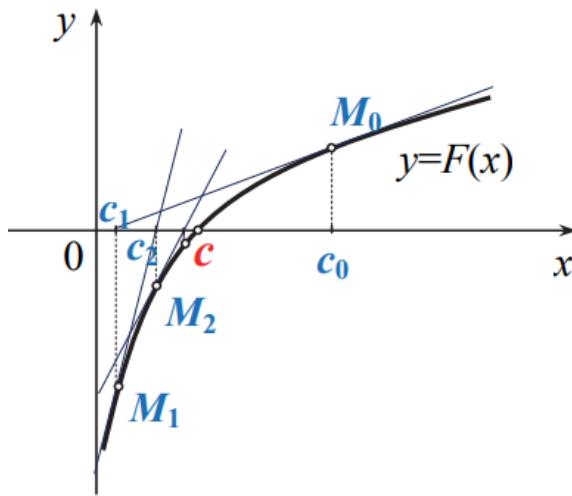
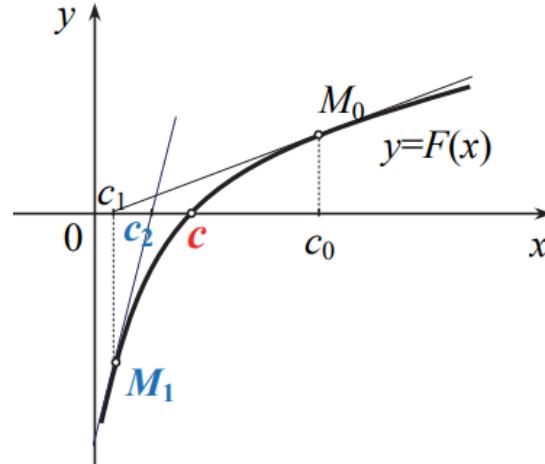
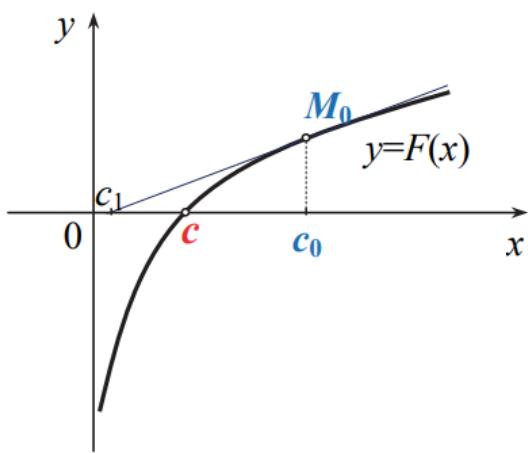
$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

до тех пор, пока не выполнится условие  $|x_{n+1} - x_n| < \varepsilon$ . Если это условие выполняется, то  $x_{n+1}$  считают приближенным значением корня с погрешностью  $\varepsilon$ .





Случай, когда  $f(a) \cdot f(b) < 0$ , и  $f'(x) \cdot f''(x) < 0$  :



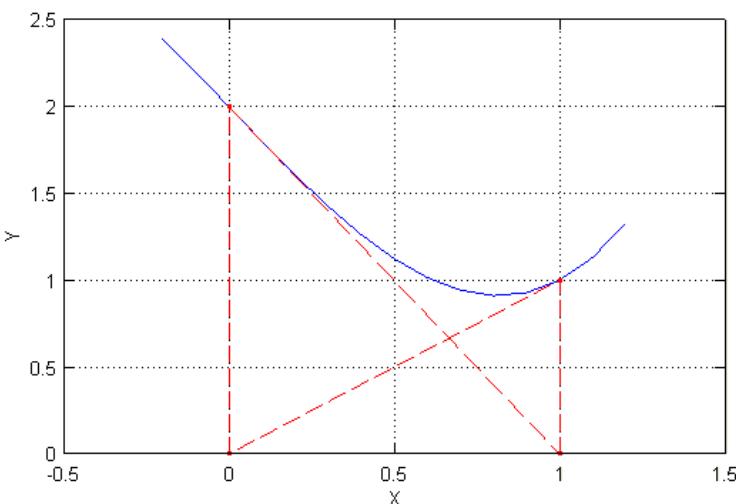
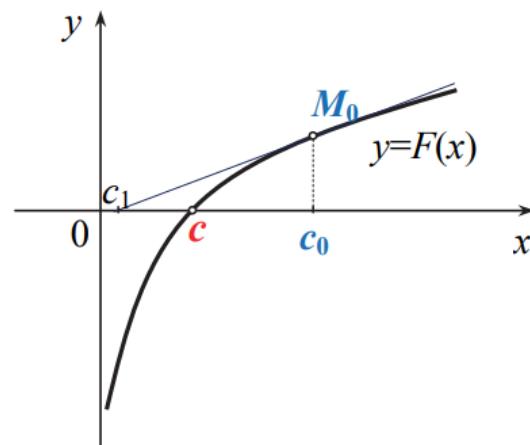
Пусть  $x = t$  принадлежит отрезку  $[a, b]$  и является корнем уравнения  $f(x) = 0$ , а функция  $f(x)$  дважды непрерывно дифференцируема на данном отрезке.

Тогда найдется такая  $\delta$ -окрестность точки  $x = t$ , что при любом выборе начального приближения  $x_0$  на отрезке  $[t-\delta, t+\delta] \in [a, b]$ .

Существует бесконечная итерационная последовательность  $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$  и эта последовательность сходится к корню  $t$ .

Достаточное условие сходимости метода Ньютона:  
 $|f(x) \cdot f''(x)| < [f'(x)]^2$ .

За исходную точку (нулевое приближение) следует выбирать тот конец отрезка  $[a, b]$ , в котором знак функции совпадает со знаком второй производной.



Если начальное приближение недостаточно близко к решению, то метод может не сойтись!



при заданной точности  $\varepsilon > 0$  вычисления следует вести до тех пор, пока не окажется выполненным неравенство:

$$|x_n - x_{n-1}| < \varepsilon.$$

Оценка погрешности метода:

$$|t - x_n| \leq \frac{|f(x_n)|}{m}, \quad \text{где} \quad m = \min_{[a,b]} |f'(x)|$$



Найти методом касательных приближенное значение корня уравнения

$$f(x) = x - \cos x = 0$$

на интервале  $[0,5; 1]$ .

Рекуррентная формула метода касательных принимает в данном случае вид:

$$x_{n+1} = x_n - \frac{x_n - \cos x_n}{1 + \sin x_n}$$



Выберем в качестве нулевого приближения  $x_0 = 0.5$  и подсчитаем следующие приближения. Результаты вычислений приведены в таблице:

N	$x_n$
0	0,500000000000
1	0,755222417106
2	0,739141666150
3	0,739085133921
4	0,739085133215
5	0,739085133215

**После двух шагов достигаем точности  $10^{-4}$ .**



Рассмотрим вычисление  $\sqrt{2}$  как задачу решения уравнения:  $x^2 - a = 0$  в области  $x > 0$ . Пропишем для вычисления корня уравнения итерационную последовательность по методу касательных. Вычислим с её помощью  $\sqrt{2}$ .

Рекуррентная формула метода касательных для уравнения  $x = \sqrt{a}$  принимает вид:

$$x_{n+1} = x_n - \frac{x_n^2 - a}{2x_n} = \frac{1}{2} \left( x_n + \frac{a}{x_n} \right)$$



Перейдём к вычислению  $\sqrt{2}$ . Помним, что  $\sqrt{2} = 1,414214$ .

Выбирая  $x_0 = 2$ , делаем несколько итераций по предыдущей формуле:

$$x_0 = 2, \quad x_1 = 1,5.$$

$$x_2 = \frac{1}{2} \left( \frac{3}{2} + \frac{4}{3} \right) = 1,416666, \quad x_3 = \frac{1}{2} \left( \frac{17}{12} + \frac{24}{17} \right) = 1,414216.$$

Третий шаг определяет  $\sqrt{2}$  с погрешностью:

$$\Delta = |\sqrt{2} - x_3| = 0,000002.$$



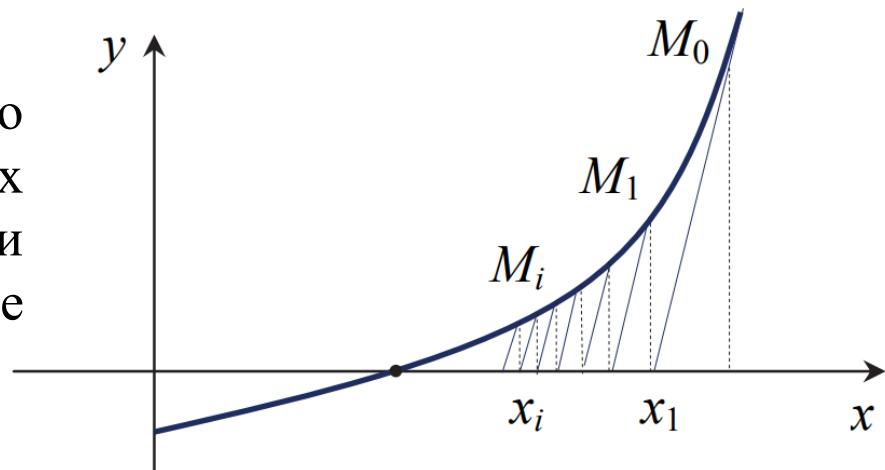
Если производная  $f'(x)$  мало изменяется на отрезке  $[a, b]$ , то можно положить:

$$f'(x_i) \approx f'(x_0).$$

Тогда следующее приближение вычисляется по формуле:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_0)}$$

Геометрически этот способ означает, что мы заменяем касательную в точках  $M_n[x_n, f(x_n)]$  прямыми, параллельными касательной к кривой  $y = f(x)$ , в ее фиксированной точке  $M_0[x_0, f(x_0)]$ .





## Домашняя работа #1

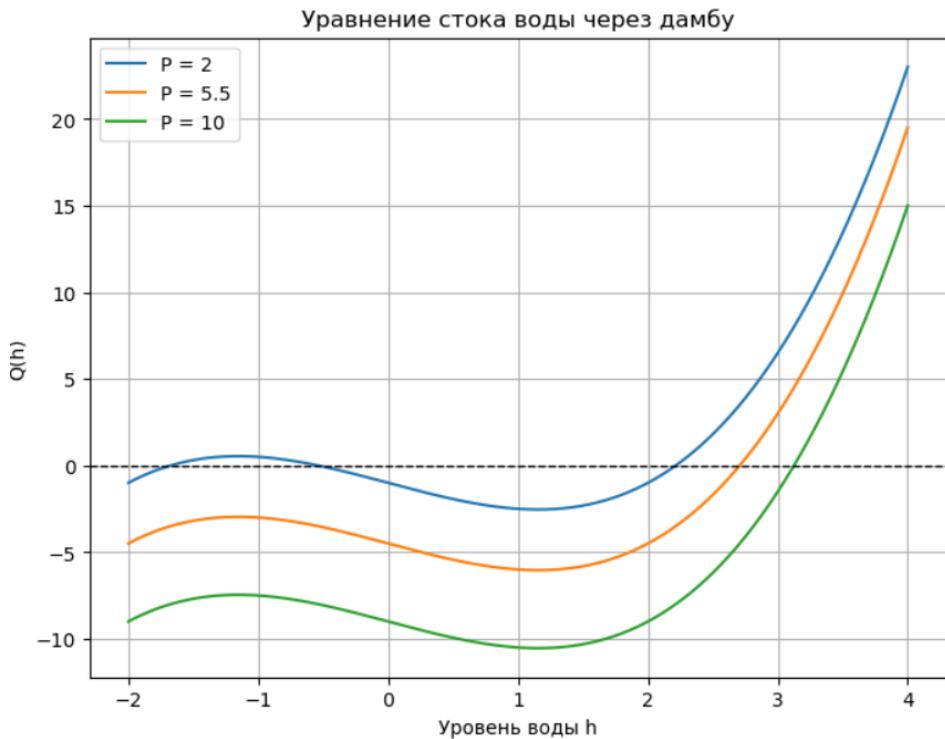
Решите предложенную далее задачу о поиске устойчивого уровня воды при помощи модифицированного метода касательных.

Разрешается использование исключительно базовых методов numpy, matplotlib, не более.

Срок сдачи работ: **до 22-го февраля.**

Работы принимаются на электронный адрес [VSChernyshenko@fa.ru](mailto:VSChernyshenko@fa.ru) в виде ноутбука в формате html. Название файла: “Фамилия\_Имя\_группа\_ДЗ1.\*”

**Не принимаются** работы являющиеся копией друг друга, работы **не** содержащие комментарии.





## Домашняя работа #1

Вы разрабатываете систему автоматического контроля уровня воды в водохранилище, питающем гидроэлектростанцию. Водохранилище имеет сложную геометрию, а сток воды через дамбу регулируется нелинейным уравнением, моделирующим скорость потока:

$$Q(h) = k_1 \cdot h^3 - k_2 \cdot h + k_3 - P = 0$$

где:

$h$  — текущий уровень воды (переменная, которую нужно найти).

$k_1 = 0.5$ ,  $k_2 = 2.0$ ,  $k_3 = 1.0$  — коэффициенты, зависящие от формы водохранилища и характеристик дамбы.

$P$  — текущий объем осадков в пересчете на поток.



## Домашняя работа #1

### Обязательные подзадачи:

- Сгенерировать массив значений  $P$  от 2 до 10 с шагом 0.5 (добавив случайные возмущения).
- Найти устойчивый уровень воды  $h$ , при котором уравнение выполняется для заданных значений  $P$ .
- Используйте Python Line Profiler для поиска узких мест решения – напишите в комментариях результаты этого анализа.
- Проанализируйте влияния численного представления на точность и скорость решения задачи:
  - Рассчитайте уровень воды  $h$ , храня все переменные поочерёдно в форматах: `numpy.float64`, `numpy.float32`, `numpy.float16`.
  - Постройте графики размера ошибки и времени работы алгоритма в зависимости от используемых форматов хранения данных.
  - Найдите предельные абсолютную и относительную погрешности результата, определите число верных значащих цифр при хранении результатов в `float64`, `float32`, `float16`.
  - Напишите в комментариях отчёт, как ограниченная разрядность влияет на результат, скорость и устойчивость метода.



В прикладной математике задачи оптимизации поиск максимума и минимума функции – один из важнейших разделов.

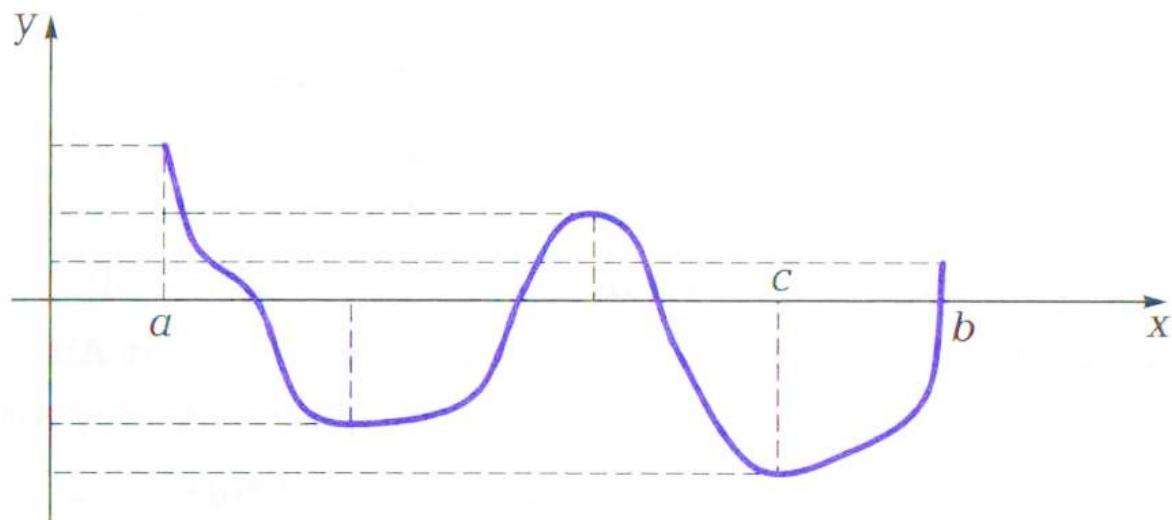
Стандартной задачей оптимизации является поиск экстремумов функции одной переменной.

Функция представленная на рисунке имеет несколько минимумов и максимумов на отрезке  $[a; b]$ :

$x = a$  – самый «высокий» из максимумов,

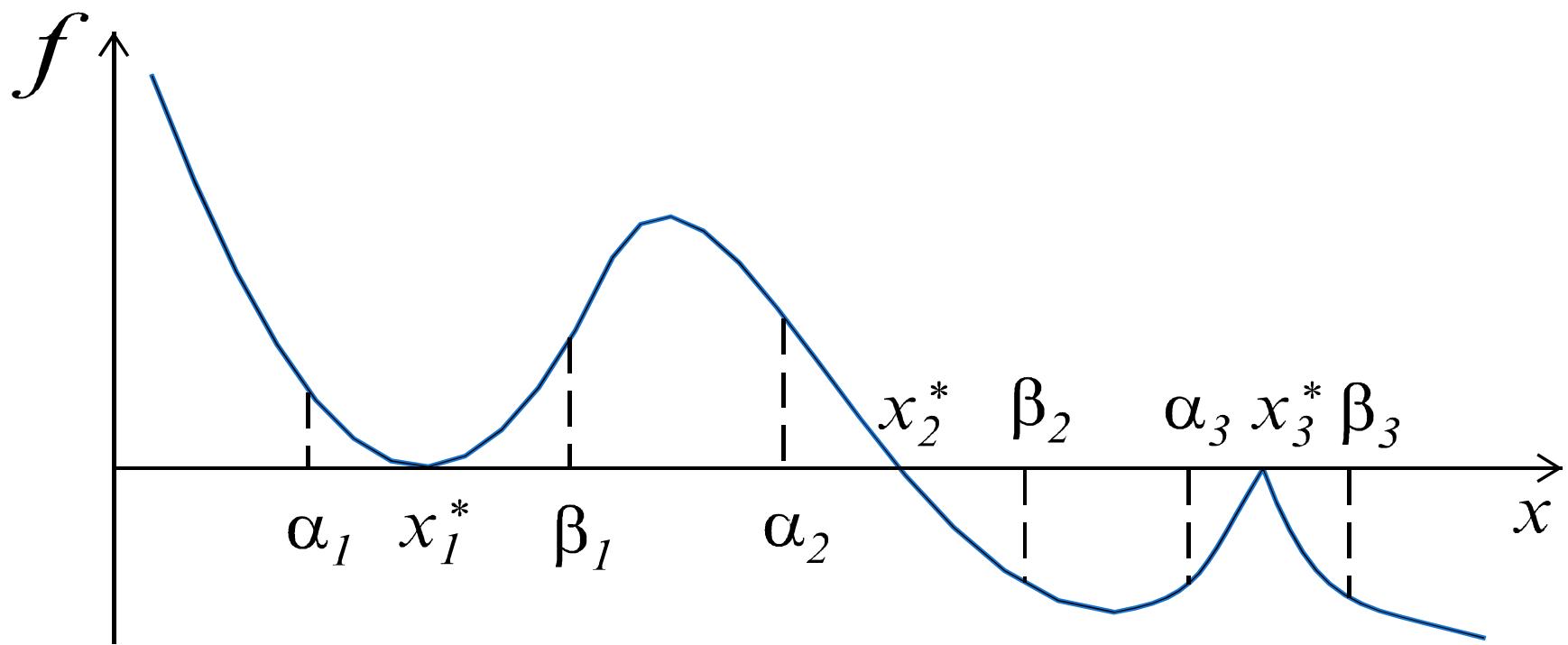
$x = c$  – самый «низкий» из минимумов.

Данные  $\max$  и  $\min$  на данном отрезке являются глобальными.





В случаях кратных и вырожденных корней, значение корня совпадает с точкой экстремума функции и для нахождения таких корней, назовем их особыми, рекомендуется использовать методы поиска минимума (максимума) функции.





Если функция является дифференцируемой на отрезке  $[a; b]$  то:

1. Точка  $x_0$  является точкой внутреннего экстремумов, если она является корнем уравнения  $f'(x)=0$ .
2. Если у функции существует вторая производная, то:
  - при  $f''(x) > 0$   $x_0$  является точкой минимума;
  - при  $f''(x) < 0$   $x_0$  является точкой максимума.



Метод половинного деления легко переносится на задачу уточнения положения точки минимума функции.

Примем  $e$  – точность.

Вычислим два значения для  $f(c)$ :  $f(c - e/2)$  и  $f(c + e/2)$

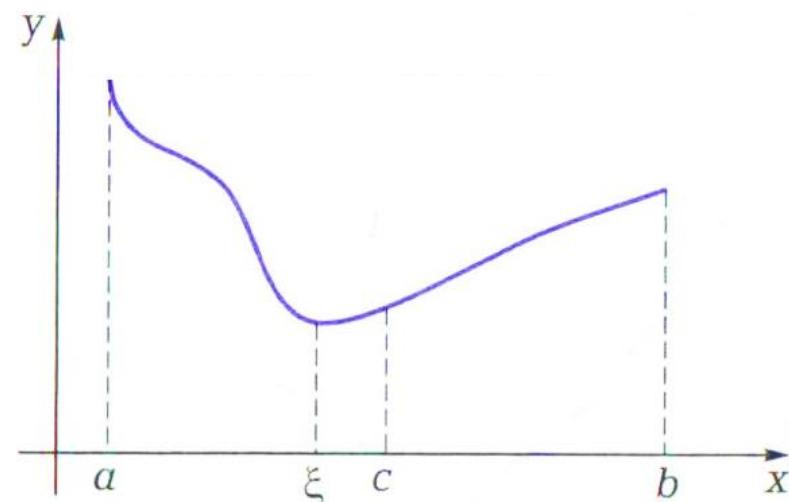
- при  $f(c - e/2) < f(c + e/2)$  делению пополам подлежит отрезок  $[a; c]$ ;
- при  $f(c - e/2) > f(c + e/2)$  делению пополам подлежит отрезок  $[c; b]$ .

Итерационный вычислительный процесс длится до тех пор пока длина очередного отрезка не станет меньше  $e$ .

Возможна ситуация:

$$f(c - e/2) = f(c + e/2)$$

Т.е. искомая точка находится между  $f(c - e/2)$  и  $f(c + e/2)$ . В этом случае результат решения задачи = с.





# ЧИСЛЕННЫЕ МЕТОДЫ РЕШЕНИЯ СИСТЕМ НЕЛИНЕЙНЫХ УРАВНЕНИЙ



## Постановка задачи

Требуется решить систему нелинейных уравнений (СНУ) вида.

$$\begin{cases} F_1(x_1, x_2, \dots, x_n) = 0, \\ F_2(x_1, x_2, \dots, x_n) = 0, \\ \dots \\ F_n(x_1, x_2, \dots, x_n) = 0, \end{cases}$$

где  $F_i(x_1, x_2, \dots, x_n) = 0$  – нелинейные уравнения ( $i = 1, 2, \dots, n$ ).

Очевидно, что численное решение систем таких уравнений тоже возможно только с помощью итерационных методов, т.е. универсальных прямых методов для решения любых систем нелинейных уравнений не существует.



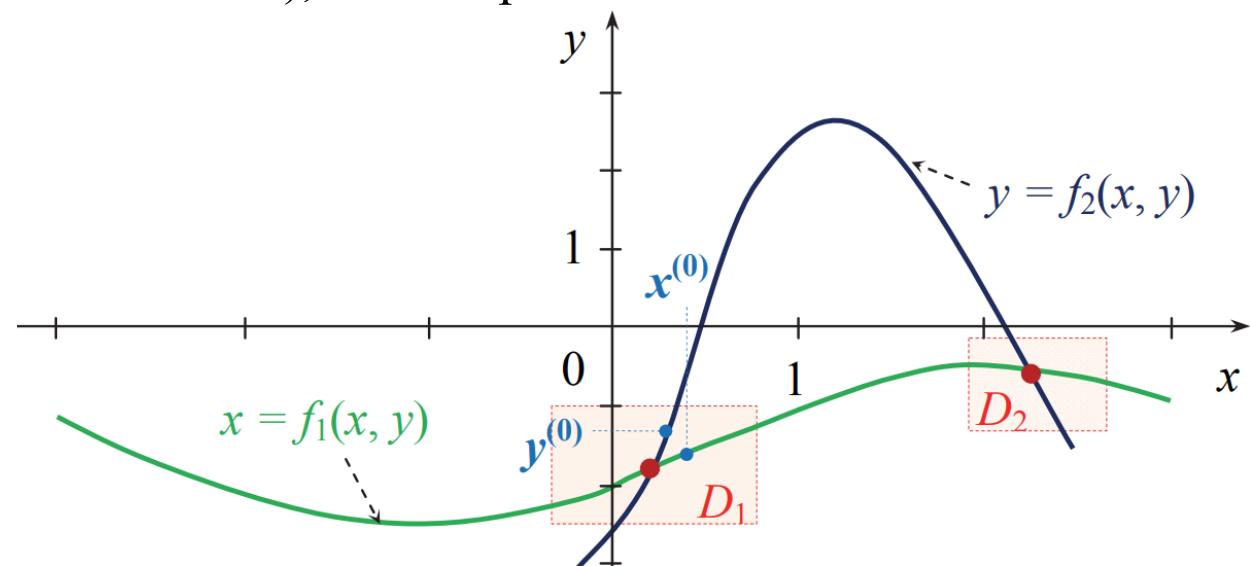
## Метод простых итераций:

Для построения схемы интеграций систему уравнений представим в виде

$$\begin{cases} x_1 = f_1(x_1, x_2, \dots, x_n), \\ x_2 = f_2(x_1, x_2, \dots, x_n), \\ \dots \\ x_n = f_n(x_1, x_2, \dots, x_n), \end{cases}$$

т.е. из каждого уравнения выразим по одной переменной  $x_i$ .

Первый этап решения СНУ – отделение корней, т.е. нахождения областей  $D_k$  ( $k$  – число корней системы), в которых система имеет только одно решение.





## Второй этап – уточнение корня.

Начальное приближение корней  $x_1^{(0)}, x_2^{(0)}, \dots, x_n^{(0)}$  выбирают из области  $D_k$ .

Подставляя эти значения в правую часть системы получим следующие приближения переменных  $x_1^{(1)}, x_2^{(1)}, \dots, x_n^{(1)}$ :

$$\begin{cases} x_1^{(1)} = f_1(x_1^{(0)}, x_2^{(0)}, x_3^{(0)}, \dots, x_n^{(0)}), \\ x_2^{(1)} = f_2(x_1^{(0)}, x_2^{(0)}, x_3^{(0)}, \dots, x_n^{(0)}), \\ x_3^{(1)} = f_3(x_1^{(0)}, x_2^{(0)}, x_3^{(0)}, \dots, x_n^{(0)}), \\ \dots \dots \dots \dots \dots \dots \dots \dots \\ x_n^{(1)} = f_n(x_1^{(0)}, x_2^{(0)}, x_3^{(0)}, \dots, x_n^{(0)}), \end{cases}$$

которые используются для получения 2-го приближения неизвестных и т.д.



Таким образом, схема итераций метода простых интеграций имеет вид:

$$\begin{cases} x_1^{(k+1)} = f_1(x_1^{(k)}, x_2^{(k)}, x_3^{(k)}, \dots, x_n^{(k)}), \\ x_2^{(k+1)} = f_2(x_1^{(k)}, x_2^{(k)}, x_3^{(k)}, \dots, x_n^{(k)}), \\ x_3^{(k+1)} = f_3(x_1^{(k)}, x_2^{(k)}, x_3^{(k)}, \dots, x_n^{(k)}), \\ \dots \dots \dots \dots \dots \dots \\ x_n^{(k+1)} = f_n(x_1^{(k)}, x_2^{(k)}, x_3^{(k)}, \dots, x_n^{(k)}), \end{cases}$$

где  $k = 0, 1, 2, \dots$ .

Для более быстрой сходимости итерационного процесса, начальное приближение следует выбирать достаточно близким к точному значению корня.

Условия остановки процесса итераций:

$$\max_i |x_i^{(k+1)} - x_i^{(k)}| < \varepsilon \text{ или } \max_i |f(x_i^{(k+1)})| < \varepsilon,$$

где  $i = \overline{1, n}$ ,  $k = 0, 1, 2, \dots$ .



Для сходимости системы процесса итераций достаточно, чтобы в области уточнения  $D_k$  выполнялось одно из следующих условий:

$$1. \max_{1 \leq i \leq n} \sum_{j=1}^n \left| \frac{\partial f_i}{\partial x_j} \right| < 1;$$

$$2. \max_{1 \leq j \leq n} \sum_{i=1}^n \left| \frac{\partial f_i}{\partial x_j} \right| < 1;$$

$$3. \sqrt{\sum_{i,j=1}^n \left( \frac{\partial f_i}{\partial x_j} \right)^2} < 1.$$

Эти условия получаются аналогично таковым в методах простых итераций и Зейделя при решении СЛАУ.



**Пример.** Используя метод простых итераций, решить следующую систему нелинейных уравнений с точностью до 0,001.

$$\begin{cases} \sin(x - 0,5) - y = 1,5 \\ 2x - \cos y = 0,6 \end{cases}$$

Решение.

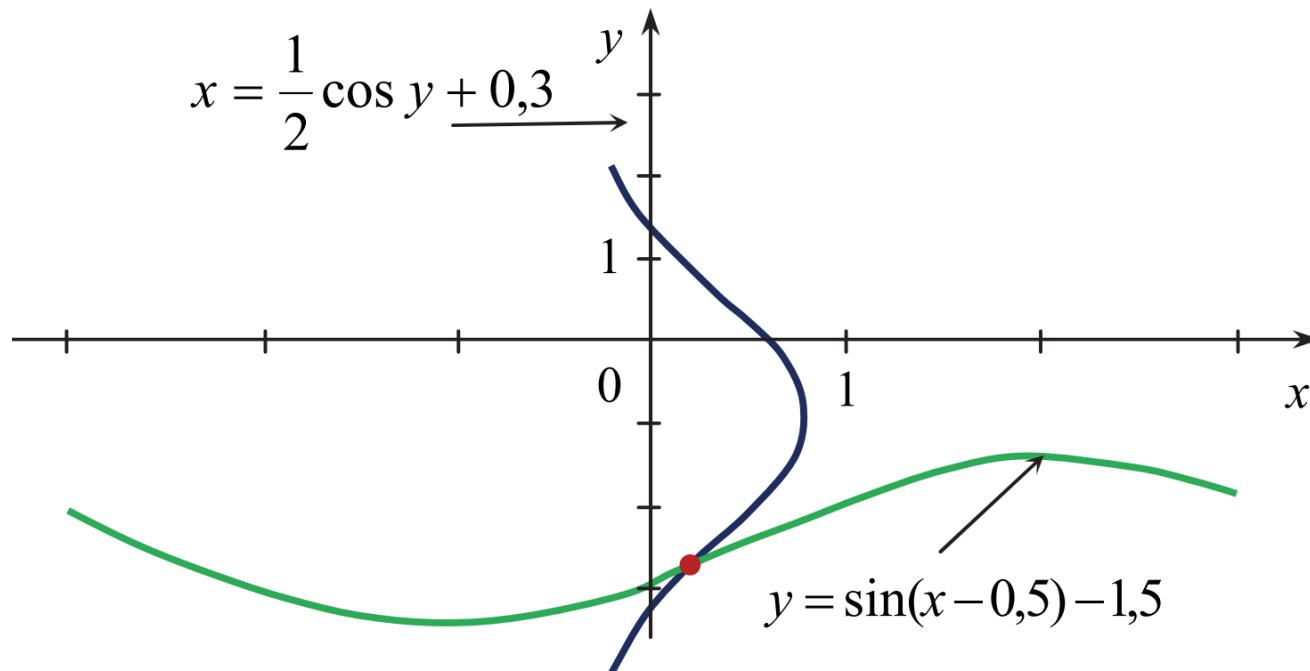
Перепишем данную систему в виде

$$\begin{cases} y = \sin(x - 0,5) - 1,5 \\ x = \frac{1}{2} \cos y + 0,3 \end{cases}$$

(из второго уравнения выразили  $x$ , из первого –  $y$ ).



Отделение корней произведем графически. Из графика видно, что система имеет одно решение, заключенное в области  $D: 0 < x < 0,25; -2 < y < -1,5$ .





Убедимся в том, что метод простых итераций применим для уточнения решения системы, для чего запишем ее в следующем виде:

$$\begin{cases} x = f_1(x, y) = \frac{1}{2} \cos y + 0,3 \\ y = f_2(x, y) = \sin(x - 0,5) - 1,5 \end{cases}$$

В области  $D \{0 < x < 0,25; -2 < y < -1,5\}$  имеем

$$\left| \frac{\partial f_1}{\partial y} \right| + \left| \frac{\partial f_2}{\partial y} \right| = \left| -\frac{1}{2} \sin y \right| \leq 0,5 < 1,$$

т.е. условие сходимости выполняется. Следовательно, в этой области  $D$ , для уточнения корней можно использовать схему:

$$\begin{cases} x_{k+1} = \frac{1}{2} \cos y_k + 0,3 \\ y_{k+1} = \sin(x_k - 0,5) - 1,5 \end{cases}$$

За начальные приближения примем  $x_0 = 0,13; y_0 = -1,8$ .



Номер итерации, $k$	Приближение корня		Проверка условия: $ x^{(k)} - x^{(k-1)}  < \varepsilon = 0,001$	Проверка условия: $ y^{(k)} - y^{(k-1)}  < \varepsilon = 0,001$
	$x^{(k)}$	$y^{(k)}$		
0	0,13	-1,8		
1	0,186399	-1,86162	не выполняется	не выполняется
2	0,156631	-1,80849	не выполняется	не выполняется
3	0,182271	-1,83666	не выполняется	не выполняется
4	0,168628	-1,81241	не выполняется	не выполняется
5	0,180365	-1,82534	не выполняется	не выполняется
6	0,174098	-1,81422	не выполняется	не выполняется
7	0,179487	-1,82016	не выполняется	не выполняется
8	0,176605	-1,81505	не выполняется	не выполняется
8	0,179082	-1,81779	не выполняется	не выполняется
10	0,177756	-1,81544	не выполняется	не выполняется
11	0,178896	-1,8167	не выполняется	не выполняется
12	0,178286	-1,81561	<b>выполняется</b>	не выполняется
13	0,17881	-1,81619	<b>выполняется</b>	<b>выполняется</b>



Для ускорения сходимости можно ввести параметр  $b$ :

$$\begin{cases} x_{k+1} = (1 - b)x_k + \frac{1}{2}b \cdot \cos y_k + 0,3b \\ y_{k+1} = (1 - b)x_k + b \sin(x_k - 0,5) - 1,5b \end{cases}$$



Метод Зейделя представляет собой модификацию метода простых итераций. Используется для решения СНУ. Систему необходимо сначала привести к виду:

$$\begin{cases} x_1^{(1)} = f_1(x_1^{(0)}, x_2^{(0)}, x_3^{(0)}, \dots, x_n^{(0)}), \\ x_2^{(1)} = f_2(x_1^{(0)}, x_2^{(0)}, x_3^{(0)}, \dots, x_n^{(0)}), \\ x_3^{(1)} = f_3(x_1^{(0)}, x_2^{(0)}, x_3^{(0)}, \dots, x_n^{(0)}), \\ \dots \dots \dots \dots \dots \dots \dots \\ x_n^{(1)} = f_n(x_1^{(0)}, x_2^{(0)}, x_3^{(0)}, \dots, x_n^{(0)}), \end{cases}$$

Начальное приближение корней  $x_1^{(0)}, x_2^{(0)}, \dots, x_n^{(0)}$  выбирают из области  $D_k$ , содержащей один корень. Тогда итерации методом Зейделя выполняются по следующей схеме:

$$\begin{cases} x_1^{(k+1)} = f_1(x_1^{(k)}, x_2^{(k)}, x_3^{(k)}, \dots, x_n^{(k)}), \\ x_2^{(k+1)} = f_2(x_1^{(k+1)}, x_2^{(k)}, x_3^{(k)}, \dots, x_n^{(k)}), \\ x_3^{(k+1)} = f_3(x_1^{(k+1)}, x_2^{(k+1)}, x_3^{(k)}, \dots, x_n^{(k)}), \\ \dots \dots \dots \dots \dots \dots \dots \\ x_n^{(k+1)} = f_n(x_1^{(k+1)}, x_2^{(k+1)}, x_3^{(k+1)}, \dots, x_{n-1}^{(k+1)}, x_n^{(k)}), \end{cases}$$

Условия остановки процесса итераций совпадают с аналогичными при использовании метода простых итераций, а именно:

$$\max_i |x_i^{(k+1)} - x_i^{(k)}| < \varepsilon \text{ или } \max_i |f(x_i^{(k+1)})| < \varepsilon,$$

где  $i = \overline{1, n}$ ,  $k = 0, 1, 2, \dots$ .

Для сходимости системы процесса итераций достаточно, чтобы в области уточнения  $D_k$  выполнялось одно из следующих условий:

$$1. \quad \max_{1 \leq i \leq n} \sum_{j=1}^n \left| \frac{\partial f_i}{\partial x_j} \right| < 1;$$

$$2. \quad \max_{1 \leq j \leq n} \sum_{i=1}^n \left| \frac{\partial f_i}{\partial x_j} \right| < 1;$$

$$3. \quad \sqrt{\sum_{i,j=1}^n \left( \frac{\partial f_i}{\partial x_j} \right)^2} < 1.$$



**Пример.** Используя метод Зейделя, решить следующую систему нелинейных уравнений с точностью до 0,001.

$$\begin{cases} \sin(x - 0,5) - y = 1,5 \\ 2x - \cos y = 0,6 \end{cases}$$

Так как ранее система уже была приведена к виду удобному для итераций, сразу запишем схему итераций по методу Зейделя:

$$\begin{cases} x^{(k+1)} = \frac{1}{2} \cos y^{(k)} + 0,3 \\ y^{(k+1)} = \sin(x^{(k)} - 0,5) - 1,5 \end{cases}$$

Как и ранее, за начальные приближения примем  $x^{(0)} = 0,13$ ;  $y^{(0)} = -1,8$ .



Номер итерации, <i>k</i>	Приближение корня		Проверка условия: $ x^{(k)} - x^{(k-1)}  < \varepsilon = 0,001$	Проверка условия: $ y^{(k)} - y^{(k-1)}  < \varepsilon = 0,001$
	$x^{(k)}$	$y^{(k)}$		
0	0,13	-1,8		
1	0,186399	-1,80849	не выполняется	не выполняется
2	0,182271	-1,81241	не выполняется	не выполняется
3	0,180365	-1,81422	не выполняется	не выполняется
4	0,179487	-1,81505	<b>выполняется</b>	<b>выполняется</b>



Пусть требуется решить систему нелинейных уравнений вида:

$$\begin{cases} F_1(x_1, x_2, \dots, x_n) = 0, \\ F_2(x_1, x_2, \dots, x_n) = 0, \\ \dots \dots \dots \dots \dots \\ F_n(x_1, x_2, \dots, x_n) = 0. \end{cases}$$

Метод Ньютона решения системы состоит в построении итерационной последовательности:

$$\begin{cases} x_1^{(k)} = x_1^{(k-1)} + \Delta x_1^{(k-1)}, \\ x_2^{(k)} = x_2^{(k-1)} + \Delta x_2^{(k-1)}, \\ x_3^{(k)} = x_3^{(k-1)} + \Delta x_3^{(k-1)}, \\ \dots \dots \dots \dots \dots \\ x_n^{(k)} = x_n^{(k-1)} + \Delta x_n^{(k-1)}. \end{cases}$$



Для нахождения неизвестных  $\Delta x_i^{(0)}, \Delta x_i^{(1)}, \dots, \Delta x_i^{(k-1)}$

воспользуемся разложением функций  $F_i(x_1, x_2, \dots, x_n)$  в ряд Тейлора в окрестности начального приближения.

Такое разложение для рассматриваемых функций в окрестности точки  $x^{(0)}$  с координатами  $x_1^{(0)}, x_2^{(0)}, \dots, x_n^{(0)}$  имеет вид:

$$\begin{cases} F_1(x_1, x_2, \dots, x_n) = F_1(x^{(0)}) + \Delta x_1^{(0)} \frac{\partial F_1}{\partial x_1}(x^{(0)}) + \Delta x_2^{(0)} \frac{\partial F_1}{\partial x_2}(x^{(0)}) + \cdots + \Delta x_n^{(0)} \frac{\partial F_1}{\partial x_n}(x^{(0)}) + R_1, \\ F_2(x_1, x_2, \dots, x_n) = F_2(x^{(0)}) + \Delta x_1^{(0)} \frac{\partial F_2}{\partial x_1}(x^{(0)}) + \Delta x_2^{(0)} \frac{\partial F_2}{\partial x_2}(x^{(0)}) + \cdots + \Delta x_n^{(0)} \frac{\partial F_2}{\partial x_n}(x^{(0)}) + R_2, \\ \dots \dots \dots \dots \dots \dots \\ F_n(x_1, x_2, \dots, x_n) = F_n(x^{(0)}) + \Delta x_1^{(0)} \frac{\partial F_n}{\partial x_1}(x^{(0)}) + \Delta x_2^{(0)} \frac{\partial F_n}{\partial x_2}(x^{(0)}) + \cdots + \Delta x_n^{(0)} \frac{\partial F_n}{\partial x_n}(x^{(0)}) + R_n. \end{cases}$$



Пренебрегая остаточными членами  $R_i$ , получаем СЛАУ с неизвестными  $\Delta x_i^{(0)}$ :

$$\begin{cases} F_1(x^{(0)}) + \Delta x_1^{(0)} \frac{\partial F_1}{\partial x_1}(x^{(0)}) + \Delta x_2^{(0)} \frac{\partial F_1}{\partial x_2}(x^{(0)}) + \cdots + \Delta x_n^{(0)} \frac{\partial F_1}{\partial x_n}(x^{(0)}) = 0, \\ F_2(x^{(0)}) + \Delta x_1^{(0)} \frac{\partial F_2}{\partial x_1}(x^{(0)}) + \Delta x_2^{(0)} \frac{\partial F_2}{\partial x_2}(x^{(0)}) + \cdots + \Delta x_n^{(0)} \frac{\partial F_2}{\partial x_n}(x^{(0)}) = 0, \\ \dots \\ F_n(x^{(0)}) + \Delta x_1^{(0)} \frac{\partial F_n}{\partial x_1}(x^{(0)}) + \Delta x_2^{(0)} \frac{\partial F_n}{\partial x_2}(x^{(0)}) + \cdots + \Delta x_n^{(0)} \frac{\partial F_n}{\partial x_n}(x^{(0)}) = 0. \end{cases}$$

перенеся свободные коэффициенты  $F_i(x^{(0)})$  вправо, получим следующую СЛАУ для вычисления  $\Delta x_i^{(0)}$ :



$$\left\{ \begin{array}{l} \Delta x_1^{(0)} \frac{\partial F_1}{\partial x_1}(x^{(0)}) + \Delta x_2^{(0)} \frac{\partial F_1}{\partial x_2}(x^{(0)}) + \cdots + \Delta x_n^{(0)} \frac{\partial F_1}{\partial x_n}(x^{(0)}) = -F_1(x^{(0)}), \\ \Delta x_1^{(0)} \frac{\partial F_2}{\partial x_1}(x^{(0)}) + \Delta x_2^{(0)} \frac{\partial F_2}{\partial x_2}(x^{(0)}) + \cdots + \Delta x_n^{(0)} \frac{\partial F_2}{\partial x_n}(x^{(0)}) = -F_2(x^{(0)}), \\ \dots \dots \dots \dots \dots \dots \\ \Delta x_1^{(0)} \frac{\partial F_n}{\partial x_1}(x^{(0)}) + \Delta x_2^{(0)} \frac{\partial F_n}{\partial x_2}(x^{(0)}) + \cdots + \Delta x_n^{(0)} \frac{\partial F_n}{\partial x_n}(x^{(0)}) = -F_n(x^{(0)}). \end{array} \right.$$

Запишем эту систему в матричной форме:

$$\begin{bmatrix} \frac{\partial F_1}{\partial x_1}(x_1^{(0)}, \dots, x_n^{(0)}) & \frac{\partial F_1}{\partial x_2}(x_1^{(0)}, \dots, x_n^{(0)}) & \cdots & \frac{\partial F_1}{\partial x_n}(x_1^{(0)}, \dots, x_n^{(0)}) \\ \frac{\partial F_2}{\partial x_1}(x_1^{(0)}, \dots, x_n^{(0)}) & \frac{\partial F_2}{\partial x_2}(x_1^{(0)}, \dots, x_n^{(0)}) & \cdots & \frac{\partial F_2}{\partial x_n}(x_1^{(0)}, \dots, x_n^{(0)}) \\ \dots & \dots & \cdots & \dots \\ \frac{\partial F_n}{\partial x_1}(x_1^{(0)}, \dots, x_n^{(0)}) & \frac{\partial F_n}{\partial x_2}(x_1^{(0)}, \dots, x_n^{(0)}) & \cdots & \frac{\partial F_n}{\partial x_n}(x_1^{(0)}, \dots, x_n^{(0)}) \end{bmatrix} \begin{bmatrix} \Delta x_1^{(0)} \\ \Delta x_2^{(0)} \\ \dots \\ \Delta x_n^{(0)} \end{bmatrix} = \begin{bmatrix} -F_1(x_1^{(0)}, \dots, x_n^{(0)}) \\ -F_2(x_1^{(0)}, \dots, x_n^{(0)}) \\ \dots \\ -F_n(x_1^{(0)}, \dots, x_n^{(0)}) \end{bmatrix}$$

↑  
Матрица-Якобиан

Определитель матрицы-Якобиан должен быть  
отличен от нуля



По найденным приращениям  $\Delta x_i^{(0)}$  и координатам нулевого приближения находятся значения первых приближений переменных:

$$\begin{cases} x_1^{(1)} = x_1^{(0)} + \Delta x_1^{(0)}, \\ x_2^{(1)} = x_2^{(0)} + \Delta x_2^{(0)}, \\ \dots \\ x_n^{(1)} = x_n^{(0)} + \Delta x_n^{(0)}. \end{cases}$$

которые используются для вычислений следующих приращений  $\Delta x_i^{(1)}$

$$\begin{bmatrix} \frac{\partial F_1}{\partial x_1}(x_1^{(1)}, \dots, x_n^{(1)}) & \frac{\partial F_1}{\partial x_2}(x_1^{(1)}, \dots, x_n^{(1)}) & \dots & \frac{\partial F_1}{\partial x_n}(x_1^{(1)}, \dots, x_n^{(1)}) \\ \frac{\partial F_2}{\partial x_1}(x_1^{(1)}, \dots, x_n^{(1)}) & \frac{\partial F_2}{\partial x_2}(x_1^{(1)}, \dots, x_n^{(1)}) & \dots & \frac{\partial F_2}{\partial x_n}(x_1^{(1)}, \dots, x_n^{(1)}) \\ \dots & \dots & \dots & \dots \\ \frac{\partial F_n}{\partial x_1}(x_1^{(1)}, \dots, x_n^{(1)}) & \frac{\partial F_n}{\partial x_2}(x_1^{(1)}, \dots, x_n^{(1)}) & \dots & \frac{\partial F_n}{\partial x_n}(x_1^{(1)}, \dots, x_n^{(1)}) \end{bmatrix} \begin{bmatrix} \Delta x_1^{(1)} \\ \Delta x_2^{(1)} \\ \dots \\ \Delta x_n^{(1)} \end{bmatrix} = \begin{bmatrix} -F_1(x_1^{(1)}, \dots, x_n^{(1)}) \\ -F_2(x_1^{(1)}, \dots, x_n^{(1)}) \\ \dots \\ -F_n(x_1^{(1)}, \dots, x_n^{(1)}) \end{bmatrix}$$

И Т.Д.

Счет прекращается, если выполняется условие:

$$\max_i |\Delta x_i| < \varepsilon$$



**Пример.** Используя метод Ньютона, решить следующую систему нелинейных уравнений с точностью до 0,001.

$$\begin{cases} \sin(x - 0,5) - y = 1,5 \\ 2x - \cos y = 0,6 \end{cases}$$

Перепишем данную систему в виде:

$$\begin{cases} F_1(x, y) = \sin(x - 0,5) - y - 1,5 \\ F_2(x, y) = 2x - \cos y - 0,6 \end{cases}$$

Сначала необходимо вычислить частные производные функций  $F_1(x, y)$  и  $F_2(x, y)$  по всем переменным:

$$\frac{\partial F_1}{\partial x} = \cos(x - 0,5), \quad \frac{\partial F_1}{\partial y} = -1, \quad \frac{\partial F_2}{\partial x} = 2, \quad \frac{\partial F_2}{\partial y} = -\sin y.$$

В качестве начальных приближений корней зададим следующие:  $x^{(0)} = 0,13$ ;  $y^{(0)} = -1,8$ .



Тогда следующие приближения будут вычисляться схеме:

$$\begin{cases} x^{(k)} = x^{(k-1)} + \Delta x^{(k-1)}, \\ y^{(k)} = y^{(k-1)} + \Delta y^{(k-1)}, \end{cases} \quad \text{где } k = 0, 1, \dots$$

А приращения будут находиться из систем вида:

$$\frac{\partial F_1}{\partial x} = \cos(x - 0,5), \frac{\partial F_1}{\partial y} = -1, \frac{\partial F_2}{\partial x} = 2, \frac{\partial F_2}{\partial y} = -\sin y.$$

$$\begin{bmatrix} \cos(x^{(k)} - 0,5) & -1 \\ 2 & -\sin y^{(k)} \end{bmatrix} \begin{bmatrix} \Delta x^{(k)} \\ \Delta y^{(k)} \end{bmatrix} = \begin{bmatrix} -(\sin(x^{(k)} - 0,5) - y^{(k)} - 1,5) \\ -(2x^{(k)} - \cos y^{(k)} - 0,6) \end{bmatrix}$$

где  $k$  – номер приближения.



Номер итерации, <i>k</i>	Приближение корня		$\Delta x^{(k)}$	$\Delta y^{(k)}$
	$x^{(k)}$	$y^{(k)}$		
0	0,13	-1,8	0,059424	-0,00621
1	0,189424	-1,80621	-0,00436	-0,00346
2	0,18506	-1,80968	-0,00229	-0,00222
3	0,18277	-1,81189	-0,00147	-0,00141
4	0,181301	-1,81331	-0,00093	-0,0009
5	0,180366	-1,8142		

Итерации были прекращены, когда выполнилось условие

$$\max\{\Delta x^{(k)}, \Delta y^{(k)}\} < 0,001$$



Пусть требуется решить систему нелинейных уравнений вида:

$$\begin{cases} F_1(x_1, x_2, \dots, x_n) = 0, \\ F_2(x_1, x_2, \dots, x_n) = 0, \\ \dots \dots \dots \dots \dots \\ F_n(x_1, x_2, \dots, x_n) = 0. \end{cases}$$

При решении СНУ с помощью модифицированного метода Ньютона также состоится итерационная последовательность, получаемая из схемы:

$$\begin{cases} x_1^{(k)} = x_1^{(k-1)} + \Delta x_1^{(k-1)}, \\ x_2^{(k)} = x_2^{(k-1)} + \Delta x_2^{(k-1)}, \\ x_3^{(k)} = x_3^{(k-1)} + \Delta x_3^{(k-1)}, \\ \dots \dots \dots \dots \dots \\ x_n^{(k)} = x_n^{(k-1)} + \Delta x_n^{(k-1)}. \end{cases}$$



В отличие от метода Ньютона в модифицированном методе при определении  $\Delta x_i^{(k-1)}$  используется схема, у которой матрица при неизвестных остается постоянной – значения всех частных производных находятся только в точке  $x^{(0)}$ :

$$\begin{bmatrix} \frac{\partial F_1}{\partial x_1}(x_1^{(0)}, \dots, x_n^{(0)}) & \frac{\partial F_1}{\partial x_2}(x_1^{(0)}, \dots, x_n^{(0)}) & \dots & \frac{\partial F_1}{\partial x_n}(x_1^{(0)}, \dots, x_n^{(0)}) \\ \frac{\partial F_2}{\partial x_1}(x_1^{(0)}, \dots, x_n^{(0)}) & \frac{\partial F_2}{\partial x_2}(x_1^{(0)}, \dots, x_n^{(0)}) & \dots & \frac{\partial F_2}{\partial x_n}(x_1^{(0)}, \dots, x_n^{(0)}) \\ \dots & \dots & \dots & \dots \\ \frac{\partial F_n}{\partial x_1}(x_1^{(0)}, \dots, x_n^{(0)}) & \frac{\partial F_n}{\partial x_2}(x_1^{(0)}, \dots, x_n^{(0)}) & \dots & \frac{\partial F_n}{\partial x_n}(x_1^{(0)}, \dots, x_n^{(0)}) \end{bmatrix} \begin{bmatrix} \Delta x_1^{(k-1)} \\ \Delta x_2^{(k-1)} \\ \dots \\ \Delta x_n^{(k-1)} \end{bmatrix} = \begin{bmatrix} -F_1(x_1^{(k-1)}, \dots, x_n^{(k-1)}) \\ -F_2(x_1^{(k-1)}, \dots, x_n^{(k-1)}) \\ \dots \\ -F_n(x_1^{(k-1)}, \dots, x_n^{(k-1)}) \end{bmatrix}$$

Счет прекращается, если выполняется условие:  
 $\max_i |\Delta x_i| < \varepsilon$

Модифицированный метод Ньютона сходится медленнее, чем метод Ньютона.



**Пример.** Используя модифицированный метод Ньютона, решить следующую систему нелинейных уравнений с точностью до 0,001.

$$\begin{cases} \sin(x - 0,5) - y = 1,5 \\ 2x - \cos y = 0,6 \end{cases}$$

Приращения переменных, согласно методу, будем находить по схеме:

$$\begin{bmatrix} \cos(x^{(0)} - 0,5) & -1 \\ 2 & -\sin y^{(0)} \end{bmatrix} \begin{bmatrix} \Delta x^{(k)} \\ \Delta y^{(k)} \end{bmatrix} = \begin{bmatrix} -(\sin(x^{(k)} - 0,5) - y^{(k)} - 1,5) \\ -(2x^{(k)} - \cos y^{(k)} - 0,6) \end{bmatrix}$$



Номер итерации, <i>k</i>	Приближение корня		$\Delta x^{(k)}$	$\Delta y^{(k)}$
	$x^{(k)}$	$y^{(k)}$		
0	0,13	-1,8	0,059424	-0,00621
1	0,189424	-1,80621	-0,00436	-0,00346
2	0,185062	-1,80967	-0,00229	-0,00222
3	0,182773	-1,81189	-0,00147	-0,00141
4	0,181304	-1,8133	-0,00094	-0,0009
5	0,180369	-1,8142		

Итерации были прекращены, когда выполнилось условие

$$\max\{\Delta x^{(k)}, \Delta y^{(k)}\} < 0,001$$

## Домашняя работа #2

Задание посвящено исследованию траектории полета бумажного самолётика при помощи решения системы нелинейных уравнений, описывающей условия достижения максимальной высоты полета. Вам нужно решить эту систему *методом простых итераций с подбором параметра*, оптимизировать код и проанализировать влияние численных форматов на точность и скорость.

Используйте **только** стандартные методы Python (math, numpy и numba).

Срок сдачи работ: **до 7-го марта, 23:59.**

Работы принимаются на электронный адрес [VSChernyshenko@fa.ru](mailto:VSChernyshenko@fa.ru) **исключительно в виде ноутбука в формате html**. Название файла: “Фамилия\_Имя\_группа\_ДЗ2.html”.

**Не принимаются** работы являющиеся копией друг друга, работы **не содержащие подробные комментарии**.

## Домашняя работа #2

Траектория полёта самолётика описывается параметрически:

$$\begin{aligned}x(t) &= v_0 \cdot \cos(\theta) \cdot t \\y(t) &= v_0 \cdot \sin(\theta) \cdot t - \frac{1}{2} g t^2 + k \cdot t\end{aligned}$$

где:

$v_0 = 5$  м/с – начальная скорость.

$\theta$  – угол броска (в радианах).

$t$  – время полёта.

$g = 9.8$  м/с<sup>2</sup> – ускорение свободного падения.

$k = 0.1$  – коэффициент сопротивления воздуха.

Для достижения максимальной высоты,  $\theta$  и  $t$  должны удовлетворять условию:

$$\theta = \arctan \left( \frac{y(t)}{x(t)} \right).$$

Найдите пару  $(\theta, t)$ , при которых самолётик **достигает максимальной высоты**.

## Домашняя работа #2

### Обязательные подзадачи:

- Определите графически примерные области, где могут находиться корни системы.
- Введите параметр  $b$  для ускорения сходимости. Подберите значение  $b$  (например, в диапазоне 0.1–0.9), чтобы итерации сходились.
- Реализуйте метод простых итераций для решения системы.
- Постройте форму найденной траектории и точку максимальной высоты.
- Проведите вычисления в трёх форматах: np.float16, np.float32 и np.float64 для заданных допустимых погрешностей [0.01, 0.001, 0.0001].
- Определите узкие места в коде и предложите их оптимизацию с помощью numba.
- Для каждого типа данных и каждого уровня погрешности вычислите абсолютную ошибку.
- Выведите таблицу с результатами:
  - значение  $(x, y, t)$  для каждого типа данных и уровня погрешности;
  - число итераций;
  - время выполнения;
  - абсолютная ошибка.
- Напишите в комментариях подробный отчёт о полученных результатах.



# ИНТЕРПОЛЯЦИЯ, ЭКСТРАПОЛЯЦИЯ, АППРОКСИМАЦИЯ



Основу математических моделей многих процессов и явлений в физике, химии, биологии и др. областях составляют уравнения различного вида. Для решения этих уравнений необходимо иметь возможность вычислить значения функций, входящих в описание математической модели рассматриваемого процесса при произвольном значении аргумента.

Используемые в математических моделях функции могут быть заданы как аналитическим способом, так и табличным, при котором функция известна только при дискретных значениях аргумента.



Пусть функция  $f(x)$  задана множеством своих значений для дискретного набора точек (таблицей). Эта таблица может быть результатом расчетов, либо экспериментальными точками.

$X$	$x_0$	$x_1$	$x_2$	...	$x_n$
$f(x)$	$y_0$	$y_1$	$y_2$	...	$y_n$

Значения аргумента  $x_i$  называются [узлами](#). (В общем случае эти узлы не являются равноотстоящими).

Требуется найти приближенные значения функции  $f(x)$  в любой произвольной точке отрезка  $[x_0; x_n]$  при помощи функции  $F(x)$ .

$$F(x) \approx f(x) \quad x \in [x_0; x_n]$$

Приближение (замена) функции  $f(x)$  заданной таблично другой функцией  $F(x)$ , заданной аналитически, называется [аппроксимацией](#).



$X$	$X_0$	$X_1$	$X_2$	...	$X_n$
$f(x)$	$y_0$	$y_1$	$y_2$	...	$y_n$

Для интерполяирования функций с большим числом узлов, применяют локальную интерполяцию. Для этого заданную сетку делят на несколько интервалов с небольшим числом узлов и на каждом из интервалов строят свой интерполяционный полином, поэтому локальную интерполяцию еще называют кусочно-полиномиальной.



Цель: Получение функциональной зависимости  $y = f(x)$ .

Подход:

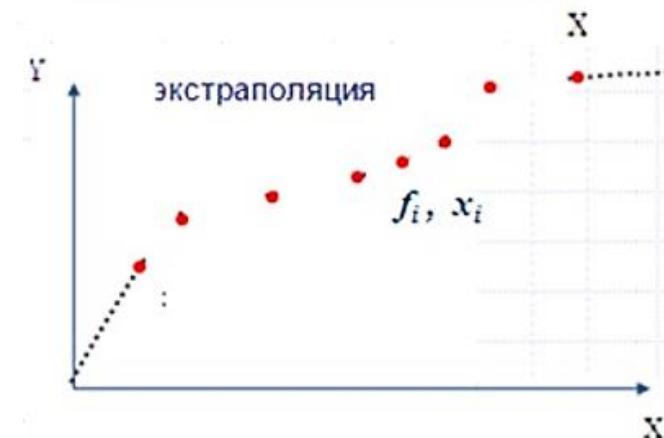
1. Интерполяция – функция должна пройти через все точки. Способ нахождения промежуточных значений величины по имеющемуся дискретному набору известных значений.
2. Экстраполяция – особый тип аппроксимации, при котором функция аппроксимируется вне заданного интервала, а не между заданными значениями.
3. Аппроксимация (например, регрессия/метод наименьших квадратов) – аппроксимирующая функция не обязательно должна проходить через все точки. Происходит замена одних объектов другими, в каком-то смысле близкими к исходным, но более простыми.



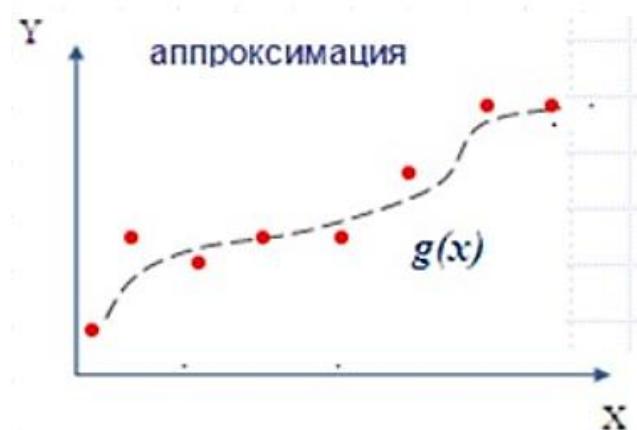
1. Интерполяция и экстраполяция.

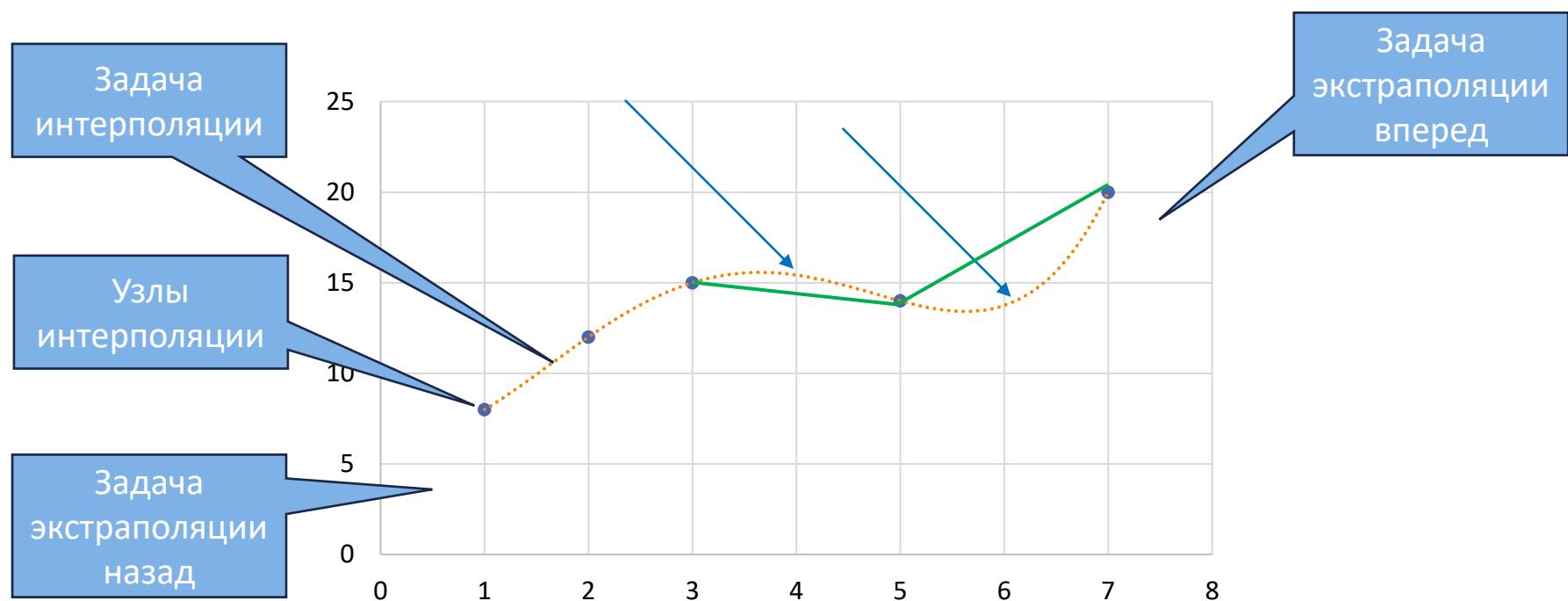


2. Экстраполяция.

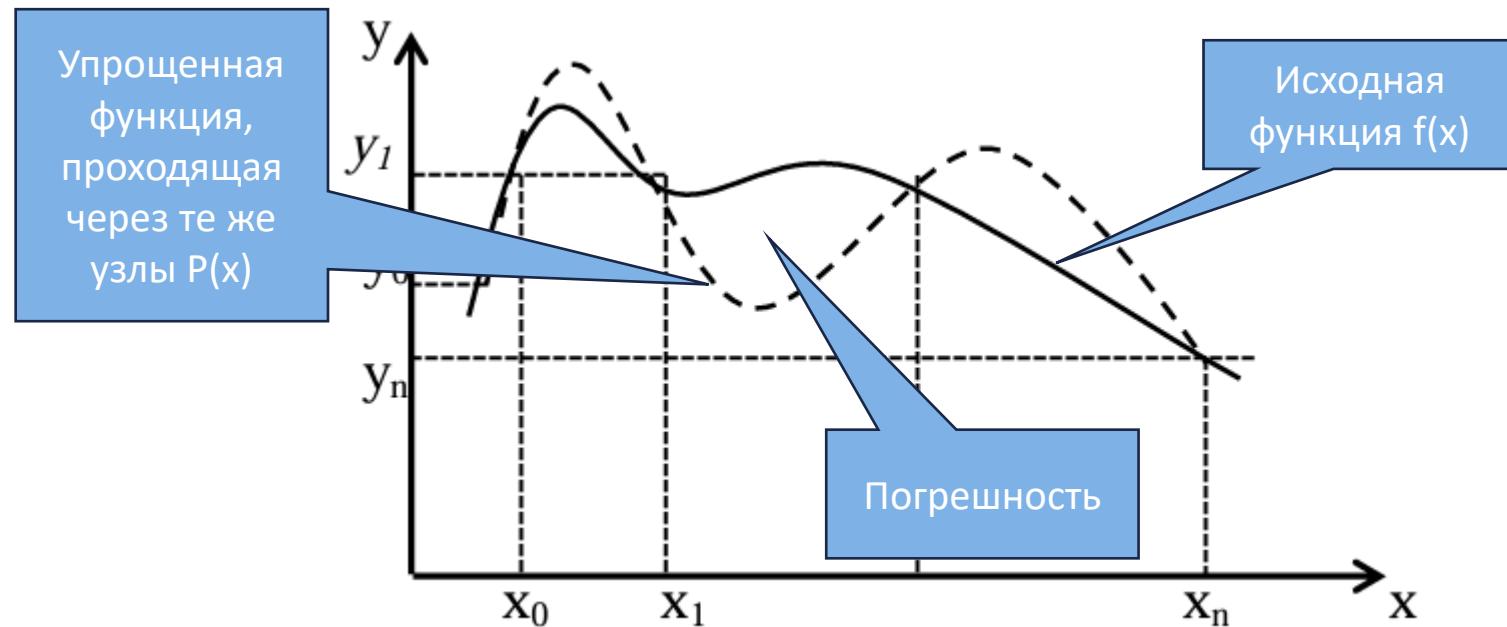


3. Аппроксимация.





*Способ восстановления пропущенных данных при сложном виде аналитической функции*





Аппроксимация – замена одних мат. объектов другими, в том или ином смысле близкими к исходным.

**Аппроксимацией** (приближением) функции  $f(x)$  называется нахождение такой функции  $F(x)$  (**аппроксимирующей функции**), которая была бы близка к заданной. Критерии близости функций  $f(x)$  и  $F(x)$  могут быть различные.

В том случае, когда приближение строится на дискретном наборе точек, аппроксимацию называют *точечной* или *дискретной*.

В том случае, когда аппроксимация проводится на непрерывном множестве точек (отрезке), аппроксимация называется *непрерывной* или *интегральной*. Примером такой аппроксимации может служить разложение функции в ряд Тейлора, то есть замена некоторой функции степенным многочленом.



Чаще всего используется **точечная** аппроксимация. При этом функция  $f(x)$  как правило неизвестна, а связь между параметрами  $x$  и  $y$  задаётся в виде некоторой таблицы  $\{x_i, y_i\}$ .

Это означает, что дискретному множеству значений аргумента  $\{x_i\}$  поставлено в соответствие множество значений функции  $\{y_i\}$  ( $i = 0, 1 \dots n$ ).

Эти значения – либо результаты расчетов, либо экспериментальные данные. На практике же могут понадобиться значения величины  $y$  и в других точках, отличных от узлов  $x_i$ . Однако получить эти значения можно лишь путем очень сложных расчетов или проведением дорогостоящих экспериментов.

В подобных случаях, оптимальным, с точки зрения экономии времени и средств, является использование имеющихся табличных данных для приближенного вычисления искомого параметра  $y$  при любом значении определяющего параметра  $x$  (в рамках некоторого интервала), поскольку точная связь  $y = f(x)$  неизвестна или использование её в расчётах затруднительно.

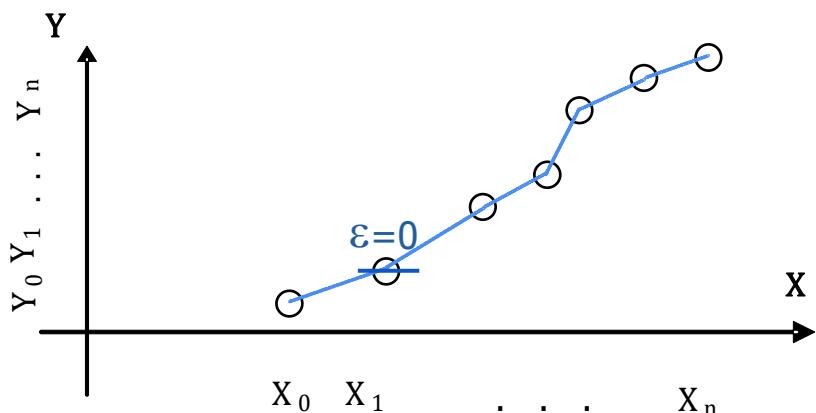
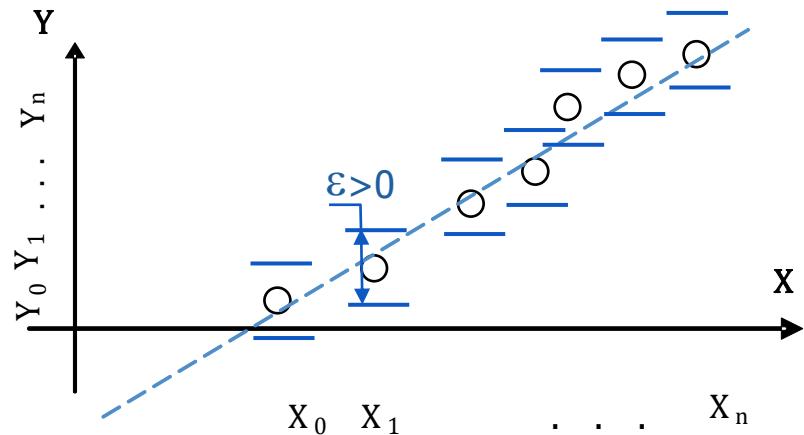


Общая погрешность аппроксимирующей функции может быть выражена как сумма локальных погрешностей в точках с координатами  $x_i$ .

$$E = \sum e_i,$$

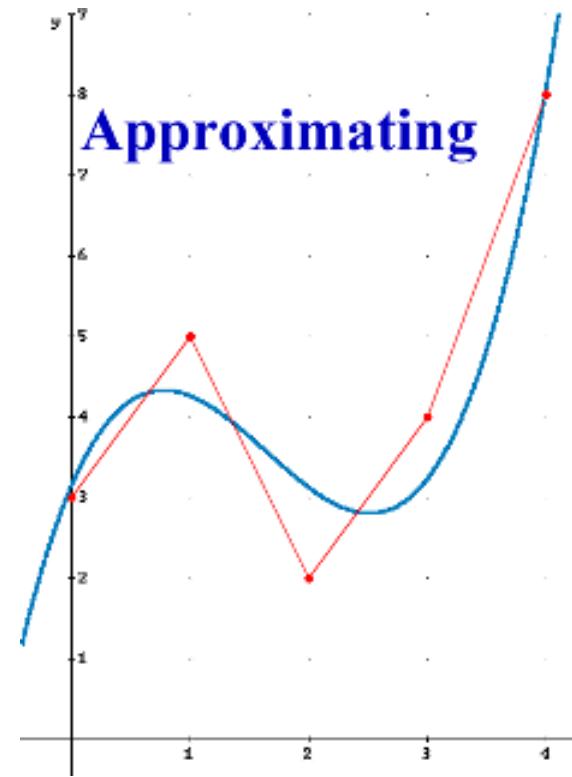
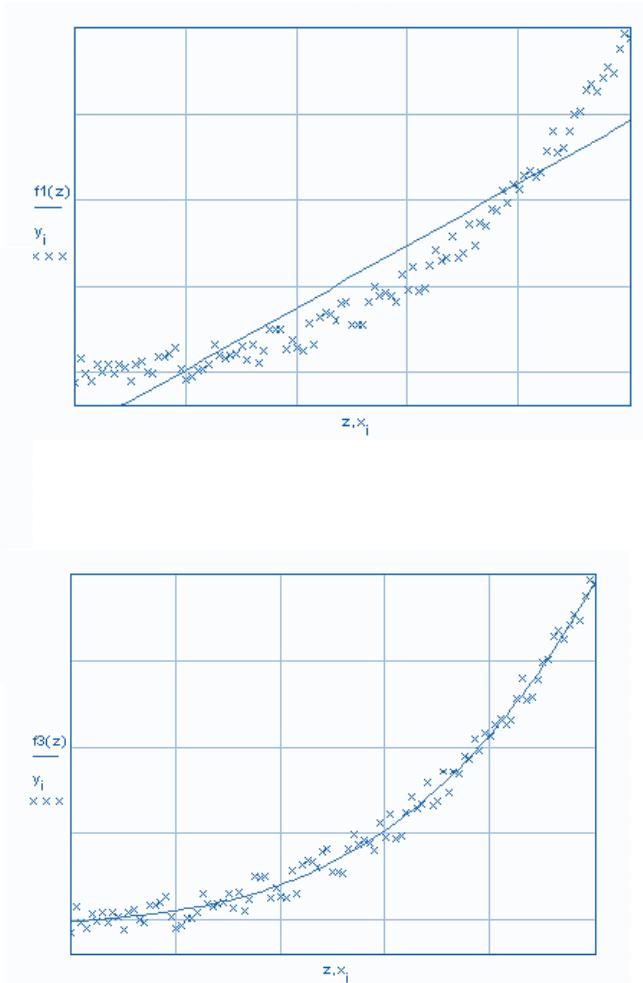
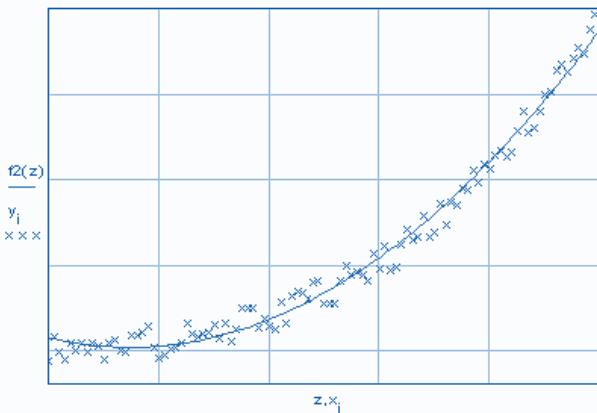
где  $e_i = |F(x_i) - f(x_i)|$ .

В общем случае при аппроксимации  $0 \leq e_i \leq \varepsilon$ . В случае, если  $\varepsilon = 0$ , т.е. налагается условие строгого совпадения значений функций  $F(x)$  и  $f(x)$  в заданных точках  $x_i$ , то данный вид аппроксимации называется интерполяцией.





Чаще всего аппроксимацию применяют для экспериментального нахождения некоторых зависимостей.





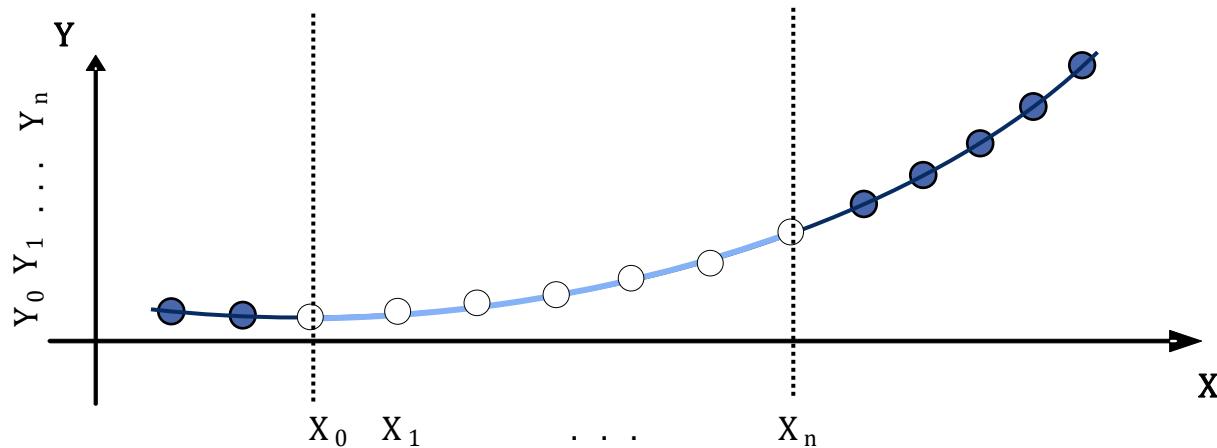
Чем проще аппроксимирующая функция, тем меньше времени требуется для решения задачи аппроксимации. Чем больше узлов, тем меньше погрешность. Для каждой конкретной аппроксимирующей функции нужно стремиться выбрать такой способ аппроксимации, который обеспечивает минимальную погрешность при минимальном количестве узлов.



При интерполяции  $F(x_i) = f(x_i)$ , что автоматически подразумевает наличие известных  $\{x_i, y_i\}$ , для некоторого определённого интервала  $[x_0, x_n]$ .

В случае, если требуется получить аппроксимацию функции за пределами известного интервала, то данный вид аппроксимации называется экстраполяцией.

$x_i$ , для которых даны  $y_i$ , называются узлами интерполяции или опорными точками.





Погрешность интерполяции определяется расстоянием между узлами интерполяции. Обусловлена погрешность тем, что график имеет изломы в узлах.

Изломы интерполяции можно устраниТЬ, если в качестве интерполирующей использовать такую функцию, график которой представляет собой плавную кривую, например, полином, проходящий через заданные в таблице точки.



1. выбор наиболее удобного способа построения аппроксимирующей функции для каждого конкретного случая;
2. оценка погрешности при замене  $f(x)$  аппроксимирующей функцией  $P(x)$  на отрезке  $[a, b]$ , поскольку эти функции совпадают только в узлах интерполяции;
3. оптимальный выбор узлов интерполирования для получения минимальной погрешности.

Как получают исходные данные: задаем интервал, шаг, количество узлов, вычисляем значения с помощью функции  $f(x)$ .

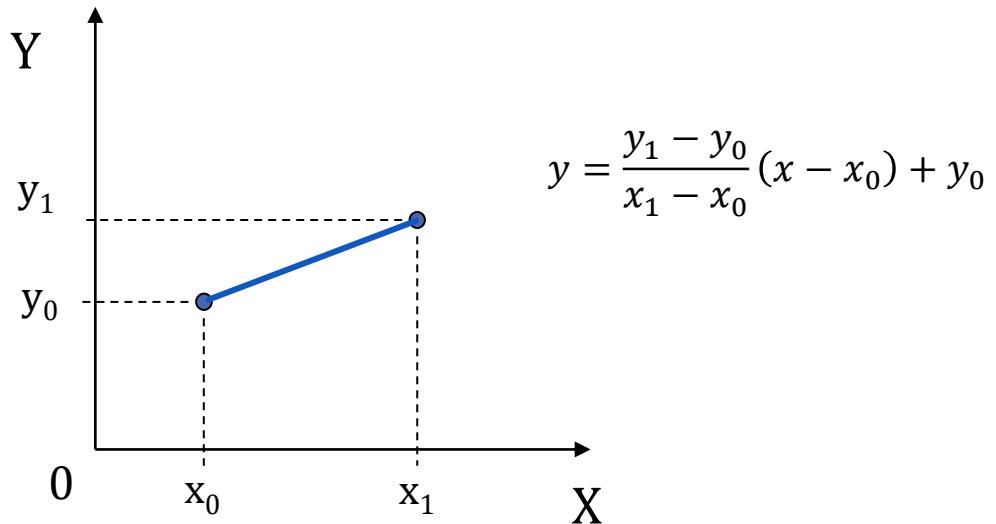
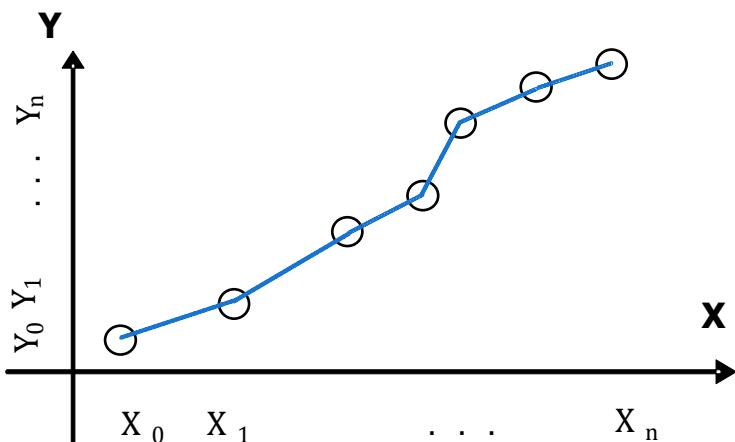
Определять большое количество значений вычислительно дорого.



Интерполяция бывает глобальной –  $F(x)$  проходит через ВСЕ точки заданного интервала  $[x_0, x_n]$ , либо локальной (кусочной) –  $f(x)$  на указанном интервале интерполируется несколькими  $F_1(x), F_2(x) \dots F_k(x)$ .

### Типы интерполяции:

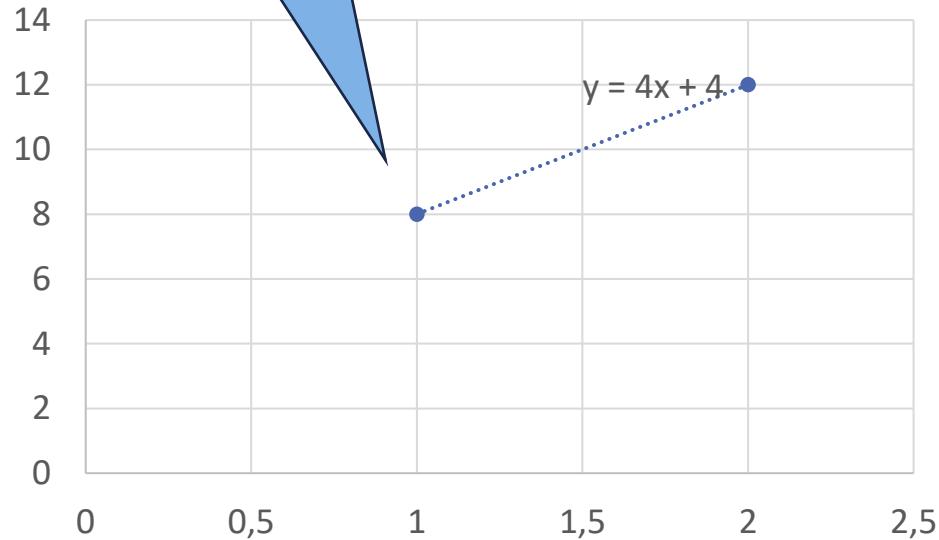
- полиномиальная,
- тригонометрическая,
- экспоненциальная.



Линейная интерполяция – простейший вид локальной полиномиальной интерполяции – замена  $f(x)$  множеством линейных функций  $F_1(x), F_2(x) \dots F_k(x)$ , каждая из которых соединяет лишь две точки.

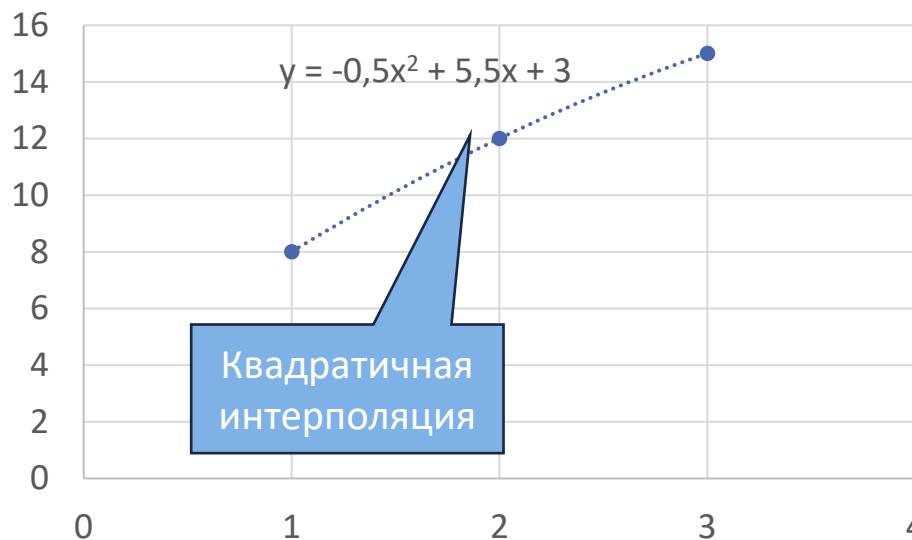
1. тригонометрические,
2. рациональные,
3. иррациональные,
4. степенные,
5. показательные,
6. полиномы

Линейная  
интерполяция



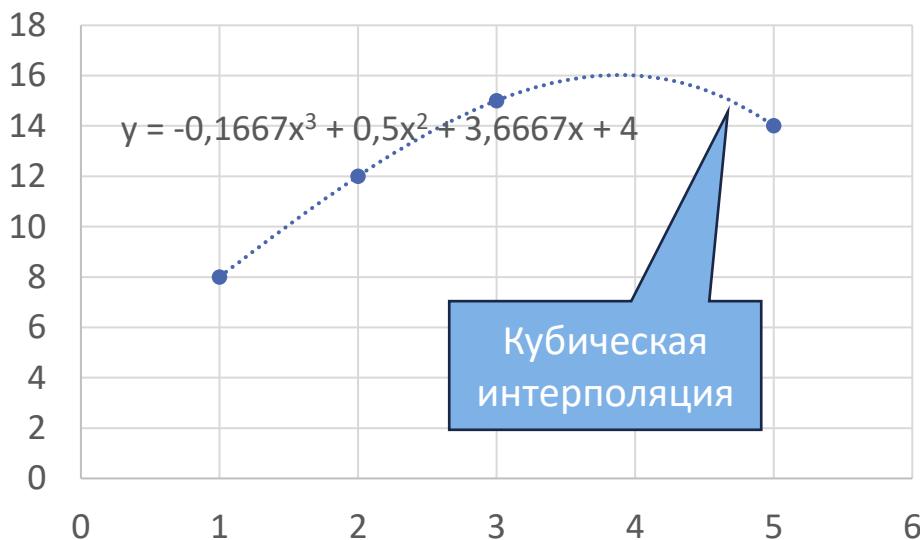
$$y = -0,5x^2 + 5,5x + 3$$

Квадратичная  
интерполяция



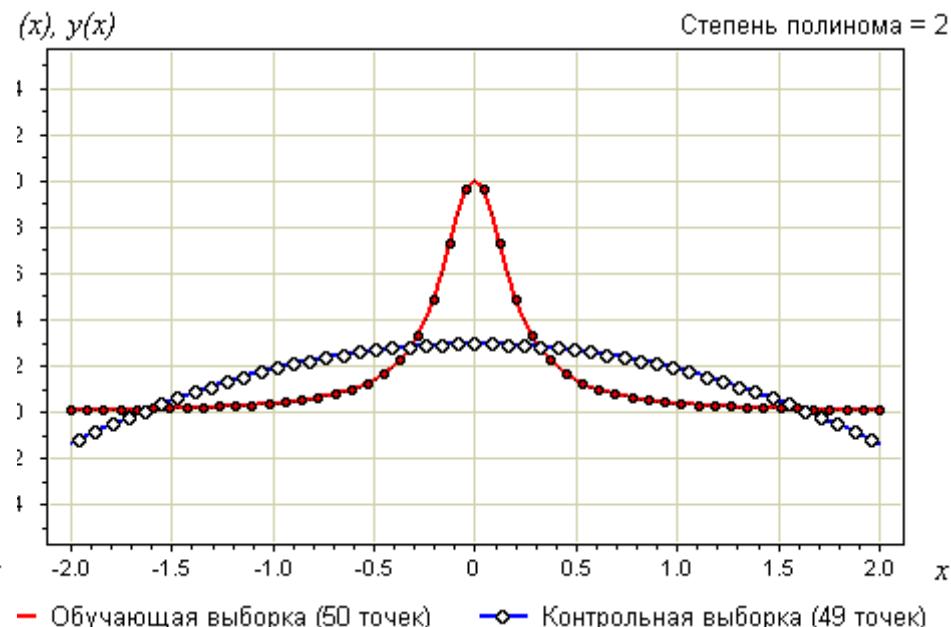
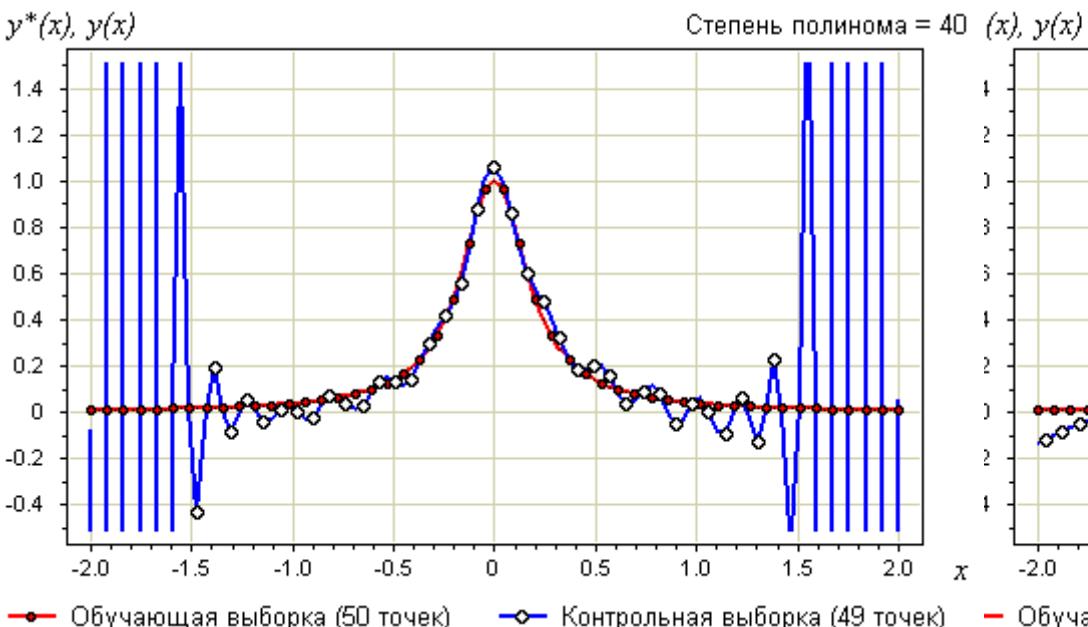
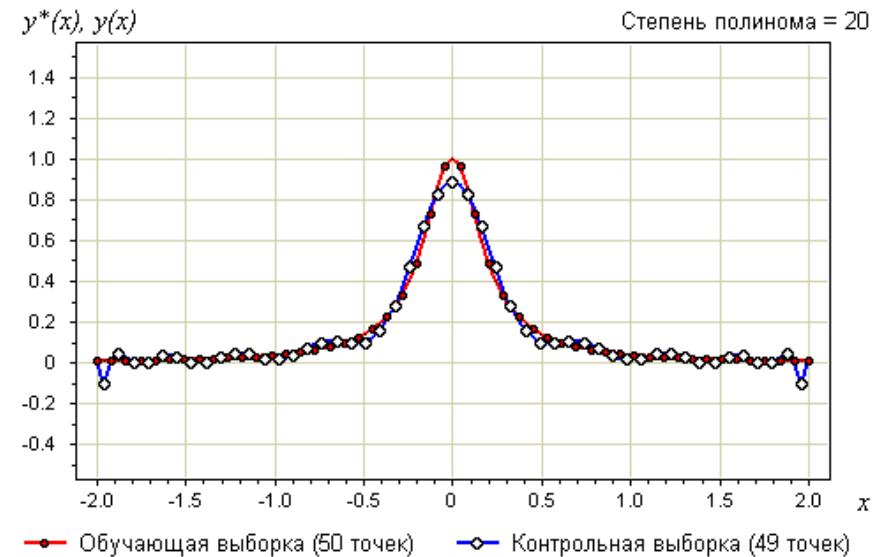
$$y = -0,1667x^3 + 0,5x^2 + 3,6667x + 4$$

Кубическая  
интерполяция



Стоит избегать:

- нежелательных явлений, связанных с применением избыточно сложных уравнений.
- нежелательных явлений, связанных с применением недостаточно сложных моделей.





Простейшим способом интерполяции функции  $f$  по таблице является ступенчатая интерполяция. Один из ее вариантов формулируется так:

$$\tilde{f}(x) = f(x_i), i : \forall j \neq i, |x - x_j| > |x - x_i|$$

То есть за значение функции  $\tilde{f}(x)$  берется значение функции  $f$  в точке, ближайшей к рассматриваемой. Более точным способом интерполяции является кусочно-линейная интерполяция. При таком подходе значение  $f(x)$  интерполируется по двум соседним с точкой  $x$  точкам.

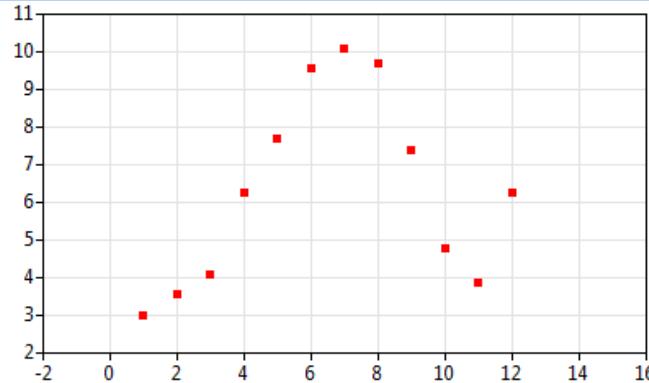
$$\tilde{f}(x) = \frac{f(x_i)(x_{i+1} - x) + f(x_{i+1})(x - x_i)}{x_{i+1} - x_i}, i : x_i < x < x_{i+1}$$

(здесь подразумевается монотонное возрастание последовательности  $x_i$ )

Интересно понять, с какой точностью интерполяционные формулы аппроксимируют функцию  $f$ . Предположим, что производная функции  $f$  ограничена величиной  $g$ . Тогда на отрезке  $[x_i, x_{i+1}]$  функция  $f$  не может отклониться от линейной интерполяции более, чем на  $h \left( g - \left| \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} \right| \right)$ .

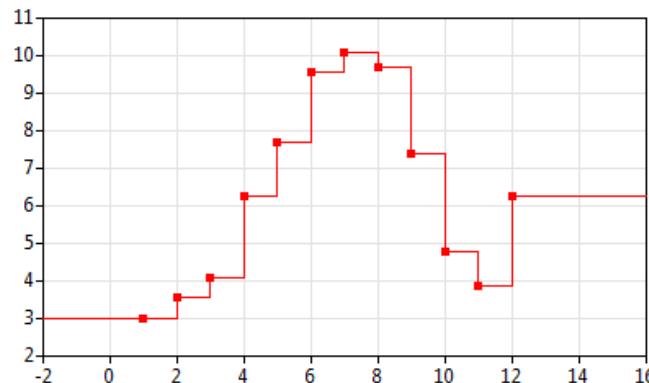


## Тип интерполяции

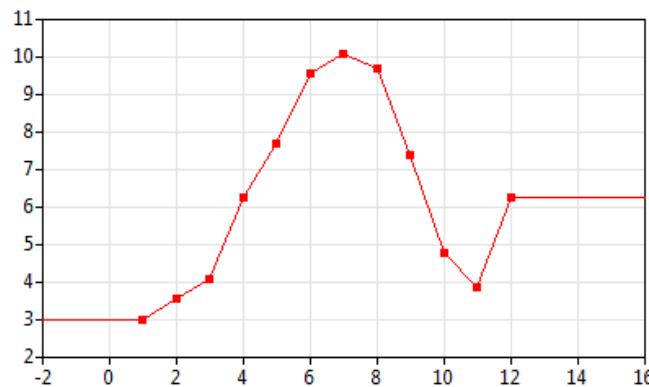


## Описание и пример

Интерполяция не используется.



Ступенчатая интерполяция. Значение функции на интервале между двумя точками равно значению функции в точке с меньшим аргументом.



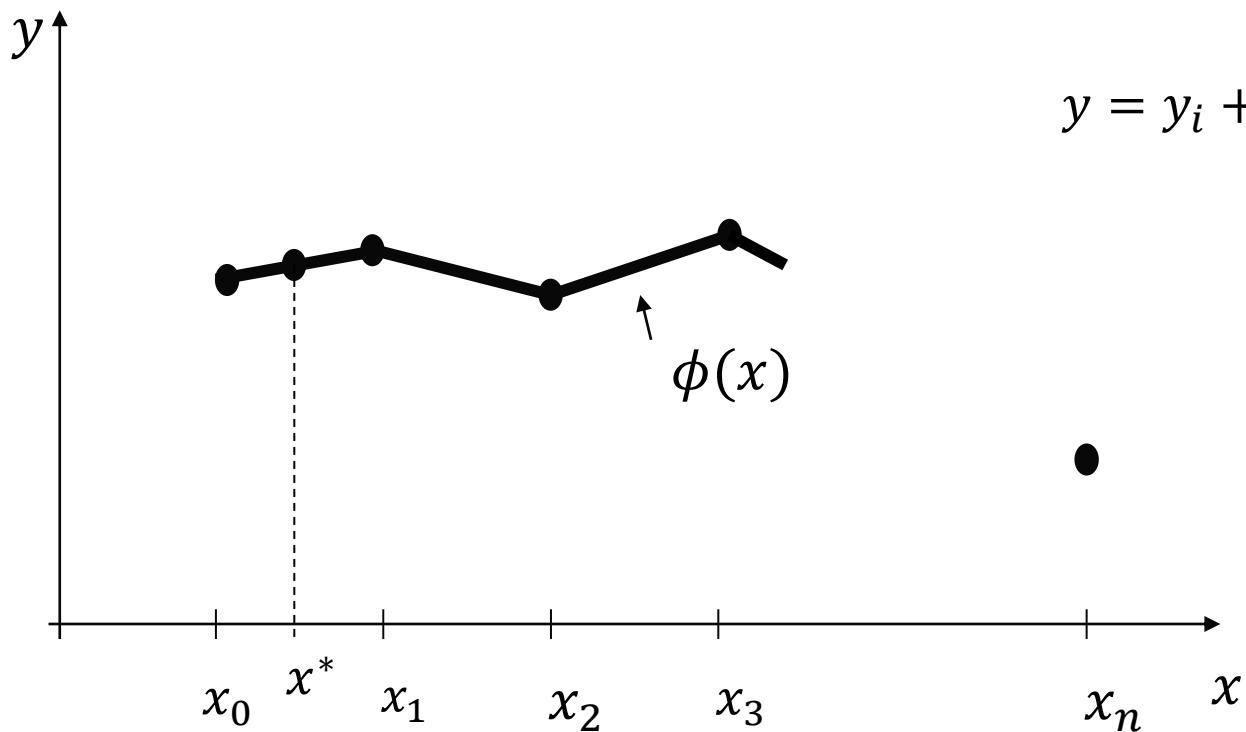
Линейная интерполяция. Точки соединяются прямыми линиями.

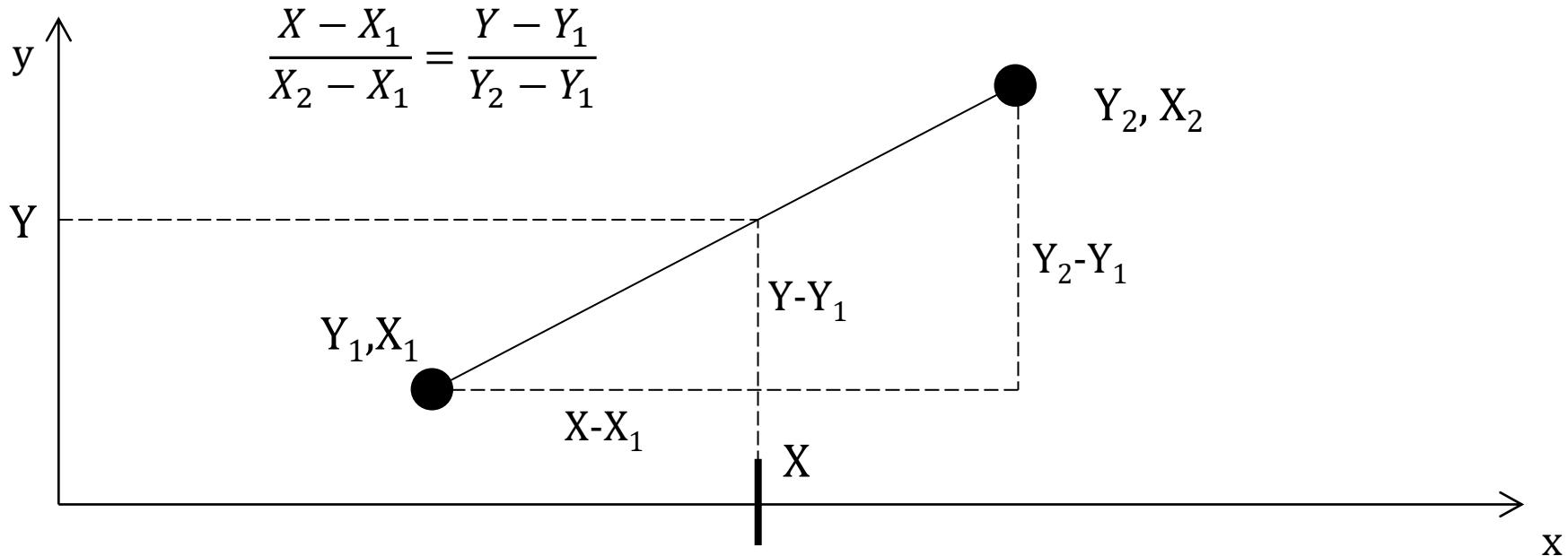


Функция  $y(x)$  аппроксимируется на каждом частичном отрезке прямой.

$$\frac{x - x_i}{x_{i+1} - x_i} = \frac{y - y_i}{y_{i+1} - y_i}$$

$$y = y_i + \frac{x - x_i}{x_{i+1} - x_i} (y_{i+1} - y_i)$$





Даны две точки:  $X_1, Y_1$  и  $X_2, Y_2$ . Найти  $Y$  для *заданной* точки  $X$ .

$$Y = Y_1 + (X - X_1)(Y_2 - Y_1)/(X_2 - X_1)$$

Пример.  $X_1 = 4, Y_1 = 10$ .  $X_2 = 8, Y_2 = 15$ . Найти  $Y$  для  $X = 5$ .

$$Y = 10 + (5 - 4)(15 - 10)/(8 - 4) = 11.25$$



*Алгебраическим интерполяционным многочленом*  $P_n(x, f, x_0, \dots, x_n)$  называется многочлен  $P_n(x) = c_0 + c_1x + c_2x^2 + \dots + c_nx^n$  степени не выше  $n$ , принимающий в точках  $x_0, x_1, \dots, x_n$  значения  $f(x_0), f(x_1), \dots, f(x_n)$ .

**Теорема.** Если заданы попарно различные узлы  $x_0, x_1, x_2, \dots, x_n$  и значения  $f(x_0), f(x_1), \dots, f(x_n)$ , то алгебраический интерполяционный многочлен существует и единственен.

**Доказательство** Сначала докажем, что существует не более чем один интерполяционный многочлен, а затем построим его. Если бы их было два, то их разность – многочлен степени не больше  $n$ , обращалась бы в 0 в  $n + 1$  точке –  $x_0, x_1, \dots, x_n$ , что невозможно для ненулевого многочлена.

В качестве примера интерполяционного многочлена можно привести [Интерполяционный многочлен Лагранжа](#).



**Интерполяционный многочлен Лагранжа** – многочлен минимальной степени, принимающий данные значения в данном наборе точек.

Для  $n + 1$  пар чисел  $(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$ , где все  $x_i$  различны, существует ТОЛЬКО ОДИН интерполяционный многочлен  $L(x)$  степени не более  $n$ , для которого  $L(x_i) = y_i$ :

$$F(x) = \sum_{i=0}^n y_i L_i(x),$$

В простейшем случае ( $n = 1$ ) – это линейный многочлен, график которого – прямая, проходящая через две заданные точки.

В качестве глобальной интерполяционной функции  $F(x)$  можно найти многочлен степени не больше  $n$ , такой, что:

$$F(x_i) = y_i.$$

Форма Лагранжа:

$$L_i(x) = \frac{(x - x_0)\dots(x - x_{i-1})(x - x_{i+1})\dots(x - x_n)}{(x_i - x_0)\dots(x_i - x_{i-1})(x_i - x_{i+1})\dots(x_i - x_n)}$$

В результате будет получен полином  $L(x) = A_0 + A_1x + A_2x^2 + \dots + A_nx^n$ , значения которого в точках  $x_i$  будут совпадать с  $y_i$ .



В случае равномерного распределения узлов интерполяции  $x_i$  выражаются через расстояние между узлами интерполяции  $h$  и начальную точку  $x_0$ :

$$x_j \equiv x_0 + jh$$

и, следовательно,

$$x_i - x_j \equiv (i - j)h$$

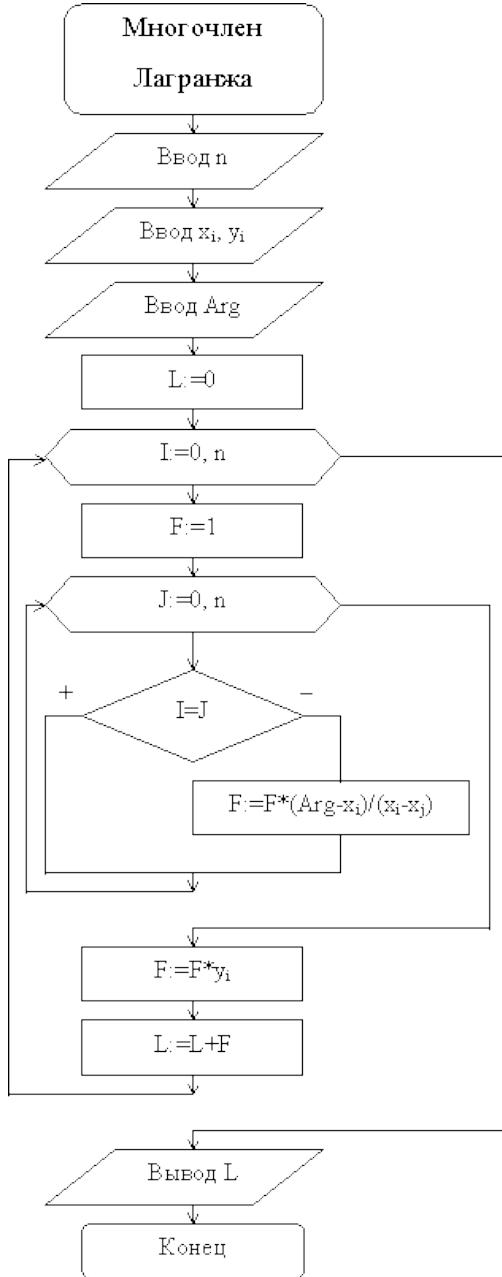
Подставив эти выражения в формулу базисного полинома и вынеся  $h$  за знаки перемножения в числителе и знаменателе, получим

$$l_i(x) = \prod_{j=0, j \neq i}^n \frac{(x - x_j)}{(x_i - x_j)} = \frac{\prod_{j=0, j \neq i}^n (x - x_0 - jh)}{h^{n-1} \prod_{j=0, j \neq i}^n (i - j)}$$

Теперь можно ввести замену переменной

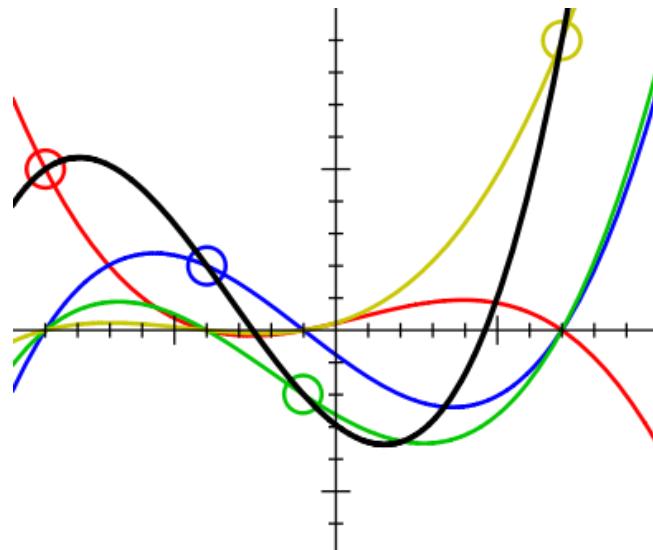
$$y = \frac{x - x_0}{h}$$

и получить полином от  $y$ , который строится с использованием только целочисленной арифметики. Недостатком данного подхода является факториальная сложность числителя и знаменателя, что требует использования длинной арифметики.



$$l_j(x) = \prod_{i=0, i \neq j}^n \frac{x - x_i}{x_j - x_i} = \frac{x - x_0}{x_j - x_0} \dots \frac{x - x_{j-1}}{x_j - x_{j-1}} \frac{x - x_{j+1}}{x_j - x_{j+1}} \dots \frac{x - x_n}{x_j - x_n}$$

$$L(x) = \sum_{j=0}^n y_j l_j(x)$$



Этот пример показывает интерполяционный многочлен Лагранжа для четырёх точек  $(-9, 5), (-4, 2), (-1, -2)$  и  $(7, 9)$ , а также полиномы  $y_j l_j(x)$ , каждый из которых проходит через одну из выделенных точек, и принимает нулевое значение в остальных  $x_i$ .



Пусть интерполяционный многочлен Лагранжа построен для известной функции  $f(x)$ . Необходимо выяснить, насколько этот многочлен близок к функции в точках отрезка  $[a, b]$ , отличных от узлов.

Погрешность интерполяции равна  $|f(x) - L_n(x)|$ . Оценку погрешности можно получить на основании следующей теоремы.

### Теорема

Пусть функция  $f(x)$  дифференцируема  $n + 1$  раз на отрезке  $[a, b]$ , содержащем узлы интерполяции  $x_i$  из  $[a, b]$ ,  $i = 0, 1, \dots, n$ . Тогда для погрешности интерполяции в точке  $x$  из  $[a, b]$  справедлива оценка:

$$|f(x) - L_n(x)| \leq M_{n+1} |\omega_{n+1}(x)| / (n + 1)!$$

$$M_{n+1} = \max_{[a,b]} |f^{n+1}(x)|,$$

$$\omega_{n+1}(x) = (x - x_0)(x - x_1) \dots (x - x_n).$$

Для максимальной погрешности интерполяции на всём отрезке  $[a, b]$  справедлива оценка:

$$\max_{[a,b]} |f(x) - L_n(x)| \leq \frac{M_{n+1}}{(n + 1)!} \max_{[a,b]} |\omega_n(x)|$$

Оценим погрешность приближения функции  $f(x) = \sqrt{x}$  в точке  $x = 116$  и на всем отрезке  $[a, b]$ , где  $a = 100$ ,  $b = 144$ , с помощью интерполяционного многочлена Лагранжа  $L_2(x)$  второй степени, построенного с узлами  $x_0 = 100$ ,  $x_2 = 144$ .

Найдем первую, вторую и третью производные функции  $f(x)$ :

$$f'(x) = \frac{1}{2}x^{-\frac{1}{2}}, \quad f''(x) = -\frac{1}{4}x^{-\frac{3}{2}}, \quad f'''(x) = \frac{3}{8}x^{-5/2}.$$
$$M_3 = \max_{[a,b]} |f'''(x)| = \frac{3}{8}100^{-5/2} = \frac{3}{8}10^{-5}.$$

получим оценку погрешности в точке  $x = 116$ :

$$|\sqrt{116} - L_2(116)| \leq \frac{1}{3!}|(116 - 100)(116 - 121)(116 - 144)| = \frac{1}{16}10^{-5} \cdot 16 \cdot 5 \cdot 28 = 1.4 \cdot 10^{-3}.$$

Оценим погрешность приближения функции  $f(x) = \sqrt{x}$  на всем отрезке:

$$\max_{[a,b]} |\sqrt{x} - L_2(x)| \leq \frac{10^{-5}}{16} \max_{[a,b]} |(x - 100)(x - 121)(x - 144)| \approx 2 \cdot 5 \cdot 10^{-3}.$$



Непараметрическое задание кривой:

$$x = x, y = f(x), z = g(x)$$

В параметрическом виде каждая координата точки кривой представлена как функция одного параметра  $t$ . Значение параметра задает координатный вектор точки на кривой ( $0 \leq t \leq 1$ ).

Кривая задаётся системой уравнений:

$$\begin{cases} x(t) = a_x t^3 + b_x t^2 + c_x t + d_x \\ y(t) = a_y t^3 + b_y t^2 + c_y t + d_y \\ z(t) = a_z t^3 + b_z t^2 + c_z t + d_z \end{cases}$$

Производная любого уравнения из системы имеет вид:

$$dx/dt = 3a_x t^2 + 2b_x t + c_x$$

Векторное представление точки на кривой:

$$P(t) = [x(t) \ y(t) \ z(t)].$$

Чтобы получить непараметрическую форму, нужно исключить  $t$  из двух уравнений и вывести одно в терминах  $x$  и  $y$ .

Параметрическая форма позволяет представить замкнутые и многозначные кривые.

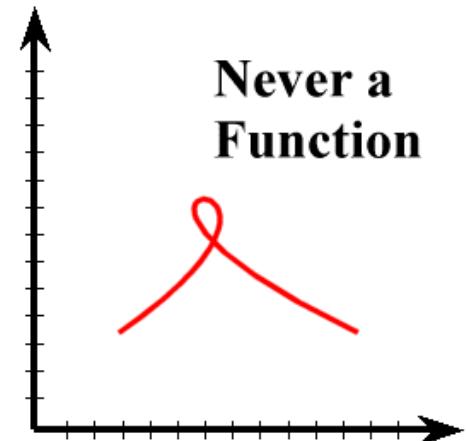


Использование параметрической записи кривых более удобно.

В общем случае во многих задачах кривые не могут быть записаны в виде уравнения  $y = f(x)$  с использованием обычных однозначных функций.

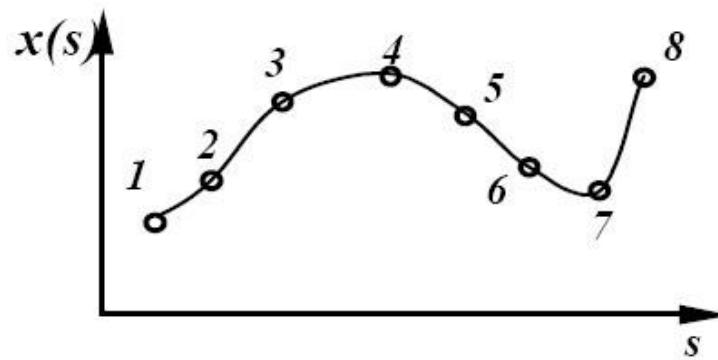
Первая причина этого в том, что формы объектов в прикладных задачах не должны зависеть от системы координат.

Кроме того, кривые могут иметь вертикальные касательные, что тесно связано с многозначностью функций. Кривая, заданная в неявном виде  $F(x, y) = 0$ , свободна от этого недостатка. Но для такой функции при вычислении текущих координат точек приходится каждый раз решать в общем случае нелинейное уравнение  $F(x, y) = 0$  от одного неизвестного  $x$  или  $y$ .



Пример плоской кривой в параметрическом виде:

$$x = x(s)$$



$$y = y(s)$$

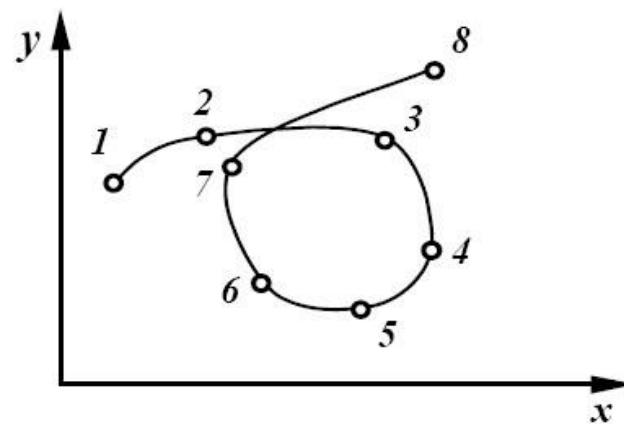
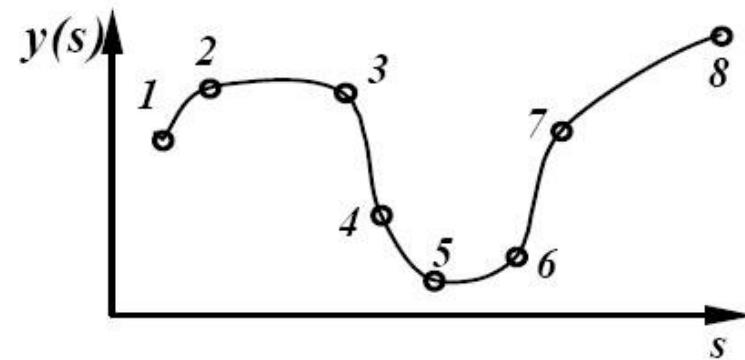
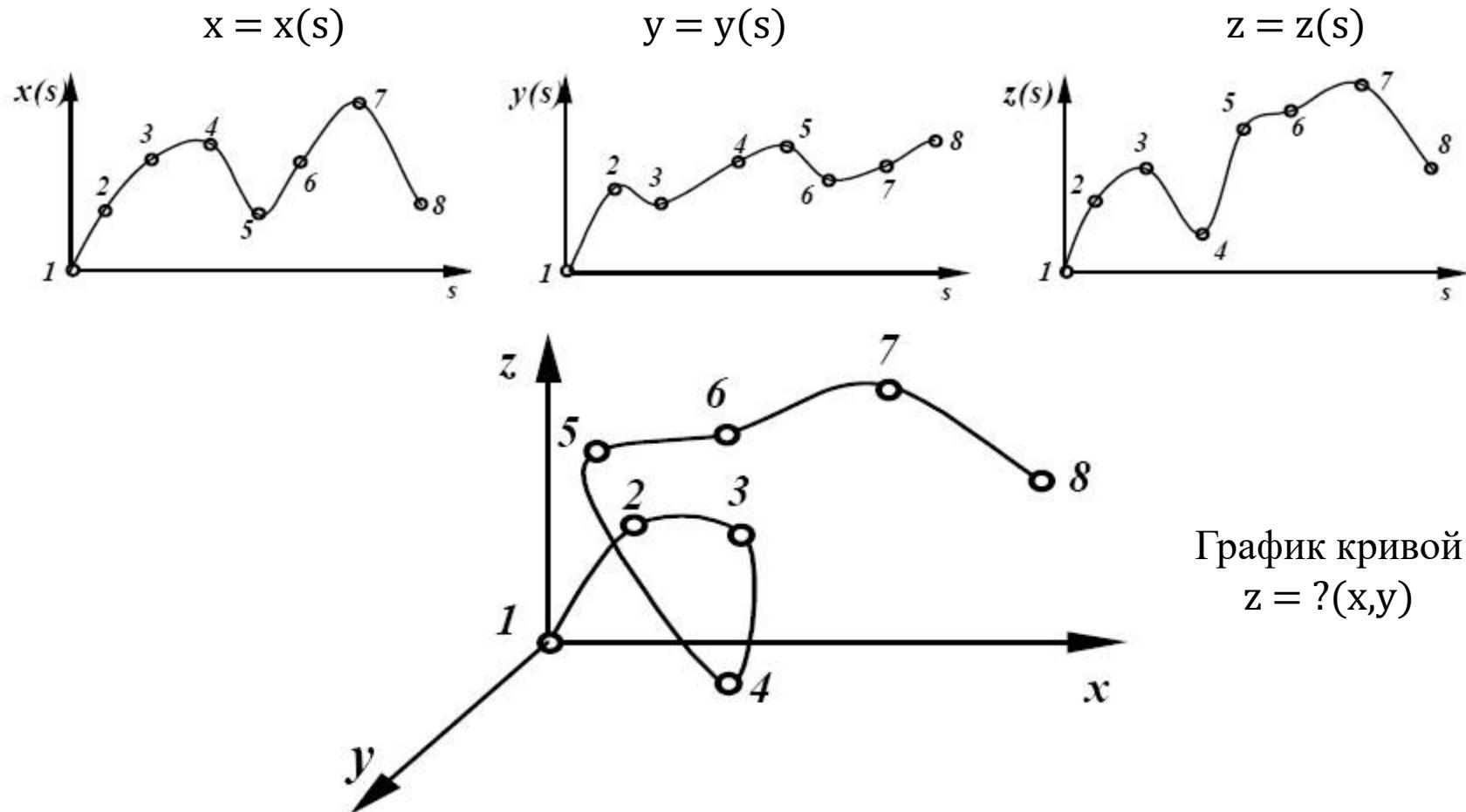


График кривой  
 $y = ?(x)$

Пример пространственной (3d) параметрическом виде:





Векторное представление производной (т.е. касательной):

$$\mathbf{P}'(t) = [x'(t) \ y'(t) \ z'(t)].$$

Так как точка на параметрической кривой определяется только значением параметра, эта форма не зависит от выбора системы координат. Конечные точки и длина кривой определяются диапазоном изменения параметра. Чаще всего удобнее нормализовать параметр  $t$  на интересующем отрезке кривой к  $0 \leq t \leq 1$ .

Осе-независимость параметрической кривой позволяет с легкостью проводить с ней аффинные преобразования.

### Основные параметрические кубические кривые:

- формы Эрмита (Hermite)
- кривые Безье (Bézier)
- кубические сплайны .

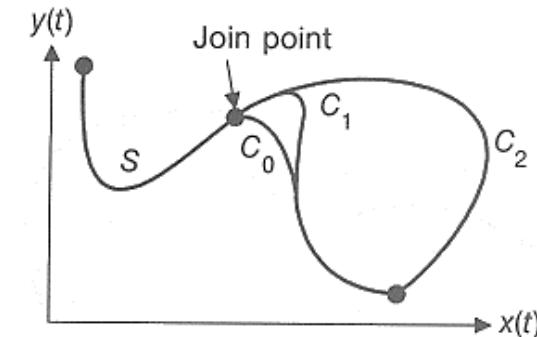
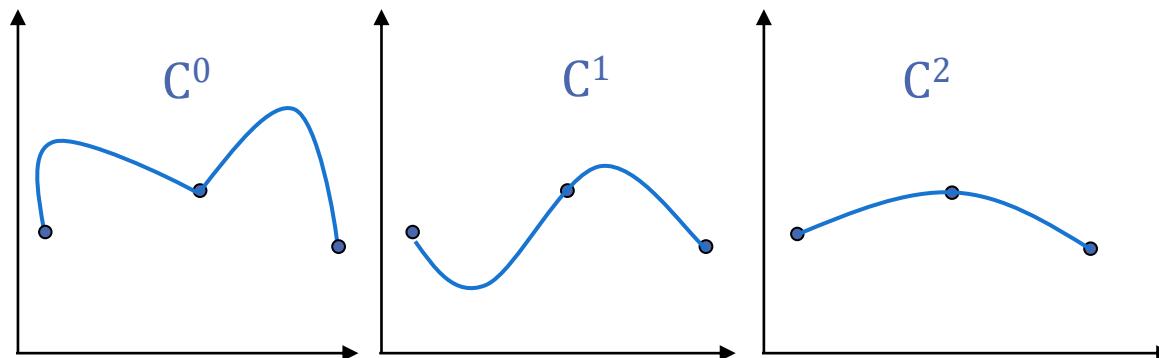


Непрерывность нулевого порядка по параметру,  $C^0$  – означает, что кривые встречаются, т.е.  $F_{k-1}(x_p) = F_k(x_p)$ .

Непрерывность первого порядка по параметру,  $C^1$  – означает, что первые производные по параметру ( $t$ ) двух кривых одинаковы в точке пересечения (стыковки), т.е.  $F'_{k-1}(x_p) = F'_k(x_p)$ .

Непрерывность второго порядка по параметру,  $C^2$  – означает, что первая и вторая производные по параметру ( $t$ ) двух кривых одинаковы в точке пересечения (стыковки), т.е.  $F'_{k-1}(x_p) = F'_k(x_p)$  и  $F''_{k-1}(x_p) = F''_k(x_p)$ .

Аналогичным образом определяется непрерывность по параметру более высоких порядков.

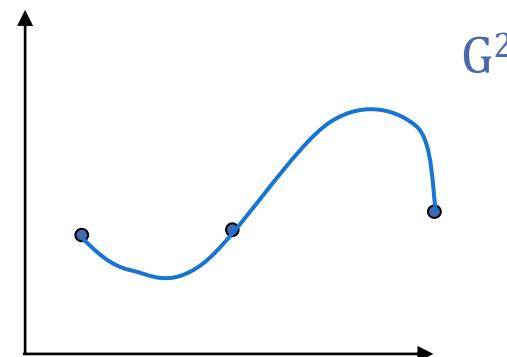
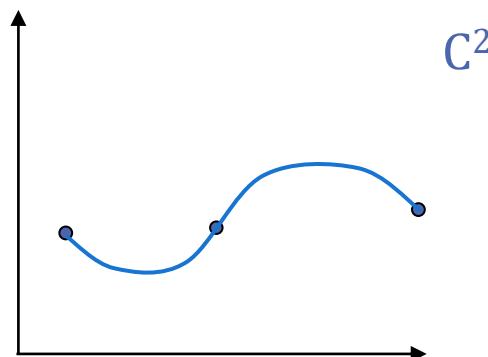




Геометрическая непрерывность нулевого порядка по параметру,  $G^0$  аналогична  $C^0$  –  $F_{k-1}(x_p) = F_k(x_p)$ .

Геометрическая непрерывность первого порядка по параметру,  $G^1$  – означает, что первые производные по параметру ( $t$ ) двух кривых пропорциональны в точке пересечения (стыковки), т.е. Вектора касательных в точке совпадают по направлению, но различаются по модулю (длине)  $F'_{k-1}(x_p) = aF'_k(x_p)$ ,  $a \neq 0.1$ .

Геометрическая непрерывность второго порядка по параметру,  $G^2$  – означает, что первая и вторая производные по параметру ( $t$ ) двух кривых пропорциональны в точке стыковки, т.е.  $F'_{k-1}(x_p) = aF'_k(x_p)$ ,  $a \neq 0.1$  и  $F''_{k-1}(x_p) = bF''_k(x_p)$ ,  $b \neq 0.1$ .



Сплайны ( $k$ -сплайны) – кусочные полиномы степени  $k$  с непрерывной производной степени  $k-1$  в точках соединения сегментов.

Иными словами, сплайнами называется набор функций, который вместе с некоторыми производными этих функций непрерывен на отрезке  $[a, b]$ , а на каждом частном интервале этого отрезка  $[x_i, x_{i+1}]$  в отдельности является некоторым многочленом невысокой степени.

В настоящее время чаще всего применяют кубический сплайн, то есть на каждом локальном интервале функция является полиномом 3-го порядка.

Сплайновая интерполяция напоминает лагранжевую тем, что требует только значения функции в узлах, но не её производных.



$P_1, P_2, P_3, P_4$  – известные значения функции в точках 0, 1, 2, 3.

Нужно вычислить значение  $Y$  для точки  $X$ , лежащей между 1 и 2.

$$D = P_2$$

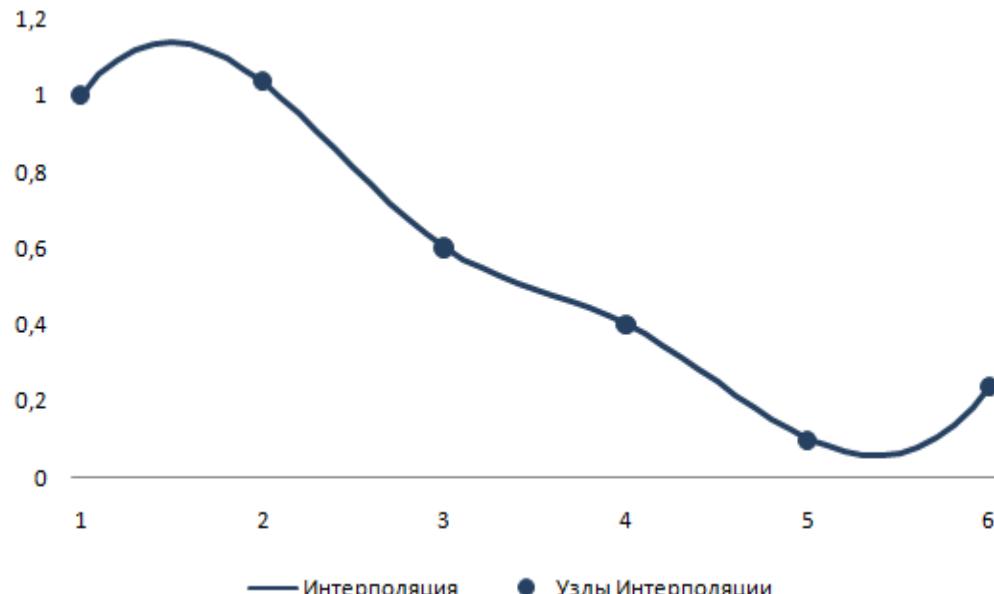
$$C = (P_3 - P_1)/2$$

$$A = -0.5 \cdot P_1 + 1.5 \cdot P_2 - 1.5 \cdot P_3 + 0.5 \cdot P_4$$

$$B = P_1 - 2.5 \cdot P_2 + 2 \cdot P_3 - 0.5 \cdot P_4$$

$$Z = X - 1$$

$$Y = A \cdot Z^3 + B \cdot Z^2 + C \cdot Z + D$$





**Интерполяционный многочлен Лагранжа** — многочлен минимальной степени, принимающий данные значения в данном наборе точек. Для  $n + 1$  пар чисел  $(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$ , где все  $x_j$  различны, существует единственный многочлен  $L(x)$  степени не более  $n$ , для которого  $L(x_j) = y_j$ .

Лагранж предложил способ вычисления таких многочленов:

$$L(x) = \sum_{i=0}^n y_i l_i(x)$$

где базисные полиномы определяются по формуле:

$$l_i(x) = \prod_{j=0, j \neq i}^n \frac{x - x_j}{x_i - x_j} = \frac{x - x_0}{x_i - x_0} \cdots \frac{x - x_{i-1}}{x_i - x_{i-1}} \cdot \frac{x - x_{i+1}}{x_i - x_{i+1}} \cdots \frac{x - x_n}{x_i - x_n}$$



## Интерполяционный полином

$$P_n(x) = y_0 l_0(x) + y_1 l_1(x) + \cdots + y_n l_n(x)$$

$l_j(x)$  – многочлены степени  $n$ .

Система уравнений

$$P_n(x_i) = y_i, \quad i = 0, 1, \dots, n$$

или

$$\begin{cases} y_0 l_0(x_0) + y_1 l_1(x_0) + \cdots + y_n l_n(x_0) = y_0 \\ \cdots \\ y_0 l_0(x_n) + y_1 l_1(x_n) + \cdots + y_n l_n(x_n) = y_n \end{cases}$$



Погрешность линейной интерполяции обусловлена тем, что график интерполирующей функции имеет изломы в узлах интерполяции. Эти изломы можно устранить, если в качестве интерполирующей использовать функцию, график которой представляет собой плавную кривую (например, полином), проходящий точно через заданные точки. Существует много разновидностей полиномов, полином Лагранжа лишь один из множества. Интерполяция полиномом Лагранжа дает высокую точность, если значения функции в смежных узлах, заданные в таблице, изменяются достаточно медленно.

Построим интерполяционный полином Лагранжа  $L_n(x)$ , следующим образом:

$$L_n(x) = l_0(x) + l_1(x) + \cdots + l_n(x) = \sum_{i=0}^n l_i(x)$$

где  $l_i(x)$  – полином.



Потребуем, чтобы в узловых точках  $x = x_j$  ( $j = 0, 1, \dots, n$  – номер узла),

$$l_i(x_j) = \begin{cases} y_i, & \text{если } j = i \\ 0, & \text{если } j \neq i \end{cases}$$

Полином  $l_i(x)$  составим следующим образом:

$$l_i(x) = C_i(x - x_0)(x - x_1) \dots (x - x_{i-1})(x - x_{i+1}) \dots (x - x_n).$$

В полиноме каждый из  $n$  сомножителей в скобках является разностью изменяющегося непрерывно аргумента  $x$  и дискретного значения  $x_j$  узла с номером  $j$ . Причем номер узла  $j$  принимает значения сначала от 0 до  $i - 1$ , затем от  $i + 1$  до  $n$ . Сомножитель  $x - x_i$  отсутствует. За счет этого  $l_i(x) = 0$  во всех узлах, кроме узла, номер которого  $j$  совпадает с номером полинома  $i$ . Таким образом, обеспечивается выполнение соотношения при  $j \neq i$ .

Для того чтобы соотношение выполнялось и при  $j = i$ , коэффициент  $C_i$  для полинома найдем из следующего условия:

$$l_i(x_i) = y_i$$

Откуда:

$$C_i = \frac{y_i}{(x_i - x_0)(x_i - x_1) \dots (x_i - x_{i-1})(x_i - x_{i+1}) \dots (x_i - x_n)}.$$

Подставляя выражение полученное в предыдущий полином, получим:

$$l_i(x) = y_i \frac{(x - x_0)(x - x_1) \dots (x - x_{i-1})(x - x_{i+1}) \dots (x - x_n)}{(x_i - x_0)(x_i - x_1) \dots (x_i - x_{i-1})(x_i - x_{i+1}) \dots (x_i - x_n)}$$

Перепишем полученное соотношение в более компактном виде:

$$l_i(x) = y_i \prod_{\substack{j=0 \\ j \neq i}}^n \frac{x - x_j}{x_i - x_j}$$



Недостатком линейной интерполяции является наличие изломов в узловых точках. Изломы будут отсутствовать, если в качестве приближающей функции  $F(x)$  используется полином вида:

$$F(x) = a_0 + a_1x + a_2x^2 + \cdots + l_n(x)$$

**Теорема:** Для фиксированной (заданной) интерполяционной таблицы существует единственный интерполяционный полином  $F(x)$ , проходящий через все её точки  $x_i, y_i$ .

Степень полинома  $F(x)$  равна числу интервалов  $n$  между узлами (на единицу меньше числа узлов).

Таблице с двумя узлами  $x_0, x_1$  соответствует полином первой степени (прямая):  $F(x) = a_0 + a_1 x$ .



$$\phi(x) = a_0 + a_1x + a_2x^2$$

$$x \in [x_{i-1}, x_{i+1}]$$

$$\begin{cases} \phi(x_{i-1}) = y_{i-1} \\ \phi(x_i) = y_i \\ \phi(x_{i+1}) = y_{i+1} \end{cases} \quad \text{– условие интерполяции}$$

$$\begin{cases} a_0 + a_1x_{i-1} + a_2x_{i-1}^2 = y_{i-1} \\ a_0 + a_1x_i + a_2x_i^2 = y_i \\ a_0 + a_1x_{i+1} + a_2x_{i+1}^2 = y_{i+1} \end{cases}$$

$$\begin{vmatrix} 1 & x_{i-1} & x_{i-1}^2 \\ 1 & x_i & x_i^2 \\ 1 & x_{i+1} & x_{i+1}^2 \end{vmatrix} \neq 0 \quad \text{– определитель Вандермонда}$$

$$a_0, a_1, a_2$$

– неизвестные переменные



В соответствии с соотношениями для каждой точки  $x_i, y_i$ , можно записать следующее уравнение:

$$F(x_i) = y_i .$$

Получится система из  $n + 1$  уравнения, решив которую можно найти  $n + 1$  значение неизвестных коэффициентов  $a_i$  интерполяционного полинома  $F(x)$ . На основании сформулированной выше теоремы эта система уравнений будет иметь решение.

**Рассмотрим пример.** Пусть надо найти интерполяционный полином (т. е. коэффициенты интерполяционного полинома) для точек, заданных следующей таблицей:

$x_i$	1	3	4
$F(x_i)$	12	4	6

$$n = 2$$

$x_0 = 1$	$x_1 = 3$	$x_2 = 4$	4
$y_0 = 12$	$y_1 = 4$	$x_0 = 6$	6



Составим следующую систему уравнений:

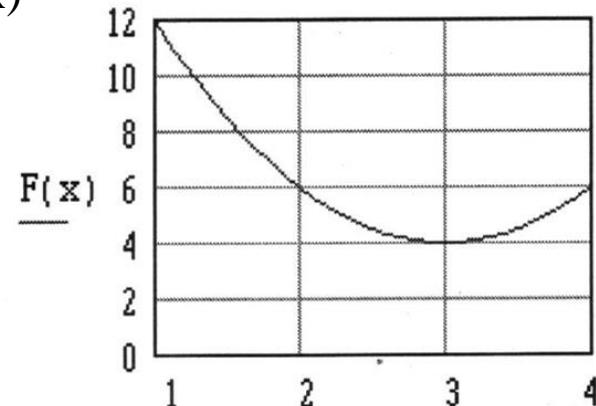
$$\begin{cases} F(x_0) = a_0 + a_1 x_0 + a_2 x_0^2 = a_0 + a_1 + a_2 = 12 \\ F(x_1) = a_0 + a_1 x_1 + a_2 x_1^2 = a_0 + a_1 \cdot 3 + a_2 \cdot 9 = 4 \\ F(x_2) = a_0 + a_1 x_2 + a_2 x_2^2 = a_0 + a_1 \cdot 4 + a_2 \cdot 16 = 6 \end{cases}$$

Решаем систему уравнений и находим коэффициенты  $a_0, a_1, a_2$  интерполяционного полинома  $F(x)$

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 3 & 9 \\ 1 & 4 & 16 \end{pmatrix}, \quad Y = \begin{pmatrix} 12 \\ 4 \\ 6 \end{pmatrix}, \quad a = A^{-1}Y \rightarrow a = \begin{pmatrix} 22 \\ -12 \\ 2 \end{pmatrix}$$

Построим график интерполяционного полинома  $F(x)$

$$F(x) = a_0 + a_1 x + a_2 x^2 \quad x = 1, 1.1, \dots, 6$$



Таким образом, интерполяционный полином  $F(x) = 22 - 12x + 2x^2$  проходит через все три точки, заданные в исходной таблице.



## Алгоритм

1. Определить отрезок  $[x_{i-1}, x_{i+1}]$ , содержащий  $x^*$ .
2. Решить систему, для определения  $a_0, a_1, a_2$ .
3. Подставить  $x^*$  в функцию  $\phi(x)$  при известных коэффициентах  $a_i$ .



**ПРИМЕР:** Найдем формулу интерполяции для  $f(x) = \tan(x)$  имеющей следующие значения:

$x_0 = -1.5$	$f(x_0) = -14,1014$
$x_1 = -0.75$	$f(x_1) = -0,931596$
$x_2 = 0$	$f(x_2) = 0$
$x_3 = 0.75$	$f(x_3) = 0,931596$
$x_4 = 1.5$	$f(x_4) = 14,1014.$

$$\ell_0(x) = \frac{x - x_1}{x_0 - x_1} \cdot \frac{x - x_2}{x_0 - x_2} \cdot \frac{x - x_3}{x_0 - x_3} \cdot \frac{x - x_4}{x_0 - x_4} = \frac{1}{243}x(2x - 3)(4x - 3)(4x + 3)$$

$$\ell_1(x) = \frac{x - x_0}{x_1 - x_0} \cdot \frac{x - x_2}{x_1 - x_2} \cdot \frac{x - x_3}{x_1 - x_3} \cdot \frac{x - x_4}{x_1 - x_4} = -\frac{8}{243}x(2x - 3)(2x + 3)(4x - 3)$$

$$\ell_2(x) = \frac{x - x_0}{x_2 - x_0} \cdot \frac{x - x_1}{x_2 - x_1} \cdot \frac{x - x_3}{x_2 - x_3} \cdot \frac{x - x_4}{x_2 - x_4} = \frac{3}{243}(2x + 3)(4x + 3)(4x - 3)(2x - 3)$$

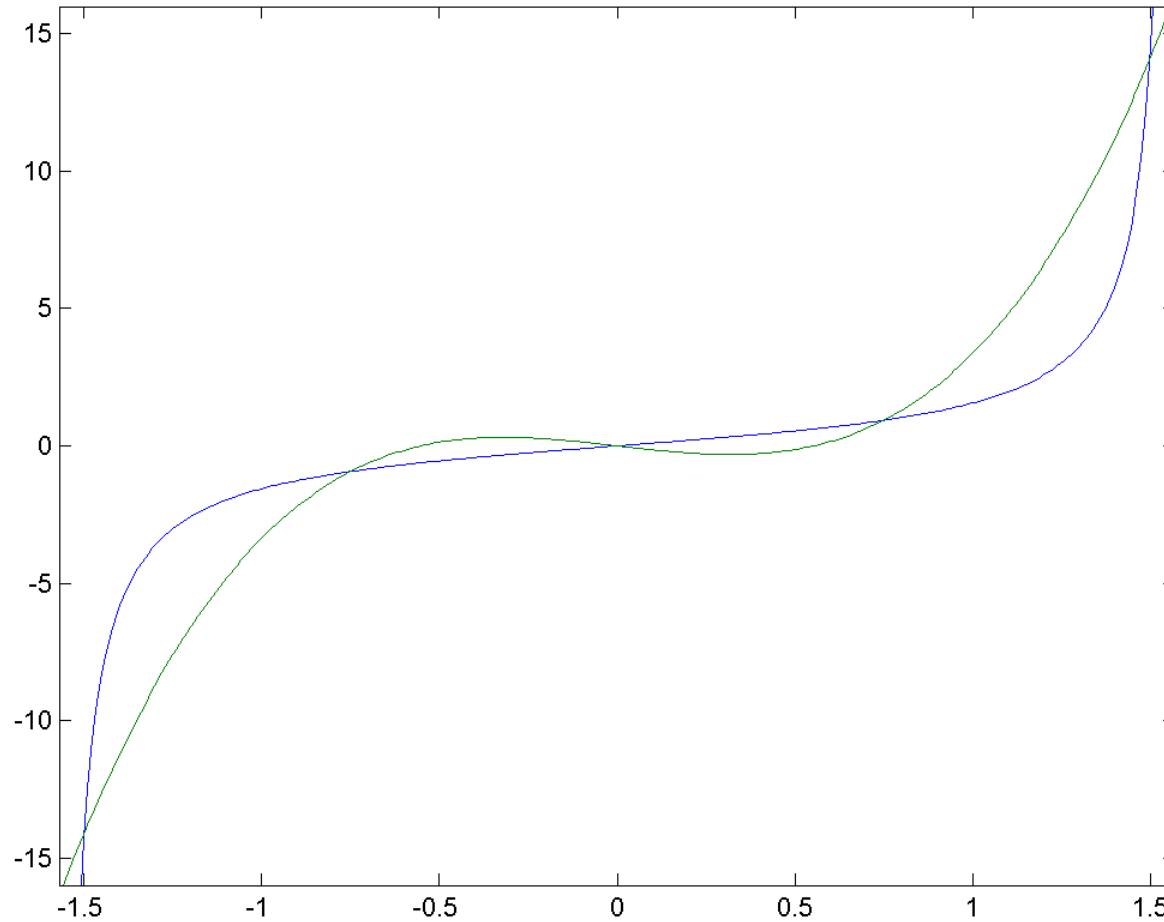
$$\ell_3(x) = \frac{x - x_0}{x_3 - x_0} \cdot \frac{x - x_1}{x_3 - x_1} \cdot \frac{x - x_2}{x_3 - x_2} \cdot \frac{x - x_4}{x_3 - x_4} = -\frac{8}{243}x(2x - 3)(2x + 3)(4x + 3)$$

$$\ell_4(x) = \frac{x - x_0}{x_4 - x_0} \cdot \frac{x - x_1}{x_4 - x_1} \cdot \frac{x - x_2}{x_4 - x_2} \cdot \frac{x - x_3}{x_4 - x_3} = \frac{1}{243}x(2x + 3)(4x - 3)(4x + 3).$$

$$\begin{aligned} L(x) &= \frac{1}{243} \left( f(x_0)x(2x - 3)(4x - 3)(4x + 3) \right. \\ &\quad - 8f(x_1)x(2x - 3)(2x + 3)(4x - 3) \\ &\quad + 3f(x_2)(2x + 3)(4x + 3)(4x - 3)(2x - 3) \\ &\quad - 8f(x_3)x(2x - 3)(2x + 3)(4x + 3) \\ &\quad \left. + f(x_4)x(2x + 3)(4x - 3)(4x + 3) \right) \\ &= 4,834848x^3 - 1,477474x. \end{aligned}$$



**ПРИМЕР:** Найдем формулу интерполяции для  $f(x) = \tan(x)$  имеющей следующие значения:



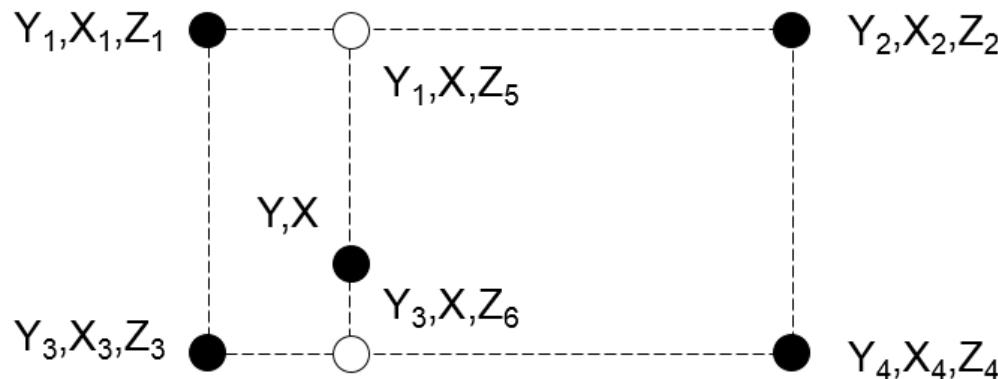
Функция тангенса и интерполяция



**Билинейная интерполяция** — расширение линейной интерполяции для функций двух переменных. Ключевая идея заключается в том, чтобы провести обычную линейную интерполяцию сначала в одном направлении, затем в перпендикулярном. Формула билинейной интерполяции интерполирует значения функции в произвольном прямоугольнике по четырем её значениям в вершинах прямоугольника и экстраполирует функцию на всю остальную поверхность.

Даны четыре точки:  $(X_1, Y_1, Z_1)$ ,  $(X_2, Y_2, Z_2)$ ,  $(X_3, Y_3, Z_3)$  и  $(X_4, Y_4, Z_4)$ .

Найти  $Z$  для заданной точки  $X, Y$ .



Пусть  $F(X_1, Y_1, X_2, Y_2, X)$  вычисляет линейную интерполяцию для точки  $X$  по точкам  $(X_1, Y_1)$  и  $(X_2, Y_2)$ .

Вычисление билинейной интерполяции:

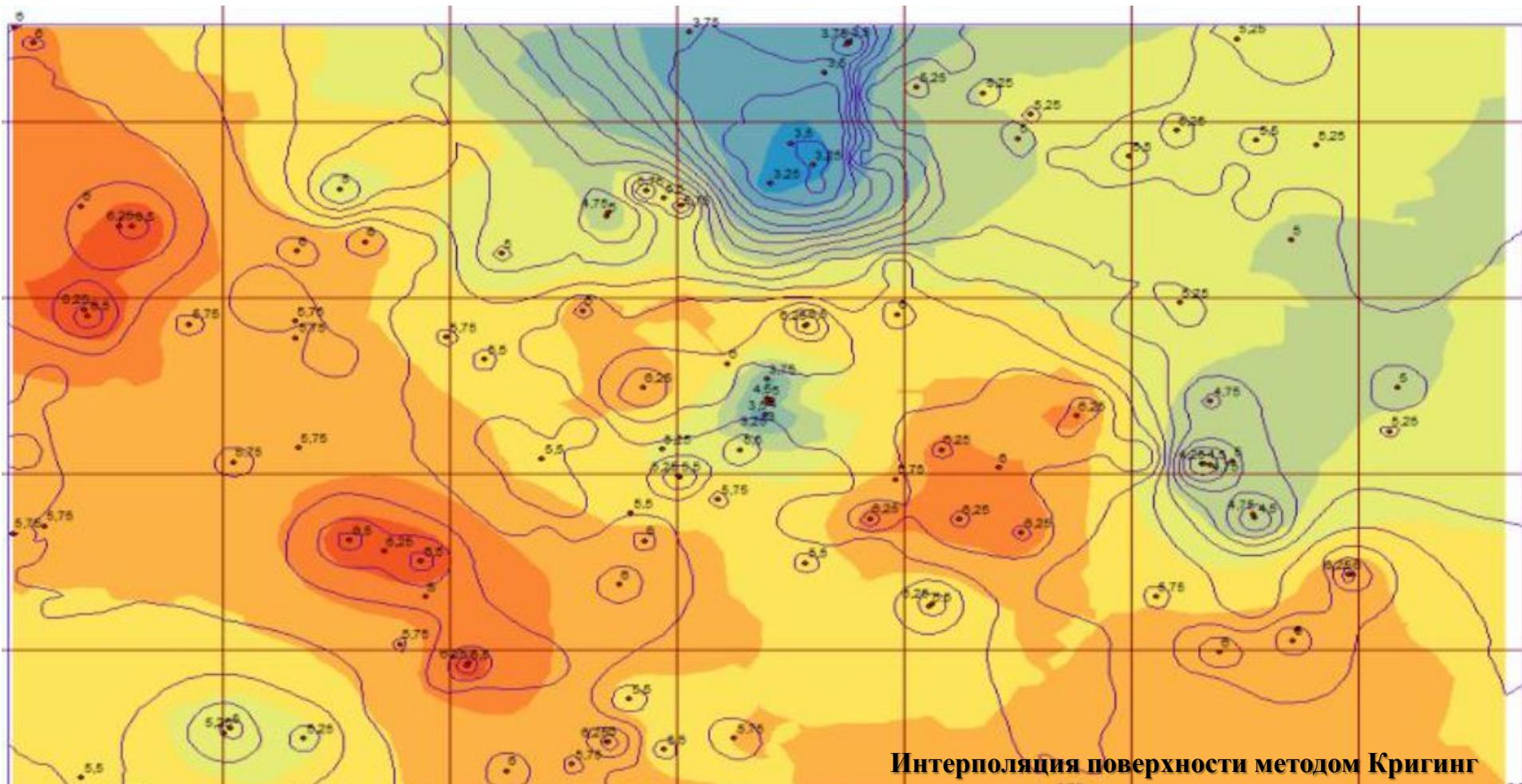
$$Z_5 = F(X_1, Z_1, X_2, Z_2, X)$$

$$Z_6 = F(X_3, Z_3, X_4, Z_4, X)$$

$$Z = F(Y_1, Z_5, Y_3, Z_6, Y)$$



Интерполяция изолиний является важным инструментом в географических информационных системах (ГИС), позволяя создавать непрерывные поверхности на основе дискретных данных. Этот метод используется для визуализации таких явлений, как рельеф местности, распределение температуры или давления. В ГИС интерполяция изолиний применяется для создания цифровых моделей рельефа (ЦМР), которые используются в различных приложениях, включая картографию, анализ ландшафта и моделирование природных процессов.





Полиномиальная интерполяция не всегда дает удовлетворительные результаты при аппроксимации функций. Несмотря на выполнение условий в узлах, интерполирующая функция может иметь значительные отклонения между узлами. Увеличение степени интерполяционного многочлена не всегда приводит к уменьшению погрешности. Возникает так называемое явление волнистости. При этом поведение полинома в окрестности какой-либо точки определяет его поведение в целом. Полиномиальная интерполяция дает особенно большие ошибки, если в окрестности левой или правой границы интервала интерполяции находится вертикальная асимптота графика функции.



$x_i$	$x_0$	$x_1$	...	$x_n$
$y_i$	$y_0$	$y_1$	...	$y_n$

Интерполяция, использующая сразу все  $n$  узлов таблицы, называется глобальной интерполяцией.

Начиная с  $n > 7$  глобальная многочленная интерполяция становится неустойчивой, поэтому обычную многочленную интерполяцию осуществляют максимум по 3-4 узлам. Интерполяцию по нескольким узлам сеточной функции называют локальной.



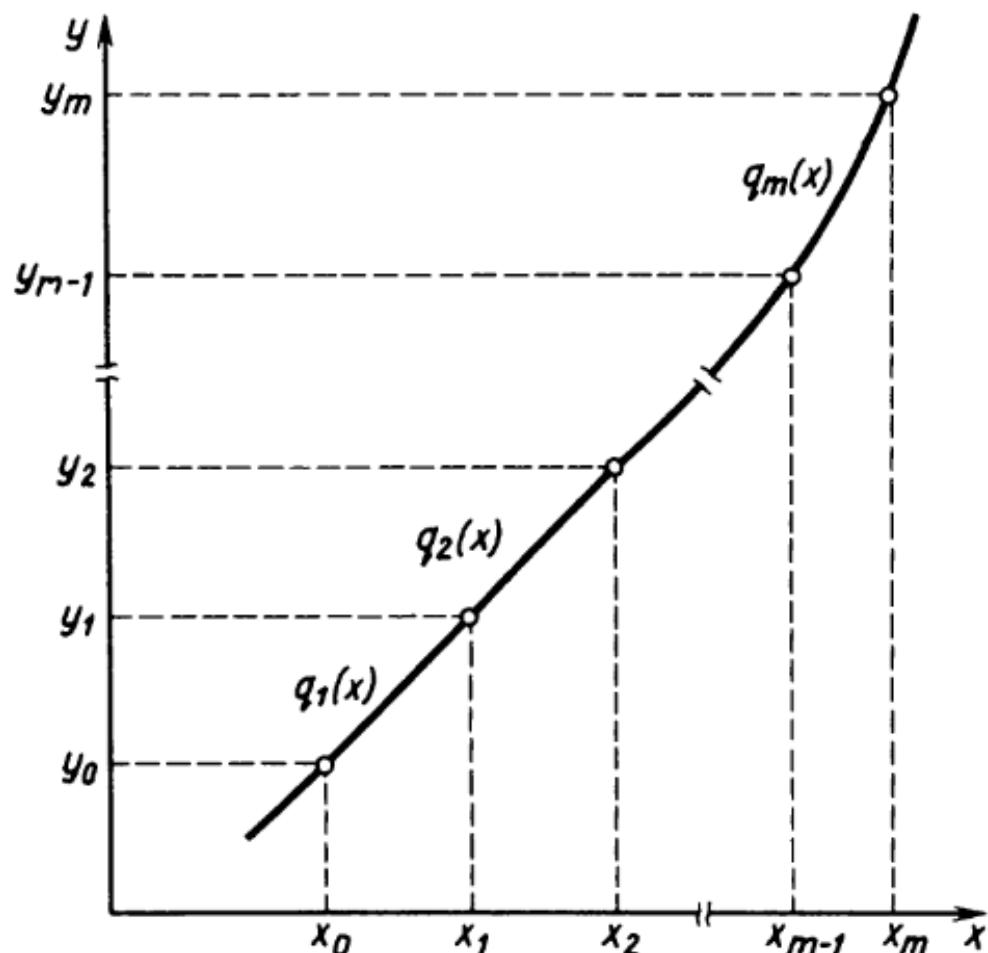
Однако локальная интерполяция обладает тем недостатком, что интерполирующая функция в узлах стыковки многочлена имеет непрерывность только нулевого порядка.

От этих недостатков свободна сплайн-интерполяция, которая требует непрерывности в узлах стыковки локальных многочленов по производным порядка один, два и т.д.



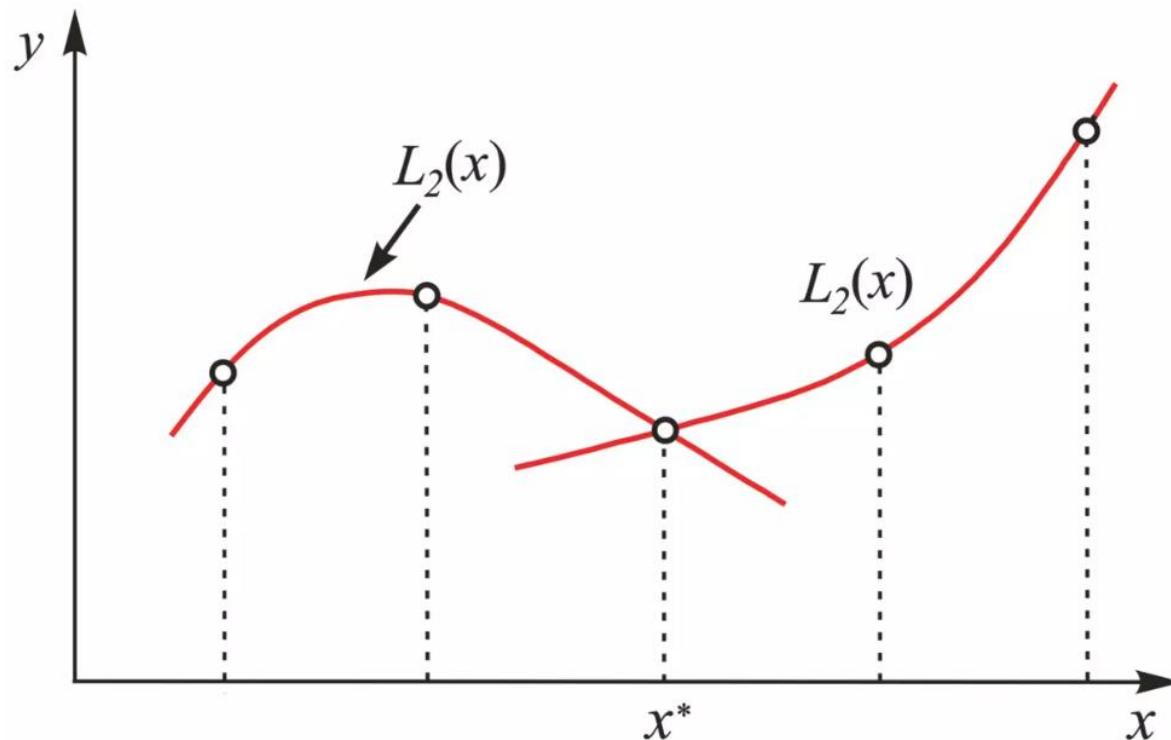
Математическая теория такой аппроксимации называется теорией сплайн-функций (от английского слова spline – рейка, линейка).

График интерполирующей функции при сплайн-интерполяции действительно напоминает гибкую линейку, закрепленную в узловых точках интерполируемой функции. Поэтому сплайн-интерполяцию выгодно применять при небольшом числе узловых точек (до 5 – 7).





Сплайном степени  $m$  дефекта  $r$  называется  $(m - r)$  раз непрерывно дифференцируемая функция, которая



на каждом отрезке  $[x_i - 1, x_i]$ ,  $i = 1, 2, \dots, n$ , представляет собой многочлен степени  $m$ .

Сплайны, удовлетворяющие условию интерполяции называются интерполяционными.



Рассмотрим интерполяцию кубическими сплайнами  $\phi^{(IV)}(x) = 0$  – группу сопряженных кубических многочленов, в местах сопряжения которых первая и вторая производные непрерывны.

Чтобы построить кубический сплайн, необходимо задать коэффициенты, которые единственным образом определяют кубический многочлен в промежутке между данными точками.

Если в качестве функции  $\varphi(x)$  выбрать полином, то степень полинома должна быть не выше третьей. Этот полином называют кубическим сплайном, который на каждом интервале записывают в виде:

$$S_i(x) = a_0 + a_1(x - x_{i-1}) + a_2(x - x_{i-1})^2 + a_3(x - x_{i-1})^3,$$

где  $a_i$  – коэффициенты сплайна;

$i = 1, 2, \dots$  – номер интервала (номер сплайна).



В отличие от полиномиальной интерполяции, когда вся аппроксимирующая функция описывается одним полиномом, при сплайновой интерполяции на каждом интервале строится отдельный полином  $\varphi(x)$  третьей степени со своими коэффициентами.

Аппроксимирующая функция  $F(x)$  представляет собой последовательность сплайнов «сшитых» между собой в точках, соответствующих узловым значениям аппроксимируемой функции  $f(x)$ .

Для определения коэффициентов  $a_i, b_i, c_i, d_i$  на всех  $n$  элементарных отрезках необходимо получить  $4n$  уравнений. Эти уравнения получаются из следующих условий “сшивания” соседних сплайнов.



Для определения 4 x п коэффициентов имеются следующие условия в узлах интерполяции:

1. Условие интерполяции  $S(x_i) = y_i, i = \overline{0, n};$
2. Непрерывность сплайнов

$$S(x_i - 0) = S(x_i + 0), i = \overline{1, n - 1};$$

3. Непрерывность производных 1-го порядка

$$S'(x_i - 0) = S' (x_i + 0), i = \overline{1, n - 1};$$

4. Непрерывность производных 2-го порядка

$$S''(x_i - 0) = S''(x_i + 0), i = \overline{1, n - 1};$$

5.  $S''(x_0) = 0;$
6.  $S''(x_n) = 0.$



Непрерывность первой производной означает, что соседние сплайны в узловых точках имеют общую касательную. Непрерывность второй производной означает, что соседние сплайны в узловых точках имеют одинаковую кривизну.

Эта система равенств содержит  $2n - 2$  уравнений.

Кроме перечисленных условий, необходимо задать условия на концах отрезка, то есть в точках  $x_0$  и  $x_n$ . В общем случае эти условия определяются конкретной задачей.

Если за пределами интервала интерполяции экстраполирующая функция представляет собой прямую линию, то это линейная экстраполяция. Следовательно, исходя из условий непрерывности второй производной, запишем еще два равенства:

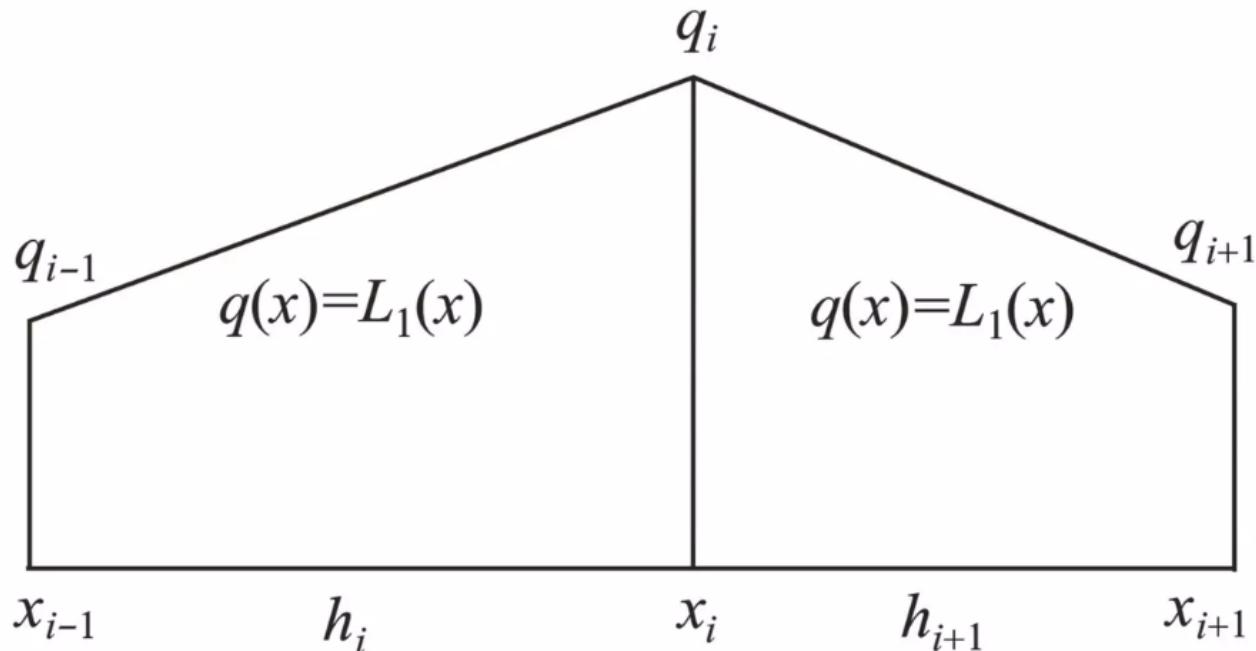
$$\phi_0''(x_0) = 0 \text{ и } \phi_n''(x_n) = 0$$

Если за пределами интервала интерполяции экстраполирующая функция представляет собой кубический или квадратический полином, то это кубическая или параболическая экстраполяция.



Пусть  $S''(x) = q(x)$ .

Рассмотрим поведение функции  $q(x)$  на отрезке  $[x_{i-1}; x_i]$ .



Поведение функции  $S''(x)$  на элементарных отрезках



Найдем вторую производную с помощью интерполяционного многочлена Лагранжа 1-й степени  $L_1(x)$  на каждом отрезке  $[x_{i-1}, x_i]$ :

$$q(x) = q_{i-1} \frac{x - x_i}{x_{i-1} - x_i} + q_i \frac{x - x_{i-1}}{x_i - x_{i-1}}$$

Выражение уже удовлетворяет условиям непрерывности производных 2-го порядка.

$$x = x_i - 0 \rightarrow (1) \rightarrow q(x_i - 0) = q_i$$

Выписывая выражение (1) для отрезка  $[x_{i-1}; x_i]$ :

$$q(x) = q_{i-1} \frac{x - x_i}{x_{i-1} - x_i} + q_i \frac{x - x_{i-1}}{x_i - x_{i-1}}$$

и подставляя в него  $x_i + 0$  вместо  $x$ , получим  $q(x_i + 0) = q_i$ , что и требовалось показать.



Этапы нахождения сплайна:

1. Проинтегрируем дважды выражение для  $q(x)$ , получим:

$$S(x) = q_{i-1} \frac{(x_i - x)^3}{6h_i} + q_i \frac{(x - x_{i-1})^3}{6h_i} + C_1 x + C_2,$$

где  $C_1$  и  $C_2$  найдем из удовлетворения значений сплайна в узлах  $x_{i-1}$ ,  $x_i$  условиям интерполяции.

$$S(x_{i-1}) = y_{i-1} = q_{i-1} \frac{(x_i - x_{i-1})^3}{6h_i} + q_i \frac{(x_{i-1} - x_{i-1})^3}{6h_i} + C_1 x_{i-1} + C_2,$$

$$S(x_i) = y_i = q_{i-1} \frac{(x_i - x_i)^3}{6h_i} + q_i \frac{(x_i - x_{i-1})^3}{6h_i} + C_1 x_i + C_2.$$



2. Решая эту СЛАУ относительно  $C_1$ ,  $C_2$  и подставляя их в выражение для  $S(x)$ , получим:

$$\begin{aligned} S(x) = & q_{i-1} \frac{(x_i - x)^3}{6h_i} + q_i \frac{(x - x_{i-1})^3}{6h_i} + \left( \frac{y_{i-1}}{h_i} - q_{i-1} \frac{h_i}{6} \right) \times \\ & \times (x_i - x) + \left( \frac{y_i}{h_i} - q_i \frac{h_i}{6} \right) (x - x_{i-1}), x \in [x_{i-1}, x_i]. \end{aligned}$$

3. Узловые значения для вторых производных  $q_i$  будем искать из условий непрерывности первых производных в узлах  $x_i$ .



4. Для нахождения производной  $S'(x_i + 0)$  запишем для отрезка  $[x_i, x_{i+1}]$

$$\begin{aligned} S(x) = & q_i \frac{(x_{i+1} - x)^3}{6h_{i+1}} + q_{i+1} \frac{(x - x_i)^3}{6h_{i+1}} + \left( \frac{y_i}{h_{i+1}} - q_i \frac{h_{i+1}}{6} \right) \times \\ & \times (x_{i+1} - x) + \left( \frac{y_{i+1}}{h_{i+1}} - q_{i+1} \frac{h_{i+1}}{6} \right) (x - x_i), x \in [x_i, x_{i+1}]. \end{aligned}$$

5. Вычисляя производные первого порядка от последних двух выражений  $S(x)$  и подставляя в них значение  $x = x_i$ , получим.

$$S'(x_i - 0) = q_{i-1} \frac{h_i}{6} + q_i \frac{h_i}{3} + \frac{y_i - y_{i-1}}{h_i},$$

$$S'(x_i + 0) = -q_i \frac{h_{i+1}}{3} - q_{i+1} \frac{h_{i+1}}{6} + \frac{y_{i+1} - y_i}{h_{i+1}}.$$



6. Приравнивая эти выражения в соответствии с условиями непрерывности первых производных в узлах интерполяции  $x_i$ , получим

$$q_{i-1} \frac{h_i}{6} + q_i \frac{h_i + h_{i+1}}{3} + q_{i+1} \frac{h_{i+1}}{6} = \frac{y_{i+1} - y_i}{h_{i+1}} - \frac{y_i - y_{i-1}}{h_i},$$
$$i = \overline{1, n-1};$$
$$q_0 = q_n = 0.$$

7. Подставляя найденные  $q_i, i = \overline{0, n}$  в выражение для  $S(x)$ , получим кубические сплайны дефекта один на каждом отрезке  $x \in [x_{i-1}, x_i], i = \overline{1, n}$ .



Для заданной таблицы с  $h = x_i - x_{i-1} = 1 = \text{const}$  выписать интерполяционные кубические сплайны дефекта один на каждом отрезке  $x \in [x_{i-1}, x_i], i = \overline{1, 4}$ . Проверить непрерывность сплайнов и их производных до второго порядка включительно в узле  $x^* = 2$ .

$i$	0	1	2	3	4
$x_i$	$x_0 = 1$	$x_1 = 2$	$x_2 = 3$	$x_3 = 4$	$x_4 = 5$
$y_i$	$y_0 = 1$	$y_1 = 3$	$y_2 = 6$	$y_3 = 9$	$y_4 = 21$



Решение. Для узлов  $x_1 = 2; x_2 = 3; x_3 = 4$  с учетом  $q_0 = q_4 = 0$  составляется СЛАУ относительно неизвестных  $q_1, q_2, q_3$ :

$$i = 1: \quad \frac{2}{3}q_1 + \frac{1}{6}q_2 = \frac{y_2 - y_1}{1} - \frac{y_1 - y_0}{1} = 1,$$

$$i = 2: \quad \frac{1}{6}q_1 + \frac{2}{3}q_2 + \frac{1}{6}q_3 = \frac{y_3 - y_2}{1} - \frac{y_2 - y_1}{1} = 0,$$

$$i = 3: \quad \frac{1}{6}q_2 + \frac{2}{3}q_3 = \frac{y_4 - y_3}{1} - \frac{y_3 - y_2}{1} = 9.$$



Вычисляются прогоночные коэффициенты по формулам:

$$A_i = \frac{-c_i}{b_i + a_i A_{i-1}}; B_i = \frac{d_i - a_i B_{i-1}}{b_i + a_i A_{i-1}}, i = 1, 2, 3$$

$$a_1 = c_3 = 0, A_1 = -\frac{1}{4}; B_1 = \frac{3}{2}; A_2 = -\frac{4}{15}; B_2 = -\frac{2}{5}$$
$$A_3 = 0; B_3 = \frac{102}{7}$$

и значения

$$q_i = A_i q_{i+1} + B_i, i = 3, 2, 1: q_3 = A_3 q_4 + B_3 = B_3 = 102/7$$

$$q_2 = A_2 q_3 + B_2 = -\frac{4}{15} \cdot \frac{102}{7} - \frac{2}{5} = -\frac{30}{7}$$

$$q_1 = A_1 q_2 + B_1 = -\frac{1}{4} \cdot \left(-\frac{30}{7}\right) + \frac{3}{2} = \frac{18}{7}$$



Для каждого из четырех интервалов выписываем сплайны:

$$i = 1: \quad S_1(x) = \frac{18}{42}(x - 1)^3 + (2 - x) + \frac{108}{42}(x - 1), \quad x \in [1; 2];$$

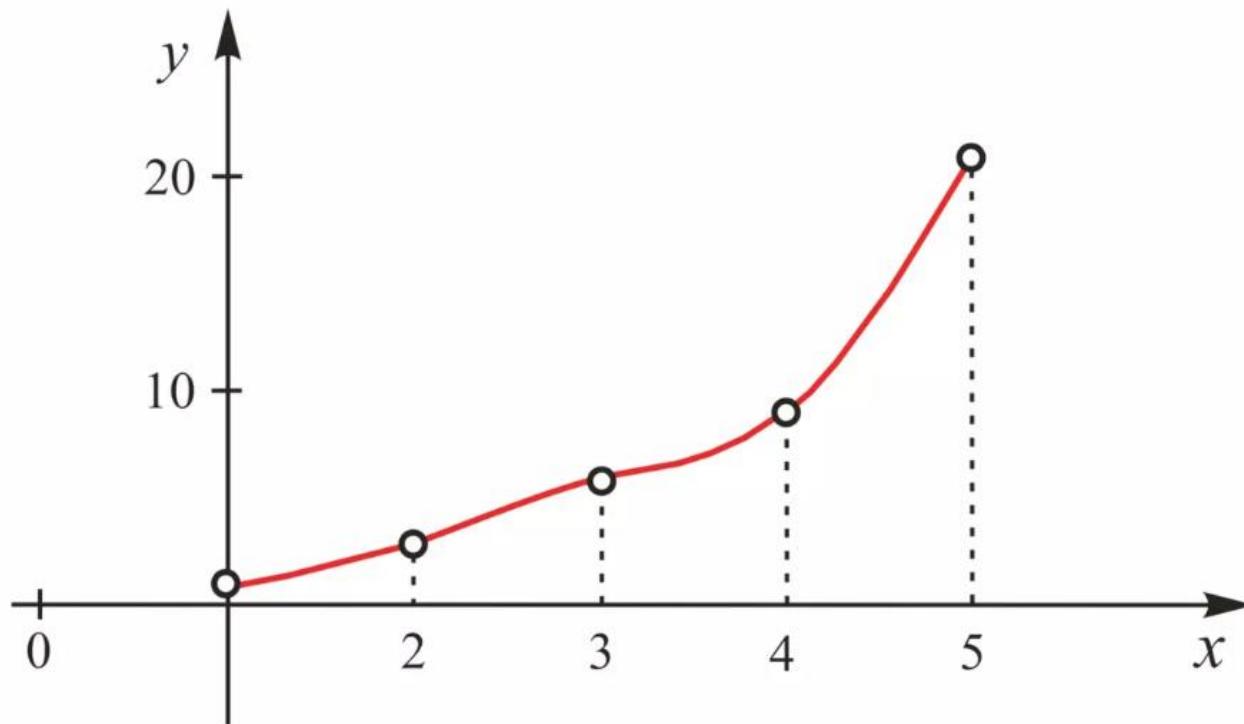
$$i = 2: \quad S_2(x) = \frac{18}{42}(3 - x)^3 + \left(-\frac{30}{42}\right)(x - 2)^3 + \\ + \frac{108}{42}(3 - x) + \frac{282}{42}(x - 2), \quad x \in [2; 3];$$

$$i = 3: \quad S_3(x) = -\frac{30}{42}(4 - x)^3 + \frac{102}{42}(x - 3)^2 + \\ + \frac{282}{42}4 - x + \frac{276}{42}(x - 3), \quad x \in [3; 4];$$



$$i = 4: S_4(x) = \frac{102}{42}(5-x)^3 + \frac{276}{42}(5-x) + 21(x-4), x \in [4; 5].$$

Построим полученные сплайны:





Проверим правильность построения сплайнов для узла  $x^* = 2$ . К нему примыкают сплайны  $S_1(x)$  и  $S_2(x)$ :

$$S_1(2 - 0) = 3; \quad S_2(2 + 0) = 3;$$

$$S'_1(2 - 0) = \frac{120}{42}; \quad S'_2(2 + 0) = \frac{120}{42};$$

$$S''_1(2 - 0) = \frac{108}{42}; \quad S''_2(2 + 0) = \frac{108}{42}.$$



**Экстраполяция** — определение будущих, ожидаемых значений величин, показателей на основе имеющихся данных о тенденциях их изменений в прошлые периоды. Математически сводится к продолжению кривой.

Методы экстраполяции во многих случаях сходны с методами интерполяции.

- Общее значение – распространение выводов, полученных из наблюдения над одной частью явления, на другую его часть.
- В маркетинге – распространение выявленных закономерностей развития изучаемого предмета на будущее.
- В статистике – распространение установленных в прошлом тенденций на будущий период (экстраполяция во времени применяется для перспективных расчетов населения); распространение выборочных данных на другую часть совокупности, не подвергнутую наблюдению.



**Метод наименьших квадратов** — математический метод, применяемый для решения различных задач, основанный на минимизации суммы квадратов отклонений некоторых функций от искомых переменных.

Суть метода наименьших квадратов (**МНК**). Задача заключается в нахождении коэффициентов линейной зависимости, при которых функция двух переменных  $a$  и  $b$

$$F(a, b) = \sum_{i=1}^n (y_i - (ax_i + b))^2$$

принимает наименьшее значение. То есть, при данных  $a$  и  $b$  сумма квадратов отклонений экспериментальных данных от найденной прямой будет наименьшей. В этом вся суть метода наименьших квадратов.

Таким образом, решение примера сводится к нахождению экстремума функции двух переменных.



Дана таблица исходных данных. Используя метод наименьших квадратов, аппроксимировать эти данные какой либо зависимостью, например линейной  $y = ax + b$  (коэффициенты  $a, b$  - неизвестные).

Находим частные производные функции по приведенной формуле  $F(a, b)$  по переменным  $a$  и  $b$ , приравниваем эти производные к нулю.

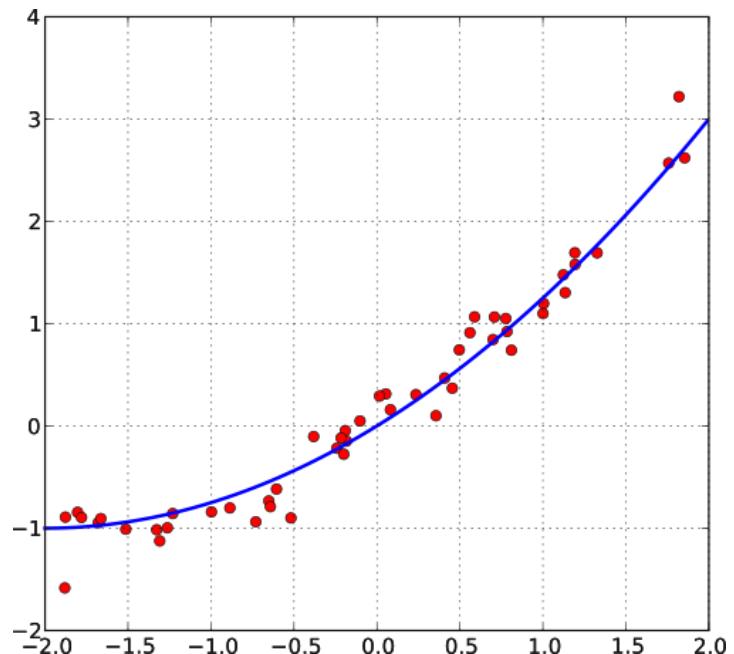
$$\begin{cases} \frac{\partial F(a, b)}{\partial a} = 0 \\ \frac{\partial F(a, b)}{\partial b} = 0 \end{cases} \Leftrightarrow \begin{cases} -2 \sum_{i=1}^n (y_i - (ax_i + b)) x_i = 0 \\ -2 \sum_{i=1}^n (y_i - (ax_i + b)) = 0 \end{cases} \Leftrightarrow$$

$$\Leftrightarrow \begin{cases} a \sum_{i=1}^n x_i^2 + b \sum_{i=1}^n x_i = \sum_{i=1}^n x_i y_i \\ a \sum_{i=1}^n x_i + b \sum_{i=1}^n 1 = \sum_{i=1}^n y_i \end{cases} \Leftrightarrow \begin{cases} a \sum_{i=1}^n x_i^2 + b \sum_{i=1}^n x_i = \sum_{i=1}^n x_i y_i \\ a \sum_{i=1}^n x_i + nb = \sum_{i=1}^n y_i \end{cases}$$



Решаем полученную систему уравнений любым методом (например методом подстановки или методом Крамера) и получаем формулы для нахождения коэффициентов по методу наименьших квадратов.

$$\begin{cases} a = \frac{n \sum_{i=1}^n x_i y_i - \sum_{i=1}^n x_i \sum_{i=1}^n y_i}{n \sum_{i=1}^n x_i^2 - (\sum_{i=1}^n x_i)^2} \\ b = \frac{\sum_{i=1}^n y_i - a \sum_{i=1}^n x_i}{n} \end{cases}$$





## Домашняя работа #3

Задание посвящено исследованию деятельности компании Rational AG (тикер: RAA.DE), а именно, анализу динамики вероятности дефолта компании в период ковида 2019-2021 гг., используя известные исторические данные.

Задание требуется выполнить используя *методы кубической сплайн-интерполяции и половинного деления*.

Используйте *только* стандартные методы Python. Можно использовать math, базовый numpy, scipy.stats.norm.cdf().

Срок сдачи работ: **до 21-го марта, 23:59.**

Работы принимаются на электронный адрес [VSChernyshenko@fa.ru](mailto:VSChernyshenko@fa.ru) **исключительно в виде ноутбука в формате html**. Название файла: “Фамилия\_Имя\_группа\_ДЗ2.html”.

**Не принимаются** работы являющиеся копией друг друга, работы **не содержащие подробные комментарии**.



## Домашняя работа #3

Воспользуйтесь ретроспективными экономическими данными, доступными по ссылке: <https://cloud.mail.ru/public/XggH/jkDghtZFR>.

Искомый датасет содержит четыре столбца:

- ***Date*** – Дата. Данные содержат среднемесячные показатели.
- ***Stock\_Price* (E)** – Цена акций Rational AG.
- ***Debt\_Level* (D)** – Общий долг компании Rational AG.
- ***Risk\_Free\_Rate* (r)** – Безрисковая ставка (соответствующие спот-ставки однолетних государственных облигаций).

### Первая подзадача:

Столбцы ***Stock\_Price*** и ***Risk\_Free\_Rate*** содержат пропуски, которые требуется восстановить на основе соседних значений, при помощи кубической сплайн-интерполяции.

Результат первого этапа: полный набор данных в каждом столбце. Полученные результаты проверьте на разумность.



## Домашняя работа #3

### Вторая подзадача:

Использовать восстановленные данные для решения нелинейных уравнений и оценки вероятности дефолта компании (согласно модели Мертона) за период 2019-2021 гг. Для каждой отметки времени:

1. Зафиксируйте стоимость активов:  $A = E + D$  (переменные введены на предыдущем слайде).
2. Решите методом половинного деления уравнение, найдя  $\sigma_A$ :

$$E = A \cdot N(d_1) - D \cdot e^{-rT} \cdot N(d_2), \text{ где:}$$

$$d_1 = \frac{\ln(A/D) + (r - \sigma_A^2/2)T}{\sigma_A \sqrt{T}}, \quad d_2 = d_1 - \sigma_A \sqrt{T}$$

$N(\cdot)$  – функция нормального распределения, `scipy.stats.norm.cdf()`. Считайте  $T = 1$ .

Добавьте найденные значения в новый столбец датасета.

3. Рассчитайте вероятность дефолта (PD) для каждой компании с использованием откалиброванных параметров:

$$PD = N\left(-\frac{\ln(A/D) + (r - \sigma_A^2/2)T}{\sigma_A \sqrt{T}}\right)$$

Добавьте найденные значения в новый столбец датасета.



## Домашняя работа #3

### Третья подзадача:

1. Напишите отчёт о проделанной работе.
2. Постройте графики (на временной оси):
  - Цен акций Rational AG.
  - Общего долга компании Rational AG.
  - Безрисковых ставок.
  - Вероятности дефолта компании Rational AG.
3. Сравните суммарное время выполнения первых двух подзадач и отличие в расчётах при хранении данных в форматах np.float16, np.float32, np.float32. Постройте соответствующие графики. Оформите выводы сравнительного анализа.



# МЕТОДЫ ЛИНЕЙНОЙ АЛГЕБРЫ ДЛЯ МАТРИЦ

Методы линейной алгебры для плотных матриц (DLA, Dense Linear Algebra) играют ключевую роль в самых различных научных и инженерных приложениях:

### Линейные системы: Решение $\mathbf{Ax} = \mathbf{b}$

- Вычислительная электродинамика; материаловедение; приложения, использующие граничные интегральные уравнения; обтекание крыла воздушным потоком; течение жидкости вокруг и т.д.

### Метод наименьших квадратов: Найти $\mathbf{x}$ для минимизации $\|\mathbf{Ax} - \mathbf{b}\|$

- Вычислительная статистика (например, линейный метод наименьших квадратов), эконометрика; теория управления; обработка сигналов; подгонка кривых и т.д.

### Задачи на собственные значения: Решение $\mathbf{Ax} = \lambda \mathbf{x}$

- Вычислительная химия; квантовая механика; материаловедение; распознавание лиц; метод главных компонент (PCA); интеллектуальный анализ данных; маркетинг; алгоритм Google PageRank; спектральная кластеризация; вибрационный анализ; сжатие данных и т.д.

### Сингулярное разложение (SVD): $\mathbf{A} = \mathbf{U} \Sigma \mathbf{V}^*$ ( $\mathbf{Au} = \sigma \mathbf{v}$ и $\mathbf{A}^* \mathbf{v} = \sigma \mathbf{u}$ )

- Поиск информации; веб-поиск; обработка сигналов; анализ больших данных; аппроксимация матриц низкого ранга; минимизация методом наименьших квадратов; псевдообратная матрица и т.д.

### Множество вариаций в зависимости от структуры $\mathbf{A}$

- Матрица  $\mathbf{A}$  может быть: симметричной, положительно определённой, трёхдиагональной, Гессенберга, ленточной, разрежённой с плотными блоками и т.д.

### Важность методов для плотных матриц

- Методы линейной алгебры для плотных матриц имеют решающее значение для разработки решателей для разреженных систем.



<https://netlib.org/>

<https://icl.utk.edu/research>

### BLAS

Набор низкоуровневых процедур для выполнения базовых операций линейной алгебры, таких как умножение векторов и матриц.

### LAPACK

Библиотека для решения систем линейных уравнений, задач на собственные значения и других задач линейной алгебры на плотных матрицах.

### ScaLAPACK

Версия LAPACK, для распределённых вычислительных систем, предназначенная для работы с большими плотными матрицами на кластерах.

### PLASMA

Разработана для выполнения операций плотной линейной алгебры на многоядерных процессорах. Включает реализации таких задач, как LU-разложение, QR-разложение, решение систем линейных уравнений и вычисление собственных значений.

### MAGMA

Предоставляет оптимизированные реализации ключевых алгоритмов, таких как разложение матриц и решение систем уравнений, а также поддерживает гибридные вычисления, комбинируя мощности CPU и GPU.

### SLATE

Ориентирована на экзафлопсные вычисления, разработана с учётом высокой масштабируемости для экстремально больших систем.

Современное программное обеспечение для многоядерных процессоров и ускорителей.

Поддержка со стороны

ECP SLATE, CEED, PEEKS, xSDK



BLAS (Basic Linear Algebra Subroutines) – библиотека низкоуровневых процедур, предназначенных для выполнения основных операций линейной алгебры. Она делится на три уровня, каждый из которых отвечает за определённый тип задач:

### Уровень 1 BLAS

$$y = \alpha x + \beta y$$

#### Объём данных и вычислений:

- Данные:  $O(n)$  – линейная зависимость от длины вектора.
- Операции:  $O(n)$  – линейное количество операций с плавающей точкой (flops).

**Ограничение по памяти:** Производительность определяется скоростью доступа к памяти. На каждое обращение к данным приходится  $O(1)$  вычислительных операций, что делает узким местом пропускную способность памяти, а не вычислительную мощность процессора.

### Уровень 2 BLAS

$$y = \alpha A x + \beta y$$

#### Объём данных и вычислений:

- Данные:  $O(n^2)$  – квадратичная зависимость от размера матрицы.
- Операции:  $O(n^2)$  – квадратичное количество операций с плавающей точкой.

**Ограничение по памяти:** Производительность зависит от скорости доступа к памяти, так как на каждое обращение к данным приходится  $O(1)$  вычислений. Это означает, что, как и в уровне 1, ключевым фактором является пропускная способность памяти.

### Уровень 3 BLAS

$$C = \alpha A B + \beta C$$

#### Объём данных и вычислений:

- Данные:  $O(n^2)$  — квадратичная зависимость от размера матриц.
- Операции:  $O(n^3)$  — кубическое количество операций с плавающей точкой.

**Эффект "поверхность-к-объёму":** Количество вычислений  $O(n^3)$  растёт быстрее, чем объём данных  $O(n^2)$ , что напоминает отношение поверхности к объёму в геометрии. Это повышает эффективность использования вычислительных ресурсов.

**Ограничение по вычислениям:** На каждое обращение к памяти приходится  $O(n)$  операций, что делает эти задачи интенсивными с точки зрения вычислений. Здесь узким местом становится вычислительная мощность процессора, а не доступ к памяти.

Принцип локальности – основан на том, что программы часто обращаются:

- к данным, расположенным рядом друг с другом (**пространственная локальность**),
- к одним и тем же данным многократно (**временная локальность**).

Цели оптимизации:

- **Максимум памяти по минимальной цене:** предоставить пользователю как можно больше памяти, используя наиболее доступные и дешёвые (оперативную память вместо кэша).
- **Максимальная скорость доступа:** Обеспечить доступ к данным на скорости самых быстрых технологий (кэш-память или ЦПУ).

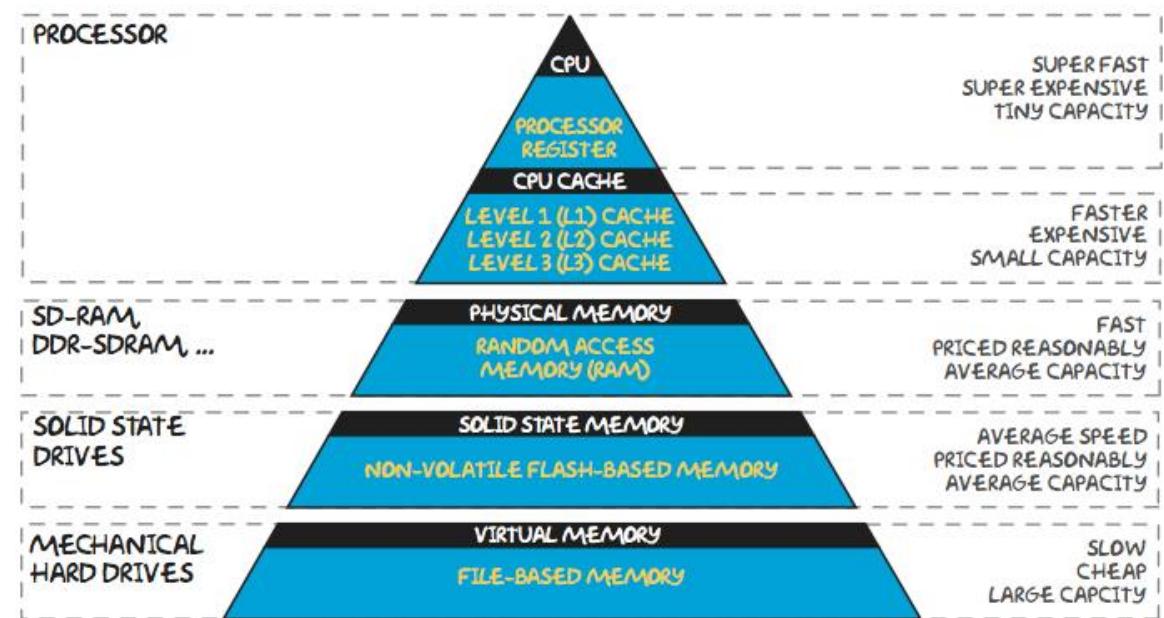
Вычисления (арифметические операции) можно выполнять только над данными, которые находятся на вершине иерархии памяти – в самых быстрых её уровнях, таких как ЦПУ или кэш процессора.

Выполнения большого числа вычислений на ограниченном объёме данных.

Увеличения **соотношения вычислений к обращениям к памяти** (flops per memory references). Чем больше операций с плавающей точкой (flops) приходится на одно обращение к памяти, тем лучше используется быстрая память и тем выше производительность.

BLAS	Обращения к памяти	Операции (flops)	Соотношение flops / обращения к памяти
<b>Уровень 1</b> Векторные операции	$3n$	$2n$	$2/3$
<b>Уровень 2</b> Матрично-векторные операции	$n^2$	$2n^2$	2
<b>Уровень 3</b> Матрично-матричные операции	$4n^2$	$2n^3$	$n/2$

BLAS более высокого уровня (особенно уровень 3) оптимизируют использование быстрой памяти, увеличивая число вычислений на каждое обращение к данным. Это делает их особенно эффективными для современных вычислительных систем, где иерархия памяти играет решающую роль в достижении высокой производительности.



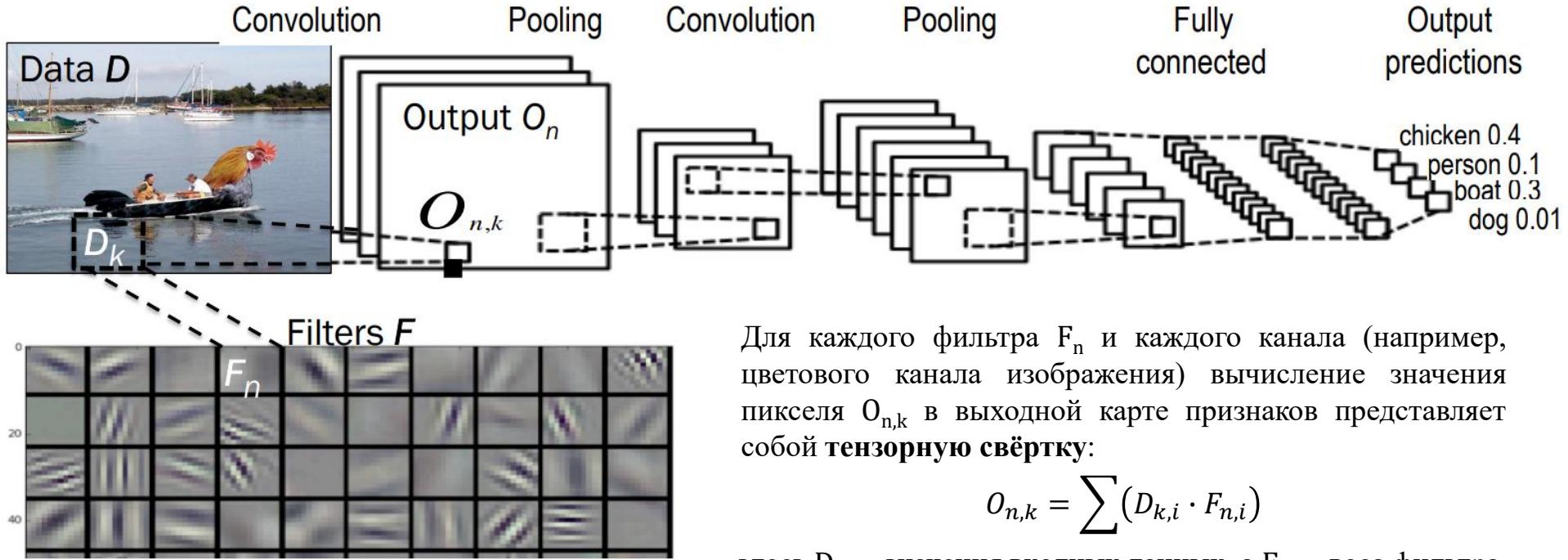
Пример скорости работы алгоритма по перемножению матриц при решении задачи классификации изображений:

Метод	Сложность времени	Ожидаемое время (с) для батча 100
NumPy (BLAS)	$O(n^3)$	~0.1
Наивное умножение	$O(n^3)$	~14.0
Блочное умножение	$O(n^3)$	~0.2 – 2.0
Алгоритм Штассена	$O(n^{2.807\dots})$	~0.4 – 1.3
Алгоритм Coppersmith-Winograd'a	$O(n^{2.37\dots})$	~0.3 – 1.0



MAGMA Batched – поддержка операций с пакетными и тензорными свёртками

Свёрточные нейронные сети (CNN), широко используются в задачах компьютерного зрения, таких как распознавание изображений.



Для каждого фильтра  $F_n$  и каждого канала (например, цветового канала изображения) вычисление значения пикселя  $O_{n,k}$  в выходной карте признаков представляет собой **тензорную свёртку**:

$$O_{n,k} = \sum (D_{k,i} \cdot F_{n,i})$$

здесь  $D_{k,i}$  – значения входных данных, а  $F_{n,i}$  – веса фильтра.

- Процесс обладает высокой степенью параллелизма, что позволяет выполнять множество операций одновременно.
- Операции мелкие, и их эффективность значительно возрастает при выполнении в пакетном режиме (batched).
- Благодаря реорганизации данных (data reshape) свёрточные операции можно преобразовать в пакетное матричное умножение (batched GEMM — General Matrix Multiplication). Это повышает производительность вычислений.
- Такой подход является одним из многих способов оптимизации обработки данных в нейронных сетях.



Таблица с характеристиками нескольких современных процессоров и графических ускорителей, иллюстрирует различия в архитектурах.

Memory hierarchies	Intel Haswell E5-2650 v3	Intel KNL 7250 DDR5   MCDRAM	ARM Cortex A57	Nvidia P100	Nvidia V100
	10 cores 368 Gflop/s 105 Watts	68 cores 2662 Gflop/s 215 Watts	4 cores 32 Gflop/s 7 Watts	56 SM 64 cores 4700 Gflop/s 250 Watts	80 SM 64 cores 7500 Gflop/s 300 Watts
REGISTERS	16/core AVX2	32/core AVX-512	32/core	256 KB/SM	256 KB/SM
L1 CACHE & GPU SHARED MEMORY	32 KB/core	32 KB/core	32 KB/core	64 KB/SM	96 KB/SM
L2 CACHE	256 KB/core	1024 KB/2cores	2 MB	4 MB	6 MB
L3 CACHE	25 MB	0...16 GB	N/A	N/A	N/A
MAIN MEMORY	64 GB	384   16 GB	4 GB	16 GB	16 GB
MAIN MEMORY BW	68 GB/s 5.4 flops/byte	115   421 GB/s 23   6 Flops/byte	26 GB/s 1.2 flops/byte	720 GB/s 6.5 flops/byte	900 GB/s 8.3 flops/byte
PCI EXPRESS GEN3x16 NVLINK	16 GB/s 23 flops/byte	16 GB/s 166 flops/byte	16 GB/s 2 flops/byte	16 GB/s 294 flops/byte	300 GB/s (NVL) 25 flops/byte
INTERCONNECT INFINIBAND EDR	12 GB/s 30 flops/byte	12 GB/s 221 flops/byte	12 GB/s 2.6 flops/byte	12 GB/s 392 flops/byte	12 GB/s 625 flops/byte
Memory hierarchies for different type of architectures Flops per byte transfer (all flop rates for 64 bit operands)					

Все значения производительности (флопс) указаны для 64-битных операций с плавающей точкой.



# МЕТОДЫ РЕШЕНИЯ ЗАДАЧ НА СОБСТВЕННЫЕ ЗНАЧЕНИЯ И СОБСТВЕННЫЕ ВЕКТОРЫ МАТРИЦ



Пусть  $A$  — действительная числовая квадратная матрица размеров  $(n \times n)$ . Ненулевой вектор  $X = (x_1, \dots, x_n)^T$  размеров  $(n \times 1)$ , удовлетворяющий условию

$$AX = \lambda X$$

называется собственным вектором матрицы  $A$ .

Число  $\lambda$  в равенстве называется собственным значением. Говорят, что собственный вектор  $X$  соответствует (принадлежит) собственному значению  $\lambda$ .

Последнее равенство равносильно однородной относительно  $X$  системе:

$$(A - \lambda E) X = 0, (X \neq 0).$$

Система имеет ненулевое решение для вектора  $X$  (при известном  $\lambda$ ) при условии  $|A - \lambda E| = 0$ . Это равенство есть характеристическое уравнение:

$$|A - \lambda E| = P_n(\lambda) = 0.$$



$$|A - \lambda E| = P_n(\lambda) = 0.$$

где  $P_n()$  — характеристический многочлен  $n$ -й степени. Корни  $\lambda_1, \lambda_2, \dots, \lambda_n$  характеристического уравнения являются собственными (характеристическими) значениями матрицы  $A$ , а соответствующие каждому собственному значению  $\lambda_i$ ,  $i = 1, \dots, n$ , ненулевые векторы  $X_i$ , удовлетворяющие системе

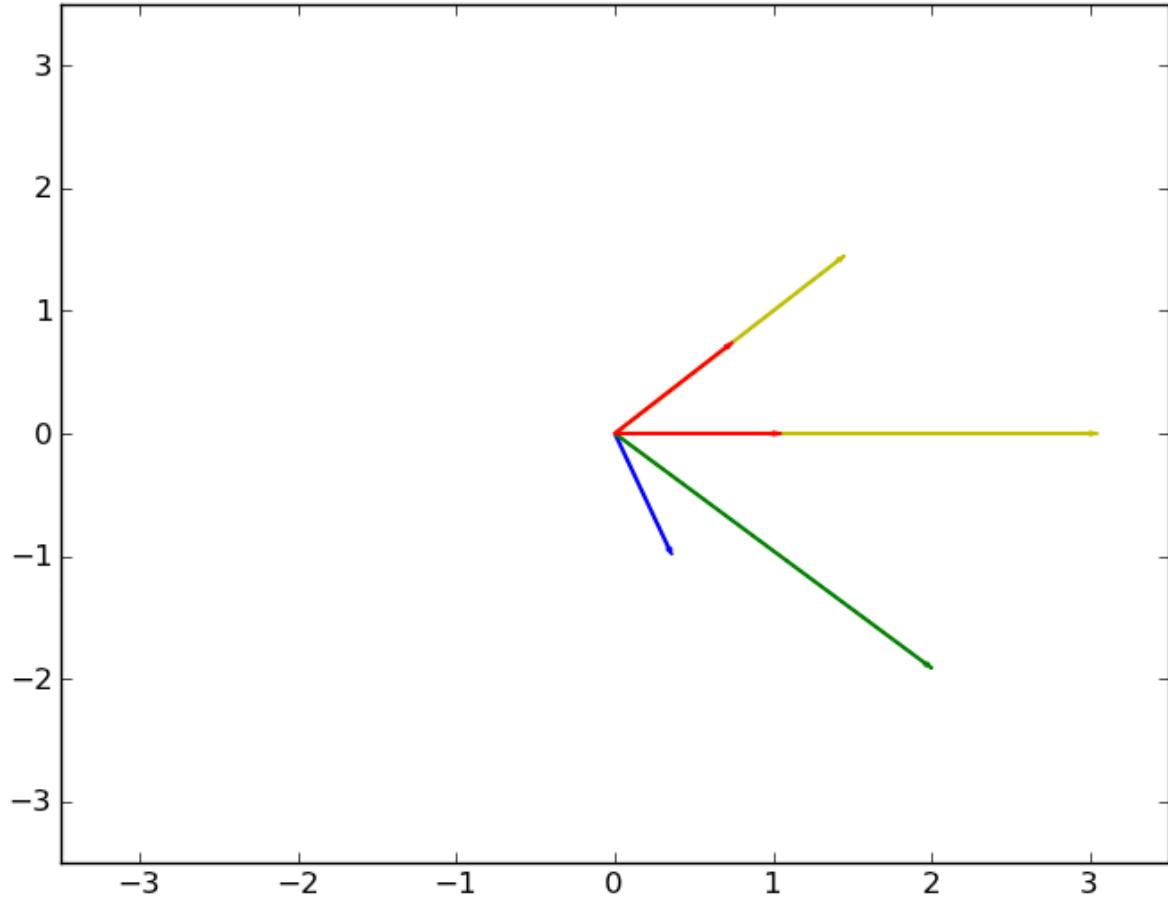
$$AX^i = \lambda_i X^i \text{ или } (A - \lambda_i E) X^i = 0, i = 1, 2, \dots, n,$$

являются собственными векторами.

Требуется найти собственные значения и собственные векторы заданной матрицы. Поставленная задача часто называется второй *задачей линейной алгебры*.



$$A = \begin{pmatrix} 3 & -1 \\ 0 & 2 \end{pmatrix}$$



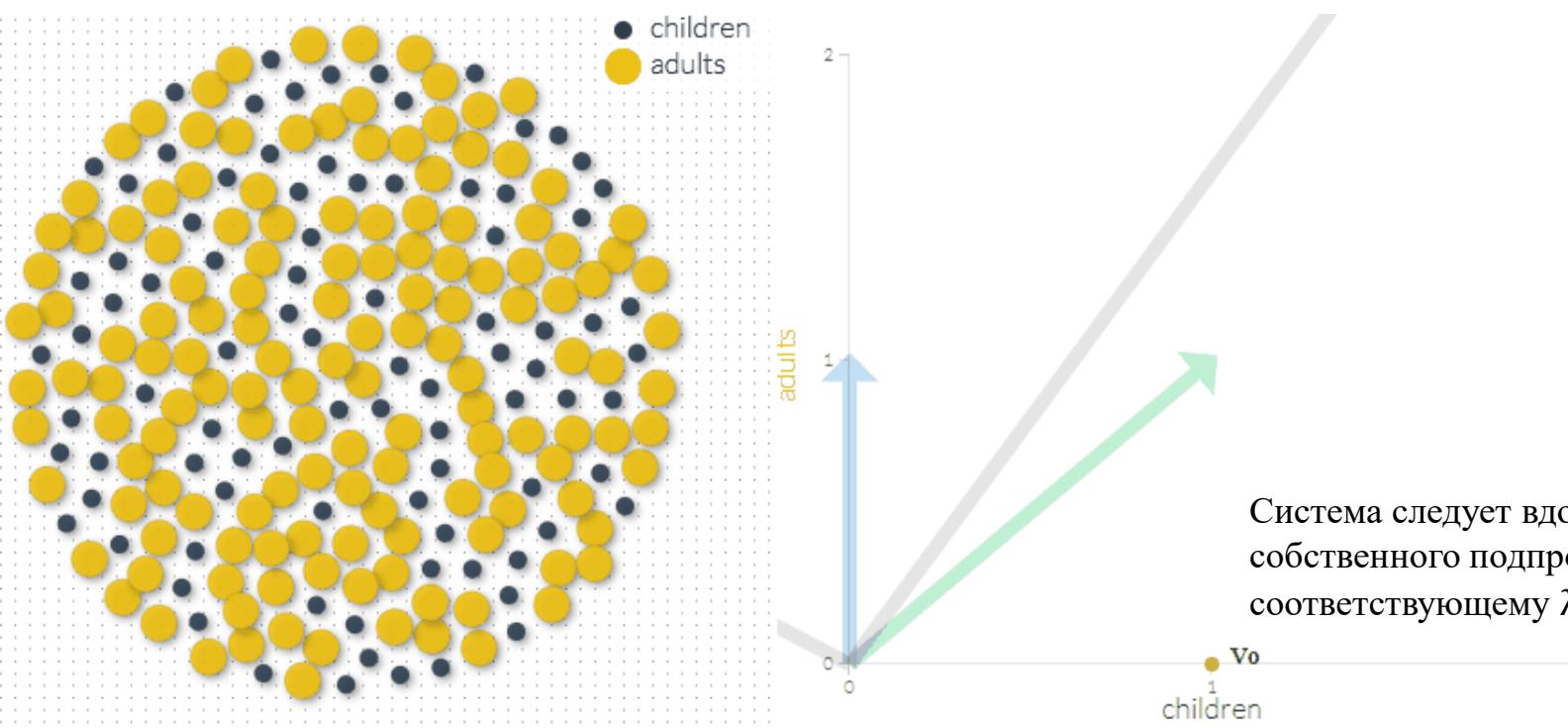


Рассмотрим тривиальную популяционную модель, аналогичную модели Мальтуса:

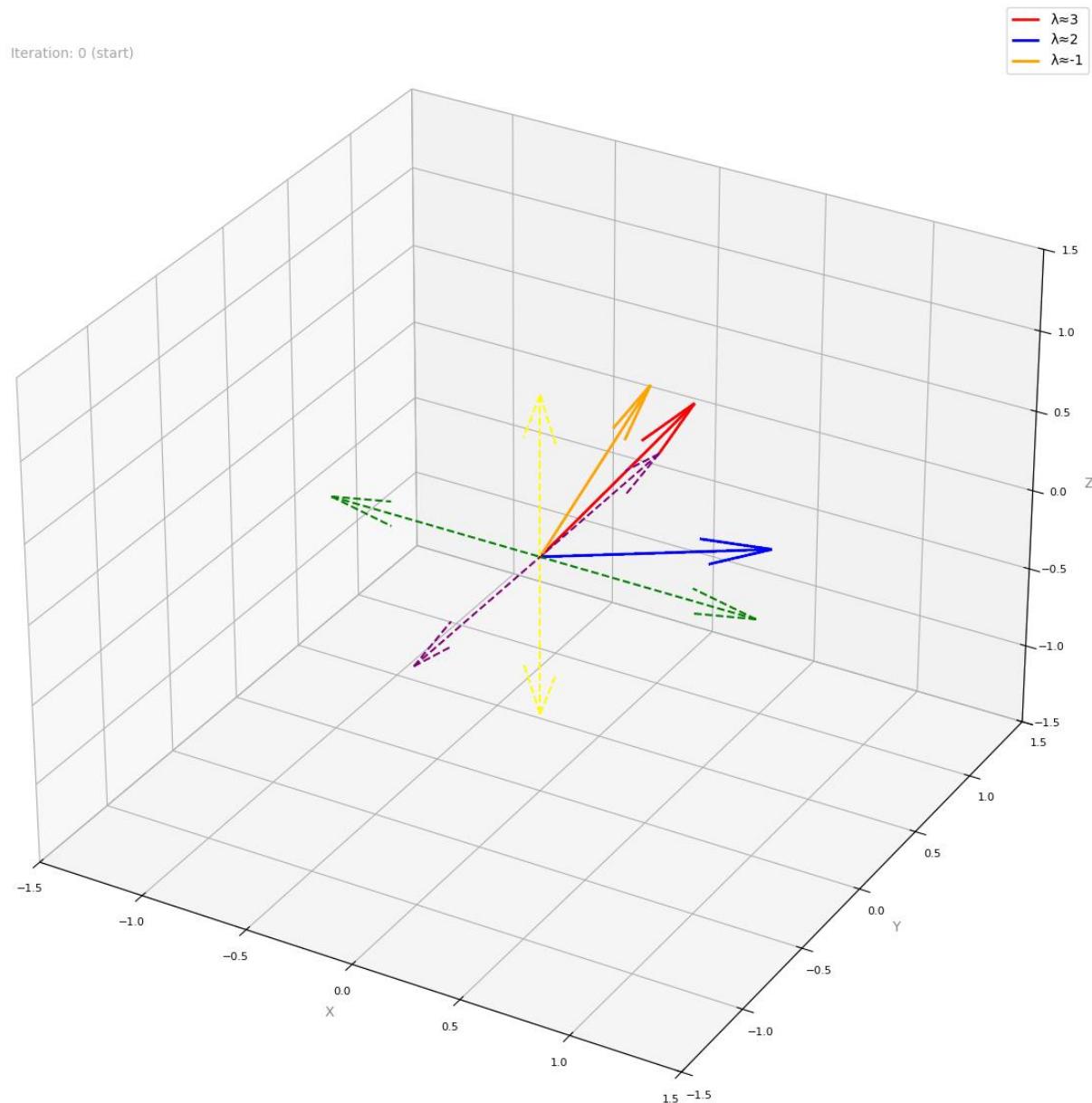
$$\begin{cases} \text{родители}_{t+1} = \text{родители}_t + \text{дети}_t \\ \text{дети}_{t+1} = \text{родители}_t \end{cases}$$

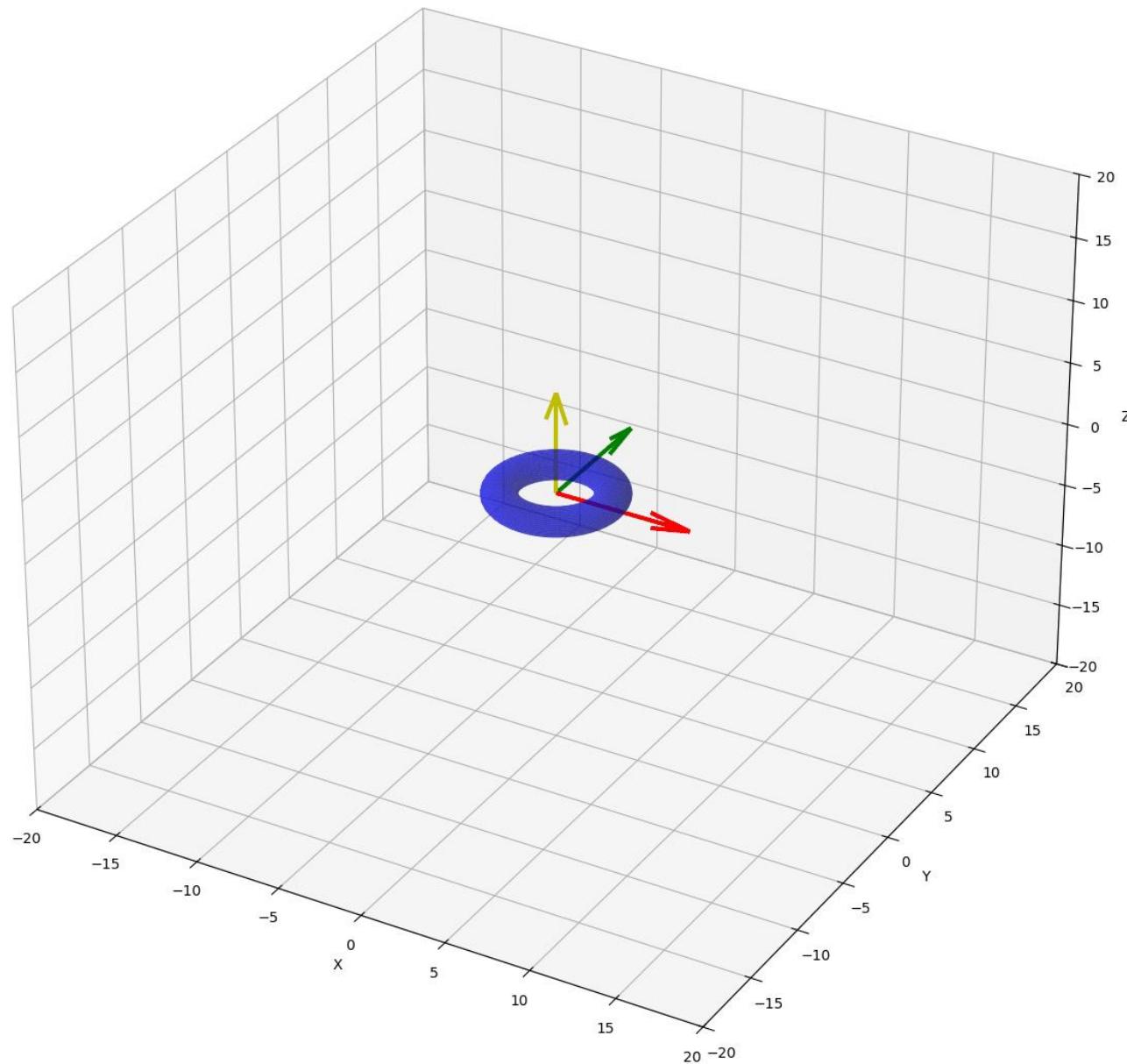
Перепишем:

$$\begin{pmatrix} \text{родители}_{t+1} \\ \text{дети}_{t+1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} \text{родители}_t \\ \text{дети}_t \end{pmatrix}$$



Система следует вдоль серой линии – собственного подпространства, соответствующему  $\lambda = (1 + \sqrt{5})/2 > 1$







Для динамических систем с матрицей  $A$ , спектр может много сообщить о поведении системы (например, о её устойчивости).

Однако для *не нормальных матриц*, спектр может быть неустойчивым относительно малых возмущений матрицы.

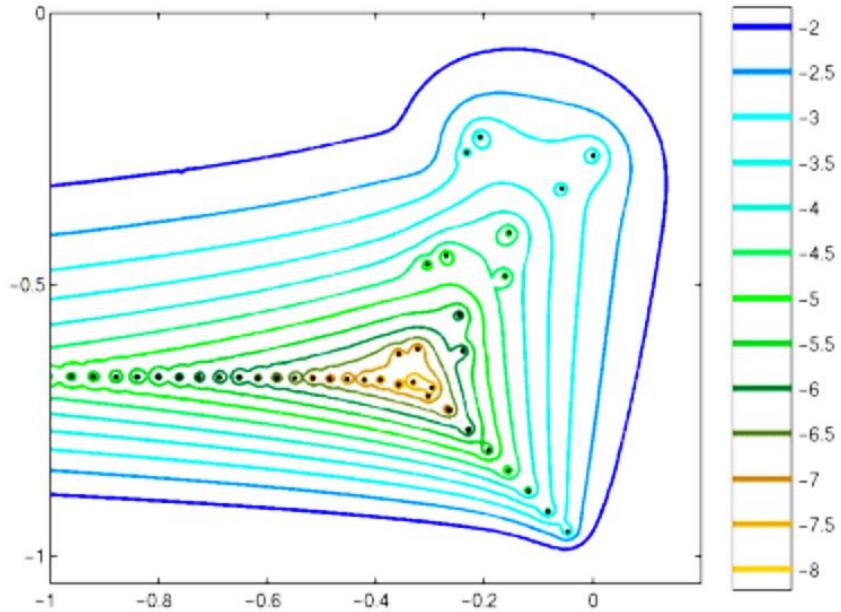
Для измерения подобных возмущений было разработана концепция **псевдоспектра**.

**Теорема:**  $A$  – **нормальная матрица**, тогда и только тогда, когда  $A = U\Lambda U^*$ , где  $U$  унитарна и  $\Lambda$  диагональна.

Любая нормальная матрица – **унитарно диагонализуема**. Это означает, что она может быть приведена к диагональному виду с помощью унитарной матрицы  $U$ . Другими словами, каждая нормальная матрица имеет ортогональный базис из собственных векторов.



- Для динамических систем с матрицей  $A$  спектр может много сообщить о поведении системы (например, о её устойчивости)
- Однако для не нормальных матриц, спектр может быть неустойчивым относительно малых возмущений матрицы
- Для измерения подобных возмущений было разработана концепция псевдоспектра.



Рассмотрим объединение всех возможных собственных значений для всевозможных возмущений матрицы  $A$ .

$$\Lambda_\epsilon(A) = \{\lambda \in \mathbb{C} : \exists E, x \neq 0 : (A + E)x = \lambda x, \quad \|E\|_2 \leq \epsilon. \}$$

Для малых  $E$  и нормальных  $A$  это круги вокруг собственных значений, для не нормальных матриц, структура может сильно отличаться.



Различают *полную* и *частичную* проблему собственных значений, когда необходимо найти весь спектр (все собственные значения) и собственные векторы либо часть спектра, например:  $\rho(A) = \max_i |\lambda_i(A)|$  и  $\min_i |\lambda_i(A)|$ . Величина  $(A)$  называется *спектральным радиусом*.

1. Если для собственного значения  $i$  найден собственный вектор  $X_i$ , то вектор  $\mu X_i$ , где  $\mu$  — произвольное число, также является собственным вектором, соответствующим этому же собственному значению  $i$ .
2. Попарно различным собственным значениям соответствуют линейно независимые собственные векторы;  $k$ -кратному корню характеристического уравнения соответствует не более  $k$  линейно независимых собственных векторов.
3. Симметрическая матрица имеет полный спектр  $\lambda_i$ ,  $i = \overline{1, n}$ , действительных собственных значений;  $k$ -кратному корню характеристического уравнения симметрической матрицы соответствует ровно  $k$  линейно независимых собственных векторов.
4. Положительно определенная симметрическая матрица имеет полный спектр действительных положительных собственных значений.

Задача состоит в ранжировании веб-страницы: какие из них являются важными, а какие нет.

В интернете страницы ссылаются друг на друга.

PageRank определяется рекурсивно. Обозначим за  $p_i$  важность  $i$ -ой страницы. Тогда определим эту важность как усреднённую важность всех страниц, которые ссылаются на данную страницу. Это определение приводит к следующей линейной системе:

$$p_i = \sum_{j \in N(i)} \frac{p_j}{L(j)}$$



$$p_i = \sum_{j \in N(i)} \frac{p_j}{L(j)}$$

где  $L(j)$  – число исходящих ссылок с  $j$ -ой страницы,  $N(i)$  – число соседей  $i$ -ой страницы. Это может быть записано следующим образом:

$$p = Gp, \quad G_{ij} = \frac{1}{L(j)}$$

то есть мы уже знаем, что у матрицы  $G$  есть собственное значение равное 1. Заметим, что  $G$  – левостохастичная матрица, то есть сумма в каждом столбце равна 1.

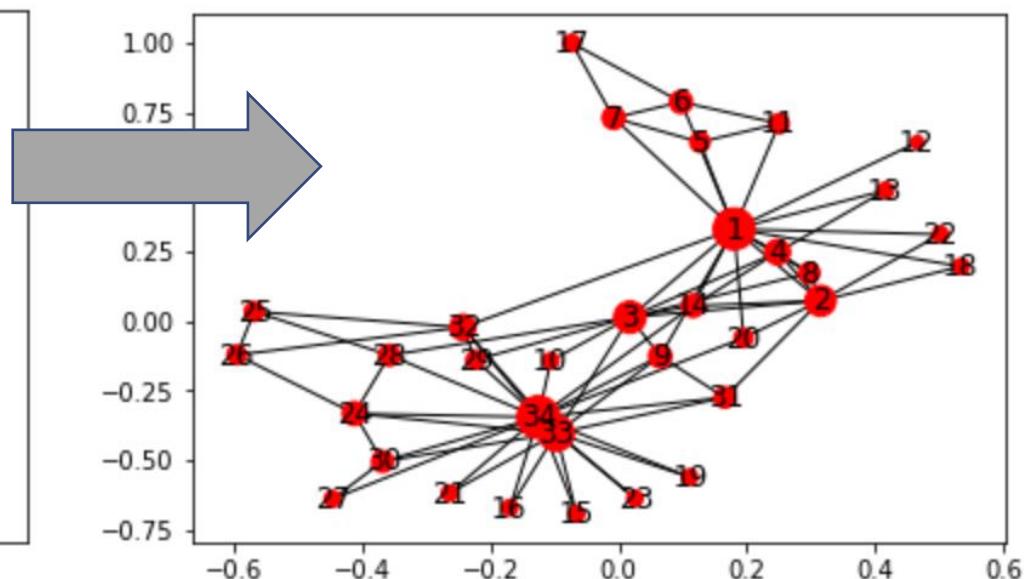
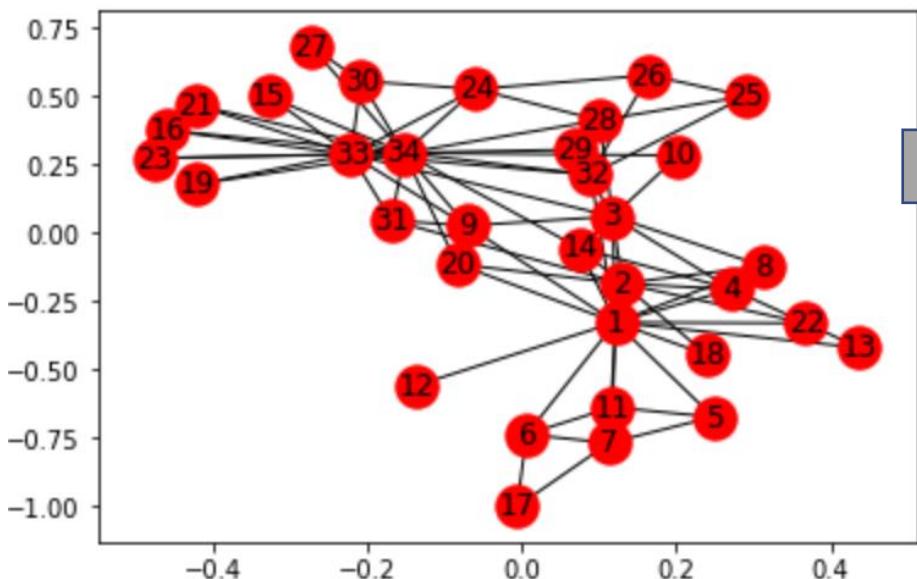


Пример:

```
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx

nt = nx.read_gml('network.gml')
nx.draw_networkx(nt)

pr = nx.algorithms.link_analysis.pagerank(nt)
pr_vector = list(pr.values())
pr_vector = np.array(pr_vector) * 3000
nx.draw_networkx(nt, node_size=pr_vector, labels=None)
```





Полную проблему собственных значений для матриц невысокого порядка ( $n \leq 10$ ) можно решить методом непосредственного развертывания. В этом случае будем иметь

$$|A - \lambda E| = \begin{vmatrix} a_{11} - \lambda & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} - \lambda & a_{23} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} - \lambda \end{vmatrix} = P_n(\lambda) = 0$$

Уравнение  $P_n(\lambda) = 0$  является нелинейным. Его решение дает  $n$ , вообще говоря, комплексных собственных значений  $\lambda_1, \lambda_2, \lambda_3, \dots, \lambda_n$ , при которых  $P_n(\lambda_i) = 0, i = 1,.., n$ . Для каждого  $\lambda_i$  может быть найдено решение однородной системы  $(A - \lambda_i E)X^i = 0, i = 1,.., n$ . Эти решения  $X^i$ , определенные с точностью до произвольной константы, образуют систему  $n$ , вообще говоря, различных векторов  $n$ -мерного пространства. В некоторых задачах несколько этих векторов (или все) могут совпадать.



## Методика решения задачи

1. Для заданной матрицы  $A$  составить характеристическое уравнение:

$$|A - E| = 0.$$

2. Решить характеристическое уравнение и найти собственные значения  $\lambda_1, \lambda_2, \lambda_3, \dots, \lambda_n$ .

3. Для каждого собственного значения составить систему (2.4):

$$(A - \lambda_i E) X^i = 0, i = 1, 2, \dots, n,$$

и найти собственные векторы  $X^i$ .

*Замечание.* Каждому собственному значению соответствует один или несколько векторов. Поскольку определитель  $|A - \lambda_i E|$  системы равен нулю, то ранг матрицы системы меньше числа неизвестных:  $\text{rang}(A - \lambda_i E) = r < n$  и в системе имеется ровно  $r$  независимых уравнений, а  $(n - r)$  уравнений являются зависимыми. Для нахождения решения системы следует выбрать  $r$  уравнений с  $r$  неизвестными так, чтобы определитель составленной системы был отличен от нуля. Остальные  $(n - r)$  неизвестных следует перенести в правую часть и считать параметрами. Придавая параметрам различные значения, можно получить различные решения системы. Для простоты, как правило, попеременно полагают значение одного параметра равным 1, а остальные равными 0.



- Часто в вычислительной практике требуется найти не весь спектр, а только некоторую его часть, например самое большое или самое маленькое собственные значения.
- Также отметим, что для Эрмитовых матриц собственные значения всегда действительны.
- Степенной метод – простейший метод вычисления максимального по модулю собственного значения.



## Задача на собственные значения

$$Ax = \lambda x, \quad \|x\|_2 = 1 \text{ для устойчивости}$$

может быть записана как итерации с неподвижной точкой, которые называются степенным методом и дают максимальное по модулю собственное значение матрицы A.

Степенной метод имеет вид

$$x_{k+1} = Ax_k, \quad x_{k+1} := \frac{x_{k+1}}{\|x_{k+1}\|_2}.$$

и  $x_{k+1} \rightarrow v_1$ , где  $Av_1 = \lambda_1 v_1$  и  $\lambda_1$  максимальное по модулю собственное значение, и  $v_1$  – соответствующий собственный вектор.

На  $(k + 1)$ -ой итерации приближение для  $\lambda_1$  может быть найдено следующим образом

$$\lambda^{(k+1)} = (Ax_{k+1}, x_{k+1}),$$

Заметим, что  $\lambda^{(k+1)}$  не требуется для  $(k + 2)$ -ой итерации, но может быть полезно для оценки ошибки на каждой итерации:

$$\|Ax_{k+1} - \lambda^{(k+1)}x_{k+1}\|$$



Метод сходится со скоростью геометрической прогрессии, с константой  $q = \left| \frac{\lambda_2}{\lambda_1} \right| < 1$ , где  $\lambda_1 > \lambda_2 \geq \dots \geq \lambda_n$ .

Это означает, что сходимость может быть сколь угодно медленной при близких значениях  $\lambda_1$  и  $\lambda_2$ .



## Анализ сходимости для $A = A^*$

- Рассмотрим степенной метод более подробно для случая эрмитовой матрицы
- Через несколько слайдов вы увидите, что любая эрмитова матрица диагонализуема, поэтому существует ортонормированный базис из собственных векторов  $v_1, \dots, v_n$  такой что  $A v_i = \lambda_i v_i$ .
- Разложим  $x_0$  в этом базисе с коэффициентами  $c_i$ :

$$x_0 = c_1 v_1 + \dots + c_n v_n.$$

- Поскольку  $v_i$  – собственные векторы, выполнены следующие равенства



- Поскольку  $v_i$  – собственные векторы, выполнены следующие равенства

$$x_1 = \frac{Ax_0}{\|Ax_0\|} = \frac{c_1\lambda_1 v_1 + \cdots + c_n\lambda_n v_n}{\|c_1\lambda_1 v_1 + \cdots + c_n\lambda_n v_n\|}$$

⋮

$$x_k = \frac{Ax_{k-1}}{\|Ax_{k-1}\|} = \frac{c_1\lambda_1^k v_1 + \cdots + c_n\lambda_n^k v_n}{\|c_1\lambda_1^k v_1 + \cdots + c_n\lambda_n^k v_n\|}$$

- Получаем следующее выражение

$$x_k = \frac{c_1}{|c_1|} \left( \frac{\lambda_1}{|\lambda_1|} \right)^k \frac{v_1 + \frac{c_2}{c_1} \frac{\lambda_2^k}{\lambda_1^k} v_2 + \cdots + \frac{c_n}{c_1} \frac{\lambda_n^k}{\lambda_1^k} v_n}{\left\| v_1 + \frac{c_2}{c_1} \frac{\lambda_2^k}{\lambda_1^k} v_2 + \cdots + \frac{c_n}{c_1} \frac{\lambda_n^k}{\lambda_1^k} v_n \right\|},$$

которое сходится к  $v_1$  при  $\left| \frac{c_1}{|c_1|} \left( \frac{\lambda_1}{|\lambda_1|} \right)^k \right| = 1$  и  $\left( \frac{\lambda_2}{\lambda_1} \right)^k \rightarrow 0$  если  $|\lambda_2| < |\lambda_1|$ .

- Степенной метод даёт оценку для максимального по модулю собственного числа или спектрального радиуса матрицы
- Одна итерация требует одного умножения матрицы на вектор. Если можно умножить вектор на матрицу за  $O(n)$  (например, она разреженная), тогда степенной метод можно использовать для больших  $n$
- Сходимость может быть медленной
- Для грубой оценки максимального по модулю собственного значения и соответствующего вектора достаточно небольшого числа итераций
- Вектор решения лежит в Крыловском подпространстве  $\{x_0, Ax_0, \dots, A^k x_0\}$  и имеет вид  $\mu A^k x_0$ , где  $\mu$  – нормировочная постоянная.



Для решения частичной проблемы собственных значений и собственных векторов в практических расчетах часто используется метод итераций. На его основе можно определить приближенно собственные значения матрицы  $A$  и *спектральный радиус*  $\rho(A) = \max |\lambda_i(A)|$ .

Пусть матрица  $A$  имеет  $n$  линейно независимых собственных векторов

$X^i, i = 1,.., n$  и собственные значения матрицы  $A$  таковы, что:

$$\rho(A) = |\lambda_1(A)| > |\lambda_2(A)| \geq \dots \geq |\lambda_n(A)|$$



## Методика решения задачи

1. Выбрать произвольное начальное (нулевое) приближение собственного вектора  $X^{1(0)}$  (второй индекс в скобках здесь и ниже указывает номер приближения, а первый индекс без скобок соответствует номеру собственного значения). Положить  $k = 0$ .

2. Найти  $X^{1(1)} = AX^{1(0)}$ ,  $\lambda_1^{(1)} = \frac{x_i^{1(1)}}{x_i^{1(0)}}$ , где  $i$  — любой номер  $1 \leq i \leq n$ , и положить  $k = 1$ .

3. Вычислить  $X^{1(k+1)} = AX^{1(k)}$ .

4. Найти  $\lambda_1^{(k+1)} = \frac{x_i^{1(k+1)}}{x_i^{1(k)}}$ , где  $x_i^{1(k+1)}$ ,  $x_i^{1(k)}$  — соответствующие координаты векторов  $X^{1(k+1)}$  и  $\underline{X^{1(k)}}$ . При этом может быть использована любая координата с номером  $i$ ,  $i = 1, n$ .

5. Если  $\Delta = |\lambda_1^{(k+1)} - \lambda_1^{(k)}| \leq \varepsilon$ , процесс завершить и положить  $\lambda_1 \cong \lambda_1^{(k+1)}$ . Если  $\Delta > \varepsilon$ , положить  $k = k + 1$  и перейти к п. 3.

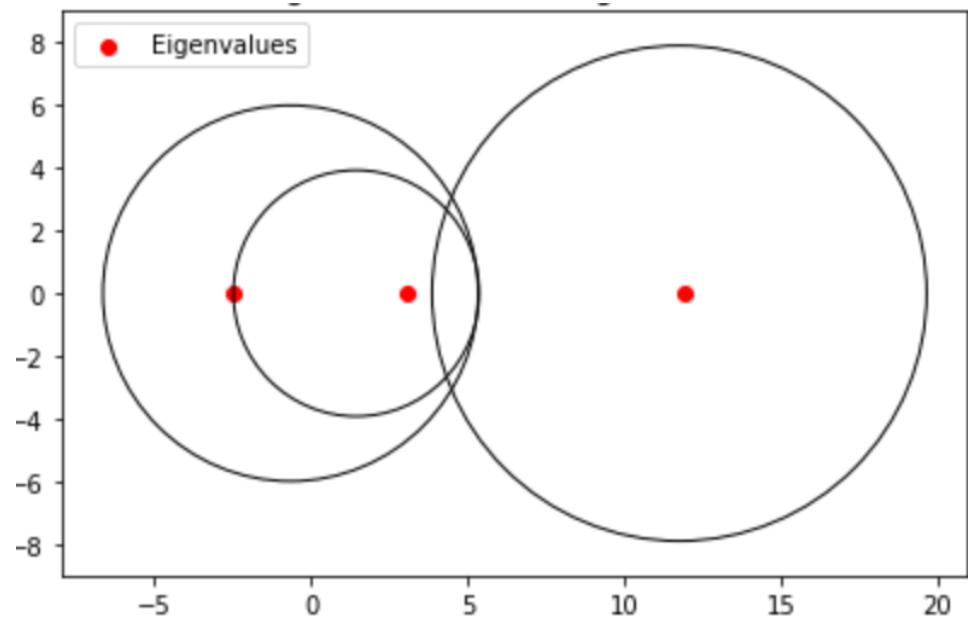


Есть теорема, которая часто помогает локализовать собственные значения. Она называется теоремой Гершгорина.

- Теорема утверждает, что все собственные значения  $\lambda_i$ ,  $i = 1, 2, \dots, n$  находятся внутри объединения кругов Гершгорина  $C_i$ , где  $C_i$  – окружность на комплексной плоскости с центром в  $a_{ii}$  и радиусом

$$r_i = \sum_{j \neq i} |a_{ij}|$$

Более того, если круги не пересекаются, то они содержат по одному собственному значению внутри каждого круга.





Сначала покажем, что если матрица  $A$  обладает строгим диагональным преобладанием, то есть

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|,$$

тогда такая матрица невырождена.

Разделим диагональную и недиагональную часть и получим

$$A = D + S = D(I + D^{-1}S),$$

где  $\|D^{-1}S\|_1 < 1$ . Поэтому, в силу теоремы о ряде Неймана, матрица  $I + D^{-1}S$  обратима и, следовательно,  $A$  также обратима.

Докажем утверждение теоремы от противного:

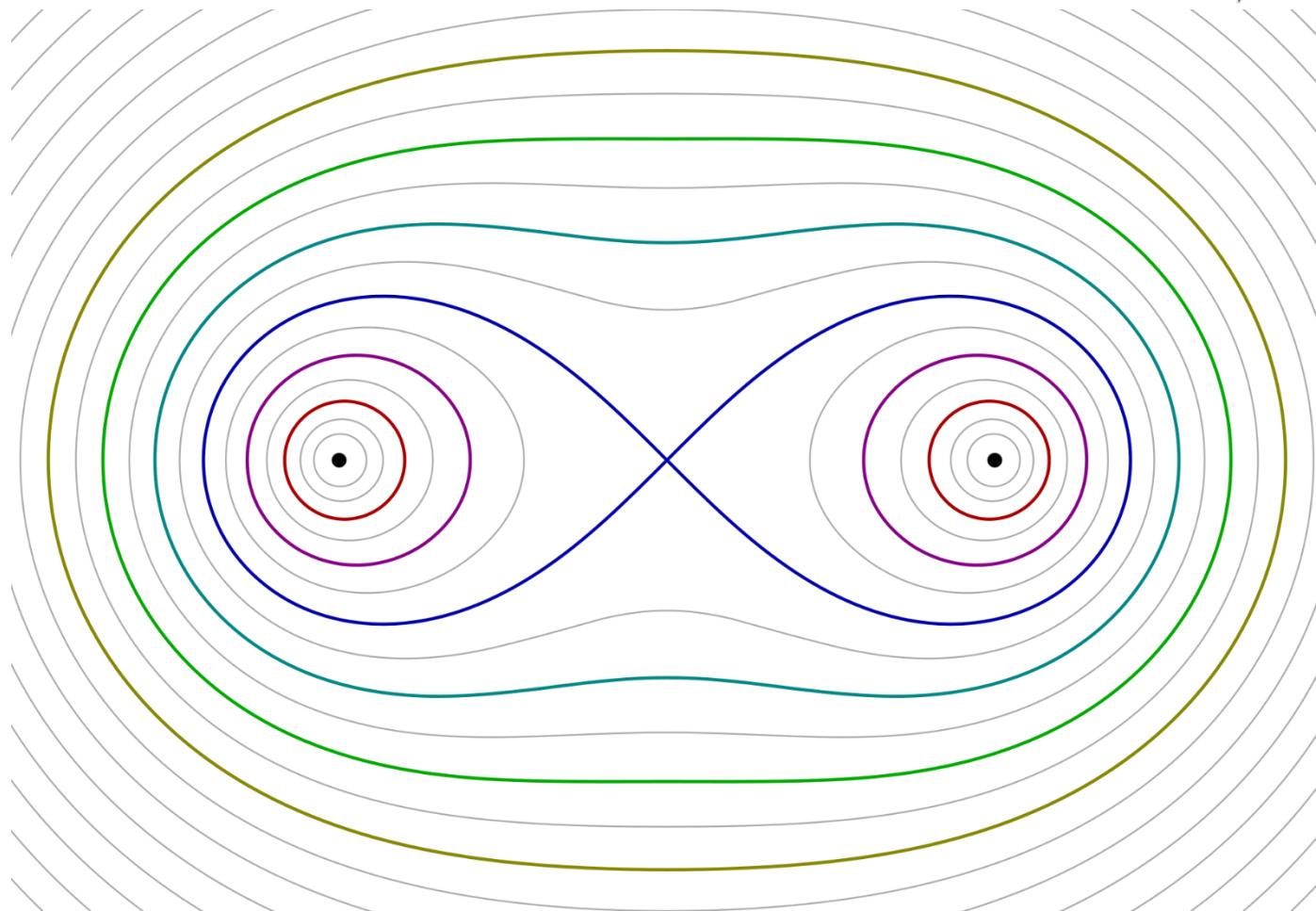
если любое из собственных чисел лежит вне всех кругов, то матрица  $(A - \lambda I)$  обладает свойством строгого диагонального преобладания поэтому она обратима

это означает, что если  $(A - \lambda I)x = 0$ , то  $x = 0$ .



- Существуют более сложные фигуры, под названием овалы Кассини, которые содержат спектр

$$M_{ij} = \{z \in \mathbb{C} : |a_{ii} - z| \cdot |a_{jj} - z| \leq r_i r_j\}, \quad r_i = \sum_{l \neq i} |a_{il}|.$$





Метод используется для решения полной проблемы собственных значений симметрической матрицы и основан на преобразовании подобия исходной матрицы  $A \in R^{n \times n}$  с помощью ортогональной матрицы  $H$ .

Две матрицы  $A$  и  $A^{(i)}$  называются *подобными* ( $A \sim A^{(i)}$  или  $A^{(i)} \sim A$ ), если  $A^{(i)} = H^{-1}AH$  или  $A = HA^{(i)}H^{-1}$ , где  $H$  — невырожденная матрица.

В методе вращений в качестве  $H$  берется *ортогональная матрица*, такая, что  $HH^T = H^TH = E$ , т. е.  $H^T = H^{-1}$ . В силу свойства ортогонального преобразования евклидова норма исходной матрицы  $A$  не меняется. Для преобразованной матрицы  $A^{(i)}$  сохраняется ее след и собственные значения  $\lambda_i$ :

$$\text{tr}A = \sum_{i=1}^n a_{ii} = \sum_{i=1}^n \lambda_i(A) = \text{tr}A^{(i)}.$$



При реализации метода вращений преобразование подобия применяется к исходной матрице  $A$  многократно:

$$A^{(k+1)} = (H^{(k)})^{-1} A^{(k)} H^{(k)} = (H^{(k)})^T A^{(k)} H^{(k)}, \quad k = 0, 1, \dots$$

Данная формула определяет итерационный процесс, где начальное приближение  $A^{(0)} = A$ . На каждой  $k$ -й итерации для некоторого выбираемого при решении задачи недиагонального элемента  $a_{ij}^{(k)}$ ,  $i \neq j$ , определяется ортогональная матрица  $H(k)$ , приводящая этот элемент  $a_{ij}^{(k+1)}$  (а также и  $a_{ji}^{(k+1)}$ ) к нулю. При этом на каждой итерации в качестве  $a_{ij}^{(k)}$  выбирается наибольший по модулю.



На каждой  $k$ -й итерации для некоторого выбираемого при решении задачи недиагонального элемента  $a_{ij}^{(k)}$ ,  $i \neq j$ , определяется ортогональная матрица  $H(k)$ , приводящая этот элемент  $a_{ij}^{(k+1)}$  (а также и  $a_{ji}^{(k+1)}$ ) к нулю. При этом на каждой итерации в качестве  $a_{ij}^{(k)}$  выбирается наибольший по модулю. Матрица  $H^{(k)}$ , называемая *матрицей вращения Якоби*, зависит от угла  $\varphi^{(k)}$  и имеет вид

$$H^{(k)} = \begin{pmatrix} 1 & \cdots & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & \cos \varphi^{(k)} & 0 & \cdots & 0 & -\sin \varphi^{(k)} & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & 1 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & 0 & 0 & \cdots & 1 & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & \sin \varphi^{(k)} & 0 & \cdots & 0 & \cos \varphi^{(k)} & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 1 \end{pmatrix} \begin{array}{l} \leftarrow i\text{-я строка} \\ \leftarrow j\text{-я строка} \end{array}$$

$\uparrow \qquad \qquad \qquad \uparrow$

$i\text{-й столбец} \qquad \qquad \qquad j\text{-й столбец}$



В данной ортогональной матрице элементы на главной диагонали единичные, кроме  $h_{ii}^{(k)} = \cos\varphi^{(k)}$  и  $h_{jj}^{(k)} = \cos\varphi^{(k)}$ , а  $h_{ij}^{(k)} = -\sin\varphi^{(k)}$ ,  $h_{ji}^{(k)} = \sin\varphi^{(k)}$  ( $h_{ij}$  — элементы матрицы  $H$ ).

Угол поворота  $\varphi^{(k)}$  определяется по формуле

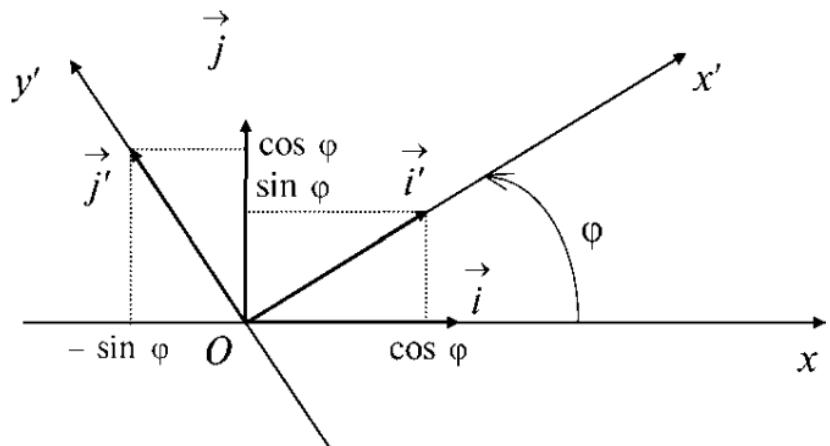
$$\operatorname{tg} 2\varphi^{(k)} = \frac{2a_{ij}^{(k)}}{a_{ii}^{(k)} - a_{jj}^{(k)}} = \bar{P}_k; \quad \varphi^{(k)} = \frac{1}{2} \operatorname{arctg} \bar{P}_k,$$

где  $|2\varphi^{(k)}| \leq \frac{\pi}{2}$ ,  $i < j$  ( $a_{ij}$  выбирается в верхней треугольной наддиагональной части матрицы  $A$ ). Заметим, что при  $a_{ii}^{(k)} = a_{jj}^{(k)}$  получается  $\varphi^{(k)} = \frac{\pi}{4}$ .



В процессе итераций сумма квадратов всех недиагональных элементов  $\sigma(A^{(k)})$  при возрастании  $k$  уменьшается, так что  $\sigma(A^{(k+1)}) < \sigma(A^{(k)})$ . Элементы  $a_{ij}^{(k)}$ , приведенные к нулю на  $k$ -й итерации, на последующей итерации немного возрастают. При  $k \rightarrow \infty$  получается монотонно убывающая ограниченная снизу нулем последовательность  $\sigma(A^{(1)}) > \sigma(A^{(2)}) > \dots > \sigma(A^{(k)})$ . Поэтому  $\sigma(A^{(k)}) \rightarrow 0$  при  $k \rightarrow \infty$ . Это и означает сходимость метода. При этом

$$A^{(k)} \rightarrow \Lambda = \begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{pmatrix}.$$





## Методика решения задачи

1. Положить  $k = 0$ ,  $A^{(0)} = A$  и задать  $\varepsilon > 0$ .
2. Выделить в верхней треугольной наддиагональной части матрицы  $A^{(k)}$  максимальный по модулю элемент  $a_{ij}^{(k)}$ ,  $i < j$ .

Если  $|a_{ij}^{(k)}| \leq \varepsilon$  для всех  $i \neq j$ , процесс завершить. Собственные значения определяются по формуле

$$\lambda_i(A^{(k)}) = a_{ii}^{(k)}, \quad i = \overline{1, n}.$$

Собственные векторы  $X^i$  находятся как  $i$ -е столбцы матрицы, получающейся в результате перемножения:

$$v_k = H^{(0)}H^{(1)}H^{(2)} \dots H^{(k-1)} = (X^1, X^2, X^3, \dots, X^n).$$

Если  $|a_{ij}^{(k)}| > \varepsilon$ , процесс продолжается.



## Методика решения задачи

3. Найти угол поворота по формуле

$$\varphi^{(k)} = \frac{1}{2} \operatorname{arctg} \frac{2a_{ij}^{(k)}}{a_{ii}^{(k)} - a_{jj}^{(k)}} \quad \left( \text{при } a_{ii}^{(k)} = a_{jj}^{(k)} \text{ получается } \varphi^{(k)} = \frac{\pi}{4} \right).$$

4. Составить матрицу вращения  $H^{(k)}$ .

5. Вычислить очередное приближение

$$A^{(k+1)} = (H^{(k)})^T A^{(k)} H^{(k)}.$$

Положить  $k = k + 1$  и перейти к п. 2.



Метод главных компонент (PCA) — способ упростить сложные данные, сократив их размерность, сохраняя при этом как можно больше информации. Например, если у вас есть данные с десятками переменных, PCA может свести их к нескольким ключевым "направлениям", которые объясняют основную изменчивость.

PCA работает с матрицей ковариации данных, которая показывает, как переменные связаны друг с другом.



Данные заданы матрицей  $X = (x_i^j)$  размерности  $n \times m$ , где  $i = \overline{1, n}$  и  $j = \overline{1, m}$ ,  $n$  – число наблюдений (объектов),  $m$  – число признаков.

Обозначим за  $C$  ( $m \times m$ ) матрицу ковариаций признаков матрицы  $X$ :

$$c_{ij} = \frac{\sum_{p=1}^n x_k^i x_k^j}{n} - \mu_i \mu_j, \forall i, j \in \{1 \dots m\},$$

$\mu_i$  – среднее значение признака  $i, i \in \{1 \dots m\}$

В матричном виде:

$$C = \frac{X^T X}{n} - \mu^T \mu, \mu = (\mu_1 \dots \mu_m)$$



Данные заданы матрицей  $X = (x_i^j)$  размерности  $n \times m$ , где  $i = \overline{1, n}$  и  $j = \overline{1, m}$ ,  $n$  – число наблюдений (объектов),  $m$  – число признаков.

Обозначим за  $C$  ( $m \times m$ ) матрицу ковариаций признаков матрицы  $X$ :

$$c_{ij} = \frac{\sum_{p=1}^n x_k^i x_k^j}{n} - \mu_i \mu_j, \forall i, j \in \{1 \dots m\},$$

$\mu_i$  – среднее значение признака  $i, i \in \{1 \dots m\}$

В матричном виде:

$$C = \frac{X^T X}{n} - \mu^T \mu, \mu = (\mu_1 \dots \mu_m)$$



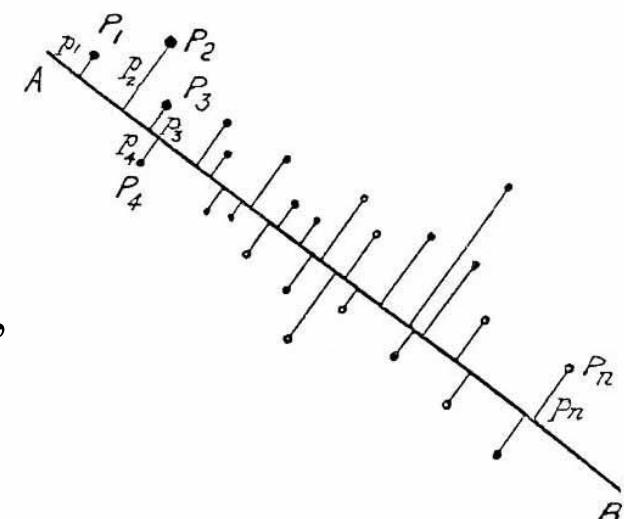
Матрица ковариации:

- Вычисляется на основе центрированных данных (где из каждой переменной вычли среднее).
- Её элементы показывают дисперсию (на диагонали) и ковариацию (вне диагонали) между переменными.

Вариация  $i$ -го признака:  $Var(x^i) = c_{ii}$

Общая вариация данных:  $Var(X) = \sum_{i=1}^m c_{ii}$

Задача: найти ортогональные векторы  $v$  такие, что  $v^T Cv \rightarrow \max$ , т.е. проекция данных на которые позволит сохранить наибольшую вариацию





Собственные векторы:

- Это направления наибольшей изменчивости данных, называемые главными компонентами.
- Каждый собственный вектор — это новая "ось", вдоль которой данные растянуты больше всего.

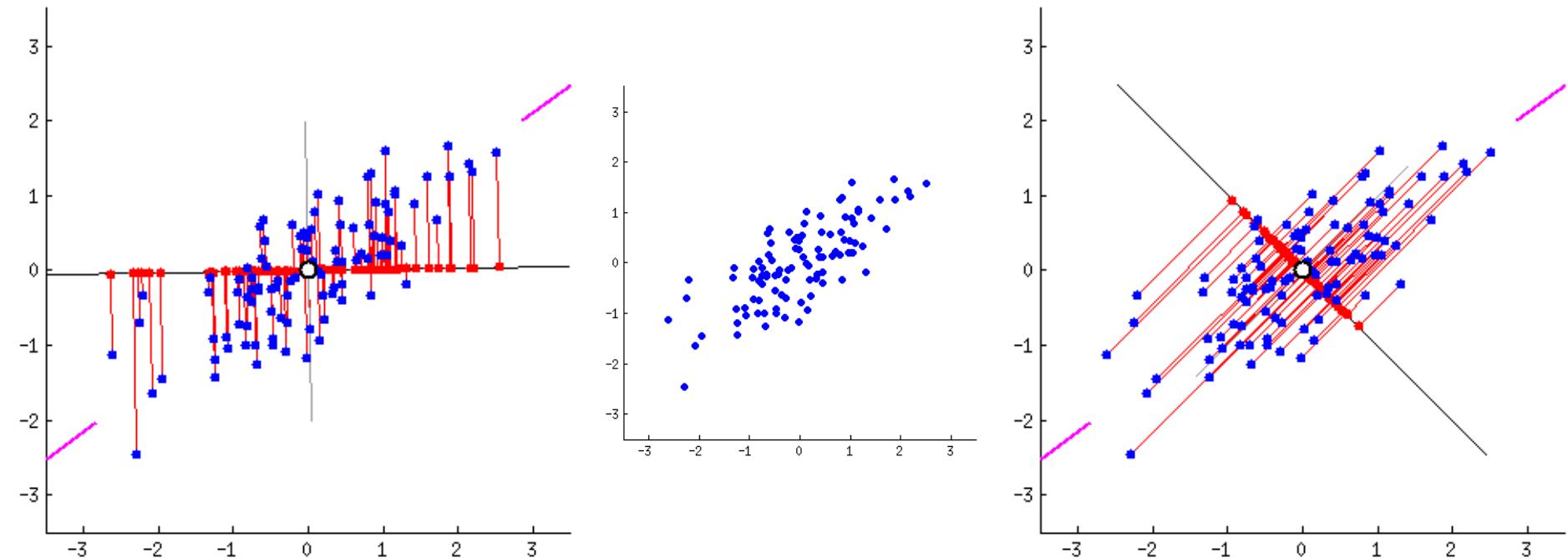
Собственные числа:

- Показывают, сколько дисперсии (информации) объясняет каждый главный компонент.
- Чем больше собственное число, тем важнее этот компонент для описания данных.



Представьте, что данные — это облако точек в пространстве. РСА находит "главные линии", вдоль которых облако вытянуто, и позволяет отбросить мелкие "шумы" в других направлениях.

Собственные векторы задают новые оси, по которым данные проще всего понять, а собственные числа показывают, сколько информации эти оси несут.



Чёрная линия характеризует ошибку аппроксимации, а красная — дисперсию.



Матрица  $C$  симметричная и положительно определена.  
Имеет место равенство:

$$C = V\Lambda V^T$$

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \lambda_m \end{bmatrix},$$

$\lambda$  – собственные значения матрицы  $C$ ,  $\sum_{i=1}^m \lambda_i = \sum_{i=1}^m c_{ii}$   
 $\lambda_1 > \lambda_2 \geq \dots \geq \lambda_m \geq 0$

$V(m \times m)$  – матрица собственных векторов матрицы  $C$ .



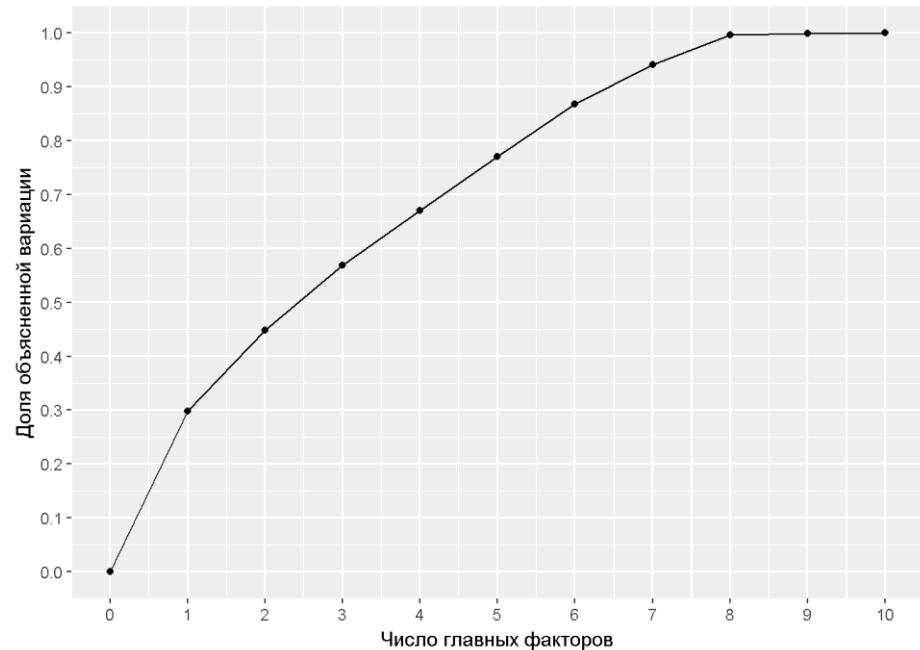
Главные компоненты:

$$U = X \cdot [v^1, v^2, \dots, v^k]^T, k < m$$

$$X^* = X \cdot V^T$$

Доля объясненной вариации:

$$\frac{\sum_{i=1}^k \lambda_i}{Var(X)}$$





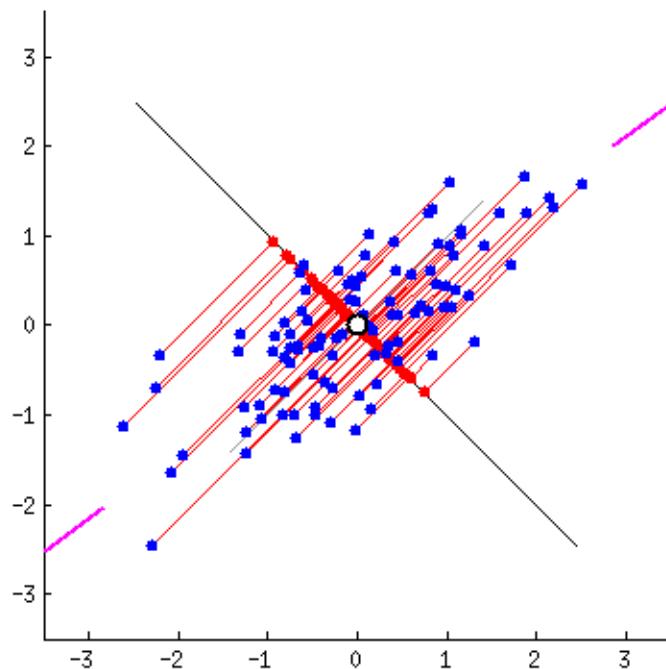
Пример ковариационной матрицы:

$$\begin{pmatrix} 1.07 & 0.63 \\ 0.63 & 0.64 \end{pmatrix}$$

Поскольку матрица симметрична, ее можно диагонализировать, перейдя к новой ортогональной системе координат, заданной ее собственными векторами:

$$\begin{pmatrix} 1.52 & 0 \\ 0 & 0.19 \end{pmatrix}$$

Это означает, что корреляция между новыми координатами равна нулю, а числа на диагонали — это собственные значения (1.52 и 0.19). Дисперсия проекции данных на любое направление представляет собой взвешенную сумму этих собственных значений. Максимальная дисперсия (1.52) достигается, если спроектировать данные на направление, соответствующее первому собственному вектору. Таким образом, первая главная компонента — это направление первого собственного вектора ковариационной матрицы.



На рисунке есть серая линия, перпендикулярная черной – вместе они образуют вращающуюся систему координат.

Пурпурные метки указывают направление первого собственного вектора ковариационной матрицы, который в данном случае равен  $(0.81, 0.58)$ .

## Домашняя работа #4

Задание посвящено реализации метода Штассена (перемножения матриц) и метода вращений (поиска спектра матрицы) с целью реализации метода главных компонент (Principal Component Analysis, PCA) для сжатия изображений.

Используйте **только** стандартные методы Python. Можно использовать math, matplotlib, PIL, time и базовый numpy.

Срок сдачи работ: **до 13-го апреля, 23:59.**

Работы принимаются на электронный адрес [VSChernyshenko@fa.ru](mailto:VSChernyshenko@fa.ru) **исключительно в виде ноутбука в формате html**. Название файла: “Фамилия\_Имя\_группа\_ДЗ4.html”.

**Не принимаются** работы являющиеся копией друг друга, работы **не содержащие подробные комментарии**.

## Домашняя работа #4

PCA – метод, который помогает упростить сложные данные. В случае изображений он позволяет "сжать" картинку, убрав менее важные детали, выделяя направления с наибольшей вариацией (собственные векторы) и их значимость (собственные значения).

- Вместо хранения всех признаков (всех столбцов пикселей в изображении), вы можете использовать меньшее количество главных компонент, которые объясняют большую часть вариации в данных. Что полезно для анализа или обработки данных.
- При этом, PCA отбрасывает компоненты с малой вариацией, которые часто связаны с шумом. Это улучшает качество данных для последующего анализа или восстановления изображения.
- В задачах машинного обучения работа с данными меньшей размерности (например, 20 компонент вместо 1024) значительно ускоряет обучение моделей и снижает потребление памяти. Что особенно важно для больших наборов данных.
- PCA позволяет спроектировать многомерные данные (например, 1024 измерений) в 2D или 3D пространство для визуального анализа, что невозможно сделать с исходными данными.

## Домашняя работа #4

Ваша цель – сжать выбранное вами изображение с помощью РСА, а затем использовать наивный метод, метод Штрассена и np.dot для перемножения матриц при вычислении ковариационной матрицы и восстановлении изображения. Вы также проверите, как число используемых собственных векторов влияет на качество восстановленного изображения.

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image # Для загрузки и обработки изображений
import urllib.request # Для скачивания изображения из интернета
import time

url = "https://.../image.png"
print("Загружаем изображение...")
urllib.request.urlretrieve(url, "Image.png")
img = Image.open("Image.png").convert('L')
img_array = np.array(img, dtype=float) / 255.0
height, width = img_array.shape

# Размер исходного файла в битах
original_size_bits = height * width * 8 # 8 бит на пиксель
original_size_kb = original_size_bits / 8 / 1024
print(f"Размер исходного файла до сжатия: {original_size_kb:.2f} KB")

# Преобразуем изображение для РСА. Для каждого столбца вычисляется среднее значение всех его элементов. Из каждого элемента
# столбца вычитается это среднее. Это нужно, чтобы данные были "сдвинуты" к нулю, что важно для анализа главных компонент (РСА).
X = img_array
X_mean = np.mean(X, axis=0)
X_centered = X - X_mean

# Вычисляем ковариационную матрицу для центрированных данных. Ковариационная матрица считается по формуле:
#
#
# cov_matrix = 
$$\frac{1}{m-1} \cdot (X_{\text{centered}}^T \cdot X_{\text{centered}})$$

```

## Домашняя работа #4

После разложения ковариационной матрицы на собственные векторы и значения вы получаете матрицу собственных векторов размером  $n \times n$ . Выбираем первые  $k$  столбцов этой матрицы, соответствующие наибольшим собственным значениям. Эти векторы — главные компоненты, которые содержат основную информацию об изображении. Обозначим выбранные векторы как  $V$ , размером  $n \times k$ .

Центрированные данные  $X_{\text{centered}}$  (размером  $m \times n$ ) проецируются на  $V$ :

$$\text{projection} = X_{\text{centered}} \cdot V$$

Сжатые данные обратно преобразуются в пространство исходной размерности:

$$X_{\text{reconstructed}} = \text{projection} \cdot V^T$$

$X_{\text{reconstructed}}$  — приближение к центрированному изображению размером  $m \times n$ . Чтобы вернуть изображение к исходному масштабу, добавляем среднее значение:

$$X_{\text{reconstructed}} = X_{\text{reconstructed}} + X_{\text{mean}}$$

Итак, вместо хранения полной матрицы  $m \times n$ , мы храним только  $\text{projection}$  ( $m \times k$ ) и  $V$  ( $n \times k$ ). При малом  $k$  это значительно экономит память.

Такой подход широко применяется в сжатии изображений, анализе данных и машинном обучении.

## Домашняя работа #4

...продолжение кода на 275-ом слайде:

```
# Находим собственные значения и векторы
# Они нужны для PCA: значения показывают "важность" направлений, а векторы – сами направления
# Сортируем их по убыванию, чтобы самые важные были первыми

# Функция для сжатия и восстановления
# Сжимает изображение с помощью PCA и восстанавливает его обратно.
# Возвращает: X_reconstructed - восстановленное изображение

# Показываем результаты для разного числа компонент
components_list = [ , , ]

plt.figure(figsize=(15, 5)) # Создаём большое окно для всех картинок
# Показываем оригинальное изображение
plt.subplot(1, len(components_list) + 1, 1)
plt.imshow(img_array, cmap='gray')
plt.title("Оригинал")
plt.axis('off')
.......
```

Размер исходного файла до сжатия: 256.00 KB

Размер файла после сжатия с 10 компонентами: 84.00 KB

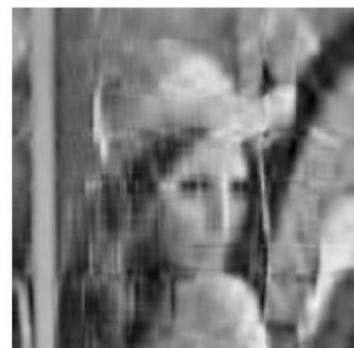
Размер файла после сжатия с 50 компонентами: 404.00 KB

Размер файла после сжатия с 100 компонентами: 804.00 KB

Оригинал



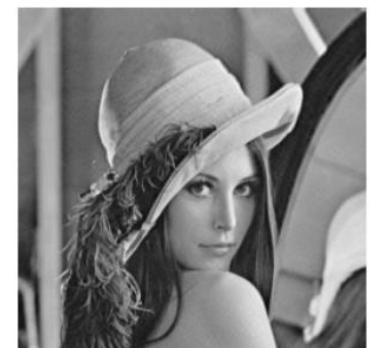
10 компонент



50 компонент



100 компонент



## Основные подзадачи:

### РЕАЛИЗАЦИЯ АЛГОРИТМА ШТРАССЕНА И НАИВНОГО МЕТОДА

Реализуйте рекурсивный алгоритм Штрассена и наивного метода для перемножения матриц.

### РЕАЛИЗАЦИЯ МЕТОДА ВРАЩЕНИЙ (ЯКОБИ)

Реализуйте итеративный метод Якоби для нахождения собственных значений и векторов симметричной матрицы.

### ЗАГРУЗКА И ПОДГОТОВКА ДАННЫХ

Загрузите ссылку на любое изображение из сети Интернет и преобразуйте его в матрицу. Центрируйте данные, вычтя среднее значение по столбцам.

### ВЫЧИСЛЕНИЕ КОВАРИАЦИОННОЙ МАТРИЦЫ

Вычислите ковариационную матрицу для центрированных данных, используя алгоритм Штрассена для перемножения матриц.

### ПОИСК СОБСТВЕННЫХ ЗНАЧЕНИЙ И ВЕКТОРОВ

Примените метод вращений (Якоби), наивный метод к ковариационной матрице для нахождения её собственных значений и векторов.

### СЖАТИЕ И ВОССТАНОВЛЕНИЕ ИЗОБРАЖЕНИЯ

Выполните сжатие изображения, используя различное число главных компонент ( $k$ ). Затем восстановите изображение на основе выбранных компонент.

### СРАВНЕНИЕ ПРОИЗВОДИТЕЛЬНОСТИ

Измерьте время выполнения вашего кода для метода Якоби, наивного метода. Сравните результаты с использованием стандартных методов из библиотеки NumPy (np.dot). Объясните полученные результаты.

### ОЦЕНКА КАЧЕСТВА

Оцените качество восстановленного изображения, вычислив среднеквадратичную ошибку (MSE) между оригинальным и восстановленным изображением. Выведите на экране полученных после сжатия по методу РСА изображения. Выведите их размер в килобайтах. Объясните полученные результаты.

### ОФОРМИТЕ ВЫВОДЫ К РАБОТЕ



Для какой матрицы легко найти весь спектр?

Существует класс матриц, для которого собственные числа можно найти очень легко, – это треугольные матрицы

$$A = \begin{pmatrix} \lambda_1 & * & * \\ 0 & \lambda_2 & * \\ 0 & 0 & \lambda_3 \end{pmatrix}.$$

Собственные числа матрицы  $A$  –  $\lambda_1, \lambda_2, \lambda_3$ , потому что детерминант имеет вид

$$\det(A - \lambda I) = (\lambda - \lambda_1)(\lambda - \lambda_2)(\lambda - \lambda_3)$$

Таким образом, вычисление собственных значений для треугольной матрицы – простая задача. Теперь на помощь приходят унитарные матрицы. Пусть  $U$  унитарная матрица, то есть  $U^*U = I$ . Тогда выполнены следующие равенства

$$\det(A - \lambda I) = \det(U(U^*AU - \lambda I)U^*) = \det(UU^*) \det(U^*AU - \lambda I) = \det(U^*AU - \lambda I),$$

где мы используем свойства детерминанта от произведения матриц,  $\det(AB) = \det(A) \det(B)$ . Это означает, что матрицы  $U^*AU$  и  $A$  имеют одинаковые характеристические многочлены, и, следовательно, одинаковые собственные значения.



Если мы приведём матрицу  $A$  к верхнетреугольному виду  $T$  с помощью унитарной матрицы  $U$ :  $U^*AU = T$ , мы решим задачу. Умножая слева и справа на  $U$  и  $U^*$  соответственно, получим нужное нам разложение:

$$A = UTU^*.$$

- Это **разложение Шура**.
- Использование унитарных матриц приводит к устойчивым алгоритмам, таким образом собственные значения вычисляются очень точно.
- Разложение Шура показывает, почему нам нужны матричные разложения: они представляют матрицу в виде произведения трёх матриц подходящей структуры.



## Теорема Шура

Каждая матрица  $A \in C^{n \times n}$  может быть представлена в виде формы Шура:

$$A = UTU^*$$

где  $U$  унитарная, а  $T$  верхнетреугольная.

1. Каждая матрица имеет как минимум один ненулевой собственный вектор (для корня характеристического многочлена матрица  $(A - \lambda I)$  вырождена и имеет нетривиальное ядро). Пусть

$$Av_1 = \lambda_1 v_1, \quad \|v_1\|_2 = 1.$$

2. Пусть  $U_1 = [v_1, v_2, \dots, v_n]$ , где  $v_2, \dots, v_n$  любые векторы ортогональные  $v_1$ . Тогда

$$U_1^* A U_1 = \begin{pmatrix} \lambda_1 & * \\ 0 & A_2 \end{pmatrix},$$

где  $A_2$  матрица размера  $(n-1) \times (n-1)$ . Она же называется блочнотреугольной формой. Теперь мы можем проделать аналогичную процедуру для матрицы  $A_2$  и так далее.



Важное приложение теоремы Шура связано с так называемыми нормальными матрицами.

**Определение.** Матрица  $A$  называется **нормальной матрицей**, если

$$AA^* = A^*A.$$

Примеры нормальных матриц: эрмитовы матрицы, унитарные матрицы.

**Теорема.** Матрица  $A$  - **нормальная матрица**, тогда и только тогда, когда  $A=U\Lambda U^*$ , где  $U$  - унитарная и  $\Lambda$  - диагональная матрицы.

**Следствие.** Любая нормальная матрица – **унитарно диагонализуема**. Это означает, что она может быть приведена к диагональному виду с помощью унитарной матрицы  $U$ . Другими словами, каждая нормальная матрица имеет ортогональный базис из собственных векторов.



Во многих задачах необходимо найти максимальный или минимальный собственный вектор и соответствующее ему значение.

Тогда, если  $A$  эрмитова матрица, отношение Релея определяется как

$$R_A(x) = \frac{(Ax, x)}{(x, x)}$$

и максимальное собственное значение равно максимальному значению  $R_A(x)$ , аналогично для минимального собственного значения.

Таким образом, мы можем использовать методы оптимизации для поиска этих экстремальных собственных значений.



QR алгоритм был предложен в 1961 г. независимо В. Н. Кублановской и J. Francis'ом.

QR алгоритм и QR разложение – разные вещи!

QR разложение – это представление матрицы в виде произведения двух матриц,  
**QR алгоритм использует QR разложение для вычисления разложения Шура.**

Рассмотрим выражение

$$A = QTQ^*$$

и перепишем его в виде

$$QT = AQ.$$

Слева замечаем QR разложение матрицы  $AQ$ .

Используем его чтобы записать одну итерацию метода неподвижной точки для разложения Шура.



Запишем следующий итерационный процесс

$$Q_{k+1}R_{k+1} = AQ_k, \quad Q_{k+1}^*A = R_{k+1}Q_k^*$$

Введём новую матрицу

$$A_k = Q_k^*AQ_k = Q_k^*Q_{k+1}R_{k+1} = \hat{Q}_kR_{k+1}$$

тогда аппроксимация для  $A_{k+1}$  имеет вид

$$A_{k+1} = Q_{k+1}^*A\hat{Q}_{k+1} = (Q_{k+1}^*A = R_{k+1}Q_k^*) = R_{k+1}\hat{Q}_k.$$

Итак, мы получили стандартную форму записи QR алгоритма.

Финальные формулы обычно записывают в **QR<sub>R</sub>Q**-форме:

1. Инициализируем  $A_0 = A$ .
2. Вычислим QR разложение матрицы  $A_k$ :  $A_k = Q_kR_k$ .
3. Обновим аппроксимацию  $A_{k+1} = R_kQ_k$ .

Продолжаем итерации пока  $A_k$  не станет достаточно треугольной (например, норма подматрицы под главной диагональю не станет достаточно мала).



- QR алгоритм сходится от первого диагонального элемента к последнему.
- По крайней мере 2-3 итерации необходимы для определения каждого диагонального элемента матрицы  $T$ .
- Каждый шаг состоит в вычислении QR разложения и одного произведения двух матриц, в результате имеем сложность  $O(n^3)$ .

Итоговая сложность -  $O(n^4)$ ?

Возможно ускорить QR алгоритм, используя сдвиги, поскольку матрица  $A_k - \lambda I$  имеет те же векторы Шура (столбцы матрицы  $U$ ).



Матрица  $A$  имеет верхне-гессенбергову форму, если  $a_{ij} = 0$ , при  $i \geq j + 2$ .

$$H = \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & * \end{bmatrix}.$$

С помощью отражений Хаусхолдера можно привести любую матрицу к верхне-гессенберговой форме:

$$U^*AU=H.$$

- Единственное отличие от вычисления разложения Шура заключается в занулении последних  $n - 2, n - 3, \dots, n - 2, n - 3, \dots$  элементов в первом, втором и так далее столбцах
- Сложность такого приведения  $O(n^3)$  операций
- Если матрица приведена к верхне-гессенберговой форме, то одна итерация QR алгоритма имеет сложность  $O(n^2)$  операций.
- Также верхне-гессенбергова форма матрицы сохраняется после выполнения одной итерации QR алгоритма.



- Если матрица  $A$  симметричная (эрмитова), то  $A=A^*$ , тогда  $H=H^*$  и верхне-гессенбергова форма оказывается трёхдиагональной матрицей.
- Будем говорить только о симметричном трёхдиагональном виде верхне-гессенберговой формы.
- Любая эрмитова матрица может быть приведена к трёхдиагональной форме с помощью отражений Хаусхолдера.

**Основная идея:** трёхдиагональная форма сохраняется при выполнении QR алгоритма, и сложность одной итерации может быть сокращена до  $O(n)$ .



Одна итерация QR алгоритма имеет следующий вид:

$$A_k = Q_k R_k, \quad A_{k+1} = R_k Q_k.$$

Если  $A_0 = A$  симметричная трёхдиагональная матрица, то эта форма сохраняется.

Работая с трёхдиагональной формой, не нужно вычислять матрицу  $Q$ : нужно лишь вычислить трёхдиагональную часть, которая получается после итерации

$$A_k = Q_k R_k, \quad A_{k+1} = R_k Q_k$$

в случае  $A_k = A_k^*$ .

Такая матрица определяется  $O(n)$  параметрами.

Вычисление QR разложения более сложное, но возможно вычислить  $A_{k+1}$  напрямую без вычисления  $Q_k$ .

Это называется неявный QR-шаг.



Все реализации неявного QR алгоритма основаны на следующей теореме:

**Теорема.** Пусть

$$Q^*AQ = H$$

верхне-гессенбергова форма матрицы. Тогда первый столбец матрицы  $Q$  определяет все остальные её столбцы. Он может быть найден из следующего уравнения:

$$AQ = QH.$$



Если у нас есть разложение вида:

$$A = X\Lambda X^{-1}, \quad A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

и

$$\Lambda = \begin{bmatrix} \Lambda_1 & 0 \\ 0 & \Lambda_2 \end{bmatrix}, \quad \lambda(\Lambda_1) = \{\lambda_1, \dots, \lambda_m\}, \quad \lambda(\Lambda_2) = \{\lambda_{m+1}, \dots, \lambda_r\},$$

а также есть зазор между собственными значениями в матрицах  $\Lambda_1$  и  $\Lambda_2$   $|\lambda_1| \geq \dots \geq |\lambda_m| > |\lambda_{m+1}| \geq \dots \geq |\lambda_r| > 0$ , тогда блок  $A^{(k)}_{21}$  матрицы  $A_k$  сходится к нулевому в процессе работы QR алгоритма со скоростью

$$\|A_{21}^{(k)}\| \leq Cq^k, \quad q = \left| \frac{\lambda_{m+1}}{\lambda_m} \right|,$$

где  $m$  размер матрицы  $\Lambda_1$ .

Таким образом, нам нужно увеличить зазор между  $\Lambda_1$  и  $\Lambda_2$ . Это можно сделать с помощью **QR алгоритма со сдвигами**.



Сходимость такого алгоритма линейная с фактором

$$A_k - s_k I = Q_k R_k, \quad A_{k+1} = R_k Q_k + s_k I$$

и

$$\left| \frac{\lambda_{m+1} - s_k}{\lambda_m - s_k} \right|,$$

где  $\lambda_m$  –  $m$ -ое большее по модулю собственное значение. Если сдвиг близок к собственному вектору, сходимость более быстрая.

- Существуют различные стратегии выбора сдвигов.
- Использование сдвигов – это общий подход к ускорению сходимости итерационных методов вычисления собственных значений.



Для того чтобы из малого собственного значения сделать большое, нужно обратить матрицу, что приводит нас к методу обратной итерации

$$x_{k+1} = (A - \lambda I)^{-1} x_k,$$

где  $\lambda$  – сдвиг, который близок к собственному значению, которое мы хотим найти. Сходимость – линейная.

Для ускорения сходимости можно использовать итерацию Релея, которая задаётся с помощью аддитивного выбора параметра сдвига:

$$x_{k+1} = (A - \lambda_k I)^{-1} x_k,$$

$$\lambda_k = \frac{(Ax_k, x_k)}{(x_k, x_k)}$$

В симметричном случае  $A=A^*$  сходимость локально кубическая, и локально квадратичная иначе.



Сингулярное разложение имеет вид

$$A=U\Sigma V^*$$

и существует для любой матрицы.

Его также можно считать способом приведения данной матрицы к диагональному виду с помощью двух унитарных преобразований:

$$\Sigma=U^*AV.$$

С помощью двусторонних преобразований Хаусхолдера мы можем привести любую матрицу к бидиагональной форме  $B$ .



Неявный QR алгоритм (со сдвигами) вычисляет собственные значения (разложение Шура). Но мы не можем применить его напрямую к бидиагональной матрице, поскольку она может быть недиагонализуема в общем случае.

Однако задачу вычисления сингулярного разложения можно свести к симметричной задаче на собственные значения двумя способами:

1. Работать с трёхдиагональной матрицей

$$T = B^* B$$

2. Работать с расширенной матрицей

$$T = \begin{bmatrix} 0 & B \\ B^* & 0 \end{bmatrix}$$

Случай 1 практически реализуем, если не формировать матрицу  $T$  явно.

Таким образом, задача вычисления сингулярных чисел может быть сведена к задаче вычисления собственных чисел симметричной трёхдиагональной матрицы.



Сингулярным разложением (SVD, Singular value decomposition) матрицы  $A \in \text{Mat}_{m \times n}(\mathbb{R})$  называется ее представление в виде

$$A = U\Sigma V^T$$

где  $\Sigma \in \text{Mat}_{m \times n}(\mathbb{R}^+)$  – диагональная матрица, элементы главной диагонали которой – сингулярные числа матрицы  $A$ ;  $U \in \text{Mat}_{m \times m}(\mathbb{R})$ ,  $V \in \text{Mat}_{n \times n}(\mathbb{R})$  – ортогональные матрицы, элементы которых – левые и правые сингулярные вектора.

Введём норму Фробениуса матрицы как

$$\|A\|_f = \sqrt{\text{tr}(A^T A)}$$



$$\|A\|_f = \sqrt{\text{tr}(A^T A)}$$

где  $\text{tr}(A^T A)$  – след матрицы  $A^T A$ . Обозначим через матрицу  $A_r$  ранга  $r < \text{rang} A$ . Возникает вопрос: как найти матрицу  $A_r$  наименее отличающуюся от  $A$  по норме Фробениуса (т.е. найти такую  $A_r$ , что  $\|A - A_r\|_f$  будет минимальна). Это можно сделать с помощью сингулярного разложения.

Теорема. Пусть  $\Sigma_r$  – матрица полученная из  $\Sigma$  заменой части диагональных элементов нулями:  $\sigma_{ii} = 0, i > r$ . Тогда  $A_r = U \Sigma_r V^T$ .

Последнее равенство можно переписать ещё в более экономичном виде:  $A_r = U_r \hat{\Sigma}_r V_r^T$ , где матрицы  $U_r, V_r$  и  $\Sigma_r$  получаются из  $U, V, \Sigma$  отсечением неиспользуемых элементов:

$$U_r = \begin{pmatrix} u_{11} & \dots & u_{1r} \\ \vdots & \ddots & \vdots \\ u_{m1} & \dots & u_{mr} \end{pmatrix}, V_r = \begin{pmatrix} v_{11} & \dots & v_{n1} \\ \vdots & \ddots & \vdots \\ v_{1r} & \dots & v_{nr} \end{pmatrix}, \hat{\Sigma}_r = \begin{pmatrix} \sigma_{11} & & \\ & \ddots & \\ & & \sigma_{rr} \end{pmatrix}$$



$$\|A\|_f = \sqrt{\text{tr}(A^T A)}$$

где  $\text{tr}(A^T A)$  – след матрицы  $A^T A$ . Обозначим через матрицу  $A_r$  ранга  $r < \text{rang} A$ . Возникает вопрос: как найти матрицу  $A_r$  наименее отличающуюся от  $A$  по норме Фробениуса (т.е. найти такую  $A_r$ , что  $\|A - A_r\|_f$  будет минимальна). Это можно сделать с помощью сингулярного разложения.

Теорема. Пусть  $\Sigma_r$  – матрица полученная из  $\Sigma$  заменой части диагональных элементов нулями:  $\sigma_{ii} = 0, i > r$ . Тогда  $A_r = U \Sigma_r V^T$ .

Последнее равенство можно переписать ещё в более экономичном виде:  $A_r = U_r \hat{\Sigma}_r V_r^T$ , где матрицы  $U_r, V_r$  и  $\hat{\Sigma}_r$  получаются из  $U, V, \Sigma_r$  отсечением неиспользуемых элементов:

$$U_r = \begin{pmatrix} u_{11} & \dots & u_{1r} \\ \vdots & \ddots & \vdots \\ u_{m1} & \dots & u_{mr} \end{pmatrix}, V_r = \begin{pmatrix} v_{11} & \dots & v_{n1} \\ \vdots & \ddots & \vdots \\ v_{1r} & \dots & v_{nr} \end{pmatrix}, \hat{\Sigma}_r = \begin{pmatrix} \sigma_{11} & & \\ & \ddots & \\ & & \sigma_{rr} \end{pmatrix}$$



Данные заданы матрицей  $X = (x_i^j)$  размерности  $n \times m$ ,  
где  $i = \overline{1, n}$  и  $j = \overline{1, m}$ ,  $n$  – число наблюдений (объектов),  
 $m$  – число признаков.

Требуется среди всех матриц такого же размера  $n \times m$  и ранга  $\leq k$  найти матрицу  $Y$ , для которой норма матрицы  $\|X - Y\|$  будет минимальной.



Решение зависит от матричной нормы.

Наиболее подходящие: Евклидова норма и норма Фробениуса.

Евклидова норма:

$$\|A\|_2 = \sqrt{\lambda_{\max}}$$

где  $\lambda_{\max}$  - максимальное  
собственное значение матрицы  $A$

Норма Фробениуса:

$$\|A\|_F = \sqrt{\sum_i \sum_j a_{ij}^2}$$



Существуют такие матрицы  $U$  и  $V$ , что выполняется равенство

$$X = U \cdot S \cdot V^T,$$

где  $U$  – матрица собственных векторов матрицы  $X \cdot X^T$ ,  $V$  – матрица собственных векторов матрицы  $X^T \cdot X$ , а матрица  $S$  размерности  $n \times m$  имеет на главной диагонали элементы  $\sigma_1, \sigma_2, \dots, \sigma_m$  и все остальные нули, где  $\sigma_i$  – сингулярные числа матрицы  $X$ , а  $\sigma_i^2$  – собственные числа матрицы  $X^T \cdot X$ .

Запишем матрицы  $U$  и  $V$  в векторном виде:

$$U = [u^1, u^2, \dots, u^n], \quad V = [v^1, v^2, \dots, v^m]$$

Тогда SVD разложение можно представить как

$$X = \sigma_1 u^1 (v^1)^T + \sigma_2 u^2 (v^2)^T + \dots + \sigma_m u^m (v^m)^T$$



Теорема Шмидта-Мирского:

Решением матричной задачи наилучшей аппроксимации в норме Евклида и в норме Фробениуса является матрица

$$X^* = \sigma_1 u^1 (v^1)^T + \sigma_2 u^2 (v^2)^T + \dots + \sigma_k u^k (v^k)^T$$

Ошибки аппроксимации:

$$\|X - X^*\|_2 = \sigma_{k+1},$$

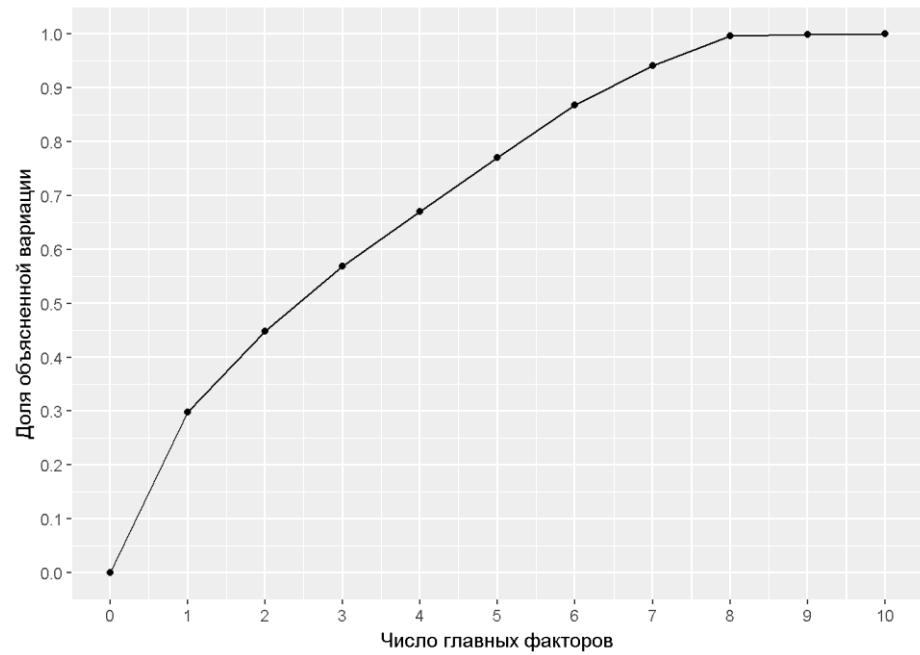
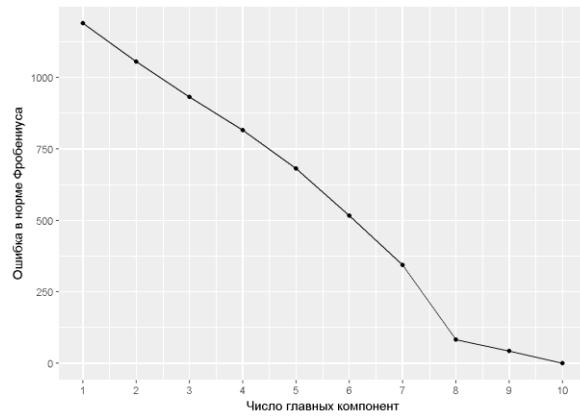
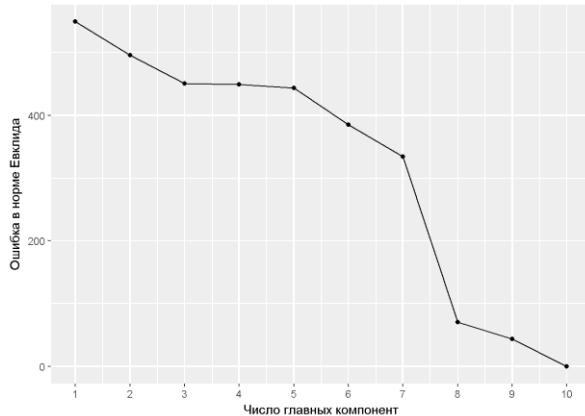
$$\|X - X^*\|_F = \sqrt{\sigma_{k+1}^2 + \sigma_{k+2}^2 + \dots + \sigma_m^2}.$$



Общая вариация данных:  $Var(X) = \sigma_1^2 + \sigma_2^2 + \dots + \sigma_m^2$

Доля объясненной вариации:  $\frac{\sigma_1^2 + \sigma_2^2 + \dots + \sigma_k^2}{Var(X)}, k < m$

Хорошим значением считается доля объясненной вариации  $\geq 80\%$





Рассмотрим произвольную матрицу  $A$  размером  $3 \times 3$ :

$$A = \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 2 \\ 2 & 0 & 1 \end{bmatrix}.$$

Цель — найти сингулярное разложение (SVD) матрицы  $A$ :

$$A = U\Sigma V^T,$$

где:

- $U$  — ортогональная матрица  $3 \times 3$ , состоящая из левых сингулярных векторов,
- $\Sigma$  — диагональная матрица  $3 \times 3$  с сингулярными значениями,
- $V^T$  — транспонированная ортогональная матрица  $3 \times 3$ , состоящая из правых сингулярных векторов.



Чтобы найти сингулярные значения, вычислим матрицу  $A^T A$ :

$$A^T = \begin{bmatrix} 1 & 0 & 2 \\ 2 & 1 & 0 \\ 0 & 2 & 1 \end{bmatrix},$$

$$A^T A = \begin{bmatrix} 1 & 0 & 2 \\ 2 & 1 & 0 \\ 0 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 2 \\ 2 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 5 & 2 & 2 \\ 2 & 5 & 2 \\ 2 & 2 & 5 \end{bmatrix}.$$

Собственные значения матрицы  $A^T A$  определяются из характеристического уравнения:

$$\det(A^T A - \lambda I) = 0,$$

$$A^T A - \lambda I = \begin{bmatrix} 5 - \lambda & 2 & 2 \\ 2 & 5 - \lambda & 2 \\ 2 & 2 & 5 - \lambda \end{bmatrix}.$$

Характеристический многочлен:

$$\det \begin{bmatrix} 5 - \lambda & 2 & 2 \\ 2 & 5 - \lambda & 2 \\ 2 & 2 & 5 - \lambda \end{bmatrix} = (5 - \lambda)[(5 - \lambda)^2 - 4] - 2[2(5 - \lambda) - 4] + 2[4 - 2(5 - \lambda)].$$

Упрощаем:

$$\begin{aligned} & (5 - \lambda)(25 - 10\lambda + \lambda^2 - 4) - 2(10 - 2\lambda - 4) + 2(4 - 10 + 2\lambda) \\ &= (5 - \lambda)(\lambda^2 - 10\lambda + 21) - 4(6 - \lambda) + 4(\lambda - 3). \end{aligned}$$



После разложения и упрощения получаем:

$$\lambda^3 - 15\lambda^2 + 54\lambda - 36 = 0.$$

Решая уравнение (например, численно), находим корни:

$$\lambda_1 = 9, \lambda_2 = 3 + \sqrt{3}, \lambda_3 = 3 - \sqrt{3}.$$

Сингулярные значения — квадратные корни собственных значений:

$$\sigma_1 = \sqrt{\lambda_1} = 3, \sigma_2 = \sqrt{3 + \sqrt{3}}, \sigma_3 = \sqrt{3 - \sqrt{3}}.$$

Для простоты численных вычислений используем приближения:

$$\sigma_2 \approx 2.334, \sigma_3 \approx 1.148.$$



Собственные векторы матрицы  $A^T A$  для каждого  $\lambda_i$ :

- Для  $\lambda_1 = 9$ :

$$A^T A - 9I = \begin{bmatrix} -4 & 2 & 2 \\ 2 & -4 & 2 \\ 2 & 2 & -4 \end{bmatrix}.$$

Решаем  $(A^T A - 9I)v_1 = 0$ . После упрощения получаем:

$$v_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \text{ нормируем: } v_1 = \frac{1}{\sqrt{3}} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.$$

- Для  $\lambda_2 = 3 + \sqrt{3}$ :

Собственный вектор находится численно или аналитически (для упрощения используем численное решение):

$$v_2 \approx \begin{bmatrix} 0.788 \\ -0.211 \\ -0.577 \end{bmatrix}.$$

- Для  $\lambda_3 = 3 - \sqrt{3}$ :

Аналогично:

$$v_3 \approx \begin{bmatrix} 0.577 \\ 0.788 \\ -0.211 \end{bmatrix}.$$



$$v_1 = \frac{1}{\sqrt{3}} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$v_2 \approx \begin{bmatrix} 0.788 \\ -0.211 \\ -0.577 \end{bmatrix}$$

$$v_3 \approx \begin{bmatrix} 0.577 \\ 0.788 \\ -0.211 \end{bmatrix}$$

Матрица  $V$ :

$$V = \begin{bmatrix} \frac{1}{\sqrt{3}} & 0.788 & 0.577 \\ \frac{1}{\sqrt{3}} & -0.211 & 0.788 \\ \frac{1}{\sqrt{3}} & -0.577 & -0.211 \end{bmatrix}.$$



Левые сингулярные векторы вычисляются как:

$$u_i = \frac{1}{\sigma_i} A v_i, \quad i = 1, 2, 3.$$

- Для  $\sigma_1 = 3, v_1 = \frac{1}{\sqrt{3}} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$ :  $A v_1 = \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 2 \\ 2 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \end{bmatrix} = \begin{bmatrix} \sqrt{3} \\ \sqrt{3} \\ \sqrt{3} \end{bmatrix}$ ,

$$u_1 = \frac{1}{3} \begin{bmatrix} \sqrt{3} \\ \sqrt{3} \\ \sqrt{3} \end{bmatrix} = \frac{1}{\sqrt{3}} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.$$

- Для  $\sigma_2 \approx 2.334, v_2 \approx \begin{bmatrix} 0.788 \\ -0.211 \\ -0.577 \end{bmatrix}$ :  $u_2 \approx \frac{1}{2.334} \begin{bmatrix} 0.366 \\ -1.154 \\ 0.788 \end{bmatrix} \approx \begin{bmatrix} 0.157 \\ -0.495 \\ 0.338 \end{bmatrix}$ .

- Для  $\sigma_3 \approx 1.148, v_3 \approx \begin{bmatrix} 0.577 \\ 0.788 \\ -0.211 \end{bmatrix}$ :  $u_3 \approx \begin{bmatrix} 0.788 \\ 0.211 \\ -0.577 \end{bmatrix}$ .



$$u_1 = \frac{1}{\sqrt{3}} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$u_2 \approx \begin{bmatrix} 0.157 \\ -0.495 \\ 0.338 \end{bmatrix}$$

$$u_3 \approx \begin{bmatrix} 0.788 \\ 0.211 \\ -0.577 \end{bmatrix}$$

Матрица  $U$ :

$$U \approx \begin{bmatrix} \frac{1}{\sqrt{3}} & 0.157 & 0.788 \\ \frac{1}{\sqrt{3}} & -0.495 & 0.211 \\ \frac{1}{\sqrt{3}} & 0.338 & -0.577 \end{bmatrix}.$$

Диагональная матрица  $\Sigma$ :

$$\Sigma = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 2.334 & 0 \\ 0 & 0 & 1.148 \end{bmatrix}.$$



## Итоговое разложение

Сингулярное разложение матрицы  $A$ :

$$A = U\Sigma V^T$$
$$= \begin{bmatrix} \frac{1}{\sqrt{3}} & 0.157 & 0.788 \\ \frac{1}{\sqrt{3}} & -0.495 & 0.211 \\ \frac{1}{\sqrt{3}} & 0.338 & -0.577 \end{bmatrix} \begin{bmatrix} 3 & 0 & 0 \\ 0 & 2.334 & 0 \\ 0 & 0 & 1.148 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \\ 0.788 & -0.211 & -0.577 \\ 0.577 & 0.788 & -0.211 \end{bmatrix}$$



## Проверка

Умножение  $U\Sigma V^T$  восстанавливает исходную матрицу  $A$  (с учётом численных погрешностей):

$$A \approx \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 2 \\ 2 & 0 & 1 \end{bmatrix}.$$



## Применение к реальной задаче

(например, по распознаванию лиц)

Представьте, что матрица  $A$  – это маленькое изображение лица ( $3 \times 3$  пикселя, для простоты). SVD позволяет:

- Найти "основные черты" лица (eigenfaces), которые хранятся в столбцах  $U$ . В примере  $u_1 = \frac{1}{\sqrt{3}} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$  — это как "средняя" черта лица.
- Сравнить новое лицо с известными, спроектировав его на эти основные черты. Это как сказать: "У этого лица такие-то пропорции глаз, носа и т.д.".

**Итого:** мы можем распознать человека, даже если у нас мало данных, или восстановить лицо с шумом (например, если часть пикселей потеряна).



## Применение к реальной задаче

### (удаление шума)

Если в данных есть ошибки (например, матрица A слегка искажена), SVD позволяет "очистить" её. Мы можем игнорировать маленькие сингулярные значения (как  $\sigma_3 \approx 1.148$ ), потому что они часто связаны с шумом.

Если в примере взять только  $\sigma_1$  и  $\sigma_2$ , мы получим матрицу, которая ближе к "идеальной" версии A.

### SVD помогает:

- Увидеть главное в сложной информации.
- Упростить работу с большими матрицами.
- Решать задачи, которые без SVD были бы очень сложными (например, распознавание лиц).



## Домашняя работа #5

Главная задача – построить классификатор лиц без использования нейронных сетей, используя только базовые методы языка Python для реализации ключевых алгоритмов, таких как умножение матриц и поиск собственных значений.

Запрещается использовать библиотеки типа NumPy для реализации матричных операций и разложений. Все алгоритмы должны быть реализованы вручную с использованием списков, циклов и других базовых конструкций Python.

**Срок сдачи работ: до 25-го апреля.**

Работы принимаются на электронный адрес [VSChernyshenko@fa.ru](mailto:VSChernyshenko@fa.ru) **только** в виде .html или .pdf файлов, название файла: “Фамилия\_Имя\_группа\_ДЗ5.\*”

**Не принимаются** работы являющиеся копией друг друга, работы не содержащие комментарии.



## Домашняя работа #5

### 1. Загрузка и подготовка данных

- Скачайте датасет AT&T Faces по ссылке:

[https://www.cl.cam.ac.uk/research/dtg/attarchive/pub/data/att\\_faces.zip](https://www.cl.cam.ac.uk/research/dtg/attarchive/pub/data/att_faces.zip)

*Каждое изображение представлено в градациях серого и имеет одинаковый размер 92x112 пикселей.*

- Возьмите первые 9 фотографий каждого из 40 человек (всего 360 изображений). Каждое изображение имеет размер 112×92 пикселей.
- Преобразуйте каждое изображение в вектор длиной 10304 ( $112 \times 92$ ) с помощью команды `np.reshape` из NumPy.
- Сформируйте матрицу  $F$  размером  $10304 \times 360$ , где каждый столбец – это вектор, соответствующий одному изображению.



## Домашняя работа #5

## 2. Вычисление и визуализация усредненного лица

- Вычислите усреднённое лицо как среднее арифметическое всех столбцов матрицы  $F$ . Обозначив его как вектор  $m$ :

$$m = \frac{1}{360} \sum_{i=1}^{360} F[:, i].$$

- Преобразуйте вектор  $m$  обратно в матрицу размером  $92 \times 112$  и визуализируйте его с помощью `matplotlib.pyplot.imshow`.
- Вычтите усреднённое лицо из каждого столбца матрицы  $F$ , чтобы получить центрированную матрицу:

$$F^* = F - m \cdot \mathbf{1}^T,$$

где  $\mathbf{1}$  — вектор из 360 единиц.





## Домашняя работа #5

## 3. Реализация SVD и малоранговая аппроксимация

Вычислите сингулярное разложение (SVD) матрицы  $F^*$ :

$$F^* = U\Sigma V^T,$$

где  $U$  – матрица левых сингулярных векторов,  $\Sigma$  – диагональная матрица сингулярных значений,  $V^T$  – матрица правых сингулярных векторов. Для этого:

- Вычислите матрицу  $A = F^{*T}F^*$  (размером  $360 \times 360$ ).
- Найдите собственные значения  $\lambda_i$  и собственные векторы  $V$  матрицы  $A$  с использованием QR-алгоритма (см. ниже).
- Сингулярные значения:  $\sigma_i = \sqrt{\lambda_i}$ .
- Левые сингулярные векторы:  $U = F^*V\Sigma^{-1}$ , где  $\Sigma$  – диагональная матрица с  $\sigma_i$  на диагонали.



## Домашняя работа #5

### 3. Реализация SVD и малоранговая аппроксимация

Реализуйте QR-алгоритм для поиска собственных значений и векторов:

- Приведите матрицу  $A$  к верхне-гессенберговой форме с помощью преобразований Хаусхолдера.
- Примените QR-алгоритм к гессенберговой матрице для нахождения собственных значений.
- Найдите собственные векторы с помощью обратной итерации или другого метода.



## Домашняя работа #5

### 3. Реализация SVD и малоранговая аппроксимация

Постройте малоранговую аппроксимацию  $F_r^* = U_r \Sigma_r V_r^T$  для  $r = 5$ , где:

$U_r$  — матрица с первыми  $r$  столбцами матрицы  $U$ .

$\Sigma_r$  — диагональная матрица с первыми  $r$  сингулярными значениями.

$V_r^T$  — первые  $r$  строк матрицы  $V^T$ .

Матрица  $U_r$  является базисом в пространстве лиц (eigenfaces), а  $W_r = \Sigma_r V_r^T$  — матрицей коэффициентов в этом базисе.

Малоранговая аппроксимация — это усечённое представление матрицы  $F^*$  в виде  $F_r^* = U_r \Sigma_r V_r^T$ , где берутся только первые  $r$  сингулярных значений и соответствующие им столбцы из  $U$  и  $V$ .

Помимо снижения размерности, такая аппроксимация способствует шумоподавлению, снижение размерности предотвращает переобучение, особенно если обучающих данных мало.



## Домашняя работа #5

## 4. Визуализация eigenfaces

- Преобразуйте первые 4 столбца матрицы  $U_r$  в матрицы размером 92x112. Формирует базис в пространстве лиц, известный как eigenfaces. Это основные направления вариации данных.
- Нарисуйте их с помощью команды subplots библиотеки matplotlib.pyplot, чтобы визуализировать eigenfaces.

Eigenface 1



Eigenface 2



Eigenface 3



Eigenface 4





## Домашняя работа #5

### 5. Классификация тестовых изображений

- Загрузите тестовую выборку из оставшихся 10-х фотографий каждого человека (всего 40 изображений). Сформируйте матрицу  $F_{\text{test}}$  размером  $10 \times 304 \times 40$  и центрируйте её:

$$F_{\text{test}}^* = F_{\text{test}} - m \cdot \mathbf{1}^T,$$

где  $\mathbf{1}$  — вектор из 40 единиц.

- Спроецируйте тестовые изображения на базис  $U_r$  (для  $r = 5$ ):

$$W_{\text{test}} = U_r^T F_{\text{test}}^*$$

Это даёт вектор коэффициентов размерности  $r$  для тестового изображения.

- Вычислите коэффициенты для обучающей выборки:

$$W = U_r^T F^*.$$



## Домашняя работа #5

### 5. Классификация тестовых изображений

Вектор  $w_{\text{test}}$  сравнивается с коэффициентами обучающих изображений из  $W_r$  (например, с помощью косинусной меры похожести).

- Для каждого столбца  $w_{\text{test}}$  в  $W_{\text{test}}$  найдите наиболее похожий столбец  $w$  в  $W$  с помощью косинусной меры похожести:

$$\cos(\theta) = \frac{w_{\text{test}}^T w}{\| w_{\text{test}} \| \| w \|}.$$

- Классифицируйте тестовое изображение, присваивая ему номер человека, соответствующий найденному столбцу  $w$  (присваивая им метку ближайшего обучающего изображения). Номер человека:  $\lfloor \text{индекс столбца}/9 \rfloor + 1$ .

Находится ближайшее обучающее изображение, и по его метке (номеру человека) определяется класс тестового изображения.

- Выведите индексы неправильно классифицированных фотографий для  $r = 5$ .



## Домашняя работа #5

### Требования

- Запрещается использовать библиотеку `numpy.linalg` или другие библиотеки для линейной алгебры. Все матричные операции, включая SVD, должны быть реализованы вручную с использованием базового функционала Python (списки, циклы и т.д.). Вы можете использовать функции для умножения матриц и т.п., разработанные вами ранее.
- Допускается использование `numpy` только для создания массивов, `reshape` и базовых операций (сложение, вычитание).
- Реализация QR-алгоритма должна быть включена для нахождения собственных значений и векторов матрицы  $F^{*T}F^*$ .
- Код должен быть читаемым и содержать комментарии, объясняющие шаги.



## Домашняя работа #5

### Требования

В рамках отчёта в конце ноутбука ответьте на следующие вопросы:

1. Почему необходимо представлять каждое изображение лица в виде вектора, а не оставлять его в виде двумерной матрицы?
2. Какое значение имеет использование первых 9 изображений каждого человека для обучения и 10-го для тестирования? Как этот выбор может повлиять на производительность классификатора?
3. Что представляет собой среднее лицо в контексте набора данных лиц? Почему важно вычитать среднее лицо из каждого изображения в обучающей выборке?
4. Почему eigenfaces часто выглядят как "призрачные" или абстрактные изображения лиц, а не как реальные лица?
5. Как выбор ранга  $r$  в низкоранговом приближении влияет на качество восстановления лиц и точность классификации? Какие компромиссы возникают при выборе меньшего или большего  $r$ ?
6. Что указывает высокое значение косинусного сходства между двумя векторами коэффициентов в контексте классификации лиц?
7. Может ли этот метод справляться с вариациями освещения, мимики или наличием препятствий (например, очков, шляп)? Почему или почему не может?



# МЕТОДЫ ЧИСЛЕННОГО ДИФФЕРЕНЦИРОВАНИЯ И ИНТЕГРИРОВАНИЯ



Если функция  $f(x)$  непрерывна на отрезке  $[a, b]$  и известна ее первообразная  $F(x)$ , то определенный интеграл от этой функции может быть вычислен по формуле Ньютона — Лейбница

$$\int_a^b f(x)dx = F(b) - F(a)$$

где  $F'(x) = f(x)$ . Однако во многих случаях возникают большие трудности, связанные с нахождением первообразной, или эта задача не может быть решена элементарными способами. Например, в элементарных функциях не выражается интеграл.

$$\int_1^2 \frac{dx}{\ln x}$$

Кроме того, в вычислительной практике часто требуется находить значения производных и определенных интегралов от сеточных функций, заданных в общем случае на неравномерной сетке  $\Omega_n = \{x_0, x_1, \dots, x_n\}$ ,  $x_{i+1} = x_i + h_{i+1}$   $i = \overline{0, n-1}$ ,  $h_{i+1} = x_{i+1} - x_i$ .

ПОСТАНОВКА ЗАДАЧИ 1  
ЧИСЛЕННОГО ИНТЕГРИРОВАНИЯ

Пусть на отрезке  $[a, b]$  на равномерной сетке  $\Omega_n (h_{i+1} = h = \text{const})$  заданы:

а) сеточная функция  $y_i = f(x_i)$ ,  $i = \overline{0, n}$ , своими значениями  $f_i = f(x_i)$  или сеточное представление формульной функции  $y = f(x)$ ;

б) желаемый порядок  $t$  точности (аппроксимации) относительно величины шага  $h$ .

Требуется с заданным порядком точности вычислить значение интеграла

$$\hat{I}_a^b \cong I_a^b = \int_a^b f(x) dx.$$

Иначе, требуется получить аппроксимационный оператор интегрирования  $\hat{I}_a^b$ , удовлетворяющий условию  $|\hat{I}_a^b - I_a^b| \leq Ch^t$ , где  $C = \text{const}$ , не зависящая от  $h$ .



## ПОСТАНОВКА ЗАДАЧИ 2 ЧИСЛЕННОГО ИНТЕГРИРОВАНИЯ

Пусть на отрезке  $[a, b]$  в общем случае на неравномерной сетке  $\Omega_n = \{x_0, x_1, \dots, x_n\}$  ( $x_{i+1} = x_i + h_{i+1}$ ,  $i = 0, n-1$ ,  $h_{i+1} = x_{i+1} - x_i = \text{var}$ ) заданы:

а) сеточная функция  $y_i = f(x_i)$ ,  $i = 0, n$ , своими значениями  $f_i = f(x_i)$  и возможно значениями производных  $f^{(p)}(x_i)$ ;

б) желаемый порядок  $t$  точности (аппроксимации) относительно шага.

Требуется с заданным порядком точности вычислить значение интеграла (получить аппроксимационный оператор)

$$\hat{I}_a^b \cong I_a^b = \int_a^b f(x) dx.$$

Отметим, что символом « $\wedge$ » здесь и далее обозначаются операторы дифференцирования и интегрирования.



Обыкновенные дифференциальные уравнения (ОДУ) – это уравнения, содержащие одну или несколько производных от искомой функции:

$$F(x, y, y', y'', y''', \dots, y^{(n)}) = 0$$

где  $x$  – независимая переменная,  $y = y(x)$  – искомая функция.

Наивысший порядок производной  $n$ , входящей в предыдущее уравнение, называют порядком дифференциального уравнения



Рассмотрим систему обыкновенных дифференциальных уравнений (ОДУ) первого порядка, записанную в виде:

$$\mathbf{y}'(x) = \mathbf{f}(x, \mathbf{y}(x))$$

Решение: любая функция  $y(x)$ , которая удовлетворяет уравнению. Решением ОДУ на интервале  $(a, b)$  называется функция  $y = \phi(x)$ , которая при ее подстановке в исходное уравнение обращает его в тождество на  $(a, b)$ .

Решение ОДУ в неявном виде  $\Phi(x, y) = 0$ , называется интегралом ОДУ.

Существует множество возможных решений. Для одного уникального решения необходимо указать независимые условия (для системы размером  $n$ ).

Например, когда  $n$  условий заданы для одной точки.

$$\mathbf{y}(0) = \mathbf{y}_0$$

Это задача Коши (задача с начальными условиями). Либо **дифференциальная задача**.



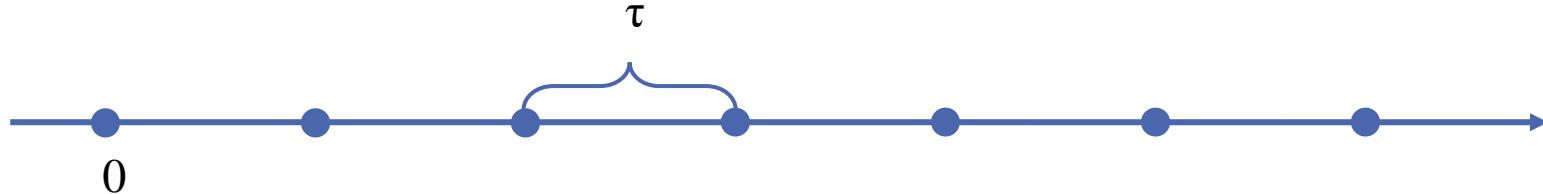
$$Ly = f$$

Точное решение

Дифференциальный оператор

Правая часть

```
graph TD; A[Точное решение] --> B["Ly = f"]; C[Дифференциальный  
оператор] --> B; D[Правая часть] --> B;
```



$$x_n = n\tau$$

$$y^n = y(x_n)$$

$$\frac{y^{n+1} - y^n}{\tau} = f(t, y^n)$$

$$y^0 = y_0$$

Разностная задача

Проекция правой части (на сетку)

Л<sub>\tau</sub>y<sub>\tau</sub> = f<sub>\tau</sub>

Разностная функция

Приближенное решение (сеточная функция)



Дифференциальная задача      Разностная задача

$$Lu = f \quad (1)$$

$$L_h u_h = f_h \quad (2)$$

**Опр 1:** Результат подстановки сеточной проекции точного решения (1) в разностную схему (2) называется невязкой

$$r_h = L_h[u]_h - f_h$$



**Опр 2:** Если  $\|r_h\|_{h \rightarrow 0} \rightarrow 0$ , то разностная задача (2) аппроксимирует дифференциальную задачу (1)

Если  $\|r_h\| \leq Ch^p$ , то аппроксимация порядка  $p$



**Опр 3:** Будем говорить, что решение разностной схемы устойчиво, если для решения двух возмущенных задач

$$L_h u_h^1 = f_h + \varepsilon_h^1$$

$$L_h u_h^2 = f_h + \varepsilon_h^2$$

Справедлива оценка

$$\|u_h^1 - u_h^2\| \leq C_1(\|\varepsilon_h^1\| + \|\varepsilon_h^2\|)$$



**Опр 4:** Будем говорить, что решение разностной задачи сходится к решению дифференциальной, если

$$\|u_h - [u]_h\| \leq C_2 h^p$$

$p$  – порядок сходимости



### Теорема (Лакса-Рябенького-Филипова):

Пусть семейство разностных схем (2) аппроксимирует дифференциальную задачу (1) и устойчиво, тогда решение задачи (2) сходится к решению задачи (1).

Причем, если аппроксимация имела порядок  $p$ , то и сходимость будет иметь порядок  $p$ .



В большинстве случаев поиск аналитического решения ОДУ первого порядка оказывается невозможным – приходится решать эту задачу численными методами.

Результатом решения ОДУ численными методами является таблица значений  $y = \varphi(x)$  на некотором множестве значений аргумента  $x$ . Поэтому при постановке задачи численного решения ОДУ первого порядка наряду с начальными условиями  $x_0, y_0$  необходимо задать область решения – отрезок  $[a; b]$  и шаг изменения аргумента  $h$ .

Таким образом, численное решение ОДУ представляет собой таблицу значений искомой функции  $y_i$  для заданной последовательности значений аргумента  $x_{i+1} = x_i + h$ ,  $i = 0, 1, \dots, n$ , где  $h = x_{i+1} - x_i$  называется шагом интегрирования.

$x$	$x_0=a$	$x_1$	$x_2$	...	$x_n=b$
$y$	$y_0$	$y_1$	$y_2$	...	$y_n$



Разложим функцию  $y(x)$  в окрестности точки  $x_0$  в ряд Тейлора:

$$y(x) = y(x_0) + (x - x_0)y'(x_0) + \frac{(x - x_0)^2}{2}y''(x_0) + \dots$$

который применяется для приближенного определения значения искомой функции  $y(x)$ . В точке  $x_0 + h$  при малых значениях  $h$  достаточно использовать только два слагаемых ряда, получим

$$y(x) = y(x_0 + h) = y(x_0) + y'(x_0)\Delta x + O(h^2)$$

где  $O(h^2)$  – бесконечно малая величина порядка  $h^2$ . Преобразуем:

$$y(x_0 + h) \approx y_0 + hf(x_0, y_0)$$

Теперь приближенное решение в точке  $x_1 = x_0 + h$  может быть вновь рассмотрено как начальное условие, следовательно, используя формулу, можно найти значение искомой функции в следующей точке  $x_2 = x_1 + h$ . Таким образом, был получен простой алгоритм решения задачи Коши, называемый методом Эйлера или методом ломаных.



## Явная схема Эйлера

$$\frac{u^{n+1} - u^n}{\tau} = f(t, u^n)$$

Разложение в ряд Тейлора

$$u^{n+1} = u^n + \tau u' + \frac{\tau^2}{2} u'' + O(\tau^3)$$

Невязка

$$\begin{aligned} r^n &= \frac{u^{n+1} - u^n}{\tau} - f(t, u^n) = \\ &= \frac{\cancel{u^n} + \tau u' + \frac{\tau^2}{2} u'' + O(\tau^3) - \cancel{u^n}}{\tau} - f(t, u^n) = \\ &= \cancel{u'} + \frac{\tau}{2} u'' + O(\tau^2) - \cancel{f(t, u^n)} = \frac{\tau}{2} u'' + O(\tau^2) \end{aligned}$$

1-ый порядок аппроксимации



## Неявная схема Эйлера

$$\frac{u^{n+1} - u^n}{\tau} = f(t, u^{n+1})$$

Разложение в ряд Тейлора

$$u^n = u^{n+1} - \tau u' + \frac{\tau^2}{2} u'' + O(\tau^3)$$

$$r^{n+1} = \frac{u^{n+1} - u^{n+1} + \tau u' - \frac{\tau^2}{2} u'' + O(\tau^3)}{\tau} -$$

$$-f(t, u^{n+1}) = -\frac{\tau}{2} u'' + O(\tau^2)$$

1-ый порядок аппроксимации



## Схема с центральной разностью

$$\frac{u^{n+1} - u^{n-1}}{2\tau} = f(t, u^n)$$

Разложение в ряд Тейлора

$$u^{n\pm 1} = u^n \pm \tau u' + \frac{\tau^2}{2} u'' \pm \frac{\tau^3}{6} u''' + O(\tau^4)$$

Невязка

$$r^n = \frac{u^{n+1} - u^{n-1}}{2\tau} - f(t, u^n) =$$

$$= \frac{2\tau u' + \frac{\tau^3}{3} u''' + O(\tau^4)}{2\tau} - f(t, u^n) =$$

$$= \frac{\tau^2}{6} u'' + O(\tau^3)$$

2-ой порядок аппроксимации



Метод Эйлера может быть представлен в виде последовательного применения формул:

$$x_1 = x_0 + h; \quad y_1 = y_0 + hy'_0 = y_0 + hf(x_0, y_0)$$

$$x_2 = x_1 + h; \quad y_2 = y_1 + hy'_1 = y_1 + hf(x_1, y_1)$$

...

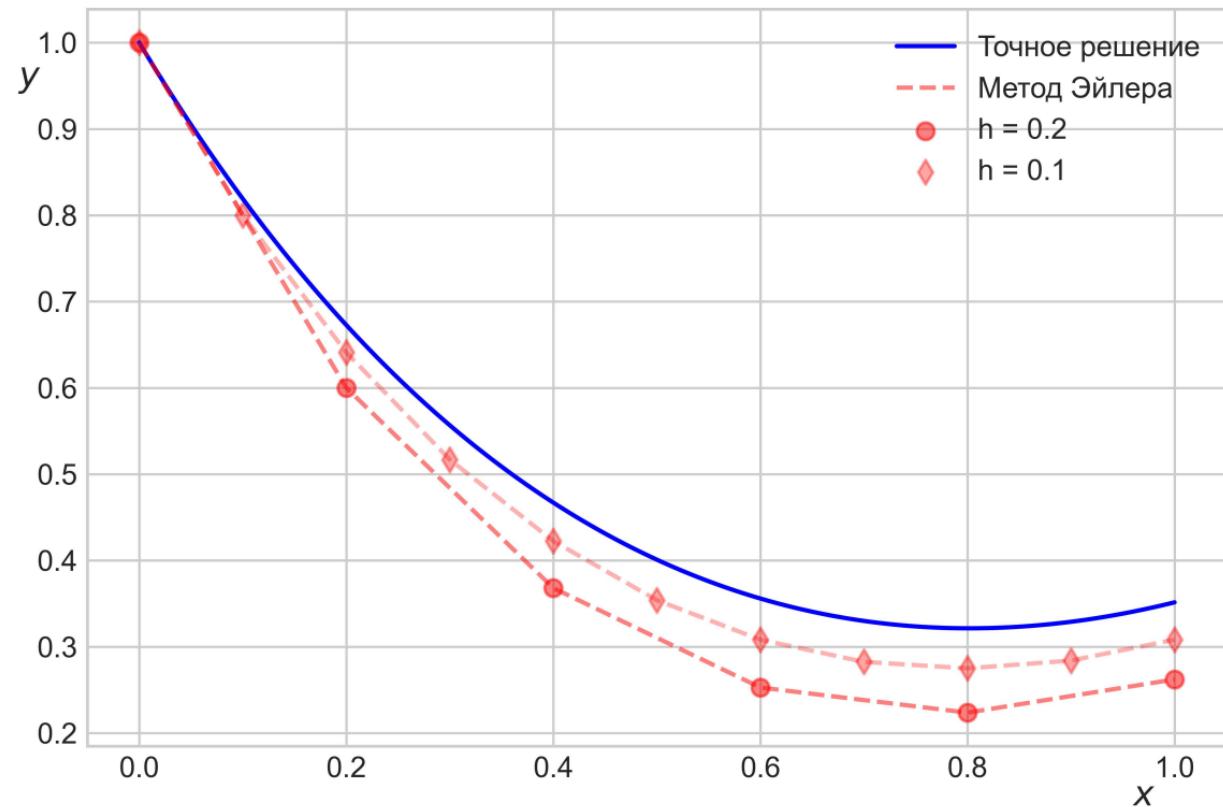
$$x_i = x_{i-1} + h; \quad y_i = y_{i-1} + y'_{i-1} = y_{i-1} + hf(x_{i-1}, y_{i-1})$$

В общем виде формула Эйлера записывается следующим образом

$$y_i = y_{i-1} + hf(x_{i-1}, y_{i-1}); \quad x_i = x_{i-1} + h$$



Второе название – «метод ломаных» обусловлено графической интерпретацией данного метода. Искомая функция  $y(x)$  заменяется ломаной линией с узлами в точках  $x_0, x_1, \dots, x_n$ . Метод Эйлера характеризуется достаточно высокой погрешностью вычисления:  $\Delta \approx O(h)$ . В дополнение, данный метод во многих случаях оказывается неустойчивым – малая ошибка (к примеру, заложенная в исходных данных) накапливается с увеличением  $x$ .



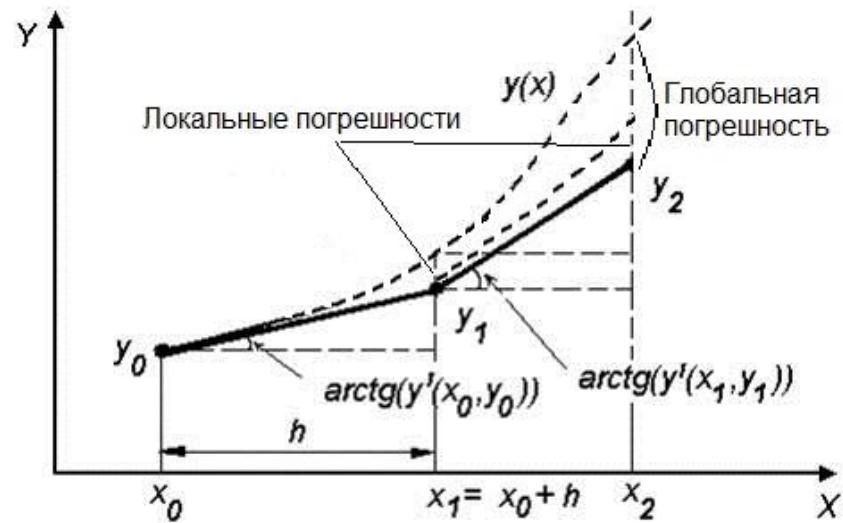
Вместе с тем, как показано на рисунке, точность метода Эйлера повышается при уменьшении размера шага вычислений  $h$ . Здесь также стоит отметить, что чрезмерно малое значение величины  $h$  приводит к снижению производительности вследствие увеличения количества вычислений: чем меньше шаг вычислений – тем больше итераций необходимо выполнить.



*Локальные погрешности* – погрешности, образовавшиеся на каждом шаге,

*Глобальная (накопленная) погрешность* – погрешность, образовавшаяся за несколько шагов.

Порядок глобальной погрешности относительно шага интегрирования на единицу ниже, чем порядок локальной погрешности. Таким образом, глобальная погрешность метода Эйлера имеет порядок  $p = 1$ :  $g_1 = C \cdot h$ , где  $C$  – некоторая постоянная.



Порядок численного метода для решения ОДУ определяется порядком его глобальной погрешности. Он может быть также определен, как количество вычислений значения производной  $f(x, y)$  искомой функции на каждом шаге. В соответствии с этим метод Эйлера является методом первого порядка.



Рассмотрим решение обыкновенного дифференциального уравнения первого порядка:

$$\frac{dy}{dx} = \frac{y}{(\cos(x))^2}$$

методом Эйлера на отрезке  $[0, 1]$  с шагом  $h = 0.1$ .  
Начальные условия:  $x_0 = 0; y_0 = 2.7183$ .

Построим таблицу значений переменной  $y_i$  при соответствующих значениях переменной  $x_i$ .



Построим таблицу значений переменной  $y_i$  при соответствующих значениях переменной  $x_i$ .

1.  $y_1 = y_0 + h \cdot \frac{y_0}{(\cos(x_0))^2} = 2.7183 + 0.1 \cdot \frac{2.7183}{(\cos(0))^2} = 2.9901$
2.  $y_2 = y_1 + h \cdot \frac{y_1}{(\cos(x_1))^2} = 2.9901 + 0.1 \cdot \frac{2.9901}{(\cos(0.1))^2} = 3.2922$
3.  $y_3 = y_2 + h \cdot \frac{y_2}{(\cos(x_2))^2} = 3.2922 + 0.1 \cdot \frac{3.2922}{(\cos(0.2))^2} = 3.6349$
4.  $y_4 = y_3 + h \cdot \frac{y_3}{(\cos(x_3))^2} = 3.6349 + 0.1 \cdot \frac{3.6349}{(\cos(0.3))^2} = 4.0332$
5.  $y_5 = y_4 + h \cdot \frac{y_4}{(\cos(x_4))^2} = 4.0332 + 0.1 \cdot \frac{4.0332}{(\cos(0.4))^2} = 4.5086$
6.  $y_6 = y_5 + h \cdot \frac{y_5}{(\cos(x_5))^2} = 4.5086 + 0.1 \cdot \frac{4.5086}{(\cos(0.5))^2} = 5.0940$
7.  $y_7 = y_6 + h \cdot \frac{y_6}{(\cos(x_6))^2} = 5.0940 + 0.1 \cdot \frac{5.0940}{(\cos(0.6))^2} = 5.8418$
8.  $y_8 = y_7 + h \cdot \frac{y_7}{(\cos(x_7))^2} = 5.8418 + 0.1 \cdot \frac{5.8418}{(\cos(0.7))^2} = 6.8404$
9.  $y_9 = y_8 + h \cdot \frac{y_8}{(\cos(x_8))^2} = 6.8404 + 0.1 \cdot \frac{6.8404}{(\cos(0.8))^2} = 8.2497$
10.  $y_{10} = y_9 + h \cdot \frac{y_9}{(\cos(x_9))^2} = 8.2497 + 0.1 \cdot \frac{8.2497}{(\cos(0.9))^2} = 10.3847$



## Программная реализация

```
1 import math
2
3
4 def func(x, y):
5     return y / (math.cos(x) ** 2)
6
7
8 def eiler(func, x0, xf, y0, h):
9     count = int((xf - x0) / h) + 1
10    y = [y0]
11    x = x0
12    for i in range(1, count):
13        y.append(y[i-1] + h * func(x, y[i-1]))
14        x += h
15
16    return y
17
18
19 print(eiler(func, 0, 1, 2.7183, 0.1))
20 #[2.7183, 2.99013, 3.2921531777519295, 3.6348964001721735,
21 #4.033167969573622, 4.508579299929644, 5.093994160733677,
22 #5.841814495589913, 6.840443308551035, 8.249681038793268, 10.384697446612579]
```



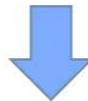
**Опр 5:** Разностная схема (2) с линейным оператором  $L_s$  называется устойчивой, если для решения уравнения (2) имеет место

$$\|u_s\| \leq M \|f_s\|$$

**Теорема:** В случае линейного оператора  $L_s$  определения устойчивости (опр.3 и опр.5) эквивалентны



$$L_h u_h = f_h$$



Канонический вид

$$u^{n+1} = R_h u^n + h \rho_n$$



оператор перехода

Достаточное условие устойчивости –  
ограниченность нормы оператора  
перехода

$$\|R_h^n\| \leq C, \quad C = \text{const}, n = 1, 2, \dots N$$



Достаточное условие устойчивости –  
ограниченность собственных чисел

$$\|R_h\| \geq |\lambda_i| \quad \rightarrow \quad |\lambda_i^n| < M$$

$|\lambda_i^n| \leq 1$  - Строгая устойчивость

$|\lambda_i^n| \leq 1 + c\tau$  - нестрогая устойчивость



$$\begin{aligned}\|u^n\| &= \|R_h u^{n-1} + h\rho_{n-1}\| \leq \|R_h\| \|u^{n-1}\| + \\&+ h\|\rho_{n-1}\| \leq \\&\leq \|R_h^2\| \|u^{n-2}\| + h(\|R_h\| \|\rho_{n-1}\| + \|\rho_{n-2}\|) \leq \cdots \\&\leq \|R_h^n\| \|y^0\| + \\&+ h(\|R_h^{n-1}\| \|\rho_{n-1}\| + \|R_h^{n-2}\| \|\rho_{n-2}\| + \cdots + \|\rho_0\|) \leq \\&\leq \max \|R_h^k\| (\|y^0\| + Nh \max_{0 \leq k \leq n-1} \|\rho_k\|)\end{aligned}$$



Пусть  $\|F\| = \max_k(\|y_0\|, \|\rho_k\|)$

Если  $\|R_h^k\| \leq C, \quad C = \text{const}, n = 1, 2, \dots N$



$\|u^n\| \leq C\|F\|(1 + Nh)$  - нестрогая  
устойчивость

Ограниченнность норм степеней оператора  
перехода является достаточным условием  
устойчивости разностных схем

Пусть, например,  $\|R_h\| \geq \max_i |\lambda_i|$

Для устойчивости необходимо

$$\forall k \in \{0, \dots, N\} \quad |\lambda_i^k| \leq C$$

Такое неравенство

$$\|R_h^k\| \leq C, \quad C = \text{const}, n = 1, 2, \dots, N$$

может выполняться в двух случаях

$|\lambda_i^n| \leq 1$  - Строгая устойчивость  $C \approx e^{MX}$

Если нужно считать неопределенное  
количество шагов по времени

$|\lambda_i^n| \leq 1 + Ct$  - Нестрогая (слабая) устойчивость  $C \approx 1$

Нужно сделать несколько  
шагов по времени



$$\frac{du}{dt} = f(t, u) \quad u(0) = u_0$$

### Явная схема Эйлера

$$\frac{u^{n+1} - u^n}{\tau} = f(t, u^n)$$

Исследование на устойчивость  
проводится для **линейных** задач



Линеаризуем правую часть в  
окрестности некоторой траектории  $u^*$

$$f(t, u^n) = f(t, u^*) + \frac{\partial f}{\partial u}(u^n - u^*)$$

Тогда

$$u^{n+1} = \left(1 + \tau \frac{\partial f}{\partial u}\right) u^n + \tau \rho_n$$

$$\rho_n = f(t, u^*) - \frac{\partial f}{\partial u} u^*$$



$$L_h u^n = f_h$$

$$L_h u^{**} = f_h$$

воздушенное  
решение



$$L_h(u^n - u^{**}) = 0$$



$$L_h \Delta^n = 0$$

$$\Delta^n = \sum_{i=1}^n c_i \lambda_i^n$$

Члены ряда независимы  $\rightarrow$  проводим  
исследование для каждого из них

$$\Delta^n \rightarrow C \lambda^n$$



Тогда

$$u^{n+1} = \left(1 + \tau \frac{\partial f}{\partial u}\right) u^n + \tau \rho_n$$



$$\Delta^{n+1} = \left(1 + \tau \frac{\partial f}{\partial u}\right) \Delta^n$$

$$\lambda = 1 + \tau \frac{\partial f}{\partial u} = 1 + c\tau \quad c = \max \left| \frac{\partial f}{\partial u} \right|$$

Нестрогая устойчивость



Если  $\left|1 + \tau \frac{\partial f}{\partial u}\right| \leq 1$

$$-1 \leq 1 + \tau \frac{\partial f}{\partial u} \leq 1$$



$$\frac{\partial f}{\partial u} \leq 0$$

$$\tau \leq 2 / \left| \frac{\partial f}{\partial u} \right|$$

Строгая устойчивость



## Неявная схема Эйлера

$$\frac{u^{n+1} - u^n}{\tau} = f(t, u^{n+1})$$

$$f(t, u^{n+1}) = f(t, u^*) + \frac{\partial f}{\partial u}(u^{n+1} - u^*)$$

$$u^{n+1} = \left( u^n + \tau \frac{\partial f}{\partial u} u^{n+1} \right) + \tau \rho_n$$



$$\Delta^{n+1} = \frac{1}{1 - \tau \frac{\partial f}{\partial u}} \Delta^n$$



$$R_h = \frac{1}{1 - \tau \frac{\partial f}{\partial u}}$$



$\frac{\partial f}{\partial u} > 0$  Строгая устойчивость

$\frac{\partial f}{\partial u} \leq 0$   $\tau \leq 2 / \left| \frac{\partial f}{\partial u} \right|$  Нестрогая  
устойчивость



## Схема с центральной разностью

$$\frac{u^{n+1} - u^{n-1}}{2\tau} = f(t, u^n)$$

$$f(t, u^n) = f(t, u^{**}) + \frac{\partial f}{\partial u}(u^n - u^{**})$$

$\lambda$  – собственные значения оператора  
перехода



## Схема с центральной разностью

$$\frac{u^{n+1} - u^{n-1}}{2\tau} = f(t, u^n)$$

$$f(t, u^n) = f(t, u^{**}) + \frac{\partial f}{\partial u}(u^n - u^{**})$$

$\lambda$  – собственные значения оператора  
перехода



$$L_h \Delta^n = 0 \quad \rightarrow \quad \frac{\Delta^{n+1} - \Delta^{n-1}}{2\tau} = \frac{\partial f}{\partial u} \Delta^n$$

$$\Delta^n \rightarrow C \lambda^n$$

$$\frac{\lambda^2 - 1}{2\tau} = \frac{\partial f}{\partial u} \lambda \quad \rightarrow \quad \lambda^2 - 2\tau \frac{\partial f}{\partial u} \lambda - 1 = 0$$

По теореме Виета  $\lambda_1 \lambda_2 = 1$

$$\frac{D}{4} = h^2 \left( \frac{\partial f}{\partial y} \right)^2 + 1 > 0$$

$$|\lambda_1| > 1 \quad |\lambda_2| < 1$$

Схема неустойчива



Примеры имитационных моделей,  
описываемых системами  
дифференциальных уравнений



## Модель хищник-жертва Лотки-Вольтерра

Модель имеет следующую форму:

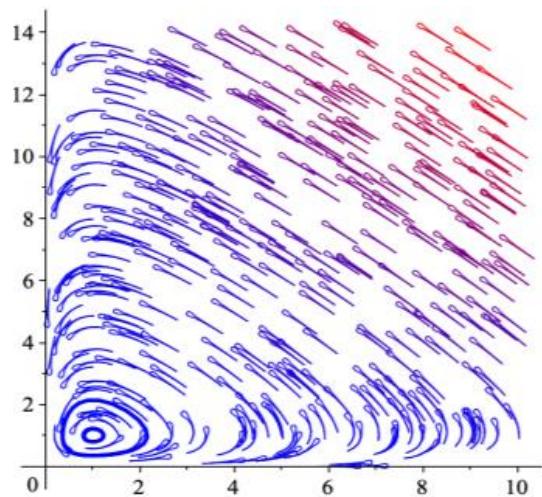
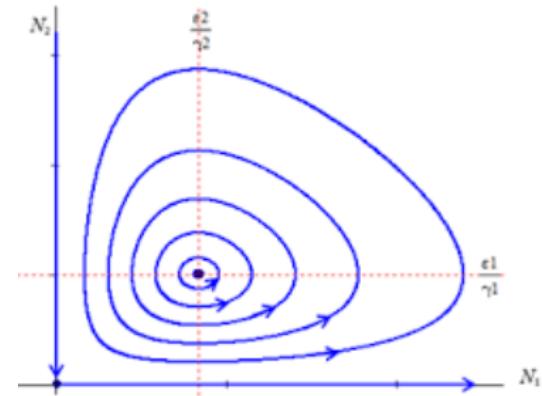
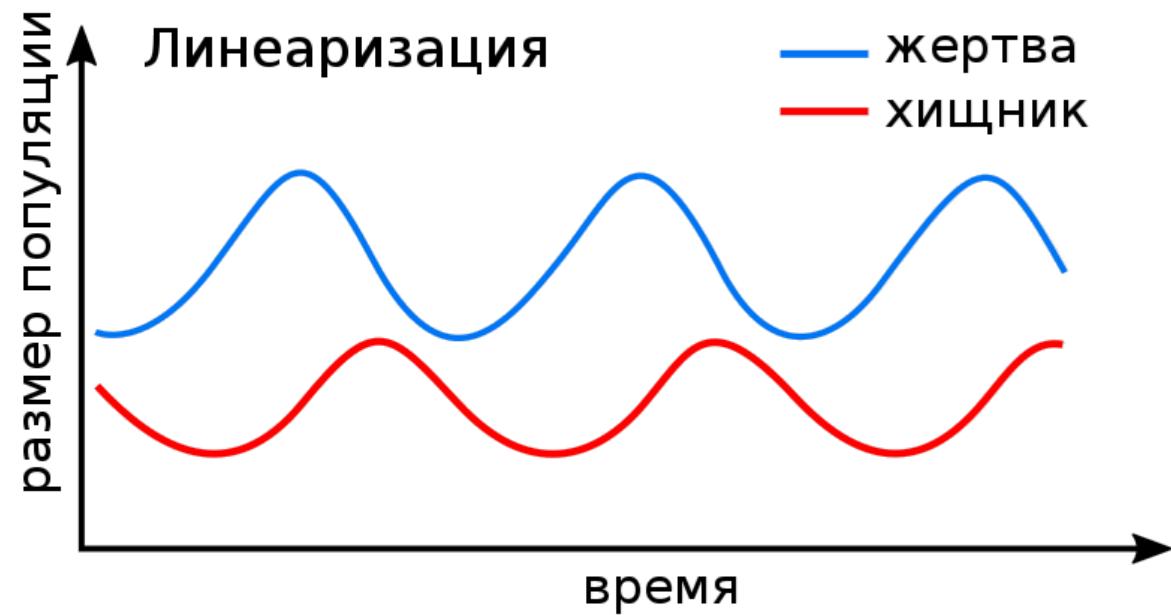
$$\begin{aligned}x' &= ax - bx^2 - cxy \\y' &= -dy + exy\end{aligned}$$

Здесь  $x$  - это размер популяции жертв ("кролики"), а  $y$  - размер популяции хищников ("лисы").

Мы используем логистические уравнения для жертв, потому что их популяция ограничена внешним (относительно системы) фактором (например, количеством "травы"). Популяция хищников контролируется внутренним фактором (размером популяции жертв), и для этой популяции мы используем уравнение Мальтуса. Поскольку без пищи ("кроликов") популяция лис не может существовать, ее параметр роста выбирается как отрицательный.



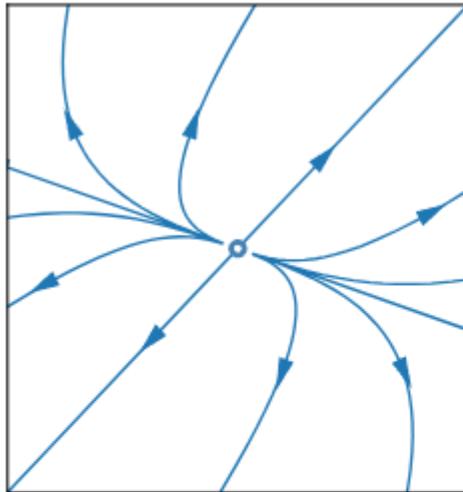
## Модель хищник-жертва Лотки-Вольтерра



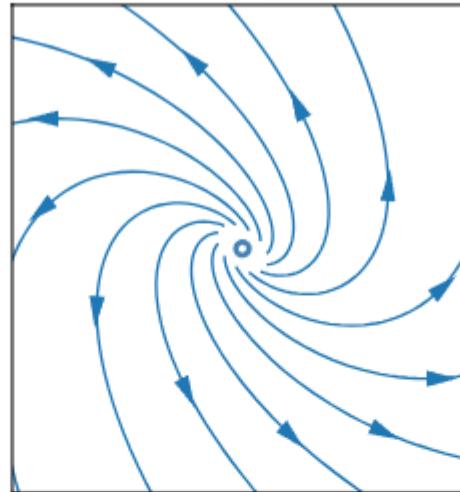


## Точки устойчивости

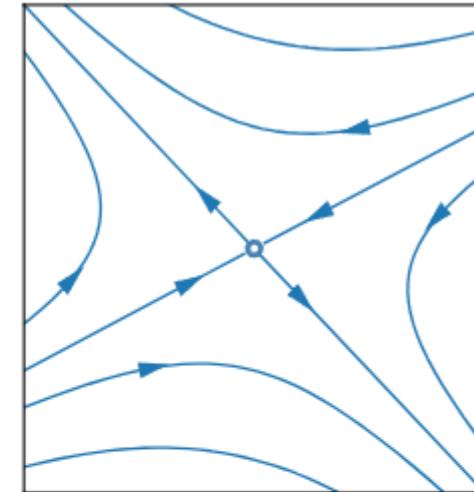
Неустойчивый узел



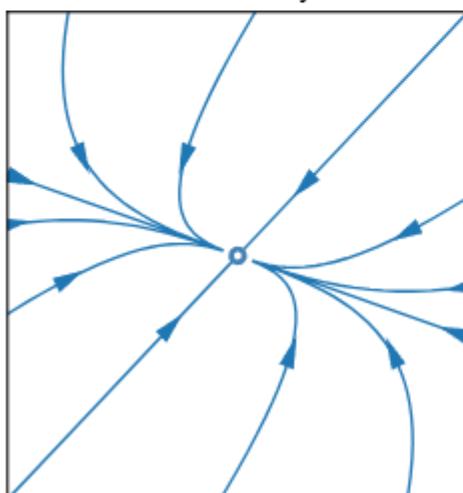
Неустойчивый фокус



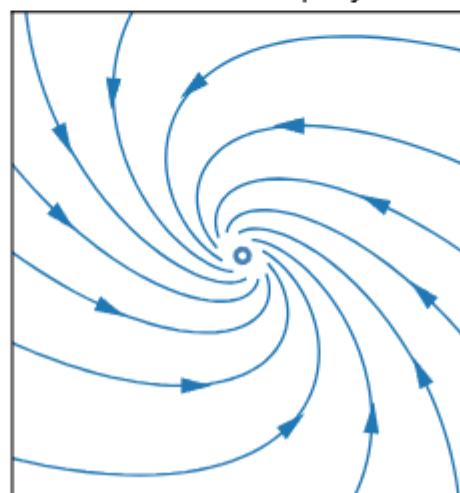
Седло



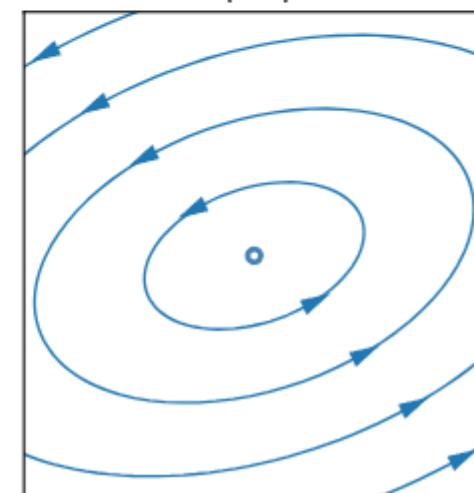
Устойчивый узел



Устойчивый фокус

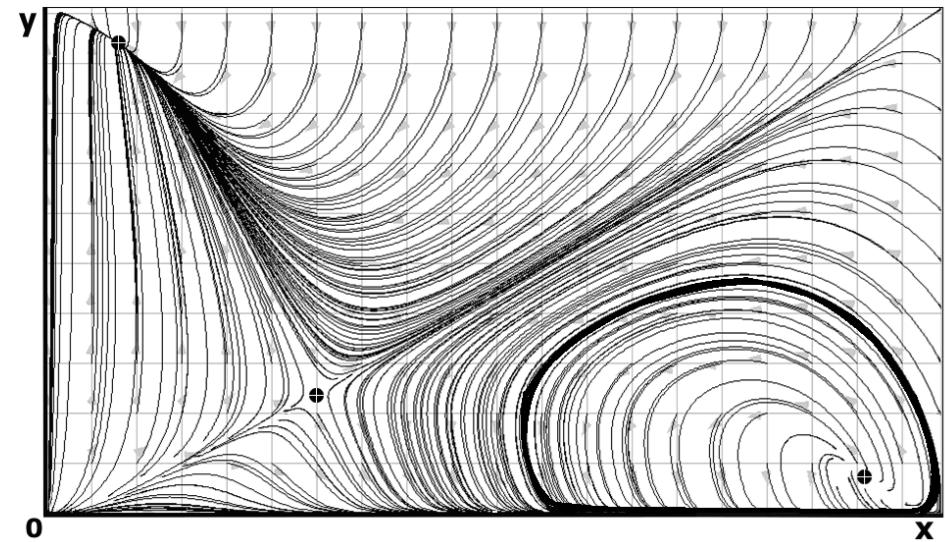
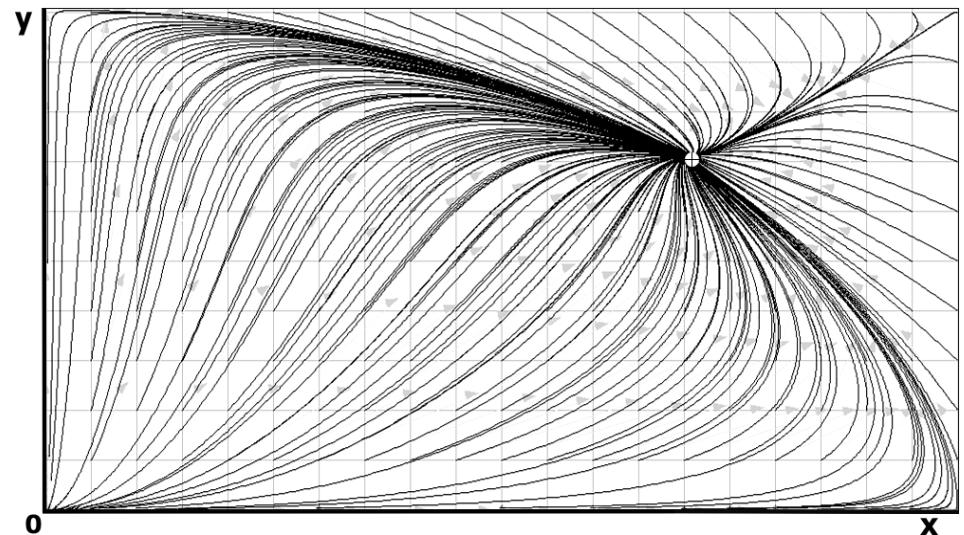
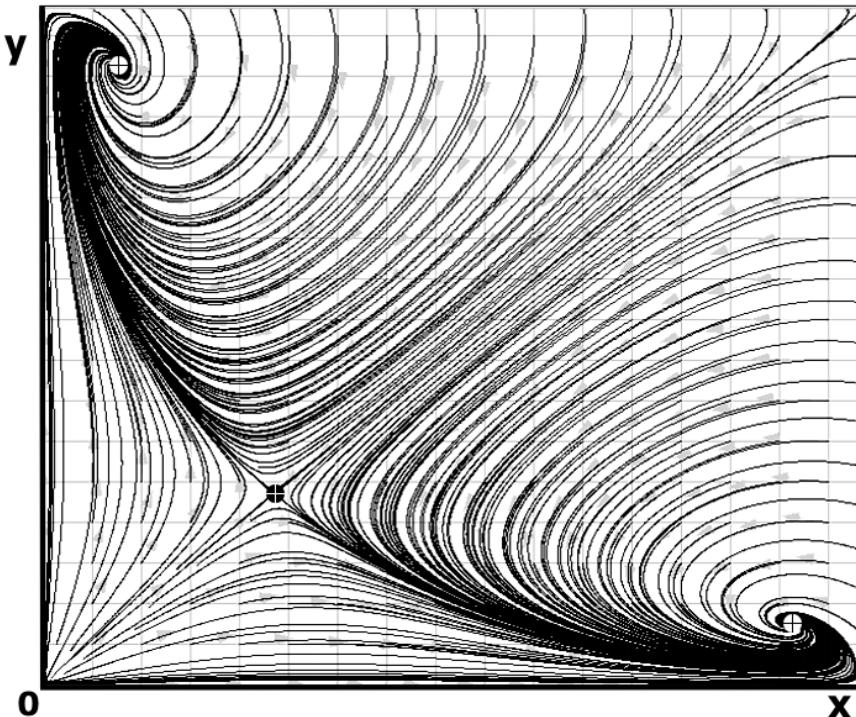


Центр





## Фазовый портрет





## Уравнения Навье-Стокса

Система дифференциальных уравнений в частных производных, описывающая движение вязкой ньютоновской жидкости.

Определим скорость изменения импульса единицы объема жидкости. Воспользуемся тензорными обозначениями

$$\frac{\partial}{\partial t} \cdot (\rho v_i) = \rho \frac{\partial v_i}{\partial t} + \frac{\partial \rho}{\partial t} v_i .$$

$$\frac{\partial}{\partial t} (\rho \cdot v_i) = -\rho \cdot v_k \frac{\partial v_i}{\partial x_k} - \frac{\partial p}{\partial x_i} - v_i \frac{\partial(\rho \cdot v_k)}{\partial x_k} = -\frac{\partial p}{\partial x_i} - \frac{\partial(\rho \cdot v_i \cdot v_k)}{\partial x_k} .$$

Давление жидкости в точке:

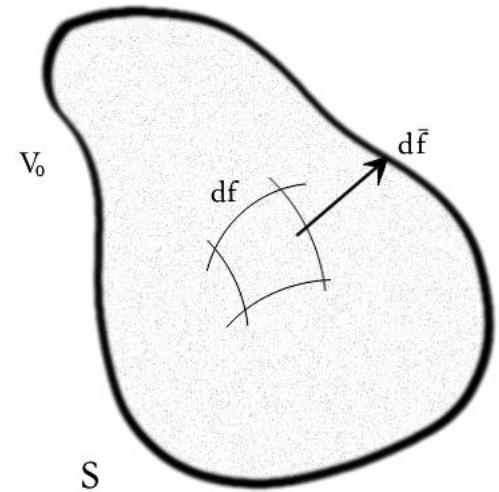
$$\frac{\partial p}{\partial x_i} = \delta_{ik} \frac{\partial \rho}{\partial x_k} .$$

Скорость жидкости в точке:

$$\frac{\partial}{\partial t} (\rho v_i) = -\frac{\partial \Pi_{ik}}{\partial x_k} , \quad \Pi_{ik} = p \delta_{ik} + \rho v_i v_k$$

Тензор плотности потока импульса в вязкой жидкости:

$$\Pi_{ik} = p \delta_{ik} + \rho v_i v_k - \sigma_{ik} = -\sigma_{ik} + \rho v_i v_k .$$





## Алгоритм метода сеток для консервативного вида уравнения Навье-Стокса (в декартовых координатах)

Система уравнений, описывающая плоское течение несжимаемой ньютоновой вязкой жидкости с постоянными свойствами при отсутствии внешних сил:

$$\begin{cases} \frac{\partial P}{\partial t} + \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0; \\ \frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} + v \frac{\partial u}{\partial y} = -\frac{1}{\hat{\rho}} \cdot \frac{\partial P}{\partial x} + \nu \left( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right); \\ \frac{\partial v}{\partial t} + u \frac{\partial v}{\partial x} + v \frac{\partial v}{\partial y} = -\frac{1}{\hat{\rho}} \cdot \frac{\partial P}{\partial y} + \nu \left( \frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right); \end{cases}$$

Где  $\hat{\rho} = \text{Re}$  и  $\nu = \frac{1}{\rho}$

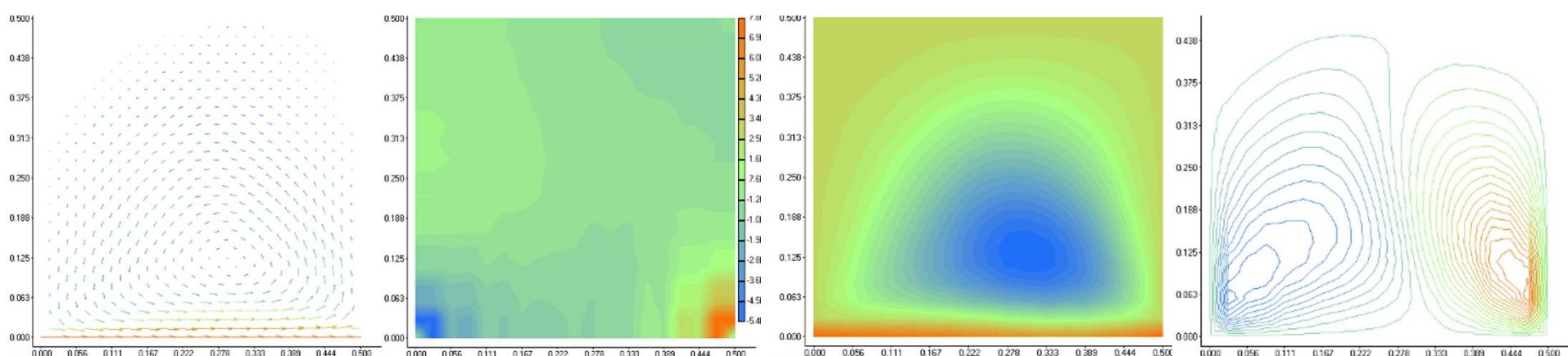
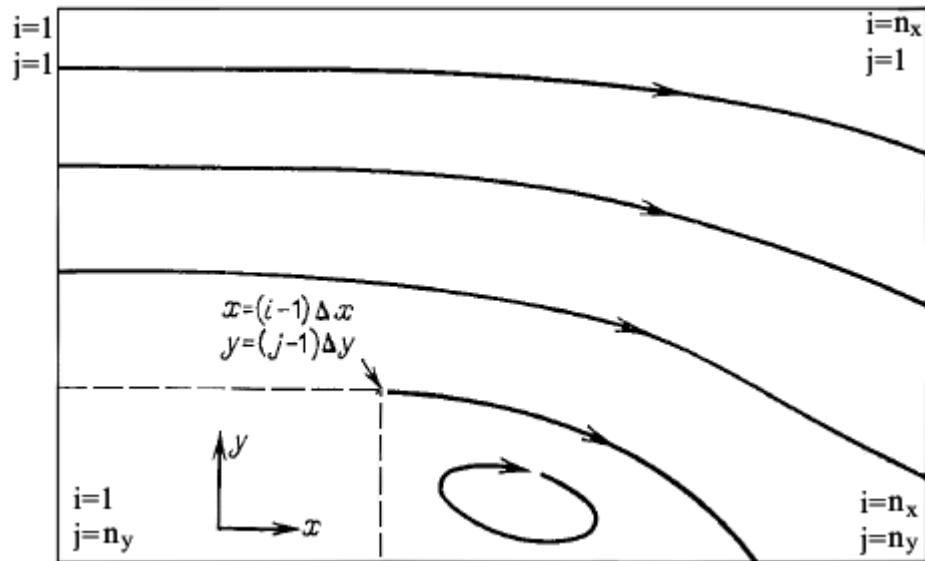
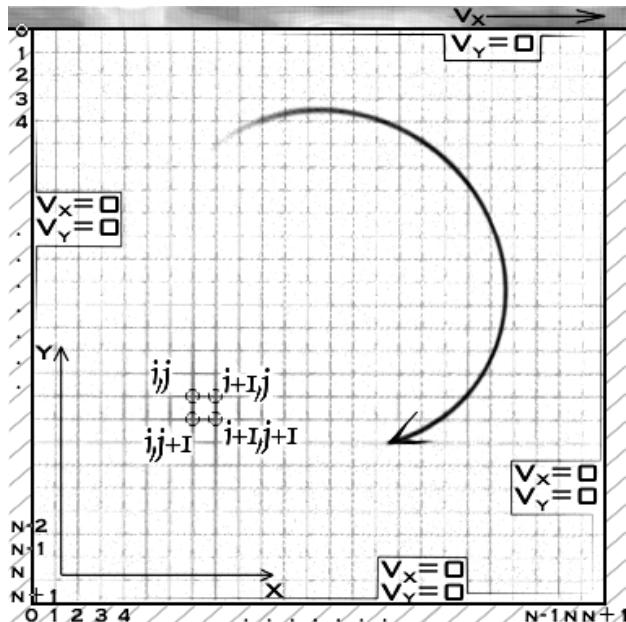


## Алгоритм метода сеток для консервативного вида уравнения Навье-Стокса (в декартовых координатах)

$$\left\{ \begin{array}{l} P(t+1, i, j) = -\Delta t \left( \frac{U(t, i+1, j) - U(t, i-1, j)}{2 \cdot \Delta x} + \frac{V(t, i, j+1) - V(t, i, j-1)}{2 \cdot \Delta y} \right) + P(t, i, j); \\ U(t+1, i, j) = -\frac{\Delta t}{Re} \cdot \frac{P(t, i+1, j) - P(t, i-1, j)}{2 \cdot \Delta x} + \\ \quad + \frac{\Delta t}{\rho} \left( \frac{U(t, i+1, j) - 2 \cdot U(t, i, j) + U(t, i-1, j)}{\Delta x^2} + \frac{U(t, i, j+1) - U(t, i, j) + U(t, i, j-1)}{\Delta y^2} \right) - \\ \quad - \Delta t \cdot \left( \frac{\partial(U^2)}{\partial x} - \frac{\partial(V \cdot U)}{\partial y} \right) + U(t, i, j); \\ V(t+1, i, j) = -\frac{\Delta t}{Re} \cdot \frac{P(t, i, j+1) - P(t, i, j-1)}{2 \cdot \Delta y} + \\ \quad + \frac{\Delta t}{\rho} \left( \frac{V(t, i+1, j) - 2 \cdot V(t, i, j) + V(t, i-1, j)}{\Delta x^2} + \frac{V(t, i, j+1) - V(t, i, j) + V(t, i, j-1)}{\Delta y^2} \right) - \\ \quad - \Delta t \cdot \left( \frac{\partial(U \cdot V)}{\partial x} + \frac{\partial(V^2)}{\partial y} \right) + V(t, i, j); \end{array} \right.$$



## Фазовый портрет





## Винеровский процесс

Разновидность марковского процесса, которая используется как отправная точка для определения стохастических процессов цен активов, это **основной процесс Винера** (basic Wiener process), или **геометрическое броуновское движение** (Geometric Brownian motion), названный в честь Норберта Винера, впервые сформулировавшего строгую математическую теорию для данного вида случайных процессов. В данном случае на исследуемую переменную воздействует большое количество случайных независимых импульсов или воздействий со стороны других переменных.



## Броуновское движение

Пусть  $S$  – рыночная цена фондового актива, а  $t$  – период времени. За малый промежуток времени  $\Delta t$  случайная переменная  $S$  изменится на  $\Delta S$ . Если  $S$  следует процессу Винера, т.е. броуновскому движению, изменение  $S$  за малый промежуток времени будет связано с  $\Delta t$  следующим соотношением:

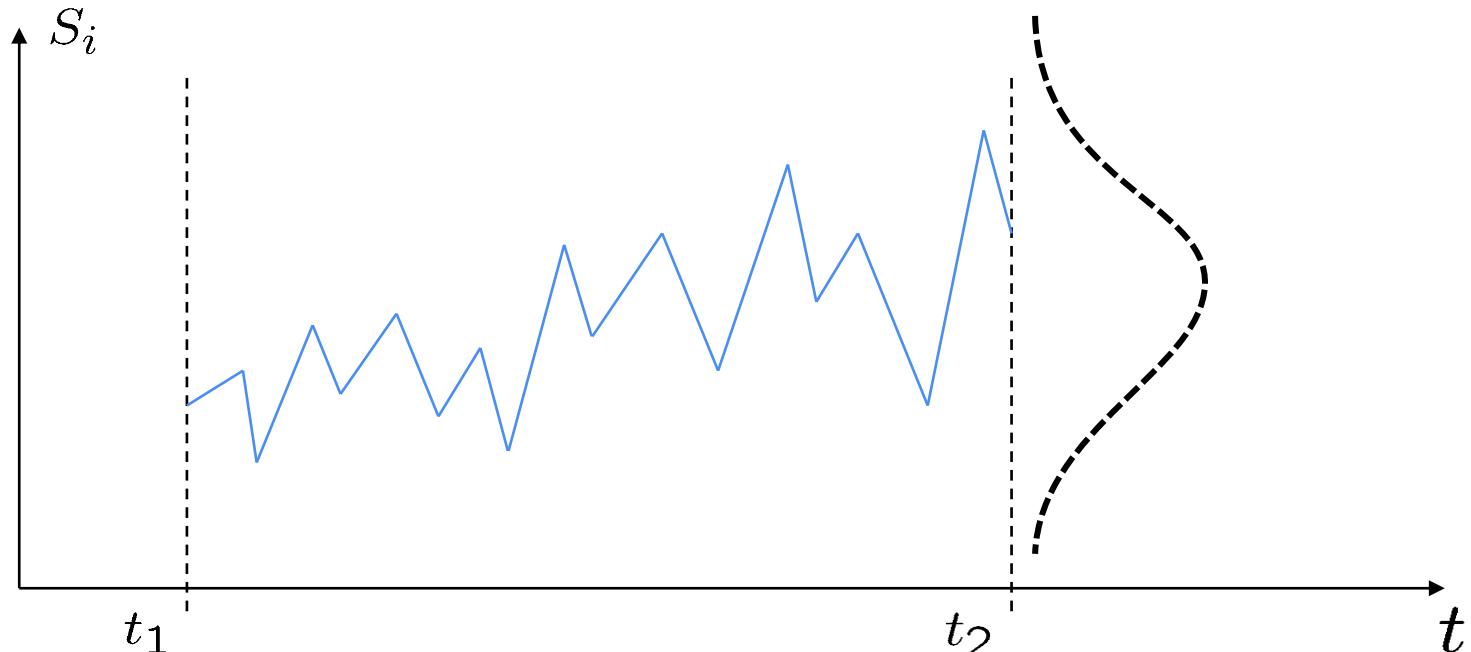
$$\Delta S = \varepsilon \times \sqrt{\Delta t} \text{ или}$$

$$dS = \varepsilon \times \sqrt{\Delta t}$$



## Геометрическое броуновское движение

Цену акции будем рассматривать как стохастический процесс.



Основное предположение: доходности распределены нормально

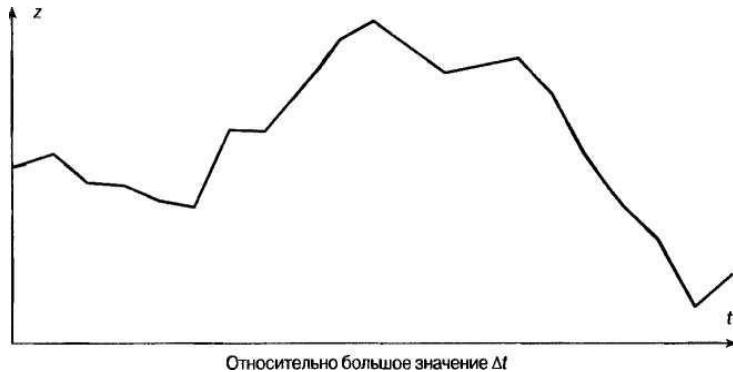


## Броуновское движение

- Стохастический процесс  $\{W_t : 0 \leq t \leq \infty\}$  является стандартным броуновским движением, если:
  1.  $W_0 = 0$ .
  2. Представляет собой непрерывную случайную последовательность.
  3. Имеет независимые, нормально распределенные приращения.



## Геометрическое броуновское движение





## Винеровский процесс

Инвестор не в состоянии точно предсказать доходность, какой бы ни была цена акции. Это значит, что стандартное отклонение изменений цены акции за короткий период времени  $\Delta t$  должно быть пропорциональным самой цене акции. Это приводит к следующей модели:

$$\frac{\Delta S}{S} = \mu \Delta t + \sigma \varepsilon \sqrt{\Delta t}$$

или

$$\Delta S = \mu S \Delta t + \sigma S \varepsilon \sqrt{\Delta t}$$

- |            |                        |
|------------|------------------------|
| $\Delta S$ | – изменение цены акций |
| $\mu$      | – ожидаемая доходность |
| $\sigma$   | – волатильность акций  |

- |                                      |   |
|--------------------------------------|---|
| $\sigma \varepsilon \sqrt{\Delta t}$ | – стохастическая компонента с дисперсией, равной $\sigma^2 T$               |
| $\varepsilon$                        | – случайная величина, имеющая стандартизированное нормальное распределение. |



## Винеровский процесс

- |                         |   |   |  |
|-------------------------|---|---|--|
| Изменение цены<br>акции | = | Ожидаемый рост с<br>текущим временем    | +<br>Влияние волатильности поведения<br>людей, "хаотически" приобретающих<br>и продающих акции, что оказывает<br>влияние на динамику цен |
| Изменение цены акции    | = | Скорость<br>детерминированного<br>тренд | + Стохастическая переменная  |
| Изменение цены акции    | = | Определенность (тренд)                  | + Неопределенность (обусловленная<br>волатильностью)   |



## Винеровский процесс

Случайный процесс  $W = \{W(t), t > 0\}$  называется Винеровским, если:

1.  $W(0; \Omega) = 0$  с вероятностью 1.
2. Случайные величины  $W(t_i; \Omega) - W(t_{i-1}; \Omega)$  (приращения Винеровского процесса) взаимонезависимы и распределены согласно нормальному закону с нулевым математическим ожиданием и дисперсией, равной  $(t_i - t_{i-1})$ .
3. Функция  $W(t; \Omega)$  непрерывна по переменной  $t$  для всех  $\Omega$ .



## Винеровский процесс

Инвестор не в состоянии точно предсказать доходность, какой бы ни была цена акции. Это значит, что стандартное отклонение изменений цены акции за короткий период времени  $\Delta t$  должно быть пропорциональным самой цене акции. Это приводит к следующей модели:

$$\frac{\Delta S}{S} = \mu \Delta t + \sigma \varepsilon \sqrt{\Delta t}$$

или

$$\Delta S = \mu S \Delta t + \sigma S \varepsilon \sqrt{\Delta t}$$

- |            |                        |
|------------|------------------------|
| $\Delta S$ | – изменение цены акций |
| $\mu$      | – ожидаемая доходность |
| $\sigma$   | – волатильность акций  |

- |                                      |   |
|--------------------------------------|---|
| $\sigma \varepsilon \sqrt{\Delta t}$ | – стохастическая компонента с дисперсией, равной $\sigma^2 T$               |
| $\varepsilon$                        | – случайная величина, имеющая стандартизированное нормальное распределение. |



## Винеровский процесс

$$\text{Будущая цена акции} = S e^{\left(r - \frac{\sigma^2}{2}\right)t + \sigma W_t}$$

Детерминированный тренд

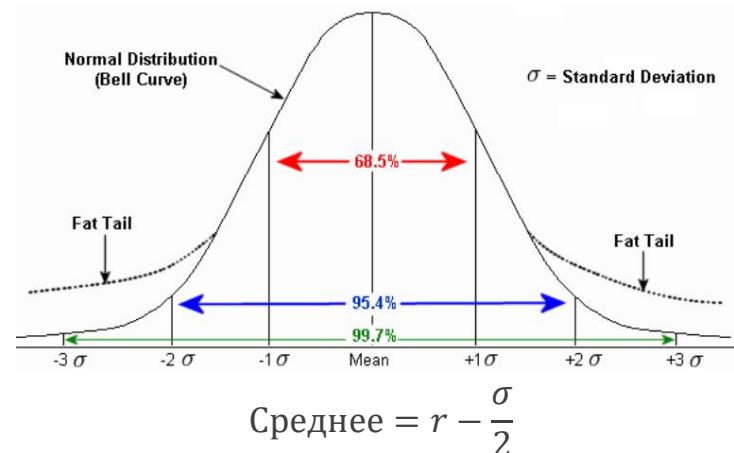
Стохастическая переменная

$S$  – текущая цена акции

$\sigma$  – процентная волатильность

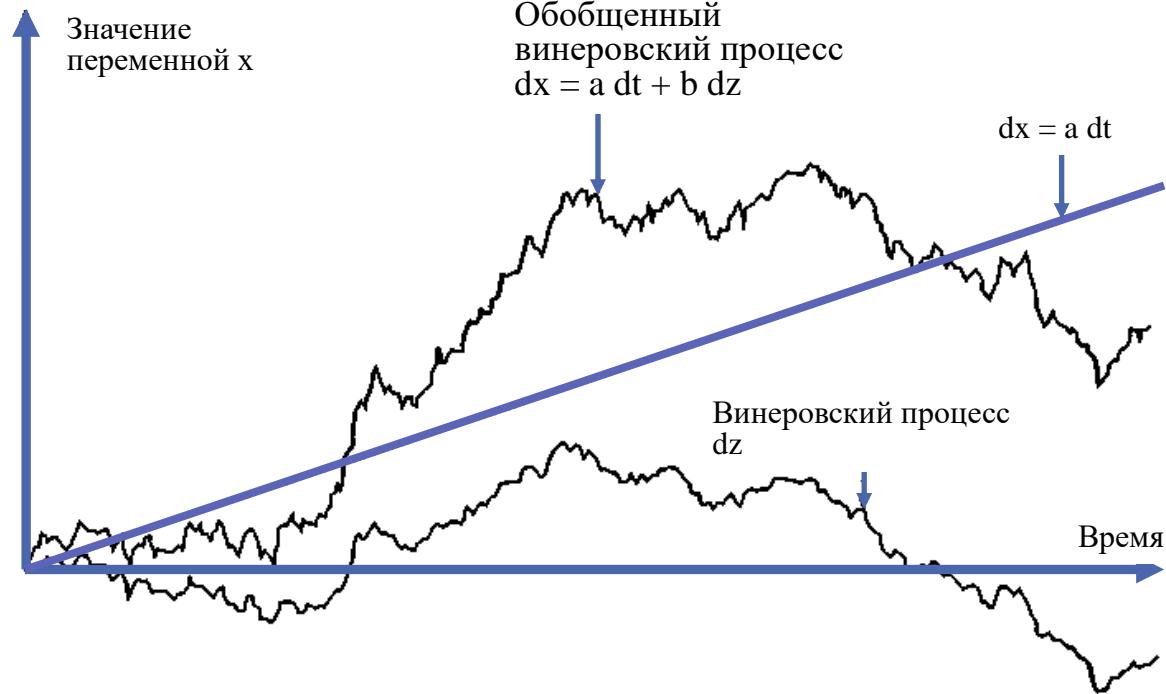
$r$  – безрисковая процентная ставка

$W_t$  – Броуновское движение





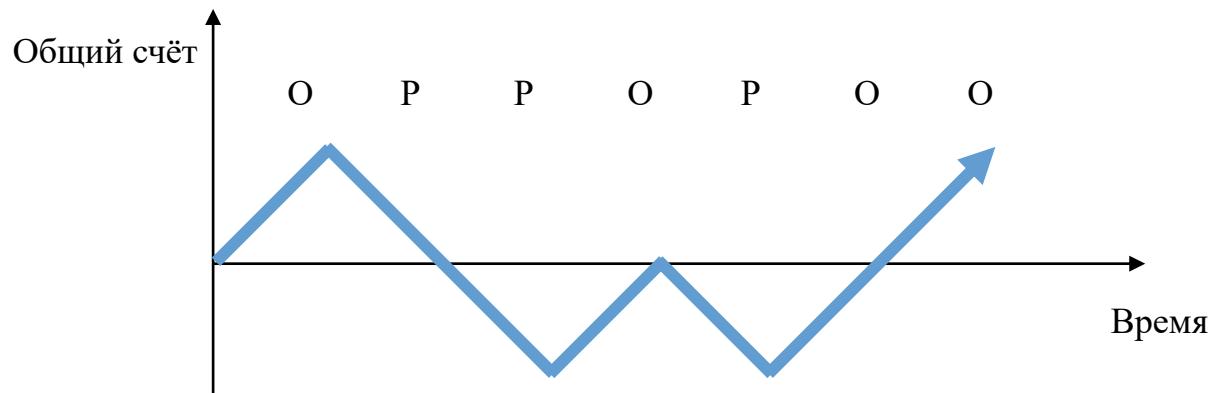
## Винеровский процесс





## Подбрасывание монеты

- Подбрасываем монету 1000 раз, прибавляя 1 всякий раз, когда выпадает «орёл», и вычитая 1 всякий раз, когда выпадает «решка», продолжая следить за общим счётом.
- Вопрос:** Какой конечный результат мы можем ожидать? Сколько раз общий счёт будет обнулён после 1000 подбрасываний?





## Подбрасывание монеты

Количество "обнулений" в первой половине: 38

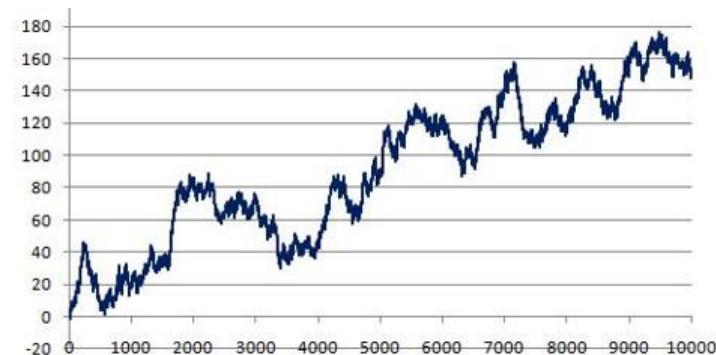
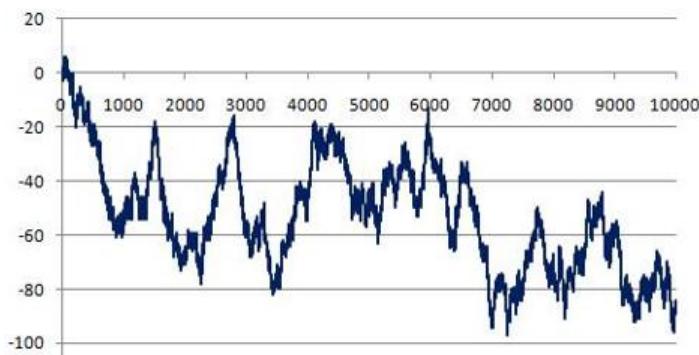
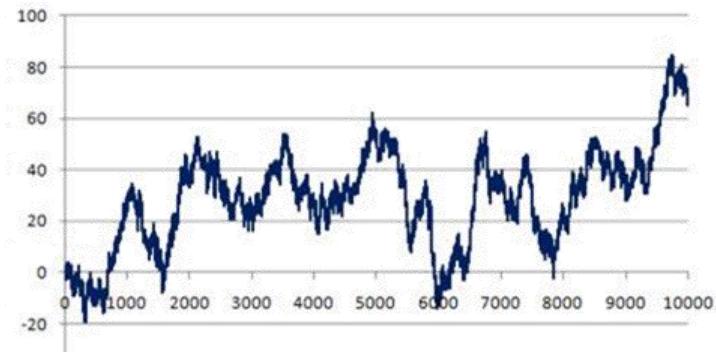
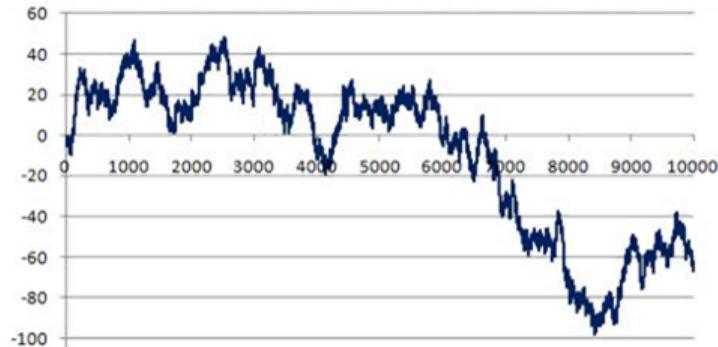
Количество "обнулений" во второй половине: 0

### Общий счёт



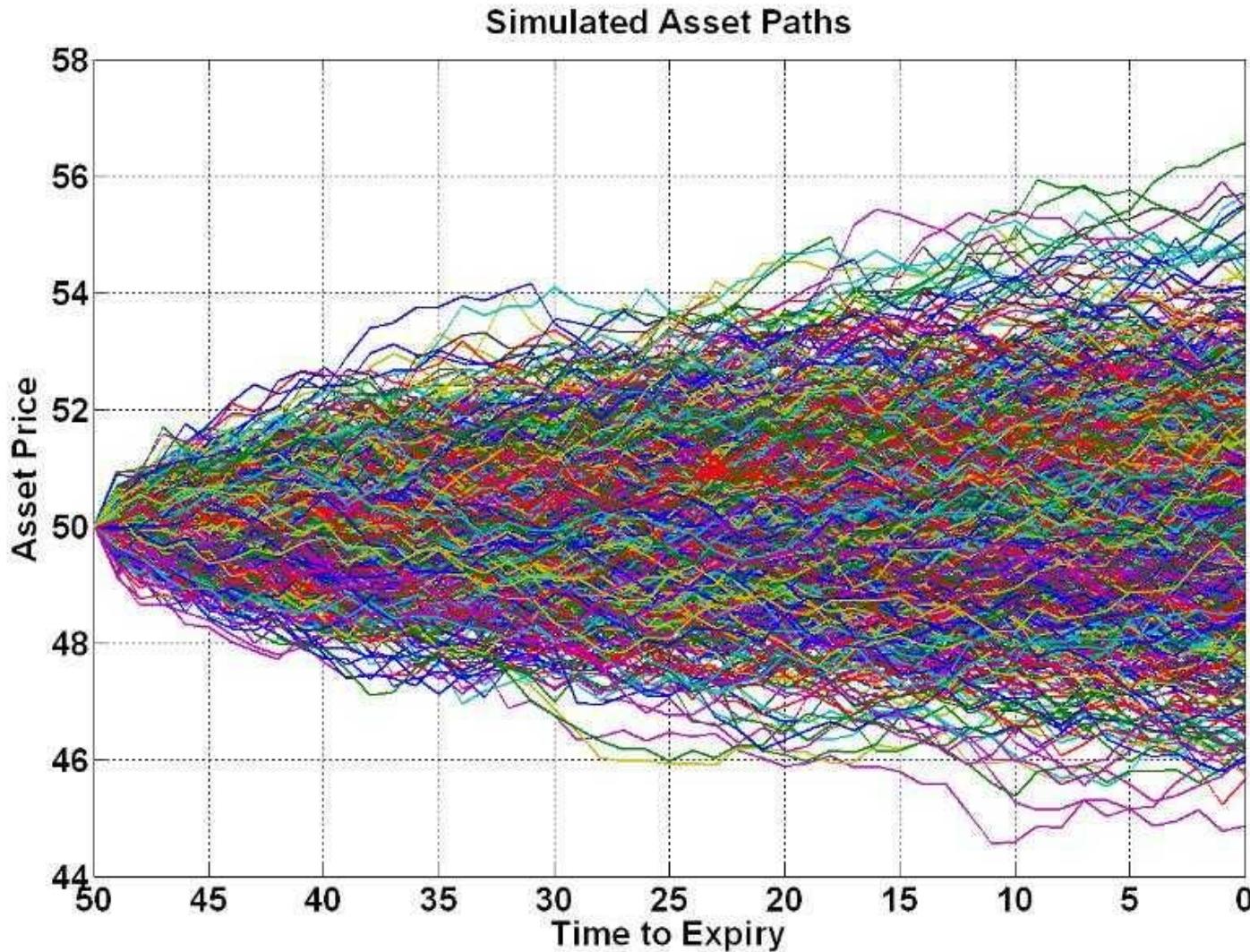


## Подбрасывание монеты





## Метод статистических испытаний (Монте-Карло)





## Моделирование спот-ставок

$$r(t) = f_{t,t} = f(0, t) + \int_0^t \mu(v, t) dv + \int_0^t \sigma_f(v, t) dW(v)$$

где

$$\frac{df(t, T)}{dt} = \mu(t, T) + \sigma_f(t, T) dW_t$$

Будучи по своей сути эволюционными, все модели, базирующиеся на винеровских процессах, эффективны лишь в случае небольших колебаний ставок, т.е. в условиях экономической стабильности и развитого финансового рынка.



## Моделирование спот-ставок

Расширим предыдущую модель путём включения компоненты, «отвечающей» за генерацию возмущений (скачков). В качестве такой компоненты может быть использован пуассоновский процесс. Тогда для общего случая подобная модель будет иметь следующий вид:

$$r(t) = \mu(t)dt + \sigma(t)dW(t) + \delta(t)dN(t)$$

где  $W$  – винеровский процесс

$N$  – пуассоновский процесс

$\delta$  – параметр пуассоновского процесса

Предлагаемый подход к моделированию временной структуры процентных ставок на базе диффузионных процессов, позволяет учитывать резкие колебания в развитии случайного процесса, что особенно соответствует реалиям российского рынка, отличительными чертами которого являются нестабильность, зависимость от неэкономических факторов, высокая степень неопределенности.



## Модель конкуренции с учётом территориальной расположности

Напомним:

$$\begin{cases} V'_t = b_V \cdot V \\ U'_t = b_U \cdot U \end{cases}$$

– система уравнений Мальтуса.

Предположим, популяции  $U$  и  $V$  – популяции каких-либо двух различных видов растений. Тогда  $b_V$  и  $b_U$  (коэффициенты Мальтуса), в этом случае, представляют собой разницу между коэффициентами вегетативного размножения и смертности. Следовательно биомасса популяций будут равномерно увеличиваться ( $b_V > 0$ ), ( $b_U > 0$ ) или уменьшаться ( $b_V < 0$ ), ( $b_U < 0$ ), не выходя за рамки своей области обитания, что приведет либо к полному вымиранию, либо к тому что биомасса популяции уйдет на бесконечность.



## Модель конкуренции с учётом территориальной расположенности

Напомним:

$$\begin{cases} V'_t = b_V \cdot V - d_V \cdot V^2 \\ U'_t = b_U \cdot U - d_U \cdot U^2 \end{cases}$$

– система логистических уравнений.

Появившиеся элементы  $d_V \cdot V^2$  и  $d_U \cdot U^2$  – ограничители роста. Если мы рассматриваем взаимодействие растительных популяций, то эта система будет предусматривать максимальное количество биомассы (количество особей) на единицу пространства.

$$\begin{cases} V'_t = b_V \cdot V - d_V \cdot V^2 - c_U \cdot U \cdot V \\ U'_t = b_U \cdot U - d_U \cdot U^2 - c_V \cdot U \cdot V \end{cases}$$

– система уравнений конкуренции.

$c_U$  и  $c_V$  - коэффициенты конкуренции (показывают насколько одна популяция угнетает другую). Коэффициенты начинают играть роль, когда на одном и том же участке пространства обитают одновременно два вида растений.



## Модель конкуренции с учётом территориальной расположности

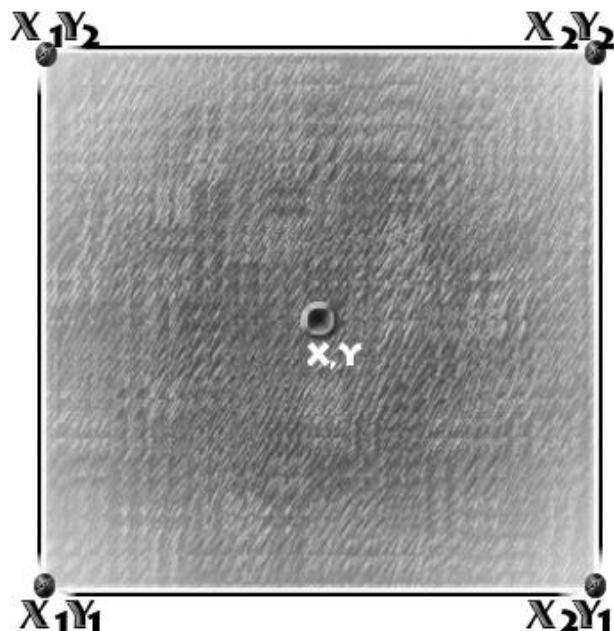
$$\begin{cases} V'_t = b_V * V - d_V * V^2 - c_U * U * V + \int_{y-Dy_1}^{y+Dy_1} \int_{x-Dx_1}^{x+Dx_1} K_U(x, y) * V(x, y, t) dx dy \\ U'_t = b_U * U - d_U * U^2 - c_V * U * V + \int_{y-Dy_2}^{y+Dy_2} \int_{x-Dx_2}^{x+Dx_2} K_V(x, y) * U(x, y, t) dx dy \end{cases}$$

ядра  $K_U(x, y)$  и  $K_V(x, y)$  – функции плотности вероятности произрастания семян – при создании этой модели полагается, что почва на всей области – неоднородная и в некоторых местах более пригодна для заселения одной популяцией, а в некоторых менее.  $U(x, y, t)$  и  $V(x, y, t)$  – плотность популяции в момент времени  $t$  в точке с координатами  $(x; y)$ .  $Dy_1$  и  $Dx_1$  – величина (расстояние) на которое распространяются семена.



## Модель конкуренции с учётом территориальной расположности

При помощи интегралов описывается процесс полноценного заселения растениями. Таким образом, на каждом шаге, каждая совокупность особей, занимающая элементарную площадку, заселяет всю лежащую на расстоянии  $Dy_1$  по у и  $Dx_1$  по х область.



Для вычисления двойного интеграла может быть применён метод трапеций:

$$\int_{X_1}^{X_2} F(x)dx \approx dx \left( \frac{F(X_1) + F(X_2)}{2} + \sum_{i=X_1+Dx}^{X_2-Dx} F(x_i) \right)$$



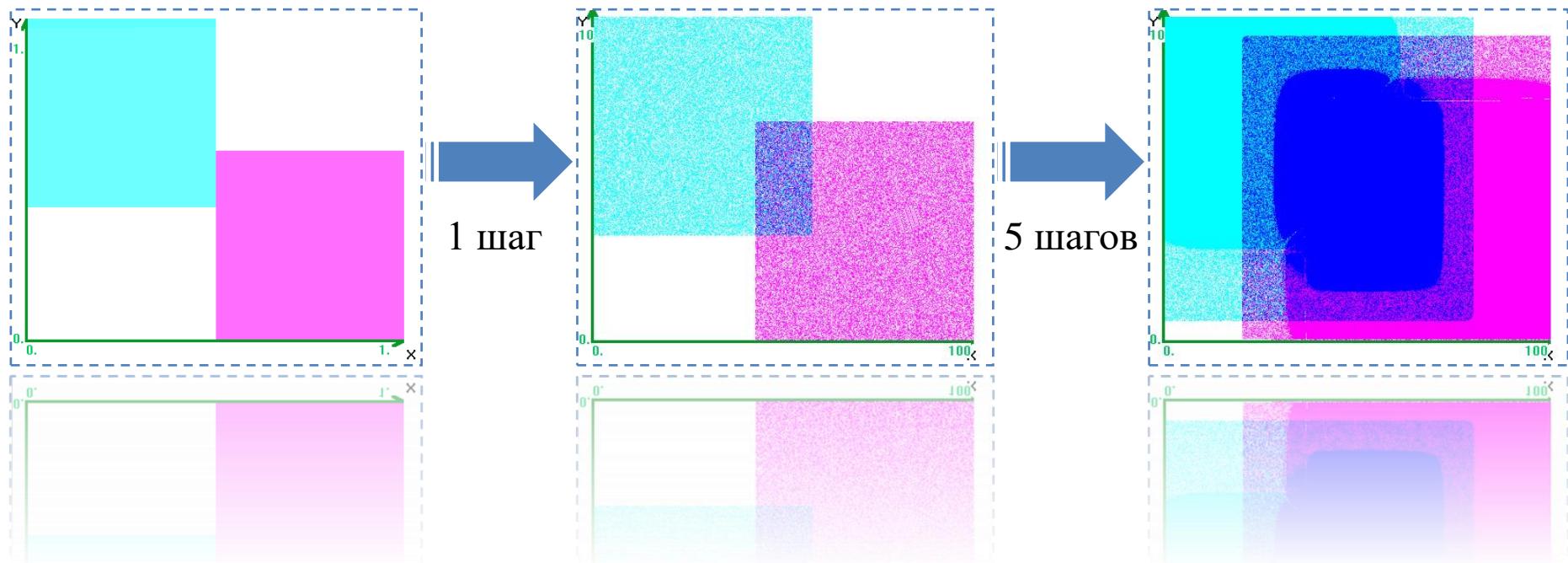
## Модель конкуренции с учётом территориальной расположности

$$\begin{cases} V'_t = b_V * V - d_V * V^2 - c_U * U * V + \int_{y-Dy_1}^{y+Dy_1} \int_{x-Dx_1}^{x+Dx_1} K_U(x, y) * V(x, y, t) dx dy \\ U'_t = b_U * U - d_U * U^2 - c_V * U * V + \int_{y-Dy_2}^{y+Dy_2} \int_{x-Dx_2}^{x+Dx_2} K_V(x, y) * U(x, y, t) dx dy \end{cases}$$

Таким образом, биомасса некой популяции на определенной элементарной площадке с координатами (x; y) в определенный момент времени зависит от значения в предыдущий момент времени, количества представителей популяции-конкурента на этой же площадке, количества представителей той же популяции в близлежащей области, месторасположения популяции, и конечно основных характеристик популяции (разница между коэффициентами вегетативного размножения и смертности и т.д. и т.п.).



## Модель конкуренции с учётом территориальной расположенности



**Опр:** Линейным разностным уравнением  $n$ -го порядка с постоянными коэффициентами называется уравнение вида

$$\sum_{k=1}^{N+1} a_{n+k} y^{n+k} = b_n f^n, \quad a_{n+k}, b_n = \text{const}$$

Если  $b_n = 0$ , то уравнение называется **однородным**

Решение разностного уравнения представимо в виде:

$$y^n = y_{\text{общ}}^n + y_{\text{ч}}^n$$

↑                      ↑  
Общее решение      Частное решение  
однородного      неоднородного  
уравнения      уравнения

Общее решение состоит из линейной комбинации решений разностного уравнения:

$$y_{\text{общ}}^n = \sum_{i=1}^N C_j y_{0j}^n$$

Будем искать общее решение однородного уравнения в виде

$$y_{\text{общ}}^n = \lambda^n$$

Уравнение вида:

$$a_n + a_{n+1}\lambda + a_{n+2}\lambda^2 + \cdots + a_{n+k}\lambda^k = 0$$

Называется **характеристическим** уравнением

$$y_{\text{общ}}^n = \sum_{i=1}^N c_j \lambda_j^n \quad , n = 0 \dots N$$

Проверим найденное решение

$$\sum_{j=1}^N c_j \lambda_j^{n-k} (a_n + a_{n+1} \lambda_j + \cdots + a_{n+k} \lambda_j^k) = 0$$

Верно  $\forall c_j$

Если какой-нибудь корень  $\lambda_j$  имеет кратность  $s$ ,  
то соответствующая частичная сумма

$$\sum_{p=0}^{s-1} c_{j+p} \lambda_{j+p}^n$$

заменяется выражением

$$(c_j + n c_{j+1} + \cdots + n^{s-1} c_{j+s-1}) \lambda_j^n$$



К полученному общему решению однородной подсистемы нужно добавить частное решение неоднородной подсистемы. Его вид зависит от вектора  $f$  правой части. Пусть они равны:

$$f_n = \mu^n (P_\alpha(n) \cos(\nu n) + R_\beta(n) \sin(\nu n))$$

$P_\alpha(n)$ ,  $R_\beta(n)$  - многочлены степеней  $\alpha$  и  $\beta$   
соответственно     $\mu$  и  $\nu$  – действительные числа

$$y_\eta^n = \mu^n (Q_\delta(n) \cos(\nu n) + S_\delta(n) \sin(\nu n))$$

$Q_\delta(n)$ ,  $S_\delta(n)$  - многочлены степени  $\delta = \max(\alpha, \beta)$



$$f_n = \mu^n (P_\alpha(n) \cos(\nu n) + R_\beta(n) \sin(\nu n))$$

Для случая, когда  $\mu$  – модуль, а  $\nu$  – аргумент какого-то одного корня характеристического уравнения полагаем

$$y_\chi^n = n^r (Q_\delta(n) \cos(\nu n) + S_\delta(n) \sin(\nu n))$$

$r$  – кратность корня

Найти решение разностного уравнение

$$y^{n+1} - 3y^n + 3y^{n-1} - y^{n-2} = ah^4n,$$
$$a, h = \text{const}$$

Найдем общее решение однородного уравнения

Подставляем в уравнение  $y_{\text{общ}}^n = \lambda^n$

Получаем характеристическое уравнение

$$\lambda^3 - 3\lambda^2 + 3\lambda - 1 = 0$$



$$\lambda_{123} = 1 \text{ - Корень кратности 3}$$

$$y_{\text{общ}}^n = (C_1 + C_2 n + C_3 n^2) 1^n$$

Найдем частное решение

$$f_n = \mu^n (P_\alpha(n) \cos(\nu n) + R_\beta(n) \sin(\nu n))$$

В правой части  $\mu = 1, \nu = 0$



Совпадают с модулем и аргументом  $\lambda$

Будем искать решение в виде

$$y_n = (A + Bn)n^3$$

Подставляем в неоднородное уравнение

$$\begin{aligned} & A(n+1)^4 + B(n+1)^3 - 3An^4 - 3Bn^3 + \\ & + 3A(n-1)^4 + 3B(n-1)^3 - A(n-2)^4 - \\ & - B(n-2)^3 = 12A(2n-1) + 6B = ah^4n \end{aligned}$$

$$A = ah^4/24 \quad B = ah^4/12$$

$$\begin{aligned} y^n &= y_{\text{общ}}^n + y_{\text{ч}}^n = \\ &= (C_1 + C_2n + C_3n^2)1^n + ah^4n^3(n+2)/24 \end{aligned}$$



ОДУ не обязательно должно быть записано в виде уравнения первого порядка, например

$$z'' + z' + 1 = 0, \quad z(0) = a, \quad z'(0) = b.$$

Можно преобразовать задачу с использованием вспомогательных переменных, таких как  $w = z'$ . В итоге:

$$\begin{pmatrix} w \\ z \end{pmatrix}' = \begin{pmatrix} -w - 1 \\ w \end{pmatrix}, \quad z(0) = a, \quad w(0) = b$$

Не всегда стоит использовать простые производные в качестве вспомогательных переменных. Пример:

$$\frac{1}{r^2} \left( r^2 P' / \rho \right)' = -4\pi\rho$$

Используем  $P(r)$  и вспомогательную переменную

$$\Pi(r) = r^2 P' / \rho$$



Предположим, что значения  $f(x)$  известны в точках  $\{x_j\}$ . Один из подходов использует эти значения для вычисления производной. Пример: предположим, что  $f(x)$  известен в точке  $x_0$  как и  $x_0 \pm h$  для некоторого небольшого шага  $h$ .

Подберём линейный многочлен  $g$  (прямая линия) к значениям  $f(x)$  в точках  $\{x_0, x_0 + h\}$ : производная на этом интервале – наклон линии.

$$f'(x_0) \simeq g'(x_0) = \frac{f(x_0 + h) - f(x_0)}{h}$$

Это приближённое значение производной с использованием прямой разности.

Аналогично, линейное приближение на интервале  $[x_0 - h, x_0]$  даёт:

$$f'(x_0) \simeq g'(x_0) = \frac{f(x_0) - f(x_0 - h)}{h}$$

Это приближённое значение производной с использованием обратной разности.



$$f'(x_0) \simeq g'(x_0) = \frac{f(x_0 + h) - f(x_0)}{h}$$

Точность аппроксимации зависит от выбора интерполирующего многочлена.  
Проверим точность, используя разложение Тейлора:

$$\begin{aligned} f(x_0 + h) &= f(x_0) + hf'(x_0) + \frac{h^2}{2!}f''(x_0) + \mathcal{O}(h^3) \\ \Rightarrow \frac{f(x_0 + h) - f(x_0)}{h} &= \frac{f(x_0) + hf'(x_0) + \frac{h^2}{2!}f''(x_0) + \mathcal{O}(h^3) - f(x_0)}{h} \\ &= f'(x_0) + \mathcal{O}(h). \end{aligned}$$

Тот же результат по точности справедлив для

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



## ПОСТАНОВКА ЗАДАЧИ 1 ЧИСЛЕННОГО ДИФФЕРЕНЦИРОВАНИЯ

Пусть на отрезке  $[a, b]$  на равномерной сетке  $\Omega_n (h_{i+1} = h = \text{const})$  заданы:

- а) сеточная функция  $y_i = f(x_i)$ ,  $i = 0, n$ , своими значениями  $f_i = f(x_i)$ ;
- б) точки  $x_j$  сетки  $\Omega_n$ , в которых требуется найти значения производных;
- в) желаемый порядок  $t$  точности (аппроксимации) относительно  $h$ .

Требуется с заданным порядком точности (аппроксимации) вычислить значения производных  $\hat{f}^{(p)}(x)|_{x=x_j}$  в точках  $x_j$  сетки, где  $p$  — порядок производной.

Иначе, требуется получить аппроксимационный оператор  $\hat{f}^{(p)}(x_j)$ , удовлетворяющий условию  $|f^{(p)}(x_j) - \hat{f}^{(p)}(x_j)| \leq Ch^t$ , где  $C = \text{const}$ , не зависящая от величины шага  $h$ .

ПОСТАНОВКА ЗАДАЧИ 2  
ЧИСЛЕННОГО ДИФФЕРЕНЦИРОВАНИЯ

Пусть на отрезке  $[a, b]$  в общем случае на неравномерной сетке  $\Omega_n = \{x_0, x_1, \dots, x_n\}$ ,  $x_{i+1} = x_i + h_{i+1}$ ,  $i = \overline{0, n-1}$ ,  $h_{i+1} = x_{i+1} - x_i$  ( $h_{i+1} = \text{var}$ ) заданы:

- а) сеточная функция  $y_i = f(x_i)$ ,  $i = \overline{0, n}$ , своими значениями  $f_i = f(x_i)$  и/или значениями интегралов  $I_i^{i+1} = \int_{x_i}^{x_{i+1}} f(x) dx$  на частичных отрезках  $[x_i, x_{i+1}]$ ;
- б) точки  $x_j$  сетки  $\Omega_n$ , в которых требуется найти значения производных;
- в) желаемый порядок  $t$  точности (аппроксимации) относительно величины шага.

Требуется с заданным порядком точности (аппроксимации) вычислить значения производных  $\hat{f}^{(p)}(x)|_{x=x_j}$  в точках  $x_j$  сетки (получить аппроксимационный оператор), где  $p$  — порядок производной.



**Двухточечный шаблон.** Выберем шаблон  $\mathbb{W}_{2,i} = (x_i, x_{i+1})$  на неравномерной сетке  $\Omega_n$ . Предполагая, что  $f(x) \in C_2[a, b]$ , разложим функцию  $f(x)$  по формуле Тейлора при  $k = 1$  относительно точки  $x_i$  с остаточным слагаемым в форме Лагранжа и найдем выражение для  $f_{i+1} = f(x_{i+1})$ :

$$f_{i+1} = f_i + h_{i+1}f'_i + \frac{h_{i+1}^2}{2}f''(\xi),$$

где  $\xi \in (x_i, x_{i+1})$ ,  $h_{i+1} = x_{i+1} - x_i$ . Отсюда получаем

$$f'_i = \frac{f_{i+1} - f_i}{h_{i+1}} - \frac{h_{i+1}}{2}f''(\xi).$$

Очевидно, справедлива оценка

$$\left| -\frac{h_{i+1}}{2}f''(\xi) \right| \leq \frac{h_{i+1}}{2} \max_{[x_i, x_{i+1}]} |f''(x)| = \frac{h_{i+1} \cdot M_{2,i}}{2},$$

где  $M_{2,i} = \max_{[x_i, x_{i+1}]} |f''(x)|$ .

Отсюда следует *функциональная формула* (функциональный оператор) для первой производной:

$$\hat{f}'_{i,c} = \frac{f_{i+1} - f_i}{h_{i+1}} - \left( \frac{h_{i+1}}{2}M_{2,i} \right).$$

**Трехточечный шаблон.** На неравномерной сетке  $\Omega_n$  выбираем трехточечный (двуухшаговый) шаблон  $\mathbb{W}_{3,i} = (x_{i-1}, x_i, x_{i+1})$ , характеризующийся шагами  $h_{i+1} = x_{i+1} - x_i$ ,  $h_i = x_i - x_{i-1}$  и параметром его нерегулярности  $\delta_{i+1} = \frac{h_{i+1}}{h_i}$ , который в общем случае не равен единице.

Аппроксимационные функциональные (точечные) формулы второго порядка в левой крайней, центральной и правой крайней точках шаблона можно получить на основе разложения функции  $f(x)$  по формуле Тейлора с остаточным слагаемым в форме Лагранжа. При этом предполагается, что  $f(x) \in C_3[a, b]$ . Это позволяет получить разностные дифференциальные операторы  $\hat{f}'(x_t)$  ( $t = i-1, i, i+1$ ) и провести оценки их погрешностей.

Разложим функцию  $f(x)$  при  $x = x_i$  и  $x = x_{i+1}$  по формуле Тейлора при  $k = 2$  относительно точки  $x_{i-1}$  с остаточным слагаемым в форме Лагранжа. В результате находим соотношения, определяющие  $f_i = f(x_i)$  и  $f_{i+1} = f(x_{i+1})$ :

$$f_i = f_{i-1} + h_i f'_{i-1} + \frac{h_i^2}{2} f''_{i-1} + \frac{h_i^3}{6} f'''(\xi_-);$$

$$f_{i+1} = f_{i-1} + H_i^{i+1} f'_{i-1} + \frac{(H_i^{i+1})^2}{2} f''_{i-1} + \frac{(H_i^{i+1})^3}{6} f'''(\xi_+),$$

где  $H_i^{i+1} = h_i + h_{i+1}$ ,  $\xi_- \in (x_{i-1}, x_i)$ ,  $\xi_+ \in (x_{i-1}, x_{i+1})$ ,  $f_{i-1}^{(p)} = f^{(p)}(x_{i-1})$ ,  $p = 0, 1, 2$ .



Приводимые здесь и ниже остаточные слагаемые для дифференциальных операторов также получаются путем дифференцирования остаточных слагаемых  $R(x)$  интерполяционных многочленов соответствующей степени. Для трехточечных формул при  $n = 2$  следует соотношение

$$R(x) = \frac{1}{3!} f'''(\xi) \cdot \omega(x), \quad \omega(x) = (x - x_{i-1})(x - x_i)(x - x_{i+1})$$

Аналогично, разложив функцию  $f(x)$  относительно точки  $x_{i+1}$  и получив соотношения для  $f_{i-1}$ ,  $f_i$ , найдем  $\hat{f}'_{i+1}$  — разностный оператор, аппроксимирующий первую производную  $f'_{i+1}$  в правой крайней точке (правосторонняя формула):

$$\hat{f}'_{i+1,v} = \frac{1}{H_i^{i+1}} \left( \delta_{i+1} f_{i-1} - \frac{(1 + \delta_{i+1})^2}{\delta_{i+1}} f_i + \frac{2\delta_{i+1} + 1}{\delta_{i+1}} f_{i+1} \right);$$

$$\hat{f}'_{i+1,c} = \frac{1}{2h} (f_{i-1} - 4f_i + 3f_{i+1}) \text{ или } \hat{f}'_{i+1,c} = \frac{1}{2h} (3\Delta f_i - \Delta f_{i-1}) \quad \left( \frac{h^2}{3} M_{3,i} \right).$$



Разложение функции  $f(x)$  относительно центральной точки  $x_i$  шаблона, получение выражений для  $f_{i-1}$ ,  $f_{i+1}$  и исключение из них слагаемого со второй производной приводят к следующим разностным операторам функционального типа, аппроксимирующими *первую производную в центральной точке* (формула центрального вида):

$$\begin{aligned}\hat{f}'_{i,v} &= \frac{1}{H_i^{i+1}} \left( \delta_{i+1} \Delta f_{i-1} + \frac{1}{\delta_{i+1}} \Delta f_i \right) = \frac{1}{H_i^{i+1}} \left( -\delta_{i+1} f_{i-1} + \frac{(\delta_{i+1}^2 - 1)}{\delta_{i+1}} f_i + \frac{1}{\delta_{i+1}} f_{i+1} \right); \\ \hat{f}'_{i,c} &= \frac{1}{2h} (f_{i+1} - f_{i-1}) \text{ или } \hat{f}'_{i,c} = \frac{1}{2h} (\Delta f_i + \Delta f_{i-1}) \quad \left( \frac{h^2}{6} M_{3,i} \right).\end{aligned}$$

Оператор  $\hat{f}'_{i,v}$  имеет остаточное слагаемое  $\frac{h_i^2}{6} \delta_{i+1} f'''(\xi)$ .

Приведенные остаточные слагаемые разностных операторов обусловливают следующие оценки их погрешностей:

$$\begin{aligned}|f'_{i-1} - \hat{f}'_{i-1,v}| &\leq \frac{h_i^2}{6} (1 + \delta_{i+1}) M_{3,i}, \quad |f'_i - \hat{f}'_{i,v}| \leq \frac{h_i^2}{6} \delta_{i+1} M_{3,i}; \\ |f'_{i+1} - \hat{f}'_{i+1,v}| &\leq \frac{h_i^2}{6} (1 + \delta_{i+1}) \delta_{i+1} M_{3,i}, \quad M_{3,i} = \max_{x \in \Pi_{3,i}} |f'''(x)|.\end{aligned}$$



Метод Рунге-Кутты используют для расчета стандартных моделей достаточно часто, так как при небольшом объеме вычислений он обладает хорошей точностью.

Для построения разностной схемы интегрирования воспользуемся разложением искомой функции  $y(x)$  в ряд Тейлора:

$$y(x_{k+1}) = y(x_k) + y'(x_k)h + y''(x_k) \frac{h^2}{2} + \dots$$

Заменим вторую производную в этом разложении выражением

$$y''(x_k) = (y'(x_k))' = f'(x_k, y(x_k)) \approx \frac{f(\tilde{x}, \tilde{y}) - f(x_k, y(x_k))}{\Delta x},$$

где  $\tilde{x} = x_k + \Delta x$ ;  $\tilde{y} = y(x_k + \Delta x)$ .



**Методе Рунге-Кутты** является широко используемым при интегрировании обыкновенных дифференциальных уравнений.

По умолчанию речь идёт о методе Рунге-Кутты четвёртого порядка точности – из-за большой распространённости данного метода, указания на тип и порядок зачастую опускаются. При этом существуют еще методы первого, второго и третьего порядка точности.



$\Delta x$  подбирается из условия достижения наибольшей точности записанного выражения.

Для дальнейших выкладок произведем замену  $\tilde{y}$  разложением в ряд Тейлора:

$$\tilde{y} = y(x_k + \Delta x) = y(x_k) + y'(x_k)\Delta x + \dots$$

Введем обозначение:  $y_k = y(x_k)$ . Тогда для исходного уравнения построим вычислительную схему:

$$y_{k+1} = y_k + f(x_k, y_k)h + \frac{h^2}{2\Delta x} \left( f(x_k + \Delta x, y_k + y'_k \Delta x) - f(x_k, y_k) \right)$$

которую преобразуем к виду:

$$\begin{aligned} y_{k+1} &= y_k + h \left[ \left( 1 - \frac{h}{2\Delta x} \right) f(x_k, y_k) + \frac{h}{2\Delta x} f(x_k + \Delta x, y_k + y'_k \Delta x) \right] = \\ &= y_k + h \left[ \left( 1 - \frac{h}{2\Delta x} \right) f(x_k, y_k) + \frac{h}{2\Delta x} f \left( x_k + \frac{\Delta x}{h} h, y_k + f(x_k, y_k) \frac{\Delta x}{h} h \right) \right] \end{aligned}$$



$$y_{k+1} = y_k + h \left[ \left(1 - \frac{h}{2\Delta x}\right) f(x_k, y_k) + \frac{h}{2\Delta x} f \left( x_k + \frac{\Delta x}{h} h, y_k + f(x_k, y_k) \frac{\Delta x}{h} h \right) \right]$$

Введем следующие обозначения:

$$\alpha = \frac{h}{2\Delta x}, \quad \beta = 1 - \frac{h}{2\Delta x}, \quad \gamma = \frac{\Delta x}{h}, \quad \delta = f(x_k, y_k) \frac{\Delta x}{h}.$$

Эти обозначения позволяют записать предыдущее выражение в форме

$$y_{k+1} = y_k + h[\beta f(x_k, y_k) + \alpha f(x_k + \gamma h, y_k + \delta h)]$$

Очевидно, что все введенные коэффициенты зависят от величины  $\Delta x$  и могут быть определены через коэффициент  $\alpha$ , который в этом случае играет роль параметра:

$$\beta = 1 - \alpha, \quad \gamma = \frac{1}{2\alpha}, \quad \delta = f(x_k, y_k) \frac{2}{\alpha}$$

Окончательно схема Рунге-Кутта принимает вид:

$$y_{k+1} = y_k + h \left[ (1 - \alpha) f(x_k, y_k) + \alpha f \left( x_k + \frac{h}{2\alpha}, y_k + f(x_k, y_k) \frac{2h}{\alpha} \right) \right].$$



$$y_{k+1} = y_k + h \left[ (1 - \alpha)f(x_k, y_k) + \alpha f\left(x_k + \frac{h}{2\alpha}, y_k + f(x_k, y_k) \frac{2h}{\alpha}\right)\right].$$

При  $\alpha = 0$  получаем как частный случай уже известную схему Эйлера:

$$y_{k+1} = y_k + hf(x_k, y_k).$$

При  $\alpha = 0,5$  получаем как частный случай метод Рунге-Кутта второго порядка / схему Эйлера с пересчетом.

$$y_{k+1} = y_k + \frac{h}{2} [f(x_k, y_k) + f(x_k + h, y_k + hf(x_k, y_k))].$$



Классический метод Рунге-Кутты 2-го порядка, он же Метод Эйлера с пересчетом, описывается следующим уравнением:

$$y_i = y_{i-1} + h \cdot f(x_i, y_i)$$

Схема является неявной, так как искомое значение  $y_i$  входит в обе части уравнения.

Затем вычисляют значение производной в точке  $(x_i, y_i)$  и окончательно полагают:

$$y_i = y_{i-1} + h \cdot \frac{f(x_{i-1}, y_{i-1}) + f(x_i, y_i^*)}{2}$$

то есть усредняют значения производных в начальной точке и в точке “грубого приближения”. Окончательно запишем рекуррентную формулу метода Рунге-Кутты 2-го порядка в следующем виде:

$$y_i = y_{i-1} + \frac{h}{2} \cdot (K_1 + K_2)$$

где:

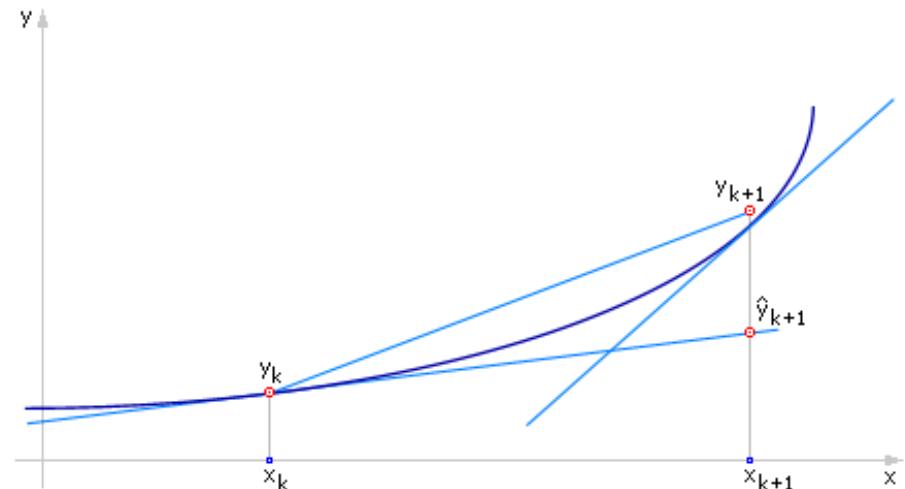
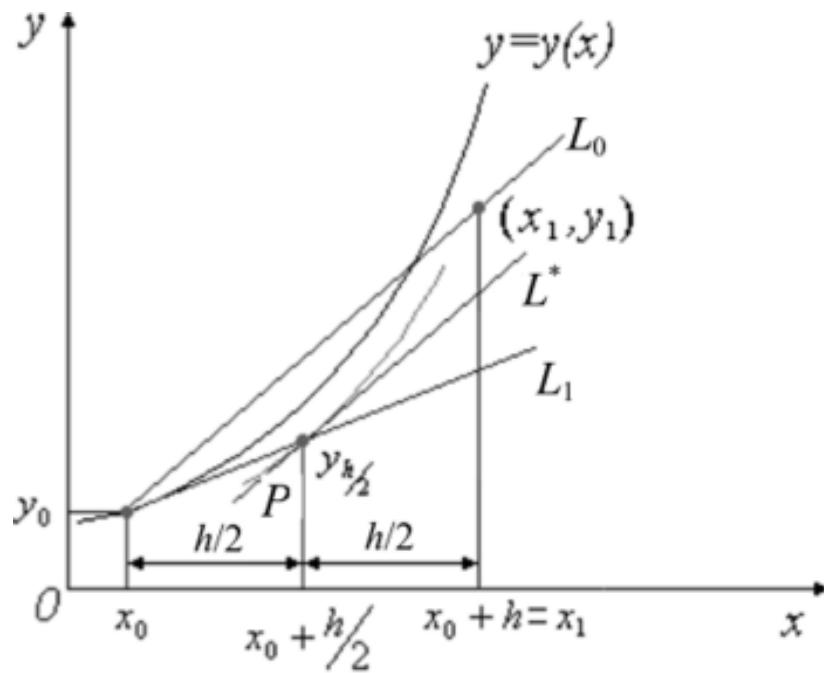
$$K_1 = f(x_{i-1}, y_{i-1})$$

$$K_2 = f(x_{i-1} + h, y_{i-1} + h \cdot K_1)$$



## Метод имеет второй порядок точности

Локальная погрешность метода Рунге–Кутты 2-го порядка  $e_2 = C \cdot h^3$ , где  $C$  – некоторая постоянная, и пропорциональна кубу шага интегрирования: при уменьшении шага в 2 раза локальная погрешность уменьшится в 8 раз.





```
In [1]: def dydx(x, y):
    return (x + y - 2)

def rungeKutta(x0, y0, x, h):
    n = round((x - x0) / h)
    y = y0

    for i in range(1, n + 1):
        k1 = h * dydx(x0, y)
        k2 = h * dydx(x0 + 0.5 * h, y + 0.5 * k1)

        y = y + (1.0 / 6.0) * (k1 + 2 * k2)
        x0 = x0 + h
    return y

x0 = 0
y = 1
x = 2
h = 0.2
print("y(x) =", rungeKutta(x0, y, x, h))
```

y(x) = 0.6455901672629244



$$y_{k+1} = y_k + h \left[ (1 - \alpha)f(x_k, y_k) + \alpha f\left(x_k + \frac{h}{2\alpha}, y_k + f(x_k, y_k) \frac{2h}{\alpha}\right)\right].$$

При  $\alpha = 1$  получаем:

$$y_{k+1} = y_k + h \cdot f\left(x_k + \frac{h}{2}, y_k + \frac{h}{2} \cdot f(x_k, y_k)\right)$$

проведение расчетов на очередном шаге интегрирования можно рассматривать как последовательность нижеследующих операций.

1. Вычисляется выражение, представляющее собой полу шаг интегрирования по схеме Эйлера, то есть определяется приближенное значение искомой функции в точке  $x_k + h/2$ :

$$y_{k+1/2} = y_k + \frac{h}{2} \cdot f(x_k, y_k)$$



1. Вычисляется выражение, представляющее собой полу шаг интегрирования по схеме Эйлера, то есть определяется приближенное значение искомой функции в точке  $x_k + h/2$ :

$$y_{k+1/2} = y_k + \frac{h}{2} \cdot f(x_k, y_k)$$

2. Для той же промежуточной точки находится приближенное значение производной:

$$y'_{k+1/2} = f\left(x_k + \frac{h}{2}, y_{k+1/2}\right)$$

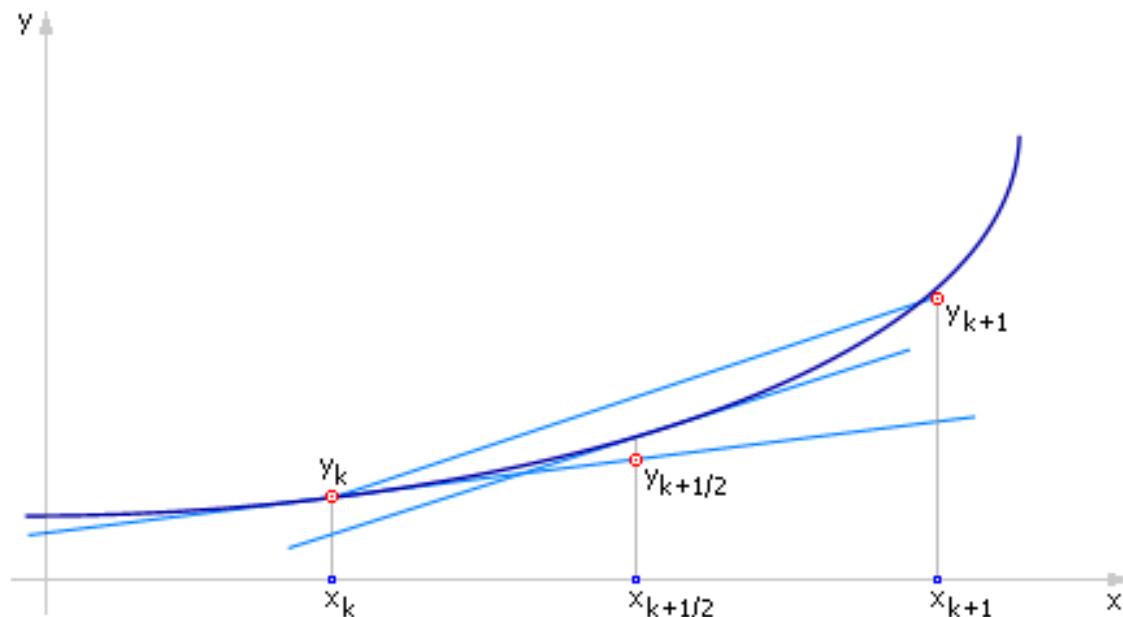
3. Определяется уточненное значение функции в конечной точке всего шага, причем по схеме Эйлера с вычисленным на предыдущем шаге значением производной

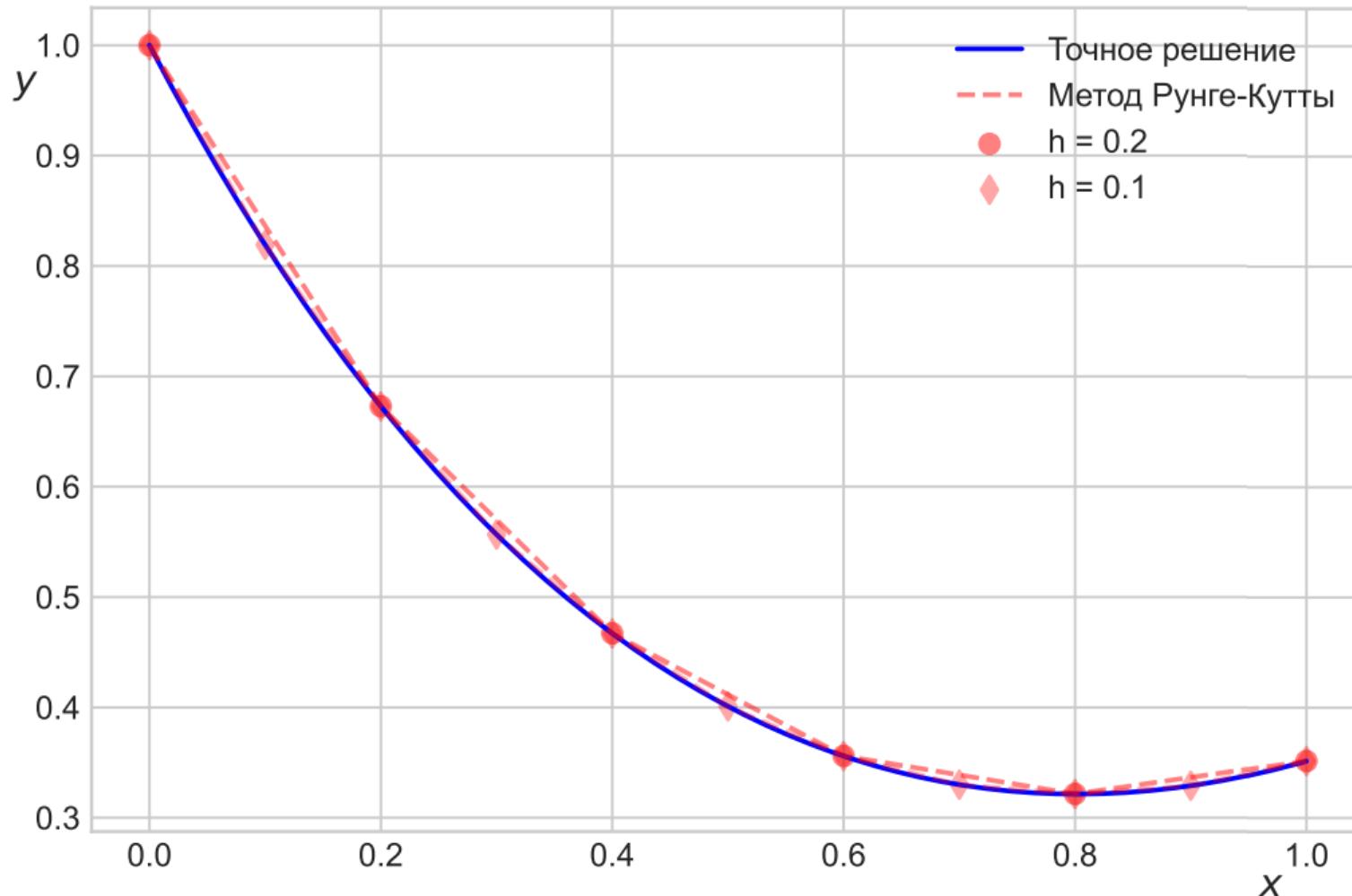
$$y_{k+1} = y_k + h \cdot y'_{k+1/2}$$



## Метод имеет второй порядок точности

Локальная погрешность метода Рунге–Кутты 2-го порядка  $e_2 = C \cdot h^3$ , где  $C$  – некоторая постоянная, и пропорциональна кубу шага интегрирования: при уменьшении шага в 2 раза локальная погрешность уменьшится в 8 раз.





В сравнении с методом Эйлера, метод Рунге-Кутты дает результаты более близкие к точному решению даже при достаточно высоких значениях шага интегрирования.



Решить задачу Коши:

$$\begin{cases} y' = x^2 - \sin 2x \\ y(0) = 2. \end{cases}$$

Пусть шаг сетки  $h = 0,2$ . Тогда  $x_0 = 0$ ,  $x_1 = 0,2$ ,  $x_2 = 0,4$  и т.д.

В нашем случае:

$$f(x, y) = x^2 - \sin 2x$$

Тогда получаем следующую схему:

$$y_{i+1} = y_i + h(x_i^2 - \sin 2x_i)$$

Используя эту схему, последовательно получаем:

$$y_1 = y_0 + h(x_0^2 - \sin 2x_0) = 2 + 0,2(0 - 0) = 2,$$

$$y_2 = y_1 + h(x_1^2 - \sin 2x_1) = 2 + 0,2(0,2 - \sin(2 \cdot 0,2)) = 1,930,$$

$$y_3 = y_2 + h(x_2^2 - \sin 2x_2) = 1,930 + 0,2(0,4 - \sin(2 \cdot 0,4)) = 1,819$$

$x_i$	2	1,8	1,6	1,4	1,2	1	0,8	0,6	0,4	0,2	0
$y_i$ методом Эйлера	2	2,000	1,930	1,819	1,704	1,632	1,650	1,803	2,128	2,652	3,389



Если шаг сетки уменьшить в 2 раза ( $h = 0,1$ ), то получим следующие значения:

$$y_1 = y_0 + h(x_0^2 - \sin 2x_0) = 2 + 0,1(0 - 0) = 2,$$

$$y_2 = y_1 + h(x_1^2 - \sin 2x_1) = 2 + 0,1(0,1 - \sin(2 \cdot 0,1)) = 1,981,$$

$$y_3 = y_2 + h(x_1^2 - \sin 2x_1) = 1,981 + 0,1(0,2 - \sin(2 \cdot 0,2)) = 1,946,$$

$$y_4 = y_3 + h(x_1^2 - \sin 2x_1) = 1,946 + 0,1(0,3 - \sin(2 \cdot 0,3)) = 1,899,$$

$x_i$	0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1
$y_i$	2	2,000	1,981	1,946	1,899	1,843	1,784	1,727	1,677	1,641	1,625

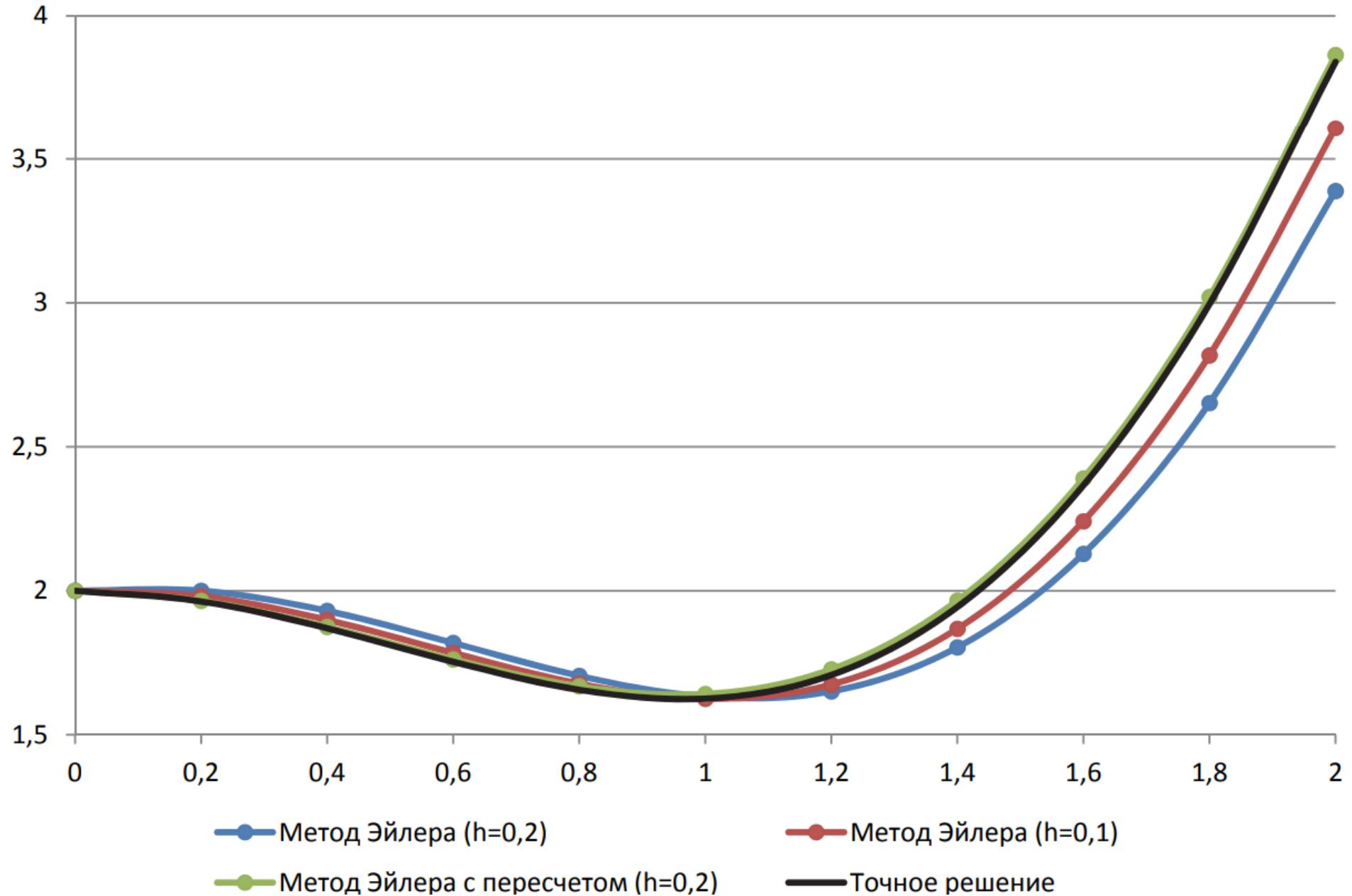
$x_i$	1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1,9	2
$y_i$	1,634	1,674	1,750	1,868	2,030	2,241	2,503	2,818	3,186	3,608



Сводная таблица решения задачи Коши

$$\begin{cases} y' = x^2 - \sin 2x \\ y(0) = 2. \end{cases}$$

$i$	0	1	2	3	4	5	6	7	8	9	10
$x_i$	0	0,2	0,4	0,6	0,8	1	1,2	1,4	1,6	1,8	2
$y_i$ (методом Эйлера, $h = 0,2$ )	2	2,000	1,930	1,819	1,704	1,632	1,650	1,803	2,128	2,652	3,389
$y_i$ (методом Эйлера, $h = 0,1$ )	2	1,981	1,899	1,784	1,677	1,625	1,674	1,868	2,241	2,818	3,608
$y_i$ (методом Эйлера с пе- ресчетом, $h = 0,2$ )	2	1,965	1,874	1,761	1,668	1,641	1,727	1,966	2,390	3,020	3,864
$y_i$ (точное)	2	1,963	1,870	1,753	1,656	1,625	1,707	1,944	2,366	2,996	3,840





Рассмотрим разностную схему Рунге-Кутта четвертого порядка точности, которую на практике использует чаще других:

$$y_i = y_{i-1} + \frac{h}{6} \cdot (k_1 + 2 \cdot k_2 + 2 \cdot k_3 + k_4)$$
$$x_i = x_{i-1} + h$$

Где

$$k_1 = f(x_{i-1}, y_{i-1})$$

$$k_2 = f\left(x_{i-1} + \frac{h}{2}, y_{i-1} + \frac{h}{2} \cdot k_1\right)$$

$$k_3 = f\left(x_{i-1} + \frac{h}{2}, y_{i-1} + \frac{h}{2} \cdot k_2\right)$$

$$k_4 = f(x_{i-1} + h, y_{i-1} + h \cdot k_3)$$

Таким образом, метод Рунге-Кутта требует на каждом шаге четырехкратного вычисления правой части уравнения  $f(x, y)$ .



$$y_i = y_{i-1} + \frac{h}{6} \cdot (k_1 + 2 \cdot k_2 + 2 \cdot k_3 + k_4)$$

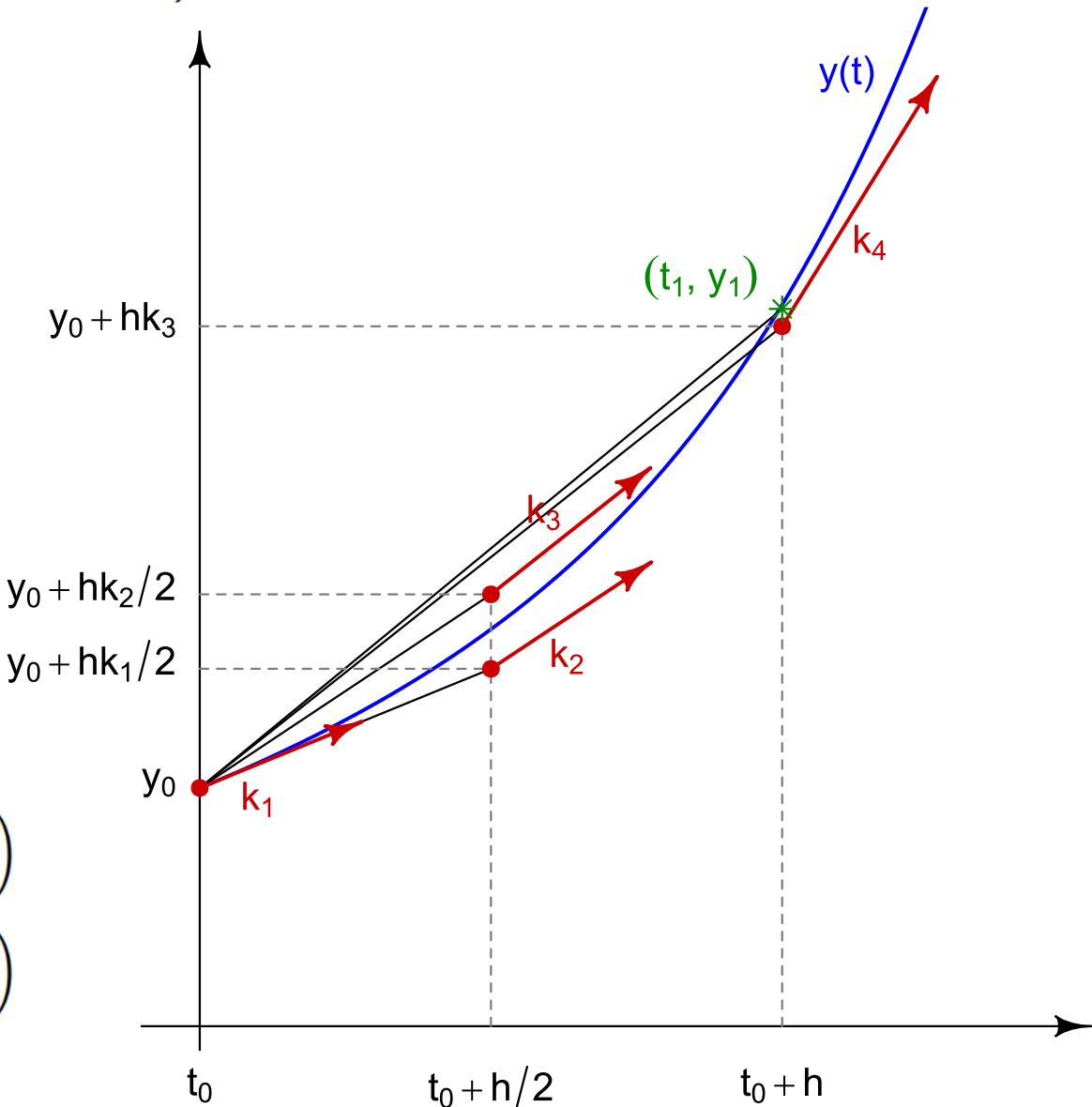
$$x_i = x_{i-1} + h$$

$$k_1 = f(x_{i-1}, y_{i-1})$$

$$k_2 = f\left(x_{i-1} + \frac{h}{2}, y_{i-1} + \frac{h}{2} \cdot k_1\right)$$

$$k_3 = f\left(x_{i-1} + \frac{h}{2}, y_{i-1} + \frac{h}{2} \cdot k_2\right)$$

$$k_4 = f(x_{i-1} + h, y_{i-1} + h \cdot k_3)$$





Рассмотрим решение обыкновенного дифференциального уравнения:

$$\frac{dy}{dx} = \frac{y}{(\cos(x))^2}$$

Воспользуемся формулами

$$y_i = y_{i-1} + \frac{h}{6} \cdot (k_1 + 2 \cdot k_2 + 2 \cdot k_3 + k_4)$$

$$x_i = x_{i-1} + h$$

и построим с их помощью таблицу искомых значений переменной  $y_i^*$ , соответствующих значениям переменной  $x$  из диапазона  $[0, 1]$  с шагом  $h = 0,1$ .

Подставим в формулы

$$k_1 = f(x_{i-1}, y_{i-1})$$

$$k_3 = f\left(x_{i-1} + \frac{h}{2}, y_{i-1} + \frac{h}{2} \cdot k_2\right)$$

$$k_2 = f\left(x_{i-1} + \frac{h}{2}, y_{i-1} + \frac{h}{2} \cdot k_1\right)$$

$$k_4 = f(x_{i-1} + h, y_{i-1} + h \cdot k_3)$$

выражение правой части исходного дифференциального уравнения и запишем формулы для определения параметров метода Рунге-Кутты:



Результаты сведем в таблице:

$x_i$	$k_1$	$k_2$	$k_3$	$k_4$	$y_i$
0.0	—	—	—	—	2.7183
0.1	2.7183	2.8614	2.8685	3.0354	3.0052
0.2	3.0354	3.2291	3.2390	3.4659	3.3291
0.3	3.4659	3.7308	3.7449	4.0580	3.7037
0.4	4.0581	4.4272	4.4481	4.8901	4.1487
0.5	4.8903	5.4184	5.4509	6.0947	4.6941
0.6	6.0951	6.8779	6.9318	7.9088	5.3878
0.7	7.9096	9.1256	9.2215	10.7866	6.3110
0.8	10.7883	12.7957	12.9832	15.6764	7.6114
0.9	15.6806	19.2742	19.6867	24.7932	9.5846
1.0	24.8050	31.9927	33.0548	44.1554	12.9022



Метод Рунге-Кутта требует большего объема вычислений, однако это окупается повышенной точностью, что дает возможность проводить счет с большим шагом. Другими словами, для получения результатов с одинаковой точностью в методе Эйлера потребуется значительно меньший шаг, чем в методе Рунге-Кутта.

С уменьшением шага  $h$  локальная погрешность метода Эйлера снижается, но при этом возрастет количество узлов, что неблагоприятно влияет на точность результатов. Поэтому метод Эйлера применяется достаточно редко при небольшом числе расчетных точек. Наиболее употребительным одношаговым методом является метод Рунге-Кутта.

