ЧИСЛЕННЫЕ МЕТОДЫ

к.ф.-м.н., доц. Чернышенко Всеволод Сергеевич
VSChernyshenko@fa.ru
- 2025 -

1. Теория погрешностей.
2. Методы решения нелинейных уравнений.
3. Методы интерполирования 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. Пантелеев. 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	$\to \infty$
II	max 5	max 15		$\to \infty$

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.

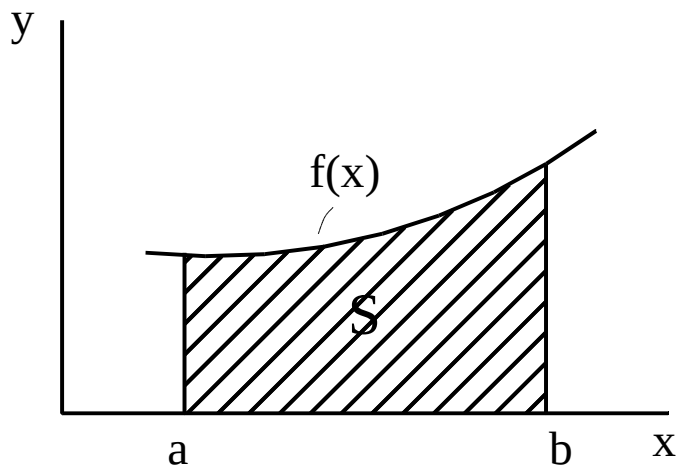
ПРЕИМУЩЕСТВА

- 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



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

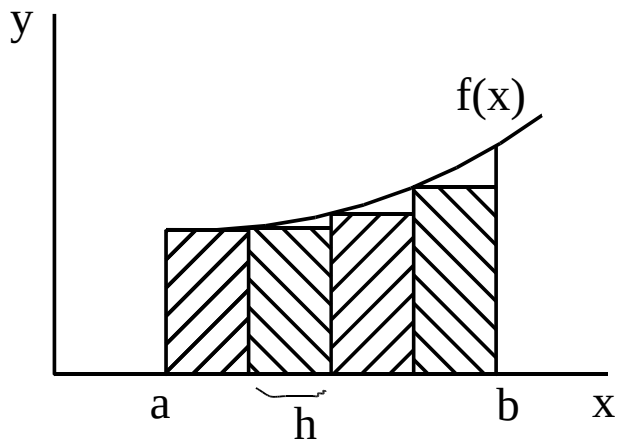
Формула прямоугольников

- 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

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

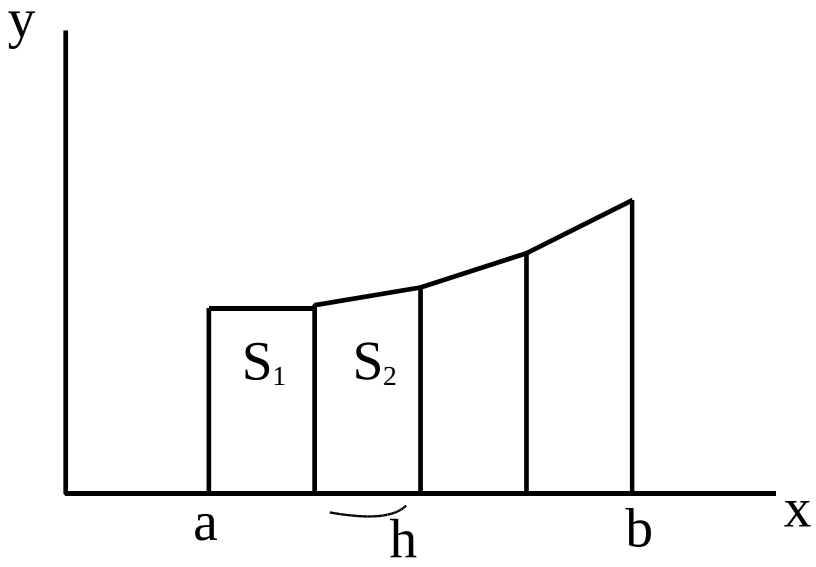


$$S = h * \sum_{x=a}^{b-h} f(x)$$

Формула трапеций

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

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



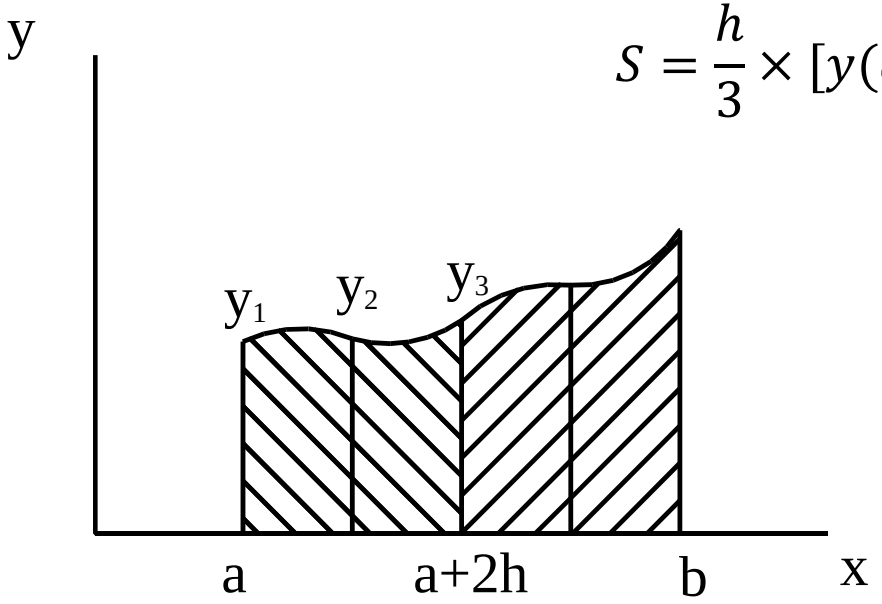
$$S = \left[\frac{f(a) + f(b)}{2} + \sum_{x=a+h}^{b-h} f(x_i)\right] \times h$$

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:

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

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



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 ε.

ТЕОРИЯ ПРИБЛИЖЁННЫХ ВЫЧИСЛЕНИЙ

Погрешность решения задачи обуславливается следующими причинами:

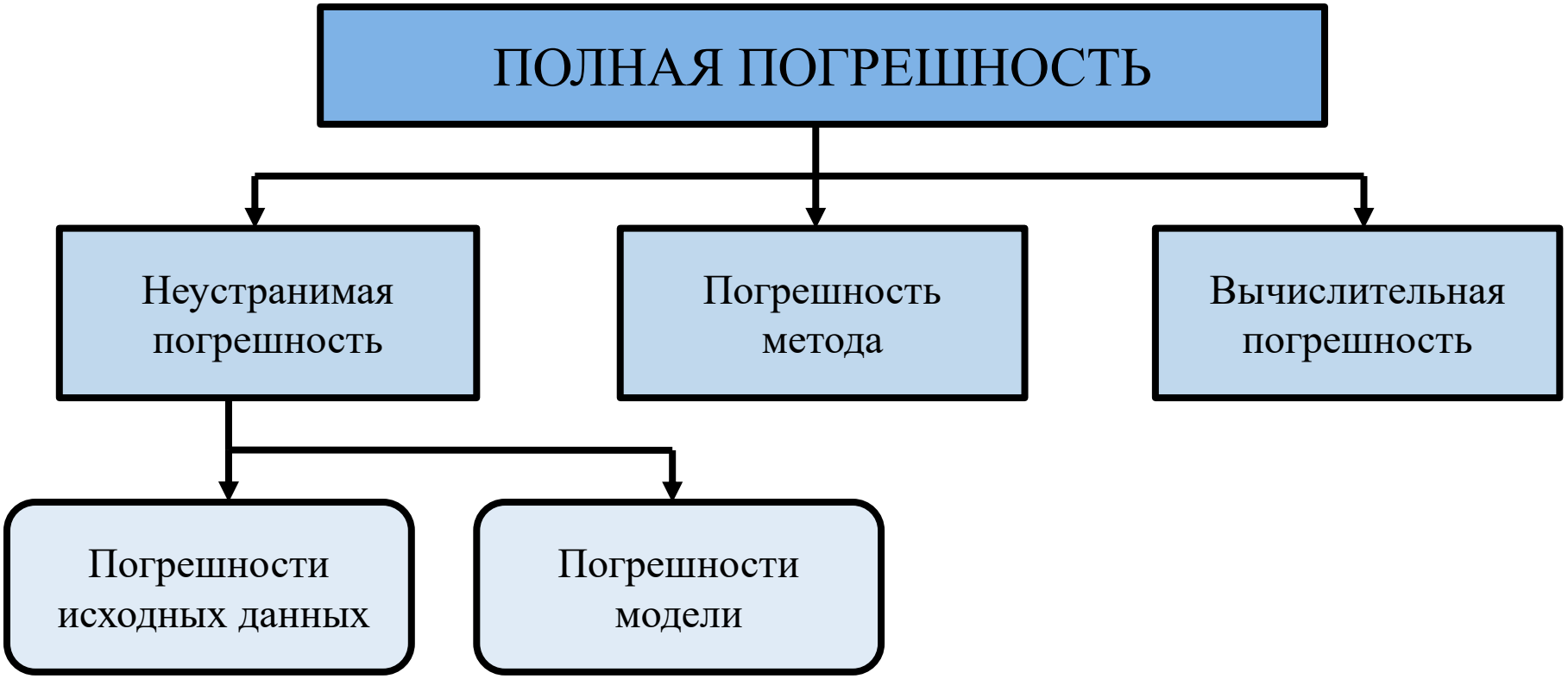
1. Математическое описание задачи является неточным, в частности, неточно заданы исходные данные описания.
2. Применяемый для решения метод часто не является точным: получение точного решения требует неограниченного или неприемлемо большого числа арифметических операций, и поэтому вместо получения точного 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 Δ 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 Δ_a

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

a	a_P	Δ	Δ_a
2,718281828459045	2,72	0,001718172	2,720 - 2,718 = 0,002 2,72 - 2,718 < 0,01

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

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

Так как точное значение X обычно бывает неизвестно, то и Δ - 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 ≈ 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 δ не превосходит величины 10^{1-n}, деленной на первую значащую цифру α_H .

$$\delta \le 10^{1-n} / \alpha_H$$

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

Решение.

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

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

Для определения абсолютной погрешности:

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

b) Аналогично вычислим: $n = 7$, $\alpha_H = 2$,

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

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

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 ± 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 ± 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 ± 0.007, возможно перейти к использованию значения 46.8, отметив, что все цифры этого числа верны в узком смысле.

Пример. Пусть a = 2,91385, Δa = 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 Δa = 0.005. Запись b = 4,1 при условии, что все цифры верны в широком смысле, подразумевает, что Δb = 0,1. но по записи c = 4,100 мы можем заключить, что Δc = 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 $\Delta_{\text{окр}}$ = 0,005, а полная погрешность $\Delta a_{\text{окр}} = \Delta a + \Delta_{\text{окр}} = 0{,}001 + 0{,}005 = 0{,}006$. 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

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

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

Решение:

$$\Delta_u = \left|\frac{\partial u}{\partial x}\right| \Delta_x + \left|\frac{\partial u}{\partial y}\right| \Delta_y = |6cos(3x + 4y)| \cdot 0{,}002 +$$

$$+|8cos(3x + 4y)| \cdot 0{,}005 =$$

$$= |2cos(3x + 4y)| \cdot (3 \cdot 0{,}002 + 4 \cdot 0{,}005) =$$

$$= \left|2cos\frac{\pi}{4}\right| \cdot 0{,}026 = \sqrt{2} \cdot 0{,}026 \approx 0{,}037$$

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 δ не превосходит величины 10^{1-n}, деленной на первую значащую цифру α_H .

$$\delta \le 10^{1-n} / \alpha_H$$

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

Решение.

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

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

Для определения абсолютной погрешности:

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

b) Аналогично вычислим: $n = 7$, $\alpha_H = 2$,

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

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

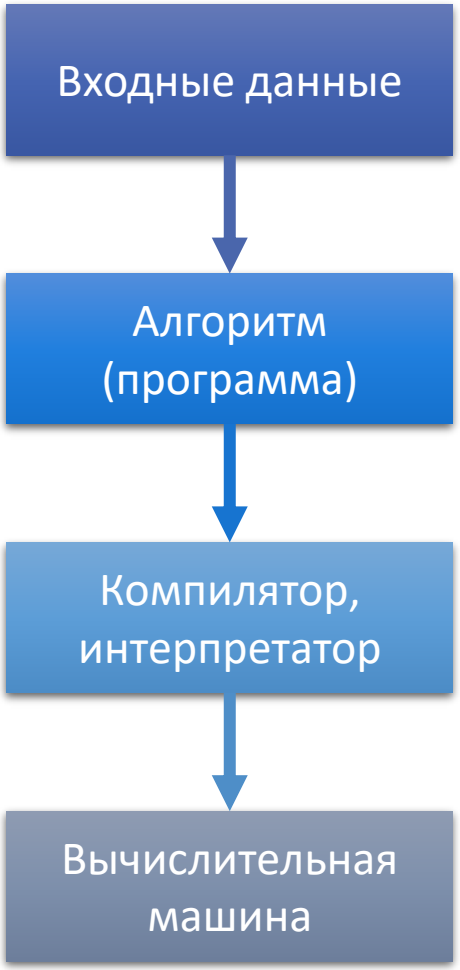
СЛОЖНОСТЬ И НОТАЦИЯ BIG-O

- **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 $n \to \infty$.

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 ($n \to \infty$) 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 – O-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: $\boldsymbol{O(n^2)}$.

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: $\boldsymbol{O(n)}$.

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.

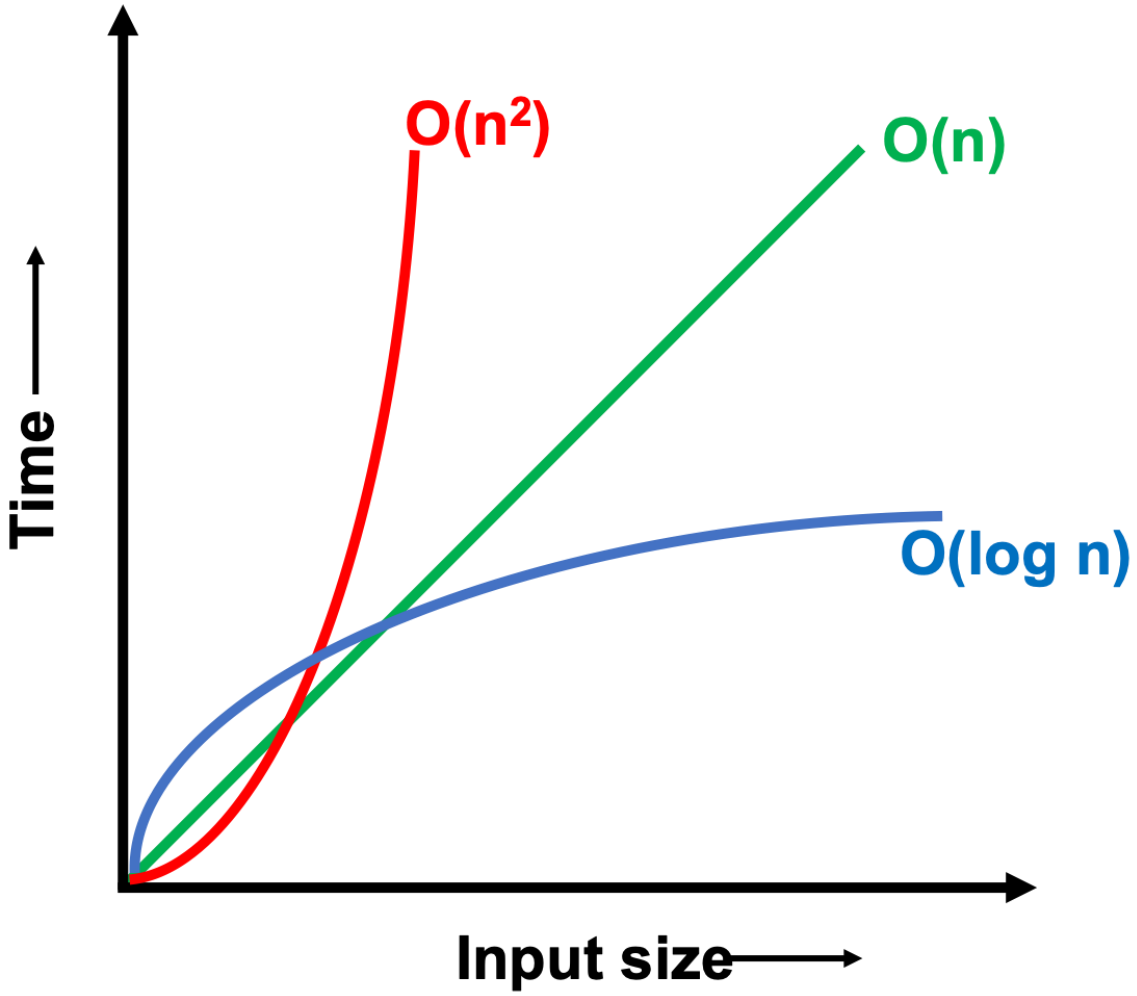
```
def divide_by_two(n):
    out = 0
    while n > 1:
        n /= 2
        out += 1
    return out
```

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: $\boldsymbol{O(logn)}$.
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-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 содержащит 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.

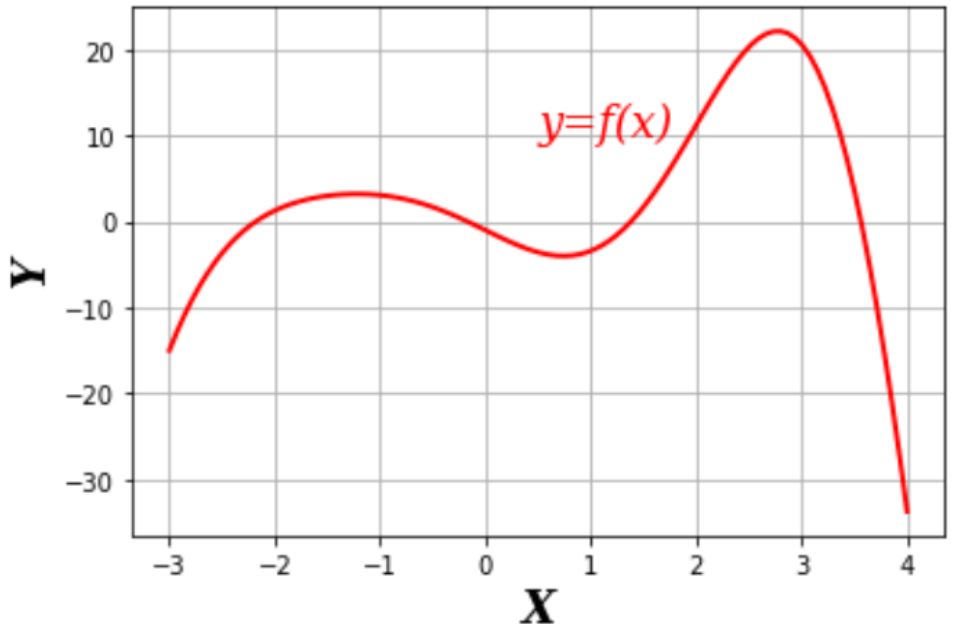
Далее будем предполагать, что $f(x)$ – непрерывная на интервале поиска корня функция.

- 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

$$x_1, x_2, \ldots, x_n .$$

Если

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

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 (α_i, β_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: $f(x) \equiv -6x + 2 = 0$.

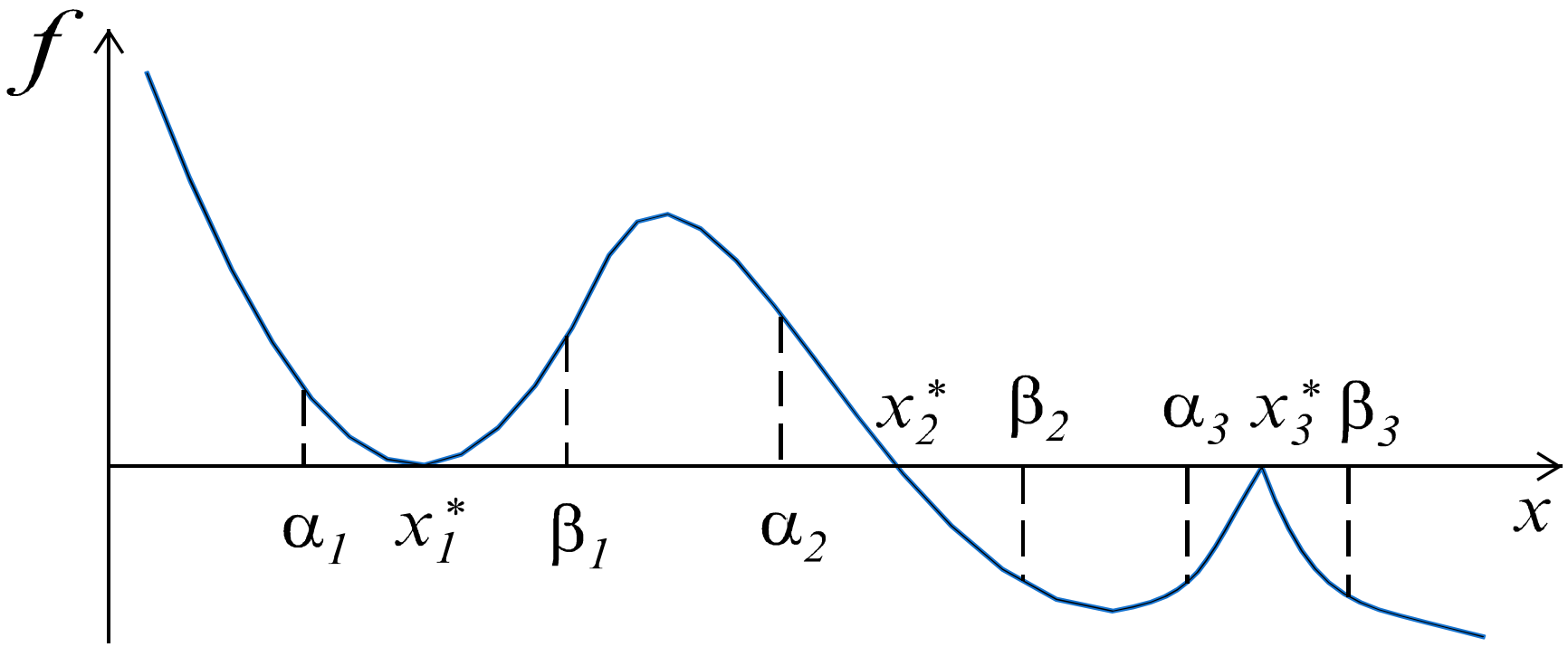
Составим приблизительную схему:

x	-∞	-3	-1	0	1	3	+∞
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:

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

Для простого корня:

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

Для вырожденного корня:

$f'(x_3^*)$ не существует,

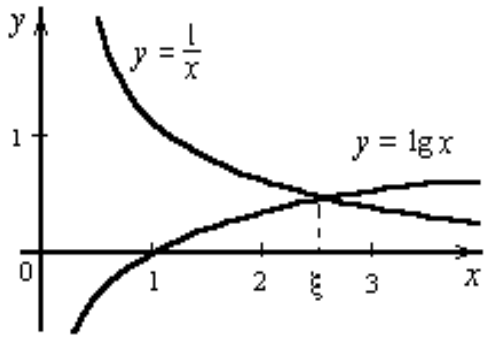
$$f(\alpha_3) \cdot f(\beta_3) > 0$$

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

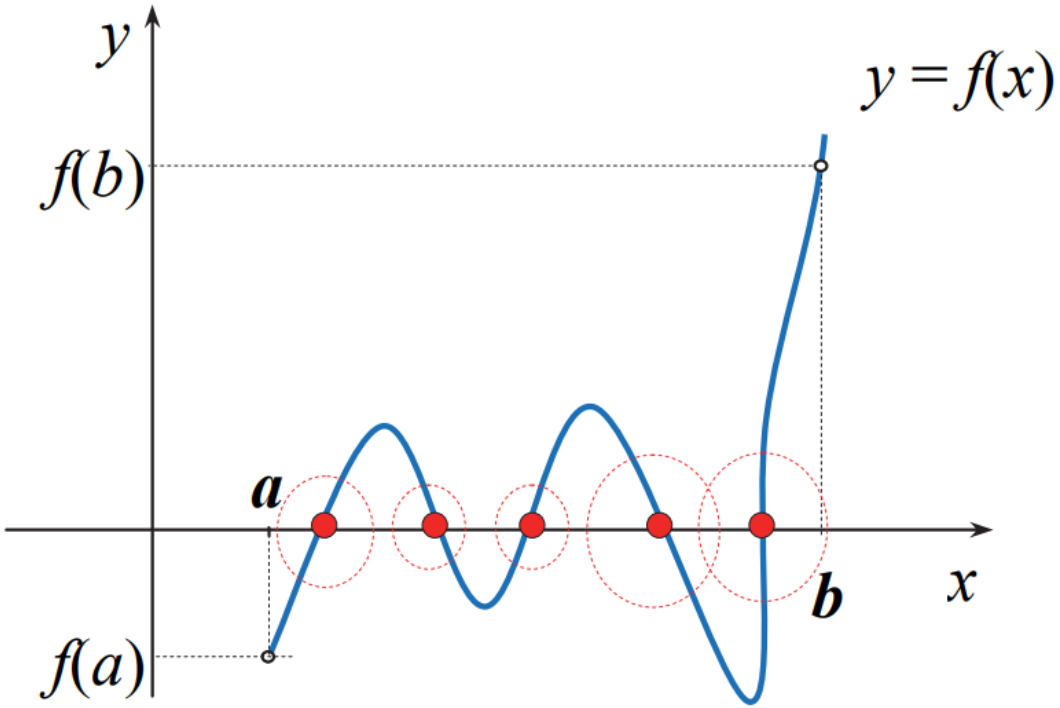
Вычисление значения простого корня с заданной точностью осуществляется одним из итерационных методов.

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.



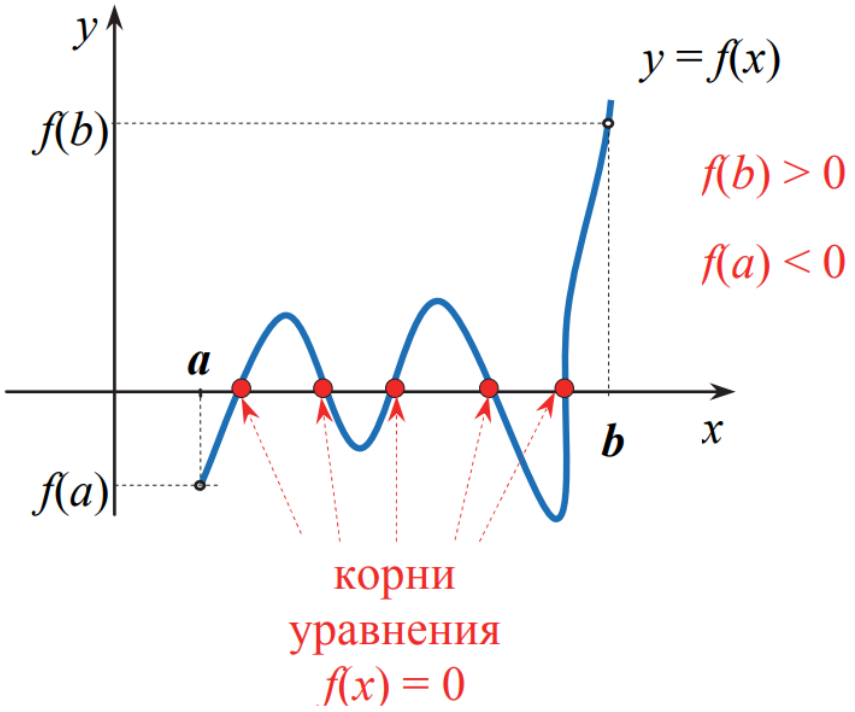
Будем считать, что уравнение имеет лишь изолированные корни, т.е. для каждого корня уравнения существует окрестность, не содержащая других корней этого уравнения.



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.

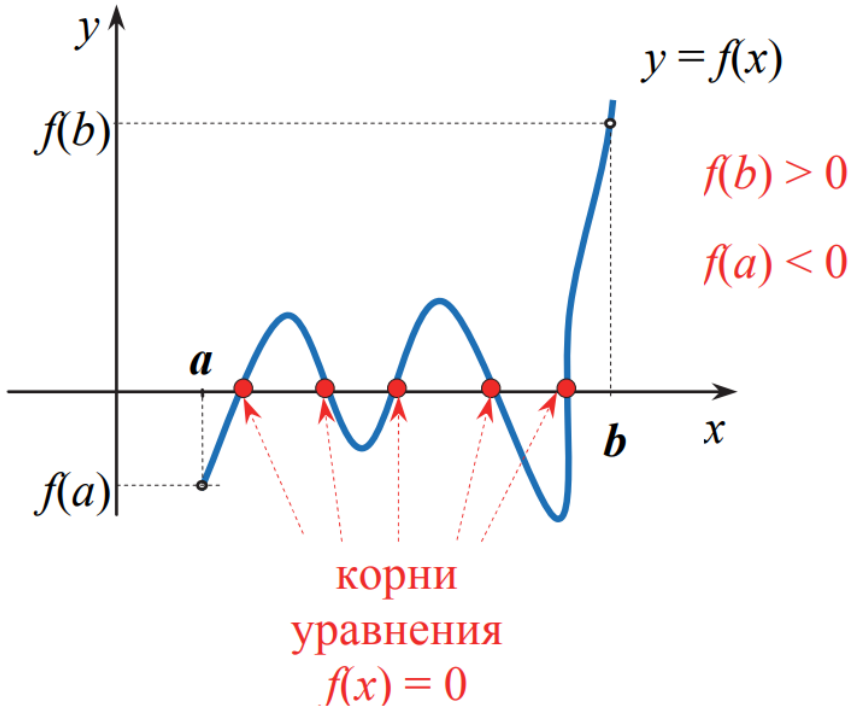
Теорема: Если непрерывная функция f(x) принимает значения разных знаков на концах отрезка [α, β], т.е. $f(\alpha)\cdot f(\beta) < 0$, то внутри этого отрезка содержится по меньшей мере один корень уравнения $f(x) = 0$, т.е. найдется хотя бы одно число $\xi \in (\alpha, \beta)$ такое, что $f(\xi) = 0$.



Корень ξ заведомо будет единственным, если производная f′(x) существует и сохраняет постоянный знак внутри интервала (α, β), т.е. если f′(x) > 0 (или f′(x) < 0) при α < x < β.

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

Определить отрезок [α, β], 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.

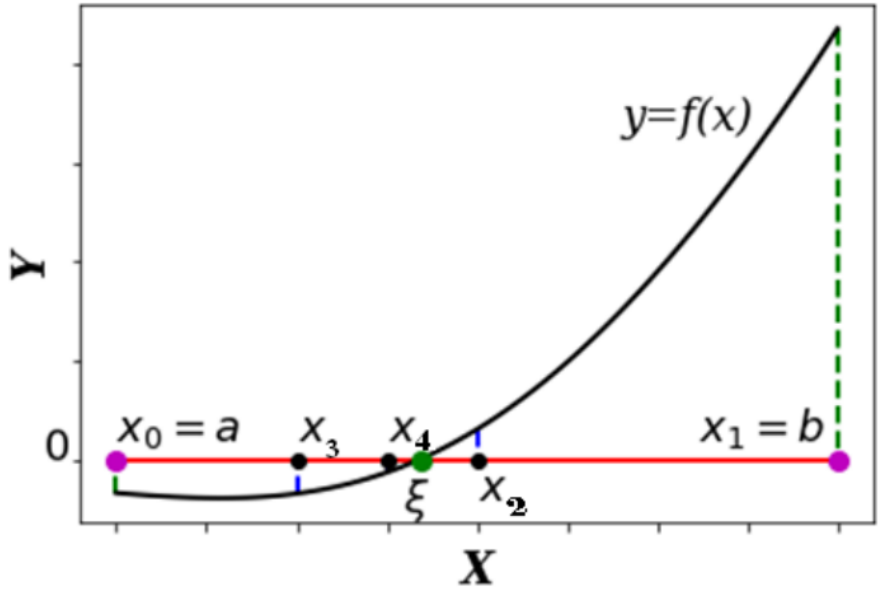
Второй этап решения уравнения – уточнение корня. Уточнить корень – значит найти его приближенное значение с заданной погрешностью ε.

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 ε.

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 ξ обозначен корень уравнения.



Пусть уравнение:

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

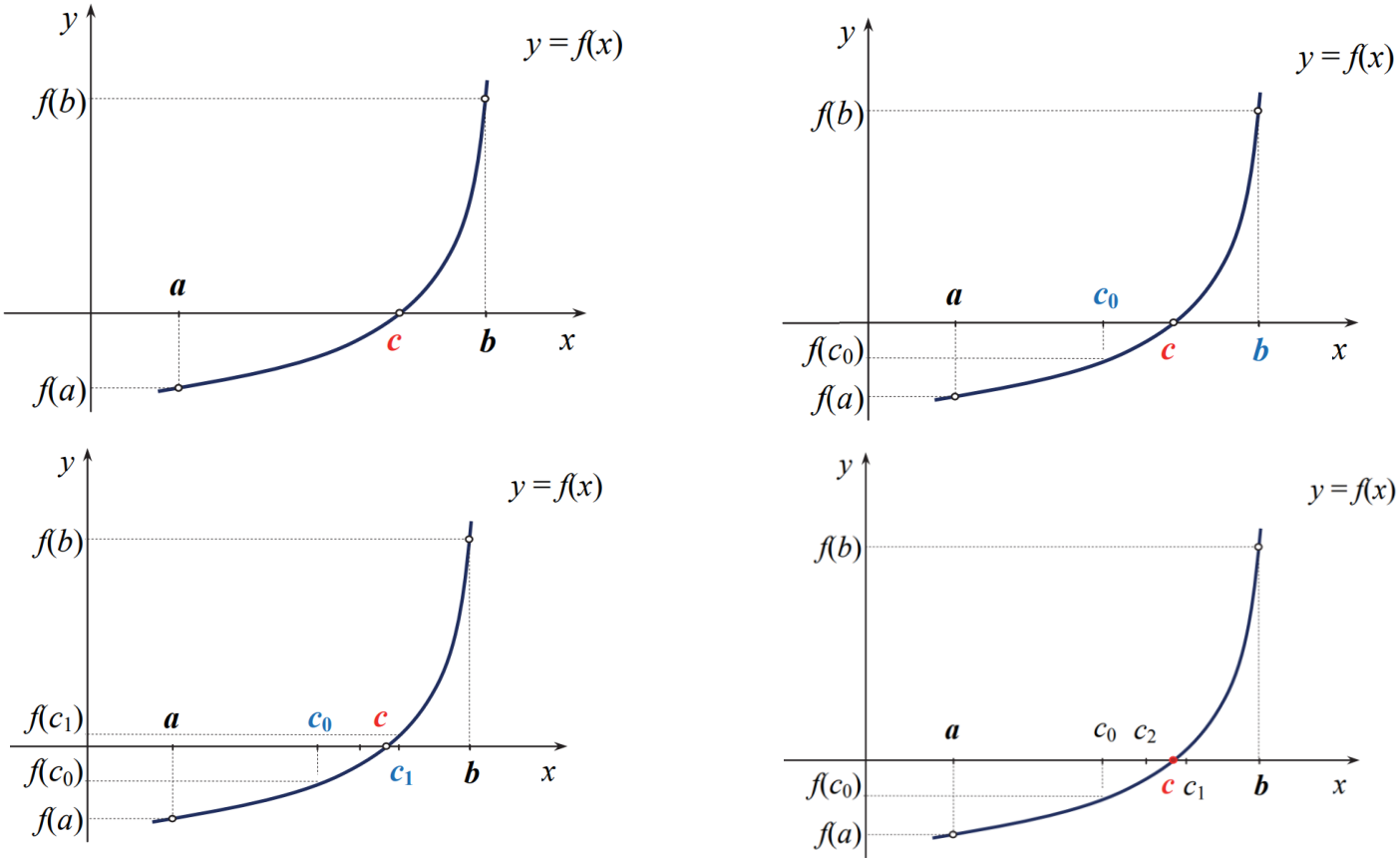
Разделим отрезок [a,b] пополам точкой $c = (a+b)/2$. Если $F(c) \neq 0$, то возможны два случая:

- 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 $f(x) = 0$.

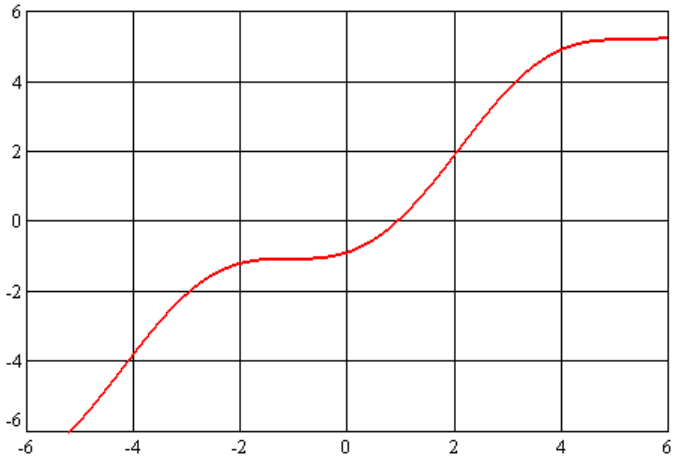
Если $F(x_i) = 0$, то корень найден.



Пример:

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

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



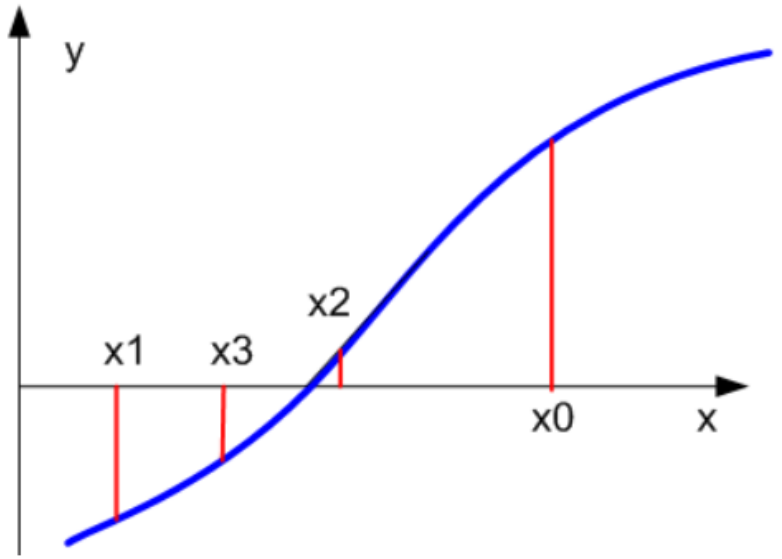
$a := 0$	$b := 2$	$f(a) = 0.866$	$f(b) = -1.906$
$x := \frac{a+b}{2} = 1$		$f(x) = -0.111$	$b := x$
$x := \frac{a+b}{2} = 0.5$		$f(x) = 0.5$	$a := x$
$x := \frac{a+b}{2} = 0.75$		$f(x) = 0.224$	$a := x$
$x := \frac{a+b}{2} = 0.875$		$f(x) = 0.064$	$a := x$
$x := \frac{a+b}{2} = 0.938$		$f(x) = -0.022$	$b := x$
$x := \frac{a+b}{2} = 0.906$		$f(x) = 0.021$	$a := x$
$x := \frac{a+b}{2} = 0.922$		$f(x) = -0.00014$	$x = 0.922$

Преимущества:

сходится для любых непрерывных функций.

Недостатки:

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



Уравнение $\mathbf{f(x) = 0}$ (1) записывают в разрешенном относительно $\mathbf{x}$ виде:

$$\mathbf{x = \varphi(x)}\,. \qquad (2)$$

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

$$\mathbf{\varphi(x) = x + \psi(x) \cdot f(x)}\,, \qquad (3)$$

где $\mathbf{\psi(x)}$ – произвольная, непрерывная, знакопостоянная функция (часто достаточно выбрать $\mathbf{\psi = const}$ 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 **φ(x)** накладываются следующие условия:

1) **φ(x)** должна быть определена и дифференцируема на отрезке [a, b], содержащем корень;
2) значения функции **φ(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

$$x_k = \varphi(x_{k-1}), \quad k = 1, 2, \ldots \qquad (4)$$

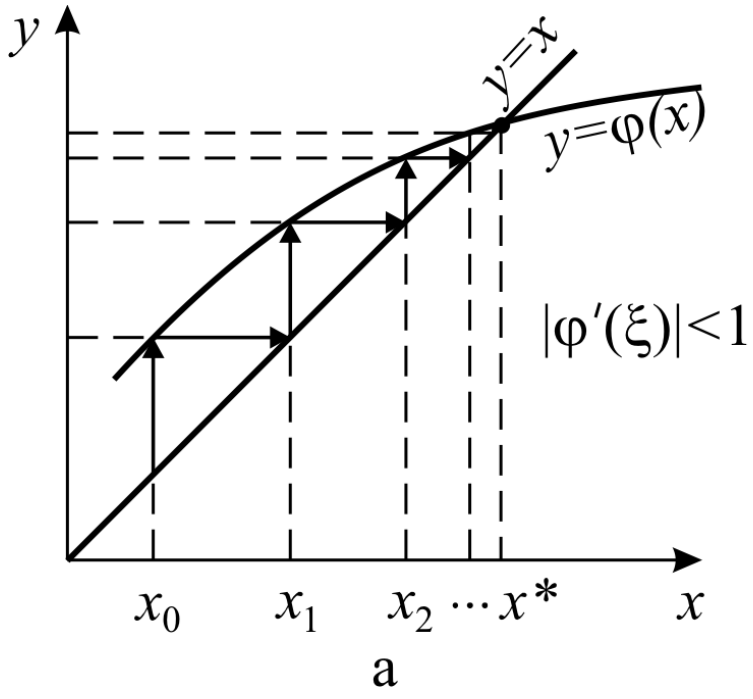
Вычисления x_k продолжаем до тех пор, пока

$$|x_k - x_{k-1}| > \varepsilon;$$

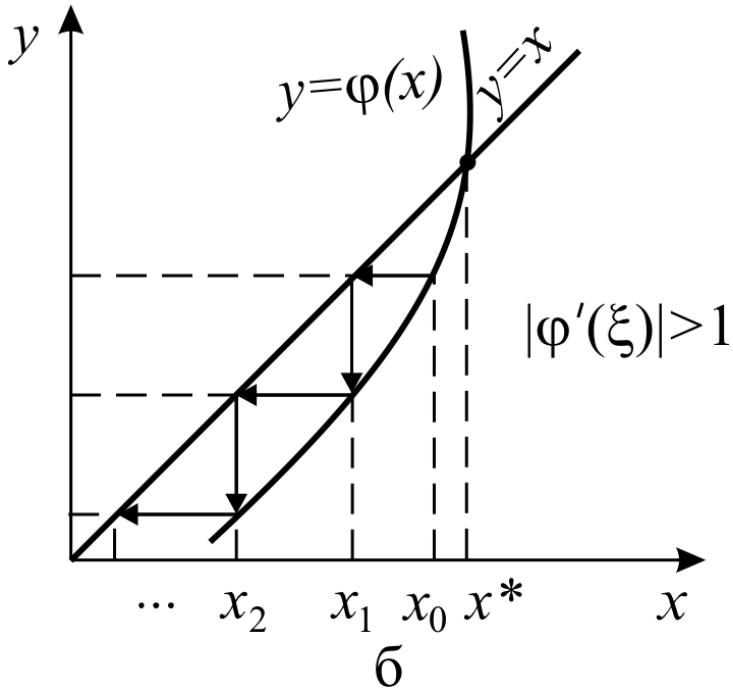
где ε – заданная точность решения.

Метод одношаговый, т.к. последовательность $x_0, x_1, \ldots, x_{k-1}$ имеет первый порядок ($m = 1$) и для начала вычислений достаточно знать одно начальное приближение: $x_0 = \alpha$, или $x_0 = \beta$, или $x_0 = (\alpha + \beta) / 2$.

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



а



б

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 (α, β), 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 $y = x$.

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

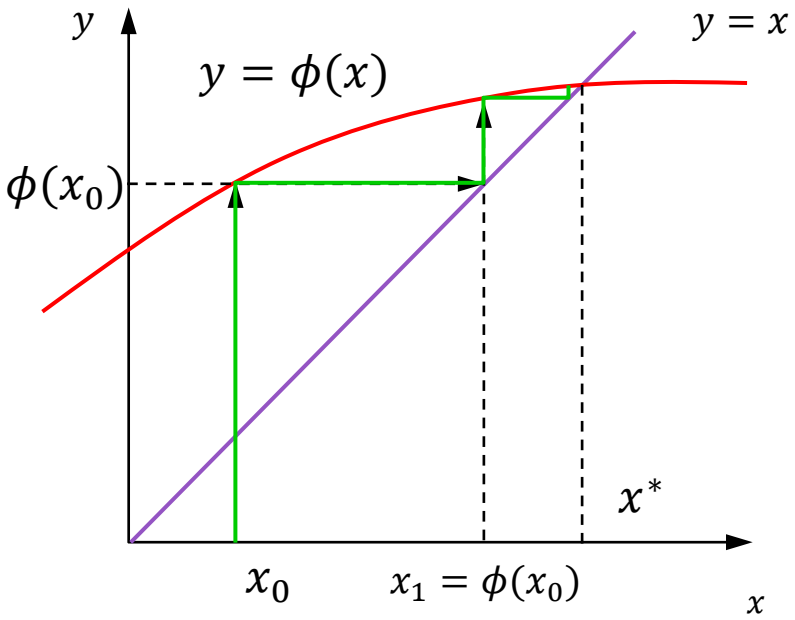
По свойству производных обратных функций на отрезке $[a, b]$ имеет место соотношение:

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

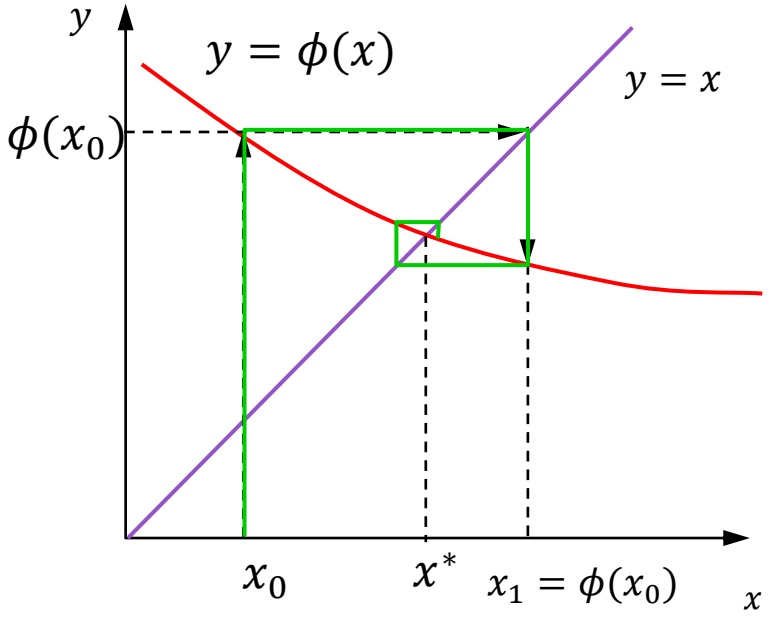
Причем уравнение $x = \psi(x)$ имеет тот же корень, что и $x = \varphi(x)$.

Сходящийся итерационный процесс:

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



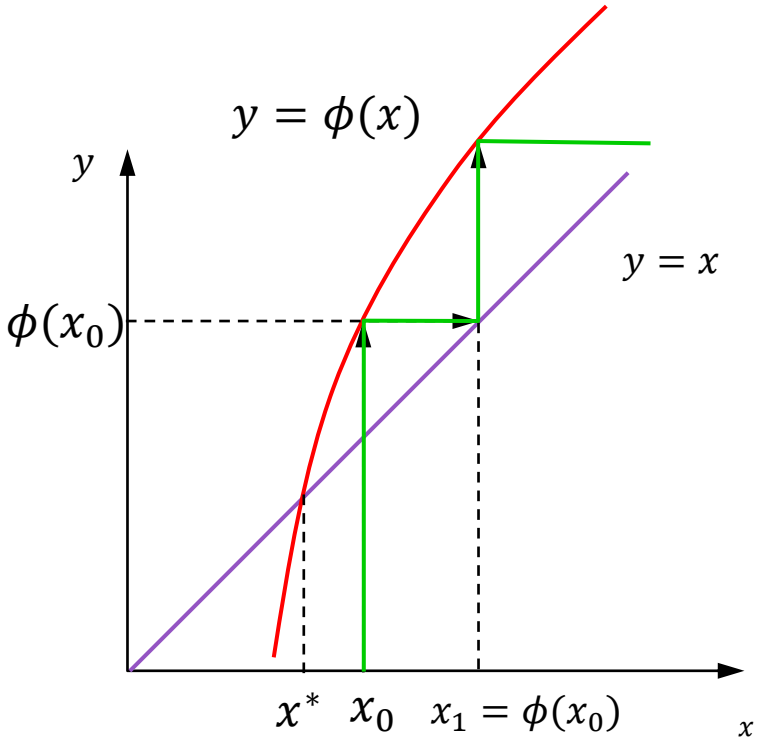
односторонняя сходимость



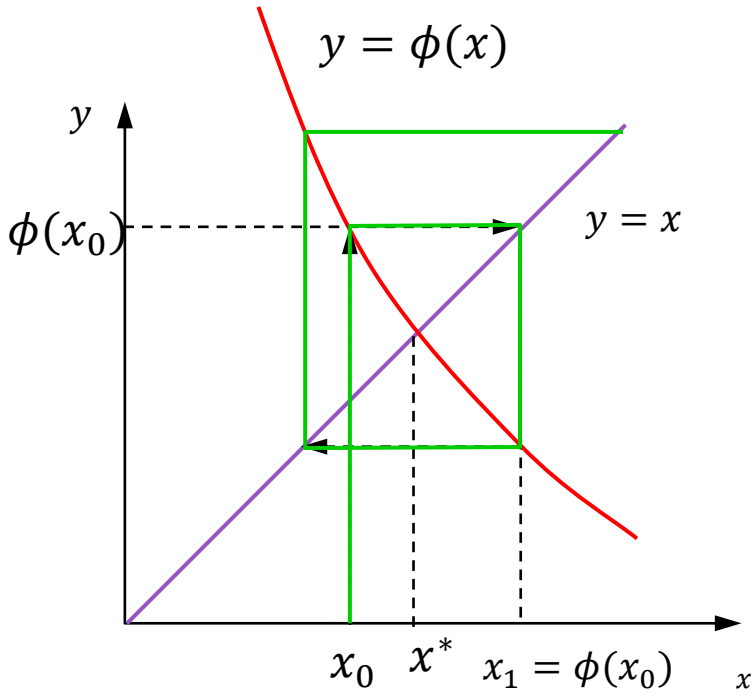
двусторонняя сходимость

Расходящийся итерационный процесс:

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



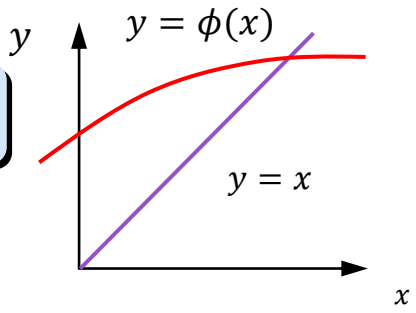
односторонняя расходимость



двусторонняя расходимость

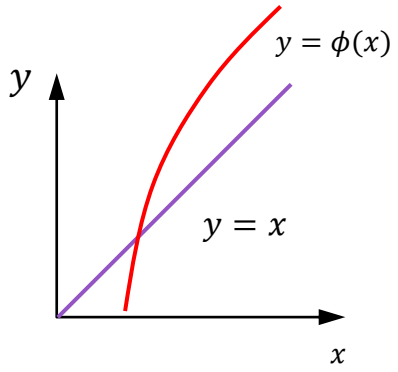
сходится

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

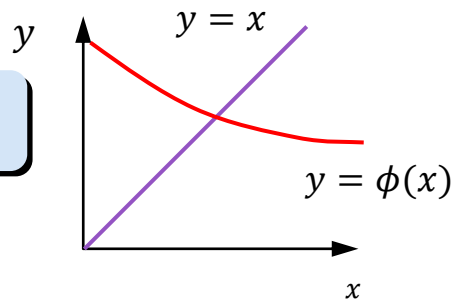


расходится

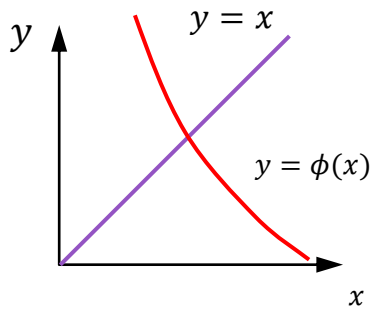
$\phi'(x) > 1$



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



$\phi'(x) < -1$



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

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

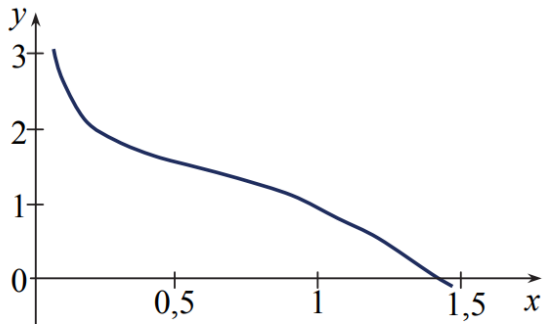
$$f'(x_0) > 0 \Rightarrow -\frac{2}{f'(x_0)} < b < 0$$

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

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 = φ(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,…

Номер итерации, k	Приближение корня, $x^{(k)}$	Проверка условия: $\lvert x_k - x_{k-1} \rvert < \varepsilon = 0{,}001$
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	**выполняется**

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:

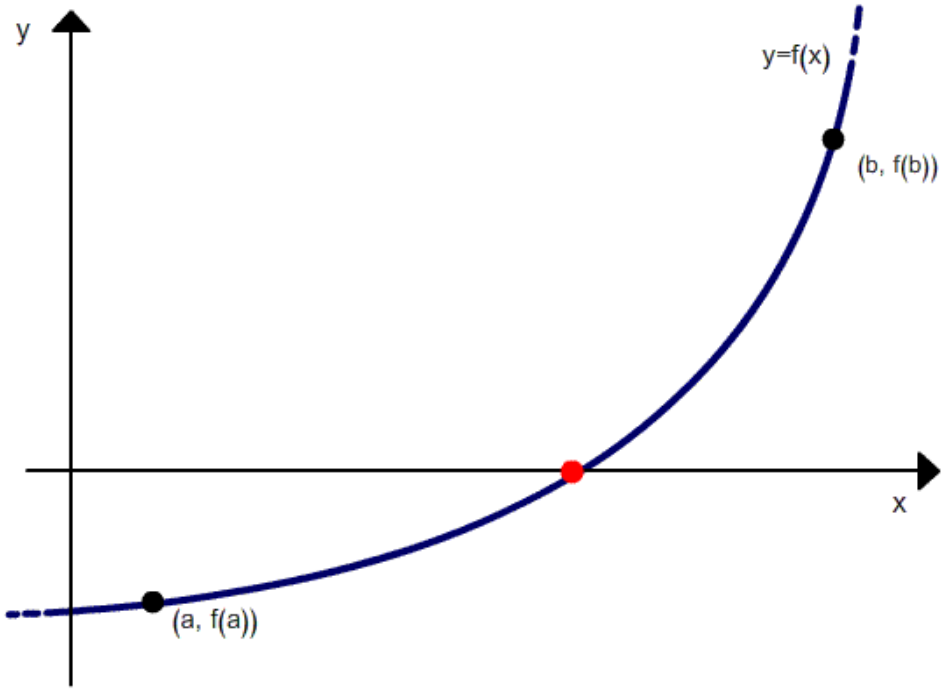
$$x_k = x_{k-1} - \frac{f(x_{k-1})h}{f(x_{k-1}) - f(x_{k-1} - h)} = \phi(x_{k-1}).$$

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

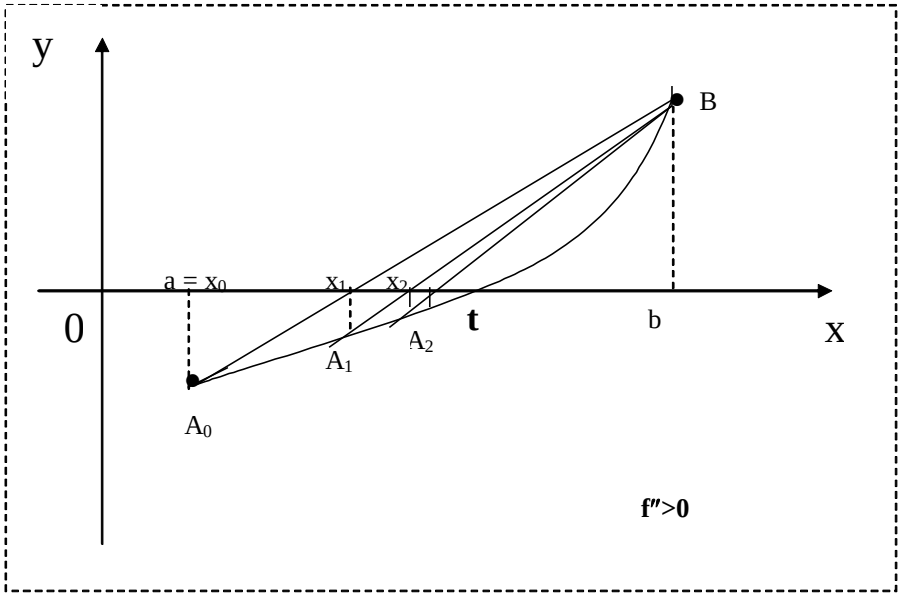
Формула для расчетов:

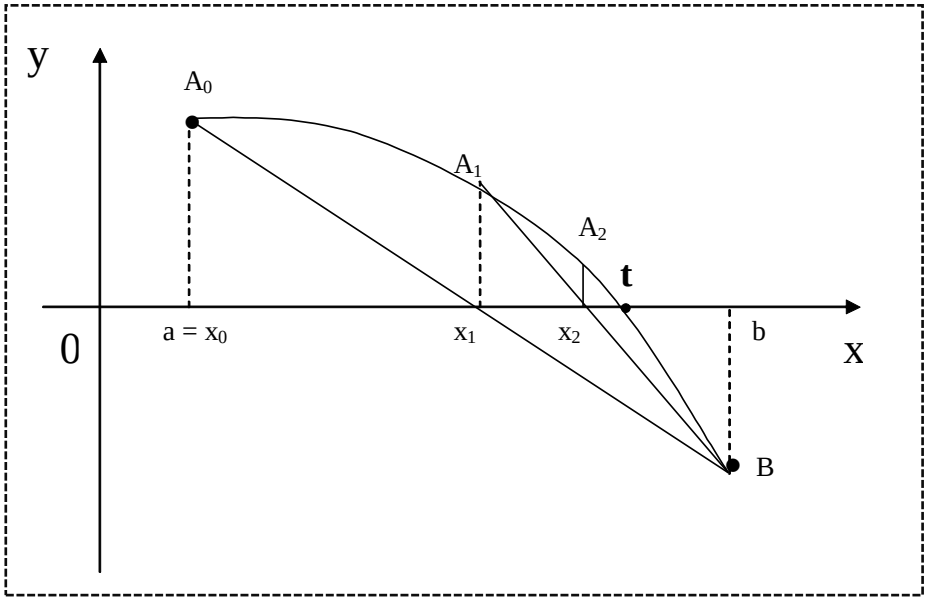
$$x_{n+1} = x_n - \frac{x_n - x_0}{f(x_n) - f(x_0)} f(x_n)$$

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

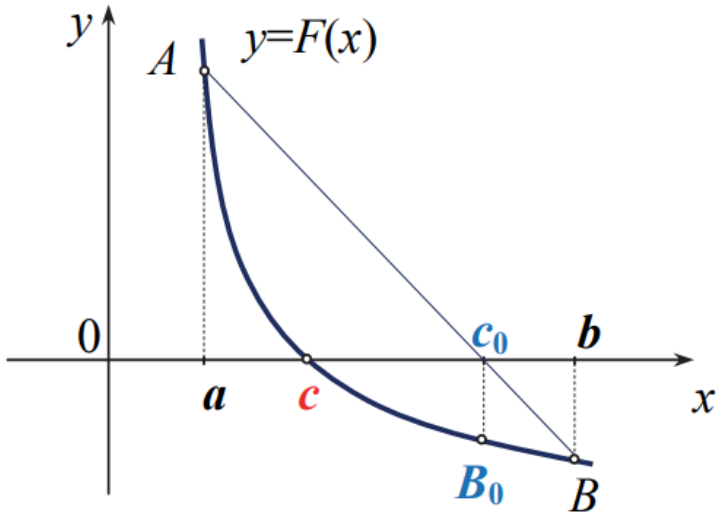
из уравнения прямой, проходящей через две точки:

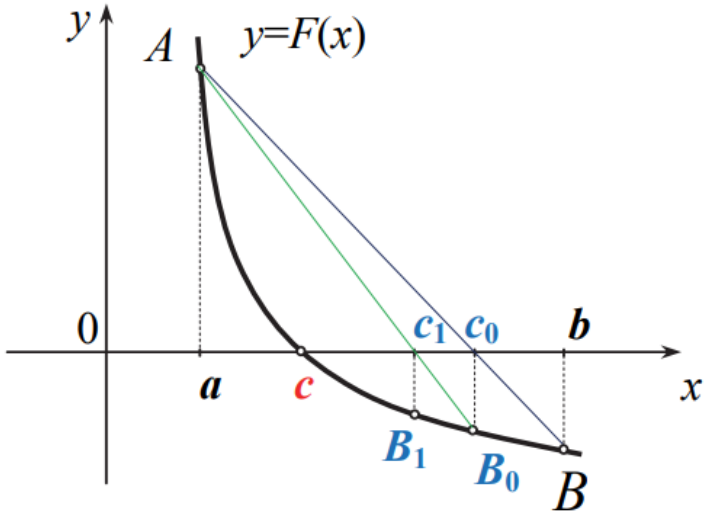
$$\frac{y - y_1}{y_2 - y_1} = \frac{x - x_1}{x_2 - x_1}$$

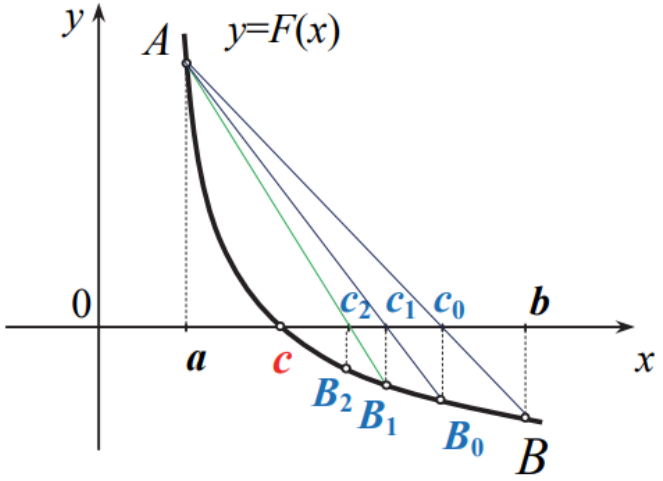
Найдем значение $x = x_1$ для которого $y = 0$.

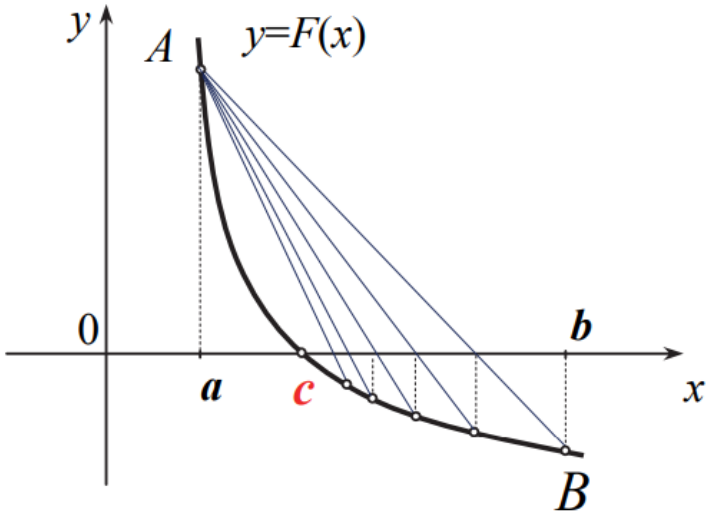
$$x_1 = a - \frac{f(a)(b - a)}{f(b) - f(a)} \quad \text{или} \quad \boxed{x_1 = x_0 - \frac{f(x_0)(b - x_0)}{f(b) - f(x_0)}}$$

Это формула метода хорд для первого шага.


y
A
y=F(x)
0
a
c
c0
b
x
B0
B


y
A
y=F(x)
0
a
c
c1
c0
b
x
B1
B0
B


y
A
y=F(x)
0
a
c
c2
c1
c0
b
x
B2
B1
B0
B


y
A
y=F(x)
0
a
c
b
x
B

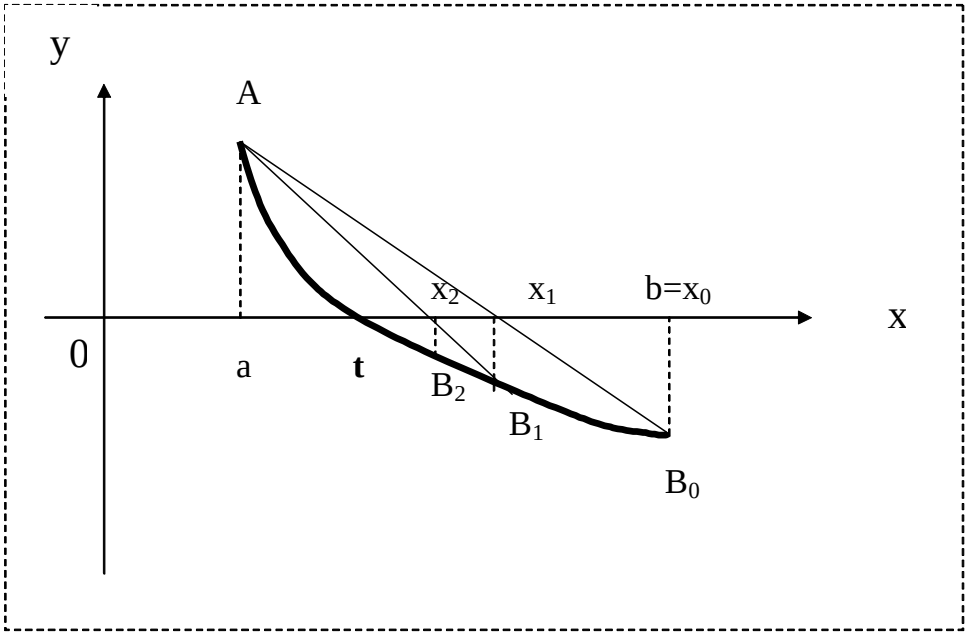
Сейчас корень внутри отрезка **[x_1,b]**.

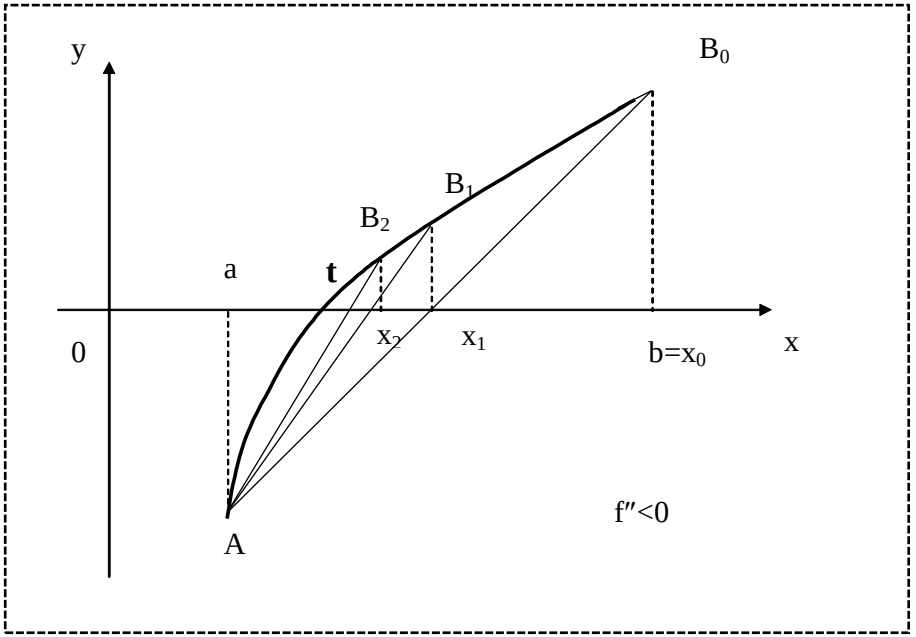
Далее:

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

Здесь B – 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.

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

Итак до (n+1)-го шага:

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

Здесь A-неподвижный конец хорды.

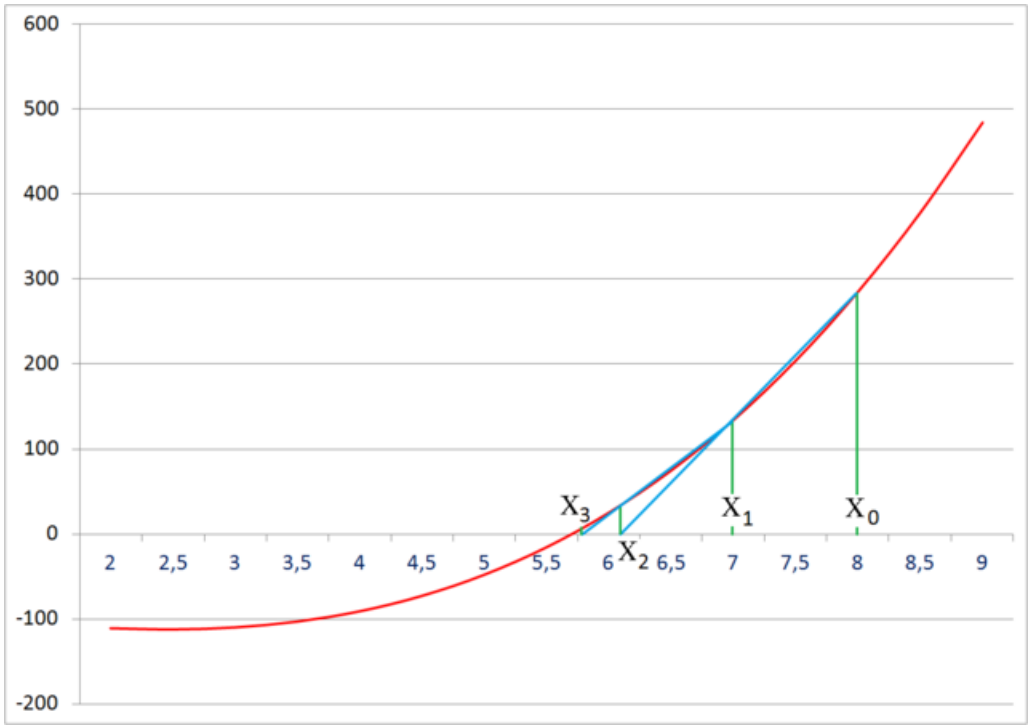
Условие неподвижной точки для метода хорд $f(x)\ f''(x) > 0$. В принципе, оба конца отрезка могут быть подвижными. Метод хорд, как правило, сходится медленнее, чем следующий метод (касательных).

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 ε.

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



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

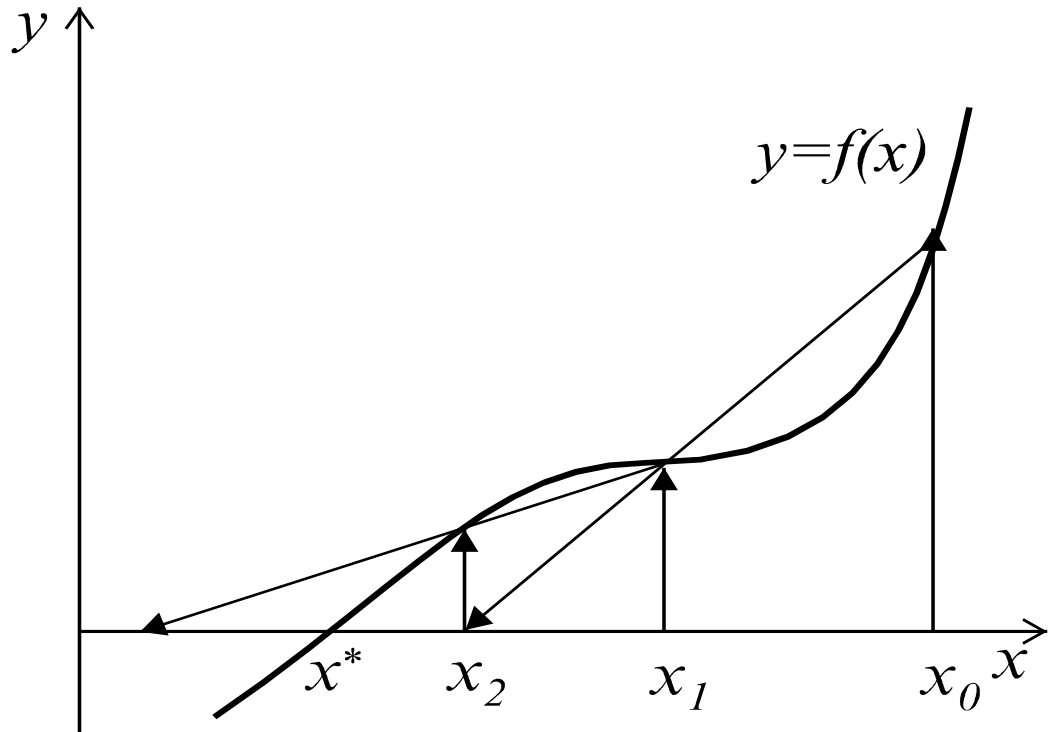
Расчетная формула метода Вегстейна:

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

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

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

Метод Вегстейна сходится медленнее метода хорд (секущих), однако, требует в 2 раза меньше вычислений $\mathbf{f(x)}$ 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 ε. Иногда удается вычислить корень уравнения и в том случае, когда условия сходимости не выполняются.

Метод одношаговый ($\boldsymbol{m} = 1$), 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.

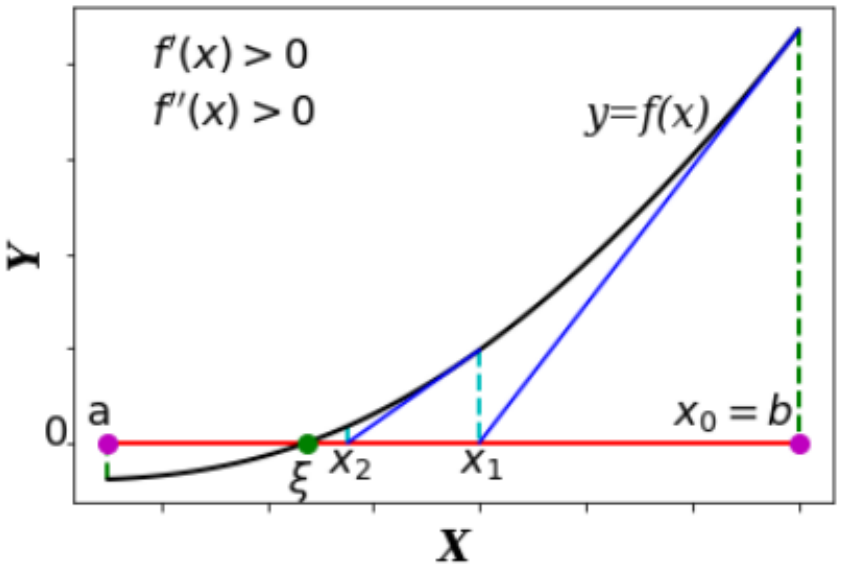
Выбор начальной точки зависит от свойств функции:

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

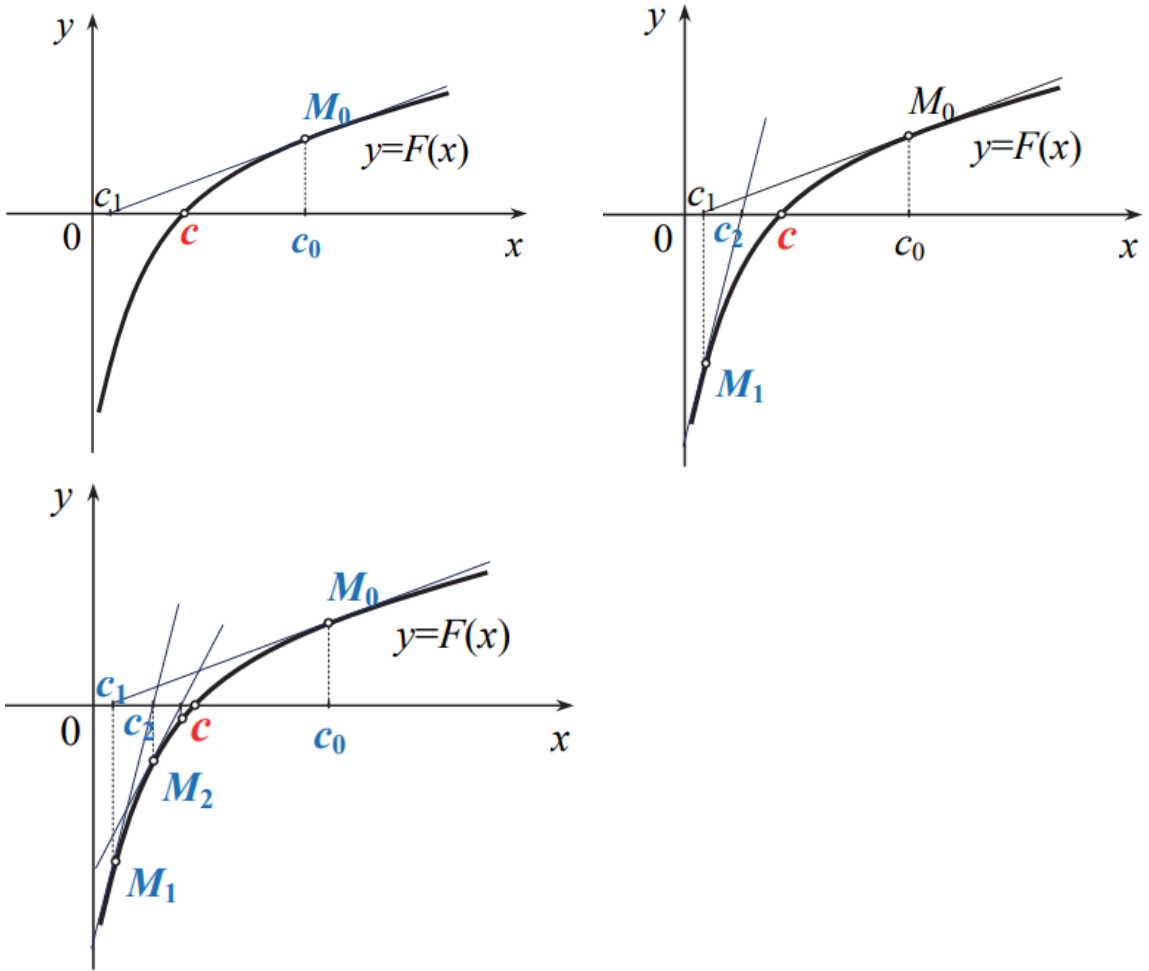
Очередное приближение x_{i+1} вычисляется по формуле:

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

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 ε.



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.

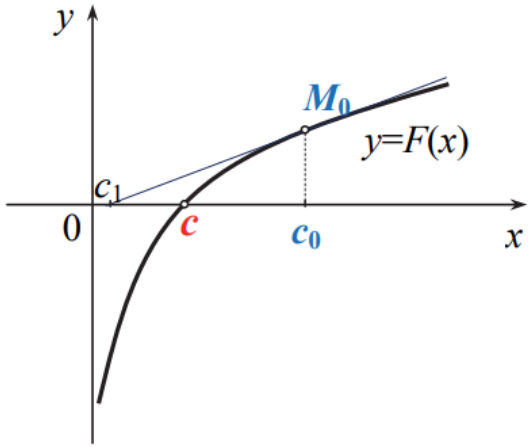
Тогда найдется такая δ-окрестность точки x = t, что при любом выборе начального приближения x_0 на отрезке $[t-\delta, t+\delta] \in [a, b]$.

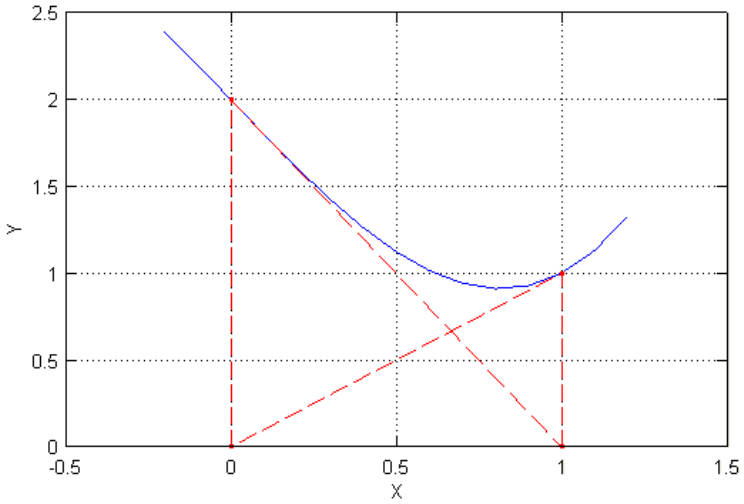
Существует бесконечная итерационная последовательность $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$ и эта последовательность сходится к корню t.

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:

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

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

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

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

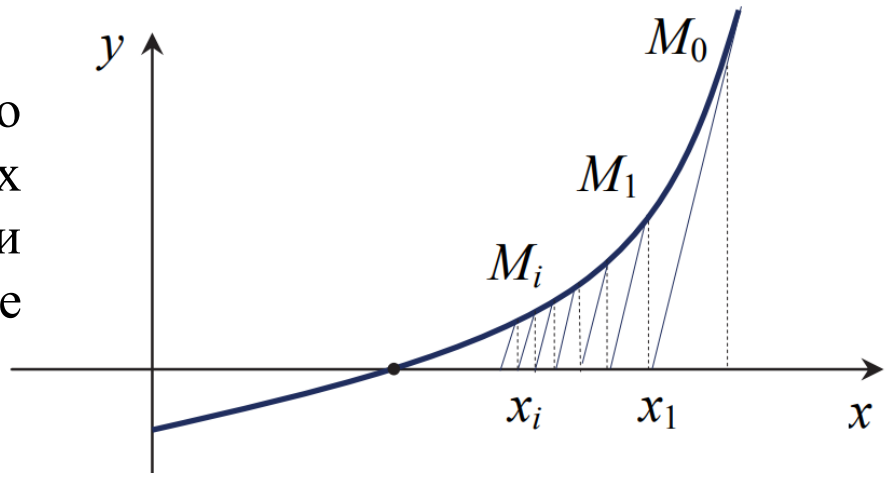
Если производная f'(x) мало изменяется на отрезке [a, b], то можно положить:

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

Тогда следующее приближение вычисляется по формуле:

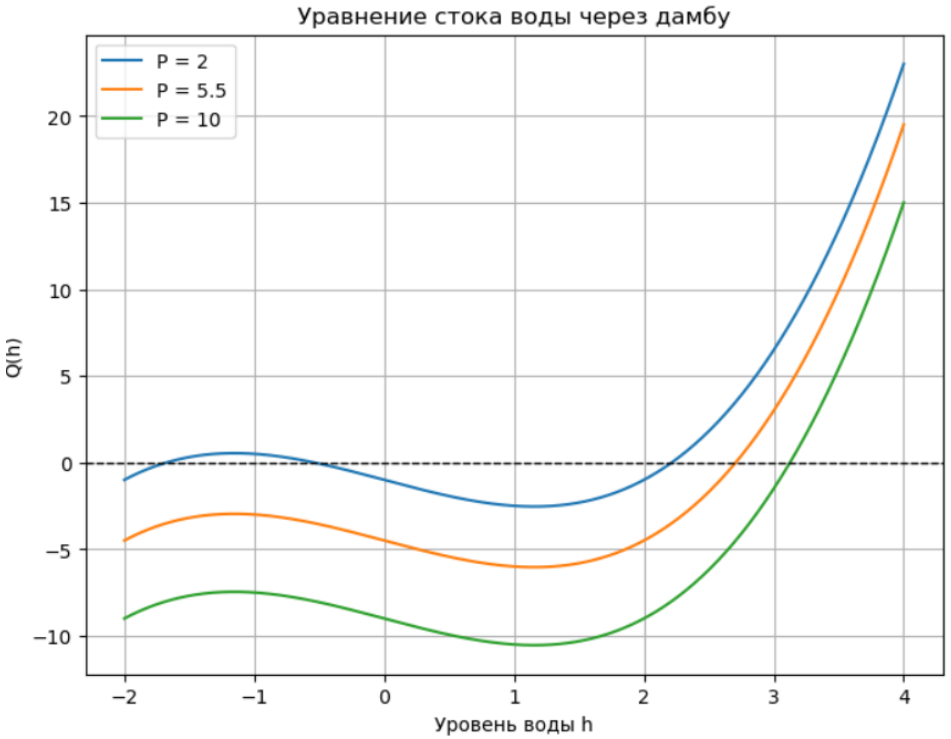
$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_0)}$$

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 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.

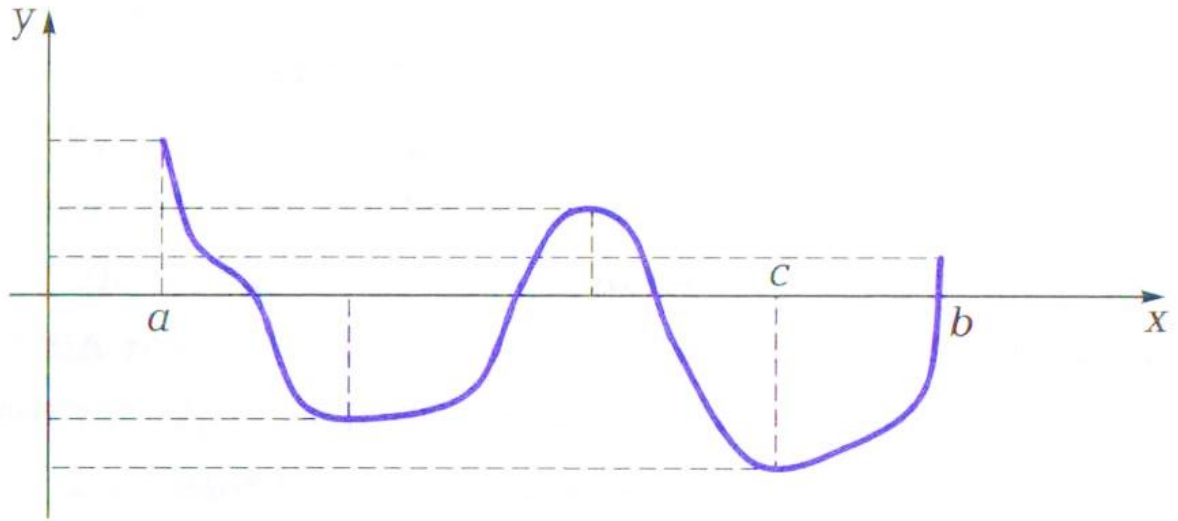
Стандартной задачей оптимизации является поиск экстремумов функции одной переменной.

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

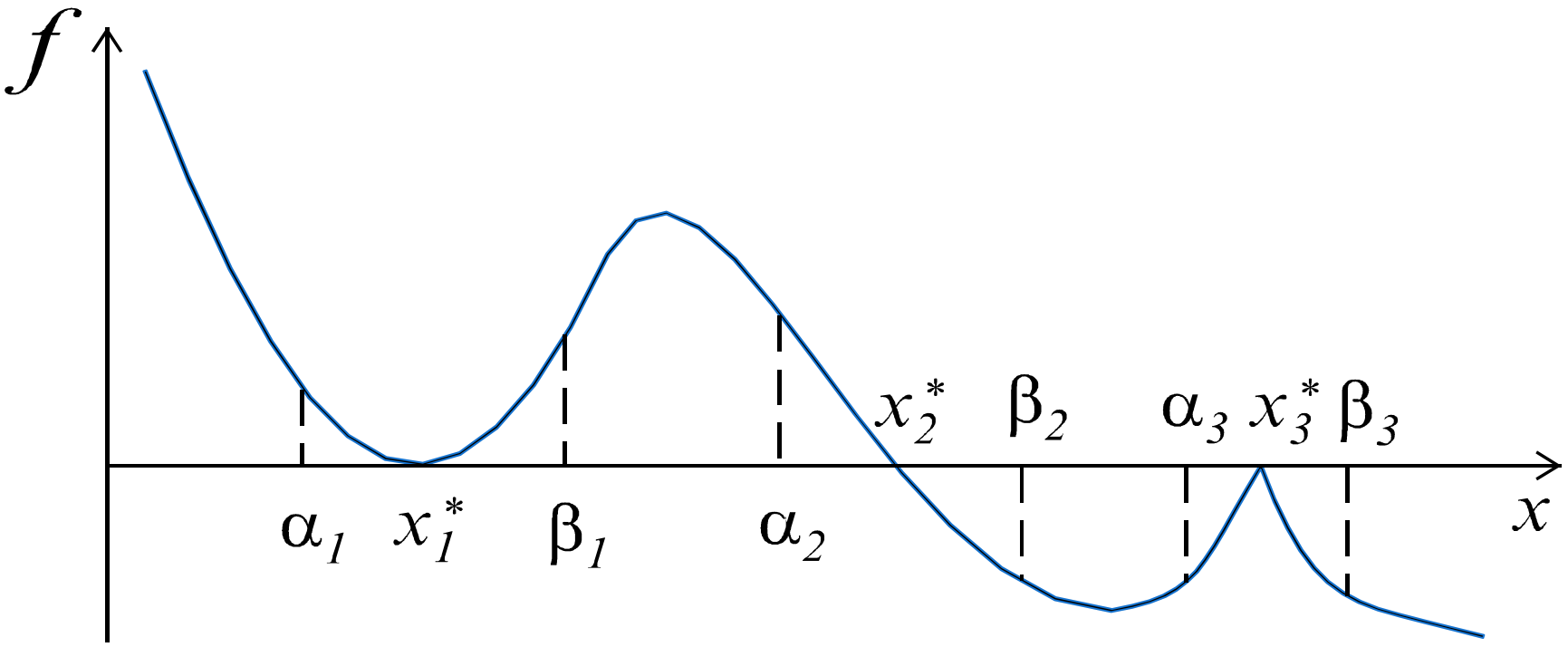
x = a – самый «высокий» из максимумов,

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 $f(c + {}^{e}/_{2})$

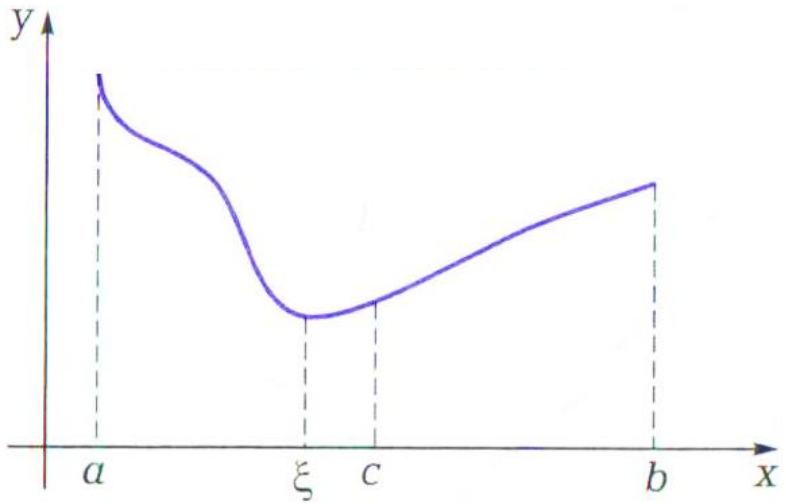
- при $f(c - {}^{e}/_{2}) < f(c + {}^{e}/_{2})$ делению пополам подлежит отрезок [a; c];
- при $f(c - {}^{e}/_{2}) > f(c + {}^{e}/_{2})$ делению пополам подлежит отрезок [c; b].

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 = c.



ЧИСЛЕННЫЕ МЕТОДЫ РЕШЕНИЯ СИСТЕМ НЕЛИНЕЙНЫХ УРАВНЕНИЙ

Постановка задачи

Требуется решить систему нелинейных уравнений (СНУ) вида.

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

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

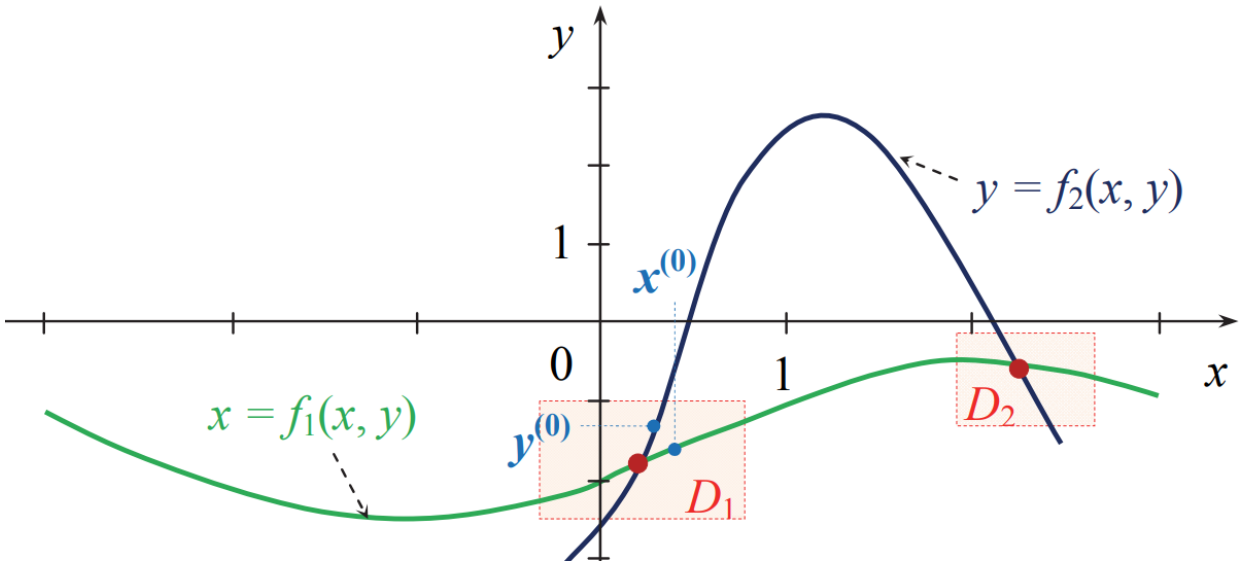
Метод простых итераций:

Для построения схемы интеграций систему уравнений представим в виде

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

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 $x_1^{(1)}, x_2^{(1)}, \dots, x_n^{(1)}$:

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

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

Таким образом, схема итераций метода простых интеграций имеет вид:

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

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:

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

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

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

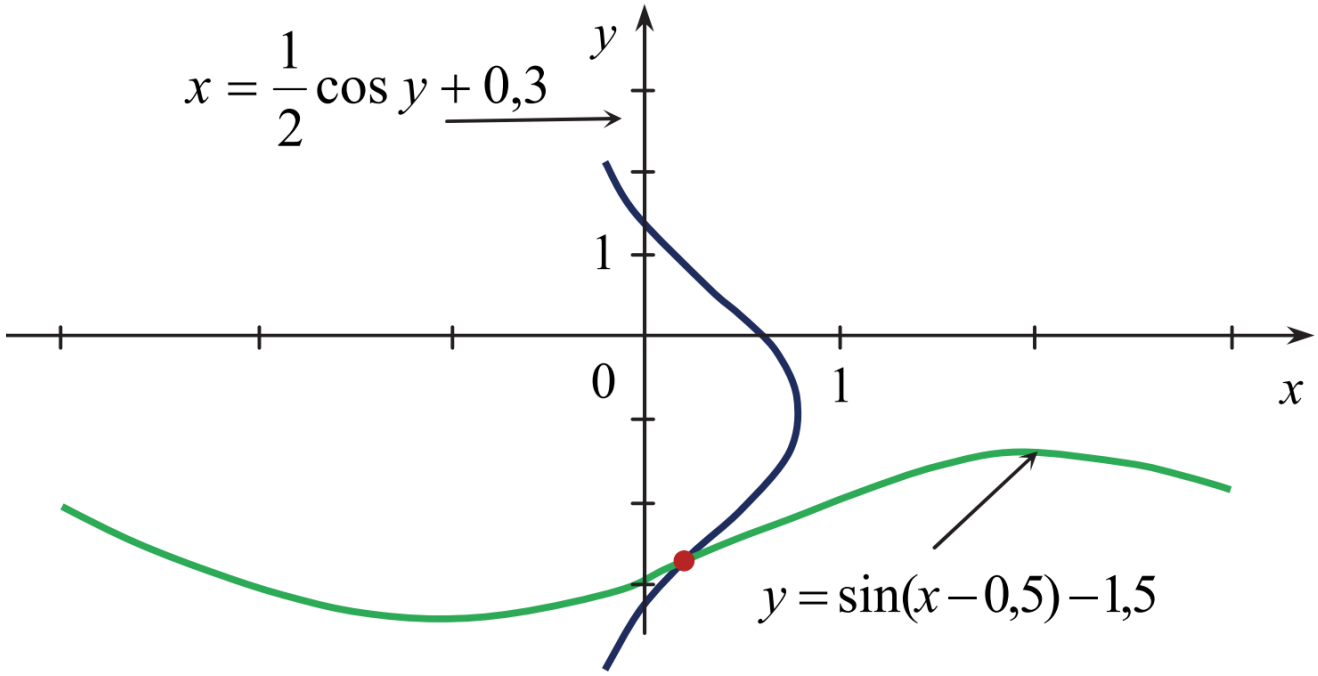
Решение.

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

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

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: $\left\|x^{(k)} - x^{(k-1)}\right\| < \varepsilon = 0{,}001$	Проверка условия: $\left\|y^{(k)} - y^{(k-1)}\right\| < \varepsilon = 0{,}001$
	$x^{(k)}$	$y^{(k)}$		
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	**выполняется**	не выполняется
13	0,17881	–1,81619	**выполняется**	**выполняется**

Для ускорения сходимости можно ввести параметр b:

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

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

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

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:

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

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

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

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

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

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

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:

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

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

Номер итерации, k	Приближение корня		Проверка условия: $\left\|x^{(k)} - x^{(k-1)}\right\| < \varepsilon = 0{,}001$	Проверка условия: $\left\|y^{(k)} - y^{(k-1)}\right\| < \varepsilon = 0{,}001$
	$x^{(k)}$	$y^{(k)}$		
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	**выполняется**	**выполняется**

Пусть требуется решить систему нелинейных уравнений вида:

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

Метод Ньютона решения системы состоит в построении итерационной последовательности:

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

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:

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

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

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

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

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

Запишем эту систему в матричной форме:

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

↑

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:

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

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

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

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:

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

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

Тогда следующие приближения будут вычисляться схеме:

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

А приращения будут находиться из систем вида:

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

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

где k – номер приближения.

Номер итерации, k	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:

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

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

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

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 $x^{(0)}$:

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

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:

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

Номер итерации, k	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 **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:

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

где:

v_0 = 5 м/с – начальная скорость.

θ – угол броска (в радианах).

t – время полёта.

g = 9.8 м/с^2 – ускорение свободного падения.

k = 0.1 – коэффициент сопротивления воздуха.

Для достижения максимальной высоты, θ и t должны удовлетворять условию:

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

Найдите пару (θ, 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).

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

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 интерполирования 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).

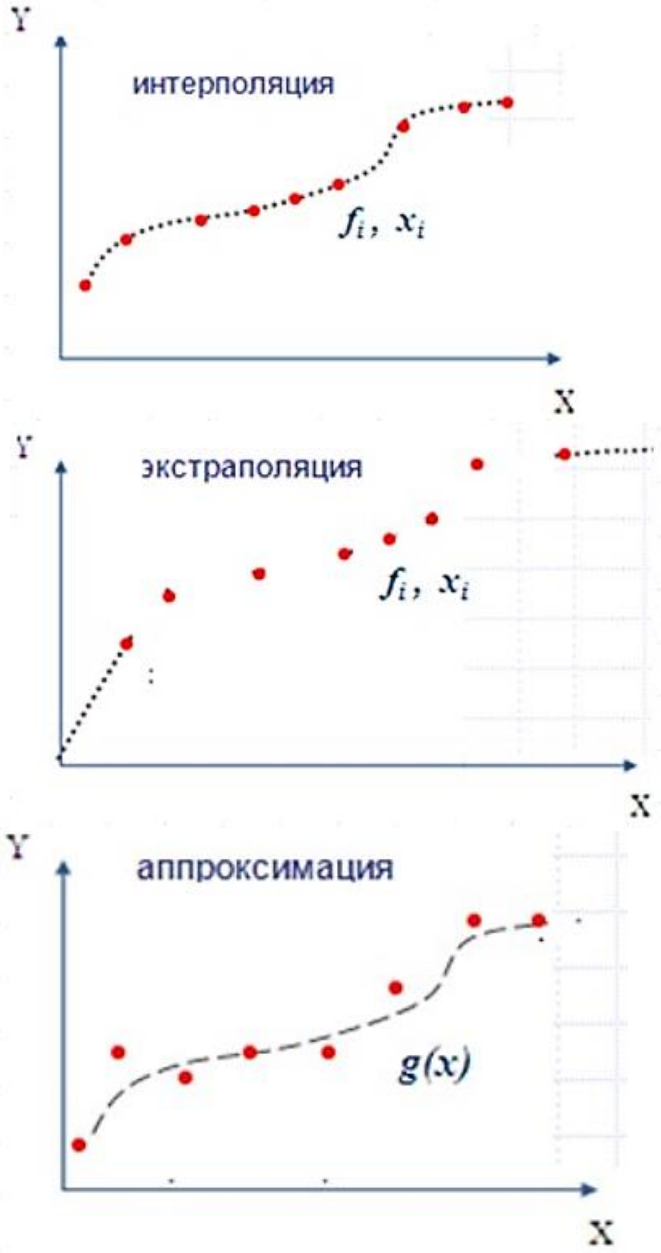
Подход:

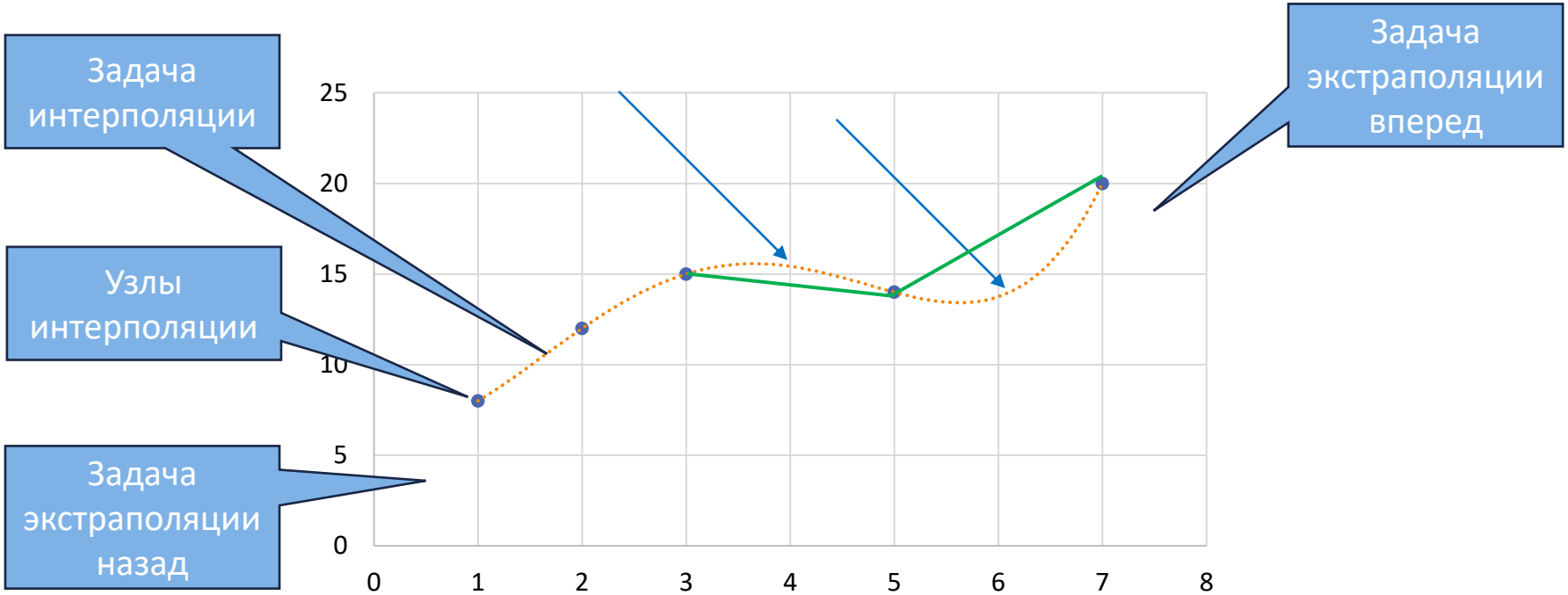
1. Интерполяция – функция должна пройти через все точки. Способ нахождения промежуточных значений величины по имеющемуся дискретному набору известных значений.
2. Экстраполяция – особый тип аппроксимации, при котором функция аппроксимируется вне заданного интервала, а не между заданными значениями.
3. 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.

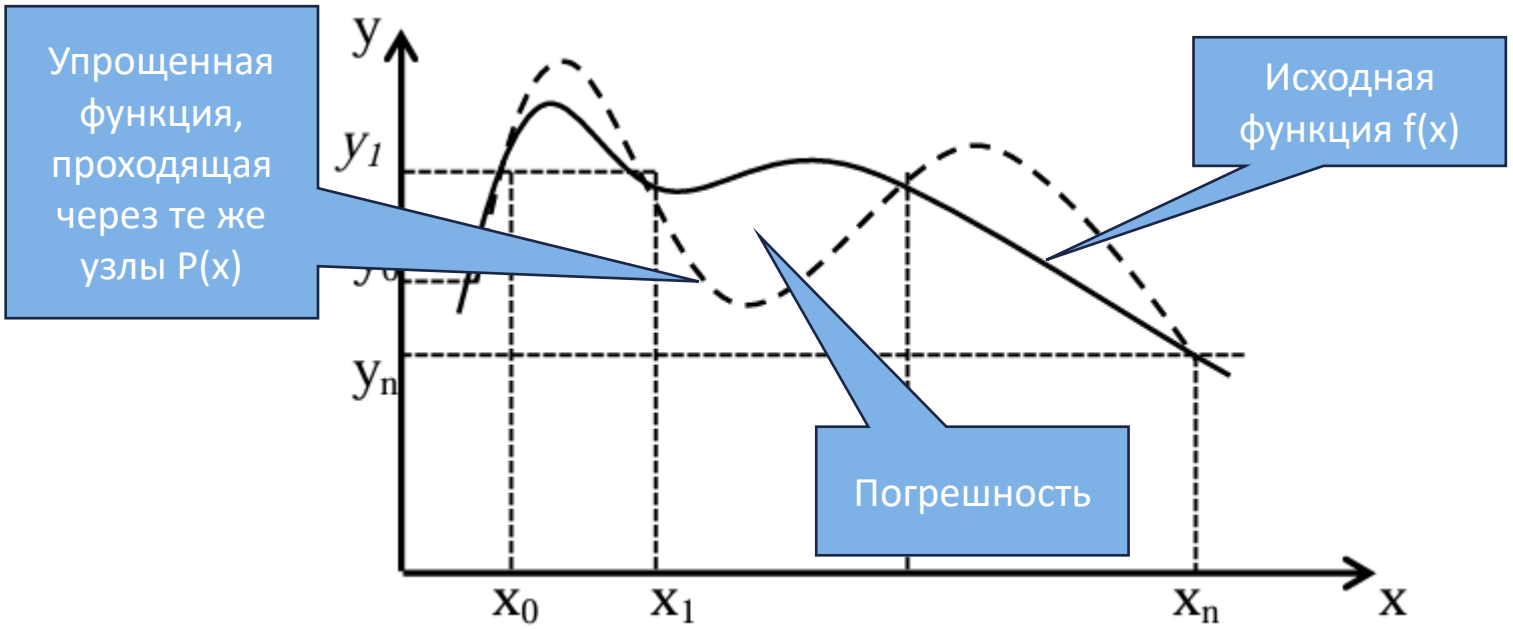
2. Экстраполяция.

3. Аппроксимация.





Способ восстановления пропущенных данных при сложном виде аналитической функции


Упрощенная функция, проходящая через те же узлы P(x)
Исходная функция f(x)
Погрешность
y
y1
yn
x0
x1
xn
x

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 $\{y_i\}$ ($i = 0, 1\ldots n$).

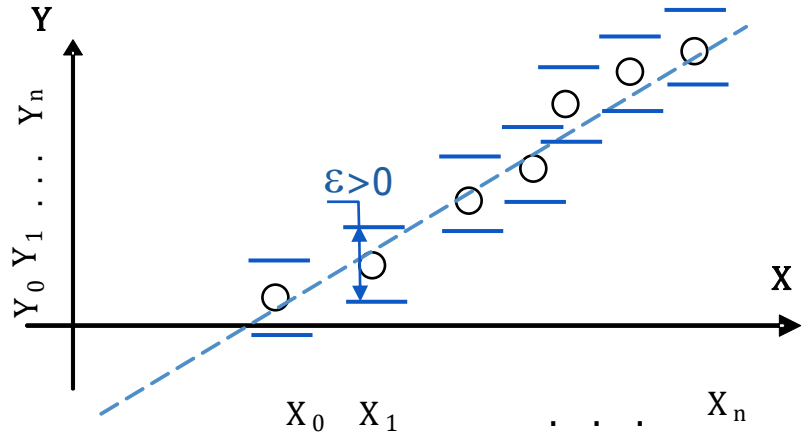
Эти значения – либо результаты расчетов, либо экспериментальные данные. На практике же могут понадобиться значения величины y и в других точках, отличных от узлов x_i. Однако получить эти значения можно лишь путем очень сложных расчетов или проведением дорогостоящих экспериментов.

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.

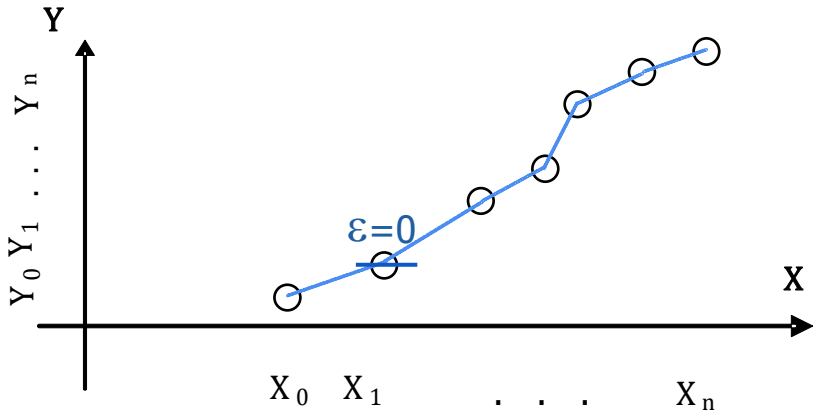
Общая погрешность аппроксимирующей функции может быть выражена как сумма локальных погрешностей в точках с координатами x_i.

$$E = \Sigma e_i ,$$

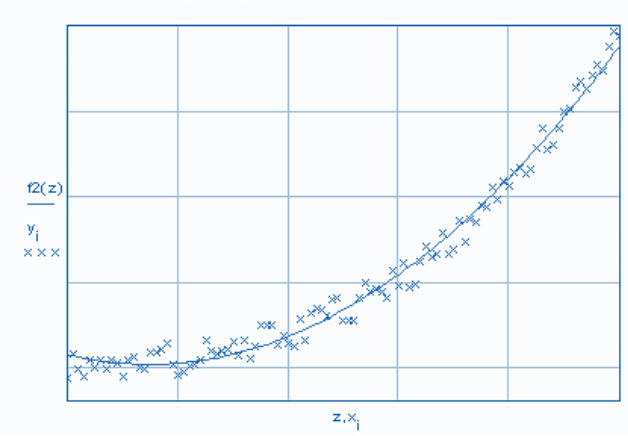
где $e_i = |F(x_i) - f(x_i)|$.

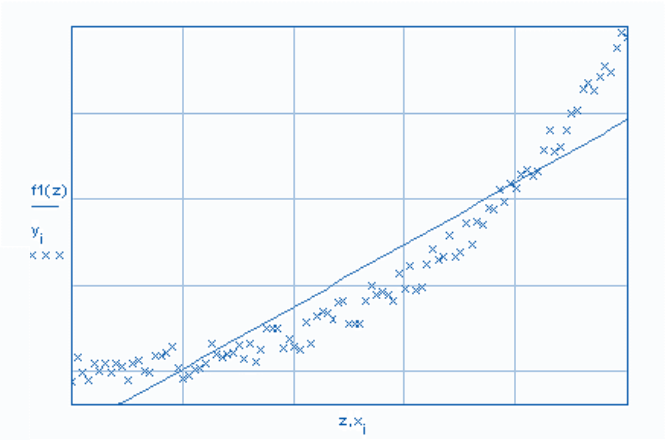


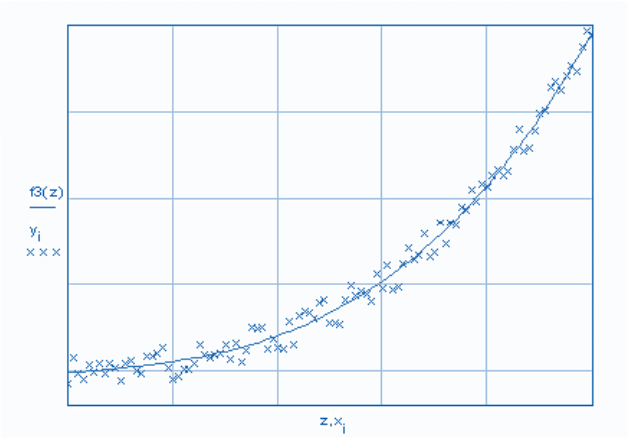
В общем случае при аппроксимации $0 \le e_i \le \varepsilon$. В случае, если $\varepsilon = 0$, т.е. налагается условие строгого совпадения значений функций F(x) и f(x) в заданных точках x_i, то данный вид аппроксимации называется интерполяцией.

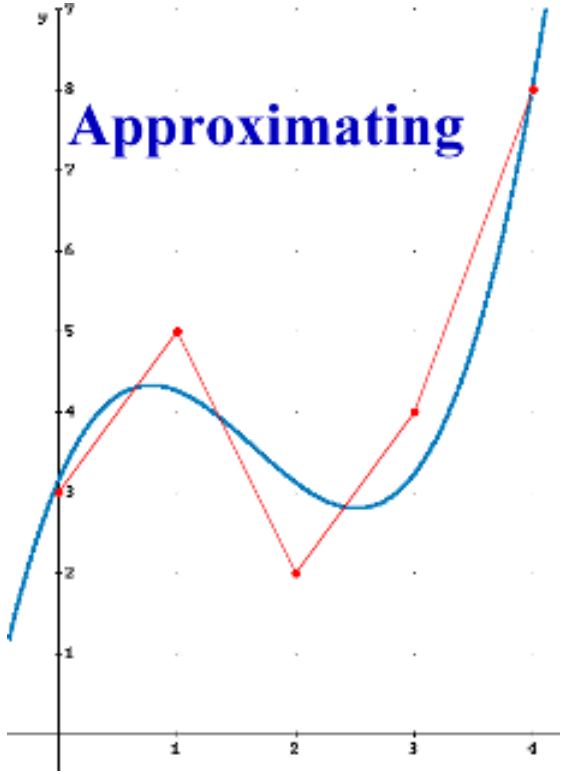


Чаще всего аппроксимацию применяют для экспериментального нахождения некоторых зависимостей.







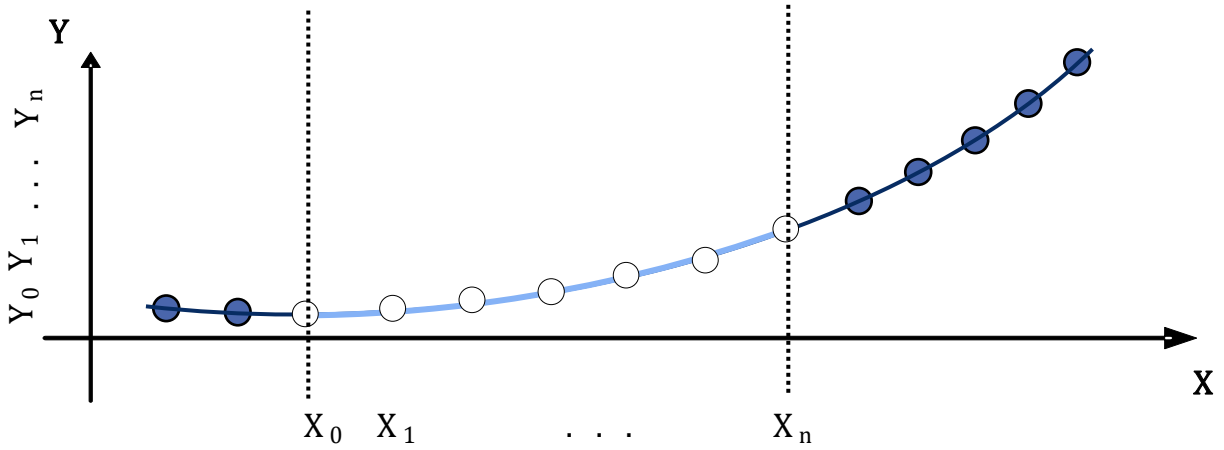


Чем проще аппроксимирующая функция, тем меньше времени требуется для решения задачи аппроксимации. Чем больше узлов, тем меньше погрешность. Для каждой конкретной аппроксимирующей функции нужно стремиться выбрать такой способ аппроксимации, который обеспечивает минимальную погрешность при минимальном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

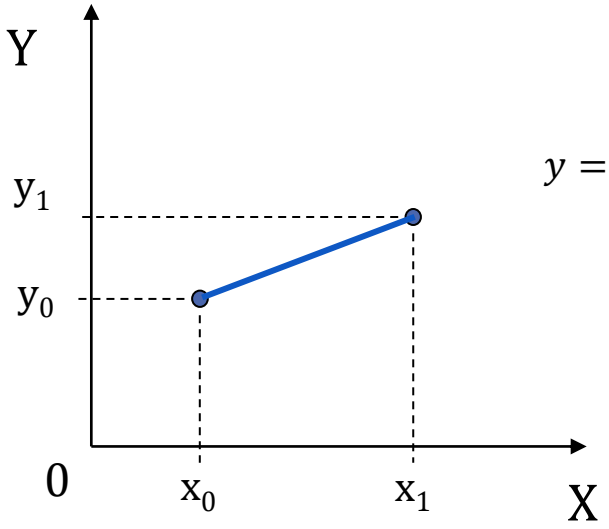
Как получают исходные данные: задаем интервал, шаг, количество узлов, вычисляем значения с помощью функции f(x).

Определять большое количество значений вычислительно дорого.

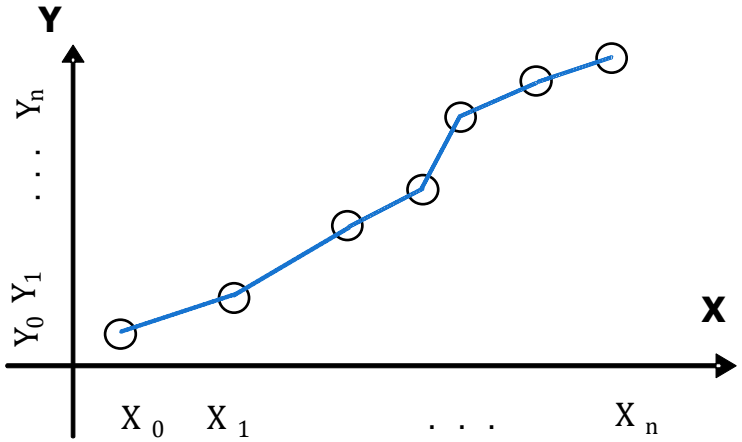
Интерполяция бывает глобальной – F(x) проходит через ВСЕ точки заданного интервала $[x_0, x_n]$, либо локальной (кусочной) – f(x) на указанном интервале интерполируется несколькими $F_1(x)$, $F_2(x)$... $F_k(x)$.

Типы интерполяции:

- полиномиальная,
- тригонометрическая,
- экспоненциальная.

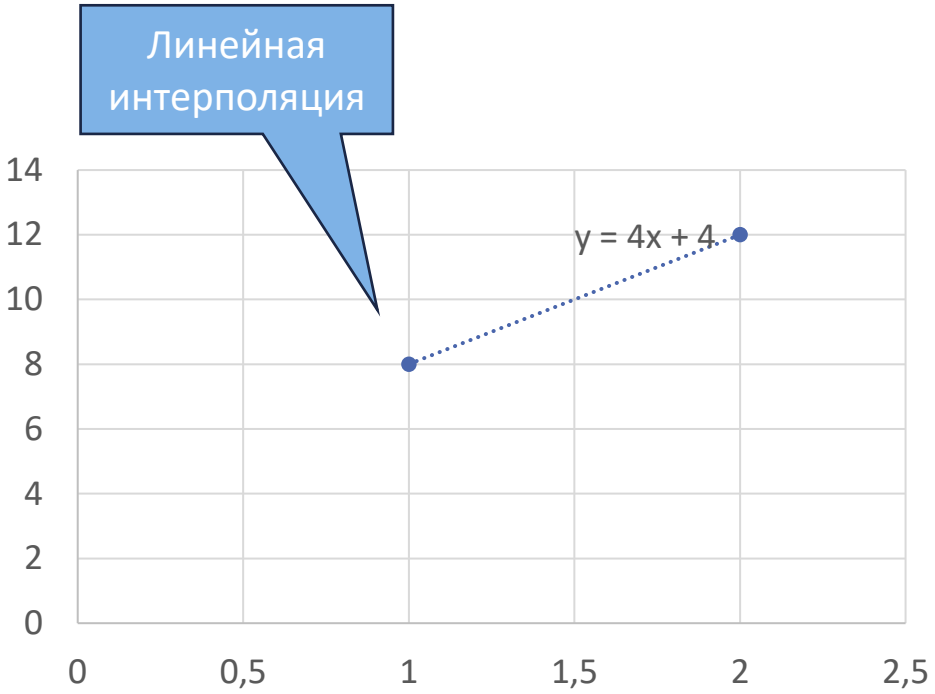


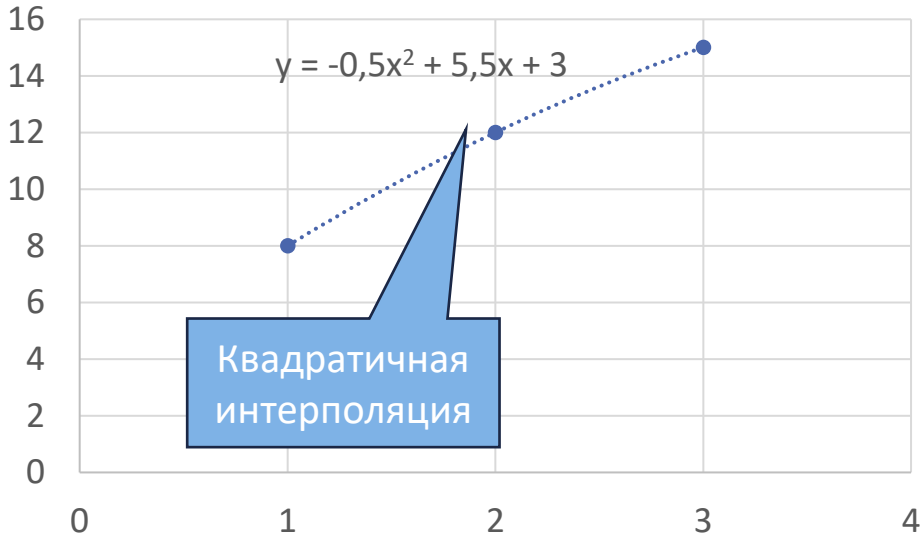
$$y = \frac{y_1 - y_0}{x_1 - x_0}(x - x_0) + y_0$$

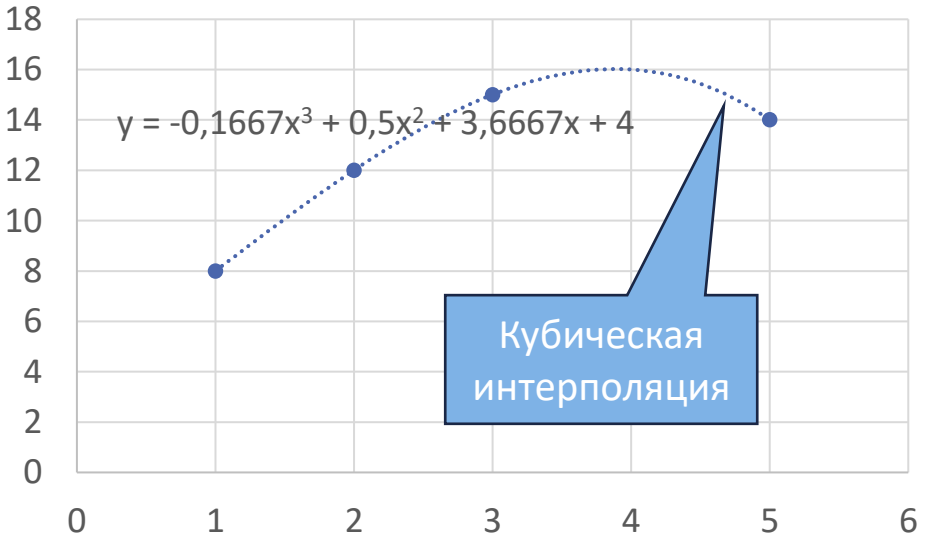


Линейная интерполяция – простейший вид локальной полиномиальной интерполяции – замена f(x) множеством линейных функций $F_1(x)$, $F_2(x)$... $F_k(x)$, каждая из которых соединяет лишь две точки.

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

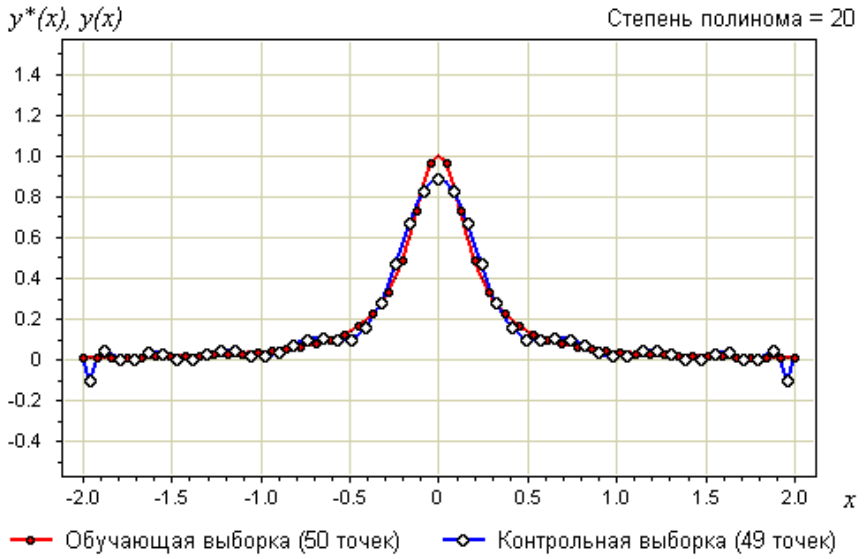


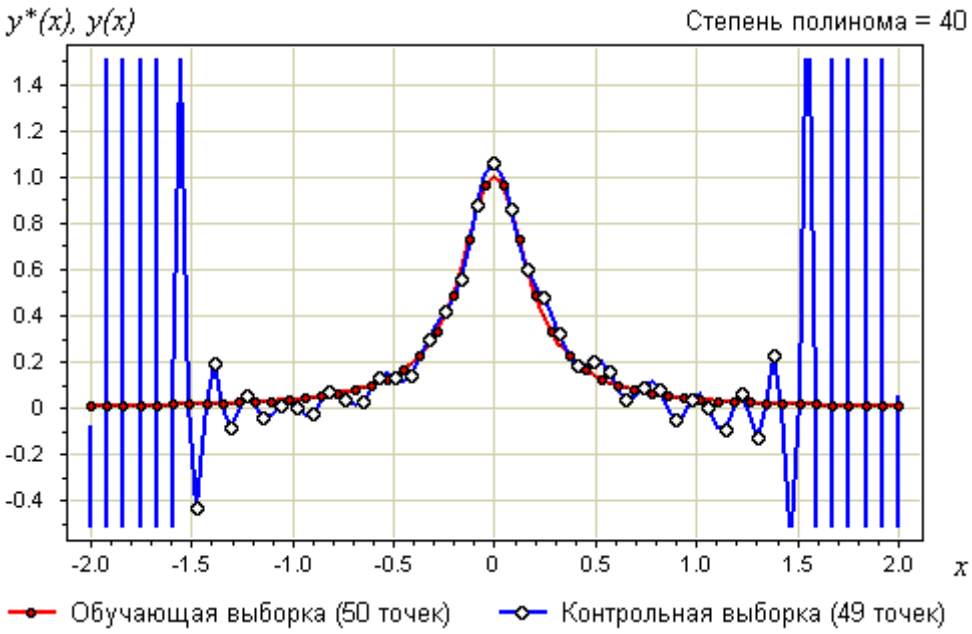


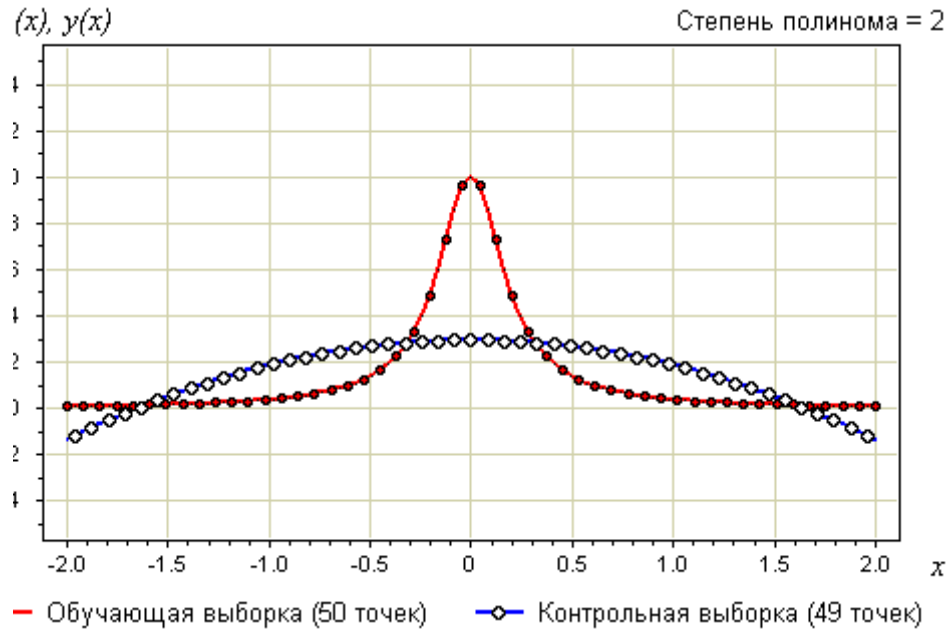


Стоит избегать:

- 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:

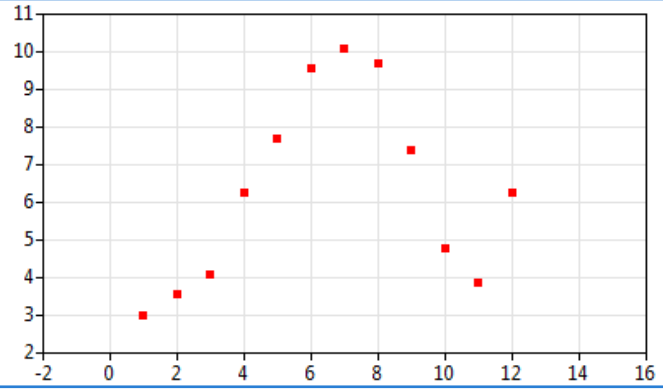
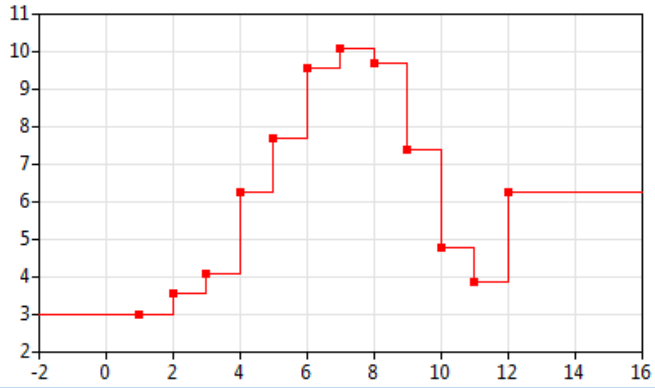
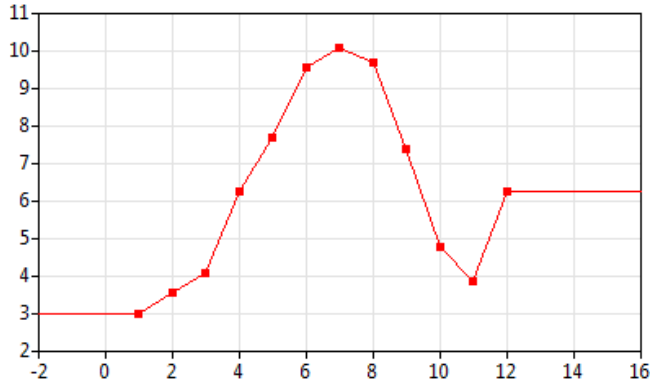
$$\tilde{f}(x) = f(x_i), i : \forall j \neq i, |x - x_j| > |x - x_i|$$

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)

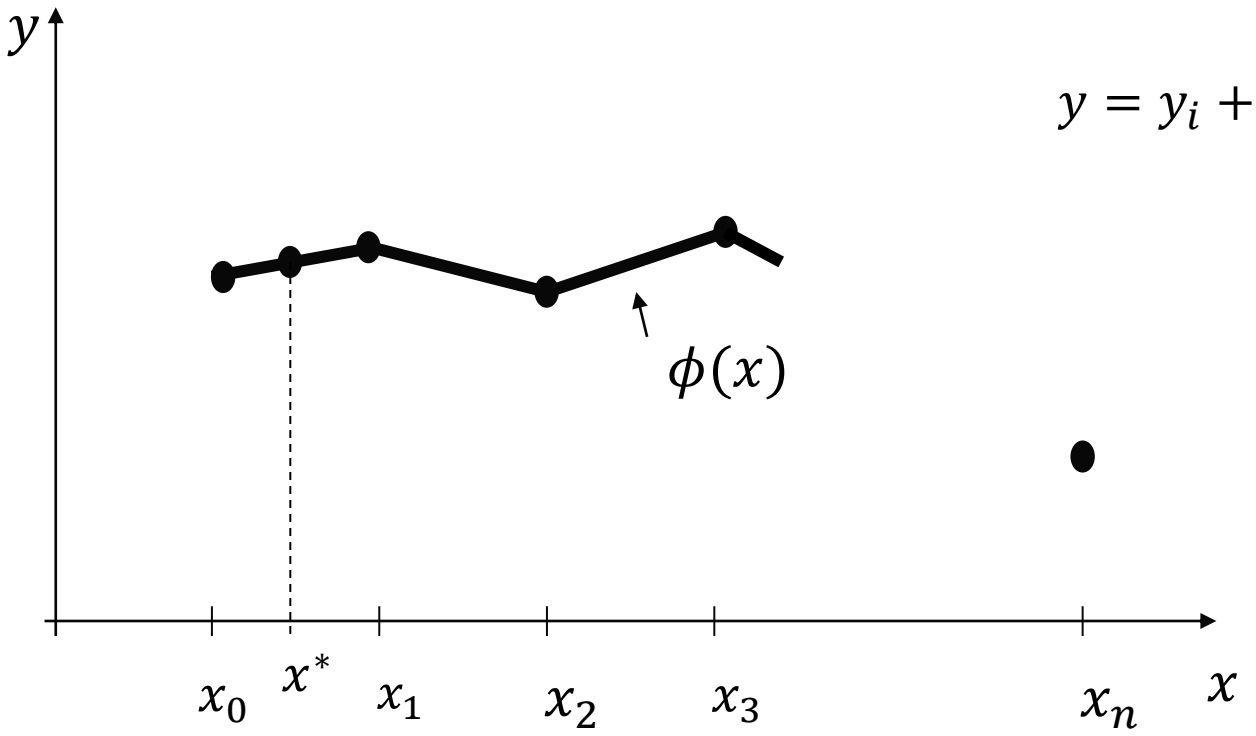
Интересно понять, с какой точностью интерполяционные формулы аппроксимируют функцию f. Предположим, что производная функции f ограничена величиной g. Тогда на отрезке $[x_i, x_{i+1}]$ функция f не может отклониться от линейной интерполяции более, чем на $h\left(g - \left|\frac{f(x_{i+1}) - f(x_i)}{x_{i+1}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

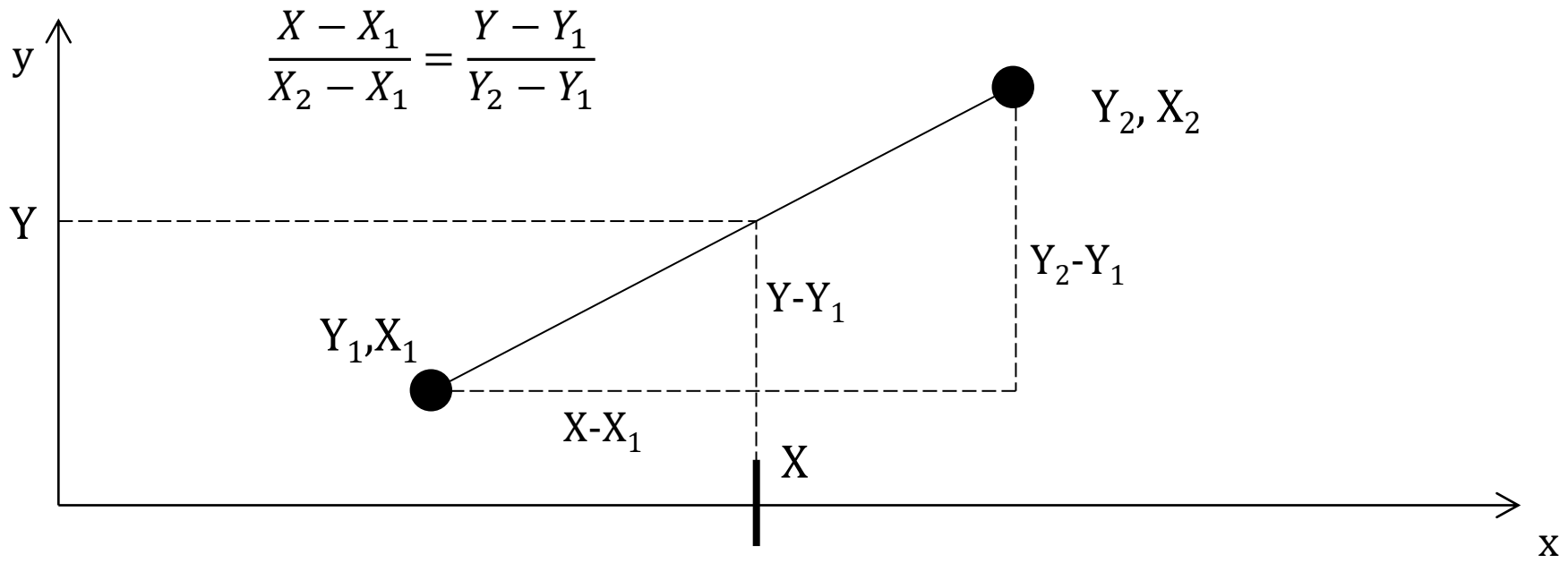
Тип интерполяции	Описание и пример
	Интерполяция не используется.
	Ступенчатая интерполяция. Значение функции на интервале между двумя точками равно значению функции в точке с меньшим аргументом.
	Линейная интерполяция. Точки соединяются прямыми линиями.

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

Теорема. Если заданы попарно различные узлы $x_0, x_1, x_2,..., x_n$ и значения $f(x_0), f(x_1),..., f(x_n)$, 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 – $x_0, x_1, ..., x_n$, 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 $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$, 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:

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

В результате будет получен полином $L(x) = A_0 + A_1x + A_2x^2 + \ldots A_nx^n$, 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,

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

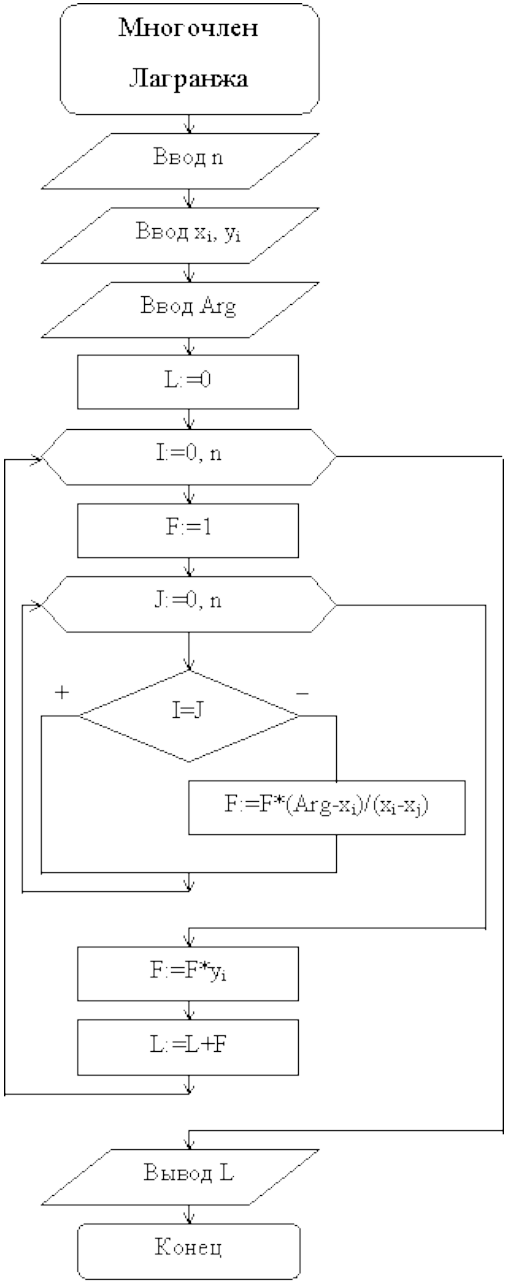
Подставив эти выражения в формулу базисного полинома и вынеся h за знаки перемножения в числителе и знаменателе, получим

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

Теперь можно ввести замену переменной

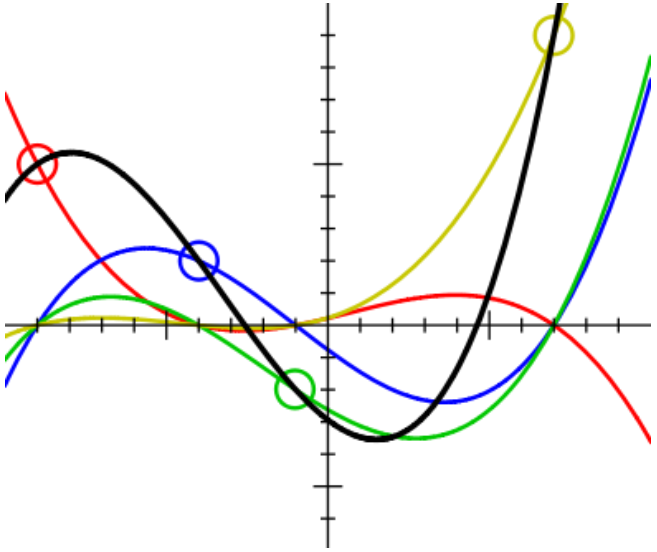
$$y = \frac{x - x_0}{h}$$

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.



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

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



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, …, n. Тогда для погрешности интерполяции в точке x из [a, b] справедлива оценка:

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

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

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

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

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

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):

$$f'(x) = \frac{1}{2}x^{-\frac{1}{2}}, \qquad f''(x) = -\frac{1}{4}x^{-\frac{3}{2}}, \qquad f'''(x) = \frac{3}{8}x^{-5/2}.$$

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

получим оценку погрешности в точке x = 116:

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

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

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

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 ($0 \le t \le 1$).

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.

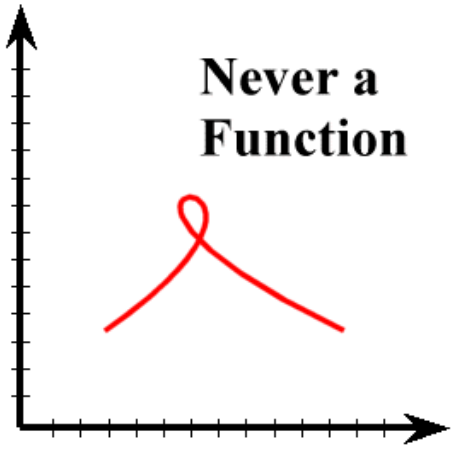
Параметрическая форма позволяет представить замкнутые и многозначные кривые.

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.

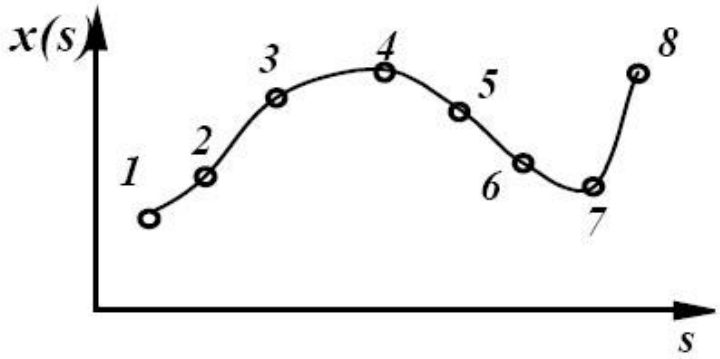
Первая причина этого в том, что формы объектов в прикладных задачах не должны зависеть от системы координат.

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.

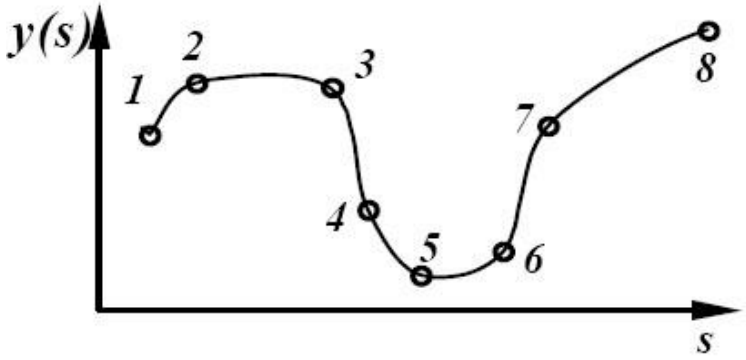


Пример плоской кривой в параметрическом виде:

x = x(s)



y = y(s)



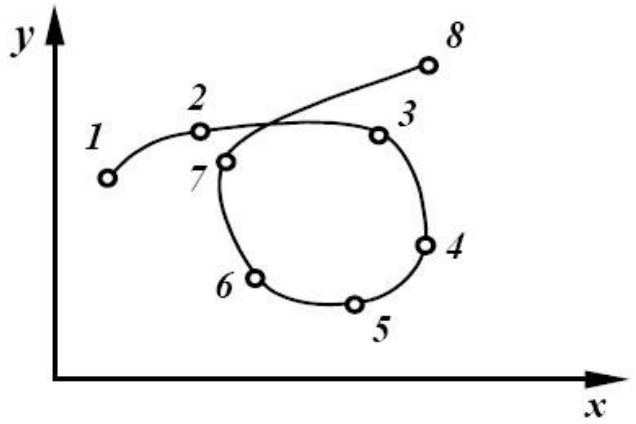
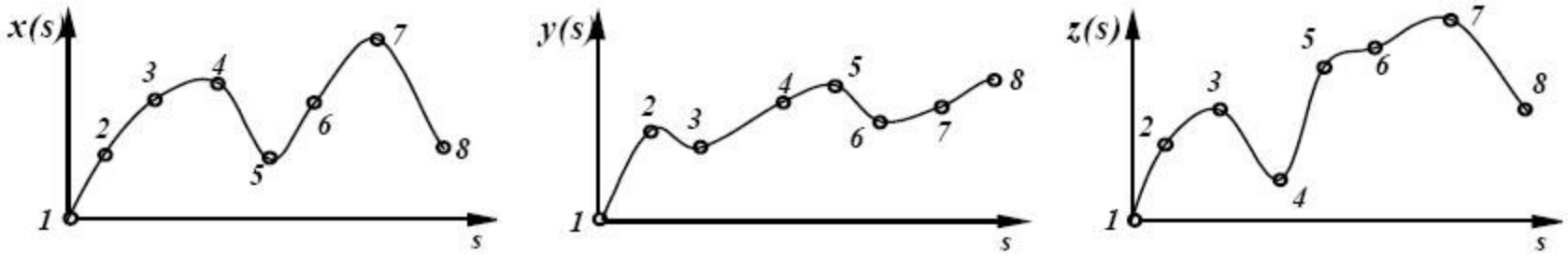


График кривой
y = ?(x)

Пример пространственной (3d) параметрическом виде:

$x = x(s)$ $y = y(s)$ $z = z(s)$



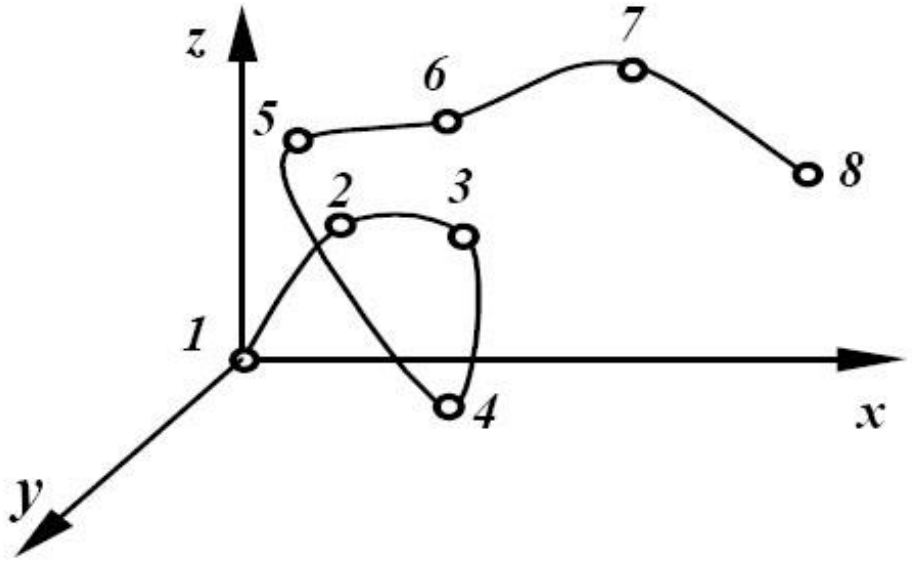


График кривой
$z = ?(x,y)$

Векторное представление производной (т.е. касательной):

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

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:

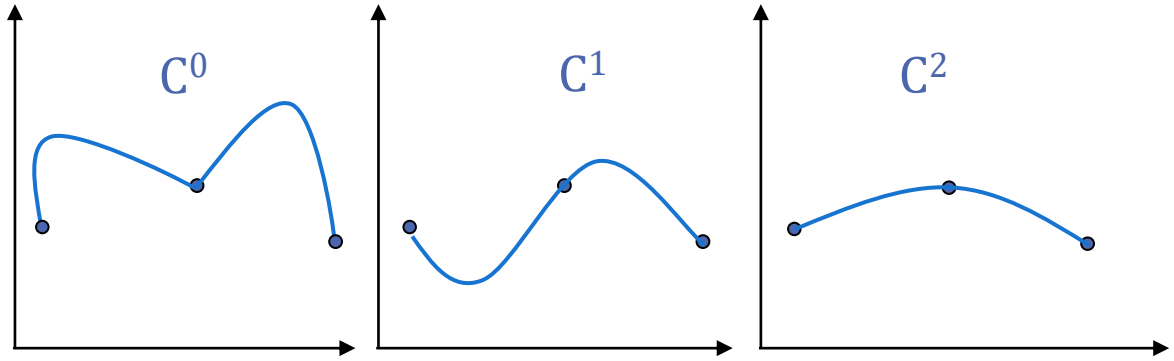
- формы Эрмита (Hermite)
- кривые Безье (Bézier)
- кубические сплайны .

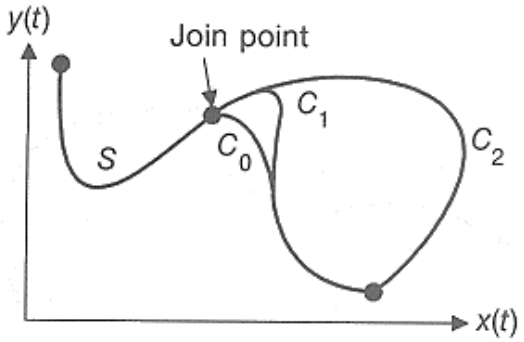
Непрерывность нулевого порядка по параметру, C^0 – означает, что кривые встречаются, т.е. $F_{k-1}(x_p) = F_k(x_p)$.

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.

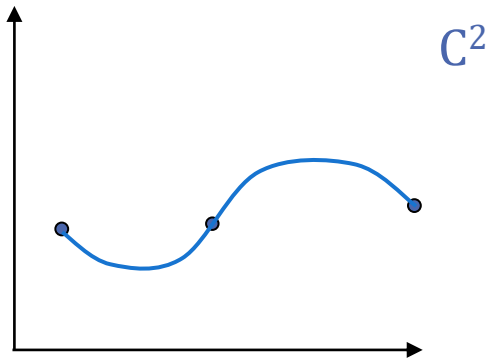


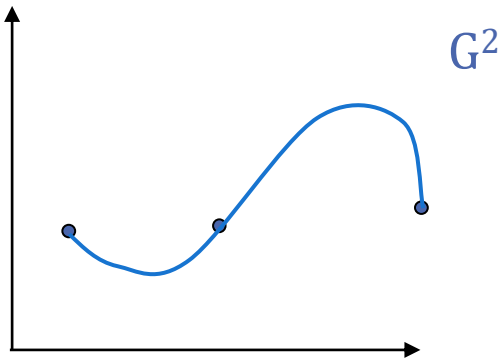


Геометрическая непрерывность нулевого порядка по параметру, G^0 аналогична C^0 – $F_{k-1}(x_p)=F_k(x_p)$.

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 $F''_{k-1}(x_p) = b, F''_k(x_p)$, $b \neq 0.1$.





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

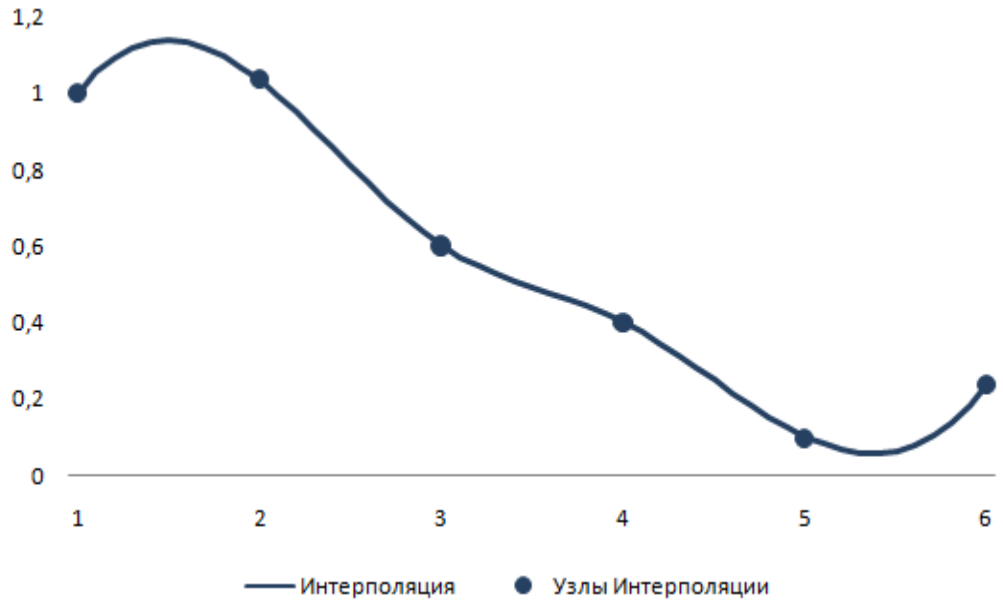
Иными словами, сплайнами называется набор функций, который вместе с несколькими производными этих функций непрерывен на отрезке **[a, b]**, а на каждом частном интервале этого отрезка $[x_i, x_{i+1}]$ в отдельности является некоторым многочленом невысокой степени.

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),..., (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

$$P_n(x_i) = y_i\,, \qquad i = 0,1,\ldots,n$$

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, ..., 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 z + 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:

$$F(x) = a_0 + a_1 x + a_2 x^2 + \cdots + l_n(x)$$

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): $F(x) = a_0 + a_1 x$.

$$\phi(x) = a_0 + a_1 x + a_2 x^2$$

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

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

$$\begin{cases} a_0 + a_1 x_{i-1} + a_2 x_{i-1}^2 = y_{i-1} \\ a_0 + a_1 x_i + a_2 x_i^2 = y_i \\ a_0 + a_1 x_{i+1} + a_2 x_{i+1}^2 = y_{i+1} \end{cases}$$

$$\begin{vmatrix} 1 & x_{i-1} & x_{i-1}^2 \\ 1 & x_i & x_i^2 \\ 1 & x_{i+1} & x_{i+1}^2 \end{vmatrix} \neq 0$$ – 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:

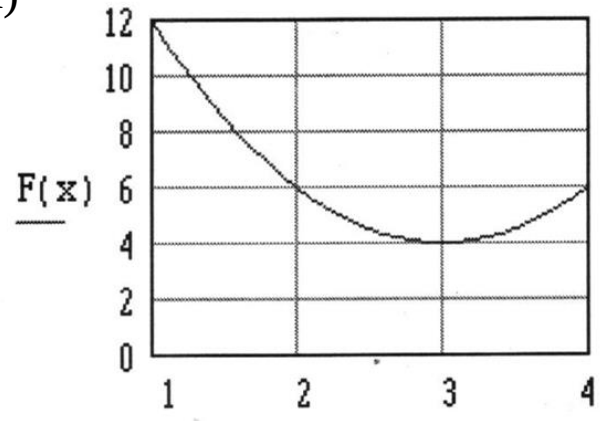
$$\begin{cases} F(x_0) = a_0 + a_1 x_0 + a_2 {x_0}^2 = a_0 + a_1 + a_2 = 12 \\ F(x_1) = a_0 + a_1 x_1 + a_2 {x_1}^2 = a_0 + a_1 \cdot 3 + a_2 \cdot 9 = 4 \\ F(x_2) = a_0 + a_1 x_2 + a_2 {x_2}^2 = a_0 + a_1 \cdot 4 + a_2 \cdot 16 = 6 \end{cases}$$

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

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

Построим график интерполяционного полинома F(x)

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



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:

$$\begin{aligned} x_0 &= -1.5 & f(x_0) &= -14,1014 \\ x_1 &= -0.75 & f(x_1) &= -0,931596 \\ x_2 &= 0 & f(x_2) &= 0 \\ x_3 &= 0.75 & f(x_3) &= 0,931596 \\ x_4 &= 1.5 & f(x_4) &= 14,1014. \end{aligned}$$

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

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

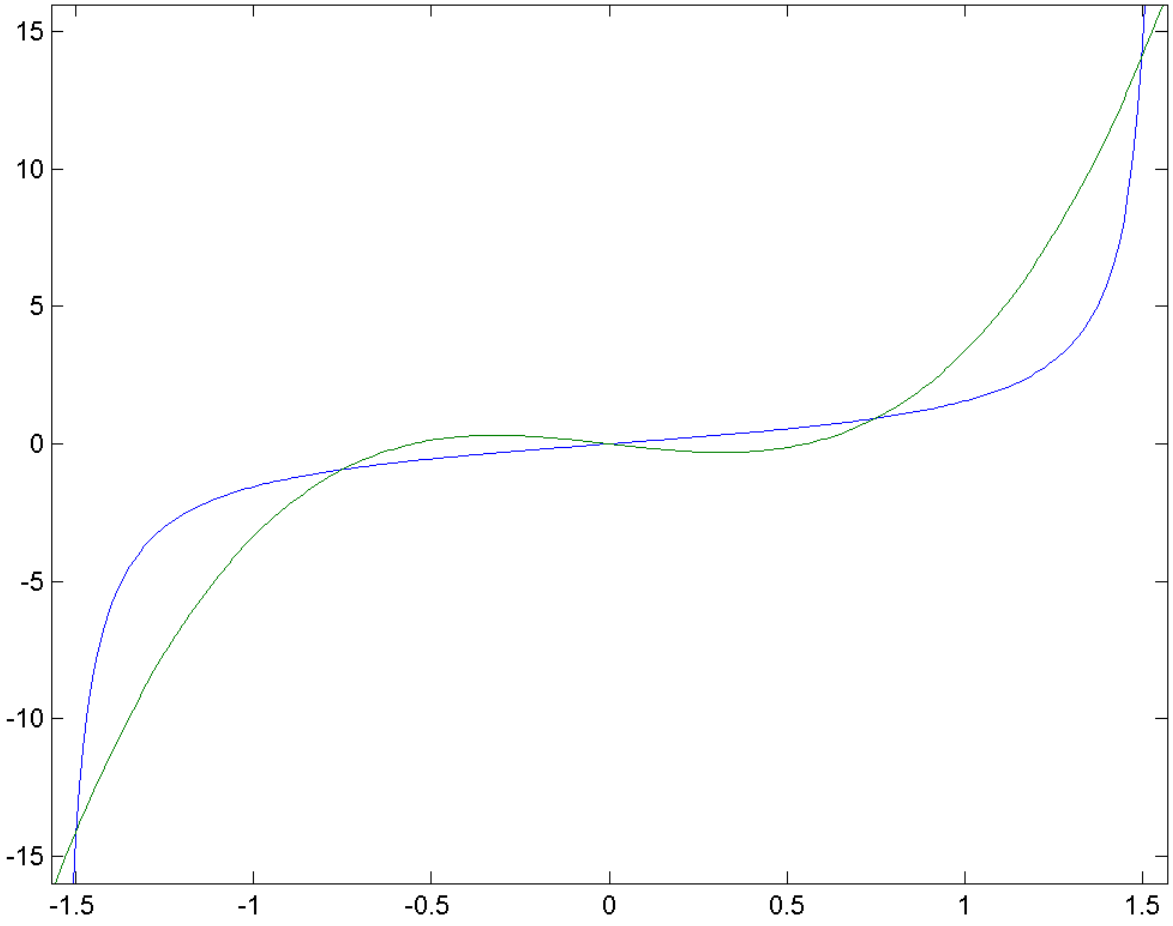
$$\ell_2(x) = \frac{x-x_0}{x_2-x_0}\cdot\frac{x-x_1}{x_2-x_1}\cdot\frac{x-x_3}{x_2-x_3}\cdot\frac{x-x_4}{x_2-x_4} = \frac{3}{243}(2x+3)(4x+3)(4x-3)(2x-3)$$

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

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

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

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

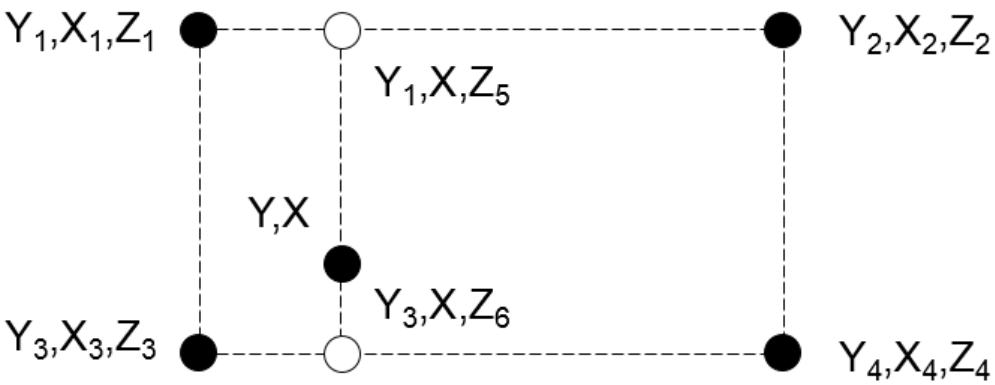


Функция тангенса и интерполяция

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:

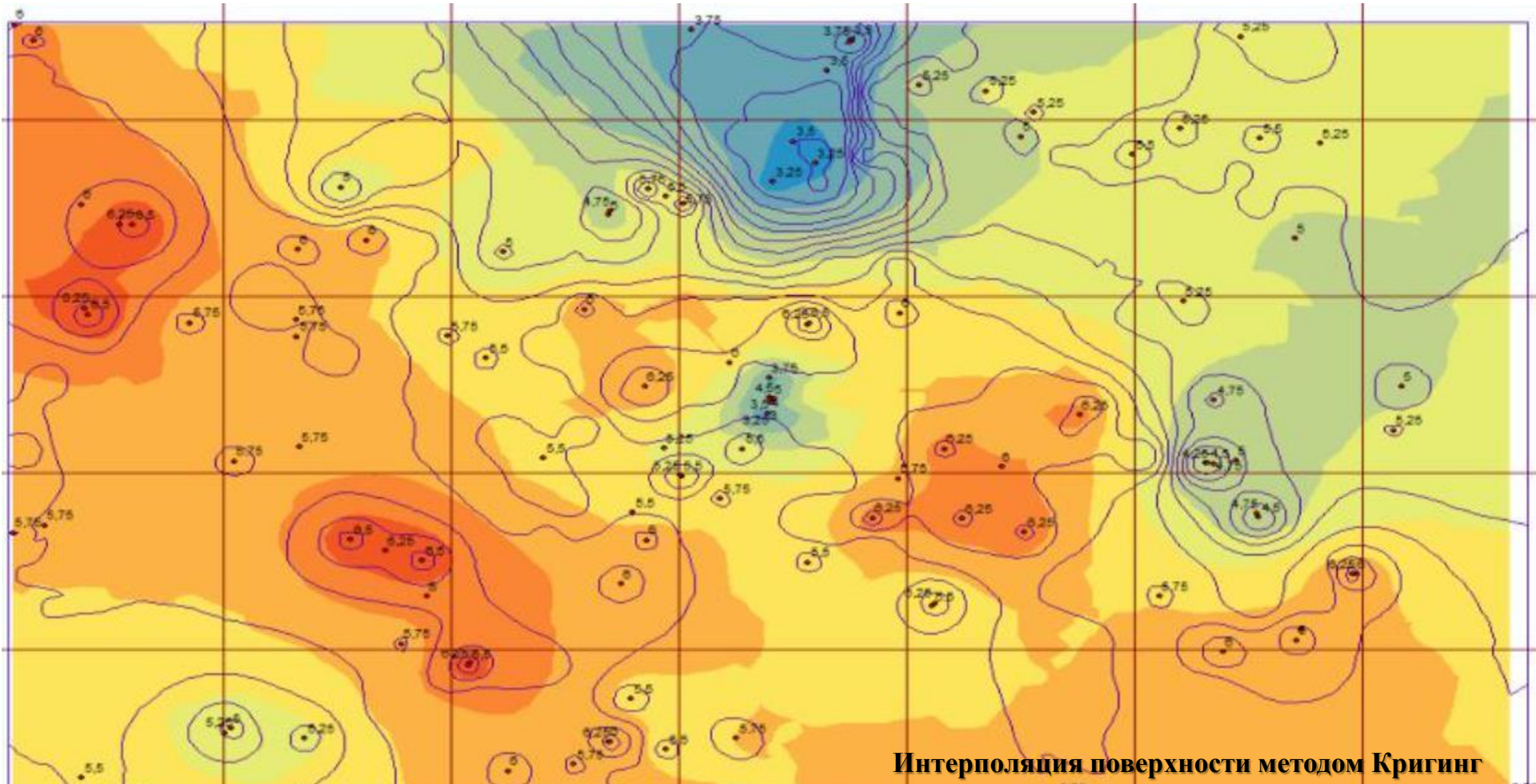
$Z_5 = F(X_1, Z_1, X_2, Z_2, X)$

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

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

ПРИМЕНЕНИЕ ИНТЕРПОЛЯЦИИ ИЗОЛИНИЙ В ГИС

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.



Интерполяция поверхности методом Кригинг

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.

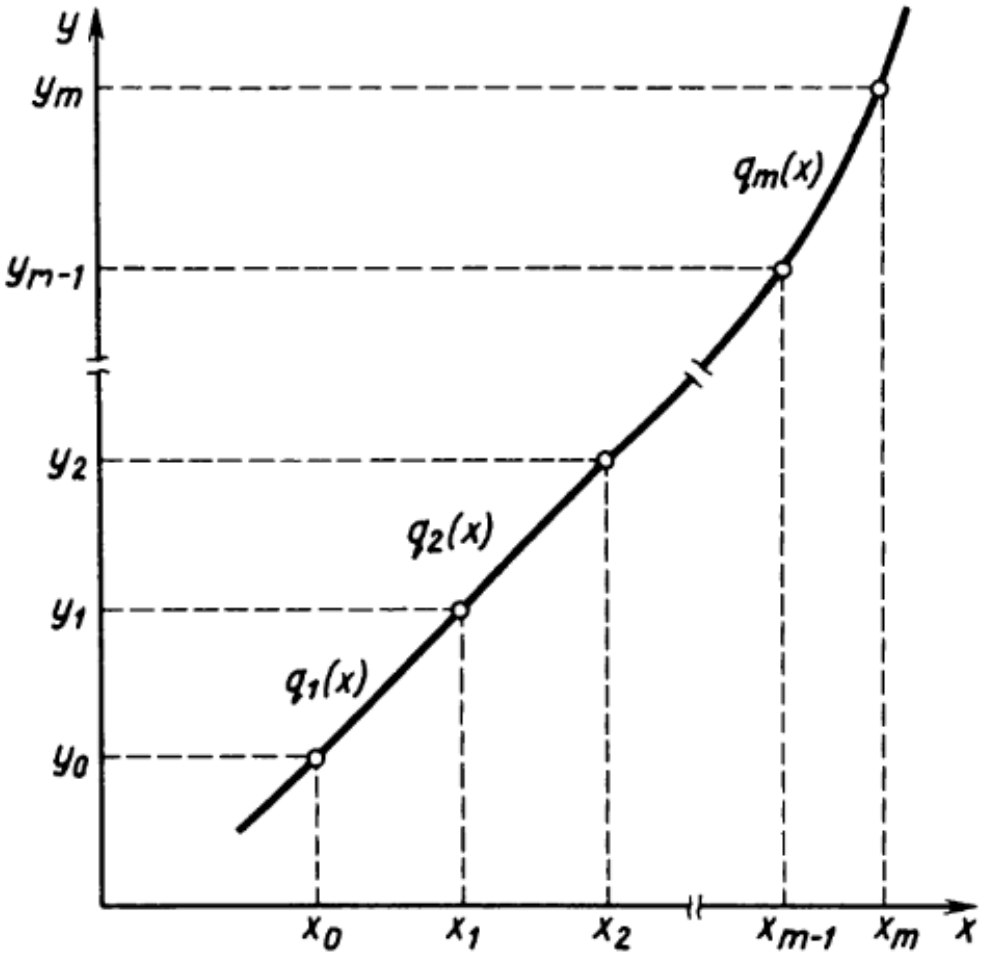
Начиная с $n > 7$ глобальная многочленная интерполяция становиться неустойчивой, поэтому обычную многочленную интерполяцию осуществляют максимум по 3-4 узлам. Интерполяцию по нескольким узлам сеточной функции называют локальной.

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

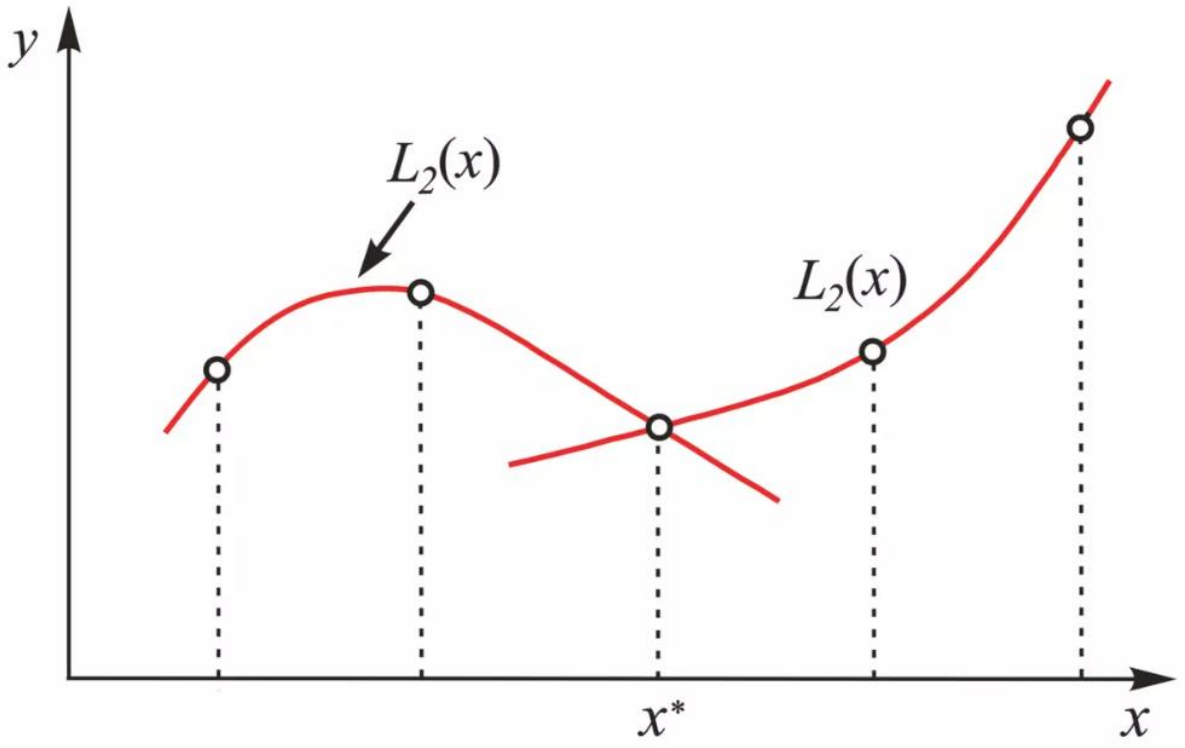
От этих недостатков свободна сплайн-интерполяция, которая требует непрерывности в узлах стыковки локальных многочленов по производным порядка один, два и т.д.

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 $[x_i - 1, x_i]$, $i = 1, 2, \ldots n$, 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;

$i = 1,2,\ldots,$ – 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 х 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.

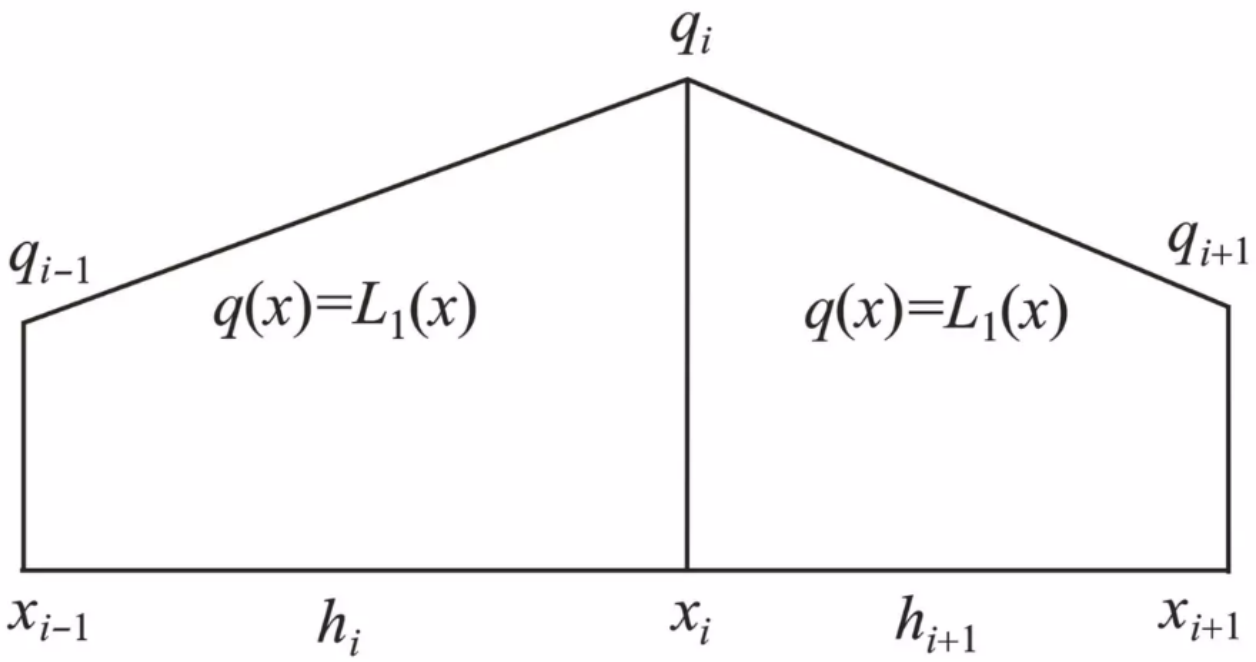
Если за пределами интервала интерполяции экстраполирующая функция представляет собой прямую линию, то это линейная экстраполяция. Следовательно, исходя из условий непрерывности второй производной, запишем еще два равенства:

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

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:

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

$$\times (x_i - x) + (\frac{y_i}{h_i} - q_i\frac{h_i}{6})(x - x_{i-1}), x \in [x_{i-1}, x_i].$$

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 $[x_i, x_{i+1}]$

$$S(x) = q_i \frac{(x_{i+1} - x)^3}{6h_{i+1}} + q_{i+1} \frac{(x - x_i)^3}{6h_{i+1}} + \left(\frac{y_i}{h_{i+1}} - q_i \frac{h_{i+1}}{6} \right) \times$$

$$\times (x_{i+1} - x) + \left(\frac{y_{i+1}}{h_{i+1}} - q_{i+1} \frac{h_{i+1}}{6} \right) (x - x_i), x \in [x_i, x_{i+1}].$$

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:

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

$$i = 2: \quad S_2(x) = \frac{18}{42}(3-x)^3 + (-\frac{30}{42})(x-2)^3 +$$

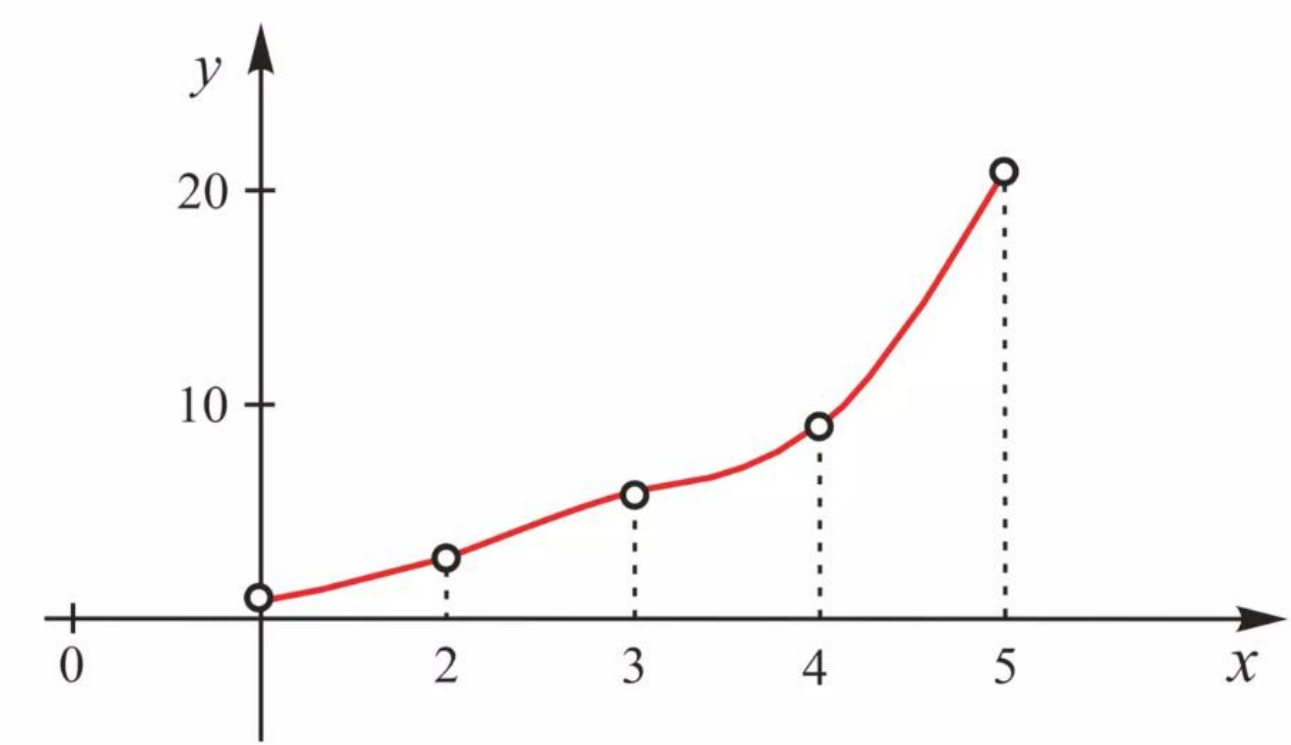
$$+\frac{108}{42}(3-x) + \frac{282}{42}(x-2), \qquad x \in [2;3];$$

$$i = 3: \quad S_3(x) = -\frac{30}{42}(4-x)^3 + \frac{102}{42}(x-3)^2 +$$

$$+\frac{282}{42}4 - x + \frac{276}{42}(x-3), \qquad x \in [3;4];$$

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

Построим полученные сплайны:



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

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

$$S_1'(2-0) = \frac{120}{42}; \qquad S_2'(2+0) = \frac{120}{42};$$

$$S_1''(2-0) = \frac{108}{42}; \qquad S_2''(2+0) = \frac{108}{42}.$$

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).

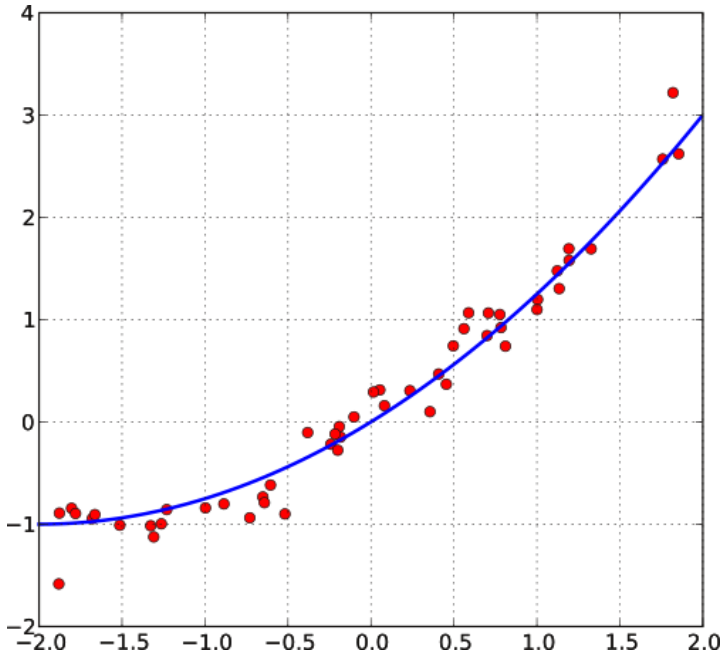
Находим частные производные функции по приведенной формуле $F(a, b)$ по переменным a и b, приравниваем эти производные к нулю.

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

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

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

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



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 **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 σ_A:

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

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

$N(\cdot)$ – 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 Ax = b

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

Метод наименьших квадратов: Найти x для минимизации ||Ax - b||

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

Задачи на собственные значения: Решение Ax = λx

- 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): A = U Σ V* (Au = σv и A*v = σu)

- 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 A

- Матрица 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.

ОБЗОР БИБЛИОТЕК ПЛОТНОЙ ЧИСЛЕННОЙ ЛИНЕЙНОЙ АЛГЕБРЫ

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.

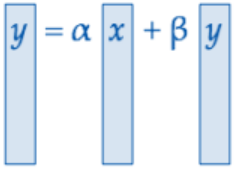
Современное программное обеспечение для многоядерных процессоров и ускорителей.

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:

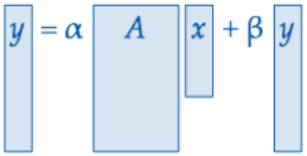
Уровень 1 BLAS



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.

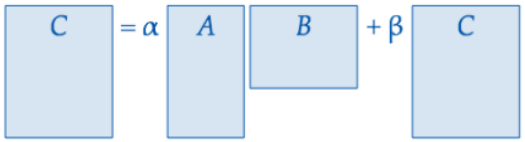
Уровень 2 BLAS



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



Объём данных и вычислений:
- Данные: O(n2) — квадратичная зависимость от размера матриц.
- Операции: O(n3) — 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).

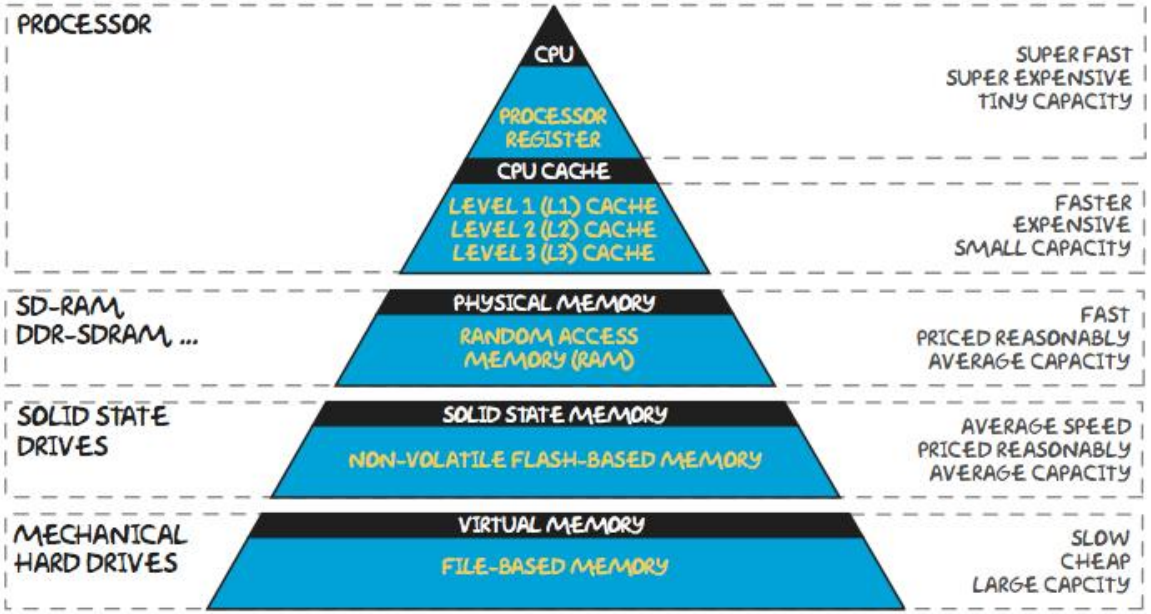
Вычисления (арифметические операции) можно выполнять только над данными, которые находятся на вершине иерархии памяти − в самых быстрых её уровнях, таких как ЦПУ или кэш процессора.

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
Уровень 1 Векторные операции	$3n$	$2n$	$2/3$
Уровень 2 Матрично-векторные операции	n^2	$2n^2$	2
Уровень 3 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.

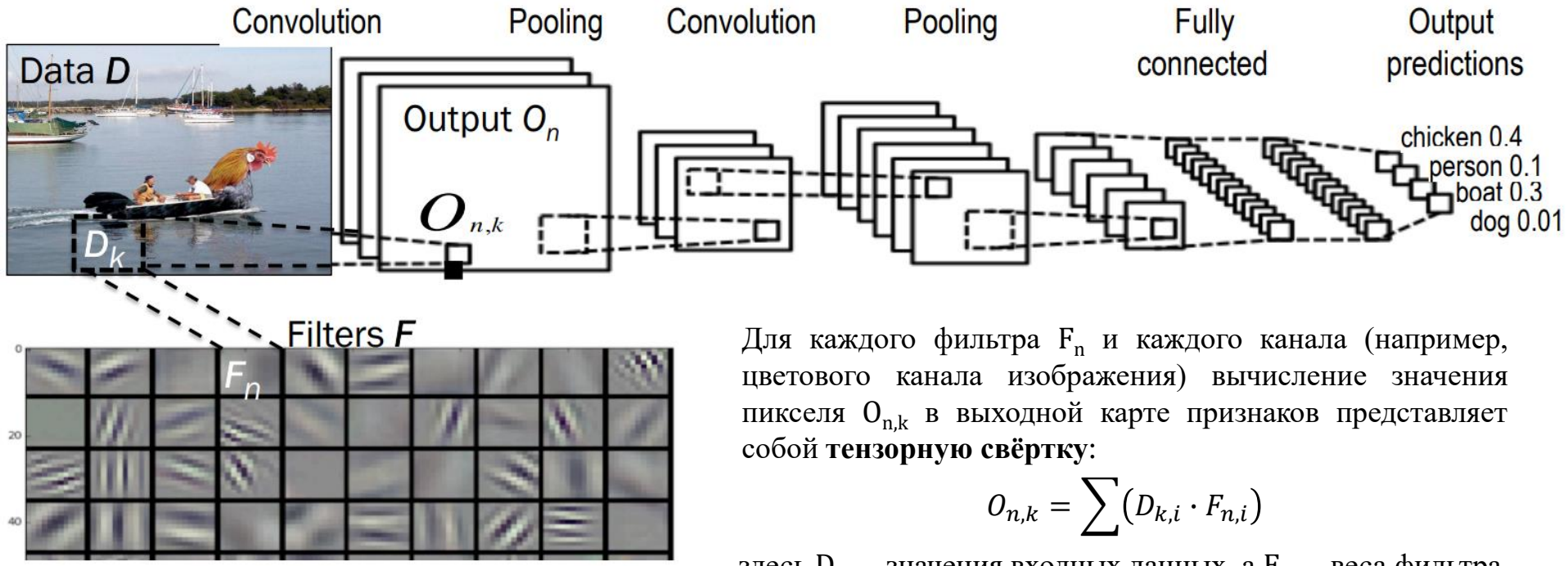


Пример скорости работы алгоритма по перемножению матриц при решении задачи классификации изображений:

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	$O(n^3)$	$\sim 0.2 - 2.0$
Алгоритм Штрассена	$O(n^{2.807\ldots})$	$\sim 0.4 - 1.3$
Алгоритм Coppersmith-Winograd'a	$O(n^{2.37\ldots})$	$\sim 0.3 - 1.0$

MAGMA Batched – поддержка операций с пакетными и тензорными свёртками

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



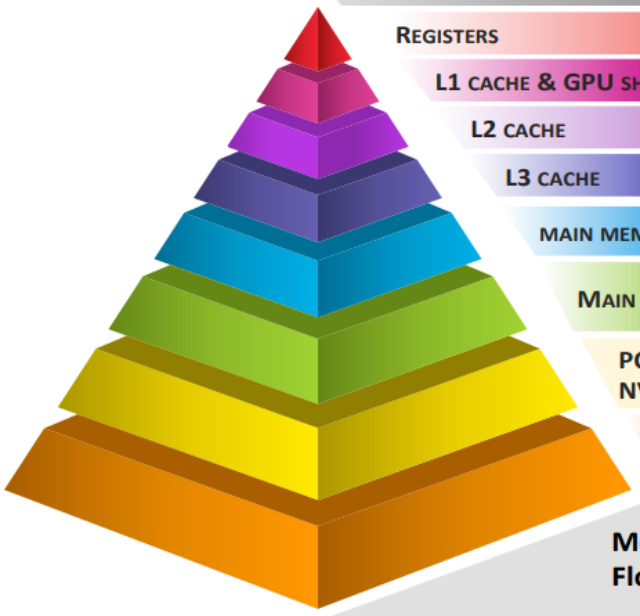
Для каждого фильтра F_n и каждого канала (например, цветового канала изображения) вычисление значения пикселя $O_{n,k}$ в выходной карте признаков представляет собой **тензорную свёртку**:

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

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 ВЕКТОРА МАТРИЦ

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

$$AX = \lambda X$$

называется собственным вектором матрицы A.

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

Последнее равенство равносильно однородной относительно X системе:

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

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

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

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

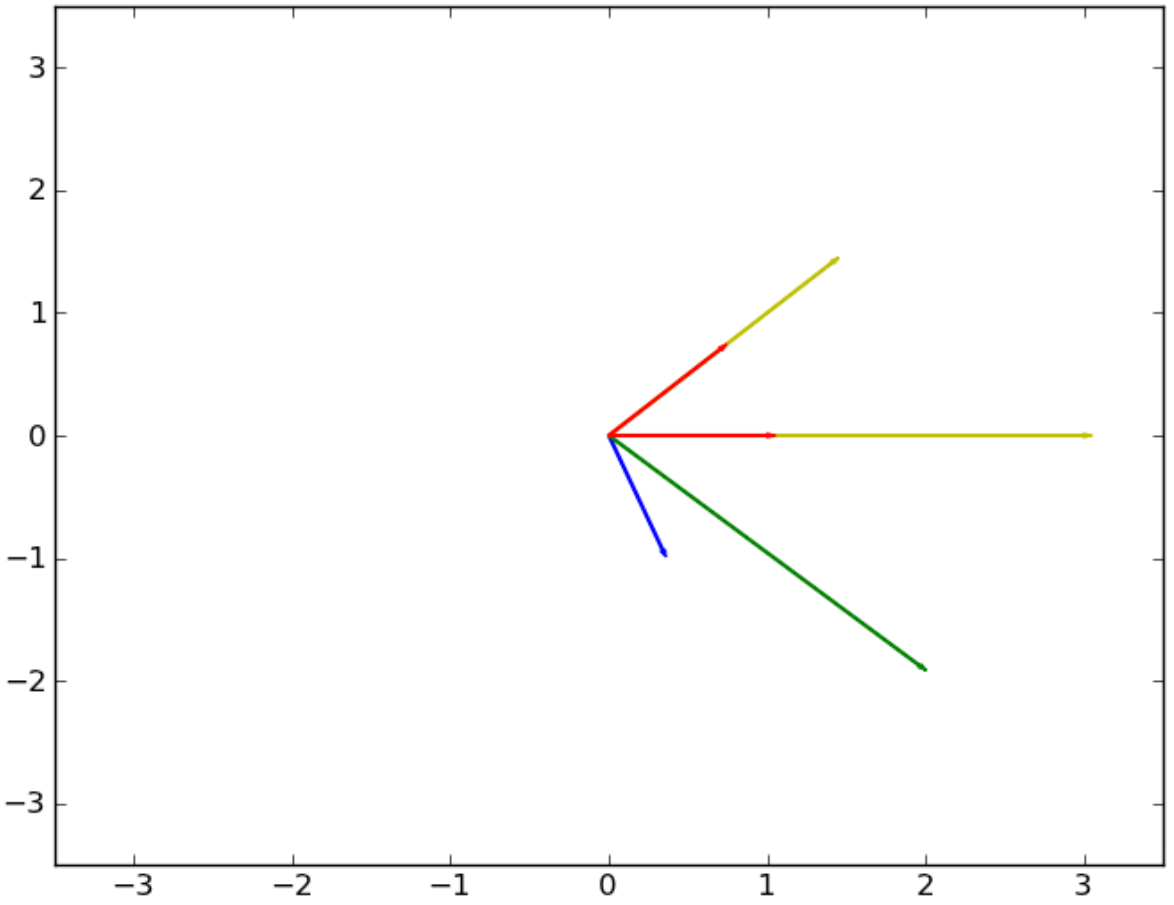
где Pn() — характеристический многочлен n-й степени. Корни $\lambda_1, \lambda_2, ..., \lambda_n$ характеристического уравнения являются собственными (характеристическими) значениями матрицы A, а соответствующие каждому собственному значению λ_i, i = 1, ..., n, ненулевые векторы X_i, удовлетворяющие системе

$$AX^i = \lambda_i X^i \text{ или } (A - \lambda_i E) X^i = 0,\ i = 1, 2, ..., n,$$

являются собственными векторами.

Требуется найти собственные значения и собственные векторы заданной матрицы. Поставленная задача часто именуется второй ***задачей линейной алгебры***.

$$A = \begin{pmatrix} 3 & -1 \\ 0 & 2 \end{pmatrix}$$

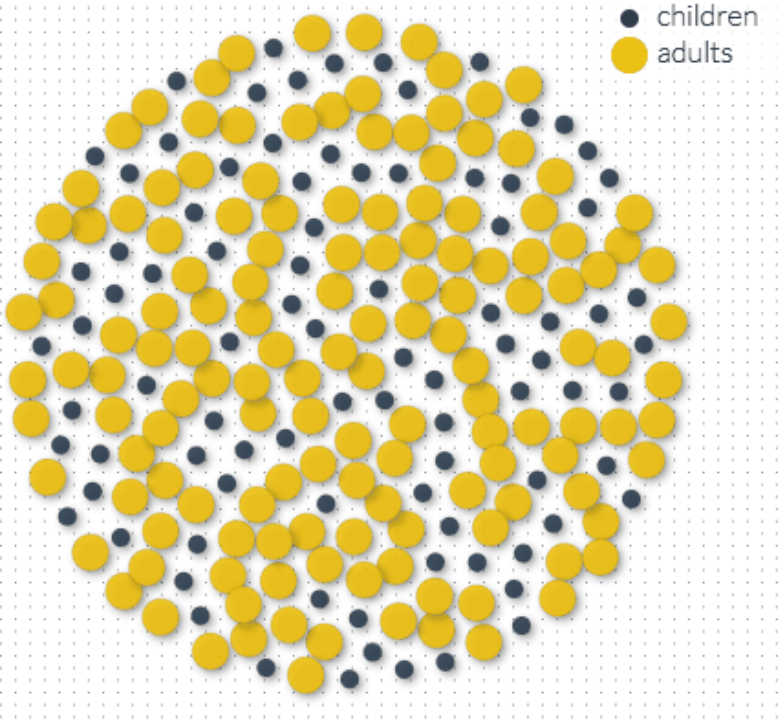


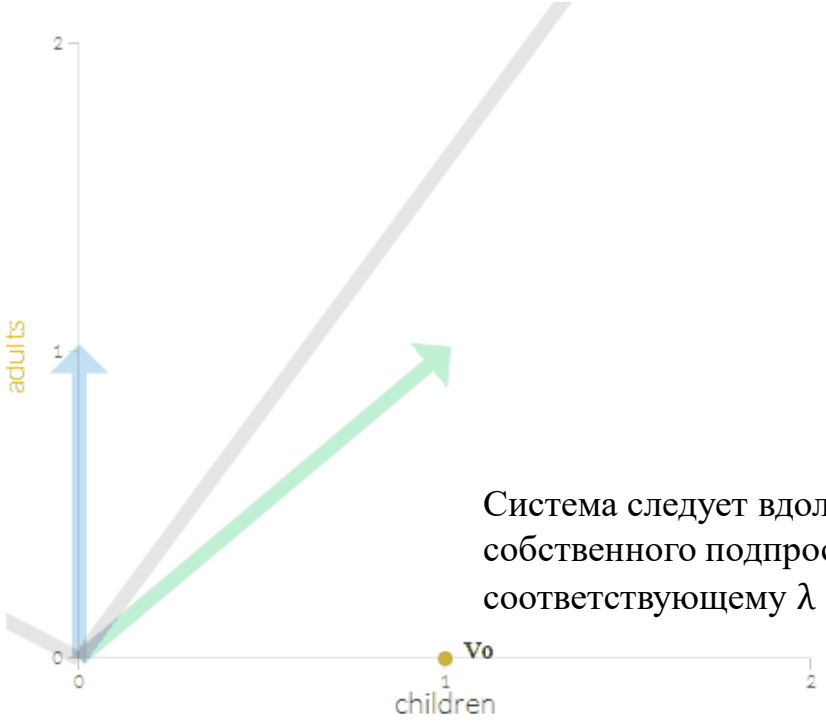
Рассмотрим тривиальную популяционную модель, аналогичную модели Мальтуса:

$$\begin{cases} \text{родители}_{t+1} = \text{родители}_t + \text{дети}_t \\ \text{дети}_{t+1} = \text{родители}_t \end{cases}$$

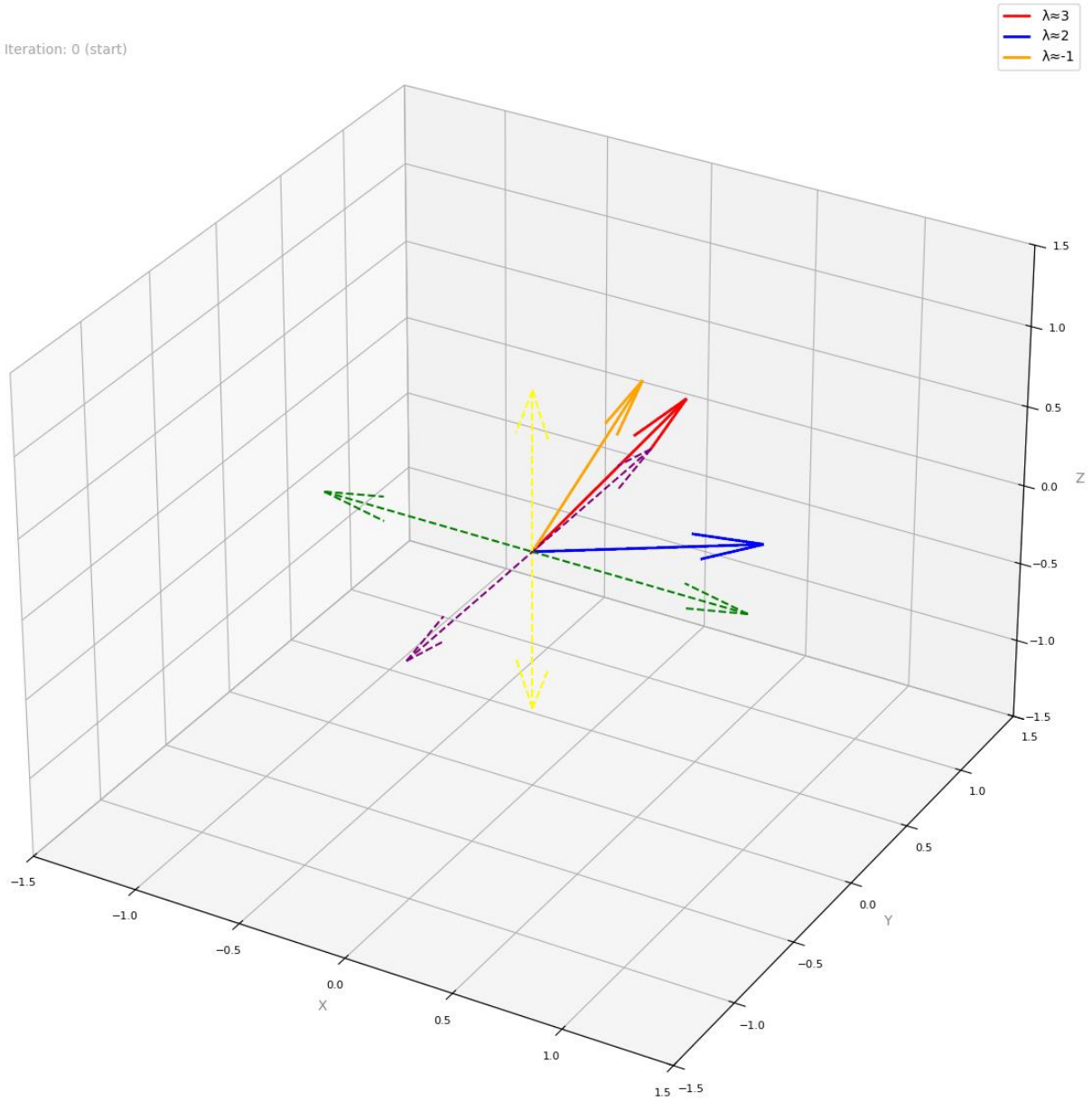
Перепишем:

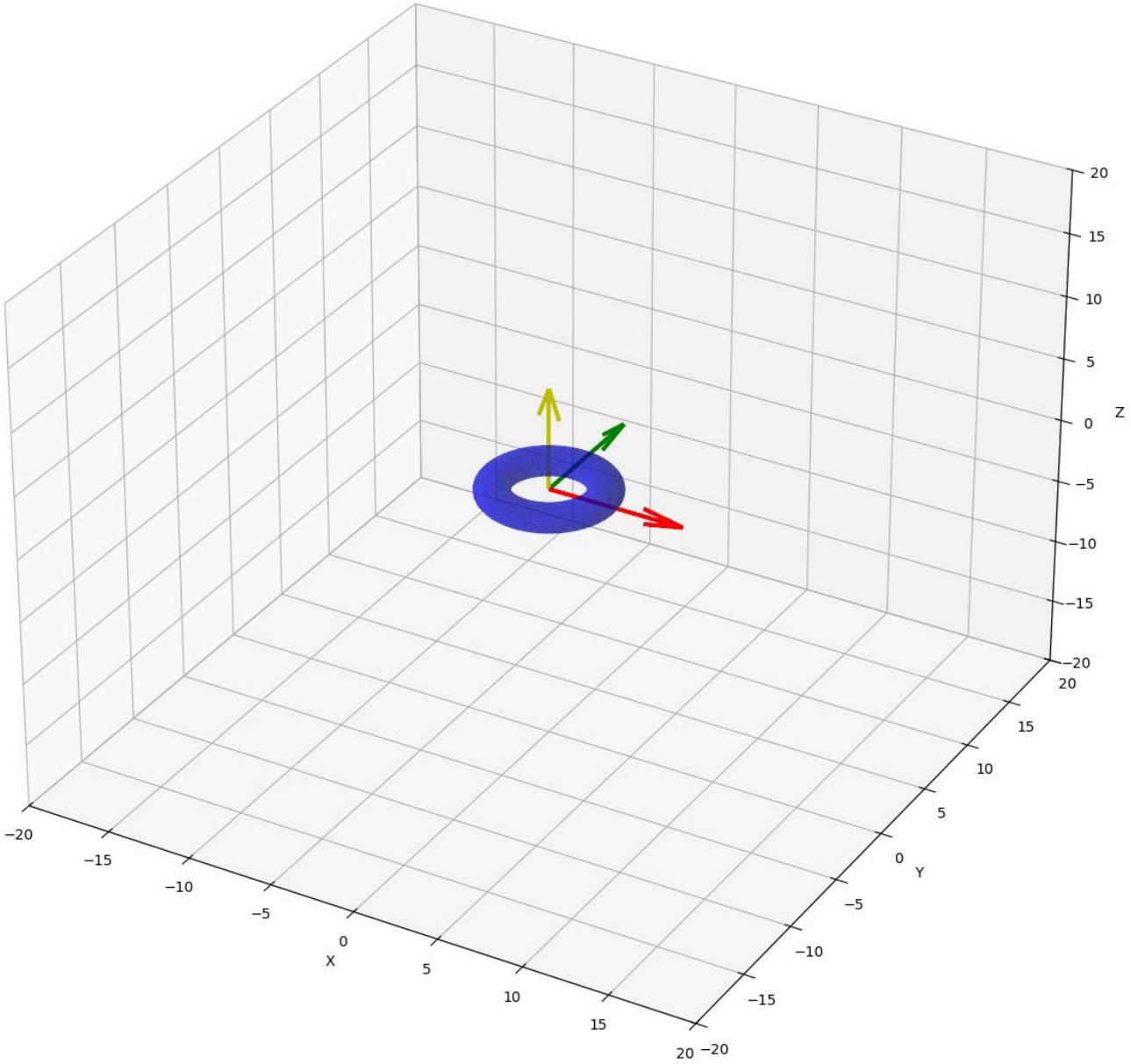
$$\begin{pmatrix} \text{родители}_{t+1} \\ \text{дети}_{t+1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} \text{родители}_t \\ \text{дети}_t \end{pmatrix}$$





Система следует вдоль серой линии – собственного подпространства, соответствующему $\lambda = (1 + \sqrt{5})/2 > 1$


Iteration: 0 (start)
λ≈3
λ≈2
λ≈-1
X
Y
Z


20
15
10
5
0
−5
−10
−15
−20
Z
−20
−15
−10
−5
0
5
10
15
20
X
20
15
10
5
0
−5
−10
−15
−20
Y

Для динамических систем с матрицей A, спектр может много сообщить о поведении системы (например, о её устойчивости).

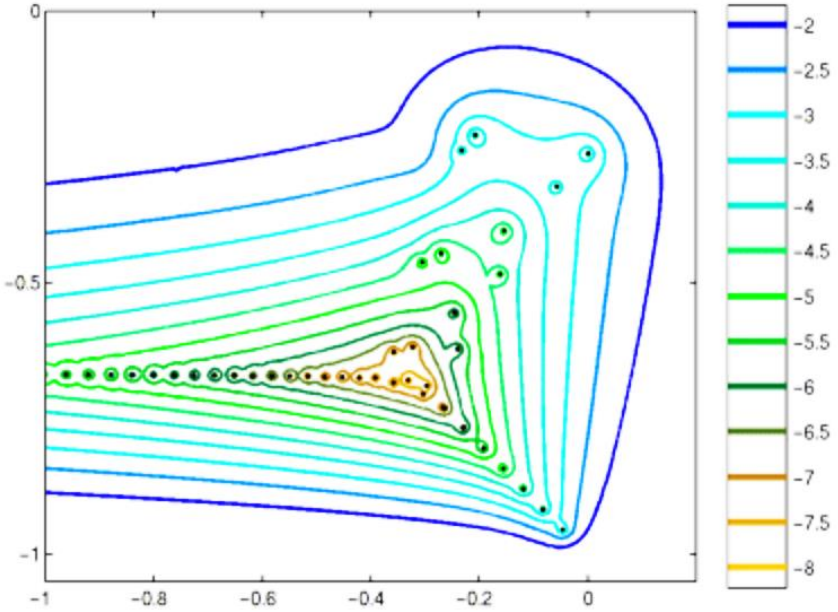
Однако для *не нормальных матриц*, спектр может быть неустойчивым относительно малых возмущений матрицы.

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 Λ 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 μX_i, где μ — 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 λ_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:

$$p_i = \sum_{j \in N(i)} \frac{p_j}{L(j)}$$

$$p_i = \sum_{j \in N(i)} \frac{p_j}{L(j)}$$

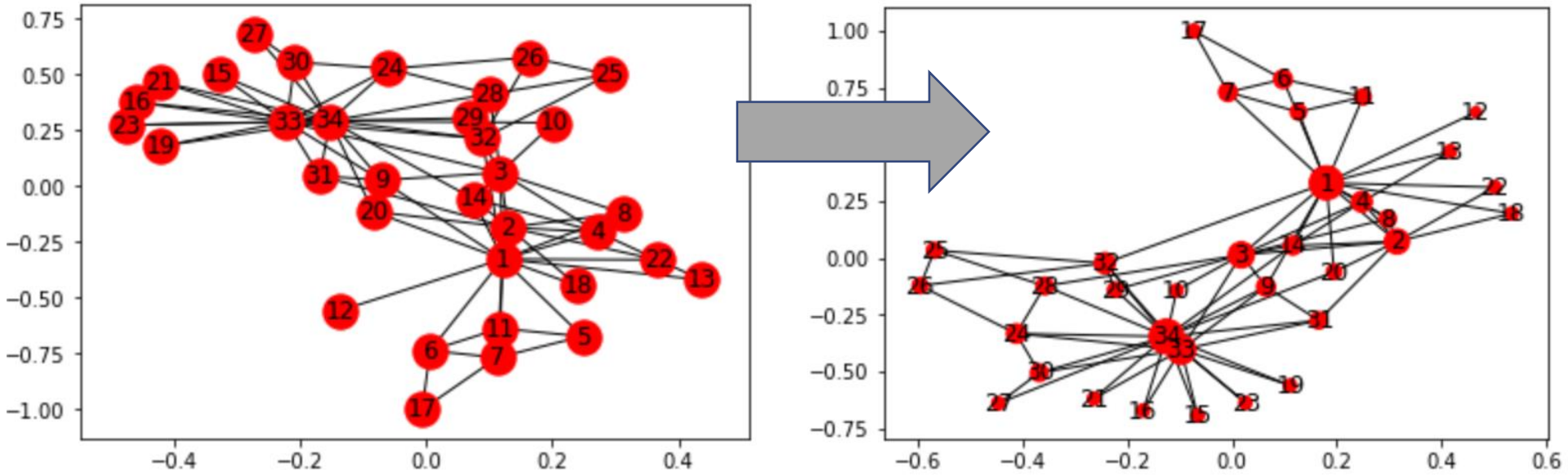
где $L(j)$ – число исходящих ссылок с j-ой страницы, $N(i)$ – число соседей i-ой страницы. Это может быть записано следующим образом:

$$p = Gp, \quad G_{ij} = \frac{1}{L(j)}$$

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 $P_n(\lambda) = 0$, $i = 1,.. n$. Для каждого λ_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 $\lambda_1, \lambda_2, \lambda_3, \ldots \lambda_n$.

3. Для каждого собственного значения составить систему (2.4):

$$(A - \lambda_i E)\, X^i = 0,\ i = 1, 2, \ldots, n,$$

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

$Ax = \lambda x, \quad \|x\|_2 = 1$ 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 $x_{k+1} \to \upsilon_1$, где $A\upsilon_1 = \lambda_1\upsilon_1$ и λ_1 максимальное по модулю собственное значение, и υ_1 – соответствующий собственный вектор.

На $(k+1)$-ой итерации приближение для λ_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 $\lambda_1 > \lambda_2 \geq \cdots \geq \lambda_n$.

Это означает, что сходимость может быть сколь угодно медленной при близких значениях у λ_1 и λ_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 $\upsilon_1,\ldots,\upsilon_n$ такой что $A\,\upsilon_i = \lambda_i\,\upsilon_i$.
- Разложим x_0 в этом базисе с коэффициентами c_i:

$$x_0 = c_1\,\upsilon_1 + \cdots + c_n\,\upsilon_n.$$

- Поскольку υ_i – собственные векторы, выполнены следующие равенства

- Поскольку υ_i – собственные векторы, выполнены следующие равенства

$$x_1 = \frac{Ax_0}{\|Ax_0\|} = \frac{c_1\lambda_1 v_1 + \cdots + c_n\lambda_n v_n}{\|c_1\lambda_1 v_1 + \cdots + c_n\lambda_n v_n\|}$$

$$\vdots$$

$$x_k = \frac{Ax_{k-1}}{\|Ax_{k-1}\|} = \frac{c_1\lambda_1^k v_1 + \cdots + c_n\lambda_n^k v_n}{\|c_1\lambda_1^k v_1 + \cdots + c_n\lambda_n^k v_n\|}$$

- 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 υ_1 при $\left|\frac{c_1}{|c_1|}\left(\frac{\lambda_1}{|\lambda_1|}\right)^k\right| = 1$ и $\left(\frac{\lambda_2}{\lambda_1}\right)^k \to 0$ 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) (напрмиер, 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 $\{x_0, A\ x_0, ..., A^k\ x_0\}$ и имеет вид $\mu A^k x_0$, где μ – 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:

$$\rho(A) = |\lambda_1(A)| > |\lambda_2(A)| \geq \ldots \geq |\lambda_n(A)|$$

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 $X^{1(1)} = AX^{1(0)}$, $\lambda_1^{(1)} = \dfrac{x_i^{1(1)}}{x_i^{1(0)}}$, где i — любой номер $1 \le i \le n$, и положить $k = 1$.

3. Вычислить $X^{1(k+1)} = AX^{1(k)}$.

4. Найти $\lambda_1^{(k+1)} = \dfrac{x_i^{1(k+1)}}{x_i^{1(k)}}$, где $x_i^{1(k+1)}$, $x_i^{1(k)}$ — соответствующие координаты векторов $X^{1(k+1)}$ и $X^{1(k)}$. При этом может быть использована любая координата с номером $i, i = \overline{1, n}$.

5. Если $\Delta = \left|\lambda_1^{(k+1)} - \lambda_1^{(k)}\right| \le \varepsilon$, процесс завершить и положить $\lambda_1 \cong \lambda_1^{(k+1)}$. Если $\Delta > \varepsilon$, положить $k = k + 1$ и перейти к п. 3.

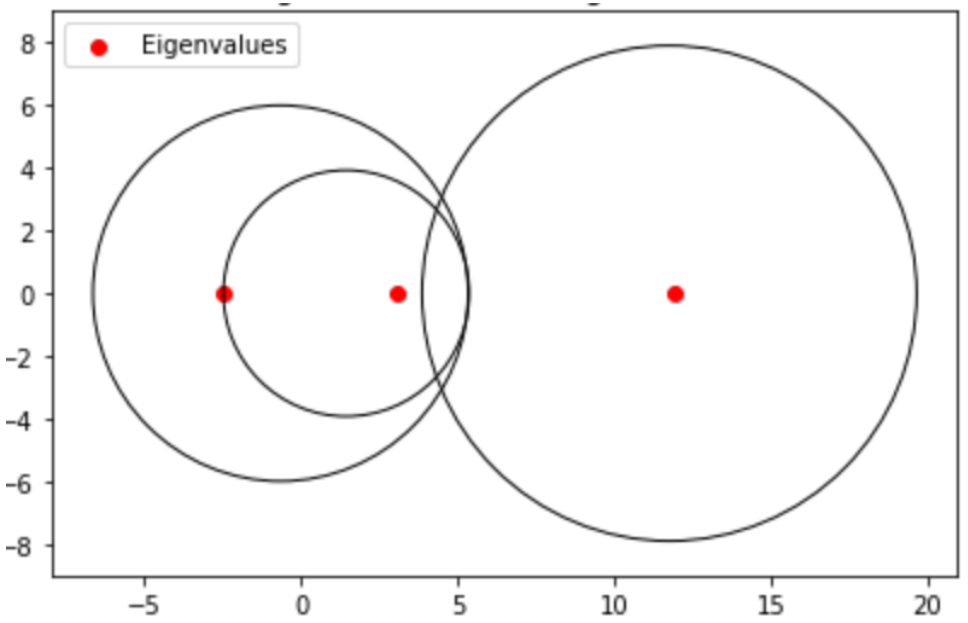
Есть теорема, которая часто помогает локализовать собственные значения.

Она называется теоремой Гершгорина.

- Теорема утверждает, что все собственные значения λ_i, $i = 1, 2, ..., n$ 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:

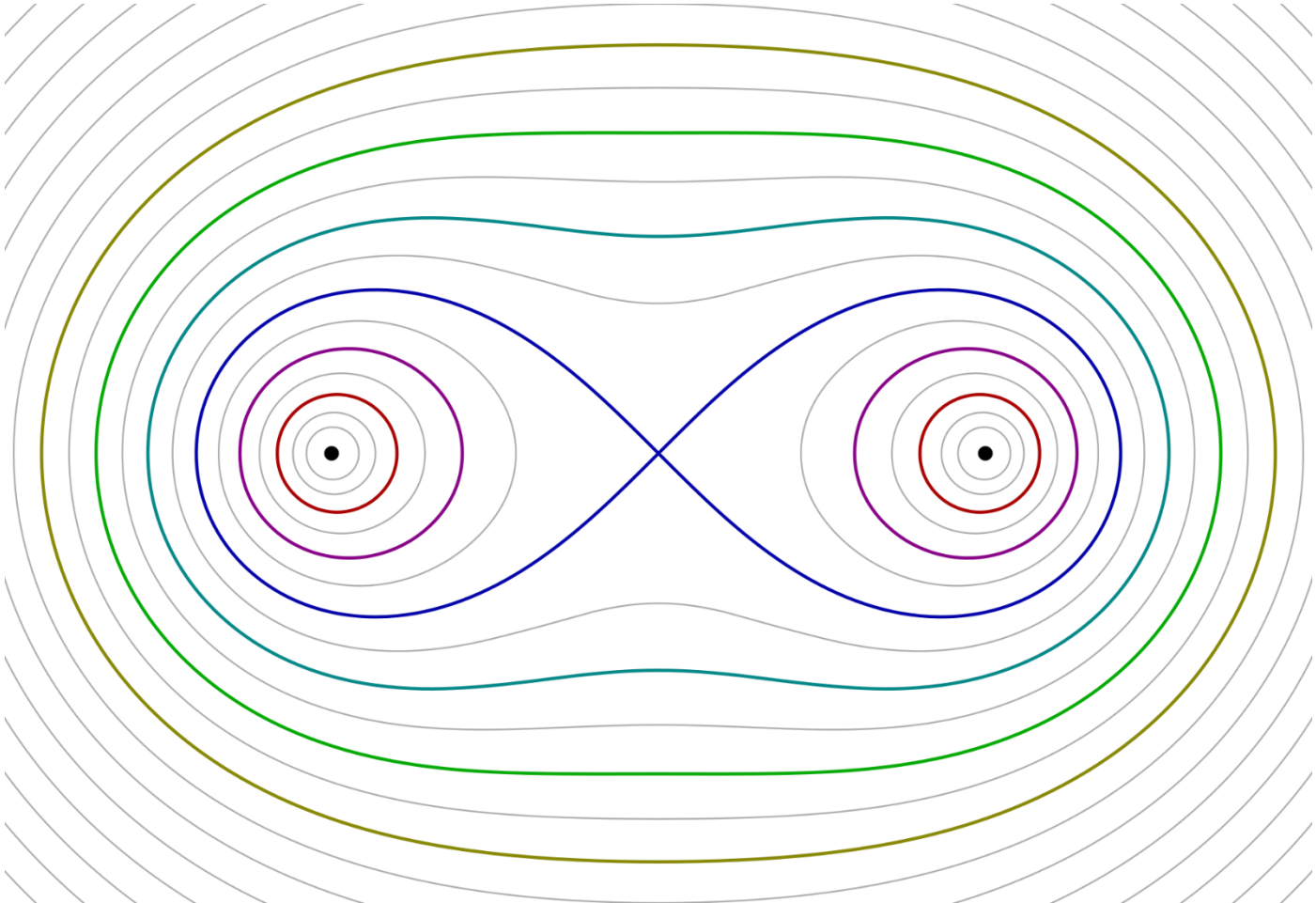
если любое из собственных чисел лежит вне всех кругов, то матрица $(A - \lambda I)$ обладает свойством строгого диагонального преобладания

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 λ_i:

$$\mathrm{tr}A = \sum_{i=1}^{n} a_{ii} = \sum_{i=1}^{n} \lambda_i(A) = \mathrm{tr}A^{(i)}.$$

При реализации метода вращений преобразование подобия применяется к исходной матрице A многократно:

$$A^{(k+1)} = (H^{(k)})^{-1} A^{(k)} H^{(k)} = (H^{(k)})^{T} A^{(k)} H^{(k)}, \quad k = 0, 1, \ldots$$

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

$$H^{(k)} = \begin{pmatrix}
1 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\
\vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & \cdots & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\
0 & \cdots & 0 & \cos\varphi^{(k)} & 0 & \cdots & 0 & -\sin\varphi^{(k)} & 0 & \cdots & 0 \\
0 & \cdots & 0 & 0 & 1 & \cdots & 0 & 0 & 0 & \cdots & 0 \\
\vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & \cdots & 0 & 0 & 0 & \cdots & 1 & 0 & 0 & \cdots & 0 \\
0 & \cdots & 0 & \sin\varphi^{(k)} & 0 & \cdots & 0 & \cos\varphi^{(k)} & 0 & \cdots & 0 \\
0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & \cdots & 0 \\
\vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 1
\end{pmatrix}$$

← i-я строка

← j-я строка

↑ i-й столбец

↑ j-й столбец

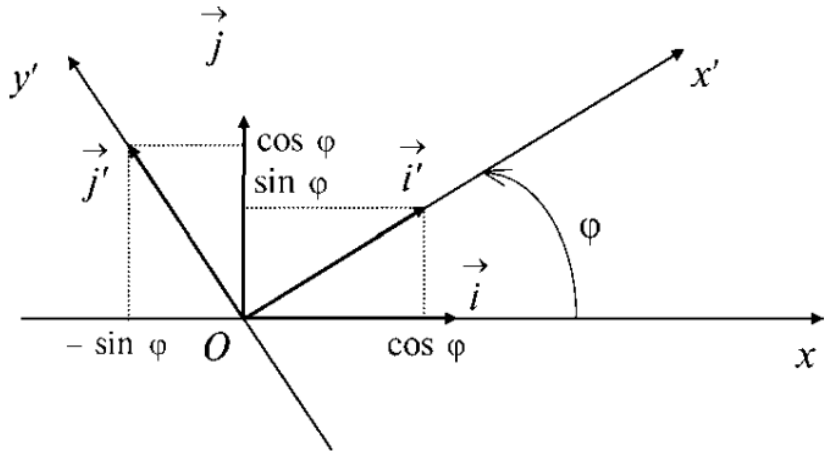
В данной ортогональной матрице элементы на главной диагонали единичные, кроме $h_{ii}^{(k)} = \cos\varphi^{(k)}$ и $h_{jj}^{(k)} = \cos\varphi^{(k)}$, а $h_{ij}^{(k)} = -\sin\varphi^{(k)}$, $h_{ji}^{(k)} = \sin\varphi^{(k)}$ (h_{ij} — элементы матрицы H).

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 $|2\varphi^{(k)}| \le \frac{\pi}{2}$, $i < j$ (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 $k \to \infty$ получается монотонно убывающая ограниченная снизу нулем последовательность $\sigma(A^{(1)}) > \sigma(A^{(2)}) > \ldots > \sigma(A^{(k)})$. Поэтому $\sigma(A^{(k)}) \to 0$ при $k \to \infty$. Это и означает сходимость метода. При этом



$$A^{(k)} \to \Lambda = \begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{pmatrix}.$$

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 $\left|a_{ij}^{(k)}\right| \le \varepsilon$ для всех $i \ne j$, 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:

$$\nu_k = H^{(0)}H^{(1)}H^{(2)} \dots H^{(k-1)} = (X^1, X^2, X^3, \dots, X^n).$$

Если $\left|a_{ij}^{(k)}\right| > \varepsilon$, 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:

$$c_{ij} = \frac{\sum_{p=1}^{n} x_k^i x_k^j}{n} - \mu_i \mu_j, \forall i,j \in \{1 \dots m\},$$

$$\mu_i - \text{среднее значение признака } i, i \in \{1 \dots m\}$$

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:

$$c_{ij} = \frac{\sum_{p=1}^{n} x_k^i x_k^j}{n} - \mu_i \mu_j, \forall i,j \in \{1 \dots m\},$$

$$\mu_i - \text{среднее значение признака } i, i \in \{1 \dots m\}$$

В матричном виде:

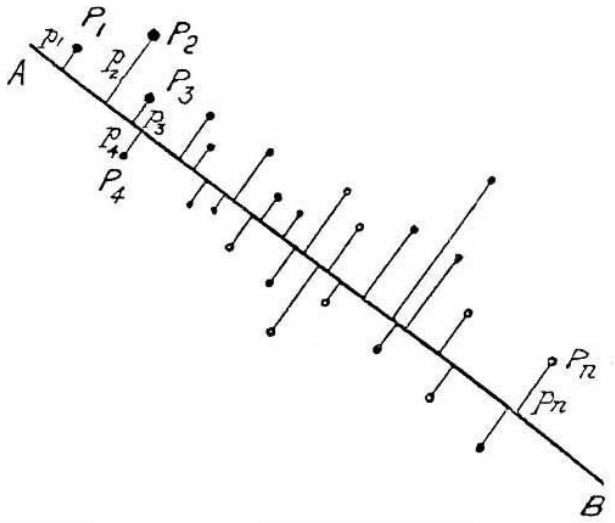
$$C = \frac{X^T X}{n} - \mu^T \mu, \mu = (\mu_1 \dots \mu_m)$$

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 ν такие, что $\nu^T C \nu \to \max$, т.е. проекция данных на которые позволит сохранить наибольшую вариацию

Собственные векторы:

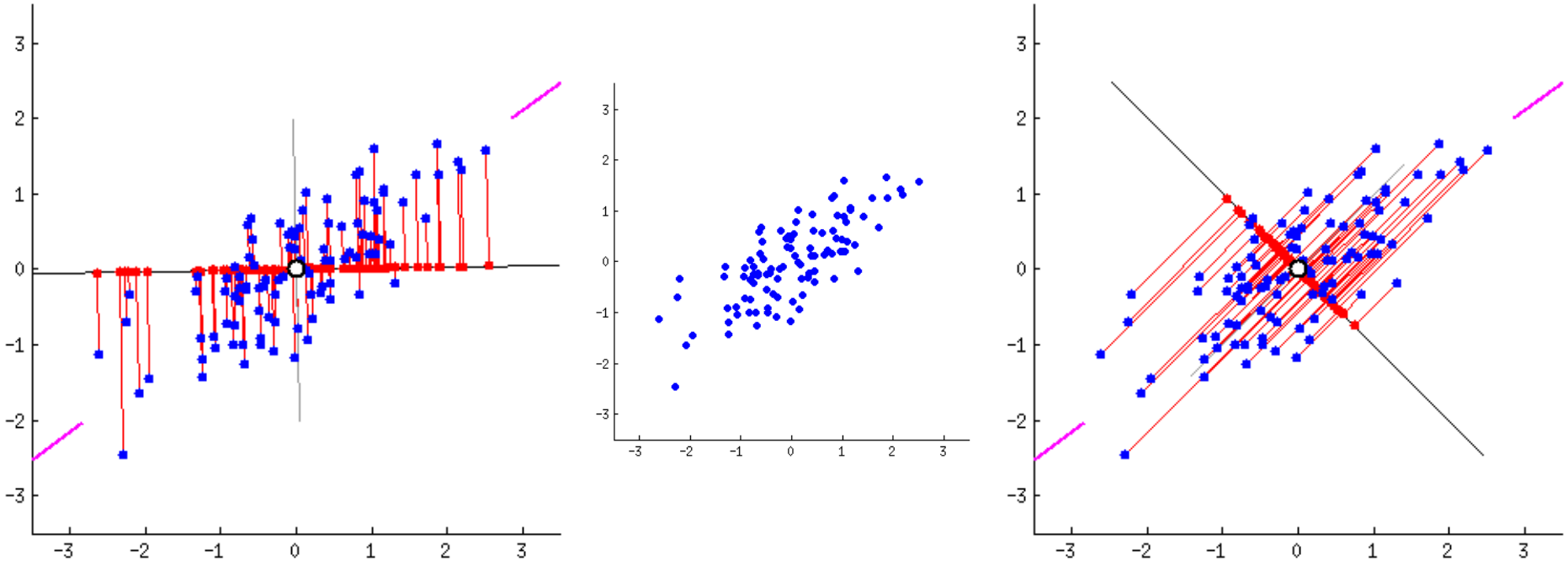
- Это направления наибольшей изменчивости данных, называемые главными компонентами.
- Каждый собственный вектор — это новая "ось", вдоль которой данные растянуты больше всего.

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. PCA 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:

$$C = V\Lambda V^T$$

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \lambda_m \end{bmatrix},$$

λ – 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.

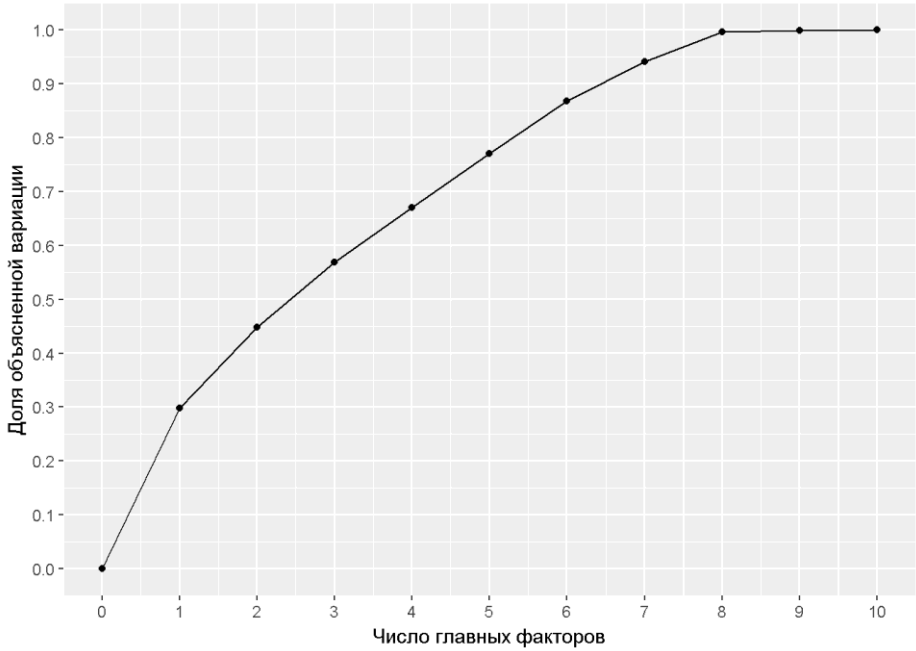
Главные компоненты:

$$U = X \cdot \left[v^1, v^2, \ldots, v^k\right]^T, k < m$$

$$X^* = X \cdot V^T$$

Доля объясненной вариации:

$$\frac{\sum_{i=1}^{k} \lambda_i}{Var(X)}$$



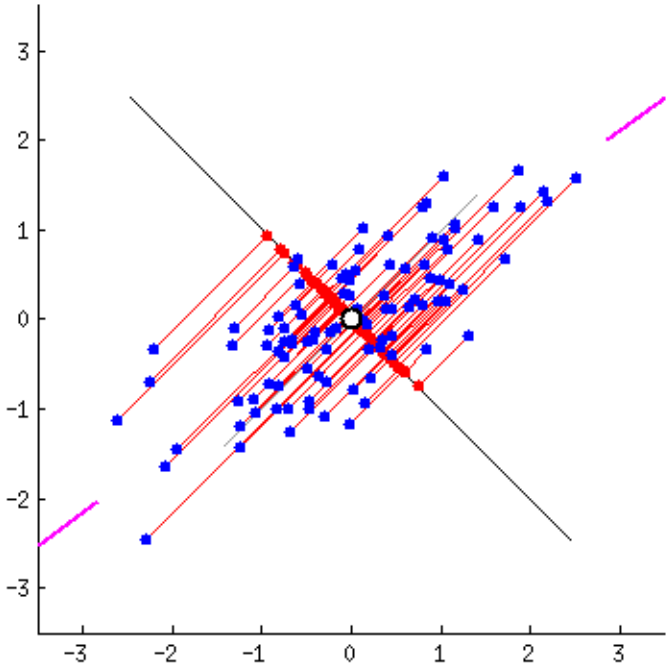
Пример ковариационной матрицы:

$$\begin{pmatrix} 1.07 & 0.63 \\ 0.63 & 0.64 \end{pmatrix}$$

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 спроецировать 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 Штрассена (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 **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 спроецировать 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 PCA, 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 PCA. 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 (PCA).
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:
#
#
#
#
```

$$\text{cov\_matrix} = \frac{1}{m-1} \cdot (X_{\text{centered}}^T \cdot X_{\text{centered}})$$

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 projection ($m \times k$) и V ($n \times k$). При малом k это значительно экономит память.

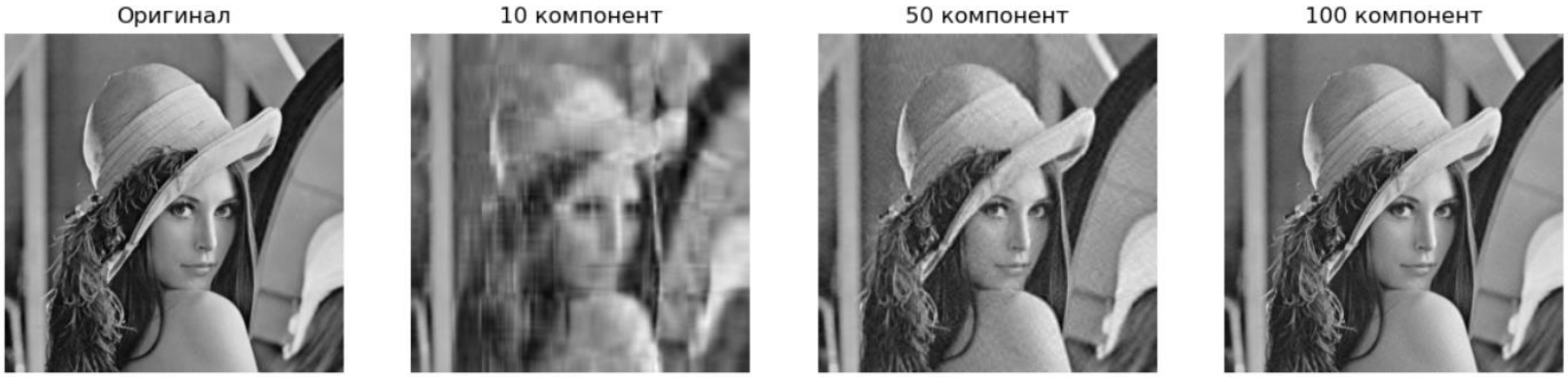
Такой подход широко применяется в сжатии изображений, анализе данных и машинном обучении.

Домашняя работа #4

…продолжение кода на 289-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
```



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 PCA 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 – λ_i, λ_2, λ_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 $U_1=[\upsilon_1, \upsilon_2,..., \upsilon_n]$, где $\upsilon_2,..., \upsilon_n$ любые векторы ортогональные υ_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 Λ - 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.

Цель: Нахождение собственных значений и векторов матрицы

QR-алгоритм — это итеративный процесс, который использует QR-разложение для постепенного приведения матрицы к треугольной форме.

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

$$A_k = Q_k^* A Q_k = Q_k^* Q_{k+1} R_{k+1} = \widehat{Q}_k R_{k+1}$$

тогда аппроксимация для A_{k+1} $\mathrm{A_{k+1}}$ имеет вид

$$A_{k+1} = Q_{k+1}^* A Q_{k+1} = (Q_{k+1}^* A = R_{k+1}Q_k^*) = R_{k+1}\widehat{Q}_k.$$

Итак, мы получили стандартную форму записи QR алгоритма.

Финальные формулы обычно записывают в **QRRQ**-форме:

1. Инициализируем A_0=A.
2. Вычислим QR разложение матрицы A_k: A_k=$Q_k R_k$.
3. Обновим аппроксимацию A_{k+1}=$R_k Q_k$.

С каждой итерацией матрица A_k всё больше приближается к форме, где её диагональные элементы — собственные значения.

Продолжаем итерации пока A_k не станет достаточно треугольной (например, норма подматрицы под главной диагональю не станет достаточно мала).

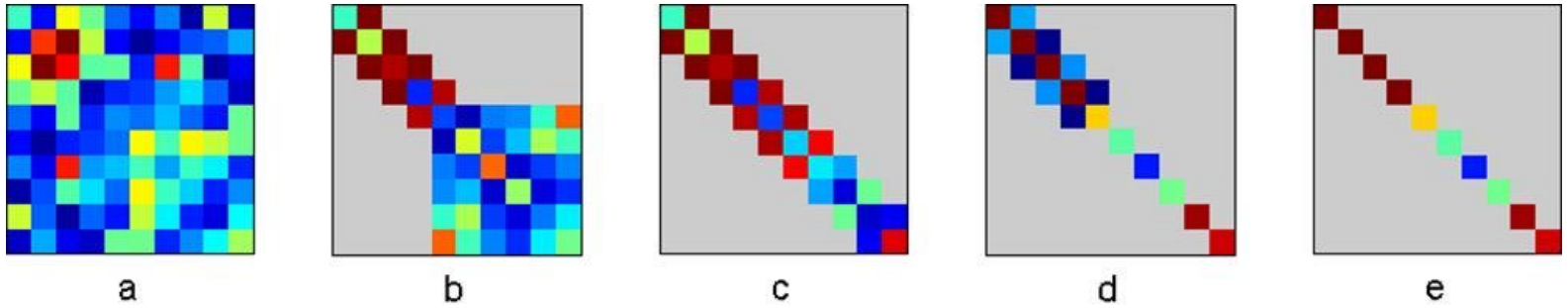
Аналогия: на каждом шаге мы "поворачиваем" матрицу (с помощью Q_k— это "поворот", который выстраивает всё в нужном направлении) и "выравниваем" (с помощью R_k), пока не получим аккуратную диагональную форму.

- 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).

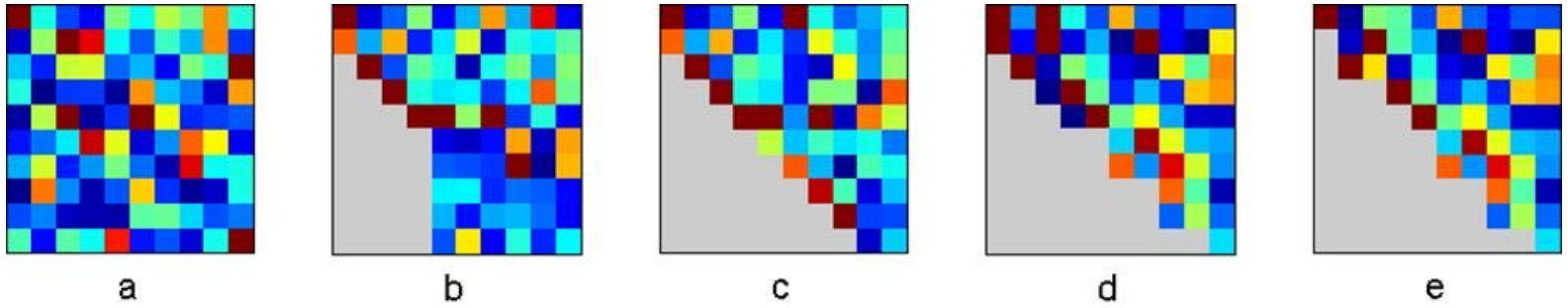
Итоговая сложность - O(n^4)?

- 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 $O(n)$.



Для симметричных матриц: матрица A_k становится диагональной, и её диагональные элементы — это собственные значения.



Для несимметричных матриц: матрица A_k становится верхнетреугольной (форма Шура), где диагональные элементы — собственные значения.

Возможно ускорить QR алгоритм, используя сдвиги, поскольку матрица $A_k - \lambda I$ имеет те же векторы Шура (столбцы матрицы U).

Одна итерация QR алгоритма имеет следующий вид:

$$A_k = Q_k R_k, \qquad A_{k+1} = R_k Q_k.$$

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

$$A_k = Q_k R_k, \qquad A_{k+1} = R_k Q_k$$

в случае $A_k = A_k^*$.

Такая матрица определяется $O(n)$ параметрами.

Вычисление QR разложения более сложное, но возможно вычислить A_{k+1} $\mathrm{A}_{\mathrm{k+1}}$ напрямую без вычисления Q_k.

Это называется неявный QR-шаг.

Зазоры:

- Зазор – разница между собственными значениями ($|\lambda_i - \lambda_j|$).
- Большие зазоры ускоряют сходимость, так как собственные значения быстрее отделяются.
- Маленькие зазоры замедляют сходимость, так как разделение значений происходит медленнее.

Аналогия: Сходимость – это игра в "найди отличия". Собственные значения — это точки на линии, и если они далеко друг от друга (большие зазоры), алгоритм быстро их "замечает" и отделяет. Но если они близко (маленькие зазоры), ему нужно больше времени, чтобы "разглядеть" различия.

Смещения в QR-алгоритме

Зачем нужны смещения?

- Ускоряют сходимость, особенно когда собственные значения близки друг к другу.
- Смещение σ_k выбирается так, чтобы приблизить матрицу к собственному значению.

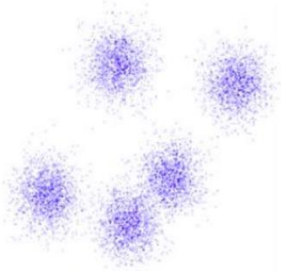
Типы смещений:

- Одинарное смещение: для действительных собственных значений.
- Двойное смещение: для комплексных собственных значений.

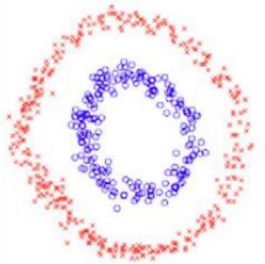
Зазор (разрыв) в спектре собственных значений может также дать понять, на сколько групп можно разделить данные.

- Compactness, e.g., k-means, mixture models
- Connectivity, e.g., spectral clustering

Представьте, что у вас есть куча точек – данные о кинофильмах. Вы хотите сгруппировать их по похожести. Спектральная кластеризация смотрит на эти точки как на вершины графа, а связи между ними – как на рёбра. Чем больше два фильма схожи, тем толще ребро между ними. Потом мы строим специальную матрицу, называемую лапласианом, и смотрим на её собственные значения.



Compactness

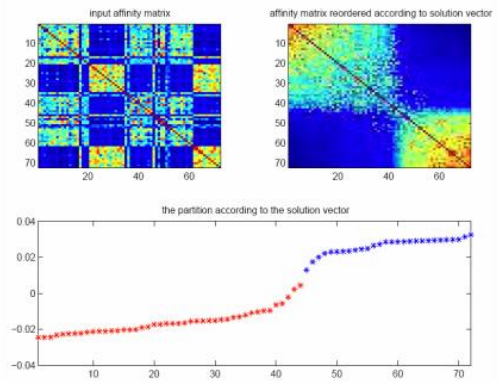


Connectivity

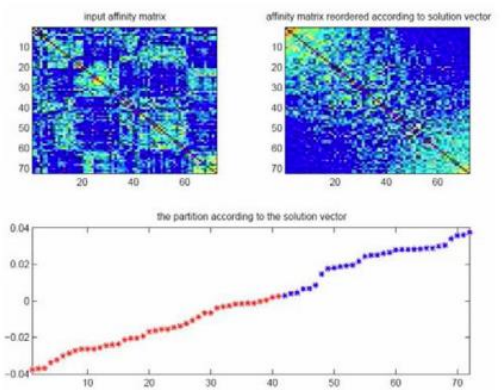
Собственные значения — это числа, которые описывают, как матрица "работает" (как данные связаны друг с другом). Если представить, что ваш граф – натянутая струна на гитаре, то собственные значения — это частоты вибраций струны. В кластеризации они подсказывают, сколько "основных вибраций" (или кластеров) есть в ваших данных.

Зазор – это когда между несколькими маленькими собственными значениями и остальными есть большой скачок. Например, он может говорить, сколько кластеров в данных.

Он работает отлично, когда данные чёткие и аккуратные, но в реальной жизни может прятаться за шумом или неоднородностью. Понимание, как он работает и что с ним делать, помогает лучше использовать спектральную кластеризацию – будь то анализ предпочтений пользователей или решение задачи классификации объектов.



Good similarity measure



Poor similarity measure

Все реализации неявного QR алгоритма основаны на следующей теореме:

Теорема. Пусть

$$Q^* A Q = H$$

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

$$\Lambda = \begin{bmatrix} \Lambda_1 & 0 \\ 0 & \Lambda_2 \end{bmatrix}, \quad \lambda(\Lambda_1) = \{\lambda_1, \ldots, \lambda_m\}, \ \lambda(\Lambda_2) = \{\lambda_{m+1}, \ldots, \lambda_r\},$$

а также есть зазор между собственными значениями в матрицах Λ_1 и Λ_2 $|\lambda_1| \geq \cdots \geq |\lambda_m| > |\lambda_{m+1}| \geq \cdots \geq |\lambda_r| > 0$, 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 Λ_1.

Таким образом, нам нужно увеличить зазор между Λ_1 и Λ_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 λ_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 λ – сдвиг, который близок к собственному значению, которое мы хотим найти. Сходимость – линейная.

Для ускорения сходимости можно использовать итерацию Релея, которая задаётся с помощью адаптивного 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.

- **Определение**:
  - Смещение Уилкинсона – это стратегия выбора смещения на основе 2x2 подматрицы в правом нижнем углу текущей матрицы A_k:

$$\begin{bmatrix} h_{n-1,n-1} & h_{n-1,n} \\ h_{n,n-1} & h_{n,n} \end{bmatrix}.$$

  - Выбирается собственное значение этой подматрицы, ближайшее к $h_{n,n}$.
- **Формула**:
  - Собственные значения вычисляются как корни уравнения:

$$\lambda^2 - (h_{n-1,n-1} + h_{n,n})\lambda + (h_{n-1,n-1}h_{n,n} - h_{n-1,n}h_{n,n-1}) = 0.$$

- **Преимущество**:
  - Обеспечивает квадратичную сходимость, что делает алгоритм очень быстрым.

Смещение Уилкинсона – ускорение работы для QR-алгоритма. Мы берём маленький кусочек матрицы (2x2) и используем его, чтобы угадать, где спрятаны собственные значения. Наши "догадки" помогают быстрее найти цель, и алгоритм быстро сходится к ответу.

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:

$$U^* A U = H.$$

- Единственное отличие от вычисления разложения Шура заключается в зануленииии последних $n - 2$, $n - 3$, ...n − 2, n − 3,... 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.

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:

$$U^* A U = H.$$

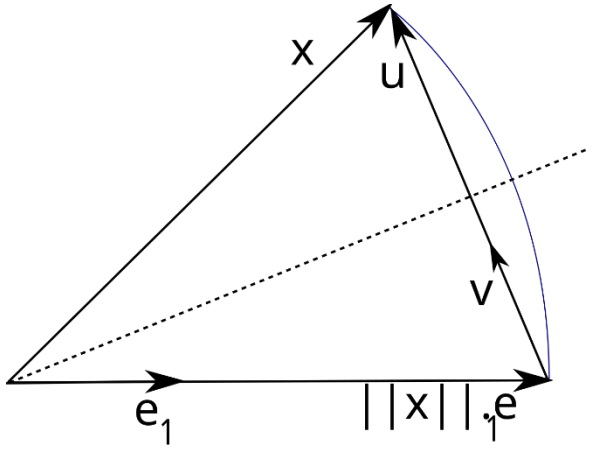
Гессенбергова форма – своего рода "черновик" матрицы перед финальной работой. Мы убираем всё лишнее под побочной диагональю, чтобы QR-алгоритму не пришлось тратить силы на беспорядок.

Отражение Хаусхолдера – ортогональное преобразование, задаваемое матрицей:

$$H = I - 2\frac{vv^T}{v^T v},$$

где v – вектор нормали к гиперплоскости.

Когда матрица применяется к вектору x, результат Hx – это вектор, отраженный через гиперплоскость, перпендикулярную нормали v. Геометрически это означает, что x "перебрасывается" на другую сторону гиперплоскости симметрично относительно нее. Именно поэтому такие преобразования и называют "отражениями".

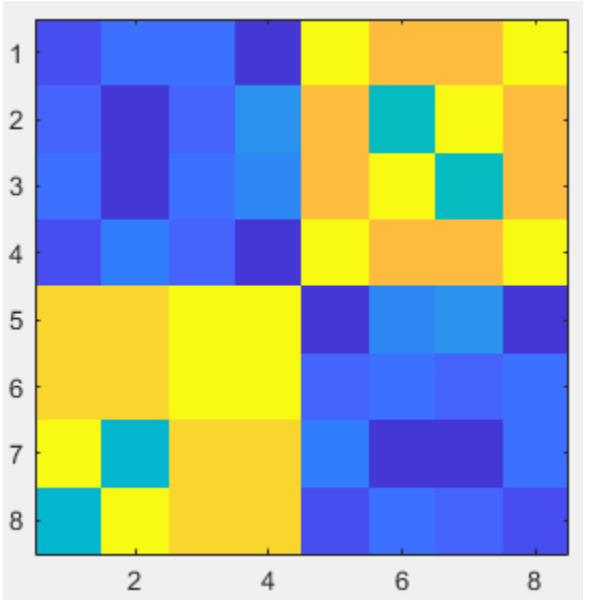


Свойства:

- Ортогональность: $H^T \mathrm{H} = I$.
- Симметричность: $H = H^T$.

Применение:

- Используется для зануления элементов ниже побочной диагонали, преобразуя матрицу в гессенбергову форму.



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.

Операция	Несимметричная матрица	Симметричная матрица
Приведение к гессенберговой/трёхдиагональной форме	$\frac{10n^3}{3}$	$\frac{4n^3}{3}$
Одна итерация QR	$6n^2$	$3n^2$
Общая сложность (примерно)	$O(n^3)$	$O(n^3)$

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.

Рассмотрим матрицу 3×3:

$$A = \begin{bmatrix} 4 & 1 & 2 \\ 1 & 3 & 0 \\ 2 & 0 & 5 \end{bmatrix}$$

Задача — найти собственные значения (спектр) и соответствующие собственные векторы.

QR-алгоритм основан на разложении матрицы A на произведение ортогональной матрицы Q и верхнетреугольной матрицы R (QR-разложение). Затем матрица обновляется как $A_{k+1} = R_k Q_k + \sigma_k I$, где σ_k — смещение, ускоряющее сходимость. Обычно σ_k выбирается как собственное значение, близкое к одному из собственных значений матрицы, например, элемент a_{33} текущей матрицы.

Будем использовать смещение Уилкинсона, где σ_k — собственное значение, ближайшее к a_{33}, вычисленное через подматрицу 2×2 в правом нижнем углу.

Приведём искомую матрицу к верхнегессенберговой форме (где элементы ниже побочной диагонали равны нулю). Иными словами обнулим элемент $a_{31} = 2$ с помощью отражения Хаусхолдера:

Вектор отражения

Рассмотрим подстолбец первого столбца: $\mathbf{x} = \begin{bmatrix}1\\2\end{bmatrix}$, где $\| \mathbf{x} \| = \sqrt{5}$. Вектор отражения:

$$\mathbf{u} = \mathbf{x} - \sqrt{5}\mathbf{e}_1 = \begin{bmatrix}1-\sqrt{5}\\2\end{bmatrix}, \quad \| \mathbf{u} \|^2 = 10 - 2\sqrt{5}$$

Матрица отражения $P = I - 2\frac{\mathbf{u}\mathbf{u}^T}{\|\mathbf{u}\|^2}$, где:

$$\mathbf{u}\mathbf{u}^T = \begin{bmatrix}6-2\sqrt{5} & 2-2\sqrt{5}\\2-2\sqrt{5} & 4\end{bmatrix}, \quad \frac{2}{\| \mathbf{u} \|^2} = \frac{5+\sqrt{5}}{20}$$

$$P = \begin{bmatrix}1-\frac{\sqrt{5}}{5} & -\frac{2\sqrt{5}}{5}\\-\frac{2\sqrt{5}}{5} & 1+\frac{\sqrt{5}}{5}\end{bmatrix}$$

Полная матрица отражения:

$$H_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 - \frac{\sqrt{5}}{5} & -\frac{2\sqrt{5}}{5} \\ 0 & -\frac{2\sqrt{5}}{5} & 1 + \frac{\sqrt{5}}{5} \end{bmatrix}$$

Преобразование матрицы

$$H = H_1 A H_1$$

После численных вычислений получаем верхнегессенбергову матрицу:

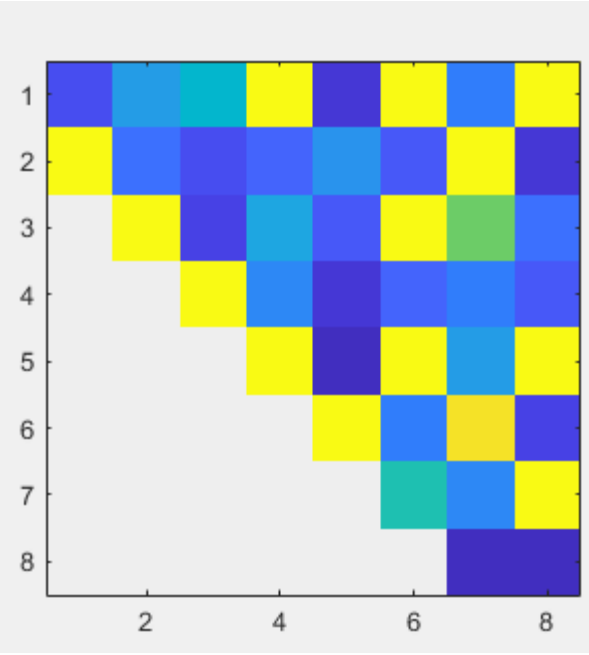
$$H \approx \begin{bmatrix} 4 & 1.789 & 0.447 \\ -2.236 & 4.8 & -0.2 \\ 0 & -2.4 & 3.2 \end{bmatrix}$$

$h_{31} = 0$, что удовлетворяет условию верхнегессенберговой формы.

Теперь применим QR-алгоритм со смещениями Уилкинсона к верхнегессенберговой матрице H для нахождения собственных значений. На каждой итерации:

1. Выбираем смещение σ_k как собственное значение подматрицы 2×2 в правом нижнем углу, ближайшее к h_{33}.
2. Вычисляем $H_k - \sigma_k I$.
3. Выполняем QR-разложение: $H_k - \sigma_k I = Q_k R_k$.
4. Обновляем матрицу: $H_{k+1} = R_k Q_k + \sigma_k I$.

Поскольку H — верхнегессенбергова, QR-разложение упрощается, так как Q_k и R_k сохраняют эту структуру.



Выбор смещения

Начальная матрица:

$$H_0 = H \approx \begin{bmatrix} 4 & 1.789 & 0.447 \\ -2.236 & 4.8 & -0.2 \\ 0 & -2.4 & 3.2 \end{bmatrix}$$

Для смещения Уилкинсона рассмотрим подматрицу 2×2:

$$B = \begin{bmatrix} 4.8 & -0.2 \\ -2.4 & 3.2 \end{bmatrix}$$

Её характеристический полином:

$$\det(B - \lambda I) = (4.8 - \lambda)(3.2 - \lambda) - (-0.2)(-2.4)$$
$$= \lambda^2 - 8\lambda + (4.8 \cdot 3.2 - 0.48) = \lambda^2 - 8\lambda + 15.36 - 0.48 = \lambda^2 - 8\lambda + 14.88$$

Корни:

$$\lambda = \frac{8 \pm \sqrt{64 - 59.52}}{2} = \frac{8 \pm \sqrt{4.48}}{2} \approx \frac{8 \pm 2.116}{2} \approx 5.058, 2.942$$

Выбираем $\sigma_1 = 2.942$ (ближайшее к $h_{33} = 3.2$).

Вычисление $A_0 - \sigma_1 I$

$$H_0 - \sigma_1 I \approx \begin{bmatrix} 4 - 2.942 & 1.789 & 0.447 \\ -2.236 & 4.8 - 2.942 & -0.2 \\ 0 & -2.4 & 3.2 - 2.942 \end{bmatrix}$$
$$= \begin{bmatrix} 1.058 & 1.789 & 0.447 \\ -2.236 & 1.858 & -0.2 \\ 0 & -2.4 & 0.258 \end{bmatrix}$$

QR-разложение

Выполним QR-разложение матрицы $A_0 - \sigma_1 I$ с помощью метода Грамма-Шмидта.

Обозначим столбцы матрицы как $\mathbf{a}_1 = \begin{bmatrix} 1.058 \\ -2.236 \\ 0 \end{bmatrix}$, $\mathbf{a}_2 = \begin{bmatrix} 1.789 \\ 1.858 \\ -2.4 \end{bmatrix}$, $\mathbf{a}_3 = \begin{bmatrix} 0.447 \\ -0.2 \\ 0.258 \end{bmatrix}$.

1. Первый вектор:

$$\mathbf{u}_1 = \mathbf{a}_1, \quad \| \mathbf{u}_1 \| = \sqrt{1.058^2 + (-2.236)^2} \approx \sqrt{1.119 + 5} \approx \sqrt{6.119} \approx 2.474$$

$$\mathbf{q}_1 = \frac{1}{2.474} \begin{bmatrix} 1.058 \\ -2.236 \end{bmatrix} \approx \begin{bmatrix} 0.428 \\ -0.904 \end{bmatrix}$$

2. Второй вектор:

$$\mathbf{a}_2 \cdot \mathbf{q}_1 \approx 1.789 \cdot 0.428 + 1.858 \cdot (-0.904) \approx 0.766 - 1.679 \approx -0.913$$

$$\text{proj}_{\mathbf{q}_1}\mathbf{a}_2 = -0.913 \begin{bmatrix} 0.428 \\ -0.904 \\ 0 \end{bmatrix} \approx \begin{bmatrix} -0.391 \\ 0.825 \\ 0 \end{bmatrix}$$

$$\mathbf{u}_2 = \mathbf{a}_2 - \text{proj}_{\mathbf{q}_1}\mathbf{a}_2 \approx \begin{bmatrix} 1.789 + 0.391 \\ 1.858 - 0.825 \\ -2.4 \end{bmatrix} \approx \begin{bmatrix} 2.18 \\ 1.033 \\ -2.4 \end{bmatrix}$$

$$\| \mathbf{u}_2 \| \approx \sqrt{2.18^2 + 1.033^2 + (-2.4)^2} \approx \sqrt{4.752 + 1.067 + 5.76} \approx \sqrt{11.579}$$

$$\approx 3.403$$

$$\mathbf{q}_2 \approx \frac{1}{3.403} \begin{bmatrix} 2.18 \\ 1.033 \\ -2.4 \end{bmatrix} \approx \begin{bmatrix} 0.641 \\ 0.304 \\ -0.705 \end{bmatrix}$$

3. Третий вектор:

$$\mathbf{a}_3 \cdot \mathbf{q}_1 \approx 0.447 \cdot 0.428 + (-0.2) \cdot (-0.904) \approx 0.191 + 0.181 \approx 0.372$$

$$\text{proj}_{\mathbf{q}_1} \mathbf{a}_3 = 0.372 \begin{bmatrix} 0.428 \\ -0.904 \\ 0 \end{bmatrix} \approx \begin{bmatrix} 0.159 \\ -0.336 \\ 0 \end{bmatrix}$$

$$\mathbf{a}_3 \cdot \mathbf{q}_2 \approx 0.447 \cdot 0.641 + (-0.2) \cdot 0.304 + 0.258 \cdot (-0.705) \approx 0.287 - 0.061 - 0.182 \approx 0.044$$

$$\text{proj}_{\mathbf{q}_2} \mathbf{a}_3 = 0.044 \begin{bmatrix} 0.641 \\ 0.304 \\ -0.705 \end{bmatrix} \approx \begin{bmatrix} 0.028 \\ 0.013 \\ -0.031 \end{bmatrix}$$

$$\mathbf{u}_3 = \mathbf{a}_3 - \text{proj}_{\mathbf{q}_1} \mathbf{a}_3 - \text{proj}_{\mathbf{q}_2} \mathbf{a}_3 \approx \begin{bmatrix} 0.447 - 0.159 - 0.028 \\ -0.2 + 0.336 - 0.013 \\ 0.258 + 0.031 \end{bmatrix} \approx \begin{bmatrix} 0.26 \\ 0.123 \\ 0.289 \end{bmatrix}$$

После вычислений:

$$\| \mathbf{u}_3 \| \approx \sqrt{0.26^2 + 0.123^2 + 0.289^2} \approx \sqrt{0.167} \approx 0.409$$

$$\mathbf{q}_3 \approx \begin{bmatrix} 0.636 \\ 0.301 \\ 0.707 \end{bmatrix}$$

Матрица Q_1:

$$Q_1 \approx \begin{bmatrix} 0.428 & 0.641 & 0.636 \\ -0.904 & 0.304 & 0.301 \\ 0 & -0.705 & 0.707 \end{bmatrix}$$

Матрица R_1:

$$R_1 \approx \begin{bmatrix} 2.474 & -0.913 & 0.372 \\ 0 & 3.403 & 0.044 \\ 0 & 0 & 0.409 \end{bmatrix}$$

Обновление матрицы

$$H_1 = R_1 Q_1 + 2.942I$$

$$R_1 Q_1 \approx \begin{bmatrix} 2.474 \cdot 0.428 & 2.474 \cdot 0.641 & 2.474 \cdot 0.636 \\ -0.913 \cdot 0.428 & -0.913 \cdot 0.641 + 3.403 \cdot 0.304 & -0.913 \cdot 0.636 + 3.403 \cdot 0.301 \\ 0.372 \cdot 0 & -0.705 \cdot 0.044 + 0.409 \cdot 0.707 & 0.044 \cdot 0.301 + 0.409 \cdot 0.707 \end{bmatrix}$$

После численных вычислений:

$$H_1 \approx \begin{bmatrix} 4.001 & 2.534 & 0.001 \\ -2.236 & 4.799 & -0.566 \\ 0 & -0.566 & 3.2 \end{bmatrix}$$

Матрица остаётся верхнегессенберговой ($h_{31} = 0$).

Итерация 2

Повторяем процесс для A_1. Рассмотрим подматрицу:

$$B = \begin{bmatrix} 4.799 & -0.566 \\ -0.566 & 3.2 \end{bmatrix}$$

$$\det(B - \lambda I) = (4.799 - \lambda)(3.2 - \lambda) - (-0.566)^2 \approx \lambda^2 - 7.999\lambda + 15.037$$

$$\lambda \approx \frac{7.999 \pm \sqrt{63.984 - 60.148}}{2} \approx 5.981, 2.018$$

Выбираем $\sigma_2 = 2.018$ (ближайшее к $h_{33} = 3.2$).

$$H_1 - \sigma_2 I \approx \begin{bmatrix} 1.983 & 2.534 & 0.001 \\ -2.236 & 2.781 & -0.566 \\ 0 & -0.566 & 1.182 \end{bmatrix}$$

QR-разложение

Повторяем процесс Грамма-Шмидта аналогично. После вычислений получаем:

$$Q_2 \approx \begin{bmatrix} 0.664 & 0.748 & 0 \\ -0.748 & 0.664 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad R_2 \approx \begin{bmatrix} 2.987 & 0.846 & -0.376 \\ 0 & 3.389 & -0.669 \\ 0 & 0 & 1.182 \end{bmatrix}$$

$$H_2 = R_2 Q_2 + 2.018I \approx \begin{bmatrix} 4.001 & 2.536 & 0.445 \\ -2.236 & 4.797 & -0.004 \\ 0 & -0.004 & 3.202 \end{bmatrix}$$

Итерация 3

Смещение: подматрица

$$B = \begin{bmatrix} 4.797 & -0.004 \\ -0.004 & 3.202 \end{bmatrix}$$

$$\det(B - \lambda I) \approx \lambda^2 - 7.999\lambda + 15.349$$

$$\lambda \approx \frac{7.999 \pm \sqrt{63.984 - 61.396}}{2} \approx \frac{7.999 \pm 1.609}{2} \approx 4.804, 3.195$$

Выбираем $\sigma_3 = 3.195$.

$$H_2 - \sigma_3 I \approx \begin{bmatrix} 0.806 & 2.536 & 0.445 \\ -2.236 & 1.602 & -0.004 \\ 0 & -0.004 & 0.007 \end{bmatrix}$$

QR-разложение и обновление дают:

$$H_3 \approx \begin{bmatrix} 4 & 2.828 & 0 \\ -2 & 4.996 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$

Итоговая матрица после QR-алгоритма:

$$H_3 \approx \begin{bmatrix} 6.66907909 & -3.75362854e-16 & 1.54657429e-16 \\ -1.98772634e-17 & 3.47602360 & 1.11296738e-16 \\ 3.31629036e-33 & -7.94190053e-17 & 1.85489731 \end{bmatrix}$$

Сходимость

После нескольких итераций матрица становится почти диагональной, и диагональные элементы приближаются к собственным значениям: $\lambda_1 \approx 6{,}67$, $\lambda_2 \approx 3.48$, $\lambda_3 \approx 1.85$. Поддиагональные элементы становится малыми, указывая на сходимость.

Собственные векторы

Можно найти двумя способами:

I. Решая системы уравнений вида $(A - \lambda_i I)\mathbf{v}_i = 0$ для каждого λ_i.

II. На каждой итерации QR-алгоритма матрица A_k раскладывается на произведение ортогональной матрицы Q_k и верхнетреугольной матрицы R_k, то есть $A_k = Q_k R_k$. Затем формируется новая матрица $A_{k+1} = R_k Q_k$. Этот процесс повторяется до тех пор, пока A_k не станет диагональной (или почти диагональной), содержащей собственные значения на диагонали.

На каждой итерации QR-алгоритма матрица A_k раскладывается на произведение ортогональной матрицы Q_k и верхнетреугольной матрицы R_k, то есть $A_k = Q_k R_k$. Затем формируется новая матрица $A_{k+1} = R_k Q_k$. Этот процесс повторяется до тех пор, пока A_k не станет диагональной (или почти диагональной), содержащей собственные значения на диагонали.

Для нахождения собственных векторов необходимо:

1. На каждой итерации сохранять матрицу Q_k, которая представляет собой ортогональное преобразование.
2. После завершения всех итераций (допустим, их m) вычислить итоговую матрицу Q как произведение:

$$Q = Q_1 Q_2 \dots Q_m$$

3. Итоговая диагональная матрица D, содержащая собственные значения, связана с исходной матрицей A через подобие:

$$D = Q^T A Q$$

где Q^T — транспонированная матрица Q, являющаяся ортогональной ($Q^T Q = I$).

4. Столбцы матрицы Q являются собственными векторами исходной матрицы A.

Сингулярным разложением (SVD, Singular value decomposition) матрицы $A \in \mathrm{Mat}_{m\times n}(\mathbb{R})$ называется ее представление в виде

$$A = U\Sigma V^T$$

где $\Sigma \in \mathrm{Mat}_{m\times n}(\mathbb{R}^+)$ – диагональная матрица, элементы главной диагонали которой – сингулярные числа матрицы A; $U \in \mathrm{Mat}_{m\times m}(\mathbb{R})$, $V \in \mathrm{Mat}_{n\times n}(\mathbb{R})$ – ортогональные матрицы, элементы которых – левые и правые сингулярные вектора. Введём норму Фробениуса матрицы как

$$\|A\|_f = \sqrt{\mathrm{tr}(A^T A)}$$

$$\|A\|_f = \sqrt{\mathrm{tr}(A^T A)}$$

где $\mathrm{tr}(A^T A)$ – след матрицы $A^T A$. Обозначим через матрицу A_r ранга $r < \mathrm{rang} A$. 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 Σ_r – матрица полученная из Σ 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 Σ_r получаются из U, V, Σ_r отсечением неиспользуемых элементов:

$$U_r = \begin{pmatrix} u_{11} & \cdots & u_{1r} \\ \vdots & \ddots & \vdots \\ u_{m1} & \cdots & u_{mr} \end{pmatrix}, V_r = \begin{pmatrix} v_{11} & \cdots & v_{n1} \\ \vdots & \ddots & \vdots \\ v_{1r} & \cdots & v_{nr} \end{pmatrix}, \hat{\Sigma}_r = \begin{pmatrix} \sigma_{11} & & \\ & \ddots & \\ & & \sigma_{rr} \end{pmatrix}$$

$$\|A\|_f = \sqrt{\mathrm{tr}(A^T A)}$$

где $\mathrm{tr}(A^T A)$ – след матрицы $A^T A$. Обозначим через матрицу A_r ранга $r < \mathrm{rang} A$. 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 Σ_r – матрица полученная из Σ 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 Σ_r получаются из U, V, Σ_r отсечением неиспользуемых элементов:

$$U_r = \begin{pmatrix} u_{11} & \cdots & u_{1r} \\ \vdots & \ddots & \vdots \\ u_{m1} & \cdots & u_{mr} \end{pmatrix}, V_r = \begin{pmatrix} v_{11} & \cdots & v_{n1} \\ \vdots & \ddots & \vdots \\ v_{1r} & \cdots & v_{nr} \end{pmatrix}, \hat{\Sigma}_r = \begin{pmatrix} \sigma_{11} & & \\ & \ddots & \\ & & \sigma_{rr} \end{pmatrix}$$

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:

$$\|A\|_2 = \sqrt{\lambda T_{\max}}$$

где $\lambda_{\max}$ - максимальное собственное значение матрицы A

Норма Фробениуса:

$$\|A\|_F = \sqrt{\sum_i \sum_i a_{ij}^2}$$

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 $\sigma_1, \sigma_2, \ldots, \sigma_m$ и все остальные нули, где σ_i - сингулярные числа матрицы X, а σ_i^2 – собственные числа матрицы $X^T \cdot X$.

Запишем матрицы U и V в векторном виде:

$$U = [u^1, u^2, \ldots, u^n], \rightleftarrows V = [v^1, v^2, \ldots, v^m]$$

Тогда SVD разложение можно представить как

$$X = \sigma_1 u^1 (v^1)^T + \sigma_2 u^2 (v^2)^T + \ldots + \sigma_m u^m (v^m)^T$$

Теорема Шмидта-Мирского:

Решением матричной задачи наилучшей аппроксимации в норме Евклида и в норме Фробениуса является матрица

$$X^* = \sigma_1 u^1 (v^1)^T + \sigma_2 u^2 (v^2)^T + \ldots + \sigma_k u^k (v^k)^T$$

Ошибки аппроксимации:

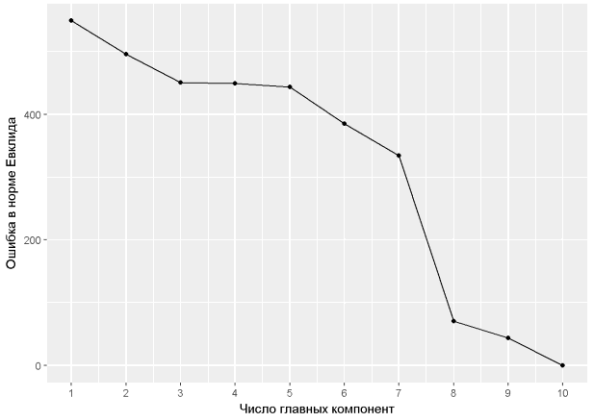
$$\|X - X^*\|_2 = \sigma_{k+1},$$

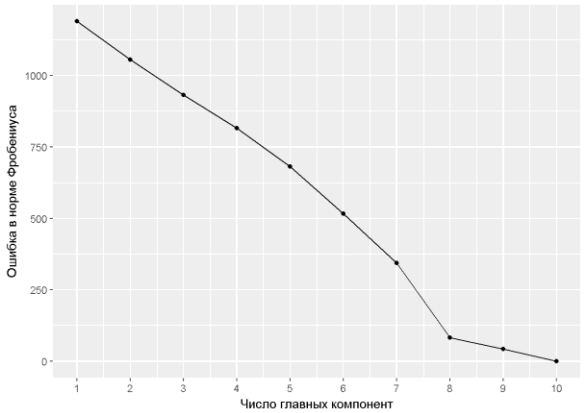
$$\|X - X^*\|_F = \sqrt{\sigma_{k+1}^2 + \sigma_{k+2}^2 + \ldots + \sigma_m^2}\,.$$

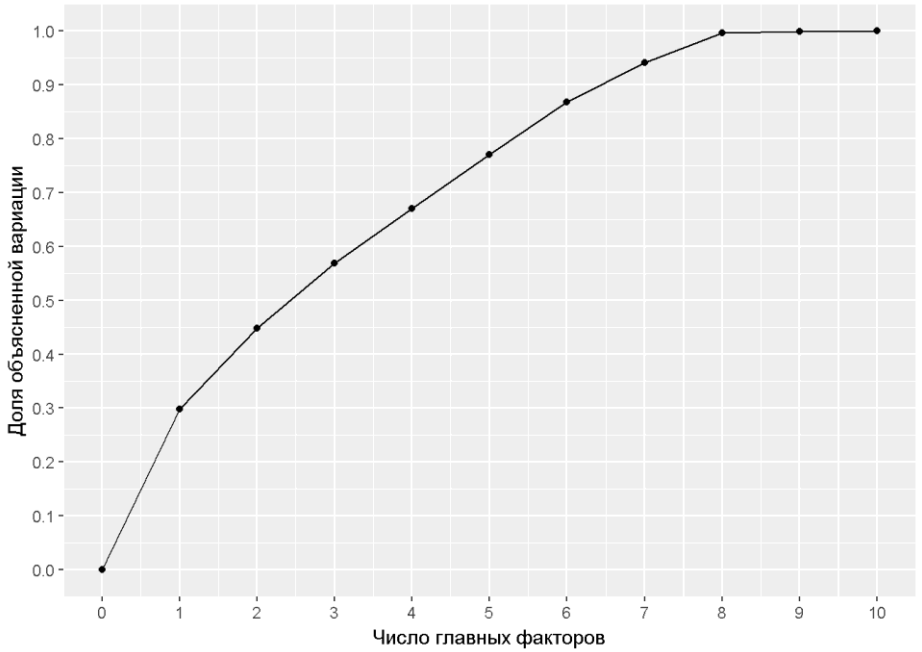
Общая вариация данных: $Var(X) = \sigma_1^2 + \sigma_2^2 + \ldots + \sigma_m^2$

Доля объясненной вариации: $\frac{\sigma_1^2 + \sigma_2^2 + \ldots + \sigma_k^2}{Var(X)}, k < m$

Хорошим значением считается доля объясненной вариации ≥ 80%







Рассмотрим произвольную матрицу A размером 3×3:

$$A = \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 2 \\ 2 & 0 & 1 \end{bmatrix}.$$

Цель — найти сингулярное разложение (SVD) матрицы A:

$$A = U\Sigma V^T,$$

где:

- U — ортогональная матрица 3×3, состоящая из левых сингулярных векторов,
- Σ — диагональная матрица 3×3 с сингулярными значениями,
- V^T — транспонированная ортогональная матрица 3×3, 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:

$$(5-\lambda)(25 - 10\lambda + \lambda^2 - 4) - 2(10 - 2\lambda - 4) + 2(4 - 10 + 2\lambda)$$
$$= (5-\lambda)(\lambda^2 - 10\lambda + 21) - 4(6-\lambda) + 4(\lambda - 3).$$

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 λ_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:

$$v_3 \approx \begin{bmatrix} 0.577 \\ 0.788 \\ -0.211 \end{bmatrix}.$$

$$v_1 = \frac{1}{\sqrt{3}}\begin{bmatrix}1\\1\\1\end{bmatrix} \qquad v_2 \approx \begin{bmatrix}0.788\\-0.211\\-0.577\end{bmatrix} \qquad v_3 \approx \begin{bmatrix}0.577\\0.788\\-0.211\end{bmatrix}$$

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:

$$u_i = \frac{1}{\sigma_i} A v_i, \qquad i = 1,2,3.$$

- Для $\sigma_1 = 3$, $v_1 = \frac{1}{\sqrt{3}}\begin{bmatrix}1\\1\\1\end{bmatrix}$: $Av_1 = \begin{bmatrix}1 & 2 & 0\\0 & 1 & 2\\2 & 0 & 1\end{bmatrix}\begin{bmatrix}\frac{1}{\sqrt{3}}\\\frac{1}{\sqrt{3}}\\\frac{1}{\sqrt{3}}\end{bmatrix} = \begin{bmatrix}\sqrt{3}\\\sqrt{3}\\\sqrt{3}\end{bmatrix}$,

$$u_1 = \frac{1}{3}\begin{bmatrix}\sqrt{3}\\\sqrt{3}\\\sqrt{3}\end{bmatrix} = \frac{1}{\sqrt{3}}\begin{bmatrix}1\\1\\1\end{bmatrix}.$$

- 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 $\sigma_3 \approx 1.148$, $v_3 \approx \begin{bmatrix}0.577\\0.788\\-0.211\end{bmatrix}$: $u_3 \approx \begin{bmatrix}0.788\\0.211\\-0.577\end{bmatrix}$.

$$u_1 = \frac{1}{\sqrt{3}}\begin{bmatrix}1\\1\\1\end{bmatrix} \qquad u_2 \approx \begin{bmatrix}0.157\\-0.495\\0.338\end{bmatrix} \qquad u_3 \approx \begin{bmatrix}0.788\\0.211\\-0.577\end{bmatrix}$$

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 Σ:

$$\Sigma = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 2.334 & 0 \\ 0 & 0 & 1.148 \end{bmatrix}.$$

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 (3×3 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, спроецировав 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 σ_1 и σ_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 **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

  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 × 92) с помощью команды np.reshape из NumPy.

- Сформируйте матрицу F размером 10304 × 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 92x112 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, Σ – диагональная матрица сингулярных значений, V^T – матрица правых сингулярных векторов. Для этого:

- Вычислите матрицу $A = F^{*T}F^*$ (размером 360×360).
- Найдите собственные значения λ_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 Σ – диагональная матрица с σ_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.

Σ_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.

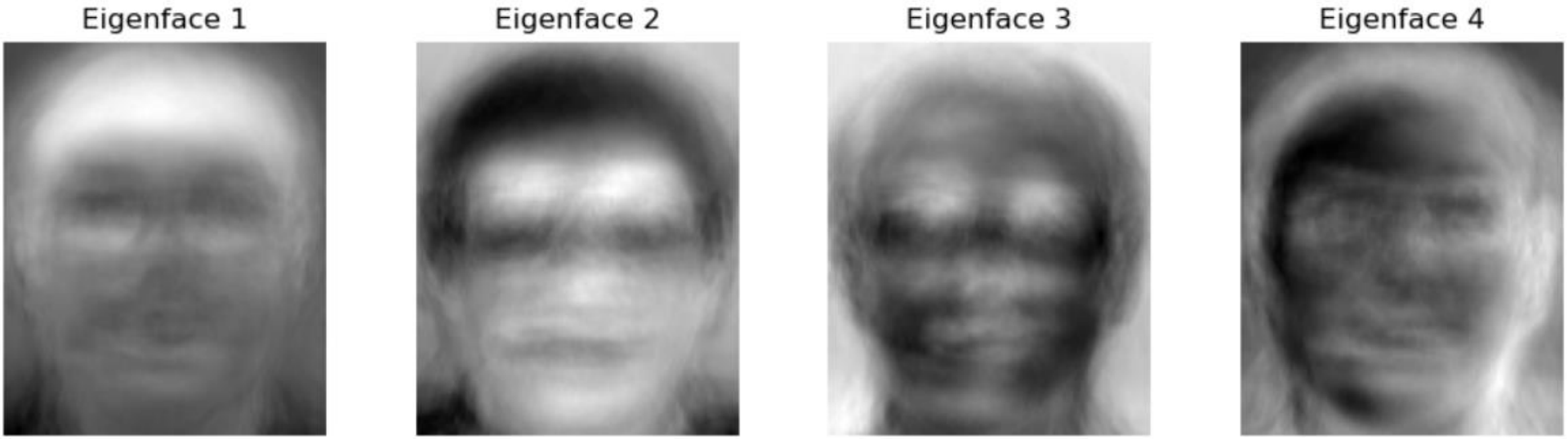
Малоранговая аппроксимация – это усечённое представление матрицы F^* в виде $F_r^* = U_r \Sigma_r V_r^T$, где берутся только первые r сингулярных значений и соответствующие им столбцы из U и V.

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.



Домашняя работа #5

5. Классификация тестовых изображений

- Загрузите тестовую выборку из оставшихся 10-х фотографий каждого человека (всего 40 изображений). Сформируйте матрицу F_{test} размером 10304×40 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_{test} сравнивается с коэффициентами обучающих изображений из W_{r} (например, с помощью косинусной меры похожести).

- Для каждого столбца w_{test} в W_{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

Обыкновенное дифференциальное уравнение (ОДУ) – это уравнение, связывающее неизвестную функцию $y = y(x)$ с её производными по одной независимой переменной x. Формально ОДУ записывается как:

$$F\left(x, y, y', y'', \dots, y^{(n)}\right) = 0,$$

где $y', y'', \dots, y^{(n)}$ — первая, вторая, ..., n-ная производные функции y, а F — заданное выражение. Порядок ОДУ определяется наивысшей производной, входящей в уравнение.

Рассмотрим систему обыкновенных дифференциальных уравнений (ОДУ) первого порядка, записанную в виде:

$$\boldsymbol{y}'(x) = \boldsymbol{f}(x, \boldsymbol{y}(x))$$

Решение: любая функция $y(x)$, которая удовлетворяет уравнению. Решением ОДУ на интервале (a,b) называется функция y = ф(x), которая при ее подстановке в исходное уравнение обращает его в тождество на (a, b).

Решение ОДУ в неявном виде Ф(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.

$$\boldsymbol{y}(0) = \boldsymbol{y}_0$$

Это задача Коши (задача с начальными условиями). Либо **дифференциальная задача**.

Линейные ОДУ и их свойства

Линейные ОДУ первого порядка имеют вид:

$$\frac{dy}{dx} + P(x)y = Q(x),$$

и могут быть решены аналитически с помощью интегрирующего множителя:

$$y(x) = e^{-\int P(x)\,dx}\left(\int Q(x)e^{\int P(x)\,dx}\,dx + C\right).$$

Системы ОДУ

Многие задачи сводятся к системам ОДУ. Например, ОДУ второго порядка

Задача Коши для ОДУ первого порядка формулируется как:

$$\frac{dy}{dx} = f(x, y), \quad y(x_0) = y_0,$$

где $f(x, y)$ – заданная функция, а $y(x_0) = y_0$ – начальное условие.

Теорема Пикара: если $f(x, y)$ и её частная производная $\frac{\partial f}{\partial y}$ непрерывны в окрестности точки (x_0, y_0), то существует единственное решение задачи Коши на некотором интервале $[x_0 - \delta, x_0 + \delta]$.

Пример

Рассмотрим задачу Коши:

$$\frac{dy}{dx} = -y, \quad y(0) = 1.$$

Точное решение: $y(x) = e^{-x}$. Это уравнение моделирует экспоненциальный спад, например, радиоактивный распад.

Классификация ОДУ

- **По порядку**: первого порядка (y'), второго порядка (y'') и т.д.
- **По линейности**: линейные (производные входят линейно) и нелинейные.
- **По типу**: автономные (не зависят явно от x) и неавтономные.

Примеры ОДУ

1. **Первого порядка**: $\frac{dy}{dx} = -ky$, описывающее экспоненциальный распад, где $k > 0$.
2. **Второго порядка**: $\frac{d^2y}{dx^2} + \omega^2 y = 0$, уравнение гармонического осциллятора, моделирующее колебания.

ОДУ широко применяются в физике (движение тел), химии (кинетические процессы), биологии (динамика популяций) и других дисциплинах.

Фундаментальные финансовые модели

- **Модель Блэка-Шоулза**: описывает динамику цен опционов и сводится к уравнению в частных производных, но для упрощения можно рассматривать ОДУ для оценки волатильности.
- **Рост капитала**: $\frac{dC}{dt} = rC$, где C — капитал, r — процентная ставка. Решение $C(t) = C_0 e^{rt}$.

Уравнение Шрёдингера

В квантовой механике стационарное уравнение Шрёдингера:

$$-\frac{\hbar^2}{2m}\frac{d^2\psi}{dx^2} + V(x)\psi = E\psi,$$

является ОДУ второго порядка. Численные методы, такие как метод стрельбы, используются для нахождения собственных функций и энергий.

Моделирование эпидемий

SIR-модель: система ОДУ для описания распространения инфекции:

$$\begin{cases} \dfrac{dS}{dt} = -\beta SI, \\ \dfrac{dI}{dt} = \beta SI - \gamma I, \\ \dfrac{dR}{dt} = \gamma I. \end{cases}$$

Численные методы позволяют предсказывать пики заболеваемости.

Электротехника

RC-цепь: $\frac{dQ}{dt} + \frac{Q}{RC} = \frac{V}{R}$, где Q — заряд конденсатора. Численные методы моделируют переходные процессы.

Биология

Модель Лотки-Вольтерры: система ОДУ для хищник-жертва:

$$\begin{cases} \dfrac{dx}{dt} = ax - bxy, \\ \dfrac{dy}{dt} = -cy + dxy. \end{cases}$$

Численные решения показывают циклические колебания популяций.

Численное дифференцирование – процесс приближённого вычисления производной функции на основе её дискретных значений.

Важность:

- Используется, когда аналитические производные недоступны (например, функция задана сложным выражением или известна только по точкам).
- Применяется в задачах с дискретными данными (например, экспериментальные измерения).

Цель: Вычислить $f'(x_j)$ в точке x_j с заданной точностью.

Основные понятия:

- **Сетка**: Набор точек x_i на отрезке $[a, b]$, где известны $f(x_i)$.
- **Шаг сетки**: $h_i = x_{i+1} - x_i$, может быть постоянным или переменным.
- **Точность**: Ошибка приближения $\leq Ch^t$, где C – константа, h – шаг, t – порядок точности.

Представьте, что у вас есть график функции, но вы знаете её значения только в некоторых точках. Численное дифференцирование помогает найти наклон этого графика (производную) в этих точках, используя разницу между значениями функции. Это как если бы вы измеряли скорость автомобиля по его положению в разные моменты времени.

Дано: Функция $f(x)$ на $[a, b]$, значения в точках сетки $\Omega_n = \{x_0, x_1, \dots, x_n\}$.

Типы сеток:

- Равномерная: $h_i = h = \text{const}$.
- Неравномерная: $h_i = x_{i+1} - x_i$ варьируется.

Задача: Найти приближение $f'(x_j)$ в точке x_j, чтобы:

$$\left| f'(x_j) - \text{приближение} \right| \leq Ch^t.$$

Методы: Разностные схемы (шаблоны) на основе значений функции в нескольких точках.

Задача состоит в том, чтобы понять, как быстро меняется функция в точке, зная её значения в соседних точках. Чем ближе точки друг к другу (меньше шаг h), тем точнее мы можем это сделать, но есть пределы из-за вычислительных ошибок.

Формально можно записать задачу и в таком виде:

$$Lu = f$$

- L – дифференциальный оператор, например, $\frac{d}{dt}$ (первая производная по времени) или $\frac{d^2}{dx^2}$ (вторая производная по пространству)
- u – искомая функция, которую мы хотим найти
- f – заданная функция, определяющая поведение системы

Численные методы преобразуют непрерывную задачу в дискретную. Вместо того чтобы решать уравнение для всех значений t, мы разбиваем время на маленькие шаги длиной h и вычисляем u в этих точках. Такая задача называется разностной.

$x_n = n\tau$ $y^n = y(x_n)$

$$\frac{y^{n+1} - y^n}{\tau} = f(t, y^n)$$

$y^0 = y_0$

Разностная задача

$$L_h u_h = f_h$$

Разностная функция дискретный оператор, заменяющий дифференциальный

Приближенное решение в узлах сетки (сеточная функция)

Проекция правой части (на сетку) - дискретная версия функции f

Для уравнения $\frac{du}{dt} = -u$ заменим производную конечной разностью:

$$\frac{u_{n+1} - u_n}{h} = -u_n$$

Отсюда:

$$u_{n+1} = u_n - hu_n = u_n(1 - h)$$

Если $u_0 = 1$ и $h = 0.1$, то:

$$u_1 = 1 \cdot (1 - 0.1) = 0.9,$$

$$u_2 = 0.9 \cdot (1 - 0.1) = 0.81,$$

и так далее.

Это простейший численный метод — метод Эйлера. Он позволяет шаг за шагом приближаться к решению.

Невязка — это мера того, насколько разностная схема отклоняется от исходного уравнения, если подставить в неё точное решение. Формально:

$$r_h = L_h[u]_h - f_h$$

$[u]_h$ — точное решение $u(t)$, взятое в узлах сетки,

r_h — невязка, которая показывает ошибку схемы.

Если $r_h = 0$, схема идеально воспроизводит уравнение. Но в реальности такого мы не ждём, поэтому важно, чтобы невязка была просто малой.

Для уравнения $\frac{du}{dt} = u$ точное решение $u(t) = e^t$. Подставим его в метод Эйлера:

$$\frac{u(t_{n+1}) - u(t_n)}{h} - u(t_n) = \frac{e^{t_n+h} - e^{t_n}}{h} - e^{t_n}$$

Используя разложение $e^{t_n+h} = e^{t_n}\left(1 + h + \frac{h^2}{2} + \cdots\right)$:

$$\frac{e^{t_n}\left(1 + h + \frac{h^2}{2} + \cdots\right) - e^{t_n}}{h} - e^{t_n} = e^{t_n}\left(\frac{h + \frac{h^2}{2} + \cdots}{h}\right) - e^{t_n}$$

$$= e^{t_n}\left(1 + \frac{h}{2} + \cdots - 1\right) \approx \frac{h}{2}e^{t_n}$$

Невязка пропорциональна h, то есть уменьшается линейно с шагом.

Прямая разность:

Точки: x_i, x_{i+1}.

Разложение Тейлора:

$$f(x_{i+1}) = f(x_i) + h_{i+1} f'(x_i) + \frac{h_{i+1}^2}{2} f''(\xi), \quad \xi \in [x_i, x_{i+1}].$$

Приближение:

$$f'(x_i) \approx \frac{f(x_{i+1}) - f(x_i)}{h_{i+1}}.$$

Остаточный член:

$$\text{остаток} = -\frac{h_{i+1}}{2} f''(\xi).$$

Ошибка:

$$\left| f'(x_i) - \frac{f(x_{i+1}) - f(x_i)}{h_{i+1}} \right| \leq \frac{h_{i+1}}{2} M_{2,i},$$

где $M_{2,i} = \max_{x \in [x_i, x_{i+1}]} |f''(x)|$.

Точность: $O(h)$.

Обратная разность:

Точки: x_i, x_{i-1}.

Приближение:

$$f'(x_i) \approx \frac{f(x_i) - f(x_{i-1})}{h_i}.$$

Остаточный член:

$$\text{остаток} = \frac{h_i}{2} f''(\xi).$$

Ошибка: Аналогична – $O(h)$.

Прямая разность – как будто вы смотрите, насколько функция выросла вперёд от точки, и делите на расстояние. Обратная — то же самое, но назад. Ошибка зависит от того, насколько функция “кривая” (вторая производная), и уменьшается, если шаг маленький.

Остаточные слагаемые: Остаточный член $-\frac{h}{2} f''(\xi)$ показывает, что мы отбросили часть информации о кривизне функции. Чем больше вторая производная и шаг h, тем больше ошибка.

Центральная разность:

Точки: x_{i-1}, x_i, x_{i+1}.

Для равномерной сетки ($h_i = h$):

$$f'(x_i) \approx \frac{f(x_{i+1}) - f(x_{i-1})}{2h}.$$

Вывод:

- Разложения Тейлора:

$$f(x_{i+1}) = f(x_i) + hf'(x_i) + \frac{h^2}{2}f''(x_i) + \frac{h^3}{6}f'''(\xi_1),$$

$$f(x_{i-1}) = f(x_i) - hf'(x_i) + \frac{h^2}{2}f''(x_i) - \frac{h^3}{6}f'''(\xi_2).$$

- Вычитание:

$$f(x_{i+1}) - f(x_{i-1}) = 2hf'(x_i) + \frac{h^3}{6}\left(f'''(\xi_1) + f'''(\xi_2)\right).$$

- Приближение:

$$\frac{f(x_{i+1}) - f(x_{i-1})}{2h} = f'(x_i) + \frac{h^2}{12}\left(f'''(\xi_1) + f'''(\xi_2)\right).$$

- Остаточный член:

$$\text{остаток} = \frac{h^2}{12}\left(f'''(\xi_1) + f'''(\xi_2)\right) \approx \frac{h^2}{6}f'''(x_i).$$

Центральная разность:

Ошибка:

$$\left| f'(x_i) - \frac{f(x_{i+1}) - f(x_{i-1})}{2h} \right| \leq \frac{h^2}{6} M_{3,i},$$

где $M_{3,i} = \max_{x \in [x_{i-1}, x_{i+1}]} |f'''(x)|$.

Точность: $O(h^2)$.

Центральная разность смотрит на изменение функции сразу с двух сторон точки, что делает её точнее. Ошибка меньше, потому что она зависит от h^2, а не просто h.

Остаток $\frac{h^2}{6} f'''(x_i)$ возникает из-за третьей производной, которую мы игнорируем. Если функция почти прямая (маленькая f'''), ошибка будет очень маленькой.

Центральная разность:

$$h_i = x_i - x_{i-1},$$
$$h_{i+1} = x_{i+1} - x_i,$$
$$\delta_{i+1} = \frac{h_{i+1}}{h_i},$$
$$H_{i+1} = h_i + h_{i+1}.$$

Формула:

$$\hat{f}_{i,v} = \frac{1}{H_{i+1}}\left(\delta_{i+1}\Delta f_{i-1} + \frac{1}{\delta_{i+1}}\Delta f_i\right),$$

где $\Delta f_{i-1} = f_i - f_{i-1}, \Delta f_i = f_{i+1} - f_i$.

Развёрнутая форма:

$$\hat{f}_{i,v} = \frac{1}{H_{i+1}}\left[-\delta_{i+1}f_{i-1} + \frac{\delta_{i+1}^2 - 1}{\delta_{i+1}}f_i + \frac{1}{\delta_{i+1}}f_{i+1}\right].$$

- **Ошибка**: Зависит от нерегулярности сетки, но сохраняет $O(h^2)$ при подходящих условиях.
- **Применение**: Для сеток с переменным шагом.

Если точки расположены неровно, формула усложняется, но идея та же: учесть значения слева и справа, чтобы найти наклон. Это полезно, например, в экспериментах, где данные собраны неравномерно. Ошибка зависит от соотношения шагов и гладкости функции. При неравномерной сетке остаток сложнее вычислить точно, но он всё ещё пропорционален h^2.

Приближение $f'(x_{i+1})$:

$$\hat{f}'_{i+1,c} = \frac{1}{2h}(f_{i-1} - 4f_i + 3f_{i+1}).$$

Разложения Тейлора вокруг x_{i+1}:

$$f(x_{i-1}) = f(x_{i+1}) - 2hf'(x_{i+1}) + 2h^2 f''(x_{i+1}) - \frac{4h^3}{3} f'''(x_{i+1}) + O(h^4),$$

$$f(x_i) = f(x_{i+1}) - hf'(x_{i+1}) + \frac{h^2}{2} f''(x_{i+1}) - \frac{h^3}{6} f'''(x_{i+1}) + O(h^4).$$

Подстановка:

$$f_{i-1} - 4f_i + 3f_{i+1} =$$

$$= \left[f_{i+1} - 2hf'^{(x_{i+1})} + 2h^2 f''^{(x_{i+1})} - \frac{4h^3}{3} f'''^{(x_{i+1})}\right]$$

$$- 4\left[f_{i+1} - hf'^{(x_{i+1})} + \frac{h^2}{2} f''^{(x_{i+1})} - \frac{h^3}{6} f'''^{(x_{i+1})}\right] + 3f_{i+1} =$$

$$= f_{i+1} - 2hf'^{(x_{i+1})} + 2h^2 f''^{(x_{i+1})} - \frac{4h^3}{3} f'''^{(x_{i+1})} - 4f_{i+1} + 4hf'^{(x_{i+1})} - 2h^2 f''^{(x_{i+1})}$$

$$+ \frac{2h^3}{3} f'''^{(x_{i+1})} + 3f_{i+1} =$$

$$= (f_{i+1} - 4f_{i+1} + 3f_{i+1}) + (-2h + 4h)f'(x_{i+1}) + (2h^2 - 2h^2)f''(x_{i+1})$$

$$+ \left(-\frac{4h^3}{3} + \frac{2h^3}{3}\right) f'''(x_{i+1}) = 2hf'(x_{i+1}) - \frac{2h^3}{3} f'''(x_{i+1}).$$

Приближение $f'(x_{i+1})$:

$$\hat{f}'_{i+1,c} = \frac{1}{2h}(f_{i-1} - 4f_i + 3f_{i+1}).$$

Приближение:

$$\frac{1}{2h}(f_{i-1} - 4f_i + 3f_{i+1}) = f'(x_{i+1}) - \frac{h^2}{3}f'''(x_{i+1}).$$

Остаточный член:

$$\text{остаток} = -\frac{h^2}{3}f'''(x_{i+1}).$$

Ошибка: $O(h^2)$.

Эта формула удобна, если нужно найти производную в крайней точке трёхточечного набора. Она точнее прямой разности, так как использует больше данных.

Остаток $-\frac{h^2}{3}f'''(x_{i+1})$ говорит, что ошибка зависит от третьей производной, как и в центральной разности, но коэффициенты немного отличаются из-за асимметрии.

Метод	Остаточный член	Погрешность	Порядок точности	Требуемая гладкость
Прямая разность	$-\frac{h}{2}f''(\xi)$	$\leq \frac{h}{2}M_{2,i}$	$O(h)$	$f \in C_2[a,b]$
Центральная разность	$-\frac{h^2}{6}f'''(\xi)$	$\leq \frac{h^2}{6}M_{3,i}$	$O(h^2)$	$f \in C_3[a,b]$

Требования:

Прямая разность: Вторая производная должна быть непрерывной.

Центральная разность: Третья производная должна быть непрерывной.

Замечание: Гладкость функции критически влияет на точность. Для функций с разрывами или резкими изменениями точность может снижаться.

Чем более гладкая функция (меньше резких скачков), тем лучше работают эти методы. Остаточные члены показывают, от чего зависит ошибка: от шага и производных.

Остаточные слагаемые: возникают из разложения Тейлора, когда мы отбрасываем высшие члены. Для прямой разности ошибка линейна (h), для центральной — квадратична (h^2).

Функция: $f(x) = x^2$, точка $x = 1$, шаг $h = 0.1$.

Точное значение: $f'(x) = 2x$, $f'(1) = 2$.

Прямая разность:

$$f'(1) \approx \frac{f(1.1) - f(1)}{0.1} = \frac{1.21 - 1}{0.1} = 2.1.$$

Ошибка: $|2.1 - 2| = 0.1$. Остаточный член: $-\frac{h}{2} f''(\xi) = -\frac{0.1}{2} \cdot 2 = -0.1$, что совпадает с ошибкой.

Центральная разность:

$$f'(1) \approx \frac{f(1.1) - f(0.9)}{2 \cdot 0.1} = \frac{1.21 - 0.81}{0.2} = 2.$$

Ошибка: $|2 - 2| = 0$. Остаточный член: $-\frac{h^2}{6} f'''(\xi) = 0$, так как $f'''(x) = 0$, что объясняет нулевую ошибку.

Вывод: Центральная разность точнее, особенно для гладких функций.

Для простой функции вроде x^2 центральная разность дала точный результат, потому что функция "слишком гладкая" (нет третьей производной). Прямая разность ошиблась из-за линейной ошибки.

Разностная задача аппроксимирует дифференциальную, если невязка стремится к нулю при $h \to 0$:

$$\| r_h \| \to 0 \quad \text{при} \quad h \to 0$$

Это значит, что схема становится всё точнее при уменьшении шага.

Порядок аппроксимации p: Если $\| r_h \| \leq Ch^p$, то p показывает, как быстро невязка уменьшается:

- $p = 1$: ошибка пропорциональна h (метод Эйлера),
- $p = 2$: ошибка пропорциональна h^2 (например, метод трапеций).

Пример

Для метода Эйлера невязка $r_h \approx \frac{h}{2} u''(t_n)$, что даёт порядок $p = 1$. Это значит, что при уменьшении h в 2 раза невязка уменьшается тоже в 2 раза.

Схема устойчива, если небольшие ошибки в начальных данных или вычислениях не разрастаются бесконтрольно. Формально, для двух решений u_h^1 и u_h^2 с возмущениями ε_h^1 и ε_h^2:

$$\| u_h^1 - u_h^2 \| \leq C_1 \left(\| \varepsilon_h^1 \| + \| \varepsilon_h^2 \| \right)$$

где C_1 — константа, не зависящая от h.

Пример

Для $\frac{du}{dt} = -u$ метод Эйлера даёт $u_{n+1} = u_n(1 - h)$. Условие устойчивости: $|1 - h| < 1$, то есть $h < 2$. Если $h > 2$, решение начнёт расти, хотя точное решение $u(t) = e^{-t}$ убывает.

Решение разностной задачи сходится к точному, если ошибка между ними исчезает при $h \to 0$:

$$\| u_h - [u]_h \| \to 0$$

Пример

Для метода Эйлера при $\frac{du}{dt} = -u$ ошибка уменьшается с h, но медленно, так как порядок сходимости — первый.

Порядок сходимости p показывает скорость уменьшения ошибки:

$$\| u_h - [u]_h \| \leq C_2 h^p$$

$p = 1$: метод Эйлера,

$p = 4$: метод Рунге-Кутта 4-го порядка.

Пример

Для $h = 0.1$ метод Эйлера даёт $u(1) \approx 0.3487$ (ошибка 0.0192), а для $h = 0.05$ — $u(1) \approx 0.3585$ (ошибка 0.0094). Ошибка уменьшается вдвое, подтверждая $p = 1$.

Устойчивость разностной схемы означает, что малые возмущения в начальных данных или правой части не приводят к значительным отклонениям в решении. Для ОДУ устойчивость анализируется через спектральный подход или тест-уравнение $y' = \lambda y$.

Анализ устойчивости метода Эйлера

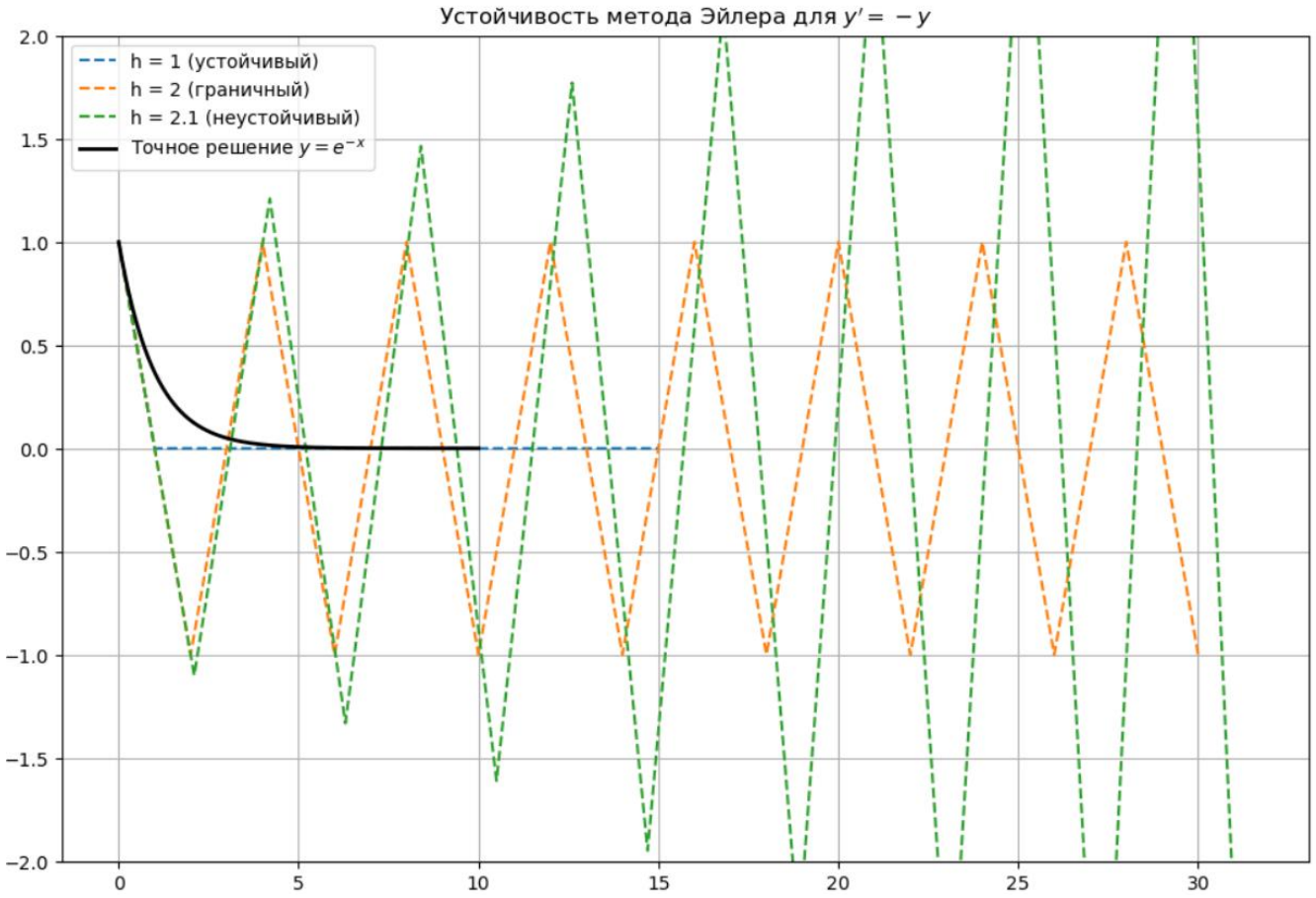
Для $y' = -y$:

$$y_{n+1} = y_n - hy_n = y_n(1 - h).$$

Схема устойчива, если $|1 - h| \leq 1$, т.е. $h \leq 2$. При $h > 2$ решение может экспоненциально расти, что некорректно.

Пример неустойчивости

Если $h = 2.1$, то $|1 - 2.1| = 1.1 > 1$, и значения y_n будут осциллировать с увеличивающейся амплитудой, что не соответствует $y(x) = e^{-x}$.



Теорема Лакса-Рябенького-Филипова

Утверждает, что разностная схема сходится к точному решению, если она:

Консистентна: аппроксимирует ОДУ с заданной точностью,

Устойчива: малые ошибки не усиливаются.

Для ОДУ это означает, что методы, такие как Эйлера или Рунге-Кутта, сходятся при правильном выборе шага h.

Дифференциальная задача	Разностная задача
$Lu = f \quad (1)$	$L_h u_h = f_h (2)$

Причем, если аппроксимация имела порядок p, то и сходимость будет иметь порядок p.

Жёсткие ОДУ — это уравнения, где численные методы требуют очень малого шага для устойчивости. Для них используются специальные методы, такие как неявные схемы.

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 $x_{i+1} = x_{i+h}$, $i = 0, 1, ..., n$, 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

$$r^n = \frac{u^{n+1}-u^n}{\tau} - f(t,u^n) =$$

$$= \frac{u^n + \tau u' + \frac{\tau^2}{2}u'' + O(\tau^3) - u^n}{\tau} - f(t,u^n) =$$

$$= u' + \frac{\tau}{2}u'' + O(\tau^2) - f(t,u^n) = \frac{\tau}{2}u'' + O(\tau^2)$$

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

$$u^{n\pm1}=u^n\pm\tau u'+\frac{\tau^2}{2}u''\pm\frac{\tau^3}{6}u''+O(\tau^4)$$

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

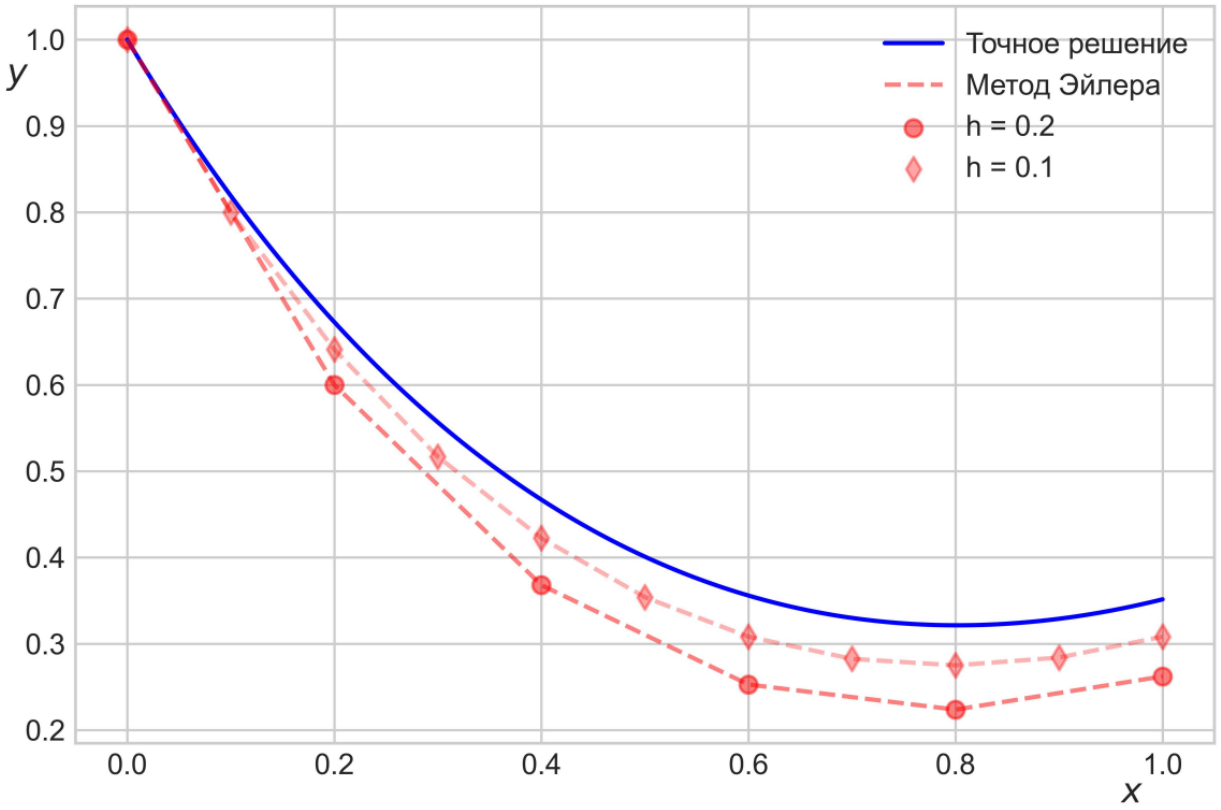
Метод Эйлера может быть представлен в виде последовательного применения формул:

$$x_1 = x_0 + h; \qquad y_1 = y_0 + hy_0' = y_0 + hf(x_0, y_0)$$
$$x_2 = x_1 + h; \qquad y_2 = y_1 + hy_1' = y_1 + hf(x_1, y_1)$$
$$\dots$$
$$x_i = x_{i-1} + h; \qquad y_i = y_{i-1} + y_{i-1}' = y_{i-1} + hf(x_{i-1}, y_{i-1})$$

В общем виде формула Эйлера записывается следующим образом

$$y_i = y_{i-1} + hf(x_{i-1}, y_{i-1}); \qquad x_i = x_{i-1} + h$$

Второе название – «метод ломаных» обусловлено графической интерпретацией данного метода. Искомая функция $y(x)$ заменяется ломаной линией с узлами в точках x_0 , x_1,..., 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: $x_0 = 0$; $y_0 = 2.7183$.

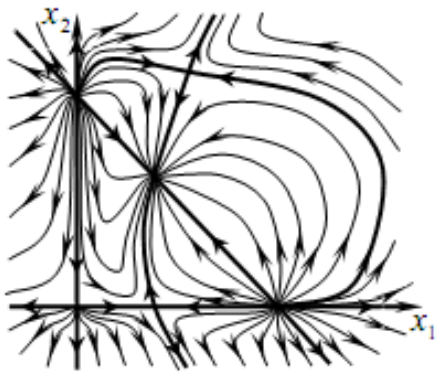
Построим таблицу значений переменной y_i при соответствующих значениях переменной x_i .

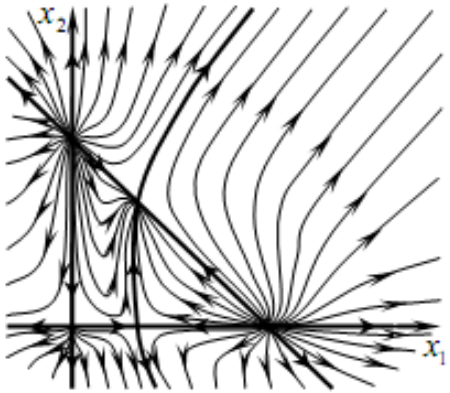
Построим таблицу значений переменной y_i при соответствующих значениях переменной x_i .

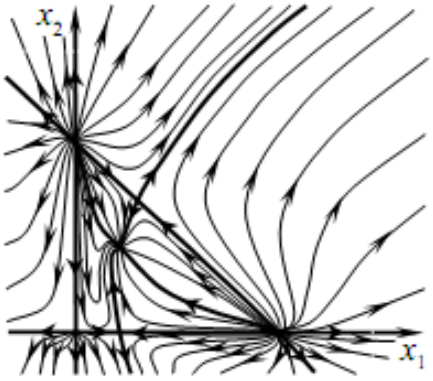
1. $y_1 = y_0 + h \cdot \frac{y_0}{(\cos(x_0))^2} = 2.7183 + 0.1 \cdot \frac{2.7183}{(\cos(0))^2} = 2.9901$
2. $y_2 = y_1 + h \cdot \frac{y_1}{(\cos(x_1))^2} = 2.9901 + 0.1 \cdot \frac{2.9901}{(\cos(0.1))^2} = 3.2922$
3. $y_3 = y_2 + h \cdot \frac{y_2}{(\cos(x_2))^2} = 3.2922 + 0.1 \cdot \frac{3.2922}{(\cos(0.2))^2} = 3.6349$
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. $y_{10} = y_9 + h \cdot \frac{y_9}{(\cos(x_9))^2} = 8.2497 + 0.1 \cdot \frac{8.2497}{(\cos(0.9))^2} = 10.3847$

Фазовый портрет – это графический способ представления всех возможных траекторий системы дифференциальных уравнений в фазовом пространстве. Он помогает понять поведение системы без необходимости решать уравнения аналитически.

- **Фазовое пространство**: пространство состояний системы. Для системы из двух уравнений это двумерная плоскость, где оси соответствуют переменным системы (например, (x) и (y)).
- **Траектории**: кривые, показывающие, как изменяется состояние системы со временем при разных начальных условиях.
- **Особые точки**: состояния, где производные равны нулю. Они определяют устойчивость и тип поведения системы.







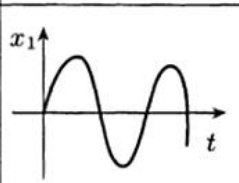
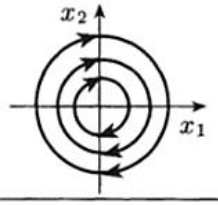
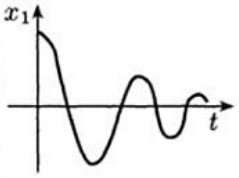
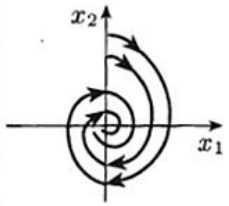
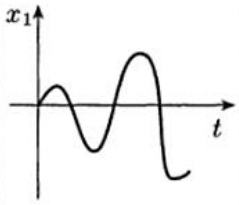
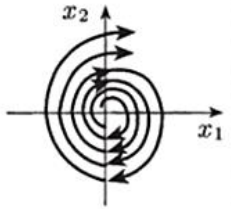
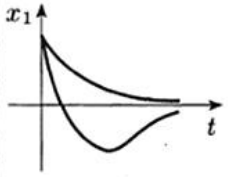
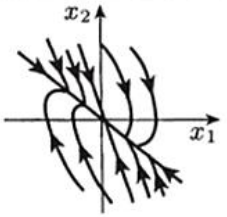
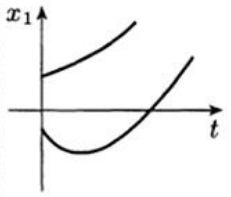
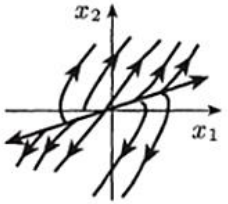
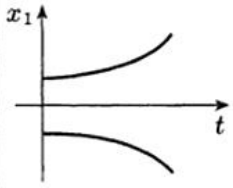
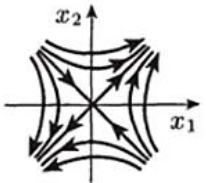
ФАЗОВЫЙ ПОРТРЕТ СИСТЕМ

Что показывает фазовый портрет?

- Эволюцию системы во времени.
- Устойчивые и неустойчивые состояния.
- Типы поведения: колебания, затухание, рост.

Типы особых точек

- **Узел**: траектории сходятся к точке или расходятся от неё.
- **Фокус**: траектории закручиваются в спираль (устойчивую или неустойчивую).
- **Седло**: траектории приближаются по одним направлениям и удаляются по другим.

Кривая переходного процесса	Фазовый портрет	Тип особой точки
		Центр
		Устойчивый фокус
		Неустойчивый фокус
		Устойчивый узел
		Неустойчивый узел
		Седло

Домашняя работа #6

Цель работы

- Реализовать численное решение системы обыкновенных дифференциальных уравнений (ОДУ), описывающей динамику инфекционного заболевания, с использованием метода Эйлера.
- Построить фазовые портреты в пространствах (F, V) и (C, V), а также график зависимости относительной характеристики поражения организма m(t) от времени.
- Провести анализ влияния начальных условий и параметров на поведение системы.

Срок сдачи работ: **до 5-го мая**.

Работы принимаются на электронный адрес VSChernyshenko@fa.ru в виде .html или .pdf файлов, название файла: “Фамилия_Имя_группа_ДЗ6.*”

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

Домашняя работа #6

Описание модели:

Будем считать, что основными действующими факторами инфекционного заболевания в момент времени t являются следующие величины :

1) концентрация патогенных размножающихся антигенов (вирусов) – $V(t)$;
2) концентрация антител $F(t)$. Под антителами понимаются субстраты иммунной системы, нейтрализующие антигены (иммуноглобулины, рецепторы клеток);
3) концентрация плазматических клеток $C(t)$. (Популяция иммунокомпетентных клеток, продуцирующих антитела);
4) относительная характеристика пораженного организма (доля поражённой ткани органа-мишени).

Домашняя работа #6

Первое уравнение описывает изменение числа антигенов (вирусов) в организме:

$$\frac{dV(t)}{dt} = \beta V(t) - \gamma F(t) V(t) \ ,$$

$\beta V(t)$ – экспоненциальный рост антигенов;

β – коэффициент размножения антигенов;

$\gamma F(t) V(t)$ – уменьшение антигенов за счёт нейтрализации антителами;

γ – коэффициент, связанный с вероятностью нейтрализации антигена антителами при встречи с ним.

Домашняя работа #6

Второе уравнение описывает рост плазматических клеток:

$$\frac{dC(t)}{dt} = \alpha F(t-\tau)V(t-\tau) - \mu_C(C(t) - C^*),$$

$\alpha F(t-\tau)V(t-\tau)$ – генерация плазматических клеток с учётом задержки τ;

τ – время, в течение которого осуществляется формирование каскада плазматических клеток;

α – коэффициент, учитывающий вероятность встречи антиген-антитело, возбуждение каскадной реакции и скорость образования новых клеток.

$-\mu_C(C(t) - C^*)$ – естественное возвращение к базовому уровню C^* за счет старения;

μ_C – коэффициент, равный обратной величине их времени жизни;

C^* – постоянный уровень плазматических клеток в здоровом организме

Домашняя работа #6

Третье уравнение описывает изменение числа антител:

$$\frac{dF(t)}{dt} = \rho\, C(t) - \mu_f F(t) - \eta\, \gamma\, V(t)\, F(t),$$

$\rho C(t)$ – производство антител плазматическими клетками.

ρ – коэффициент, обратно пропорциональный времени распада антител.

$\mu_f F(t)$ – естественный распад антител.

μ – скорость производства антител одной плазматической клеткой.

$\eta\gamma V(t)F(t)$ – потеря антител при связывании с антигенами.

Домашняя работа #6

Построенные уравнения не учитывают ослабление жизнедеятельности организма в ходе заболевания, связанного с ослаблением органов, обеспечивающих поставку лимфоцитов, антител и т. д., необходимых для борьбы с размножающимися антигенами. Примем гипотезу, что производительность органов связана с размерами поражения органа-мишени.

Пусть M – характеристика здорового органа (масса или площадь), а M'(t) – соответствующая характеристика здоровой части пораженного органа, m(t) = 1 - M'(t)/M – характеристика (относительная) поражения органа-мишени. Для здорового органа она равна нулю, а для полностью пораженного – 1.

Домашняя работа #6

Четвертое уравнение:

$$\frac{dm(t)}{dt} = \sigma V(t) - \mu_m m(t).$$

$\sigma V(t)$ – рост поражения органа пропорционально концентрации антигенов;

σ – некоторая постоянная, своя для каждого заболевания.

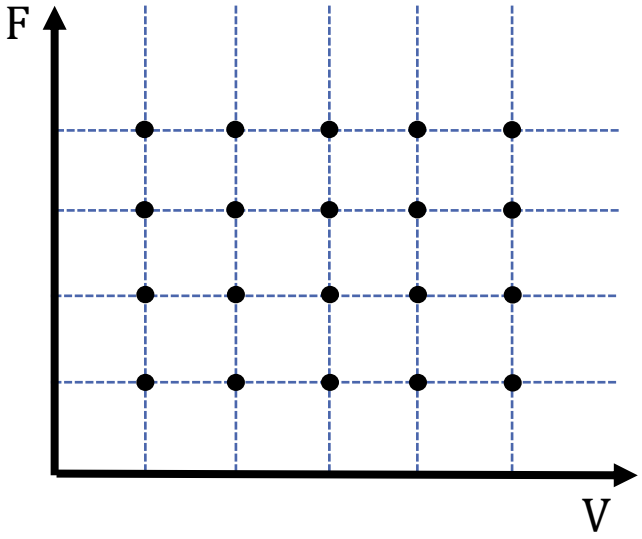
$-\mu_m m(t)$ – восстановление организма.

Пример значений коэффициентов (в своём исследовании выберите альтернативные):

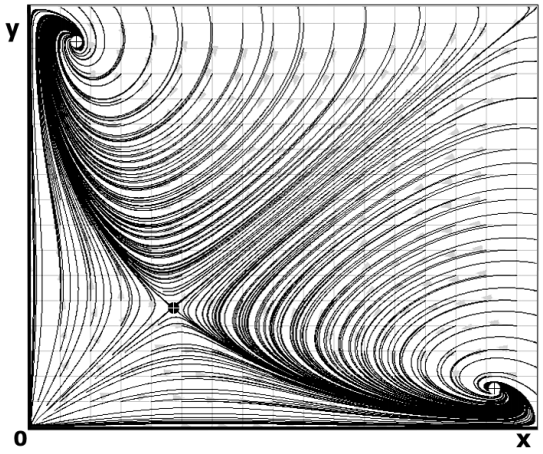
Параметр	β	γ	α	μ_C	ρ	η	μ_f	σ	μ_m	C^*	τ
Значения	1	0,8	1000	0,5	0,17	10	0,2	10	0,12	1	0,5
Размерность	сут $^{-1}$	$\frac{\text{мл}}{\text{част} \cdot \text{сут}}$	$\frac{\text{клет} \cdot \text{мл}}{\text{част} \cdot \text{молек} \cdot \text{сут}}$	сут $^{-1}$	$\frac{\text{молек}}{\text{клет} \cdot \text{сут}}$	$\frac{\text{молек}}{\text{част}}$	сут $^{-1}$	$\frac{\text{мл}}{\text{част} \cdot \text{сут}}$	сут $^{-1}$	$\frac{\text{клет}}{\text{мл}}$	сут

Домашняя работа #6

Для построения фазового портрета требуется построить достаточно большое количество изолиний. Наиболее естественным вариантом будет брать разные сочетания исходных значений F / C / V на сетке:



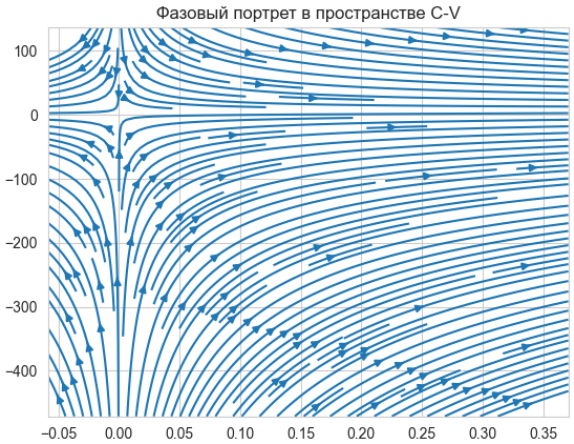
“Выходя” из каждого узла сетки (в цикле) можно будет получить график подобного вида:

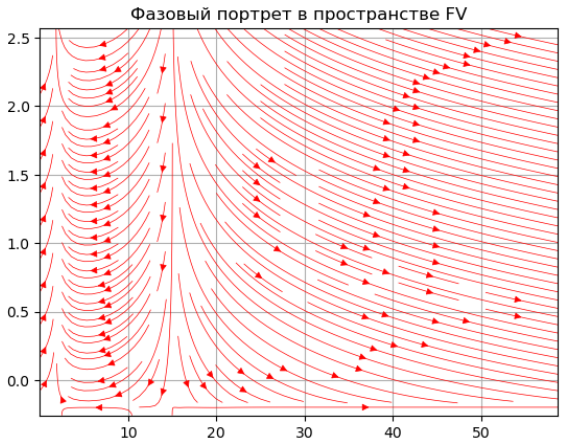


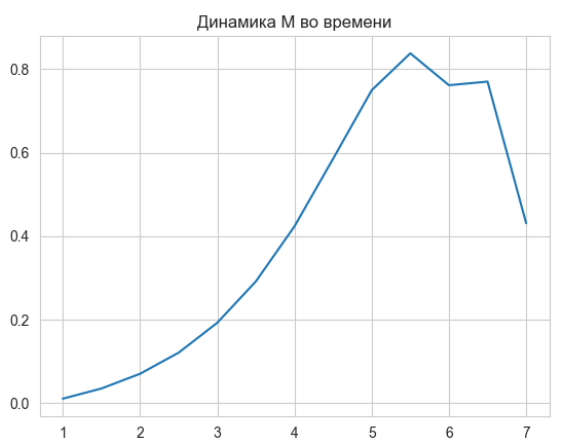
Домашняя работа #6

Задание

1. **Реализуйте метод Эйлера**:
   - Численно решите систему ОДУ с шагом h.
   - Учитывайте задержку τ путём хранения истории значений $F(t)$ и $V(t)$ за интервал $[t-\tau, t]$.
2. **Постройте фазовые портреты**:
   - В пространстве (F, V): траектории зависимости антител F от антигенов V.
   - В пространстве (C, V): траектории зависимости плазматических клеток C от антигенов V.
   - Используйте достаточно плотную сетку начальных условий.
3. **График динамики $m(t)$**:
   - Постройте зависимость $m(t)$ от времени t в интервале $[0, T]$.
4. **Анализ результатов**:
   - Определите сценарии: выздоровление ($V \to 0$, $m \to 0$), хроническое течение (V стабильно), летальный исход ($m \to 1$).







Домашняя работа #6

В рамках отчёта в конце ноутбука ответьте на следующие вопросы:

1. Как фазовый портрет помогает интерпретировать устойчивость и поведение системы?

2. Как задержка τ влияет на динамику плазматических клеток $C(t)$? Объясните возможные колебания в системе.

3. В чём суть метода Эйлера? Как шаг h влияет на точность решения, и какие недостатки могут возникнуть при большом h?

4. Почему учет задержки τ важен для моделирования иммунного ответа? Приведите пример.

5. Как начальные условия $V(0)$ и $F(0)$ влияют на траектории фазового портрета? Объясните в контексте поставленной задачи.

Примеры имитационных моделей, описываемых системами дифференциальных уравнений

Модель хищник-жертва Лотки-Вольтерра

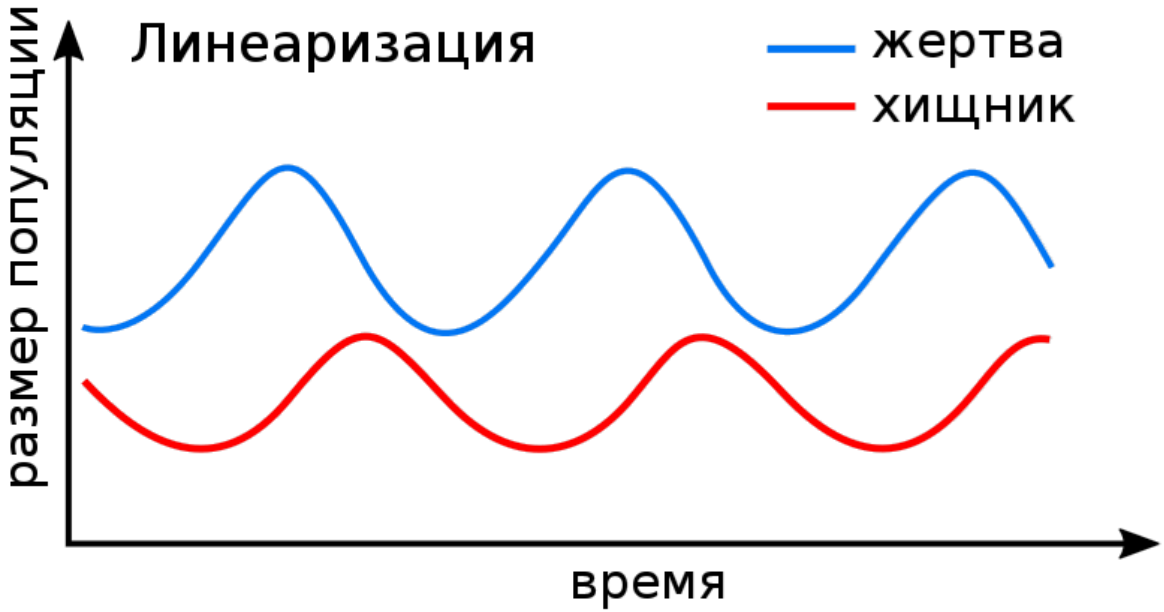
Модель имеет следующую форму:

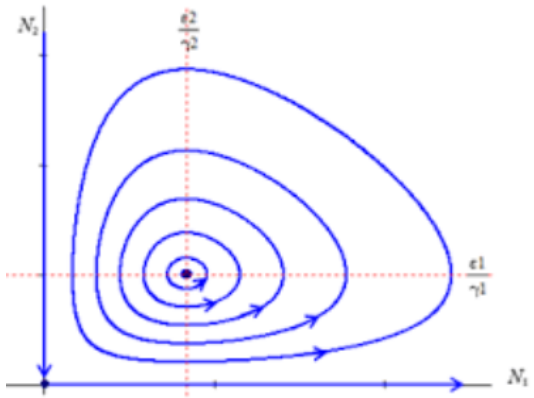
$$x' = a\,x - b\,x^2 - c\,xy$$
$$y' = -d\,y + e\,xy$$

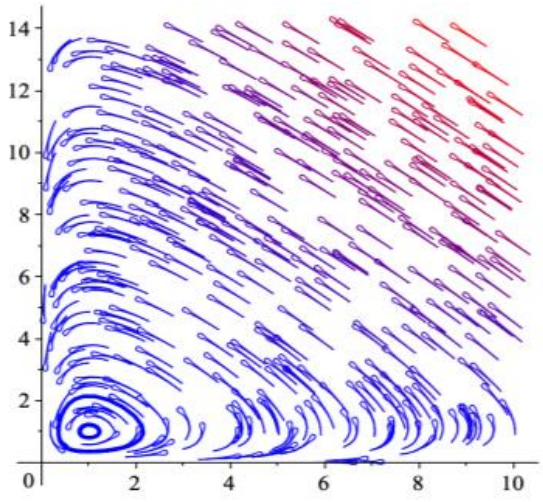
Здесь x - это размер популяции жертв ("кролики"), а y - размер популяции хищников ("лисы").

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.

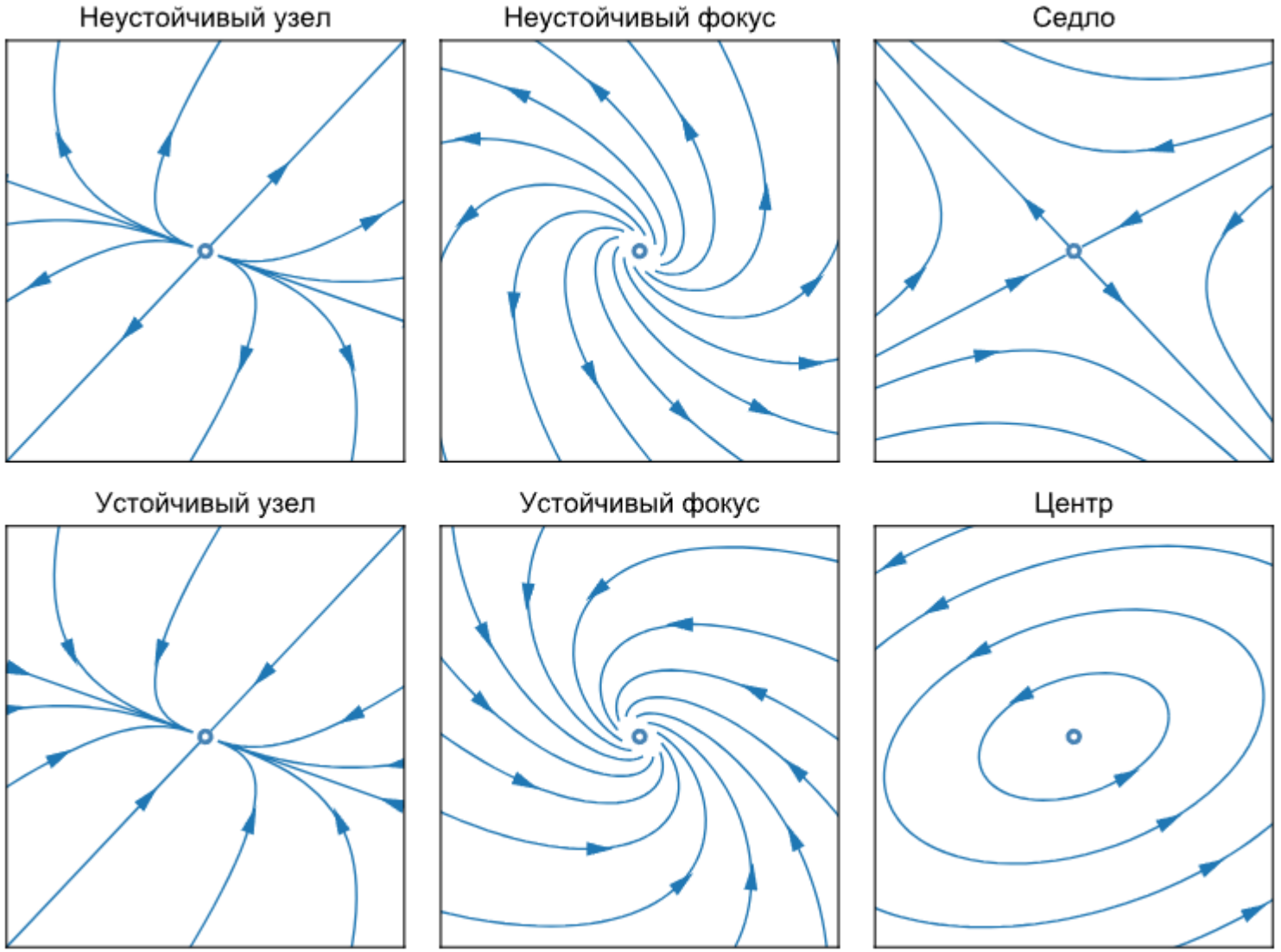
Модель хищник-жертва Лотки-Вольтерра



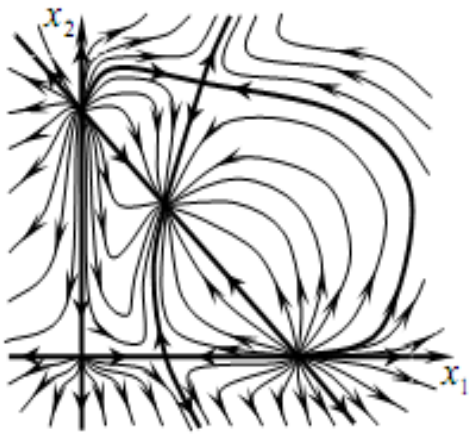


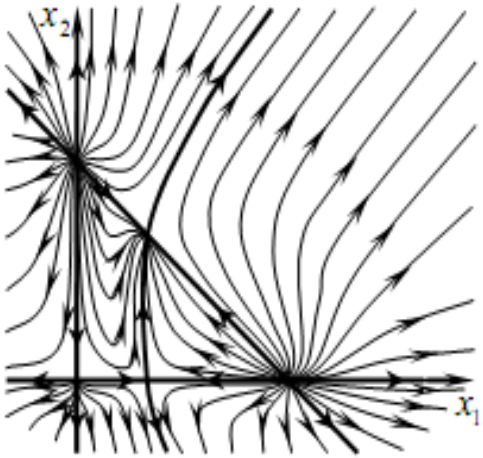


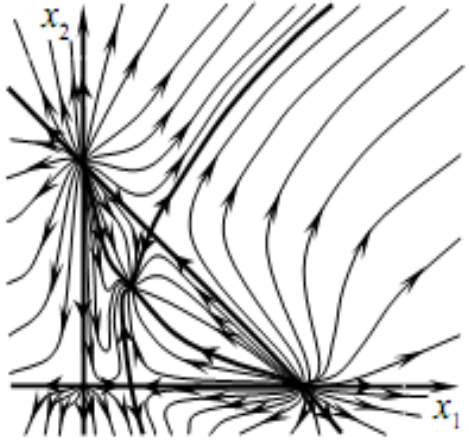
Точки устойчивости

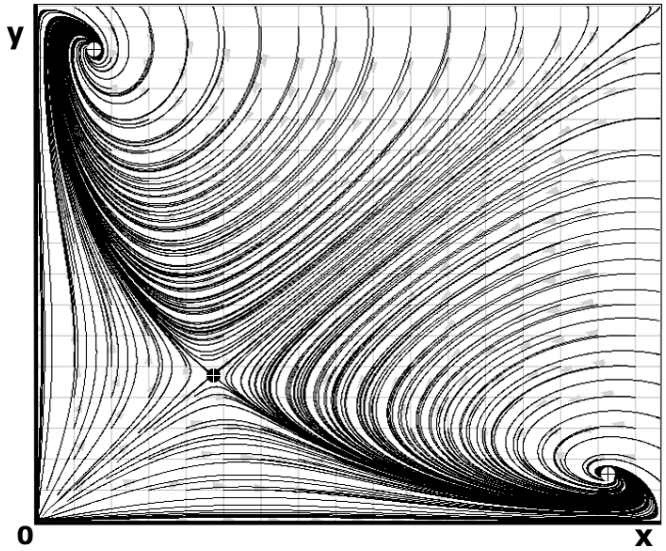


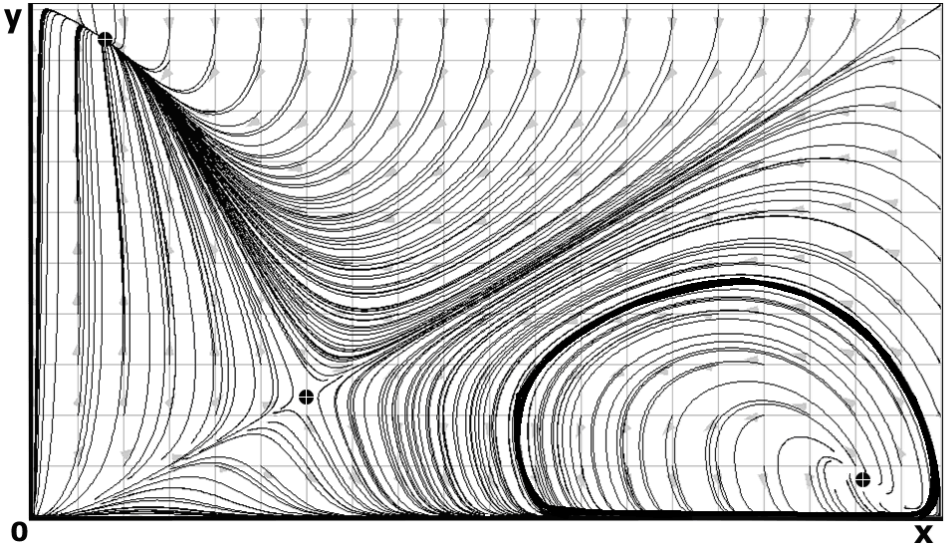
Фазовый портрет



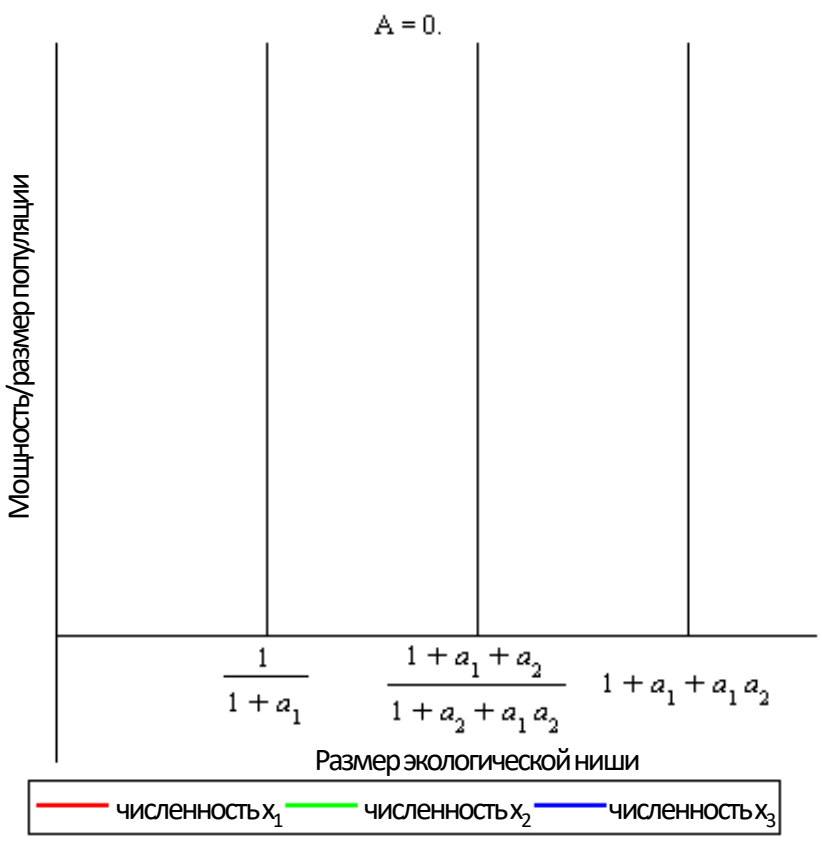


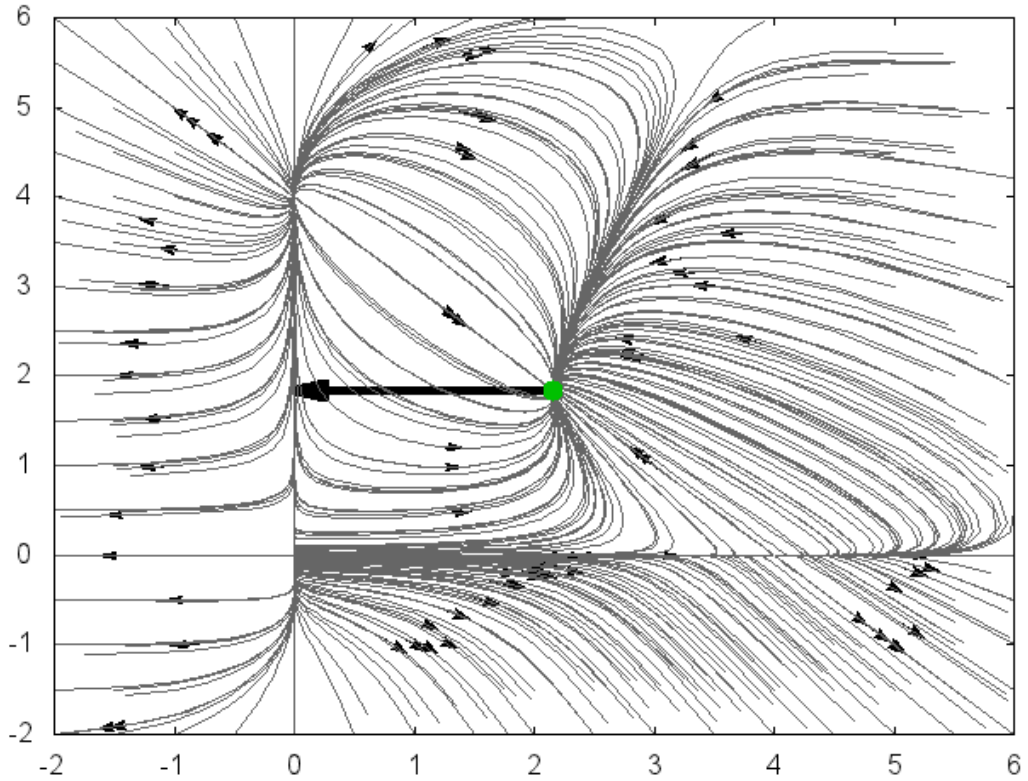






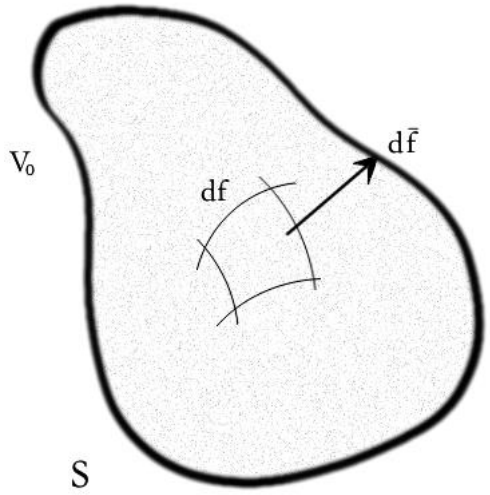
Фазовый портрет





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:

$$\frac{\partial}{\partial t}(\rho v_i) = -\frac{\partial \Pi_{ik}}{\partial x_k}\ , \qquad \Pi_{ik} = p\delta_{ik} + \rho v_i v_k$$

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:

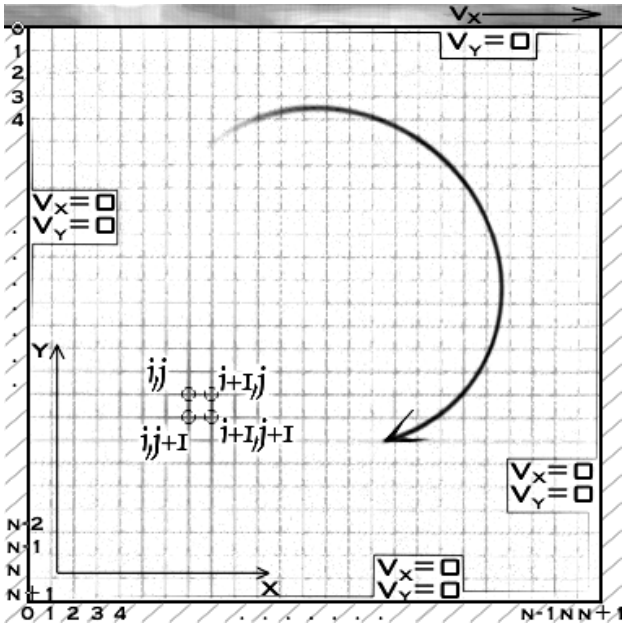
$$\begin{cases} \dfrac{\partial P}{\partial t} + \dfrac{\partial u}{\partial x} + \dfrac{\partial v}{\partial y} = 0; \\ \dfrac{\partial u}{\partial t} + u\dfrac{\partial u}{\partial x} + v\dfrac{\partial u}{\partial y} = -\dfrac{1}{\hat{\rho}} \cdot \dfrac{\partial P}{\partial x} + \nu\left(\dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2}\right); \\ \dfrac{\partial v}{\partial t} + u\dfrac{\partial v}{\partial x} + v\dfrac{\partial v}{\partial y} = -\dfrac{1}{\hat{\rho}} \cdot \dfrac{\partial P}{\partial y} + \nu\left(\dfrac{\partial^2 v}{\partial x^2} + \dfrac{\partial^2 v}{\partial y^2}\right); \end{cases}$$

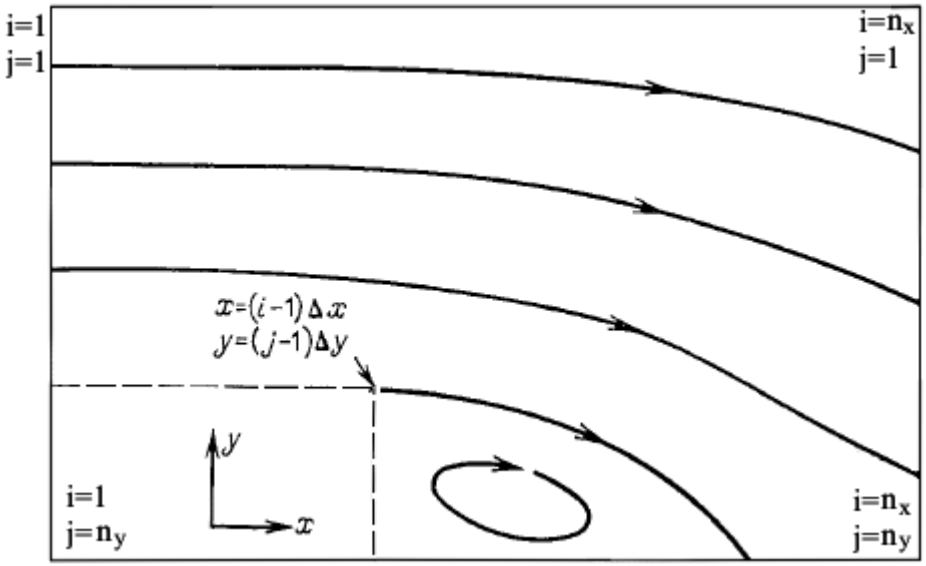
Где $\hat{\rho} = \mathrm{Re}$ и $\nu = \dfrac{1}{\rho}$

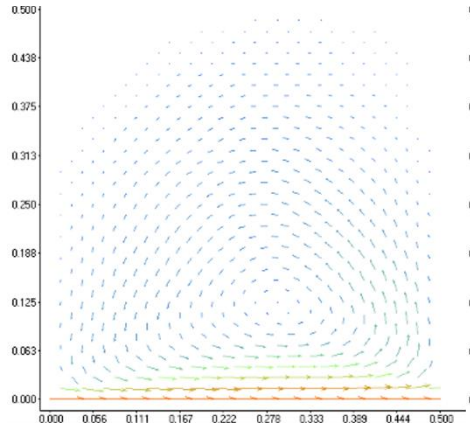
Алгоритм метода сеток для консервативного вида уравнения Навье-Стокса (в декартовых координатах)

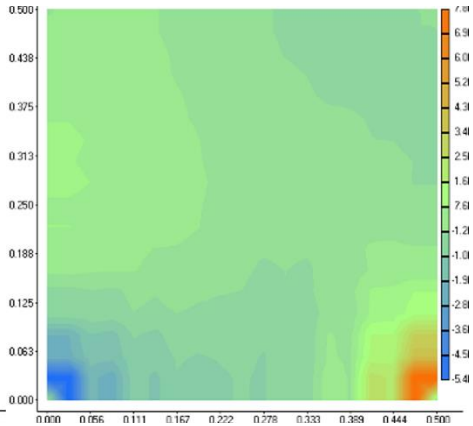
$$
\begin{cases}
P(t+1,i,j) = -\Delta t\left(\dfrac{U(t,i+1,j)-U(t,i-1,j)}{2\cdot\Delta x}+\dfrac{V(t,i,j+1)-V(t,i,j-1)}{2\cdot\Delta y}\right)+P(t,i,j); \\
U(t+1,i,j) = -\dfrac{\Delta t}{\mathrm{Re}}\cdot\dfrac{P(t,i+1,j)-P(t,i-1,j)}{2\cdot\Delta x}+ \\
\quad +\dfrac{\Delta t}{\rho}\left(\dfrac{U(t,i+1,j)-2\cdot U(t,i,j)+U(t,i-1,j)}{\Delta x^2}+\dfrac{U(t,i,j+1)-U(t,i,j)+U(t,i,j-1)}{\Delta y^2}\right)- \\
\quad -\Delta t\cdot\left(\dfrac{\partial(U^2)}{\partial x}-\dfrac{\partial(V\cdot U)}{\partial y}\right)+U(t,i,j); \\
V(t+1,i,j) = -\dfrac{\Delta t}{\mathrm{Re}}\cdot\dfrac{P(t,i,j+1)-P(t,i,j-1)}{2\cdot\Delta y}+ \\
\quad +\dfrac{\Delta t}{\rho}\left(\dfrac{V(t,i+1,j)-2\cdot V(t,i,j)+V(t,i-1,j)}{\Delta x^2}+\dfrac{V(t,i,j+1)-V(t,i,j)+V(t,i,j-1)}{\Delta y^2}\right)- \\
\quad -\Delta t\cdot\left(\dfrac{\partial(U\cdot V)}{\partial x}+\dfrac{\partial(V^2)}{\partial y}\right)+V(t,i,j);
\end{cases}
$$

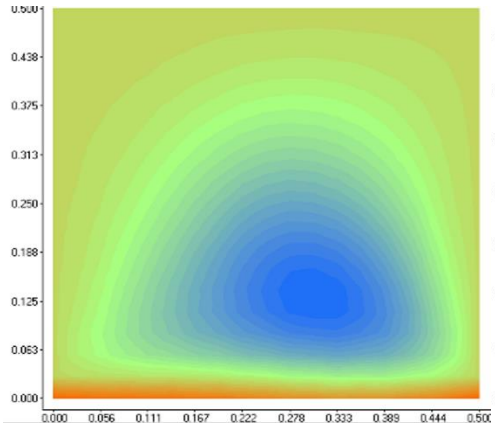
Фазовый портрет

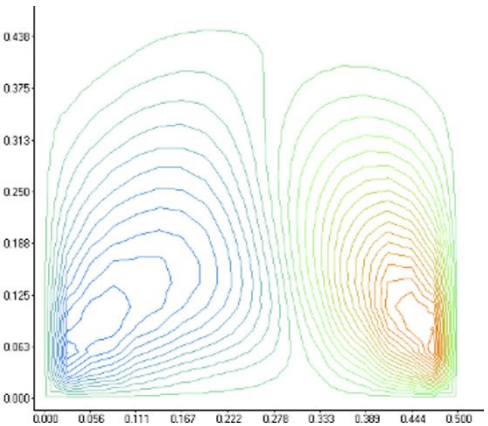












Винеровский процесс

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

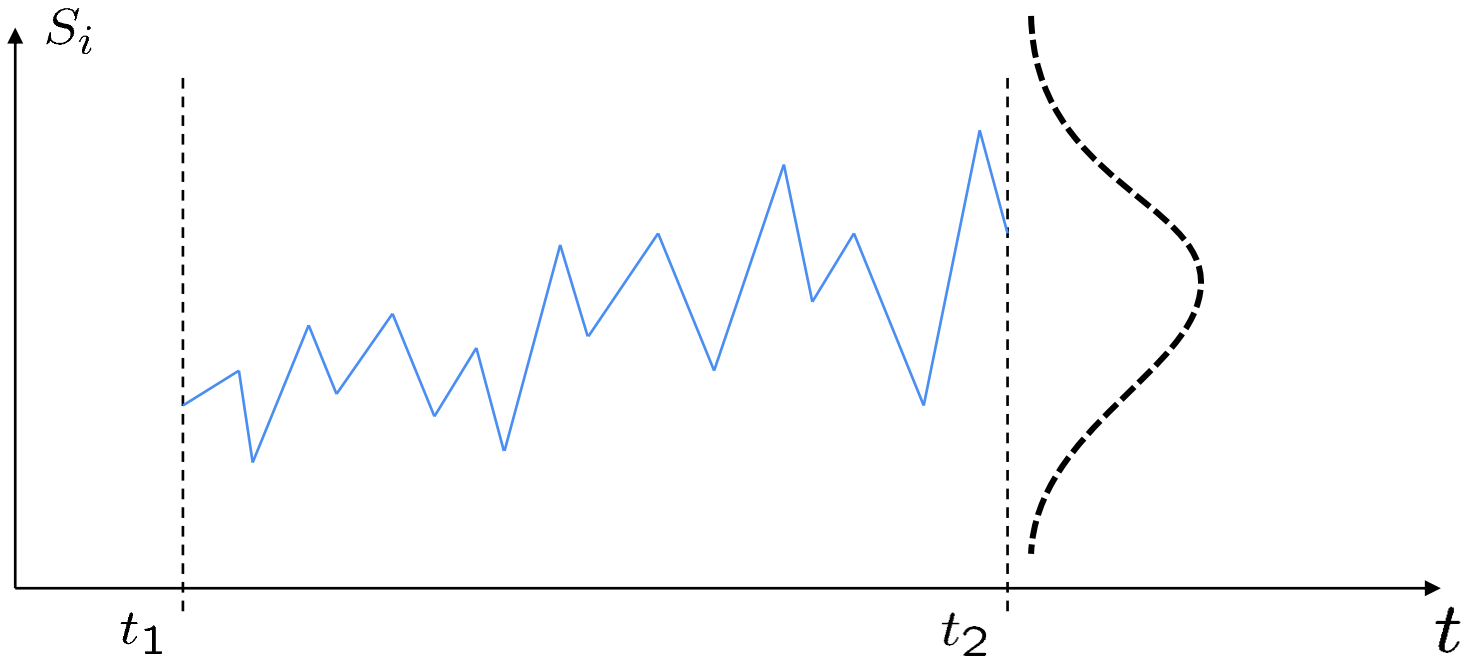
Пусть S – рыночная цена фондового актива, а t – период времени. За малый промежуток времени Δt случайная переменная S изменится на ΔS. Если S следует процессу Винера, т.е. броуновскому движению, изменение S за малый промежуток времени будет связано с Δt следующим соотношением:

$$\Delta S = \varepsilon \times \sqrt{\Delta t} \text{ или}$$

$$dS = \varepsilon \times \sqrt{\Delta t}$$

Геометрическое броуновское движение

Цену акции будем рассматривать как стохастический процесс.



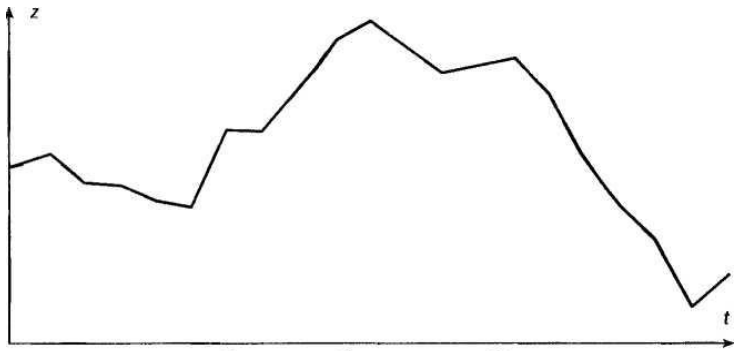
Основное предположение: **доходности распределены нормально**

Броуновское движение

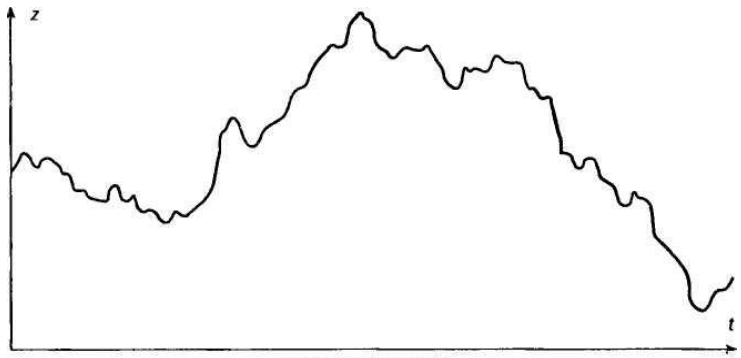
Стохастический процесс $\{W_t : 0 \leq t \leq \infty\}$ является стандартным броуновским движением, если:

1. $W_0 = 0$.
2. Представляет собой непрерывную случайную последовательность.
3. Имеет независимые, нормально распределенные приращения.

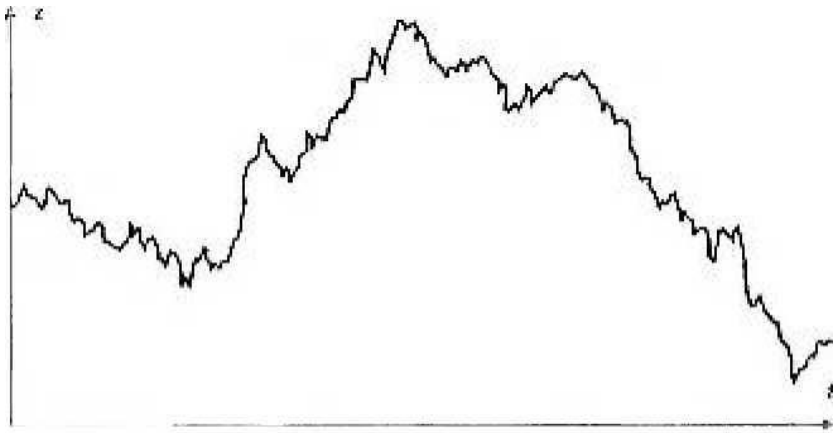
Геометрическое броуновское движение



Относительно большое значение Δt



Меньшее значение Δt



Истинный процесс при $\Delta t \to 0$

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 Δt должно быть пропорциональным самой цене акции. Это приводит к следующей модели:

$$\frac{\Delta S}{S} = \mu \Delta t + \sigma \varepsilon \sqrt{\Delta t}$$

или

$$\Delta S = \mu S \Delta t + \sigma S \varepsilon \sqrt{\Delta t}$$

ΔS – изменение цены акций

μ – ожидаемая доходность

σ – волатильность акций

$\sigma \varepsilon \sqrt{\Delta t}$ – стохастическая компонента с дисперсией, равной $\sigma^2 T$

ε – случайная величина, имеющая стандартизированное нормальное распределение.

Винеровский процесс

Изменение цены акции	=	Ожидаемый рост с течением времени	+	Влияние волатильности поведения людей, “хаотически” приобретающих и продающих акции, что оказывает влияние на динамику цен
Изменение цены акции	=	Скорость детерминированного тренда	+	Стохастическая переменная
Изменение цены акции	=	Определенность (тренд)	+	Неопределенность (обусловленная 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 Ω.

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 Δt должно быть пропорциональным самой цене акции. Это приводит к следующей модели:

$$\frac{\Delta S}{S} = \mu \Delta t + \sigma \varepsilon \sqrt{\Delta t}$$

или

$$\Delta S = \mu S \Delta t + \sigma S \varepsilon \sqrt{\Delta t}$$

ΔS – изменение цены акций

μ – ожидаемая доходность

σ – волатильность акций

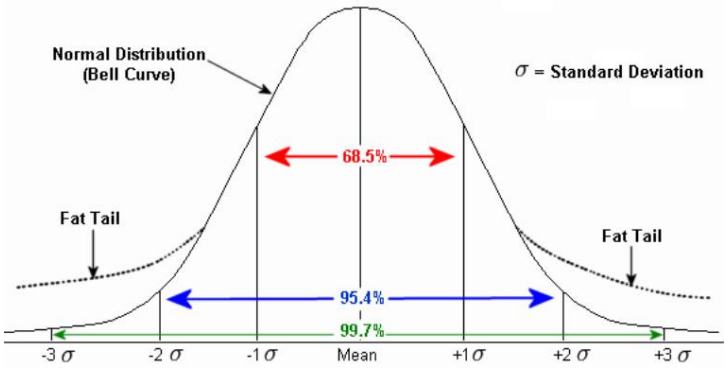
$\sigma \varepsilon \sqrt{\Delta t}$ – стохастическая компонента с дисперсией, равной $\sigma^2 T$

ε – случайная величина, имеющая стандартизированное нормальное распределение.

Винеровский процесс

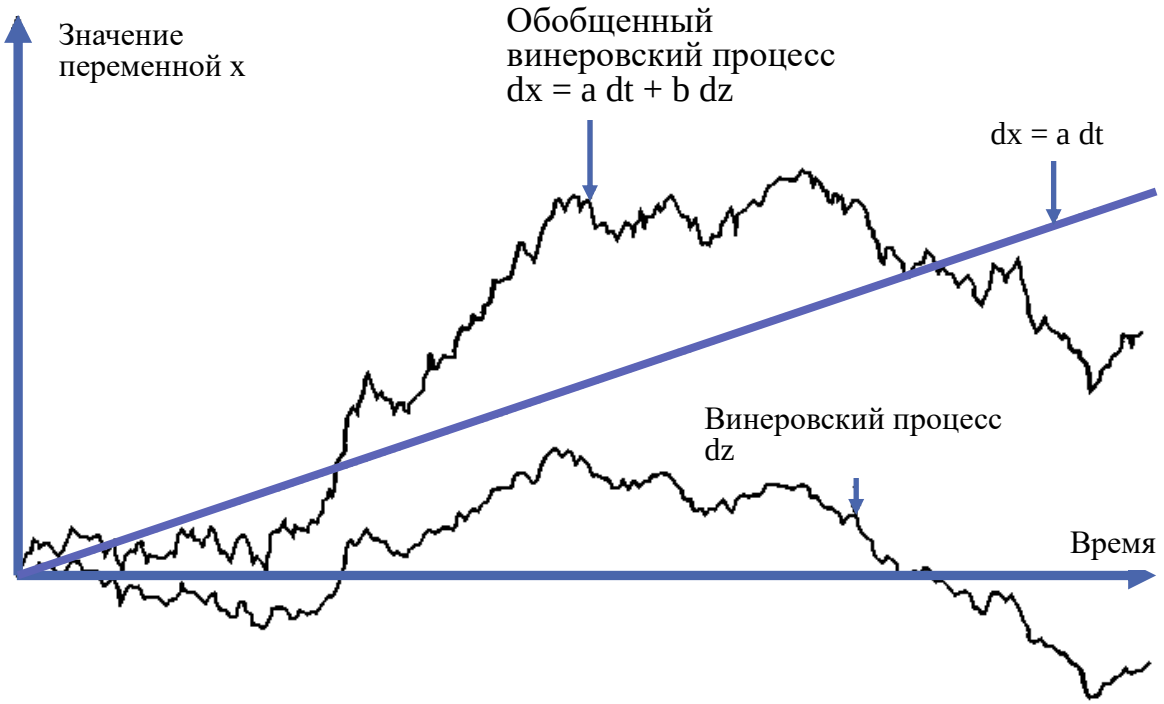
$$\text{Будущая цена акции} = \underbrace{Se^{\left(r-\frac{\sigma^2}{2}\right)t}}_{\text{Детерминированный тренд}} + \underbrace{\sigma W_t}_{\text{Стохастическая переменная}}$$

S – текущая цена акции
σ – процентная волатильность
r – безрисковая процентная ставка
W_t – Броуновское движение



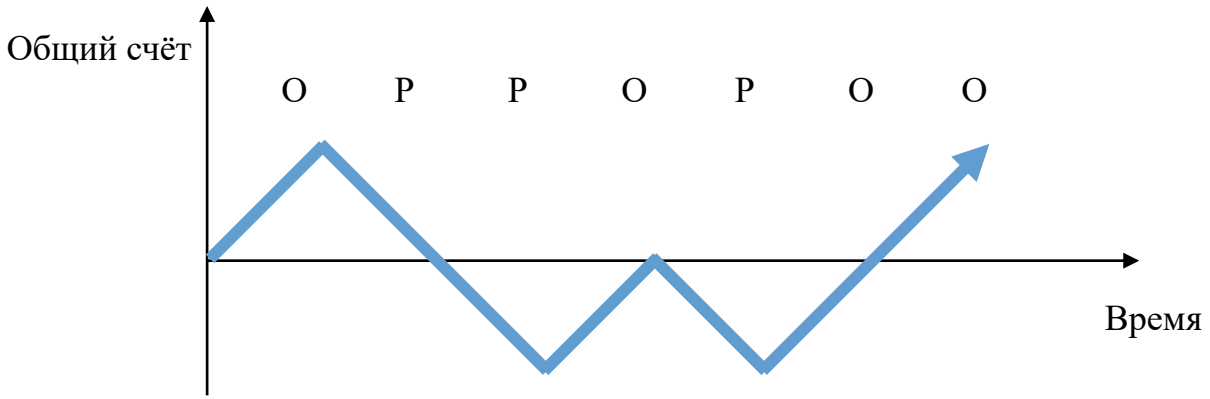
$$\text{Среднее} = r - \frac{\sigma}{2}$$

Винеровский процесс



Подбрасывание монеты

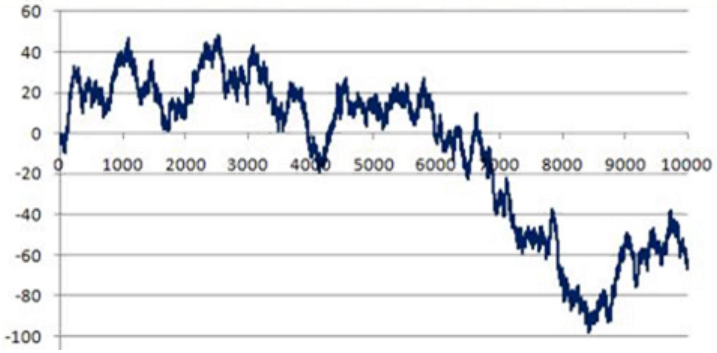
- Подбрасываем монету 1000 раз, прибавляя 1 всякий раз, когда выпадает «орёл», и вычитая 1 всякий раз, когда выпадает «решка», продолжая следить за общим счётом.
- Вопрос: Какой конечный результат мы можем ожидать? Сколько раз общий счёт будет обнулён после 1000 подбрасываний?

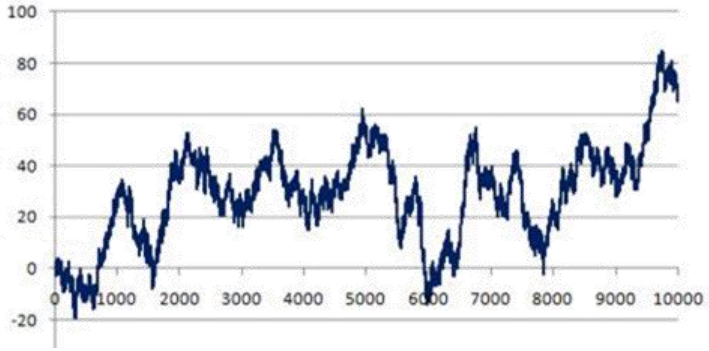


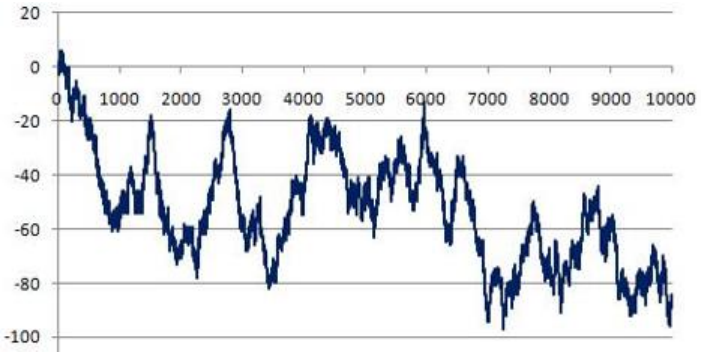
Подбрасывание монеты

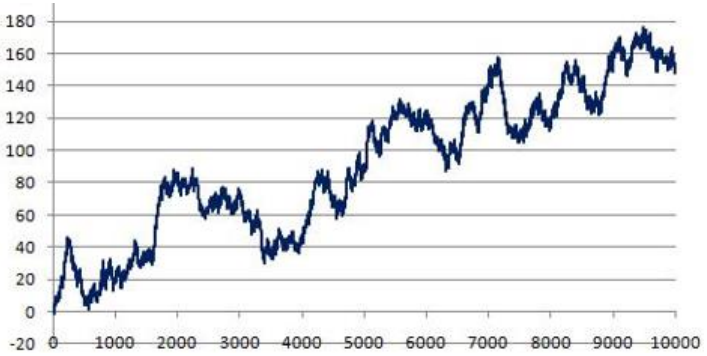
Количество "обнулений" в первой половине: 38
Количество "обнулений" во второй половине: 0

Подбрасывание монеты

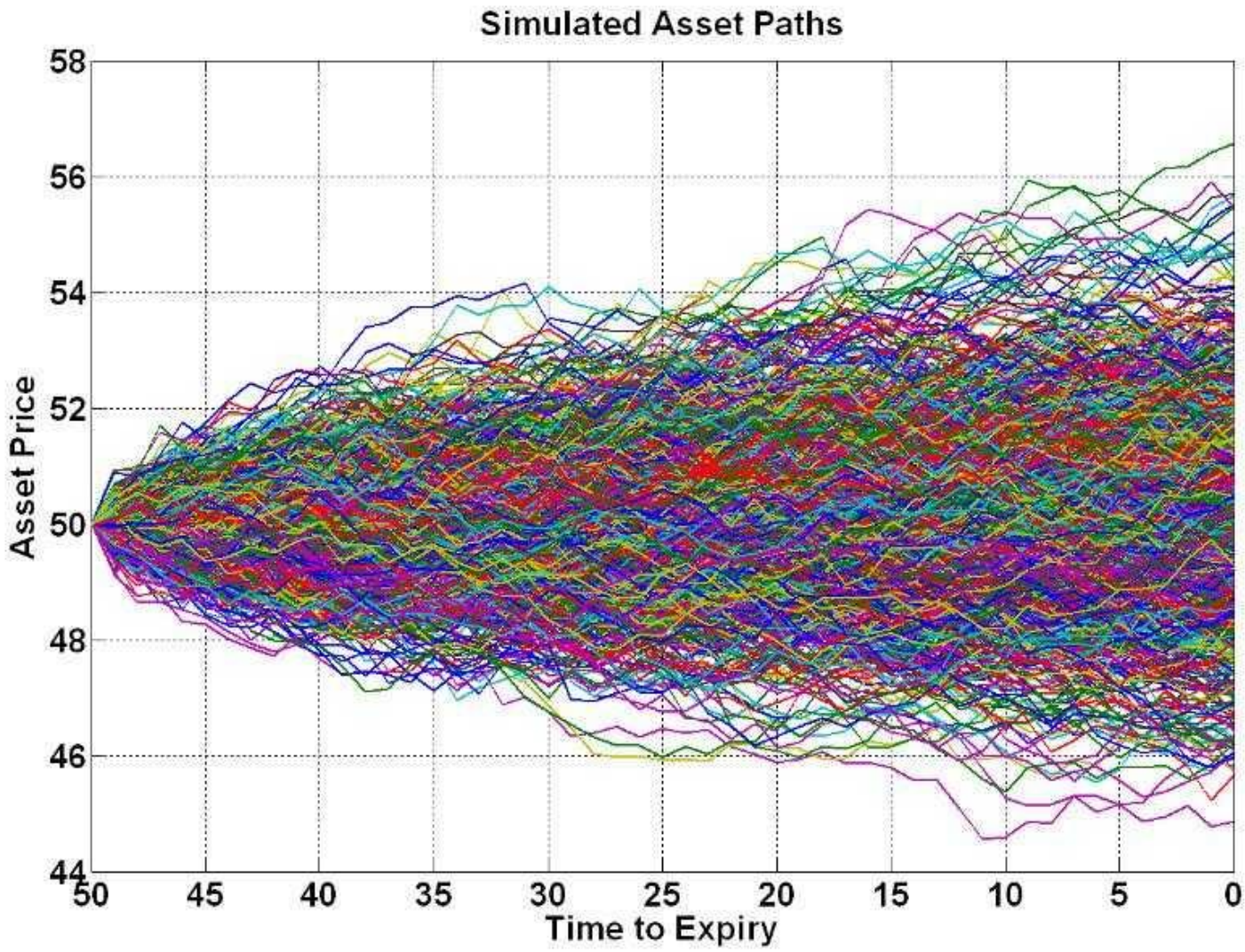








Метод статистических испытаний (Монте-Карло)



Моделирование спот-ставок

$$r(t) = f_{t,t} = f(0,t) + \int_o^t \mu(v,t)dv + \int_o^t \sigma_f(v,t)dW(v)$$

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
δ – 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 расположенности

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.

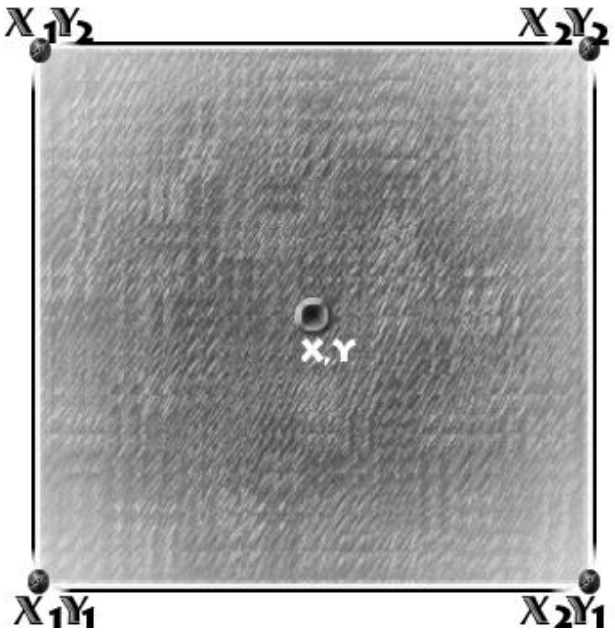
Модель конкуренции с учётом территориальной расположенности

$$\begin{cases} V_t' = b_V * V - d_V * V^2 - c_U * U * V + \int_{y-Dy_1}^{y+Dy_1} \int_{x-Dx_1}^{x+Dx_1} K_U(x,y) * V(x,y,t)dxdy \\ U_t' = b_U * U - d_U * U^2 - c_V * U * V + \int_{y-Dy_2}^{y+Dy_2} \int_{x-Dx_2}^{x+Dx_2} K_V(x,y) * U(x,y,t)dxdy \end{cases}$$

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 расположенности

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 y и Dx_1 по x 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 расположенности

$$\begin{cases} V_t' = b_V * V - d_V * V^2 - c_U * U * V + \int_{y-Dy_1}^{y+Dy_1} \int_{x-Dx_1}^{x+Dx_1} K_U(x,y) * V(x,y,t) dxdy \\ U_t' = b_U * U - d_U * U^2 - c_V * U * V + \int_{y-Dy_2}^{y+Dy_2} \int_{x-Dx_2}^{x+Dx_2} K_V(x,y) * U(x,y,t) dxdy \end{cases}$$

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

$$\sum_{k=1}^{N+1} a_{n+k} y^{n+k} = b_n f^n, \qquad a_{n+k} \quad , b_n = \text{const}$$

Если $b_n = 0$, то уравнение называется **однородным**

Решение разностного уравнения представимо в виде:

$$y^n = y^n_{\text{общ}} + y^n_{\text{ч}}$$

Общее решение однородного уравнения

Частное решение неоднородного уравнения

Общее решение состоит из линейной комбинации решений разностного уравнения:

$$y^n_{\text{общ}} = \sum_{i=1}^{N} C_j y^n_{\text{o}j}$$

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

$$y_{\text{общ}}^{n} = \sum_{i=1}^{N} C_j \lambda_j^n \qquad , n = 0 \ldots N$$

Проверим найденное решение

$$\sum_{j=1}^{N} C_j \lambda_j^{n-k}\left(a_n + a_{n+1}\lambda_j + \cdots + a_{n+k}{\lambda_j}^k\right) = 0$$

Верно $\forall C_j$

Если какой-нибудь корень λ_j имеет кратность s, то соответствующая частичная сумма

$$\sum_{p=0}^{s-1} C_{j+p}\lambda_{j+p}^n$$

заменяется выражением

$$(C_j + nC_{j+1} + \cdots + n^{s-1}C_{j+s-1})\lambda_j^n$$

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 α и β соответственно μ и ν – действительные числа

$$y_{\text{ч}}^n = \mu^n(Q_\delta(n)\cos(\nu n) + S_\delta(n)\sin(\nu n))$$

$Q_\delta(n), S_\delta(n)$ - 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 μ – модуль, а ν – аргумент какого-то одного корня характеристического уравнения полагаем

$$y_{\text{ч}}^n = n^r \, (Q_\delta(n)\cos(\nu n) + S_\delta(n)\sin(\nu n))$$

r – кратность корня

Найти решение разностного уравнение

$$y^{n+1} - 3y^n + 3y^{n-1} - y^{n-2} = ah^4 n,$$
$$a, h = \text{const}$$

Найдем общее решение однородного уравнения

Подставляем в уравнение $y^n_{\text{общ}} = \lambda^n$

Получаем характеристическое уравнение

$$\lambda^3 - 3\lambda^2 + 3\lambda - 1 = 0$$

$\lambda_{123} = 1$ - Корень кратности 3

$$y^n_{\text{общ}} = (C_1 + C_2 n + C_3 n^2)1^n$$

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 λ

Будем искать решение в виде

$$y_{\text{ч}}^n = (A + Bn)n^3$$

Подставляем в неоднородное уравнение

$$A(n+1)^4+B(n+1)^3-3An^4-3Bn^3+$$

$$+3A(n-1)^4+3B(n-1)^3-A(n-2)^4-$$

$$-B(n-2)^3=12A(2n-1)+6B=ah^4n$$

$$A=ah^4/24 \qquad B=ah^4/12$$

$$y^n=y^n_{\text{общ}}+y^n_{\text{ч}}=$$

$$=(C_1+C_2n+C_3n^2)1^n+ah^4n^3(n+2)/24$$

ОДУ не обязательно должно быть записано в виде уравнения первого порядка, например

$$z'' + z' + 1 = 0, \qquad z(0) = a, \;\; z'(0) = b.$$

Можно преобразовать задачу с использованием вспомогательных переменных, таких как $w = z'$. В итоге:

$$\begin{pmatrix} w \\ z \end{pmatrix}' = \begin{pmatrix} -w - 1 \\ w \end{pmatrix}, \quad z(0) = a, \; w(0) = b$$

Не всегда стоит использовать простые производные в качестве вспомогательных переменных. Пример:

$$\tfrac{1}{r^2}\left(r^2 P'/\rho\right)' = -4\pi\rho$$

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:

$$\begin{aligned} f(x_0+h) &= f(x_0) + hf'(x_0) + \tfrac{h^2}{2!} f''(x_0) + \mathcal{O}(h^3) \\ \Rightarrow \quad \frac{f(x_0+h) - f(x_0)}{h} &= \frac{f(x_0) + hf'(x_0) + \frac{h^2}{2!} f''(x_0) + \mathcal{O}(h^3) - f(x_0)}{h} \\ &= f'(x_0) + \mathcal{O}(h). \end{aligned}$$

Тот же результат по точности справедлив для

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

Задачи, в которых необходимо решить систему из нескольких дифференциальных уравнений с несколькими искомыми функциями, очень распространены в моделировании систем.

Будем рассматривать системы, в которых число неизвестных функций совпадает с числом уравнений, разрешенных относительно производных.

К примеру, система из двух уравнений с двумя неизвестными функциями y и z от одного и того же аргумента x имеет вид:

$$\begin{cases} y' = f_1(x, y, z) \\ z' = f_2(x, y, z) \end{cases}$$

при этом штрих означает производную по x.

Общий вид системы из n уравнений с n неизвестными функциями x_1, x_2, . . . , x_n от переменной t имеет вид:

$$\begin{cases} \dfrac{dx_1}{dt} = f_1(t, x_1, x_2, \ldots, x_n) \\ \dfrac{dx_2}{dt} = f_2(t, x_1, x_2, \ldots, x_n) \\ \ldots \\ \dfrac{dx_n}{dt} = f_n(t, x_1, x_2, \ldots, x_n) \end{cases}$$

Ранее мы рассмотрели численные методы решения обыкновенных дифференциальных уравнений вида $y' = f(x, y)$ (методы Эйлера и Рунге-Кутты). Данные методы применяются и в случае решения систем обыкновенных дифференциальных уравнений.

Пусть дана следующая система обыкновенных дифференциальных уравнений:

$$\begin{cases} \dfrac{dy_1}{dx} = f_1\left(x, y_1, y_2\right) \\ \dfrac{dy_2}{dx} = f_2\left(x, y_1, y_2\right) \end{cases}$$

с начальными условиями:

$$y_1\big|_{x=x_0} = y_{01}$$
$$y_2\big|_{x=x_0} = y_{02}$$

При использовании метода Эйлера, расчетные формулы примут следующий вид:

$$\begin{cases} y_{(i),1} = y_{i-1,1} + h \cdot f_1\left(x_{i-1}, y_{(i-1),1}, y_{(i-1),2}\right) \\ y_{(i),2} = y_{i-1,2} + h \cdot f_2\left(x_{i-1}, y_{(i-1),1}, y_{(i-1),2}\right) \\ x_i = x_{i-1} + h \end{cases}$$

где h – шаг интегрирования; $f_1(x_i, y_{i,1}, y_{i,2})$ и $f_2(x_i, y_{i,1}, y_{i,2})$ – правые части дифференциальных уравнений.

Пусть требуется решить систему дифференциальных уравнений первого порядка:

$$\begin{cases} \dfrac{dy_1}{dx} = y_2 \\ \dfrac{dy_2}{dx} = e^{-x \cdot y_1} \end{cases}$$

методом Эйлера на отрезке [0, 1] с шагом $h = 0.1$. Начальные условия: $x_0 = 0$; $y_1(0) = 0$; $y_2(0) = 0$. Запишем выражения для $y_{i,1}$ и $y_{i,2}$:

$$\begin{cases} y_{i,1} = y_{(i-1),1} + 0.1 \cdot y_{(i-1),2} \\ y_{i,2} = y_{(i-1),2} + 0.1 \cdot e^{-x_{i-1} \cdot y_{(i-1),1}} \\ x_i = x_{i-1} + h \end{cases}$$

Результаты вычислений сведем в таблице:

i	x_i	$y_{i,1} = y_{(i-1),1} + 0.1 \cdot y_{(i-1),2}$	$y_{i,2} = y_{(i-1),2} + 0.1 \cdot e^{-x_{i-1} \cdot y_{(i-1),1}}$
0	0.0	0.0000	0.0000
1	0.1	0.0000	0.1000
2	0.2	0.0100	0.2000
3	0.3	0.0300	0.2998
4	0.4	0.0600	0.3989
5	0.5	0.0999	0.4965
6	0.6	0.1495	0.5917
7	0.7	0.2087	0.6831
8	0.8	0.2770	0.7695
9	0.9	0.3539	0.8496
10	1.0	0.4389	0.9223

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:

$$y(x_{k+1}) = y(x_k) + y'(x_k)h + y''(x_k)\frac{h^2}{2} + \cdots$$

Заменим вторую производную в этом разложении выражением

$$y''(x_k) = \big(y'(x_k)\big)' = f'\big(x_k, y(x_k)\big) \approx \frac{f(\tilde{x}, \tilde{y}) - f\big(x_k, y(x_k)\big)}{\Delta x},$$

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 четвертого порядка точности – из-за большой распространенности данного метода, указания на тип и порядок зачастую опускаются. При этом существуют еще методы первого, второго и третьего порядка точности.

Δx подбирается из условия достижения наибольшей точности записанного выражения.

Для дальнейших выкладок произведем замену $\tilde{y}$ разложением в ряд Тейлора:

$$\tilde{y} = y(x_k + \Delta x) = y(x_k) + y'(x_k)\Delta x + \cdots$$

Введем обозначение: $y_k = y(x_k)$. Тогда для исходного уравнения построим вычислительную схему:

$$y_{k+1} = y_k + f(x_k, y_k)h + \frac{h^2}{2\Delta x}\left(f\left(x_k + \Delta x,\ y_k + y'_k \Delta x\right) - f(x_k, y_k)\right)$$

которую преобразуем к виду:

$$y_{k+1} = y_k + h\left[\left(1 - \frac{h}{2\Delta x}\right) f(x_k, y_k) + \frac{h}{2\Delta x} f\left(x_k + \Delta x,\ y_k + y'_k \Delta x\right)\right] =$$

$$= y_k + h\left[\left(1 - \frac{h}{2\Delta x}\right) f(x_k, y_k) + \frac{h}{2\Delta x} f\left(x_k + \frac{\Delta x}{h}h,\ y_k + f(x_k, y_k)\frac{\Delta x}{h}h\right)\right]$$

$$y_{k+1} = y_k + h\left[\left(1 - \frac{h}{2\Delta x}\right) f(x_k, y_k) + \frac{h}{2\Delta x} f\left(x_k + \frac{\Delta x}{h} h,\ y_k + f(x_k, y_k)\frac{\Delta x}{h} h\right)\right]$$

Введем следующие обозначения:

$$\alpha = \frac{h}{2\Delta x}, \qquad \beta = 1 - \frac{h}{2\Delta x}, \qquad \gamma = \frac{\Delta x}{h}, \qquad \delta = f(x_k, y_k)\frac{\Delta x}{h}.$$

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 Δx и могут быть определены через коэффициент α, который в этом случае играет роль параметра:

$$\beta = 1 - \alpha, \qquad \gamma = \frac{1}{2\alpha}, \qquad \delta = f(x_k, y_k)\frac{2}{\alpha}$$

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.

$$y_{k+1} = y_k + \frac{h}{2}\left[f(x_k, y_k) + f\left(x_k + h,\ y_k + hf(x_k, y_k)\right)\right].$$

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:

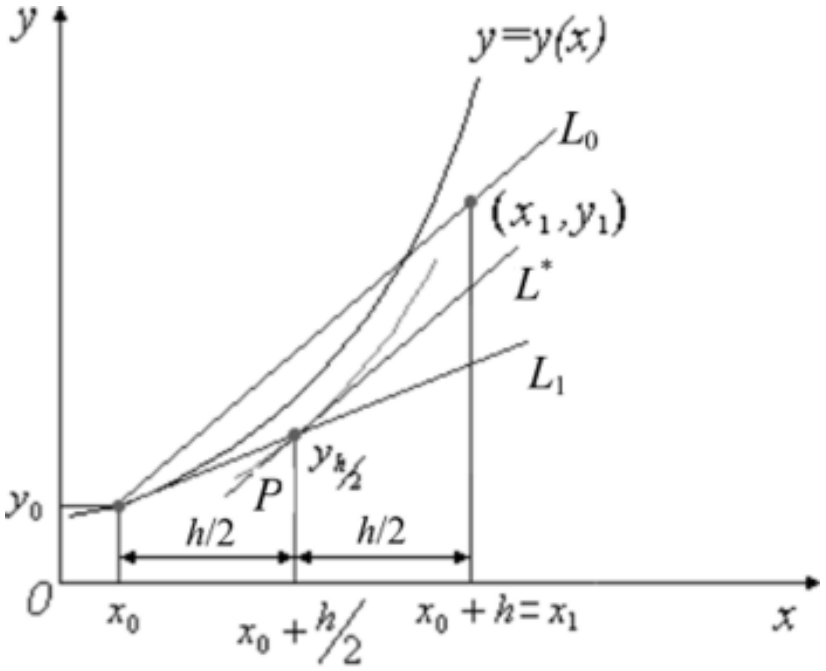
$$y_i = y_{i-1} + \frac{h}{2} \cdot (K_1 + K_2)$$

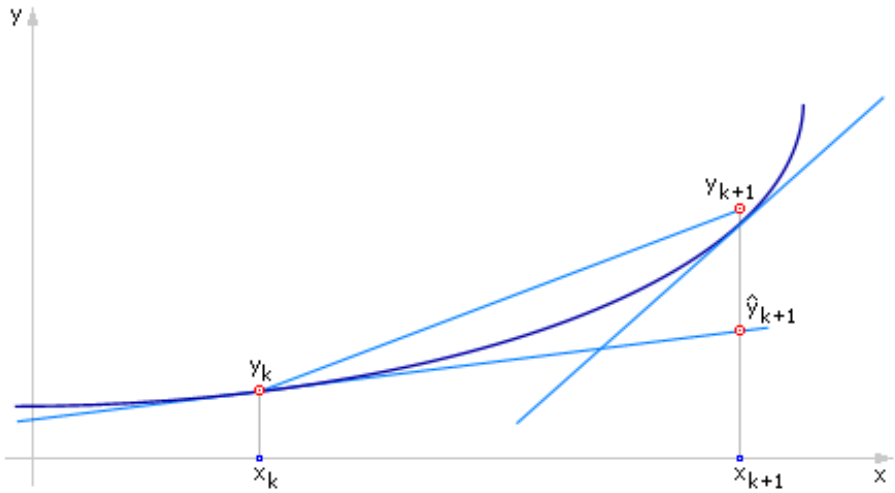
где:

$$K_1 = f(x_{i-1}, y_{i-1})$$
$$K_2 = f(x_{i-1} + h, y_{i-1} + h \cdot K_1)$$

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.





$$y_{k+1} = y_k + h\left[(1-\alpha)f(x_k, y_k) + \alpha f\left(x_k + \frac{h}{2\alpha}, y_k + f(x_k, y_k)\frac{2h}{\alpha}\right)\right].$$

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 полушаг 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 полушаг 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:

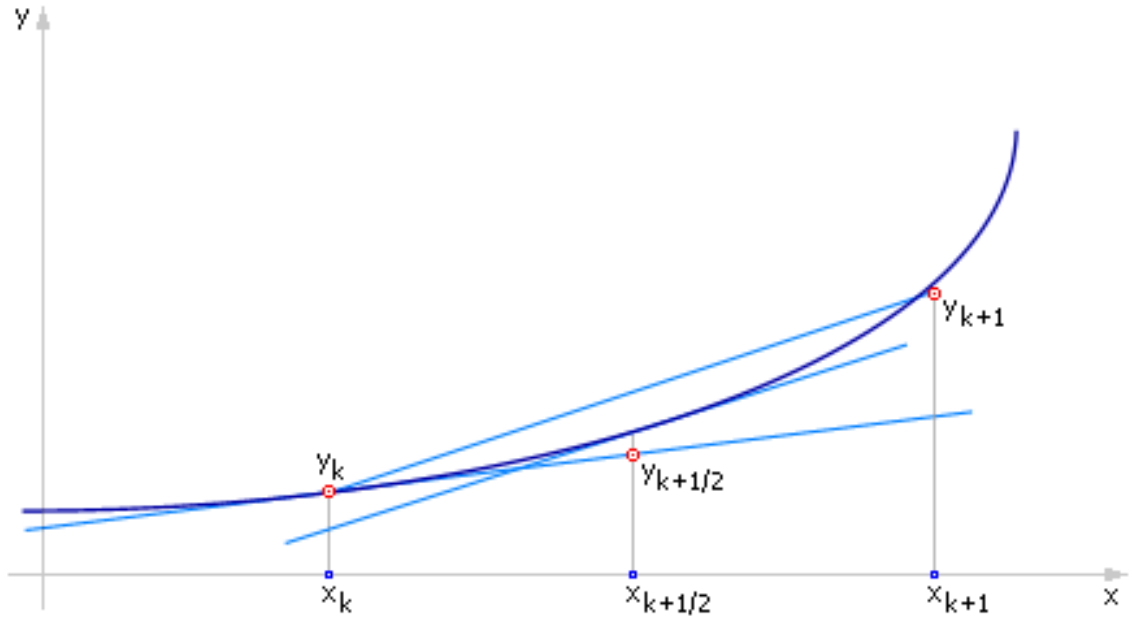
$$y'_{k+1/2} = f\left(x_k + \frac{h}{2}, y_{k+1/2}\right)$$

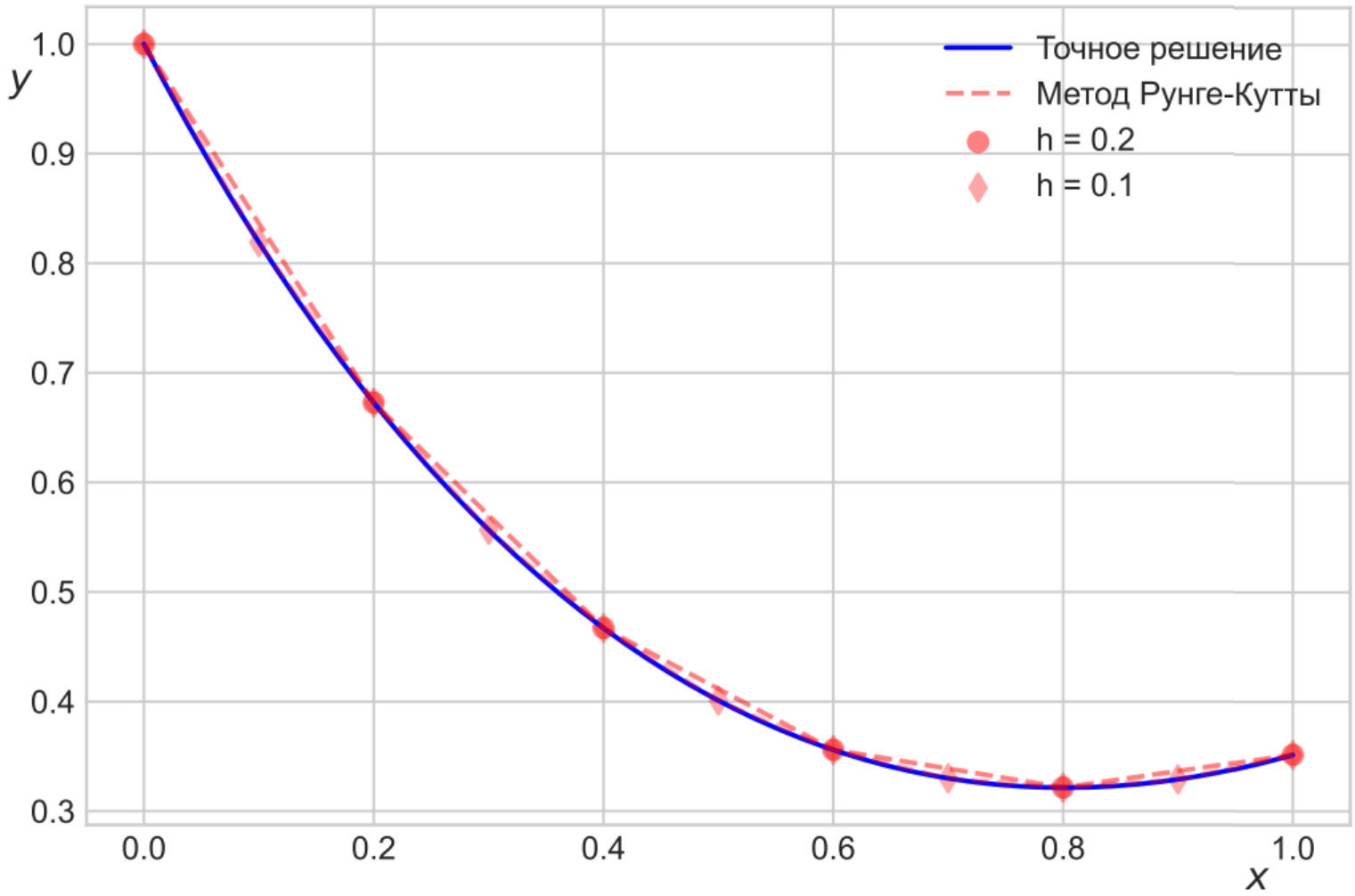
3. Определяется уточненное значение функции в конечной точке всего шага, причем по схеме Эйлера с вычисленным на предыдущем шаге значением производной

$$y_{k+1} = y_k + h \cdot y'_{k+1/2}$$

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:

$$y_1 = y_0 + h(x_0^2 - \sin 2x_0) = 2 + 0{,}1(0 - 0) = 2,$$
$$y_2 = y_1 + h(x_1^2 - \sin 2x_1) = 2 + 0{,}1(0{,}1 - \sin(2 \cdot 0{,}1)) = 1{,}981,$$
$$y_3 = y_2 + h(x_1^2 - \sin 2x_1) = 1{,}981 + 0{,}1(0{,}2 - \sin(2 \cdot 0{,}2)) = 1{,}946,$$
$$y_4 = y_3 + h(x_1^2 - \sin 2x_1) = 1{,}946 + 0{,}1(0{,}3 - \sin(2 \cdot 0{,}3)) = 1{,}899,$$

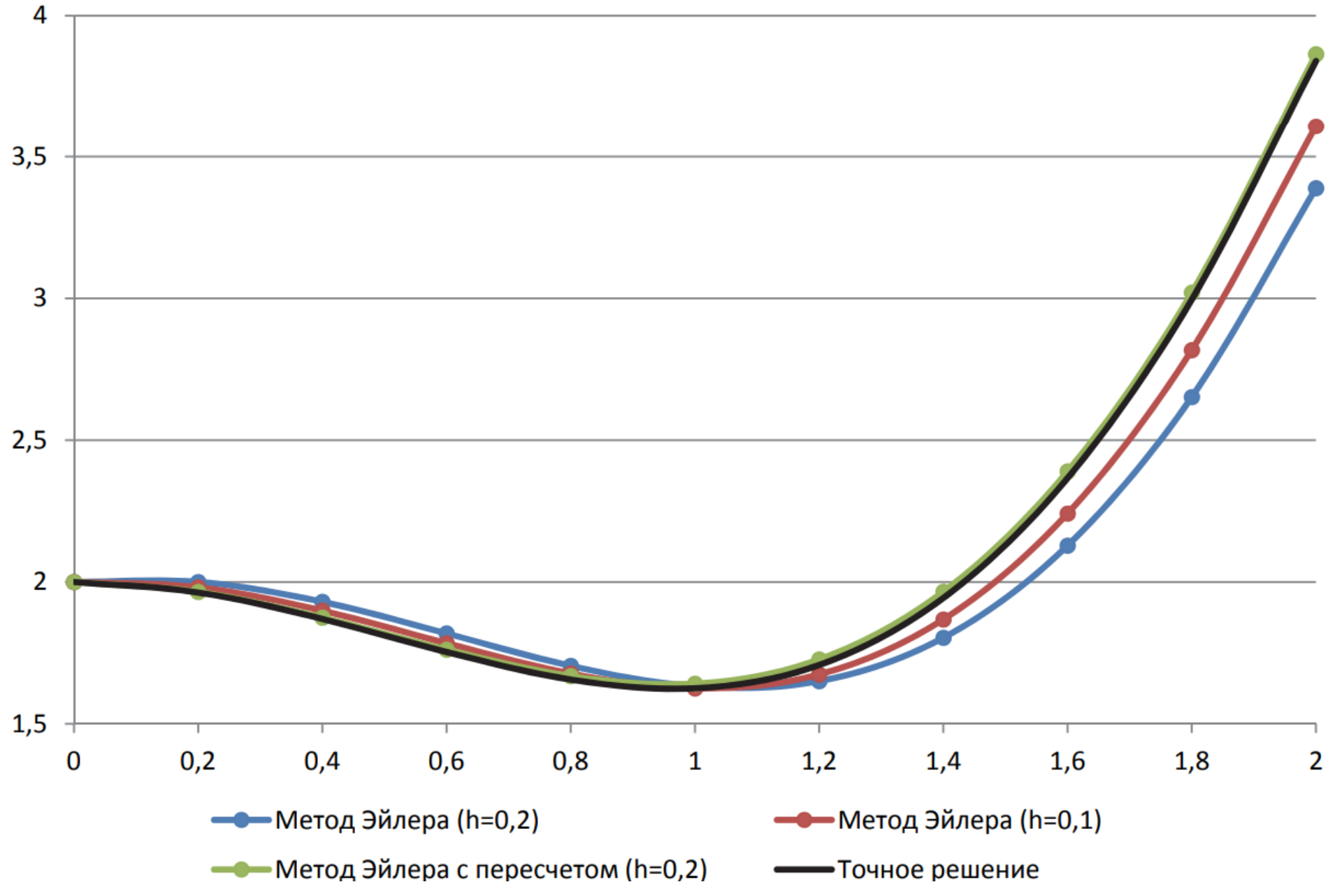
x_i	0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1
y_i	2	2,000	1,981	1,946	1,899	1,843	1,784	1,727	1,677	1,641	1,625

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 пересчетом, $h = 0{,}2$)	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


4
3,5
3
2,5
2
1,5
0
0,2
0,4
0,6
0,8
1
1,2
1,4
1,6
1,8
2
Метод Эйлера (h=0,2)
Метод Эйлера (h=0,1)
Метод Эйлера с пересчетом (h=0,2)
Точное решение

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).

$$y_i = y_{i-1} + \frac{h}{6} \cdot (k_1 + 2 \cdot k_2 + 2 \cdot k_3 + k_4)$$

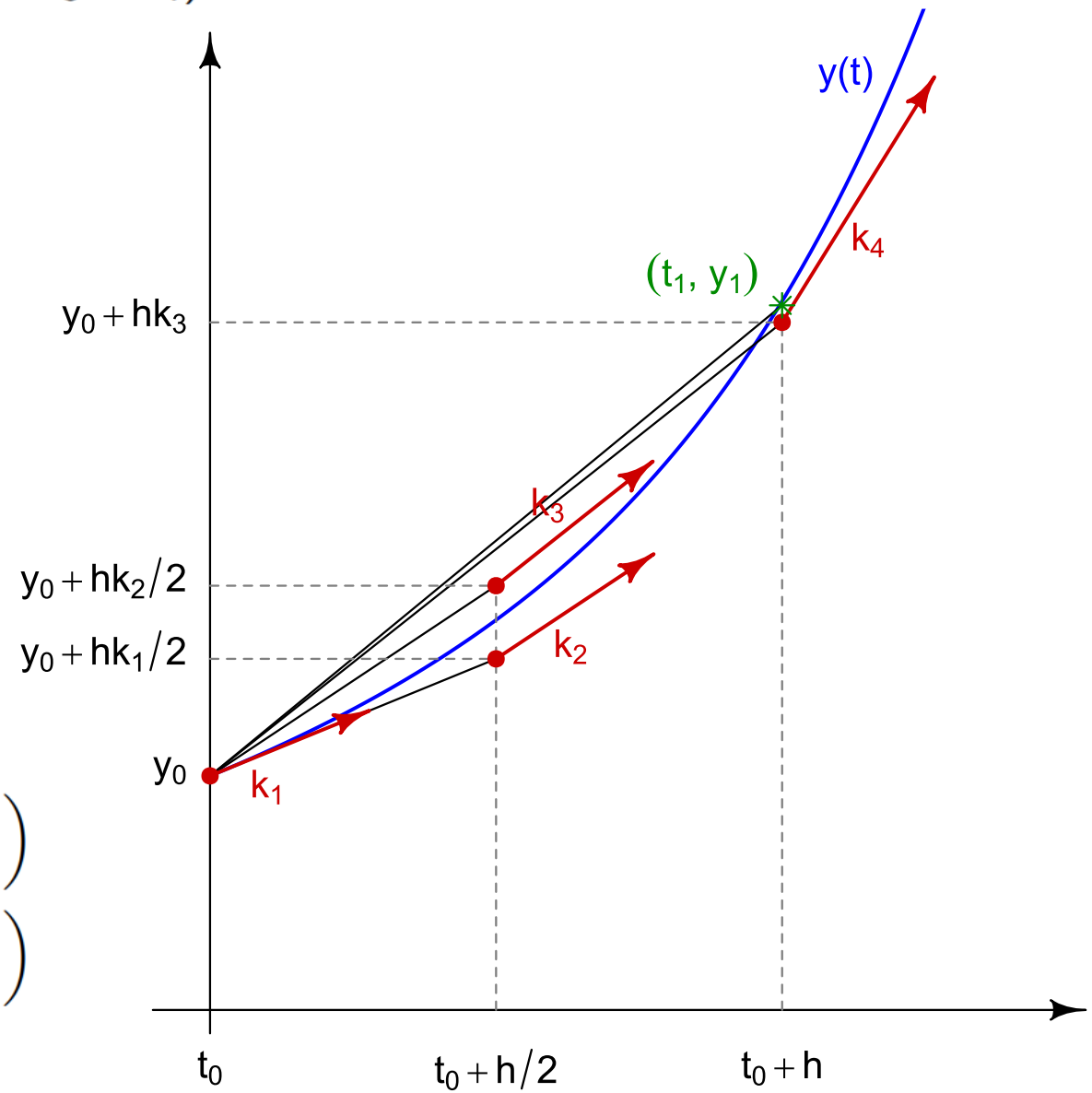
$$x_i = x_{i-1} + h$$

$$k_1 = f\left(x_{i-1}, y_{i-1}\right)$$

$$k_2 = f\left(x_{i-1} + \frac{h}{2}, y_{i-1} + \frac{h}{2} \cdot k_1\right)$$

$$k_3 = f\left(x_{i-1} + \frac{h}{2}, y_{i-1} + \frac{h}{2} \cdot k_2\right)$$

$$k_4 = f\left(x_{i-1} + h, y_{i-1} + h \cdot k_3\right)$$



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 *yi*, соответствующих значениям переменной x из диапазона [0, 1] с шагом $h = 0{,}1$.

Подставим в формулы

$$k_1 = f(x_{i-1}, y_{i-1}) \qquad k_3 = f\left(x_{i-1} + \frac{h}{2}, y_{i-1} + \frac{h}{2} \cdot k_2\right)$$
$$k_2 = f\left(x_{i-1} + \frac{h}{2}, y_{i-1} + \frac{h}{2} \cdot k_1\right) \qquad k_4 = f(x_{i-1} + h, y_{i-1} + h \cdot k_3)$$

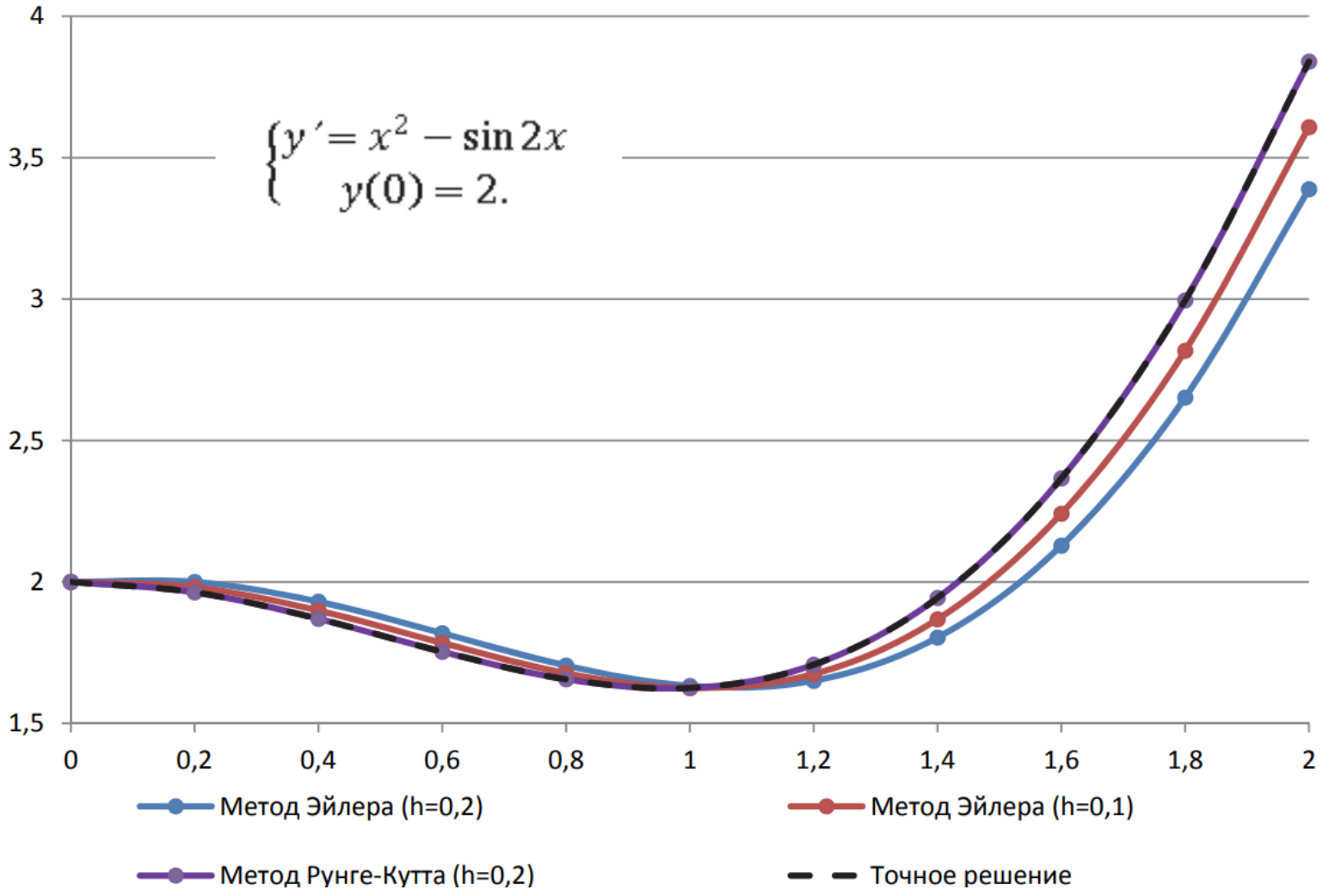
выражение правой части исходного дифференциального уравнения и запишем формулы для определения параметров метода Рунге-Кутты:

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 повлияет на точность результатов. Поэтому метод Эйлера применяется достаточно редко при небольшом числе расчетных точек. Наиболее употребительным одношаговым методом является метод Рунге-Кутта.


$\begin{cases} y' = x^2 - \sin 2x \\ y(0) = 2. \end{cases}$
4
3,5
3
2,5
2
1,5
0
0,2
0,4
0,6
0,8
1
1,2
1,4
1,6
1,8
2
Метод Эйлера (h=0,2)
Метод Эйлера (h=0,1)
Метод Рунге-Кутта (h=0,2)
Точное решение

Решение систем обыкновенных дифференциальных уравнений с помощью метода Рунге-Кутта.

Рассмотренные методы могут быть использованы также для решения систем дифференциальных уравнений. Покажем это для системы двух уравнений вида

$$\frac{dy}{dx} = \phi(x, y, z),$$

$$\frac{dz}{dx} = \psi(x, y, z).$$

Начальные условия зададим в виде

$$y(x_0) = y_0, \quad z(x_0) = z_0 .$$

$$\frac{dy}{dx} = \phi(x, y, z), \qquad \frac{dz}{dx} = \psi(x, y, z). \qquad y(x_0) = y_0, \quad z(x_0) = z_0$$

Запишем формулу Рунге-Кутта для системы двух уравнений:

$$y_{i+1} = y_i + \frac{1}{6}(k_0 + 2k_1 + 2k_2 + k_3), \qquad i = 0,1,\ldots$$

$$z_{i+1} = z_i + \frac{1}{6}(l_0 + 2l_1 + 2l_2 + l_3),$$

$$k_0 = h\phi(x_i, y_i, z_i), \quad l_0 = h\psi(x_i, y_i, z_i),$$

$$k_1 = h\phi\left(x_i + \frac{h}{2}, y_i + \frac{k_0}{2}, z_i + \frac{l_0}{2}\right), \quad l_1 = h\psi\left(x_i + \frac{h}{2}, y_i + \frac{k_0}{2}, z_i + \frac{l_0}{2}\right),$$

$$k_2 = h\phi\left(x_i + \frac{h}{2}, y_i + \frac{k_1}{2}, z_i + \frac{l_1}{2}\right), \quad l_2 = h\psi\left(x_i + \frac{h}{2}, y_i + \frac{k_1}{2}, z_i + \frac{l_1}{2}\right),$$

$$k_3 = h\phi(x_i + h, y_i + k_2, z_i + l_2), \quad l_3 = h\psi(x_i + h, y_i + k_2, z_i + l_2).$$

Пусть дана следующая система обыкновенных дифференциальных уравнений:

$$\begin{cases} \dfrac{dy_1}{dx} = f_1(x, y_1, y_2) \\ \dfrac{dy_2}{dx} = f_2(x, y_1, y_2) \end{cases}$$

с начальными условиями:

$$y_1\big|_{x=x_0} = y_{01}$$
$$y_2\big|_{x=x_0} = y_{02}$$

При использовании метода Рунге-Кутты, расчетные формулы примут следующий вид:

$$\begin{cases} y_{i,1} = y_{(i-1),1} + h/6 \cdot (k_{1,1} + 2 \cdot k_{2,1} + 2 \cdot k_{3,1} + k_{4,1}) \\ y_{i,2} = y_{(i-1),2} + h/6 \cdot (k_{1,2} + 2 \cdot k_{2,2} + 2 \cdot k_{3,2} + k_{4,2}) \\ x_i = x_{i-1} + h \end{cases}$$

где:

$$k_{1,1} = f_1\left(x, y_{(i-1),1}, y_{(i-1),2}\right);$$

$$k_{2,1} = f_1\left(x + \frac{h}{2}, y_{(i-1),1} + k_{1,1} \cdot \frac{h}{2}, y_{(i-1),2} + k_{1,2} \cdot \frac{h}{2}\right);$$

$$k_{3,1} = f_1\left(x + \frac{h}{2}, y_{(i-1),1} + k_{2,1} \cdot \frac{h}{2}, y_{(i-1),2} + k_{2,2} \cdot \frac{h}{2}\right);$$

$$k_{4,1} = f_1\left(x + h, y_{(i-1),1} + k_{3,1} \cdot h, y_{(i-1),2} + k_{3,2} \cdot h\right);$$

$$k_{1,2} = f_2\left(x, y_{(i-1),1}, y_{(i-1),2}\right);$$

$$k_{2,2} = f_2\left(x + \frac{h}{2}, y_{(i-1),1} + k_{1,1} \cdot \frac{h}{2}, y_{(i-1),2} + k_{1,2} \cdot \frac{h}{2}\right);$$

$$k_{3,2} = f_2\left(x + \frac{h}{2}, y_{(i-1),1} + k_{2,1} \cdot \frac{h}{2}, y_{(i-1),2} + k_{2,2} \cdot \frac{h}{2}\right);$$

$$k_{4,2} = f_2\left(x + h, y_{(i-1),1} + k_{3,1} \cdot h, y_{(i-1),2} + k_{3,2} \cdot h\right).$$

где h – шаг интегрирования; $f_1\left(x_i, y_{(i-1),1}, y_{(i-1),2}\right)$ и $f_2\left(x_i, y_{(i-1),1}, y_{(i-1),2}\right)$ – правые части дифференциальных уравнений, $k_{1,j}$, $k_{2,j}$, $k_{3,j}$, $k_{4,j}$ – параметры метода Рунге-Кутты для j-го уравнения.

Пусть требуется решить систему дифференциальных уравнений первого порядка:

$$\begin{cases} \frac{dy_1}{dx} = y_2 \\ \frac{dy_2}{dx} = e^{-x \cdot y_1} \end{cases}$$

методом Рунге-Кутты на отрезке $[0, 1]$ с шагом $h = 0.1$. Начальные условия: $x_0 = 0$; $y_1(0) = 0$; $y_2(0) = 0$.

Запишем выражения для нахождения значений искомых переменных $y_{i,1}$ и $y_{i,2}$:

$$k_{1,1} = y_{(i-1),2};$$
$$k_{2,1} = y_{(i-1),2} + k_{1,2} \cdot \frac{h}{2};$$
$$k_{3,1} = y_{(i-1),2} + k_{2,2} \cdot \frac{h}{2};$$
$$k_{4,1} = y_{(i-1),2} + k_{3,2} \cdot h;$$

$$k_{1,2} = \exp\left(-x_i \cdot y_{(i-1),1}\right);$$
$$k_{2,2} = \exp\left[-\left(x_i + \frac{h}{2}\right) \cdot \left(y_{(i-1),1} + k_{1,1} \cdot \frac{h}{2}\right)\right]$$
$$k_{3,2} = \exp\left[-\left(x_i + \frac{h}{2}\right) \cdot \left(y_{(i-1),1} + k_{2,1} \cdot \frac{h}{2}\right)\right]$$
$$k_{4,2} = \exp\left[-\left(x_i + h\right) \cdot \left(y_{(i-1),1} + k_{3,1} \cdot h\right)\right]$$

$$\begin{cases} y_{i,1} = y_{(i-1),1} + \dfrac{0.1}{6} \cdot \left(k_{1,1} + 2 \cdot k_{2,1} + 2 \cdot k_{3,1} + k_{4,1}\right) \\ y_{i,2} = y_{(i-1),2} + \dfrac{0.1}{6} \cdot \left(k_{1,2} + 2 \cdot k_{2,2} + 2 \cdot k_{3,2} + k_{4,2}\right) \\ x_i = x_{i-1} + 0.1 \end{cases}$$

Результаты вычислений сведем в таблице:

i	x_i	$k_{1,1}$	$k_{2,1}$	$k_{3,1}$	$k_{4,1}$	$y_{i,1}$	$k_{1,2}$	$k_{2,2}$	$k_{3,2}$	$k_{4,2}$	$y_{i,2}$
0	0.0	—	—	—	—	0.0000	—	—	—	—	0.0000
1	0.1	0.0000	0.0500	0.0500	0.1000	0.0050	1.0000	1.0000	0.9999	0.9995	0.1000
2	0.2	0.1000	0.1500	0.1499	0.1998	0.0200	0.9995	0.9985	0.9981	0.9960	0.1998
3	0.3	0.1998	0.2496	0.2494	0.2990	0.0449	0.9960	0.9925	0.9919	0.9866	0.2990
4	0.4	0.2990	0.3483	0.3480	0.3968	0.0797	0.9866	0.9793	0.9784	0.9686	0.3968
5	0.5	0.3968	0.4453	0.4446	0.4923	0.1242	0.9686	0.9562	0.9551	0.9398	0.4924
6	0.6	0.4924	0.5393	0.5384	0.5844	0.1781	0.9398	0.9214	0.9202	0.8987	0.5844
7	0.7	0.5844	0.6293	0.6281	0.6716	0.2409	0.8987	0.8739	0.8727	0.8448	0.6717
8	0.8	0.6717	0.7139	0.7124	0.7529	0.3122	0.8448	0.8139	0.8126	0.7790	0.7529
9	0.9	0.7529	0.7919	0.7901	0.8271	0.3913	0.7790	0.7427	0.7415	0.7032	0.8271
10	1.0	0.8271	0.8623	0.8603	0.8933	0.4774	0.7032	0.6630	0.6619	0.6204	0.8933

Рассмотрим следующую модель химических реакций:

$$A \to B \to C$$

с константами скоростей k_1 и k_2. Уравнения, описывающие скорость изменения концентраций компонентов по времени, записываются следующим образом:

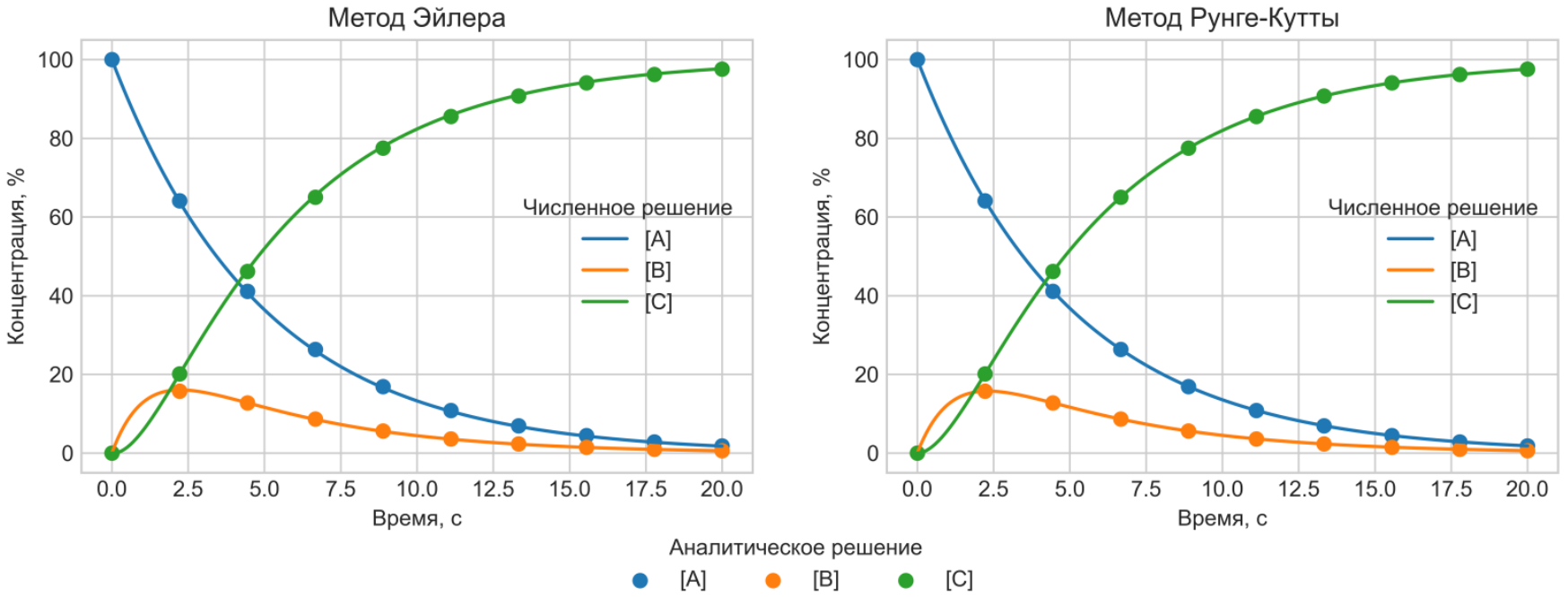
$$\begin{cases} \frac{d[A]}{dt} = -k_1 \cdot [A] \\ \frac{d[B]}{dt} = k_1 \cdot [A] - k_2 \cdot [B] \\ \frac{d[C]}{dt} = k_2 \cdot [B] \end{cases}$$

или:

$$\begin{cases} \frac{d[A]}{dt} = -k_1 \cdot y_1 \\ \frac{d[B]}{dt} = k_1 \cdot y_1 - k_2 \cdot y_2 \\ \frac{d[C]}{dt} = k_2 \cdot y_2 \end{cases}$$

Зададимся значениями констант: $k_1 = 0.2$ c^{-1}, $k_2 = 0.8$ c^{-1} и начальными условиями: $y_1(0) = 100$, $y_2(0) = 0$, $y_3(0) = 0$.

Графическая визуализация



Изменение концентрации реагирующих веществ во времени

Домашняя работа #7

Сгенерировать в виде gif–анимации динамику развития и перемещения в пространстве двух популяций при помощи модели конкуренции с учётом территориальной расположенности:

$$\begin{cases} V_t' = b_V * V - d_V * V^2 - c_U * U * V + \int_{y-Dy_1}^{y+Dy_1} \int_{x-Dx_1}^{x+Dx_1} K_U(x,y) * V(x,y,t) dxdy \\ U_t' = b_U * U - d_U * U^2 - c_V * U * V + \int_{y-Dy_2}^{y+Dy_1} \int_{x-Dx_2}^{x+Dx_2} K_V(x,y) * U(x,y,t) dxdy \end{cases}$$

Срок сдачи работ: **до 19-го мая**.

Работы принимаются на электронный адрес VSChernyshenko@fa.ru в виде отдельного файла **.gif**, а также самого ноутбука – исключительно в формате **html** или **pdf**. Название файла: “Фамилия_Имя_группа_ДЗ7.*”

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

Домашняя работа #7

Модель конкуренции с учётом территориальной расположенности

$$\begin{cases} V_t' = b_V * V - d_V * V^2 - c_U * U * V + \int_{y-Dy_1}^{y+Dy_1} \int_{x-Dx_1}^{x+Dx_1} K_U(x,y) * V(x,y,t) dxdy \\ U_t' = b_U * U - d_U * U^2 - c_V * U * V + \int_{y-Dy_2}^{y+Dy_2} \int_{x-Dx_2}^{x+Dx_2} K_V(x,y) * U(x,y,t) dxdy \end{cases}$$

Биомасса популяции в конкретный момент времени на элементарной площадке с координатами (x; y) зависит от: размера популяции 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.).

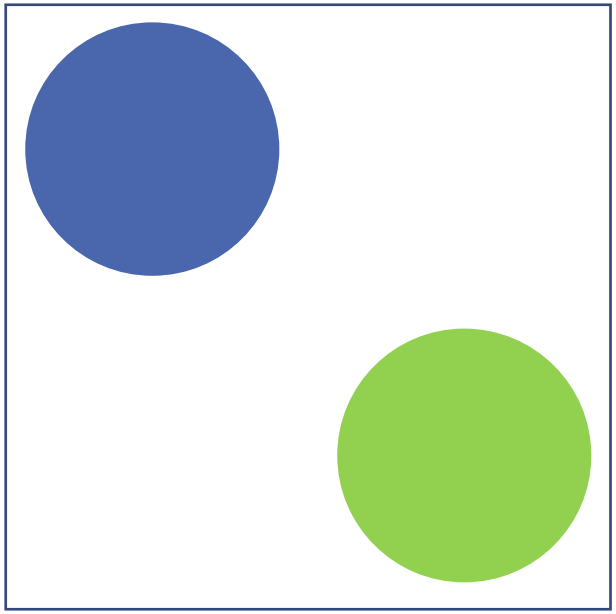
См. слайды 457-461.

Домашняя работа #7

Две круглые области, находящиеся в левом верхнем и правом нижнем углах, представляют собой две растительные популяции U и V. Область закрашенная синим цветом – область заселенная представителями U, а зелёным – область заселенная представителями V. Первоначальные значения U(x,y,t) и V(x,y,t) в этих областях задаются коэффициентами U_0 и V_0. Коэффициент R задает размеры этих областей (радиусы).

Общая территория рассматривается на сетке, с количеством узлов по вертикали и горизонтали равным, минимум, 801.

Главная задача программы – рассчитать на всей области для каждой элементарной площадки количество биомассы на ней, т.е. рассчитать новые значения U(x, y, t) и V(x, y, t).



Домашняя работа #7

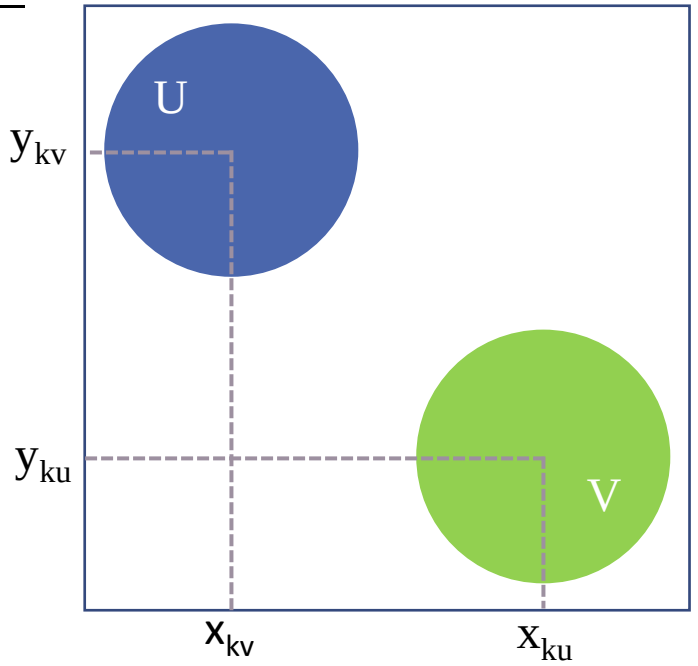
$$\begin{cases} V_t' = b_V * V - d_V * V^2 - c_U * U * V + \int_{y-Dy_1}^{y+Dy_1} \int_{x-Dx_1}^{x+Dx_1} K_U(x,y) * V(x,y,t) dxdy \\ U_t' = b_U * U - d_U * U^2 - c_V * U * V + \int_{y-Dy_2}^{y+Dy_2} \int_{x-Dx_2}^{x+Dx_2} K_V(x,y) * U(x,y,t) dxdy \end{cases}$$

Будем полагать, функции K_V и K_U имеют следующий вид:

$$K_V = e^{-\frac{(x-x_{kv})^2}{2\sigma_{V,x}^2} - \frac{(y-y_{kv})^2}{2\sigma_{V,y}^2}}$$

$$K_U = e^{-\frac{(x-x_{ku})^2}{2\sigma_{U,x}^2} - \frac{(y-y_{ku})^2}{2\sigma_{U,y}^2}}$$

Коэффициенты x_{kv}, y_{kv}, x_{ku}, y_{ku} определяют координаты, в которых популяция набирает самую большую скорость роста (наиболее подходящие для неё условия). По умолчанию коэффициенты расставлены так, что пик роста у одной популяции должен наблюдаться на территории, занимаемой второй и наоборот.



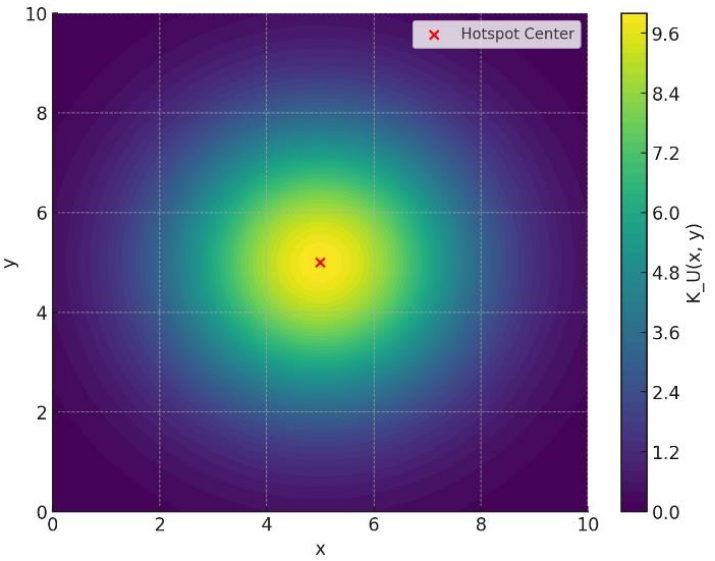
Домашняя работа #7

Функции K_V и K_U определяющие распределение прироста популяции представляют собой двухмерные функции Гаусса:

$$K_V = I_V \cdot e^{-\frac{(x-x_{kv})^2}{2\sigma_{V,x}^2} - \frac{(y-y_{kv})^2}{2\sigma_{V,y}^2}}$$

$$K_U = I_U \cdot e^{-\frac{(x-x_{ku})^2}{2\sigma_{U,x}^2} - \frac{(y-y_{ku})^2}{2\sigma_{U,y}^2}}$$

- I_V, I_U – «Интенсивность» пятна, задающего максимальное значение функции.
- $\sigma_{V,x}$, $\sigma_{V,y}$, $\sigma_{U,x}$, $\sigma_{U,y}$ – Стандартные отклонения, определяющие «вытянутость» пятна по x и y:
  - малые σ: узкое, резко выраженное пятно;
  - большие σ: широкая, менее выраженное пятно;
  - при $\sigma_x \neq \sigma_y$, пятна вытягиваются в соответствующем направлении.



Домашняя работа #7

- Каждая популяция отображается не одним цветом, а градиентом цвета, в зависимости от своей биомассы/численности. (Например: если в клетке одна особь — цвет бледно-голубой, 100 особей — тёмно-синий).
- Построенный временной срез должен включать в себя эпохи когда популяции встречаются, продвигаются дальше пока каждая устойчиво не занимает противоположный угол пространства. При этом шаг по времени может быть не слишком маленьким, чтобы анимация выглядела сколь-нибудь динамично.
- Должна быть дана интерпретация модели, а частности, объясняться, почему для каждой была выбрана следующая скорость роста, ёмкость ниши, значение коэффициента конкуренции, $Dx_{1,2}$, $Dy_{1,2}$. Соответствующие коэффициенты для разных популяций не должны быть равны между собой.

Домашняя работа #7

Обязательные условия :

- Расчёт значений функций на каждом шаге должен совершаться при помощи **метода Рунге-Кута 4-го порядка**.
- Разрешается использование методов пакета matplotlib, math, следующих методов пакета Numpy: array, linspace, meshgrid, zeros, zeros_like, shape, random, sqrt, log, exp, sin, cos, tan. Наличие иных методов приводит к аннулированию оценки работы.
- В начале вашего ноутбука укажите, какие реальные популяции вы моделируете. Приведите краткое обоснование, почему эти популяции подходят для данной модели конкуренции. На основе выбранных популяций объясните, как вы выбрали значения для параметров модели (таких как темпы роста, ёмкости ниши, коэффициенты конкуренции, коэффициенты задающие скорость распространения по территории). Убедитесь, что ваши параметры отражают экологические характеристики выбранных популяций.

Домашняя работа #7

Обязательные условия. Ответьте на следующие вопросы в конце ноутбука:

1. При описании каких популяций и территорий могут отличаться $\sigma_{V,x}$ и $\sigma_{V,y}$; $\sigma_{U,x}$ и $\sigma_{U,y}$?
2. Что произойдёт с результатами, если заменить его на метод Эйлера?
3. Как вы определили шаг по времени Δt?
4. Какие выводы о взаимодействии популяций вы сделали, глядя на анимацию? Как визуализация помогла вам лучше понять модель?
5. Как можно изменить модель, чтобы она лучше соответствовала реальным наблюдениям?
6. Какие ограничения есть у вашей модели? Например, какие упрощения вы сделали?
7. Предложите, как можно улучшить модель. Что бы вы добавили или изменили, чтобы она стала более реалистичной или полезной?

Домашняя работа #7

Опционально:

- Для построения анимации и сохранения итоговой версии в формате gif-файла предлагается использование Matplotlib, класса Animation (пример: https://www.geeksforgeeks.org/using-matplotlib-for-animations/). Возможно использование альтернативных пакетов.

- Расширение-усложнение модели (например, включение дополнительных факторов, стохастических эффектов окружающей среды, сезонных изменений, дополнительных и отличающихся популяций, дополнительных экологических взаимодействий, анизотропных диффузий) с детальным описанием этих модификаций, причин, по которым они были включены, будет щедро оцениваться в режиме **бонусных баллов** (*неограниченно*), а срок сдачи работы увеличивается на неделю.

Исследование на устойчивость – ключевой аспект численных методов для решения дифференциальных уравнений. Устойчивость определяет, насколько надёжно схема сохраняет точность решения при наличии возмущений.

Разностные схемы используются для численного решения дифференциальных уравнений. Устойчивость схемы определяет, насколько она чувствительна к малым возмущениям в начальных данных или правой части. Формально, схема считается устойчивой, если решение u_h удовлетворяет оценке:

$$\| u_h \| \leq M \| f_h \|,$$

где u_h – решение, f_h – правая часть, а M – константа, не зависящая от шага сетки h.

Теорема: В случае линейного оператора определения устойчивости сводятся к существованию обратного оператора L_h^{-1} с ограниченной нормой:

$$\| L_h^{-1} \| \leq M.$$

Это гарантирует, что малые изменения в f_h не приводят к большим ошибкам в u_h.

Для анализа устойчивости часто рассматривают оператор перехода R_h, который связывает решение на следующем шаге с текущим: $u^{n+1} = R_h u^n$. Устойчивость зависит от поведения R_h при многократном применении, то есть от R_h^n :

$$\| R_h^n \| \leq C,$$

где C – константа, а n – число шагов. Это означает, что ошибки не растут экспоненциально.

Ограниченность собственных чисел

Одно из ключевых условий устойчивости связано с собственными числами λ_i оператора R_h:

- Модуль собственных чисел должен быть ограничен: $|\lambda_i| \leq 1$.
- Это гарантирует, что ошибки в решении не растут экспоненциально при увеличении числа шагов n.

Однако само по себе $|\lambda_i| \leq 1$ не всегда достаточно. Поведение решения зависит также от структуры оператора R_h, особенно если есть собственные числа с $|\lambda_i| = 1$. Здесь появляются корневые векторы.

Корневые векторы (*обобщённые собственные векторы*) возникают, когда оператор R_h недиагонализируем (то есть имеет кратные собственные числа, и его жорданова форма содержит ненулевые элементы над главной диагональю). Для кратного собственного числа λ_i корневые векторы определяют дополнительные компоненты решения, которые могут вносить вклад в его рост.

Для примера, рассмотрим жорданов блок размера 2:

$$J = \begin{pmatrix} \lambda & 1 \\ 0 & \lambda \end{pmatrix}.$$

- Существует собственный вектор $\mathbf{v}_1$, для которого $A\mathbf{v}_1 = \lambda\mathbf{v}_1$.
- И корневой вектор $\mathbf{v}_2$, для которого:

$$A\mathbf{v}_2 = \lambda\mathbf{v}_2 + \mathbf{v}_1.$$

Общее определение корневого вектора: это вектор $\mathbf{v}$, который удовлетворяет уравнению:

$$(A - \lambda I)^k \mathbf{v} = 0,$$

где k — некоторое целое число, называемое высотой вектора. Для собственных векторов $k = 1$, а для корневых векторов $k > 1$.

Собственные векторы – частный случай корневых векторов, когда $k = 1$. Они напрямую связаны с собственным значением λ через простое масштабирование.

Корневые векторы – более общее понятие. Они появляются в цепочках обобщённых векторов, связанных с жордановыми блоками, и описывают поведение системы при кратных собственных значениях.

Пример

Если у матрицы есть собственное значение λ с алгебраической кратностью 2, но только один собственный вектор, то второй вектор в базисе будет корневым. Например:

- $\mathbf{v}_1$: $A\mathbf{v}_1 = \lambda\mathbf{v}_1$ (собственный вектор),
- $\mathbf{v}_2$: $A\mathbf{v}_2 = \lambda\mathbf{v}_2 + \mathbf{v}_1$ (корневой вектор).

Корневые векторы не являются собственными векторами в строгом смысле, но включают их как частный случай. Они обобщают понятие собственных векторов и играют важную роль в анализе матриц с кратными собственными значениями.

Если $|\lambda_i| = 1$, устойчивость схемы зависит от того, как ведут себя соответствующие корневые векторы при многократном применении R_h. Рассмотрим это на примере:

1. **Простое собственное число ($|\lambda_i| = 1$, кратность 1):**
   - Решение пропорционально λ_i^n.
   - Так как $|\lambda_i| = 1$, то $|\lambda_i^n| = 1$, и решение остаётся ограниченным (не растёт экспоненциально).
   - Корневые векторы здесь не играют роли, так как их нет.
2. **Кратное собственное число ($|\lambda_i| = 1$, кратность > 1):**
   - В жордановой форме оператора R_h появляются блоки, и решение включает не только λ_i^n, но и полиномиальные множители вида n^k, где k – размер жорданова блока минус 1.
   - Например, для жорданова блока размера 2 решение будет содержать член $n\lambda_i^n$.
   - Так как $|\lambda_i| = 1$, то $|\lambda_i^n| = 1$, но множитель n растёт линейно с числом шагов n.
   - Если кратность ещё выше (например, блок размера 3), появляются члены $n^2\lambda_i^n$, и рост становится квадратичным.

"Ограниченность корневых векторов" означает, что эти полиномиальные множители не приводят к неконтролируемому росту решения. Формально, норма $\| R_h^n \|$ должна оставаться ограниченной константой C для всех n:

$$\| R_h^n \| \leq C.$$

Если $|\lambda_i| = 1$ и есть кратность, рост типа n или n^2 допустим для нестрогой устойчивости (см. ниже), но он не должен быть экспоненциальным.

Итак, наша задача сводится к двум уравнениям:

1. дифференциальное $Lu = f$;
2. его разностная аппроксимация $L_h u_h = f_h$.

Здесь L – дифференциальный оператор, u – точное решение, L_h – дискретный оператор, а u_h – численное решение.

Невязка $r_h = L_h[u]_h - f_h$ (где $[u]_h$ – проекция точного решения на сетку) позволяет оценить, насколько хорошо разностная схема приближает исходную задачу. Устойчивость связана с тем, как эти ошибки эволюционируют во времени.

Одним из фундаментальных условий устойчивости является **ограниченность собственных чисел** оператора.

Рассмотрим оператор перехода R_h, который определяет, как решение переходит от одного временного шага к следующему:

$$u^{n+1} = R_h u^n + h p_n,$$

где h – шаг по времени; p_n позволяет учесть влияние факторов, не включённых в R_h на шаге n.

Норма оператора $\| R_h \| \geq |\lambda_i|$, где λ_i – собственные значения R_h. Для устойчивости требуется, чтобы $|\lambda_i^n| < M$ (где M – константа) при любом n.

Это означает, что собственные значения должны лежать внутри или на границе единичного круга на комплексной плоскости ($|\lambda_i| \leq 1$), чтобы предотвратить экспоненциальный рост ошибок. Иными словами:

$$\lambda = 1 + \tau \frac{\partial f}{\partial u},$$

где устойчивость зависит от выбора шага τ.

Ограниченность норм степеней оператора

Ограниченность норм степеней оператора перехода является достаточным условием устойчивости разностных схем:

$$\| R_h^n \| \leq C \quad \text{для всех } n.$$

Для устойчивости необходимо, чтобы эта норма не возрастала бесконтрольно.

Строгая устойчивость необходима, если мы считаем **неопределённое количество шагов по времени**.

Например, при моделировании долгосрочных процессов. Здесь требуется, чтобы $|\boldsymbol{\lambda_i^n}| < \mathbf{1}$ (строго меньше 1), что обеспечивает затухание возмущений со временем.

$$\| u^n \| \leq \max_{0 \leq k \leq n-1} \| R_h^k \| (\| y^0 \| + Nh \max \| p_k \|)$$

если $\| R_h \| < 1$, ошибки экспоненциально уменьшаются, что идеально для длительных вычислений.

Требуется, когда нужно выполнять вычисления на неопределённо большом числе шагов.

Последствия:

- $|\lambda_i^n| \to 0$ при $n \to \infty$,
- $\| R_h^n \| \to 0$,
- Возмущения затухают со временем.

Корневые векторы здесь не возникают, так как $|\lambda_i| < 1$ гарантирует экспоненциальное убывание всех компонент решения.

Нестрогая (слабая) устойчивость применима, если нам нужно сделать **лишь несколько шагов по времени**.

Здесь достаточно, чтобы $|\boldsymbol{\lambda_i^n}| \leq \mathbf{1} + \boldsymbol{c\tau}$ (где c – константа, τ – шаг времени).

Что допускает небольшой рост ошибок, но не приводит к их чрезмерному росту на коротких интервалах.

$$\| u^n \| \leq C \| F \| (1 + Nh)$$

показывает, что ошибка ограничена линейной функцией числа шагов.

Подходит для задач с конечным числом шагов.

Условие: $|\lambda_i| \leq 1$, и для $|\lambda_i| = 1$ соответствующие корневые векторы должны быть ограничены.

Последствия:

- $\| R_h^n \|$ остаётся ограниченной, но не обязательно убывает.
- Если $|\lambda_i| = 1$ с кратностью, решение может расти полиномиально (например, n или n^2), что приемлемо для ограниченного n.

Пример: решение колеблется (как при $\lambda_i = -1$) или растёт линейно, но не экспоненциально.

Для разностных схем, таких как те, что представлены на слайдах 8 и 15 ($u^{n+1} = R_h u^n + h p_n$), устойчивость определяется **ограниченностью норм степеней оператора перехода**.

Это условие выражается как $\| R_h^n \| \leq C$ (где C – константа) для всех n. Если норма R_h^n остаётся ограниченной, схема не усиливает ошибки при увеличении числа шагов.

$$\| u^n \| \leq \| R_h^n \| \| y^0 \| + h \sum_{k=0}^{n-1} \| R_h^{n-1-k} \| \| p_k \|,$$

что можно свести к выражению:

$$\| u^n \| \leq \max \| R_h^k \| (\| y^0 \| + Nh \max \| p_k \|).$$

Это показывает, что устойчивость зависит от максимальной нормы оператора перехода и накопления возмущений.

Рассмотрим **явную схему Эйлера** – простой и широко используемый метод:

Для уравнения $\frac{du}{dt} = f(t,u)$ с начальным условием $u(0) = u_0$ схема имеет вид:

$$\frac{u^{n+1} - u^n}{\tau} = f(t,u^n) \quad \text{или} \quad u^{n+1} = u^n + \tau f(t,u^n).$$

Для линеаризованной формы:

$$u^{n+1} = \left(1 + \tau \frac{\partial f}{\partial u}\right) u^n + \tau P_n.$$

Несмотря на простоту, устойчивость этой схемы ограничена: шаг τ должен быть малым, чтобы $|\lambda| = \left|1 + \tau \frac{\partial f}{\partial u}\right| \leq 1$, иначе решение может стать нестабильным.

Рассмотрим явную схему Эйлера для $\frac{du}{dt} = \lambda u$:

$$u^{n+1} = u^n + \tau\lambda u^n = (1 + \tau\lambda)u^n.$$

Здесь $R_h = 1 + \tau\lambda$, и $\lambda_i = 1 + \tau\lambda$. Условие устойчивости: $|1 + \tau\lambda| \leq 1$.

- Если $\lambda = -2$, $\tau = 1$, то $\lambda_i = 1 + 1 \cdot (-2) = -1$, $|\lambda_i| = 1$.
- Решение: $u^n = (-1)^n u^0$, которое колеблется, но остаётся ограниченным.
- Это случай нестрогой устойчивости, и корневые векторы не возникают, так как λ_i простое.

Если же R_h имеет кратное $\lambda_i = 1$, то рост n в решении допустим для нестрогой устойчивости, но для строгой устойчивости требуется $|\lambda_i| < 1$.

Для анализа устойчивости, особенно в нелинейных задачах, требуется **линеаризация**. Например, нелинейная функция $f(t, u^n)$ может аппроксимироваться в окрестности траектории u^* в виде:

$$f(t, u^n) = f(t, u^*) + \frac{\partial f}{\partial u}(u^n - u^*).$$

Это позволяет свести задачу к линейной форме $u^{n+1} = \left(1 + \tau \frac{\partial f}{\partial u}\right) u^n + \tau P_n$, где $P_n = f(t, u^*) - \frac{\partial f}{\partial u} u^*$. Линеаризация упрощает анализ собственных значений оператора перехода, что необходимо для оценки устойчивости, особенно для явной схемы Эйлера.

Возмущённое решение возникает при добавлении малых ошибок (например, в начальных условиях или правой части).

Такие ошибки могут появляться по разным причинам:

- *Округление чисел*: компьютеры работают с ограниченной точностью, и это вносит погрешности.
- *Приближение данных*: начальные или граничные условия часто задаются не точно, а с некоторой погрешностью.
- *Дискретизация*: когда непрерывную задачу (например, дифференциальное уравнение) заменяют дискретной (сеточной), это тоже добавляет неточности.

Основная идея в том, чтобы посмотреть, как эти маленькие ошибки влияют на итоговое решение. Если схема устойчива, то небольшие погрешности не превратятся в огромные отклонения. А если неустойчива — даже крошечные ошибки могут всё испортить.

Возмущённое решение u^* удовлетворяет $L_h u^* = f_h$, ошибка:

$$\Delta^n = u^n - u^* \ (u^* - \text{"идеальное" решение})$$

Подставим это уравнение в исходное, получим: $L_h \Delta^n = 0$.

Если ошибка Δ^n растёт со временем – схема неустойчива. Если затухает или остаётся маленькой – схема устойчива.

Если у нас есть возмущённое решение возмущение u^n отличающееся от u^*, тогда ошибка $\Delta^n = u^n - u^*$ будет развиваться как:

$$\Delta^{n+1} = \Delta^n + \tau[f(t, u^* + \Delta^n) - f(t, u^*)]$$

Если Δ^n маленькое, мы можем приблизить разницу $f(t, u^* + \Delta^n) - f(t, u^*)$ через производную:

$$f(t, u^* + \Delta^n) \approx f(t, u^*) + \frac{\partial f}{\partial u}\Delta^n$$

Тогда уравнение для ошибки упрощается до:

$$\Delta^{n+1} = \left(1 + \tau\frac{\partial f}{\partial u}\right)\Delta^n$$

Коэффициент $1 + \tau\frac{\partial f}{\partial u}$ называют *фактором усиления возмущений* и обозначают как λ:

$$\lambda = 1 + \tau\frac{\partial f}{\partial u}$$

Теперь всё сводится к значению λ:

- Если $|\lambda| \leq 1$, то ошибка Δ^n либо уменьшается, либо остаётся ограниченной. Это значит, что схема устойчива: малые возмущения не разрастаются.
- Если $|\lambda| > 1$, ошибка растёт с каждым шагом, и схема становится неустойчивой: даже крошечные погрешности быстро приведут к большим отклонениям.

Таким образом, проверяя $|\lambda|$, мы можем судить о надёжности схемы.

Приходим к понятию **спектрального признака устойчивости** – инструмента для анализа разностных схем. Спектральный признак утверждает, что схема устойчива, если все собственные значения λ_i матрицы R_h удовлетворяют условию:

$$|\lambda_i| \leq 1 \quad \text{(для строгой устойчивости: } |\lambda_i| < 1\text{)}.$$

Если $|\lambda_i| \leq 1$, возмущения в решении либо затухают, либо остаются ограниченными, что соответствует устойчивости.

Если же $|\lambda_i| > 1$, возмущения растут экспоненциально, и схема становится неустойчивой. Здесь нужно либо уменьшить шаг τ, либо выбрать другую схему.

Для задач с неопределённым числом шагов (например, моделирование долгосрочных процессов) требуется строгая устойчивость ($|\lambda_i| < 1$), чтобы ошибки затухали. Для задач с небольшим числом шагов достаточно нестрогой устойчивости ($|\lambda_i| \leq 1$).

Неявная схема Эйлера часто используется для повышения устойчивости по сравнению с явной схемой. Для уравнения $\frac{du}{dt} = f(t, u)$ с начальным условием $u(0) = u_0$, неявная схема Эйлера записывается как:

$$\frac{u^{n+1} - u^n}{\tau} = f(t_{n+1}, u^{n+1}),$$

где τ – шаг по времени. Это уравнение требует решения для u^{n+1} на каждом шаге, что делает метод более сложным в вычислительном плане, но зато значительно более устойчивым.

Рассмотрим линеаризованную форму: $f(t,u) = \frac{\partial f}{\partial u} u$. Схема принимает вид:

$$u^{n+1} - u^n = \tau \frac{\partial f}{\partial u} u^{n+1}.$$

Перегруппируем члены:

$$u^{n+1} - \tau \frac{\partial f}{\partial u} u^{n+1} = u^n \quad \Rightarrow \quad u^{n+1}\left(1 - \tau \frac{\partial f}{\partial u}\right) = u^n.$$

Тогда оператор перехода R_h:

$$u^{n+1} = \frac{u^n}{1 - \tau \frac{\partial f}{\partial u}}, \quad R_h = \frac{1}{1 - \tau \frac{\partial f}{\partial u}}.$$

Собственное значение оператора перехода $\lambda = \frac{1}{1-\tau\frac{\partial f}{\partial u}}$.

Для устойчивости необходимо $|\lambda| \leq 1$. Если $\frac{\partial f}{\partial u} < 0$ (часто встречается в задачах затухания), то знаменатель $1 - \tau \frac{\partial f}{\partial u} > 1$, и $|\lambda| < 1$. Это означает, что неявная схема Эйлера устойчива даже при больших шагах τ, в отличие от явной схемы, где шаг ограничен условием $\tau \leq \frac{2}{\left|\frac{\partial f}{\partial u}\right|}$.

Теперь обратимся к схеме с центральной разностью, которая часто используется для задач с производными второго порядка. Рассмотрим уравнение теплопроводности $\frac{\partial \mathrm{u}}{\partial \mathrm{t}} = \frac{\partial^2 \mathrm{u}}{\partial \mathrm{x}^2}$, в котором скорость изменения температуры во времени пропорциональна второй производной по пространству, что описывает, как тепло "размазывается" от горячих областей к холодным. Схема с центральной разностью по пространству и явным шагом по времени:

$$\frac{u^{n+1} - u^n}{\tau} = \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{h^2},$$

где h – шаг по пространству, i – индекс узла сетки. Перепишем это уравнение:

$$u_i^{n+1} = u_i^n + \frac{\tau}{h^2}(u_{i+1}^n - 2u_i^n + u_{i-1}^n).$$

Обозначим $\alpha = \frac{\tau}{h^2}$, тогда:

$$u_i^{n+1} = \alpha u_{i+1}^n + (1 - 2\alpha)u_i^n + \alpha u_{i-1}^n.$$

Для анализа устойчивости используем метод фон Неймана: предположим, что решение имеет вид $u_i^n = \xi^n e^{ikx_i}$, где k – волновое число, $x_i = ih$. Подставим в уравнение:

$$\xi^{n+1} e^{ikx_i} = \alpha \xi^n e^{ikx_{i+1}} + (1 - 2\alpha)\xi^n e^{ikx_i} + \alpha \xi^n e^{ikx_{i-1}}.$$

Учитывая $x_{i+1} = (i+1)h$, $x_{i-1} = (i-1)h$, делим на $\xi^n e^{ikx_i}$:

$$\xi = \alpha e^{ikh} + (1 - 2\alpha) + \alpha e^{-ikh}.$$

Используя $e^{ikh} + e^{-ikh} = 2\cos(kh)$, получаем:

$$\xi = 1 - 2\alpha + 2\alpha\cos(kh) = 1 - 2\alpha\big(1 - \cos(kh)\big).$$

Так как $1 - \cos(kh) = 2\sin^2(kh/2)$, то:

$$\xi = 1 - 4\alpha \sin^2\left(\frac{kh}{2}\right).$$

$$\xi = 1 - 4\alpha \sin^2\left(\frac{kh}{2}\right).$$

Фактор усиления ξ – это собственное значение оператора перехода для данной волны. Для устойчивости нужно $|\xi| \leq 1$:

$$-1 \leq 1 - 4\alpha \sin^2\left(\frac{kh}{2}\right) \leq 1.$$

Левое неравенство: $1 - 4\alpha \sin^2\left(\frac{kh}{2}\right) \geq -1$:

$$4\alpha \sin^2\left(\frac{kh}{2}\right) \leq 2 \quad \Rightarrow \quad \alpha \leq \frac{1}{2\sin^2\left(\frac{kh}{2}\right)}.$$

Максимум $\sin^2\left(\frac{kh}{2}\right) = 1$, поэтому $\alpha \leq \frac{1}{2}$, или:

$$\tau \leq \frac{h^2}{2}.$$

Правое неравенство ($1 - 4\alpha \sin^2 \leq 1$) выполняется автоматически. Таким образом, схема с центральной разностью устойчива только при $\tau \leq \frac{h^2}{2}$, что накладывает строгие ограничения на шаг по времени.

Решение обыкновенных дифференциальных уравнений второго (и выше) порядка с помощью метода Рунге-Кутта.

К решению систем уравнений сводятся также задача Коши для уравнений высших порядков. Рассмотрим задачу Коши для уравнений второго порядка:

$$\begin{cases} \dfrac{d^2y}{dx^2} = f(x, y, y'), \\ y(x_0) = y_0,\ y'(x_0) = z_0 \end{cases}$$

Данную задачу можно свести к системе обыкновенных дифференциальных уравнений первого порядка:

$$\begin{cases} \dfrac{dz}{dx} = f(x, y, z), \\ \dfrac{dy}{dx} = z, \\ y(x_0) = y_0,\ z(x_0) = z_0, \end{cases}$$

Однозначная взаимосвязь причины и следствия: если задано некоторое начальное состояние системы при $t = t_0$, то оно однозначно определяет состояние системы в любой момент времени $t > t_0$.

Пример, равноускоренное движение

$$v(t) = v(t_0) + at$$

Особенностью одношаговых методов является то, что *для получения решения в каждом новом расчетном узле достаточно иметь значение сеточной функции лишь в предыдущем узле*.

Это позволяет непосредственно начать счет при i = 0 по известным начальным значениям.

Эта особенность допускает изменение шага в любой точке в процессе счета, что позволяет строить численные алгоритмы с автоматическим выбором шага.

Задача Коши

$$\frac{dy}{dx} = f(x, y)$$

$$y(x_0) = y_0$$

Для вычисления значений y_{i+1} с помощью k-шагового метода используются результаты не одного, а k предыдущих шагов, т.е. значения y_i, y_{i-1}, y_{i-2} ,..., y_{i-k+1} .

Многошаговые методы могут быть построены следующим образом.

Запишем исходное уравнение $\frac{dy}{dx} = f(x, y)$ в виде

$$dy(x) = f(x, y)dx$$

Проинтегрируем обе части этого уравнения по x на отрезке $[x_i , x_{i+1}]$:

$$\int_{x_i}^{x_{i+1}} dy(x) = \int_{x_i}^{x_{i+1}} f(x, y)dx$$

$$\int_{x_i}^{x_{i+1}} dy(x) = \int_{x_i}^{x_{i+1}} f(x,y)dx$$

Интеграл от левой части вычисляется легко.

$$\int_{x_i}^{x_{i+1}} dy(x) = y(x_{i+1}) - y(x_i) \approx y_{i+1} - y_i$$

Для вычисления интеграла от правой части уравнения сначала строится интерполяционный многочлен $P_{k-1}(x)$ степени $(k - 1)$ для аппроксимации функции $f(x, y)$ на отрезке $[x_i , x_{i+1}]$ по значениям

$$f(x_{i-k+1}, y_{i-k+1}), f(x_{i-k+2}, y_{i-k+2}), \ldots, f(x_i, y_i)$$

После этого можно положить, что

$$\int_{x_i}^{x_{i+1}} f(x,y)dx \approx \int_{x_i}^{x_{i+1}} P_{k-1}(x)dx.$$

$$\int_{x_i}^{x_{i+1}} dy(x) = y(x_{i+1}) - y(x_i) \approx y_{i+1} - y_i$$

$$\int_{x_i}^{x_{i+1}} f(x,y)dx \approx \int_{x_i}^{x_{i+1}} P_{k-1}(x)dx.$$

Приравнивая эти выражения, получаем формулу для определения неизвестного значения сеточной функции y_{i+1} в узле x_{i+1}:

$$y_{i+1} = y_i + \int_{x_i}^{x_{i+1}} P_{k-1}(x)dx$$

На основе этой формулы можно строить различные многошаговые методы любого порядка точности. Порядок точности зависит от степени интерполяционного многочлена $P_{k-1}(x)$, для построения которого используются значения сеточной функции y_i, y_{i-1} , … , y_{i-k+1} вычисленные на k предыдущих шагах.

Задача Коши

$$\frac{dy}{dx} = f(x, y)$$

$$y(x_0) = y_0$$

Широко распространенным семейством многошаговых методов являются методы Адамса. Простейший из них, получающийся при $k = 1$, совпадает с рассмотренным ранее методом Эйлера первого порядка точности.

В практических расчетах чаще всего используется вариант метода Адамса 4-го порядка. Именно его обычно и называют методом Адамса. Рассмотрим этот метод.

Явный метод Адамса: Использует предыдущие значения y_n, y_{n-1}, ... для аппроксимации следующего значения y_{n+1}. Формула для трехшагового метода Адамса-Башфорта:

$$y_{n+1} = y_n + \frac{h}{12}(23f(t_n, y_n) - 16f(t_{n-1},) + 5f(t_{n-2}, y_{n-2}))$$

Неявный метод Адамса: Использует текущие и будущие значения для более точного результата. Формула для трехшагового метода Адамса-Мултона:

$$y_{n+1} = y_n + \frac{h}{12}((5f(t_{n+1}, y_{n+1}) + 8f(t_n, y_n) - f(t_{n-1}, y_{n-1})))$$

Явные методы

Преимущества: Простота реализации и вычислительная эффективность.

Недостатки: Ограниченная стабильность, особенно для жестких систем.

Неявные методы

Преимущества: Более высокая стабильность, подходящие для жестких систем.

Недостатки: Более сложная реализация и необходимость решения нелинейных уравнений на каждом шаге.

Задача Коши

$$\frac{dy}{dx} = f(x,y)$$

$$y(x_0) = y_0$$

$$y_{i+1} = y_i + \int_{x_i}^{x_{i+1}} P_{k-1}(x)dx$$

Пусть найдены значения y_{i-3}, y_{i-2}, y_{i-1}, y_i в четырех последовательных узлах (k = 4). При этом имеются также вычисленные ранее значения правой части уравнения

$$f_{i-3} = f(x_{i-3}, y_{i-3});$$
$$f_{i-2} = f(x_{i-2}, y_{i-2});$$
$$f_{i-1} = f(x_{i-1}, y_{i-1});$$
$$f_i = f(x_i, y_i)$$

В качестве интерполяционного $P_3(x)$ можно взять многочлен Ньютона. В случае постоянного шага h конечные разности для правой части в узле x_i имеют вид:

$$\Delta f_i = f_i - f_{i-1},$$
$$\Delta^2 f_i = f_i - 2f_{i-1} + f_{i-2},$$
$$\Delta^3 f_i = f_i - 3f_{i-1} + 3f_{i-2} - f_{i-3}.$$

Разностная схема четвертого порядка метода Адамса записывается в виде:

$$y_{i+1} = y_i + hf_i + \frac{h^2}{2}\Delta f_i + \frac{5h^3}{12}\Delta^2 f_i + \frac{3h^4}{8}\Delta^3 f_i$$

Сравнивая метод Адамса с методом Рунге-Кутта той же точности, отмечаем его экономичность, поскольку он требует вычисления лишь одного значения правой части на каждом шаге (метод Рунге-Кутта – четырех).

При этом, метод Адамса неудобен тем, что невозможно начать счет по одному лишь известному значению y_0 . Расчет может быть начат лишь с узла x_3.

Задача Коши

$$\frac{dy}{dx} = f(x, y)$$

$$y(x_0) = y_0$$

$$f_{i-3} = f(x_{i-3}, y_{i-3});$$
$$f_{i-2} = f(x_{i-2}, y_{i-2});$$
$$f_{i-1} = f(x_{i-1}, y_{i-1});$$
$$f_i = f(x_i, y_i)$$

$$\Delta f_i = f_i - f_{i-1},$$
$$\Delta^2 f_i = f_i - 2f_{i-1} + f_{i-2},$$
$$\Delta^3 f_i = f_i - 3f_{i-1} + 3f_{i-2} - f_{i-3}.$$

Значения y_1, y_2, y_3, необходимые для вычисления y_4, нужно получить каким-либо другим способом (например, методом Рунге-Кутта), что существенно усложняет алгоритм.

Кроме того, метод Адамса не позволяет (без усложнения формул) изменить шаг h в процессе счета; этого недостатка лишены одношаговые методы.

Метод Адамса легко распространяется на системы дифференциальных уравнений.

Рассмотрим еще одно семейство многошаговых методов, которые используют неявные схемы, – **метод прогноза и коррекции** (они называются также методами **предиктор-корректор**). Суть этих методов состоит в следующем.

На каждом шаге вводятся два этапа, использующих многошаговые методы:

1) с помощью явного метода (**предиктора**) по известным значениям функции в предыдущих узлах находится начальное приближение $y_{i+1} = y_{i+1}^{(0)}$ в новом узле.

2) используя неявный метод (**корректор**), в результате итераций находятся приближения $y_{i+1}^{(1)}, y_{i+1}^{(2)}, \ldots$

Один из вариантов метода прогноза и коррекции может быть получен на основе метода Адамса четвертого порядка:

на этапе предиктора

$$y_{i+1} = y_i + \frac{h}{24}(55f_i - 59f_{i-1} + 37f_{i-2} - 9f_{i-3})$$

на этапе корректора

$$y_{i+1} = y_i + \frac{h}{24}(9f_{i+1} + 19f_i - 5f_{i-1} + f_{i-2})$$

Явная схема используется на каждом шаге один раз, а с помощью неявной схемы строится итерационный процесс вычисления y_{i+1}, поскольку это значение входит в правую часть выражения $f_{i+1} = f(x_{i+1}, y_{i+1})$. Расчет по этому методу может быть начат только со значения y_4. Необходимые при этом y_1, y_2, y_3 находятся по методу Рунге-Кутта, y_0 задается начальным условием.

Задача Коши

$$\frac{dy}{dx} = f(x, y)$$

$$y(x_0) = y_0$$

$$f_{i-3} = f(x_{i-3}, y_{i-3});$$
$$f_{i-2} = f(x_{i-2}, y_{i-2});$$
$$f_{i-1} = f(x_{i-1}, y_{i-1});$$
$$f_i = f(x_i, y_i)$$

$$\Delta f_i = f_i - f_{i-1},$$
$$\Delta^2 f_i = f_i - 2f_{i-1} + f_{i-2},$$
$$\Delta^3 f_i = f_i - 3f_{i-1} + 3f_{i-2} - f_{i-3}.$$

Точность численного решения можно повысить различными способами. Например, путем уменьшения значения шага h. Однако этот путь ограничен требованием экономичности, поскольку получение решения с необходимой точностью может потребовать огромного объема вычислений.

На практике часто для повышения точности численного решения без существенного увеличения машинного времени используется **метод Рунге**. Он состоит в том, что проводятся повторные расчеты по одной разностной схеме с различными шагами. Уточненное решение в совпадающих при разных расчетах узлах строится с помощью проведенной серии расчетов.

Предположим, что проведены две серии расчетов по схеме порядка k соответственно с шагами h и h/2. В результате расчетов получены множества значений сеточной функции y_h и $y_{h/2}$. Тогда в соответствии с методом Рунге уточненное значение y_h^* сеточной функции в узлах сетки с шагом h вычисляется по формуле

$$y_h^* = \frac{2^k y_{h/2} - y_h}{2^k - 1} + O(h^{k+1})$$

Порядок точности этого решения равен *k+1*, хотя используемая разностная схема имеет порядок точности *k*. Таким образом, решение задачи на двух сетках позволяет на порядок повысить точность результатов.

Для схемы Эйлера первого порядка точности (k = 1) формула Рунге принимает вид:

$$y_h^* = 2y_{h/2} - y_h + O(h^2)$$

Аналогично можно записать формулу для уточнения решения, получаемого по методу Рунге-Кутта при k = 4.

Численная устойчивость – это свойство численного метода, которое гарантирует, что небольшие ошибки, возникающие из-за округления чисел, дискретизации или неточностей в начальных данных, не приводят к катастрофическому росту погрешностей в решении. Без устойчивости даже точная аппроксимация уравнения может дать неверные результаты из-за накопления ошибок.

Согласованность метода означает, что разностная схема правильно приближает исходное дифференциальное уравнение при уменьшении шага дискретизации h. Рассмотрим задачу Коши для уравнения $\frac{dy}{dx} = f(x, y)$ с начальным условием $y(x_0) = y_0$. Разностная схема аппроксимирует это уравнение, и её точность измеряется через *невязку* ρ_{n+k}, которая получается при подстановке точного решения $y(x)$ в разностное уравнение.

Формально, метод согласован, если максимальная невязка стремится к нулю:

$$\max_{0 \le n \le N} \| \rho_{n+k} \| \to 0 \quad \text{при} \quad h \to 0.$$

Невязка определяется как:

$$\rho_{n+k} = \sum_{i=0}^{k} \alpha_i \, y(x_{n+i}) - h \sum_{i=0}^{n} \beta_i \, f\big(x_{n+i}, y(x_{n+i})\big),$$

где α_i и β_i – коэффициенты метода, h – шаг дискретизации, а $y(x_{n+i})$ – точное решение в точке x_{n+i}.

Если $\| \rho_{n+k} \| = O(h^{s+1})$, то метод имеет **порядок согласованности** s. Число s называется порядком аппроксимации, а $O(h^{s+1})$ – погрешностью дискретизации. Например, метод Эйлера имеет $s = 1$, так как его невязка уменьшается пропорционально h^2.

Нуль-устойчивость (или нулевая устойчивость) проверяет, как метод ведёт себя в пределе $h \to 0$, когда внешние возмущения отсутствуют. Она связана с характеристическим полиномом разностной схемы, который для линейного мультишагового метода вида:

$$\sum_{i=0}^{k} \alpha_i \, y_{n+i} = h \sum_{i=0}^{n} \beta_i \, f\left(x_{n+i}, y(x_{n+i})\right)$$

записывается как:

$$p(\theta) = \sum_{i=0}^{k} \alpha_i \, \theta^i.$$

Метод нуль-устойчив, если все корни θ_i полинома $p(\theta)$ удовлетворяют условиям:

- $|\theta_i| \leq 1$ (корни лежат внутри или на единичной окружности),
- Корни на единичной окружности ($|\theta_i| = 1$) являются простыми (не кратными).

Метод нуль-устойчив, если все корни θ_i полинома $p(\theta)$ удовлетворяют условиям:

- $|\theta_i| \leq 1$ (корни лежат внутри или на единичной окружности),
- Корни на единичной окружности ($|\theta_i| = 1$) являются простыми (не кратными).

Классификация корней:

- Главный корень: $\theta_1 = 1$, который всегда присутствует у согласованного метода.
- Посторонние корни: $|\theta_i| \leq 1,\ i = 2,3, \dots, k$, которые не должны вызывать неустойчивость.

Нуль-устойчивость определяет, как погрешности аппроксимации и другие ошибки развиваются при $h \to 0$ и фиксированном интервале $x_k - x_0 = Nh$. Если метод не нуль-устойчив, ошибки могут расти даже при уменьшении шага.

Сходимость – свойство метода, при котором численное решение y_n приближается к точному решению $y(x)$ при $h \to 0$:

$$y_n \to y(x) \quad \text{при} \quad h \to 0, \quad n = \frac{x - x_0}{h}.$$

Полная погрешность в узле x_n определяется как $y_n - y(x_n)$, и метод сходится, если эта погрешность стремится к нулю для всех $x \in [x_0, x_k]$.

Ключевая теорема (аналог теоремы Лакса-Рябенького-Филипова для мультишаговых методов) утверждает, что метод класса линейных мультишаговых методов сходится тогда и только тогда, когда он:

- Согласован.
- Нуль-устойчив.

Рассмотрим явный метод Эйлера для уравнения $\frac{dy}{dx} = f(x, y)$:

$$y_{n+1} = y_n + hf(x_n, y_n).$$

Его характеристический полином:

$$p(\theta) = \theta - 1,$$

с единственным корнем $\theta = 1$, который прост и лежит на единичной окружности. Метод нуль-устойчив. Невязка метода Эйлера имеет порядок $O(h^2)$, что даёт порядок согласованности $s = 1$. Таким образом, метод Эйлера сходится.

Понимание согласованности и нуль-устойчивости позволяет выбирать методы, которые будут давать точные и надёжные результаты. Например, методы с высоким порядком согласованности (большое s) точнее, но могут быть сложнее в реализации. Нуль-устойчивость же защищает от неконтролируемого роста ошибок.

Понятие	Описание
Согласованность	Невязка ρ_{n+k} стремится к нулю при $h \to 0$, с порядком $O(h^{s+1})$.
Нуль-устойчивость	Корни характеристического полинома $p(\theta)$ лежат внутри или на единичной окружности, с простыми корнями на окружности.
Сходимость	Численное решение y_n приближается к точному $y(x)$ при $h \to 0$, если метод согласован и нуль-устойчив.

Краевые задачи – это задачи для обыкновенных дифференциальных уравнений (ОДУ), в которых условия задаются на границах области, например, на концах интервала $[a, b]$. Рассмотрим типичную краевую задачу для уравнения второго порядка:

$$L[y] = f(x), \quad a < x < b,$$

где $L[y] = a_2(x)y'' + a_1(x)y' + a_0(x)y$ – линейный дифференциальный оператор второго порядка, а $f(x)$ – заданная функция (неоднородность). Граничные условия обычно имеют вид:

$$\alpha_1 y(a) + \beta_1 y'(a) = 0, \quad \alpha_2 y(b) + \beta_2 y'(b) = 0,$$

где α_i и β_i – константы, определяющие тип условий (например, Дирихле, Неймана или смешанные).

Краевые задачи отличаются от задач Коши (или начальных задач) способом задания условий:

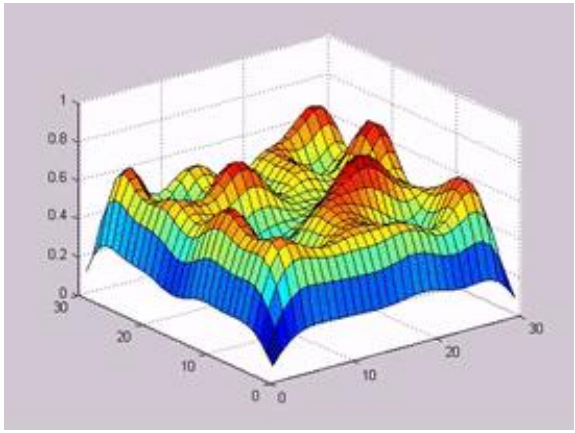
- **Задачи Коши**: Условия задаются в одной точке, например, $y(x_0) = y_0$, $y'(x_0) = y_0{}'$. Это подходит для процессов, развивающихся во времени, таких как движение тела или химическая реакция.
- **Краевые задачи**: Условия задаются на границах интервала, например, $(a) = 0$, $y(b) = 0$. Это типично для стационарных процессов или систем, ограниченных в пространстве, таких как температура в стержне или деформация балки.

Краевые задачи сложнее, так как решение должно одновременно удовлетворять условиям на обоих концах интервала, что может привести к отсутствию решения или его неуникальности в зависимости от оператора и условий.

Краевые задачи находят широкое применение в реальных задачах:

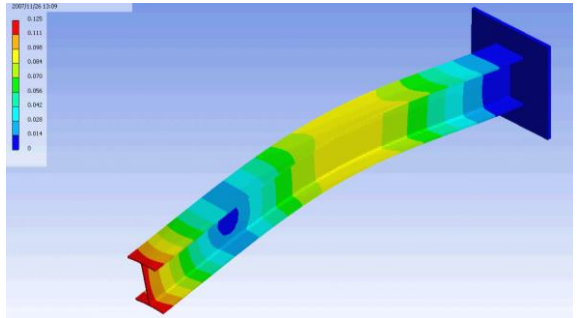
- **Теплопроводность**: Определение стационарного распределения температуры в стержне с фиксированными температурами на концах.

$$\frac{\partial u}{\partial t} - a^2 \left(\frac{\partial^2 u}{\partial x_1^2} + \frac{\partial^2 u}{\partial x_2^2} + \cdots + \frac{\partial^2 u}{\partial x_n^2} \right) = f(x,t).$$

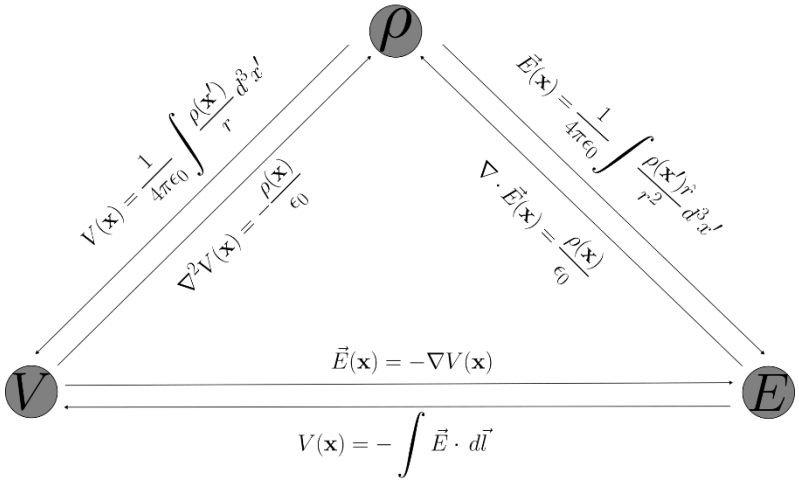


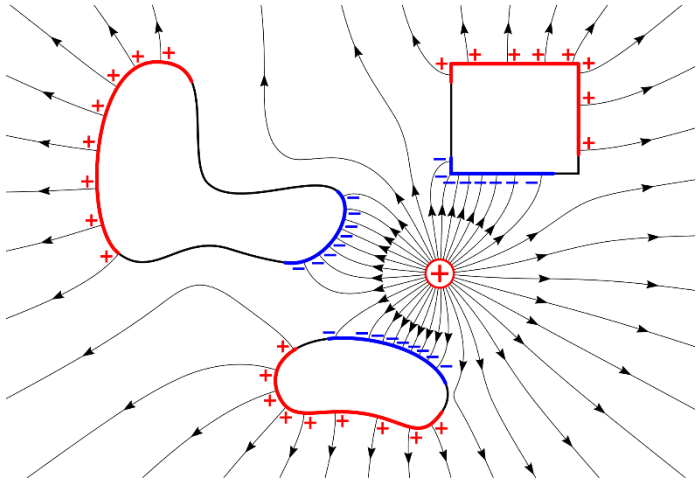
- **Механика**: Расчёт деформации балки, закреплённой на концах.

$$S = \int_{t_1}^{t_2} \int_0^L \left[\frac{1}{2}\mu \left(\frac{\partial w}{\partial t} \right)^2 - \frac{1}{2} EI \left(\frac{\partial^2 w}{\partial x^2} \right)^2 + q(x) w(x,t) \right] dx dt.$$



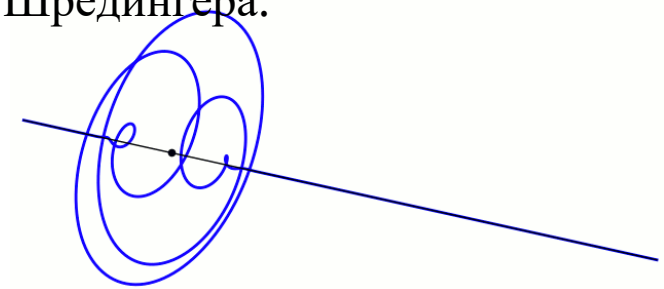
- **Электродинамика**: Нахождение электрического потенциала в области с заданными потенциалами на границах.





- **Квантовая механика**: Решение уравнения Шрёдингера.

$$i\hbar\frac{\partial}{\partial t}\Psi(x,t) = \left[-\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2} + V(x,t)\right]\Psi(x,t).$$



Эти задачи требуют численных методов, таких как метод конечных разностей или метод конечных элементов, для получения приближённых решений, особенно когда аналитическое решение невозможно.

Функция Грина – мощный инструмент для решения краевых задач, позволяющий выразить решение в виде интеграла, что упрощает анализ для различных внешних воздействий. Для задачи:

$$L[y] = f(x), \quad a < x < b,$$

с граничными условиями:

$$\alpha_1 y(a) + \beta_1 y'(a) = 0, \quad \alpha_2 y(b) + \beta_2 y'(b) = 0,$$

решение записывается как:

$$y(x) = \int_a^b G\,(x, s) f(s)\, ds,$$

где $G(x, s)$ — функция Грина, зависящая от точки наблюдения x и точки источника s.

Функция Грина удовлетворяет следующим свойствам:

- $L[G(x,s)] = \delta(x-s)$, где $\delta(x-s)$ — дельта-функция Дирака.
- $G(x,s)$ непрерывна при $x = s$: $G(s-,s) = G(s+,s)$.
- Первая производная $G_x(x,s)$ имеет разрыв при $x = s$: $G_x(s+,s) - G_x(s-,s) = \frac{1}{a_2(s)}$, где $a_2(s)$ — коэффициент при второй производной в L.
- $G(x,s)$ удовлетворяет граничным условиям: $\alpha_1 G(a,s) + \beta_1 G_x(a,s) = 0$, $\alpha_2 G(b,s) + \beta_2 G_x(b,s) = 0$.

Для построения функции Грина выполняем следующие шаги:

1. Решаем однородное уравнение $L[y] = 0$, находя два линейно независимых решения $y_1(x)$ и $y_2(x)$.
2. Записываем $G(x, s)$ в виде:

$$G(x,s) = \begin{cases} a(s)y_1(x) + b(s)y_2(x), & a < x < s \\ c(s)y_1(x) + d(s)y_2(x), & s < x < b \end{cases}$$

3. Определяем коэффициенты $a(s)$, $b(s)$, $c(s)$, $d(s)$ из системы уравнений, включающей:
   - Граничные условия: $\alpha_1 G(a,s) + \beta_1 G_x(a,s) = 0$, $\alpha_2 G(b,s) + \beta_2 G_x(b,s) = 0$.
   - Условие непрерывности: $G(s-,s) = G(s+,s)$.
   - Условие разрыва производной: $G_x(s+,s) - G_x(s-,s) = \frac{1}{a_2(s)}$.

Рассмотрим задачу:

$$y'' = f(x), \quad 0 < x < 1, \quad y(0) = y(1) = 0.$$

1. Проверяем тривиальность решения однородной задачи ($f(x) \equiv 0$):

$$y'' = 0 \quad \Rightarrow \quad y(x) = C_1 x + C_2.$$

Применяем граничные условия:

$$y(0) = C_2 = 0, \quad y(1) = C_1 + C_2 = 0 \quad \Rightarrow \quad C_1 = 0, \quad C_2 = 0.$$

Таким образом, $y(x) \equiv 0$, и функция Грина существует.

2. Находим решения однородного уравнения: $y_1(x) = x$, $y_2(x) = 1$.

3. Записываем функцию Грина:

$$G(x,s) = \begin{cases} a(s)x + b(s), & 0 < x < s \\ c(s)x + d(s), & s < x < 1 \end{cases}$$

4. Решаем систему для коэффициентов, используя граничные условия, непрерывность и разрыв производной. В результате получаем:

$$G(x,s) = \begin{cases} x(1-s), & 0 \le x \le s \\ s(1-x), & s \le x \le 1 \end{cases}$$

5. Решение задачи:

$$y(x) = \int_0^1 G\,(x,s) f(s)\,ds.$$

Характеристика	Задача Коши	Краевая задача
Условия	Заданы в одной точке $(y(x_0), y'(x_0))$	Заданы на границах $(y(a), y(b))$
Применение	Динамические процессы (движение, реакции)	Стационарные процессы (температура, деформация)
Методы решения	Явные/неявные схемы (Эйлера, Рунге-Кутта)	Функции Грина, конечные разности, элементы

Изменение во времени состояния системы является случайным (его нельзя однозначно предсказать) и невоспроизводимым (процесс нельзя повторить). Пример: броуновская частица.

Понятия детерминизм и хаос прямо противоположны по смыслу. Детерминизм ассоциируется с полной предсказуемостью и воспроизводимостью, хаос – с полной непредсказуемостью и невоспроизводимостью.

Характеристики хаотического движения:

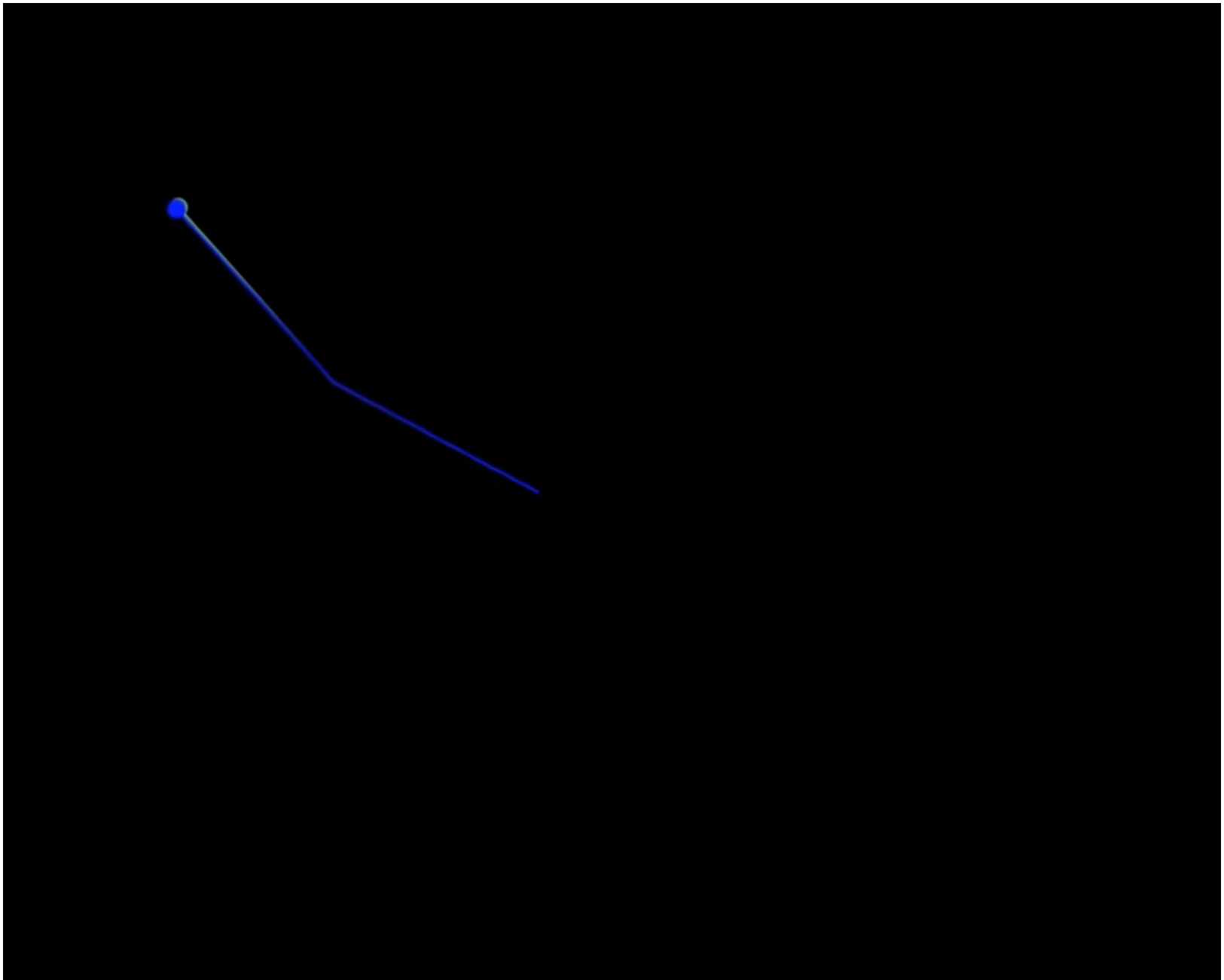
- Чувствительность к начальным условиям
- Топологическое смешивание
- Тонкости определения

Теория хаоса — раздел математики, изучающий системы, которые на первый взгляд кажутся упорядоченными, но на самом деле обладают хаотическим поведением.

Она также изучает системы, которые внешне кажутся хаотичными, но на самом деле содержат скрытый порядок.

Теория хаоса исследует нелинейные динамические системы, которые крайне чувствительны к начальным условиям. Этот эффект часто называют эффектом бабочки.

Детерминированная природа таких систем не делает их предсказуемыми. Это поведение называется детерминированным хаосом или просто хаосом.



https://web.mit.edu/jorloff/www/chaosTalk/double-pendulum/double-pendulum-en.html
https://visualize-it.github.io/double_pendulum/simulation.html

https://www.myphysicslab.com/pendulum/double-pendulum-en.html

Казалось бы, структурно неустойчивые в критических точках функции непригодны для описания реальности. Но, как правило, функции, возникающие в физических приложениях, содержат некоторые параметры, значения которых могут изменяться в определенном диапазоне. В таких случаях мы имеем дело с семейством функций, зависящим от параметра. Может случиться, что при изменении последнего с неизбежностью достигается значение, соответствующее структурно неустойчивой критической точке, которая тем самым приобретает вполне реальный смысл. Более того, именно эта точка, будучи одной из реализаций семейства критических точек, является наиболее важной, поскольку с ней связаны качественные изменения в поведении системы.

Появление топологически неэквивалентных фазовых портретов при изменении параметров динамической системы называется **бифуркацией**.

Бифуркация — это качественное изменение поведения динамической системы при бесконечно малом изменении её параметров, когда ее параметры проходят через некоторые бифуркационные (критические) значения. Эти значения еще называют точками бифуркации.

Центральным понятием теории бифуркации является понятие (не)грубой системы. Берётся какая-либо динамическая система и рассматривается такое (много)параметрическое семейство динамических систем, что исходная система получается в качестве частного случая — при каком-либо одном значении параметра (параметров). Если при значении параметров, достаточно близких к данному, сохраняется качественная картина разбиения фазового пространства на траектории, то такая система называется грубой. В противном случае, если такой окрестности не существует, то система называется негрубой.

Термин "бифуркация" буквально означает "раздвоение", но обычно применяется в более широком смысле для обозначения всевозможных качественных перестроек различных объектов при изменении параметров, от которых они зависят. Общая задача исследования точек бифуркации как математическая проблема состоит в их классификации и анализе поведения семейств функций вблизи структурно неустойчивых критических точек. Понятие бифуркации позволяет глубже проникнуть в сущность структурной неустойчивости, выявляя ее следствия.

Параметр, изменение которого приводит к бифуркации, называется критическим параметром (**бифуркационным параметром**), а значение этого параметра, при котором происходит бифуркация, называется **критическим значением**.

Точка в параметрическом пространстве (пространстве, в котором каждой точке соответствует определённое состояние системы, а положение этой точки определяется значениями параметров и переменных состояния), в которой происходит бифуркация, называется **точкой бифуркации**. Из точки бифуркации могут исходить несколько решений (устойчивых и неустойчивых).

Точка бифуркации, из которой все исходящие решения устойчивы, называется **точкой притяжения** (или **аттрактором**).

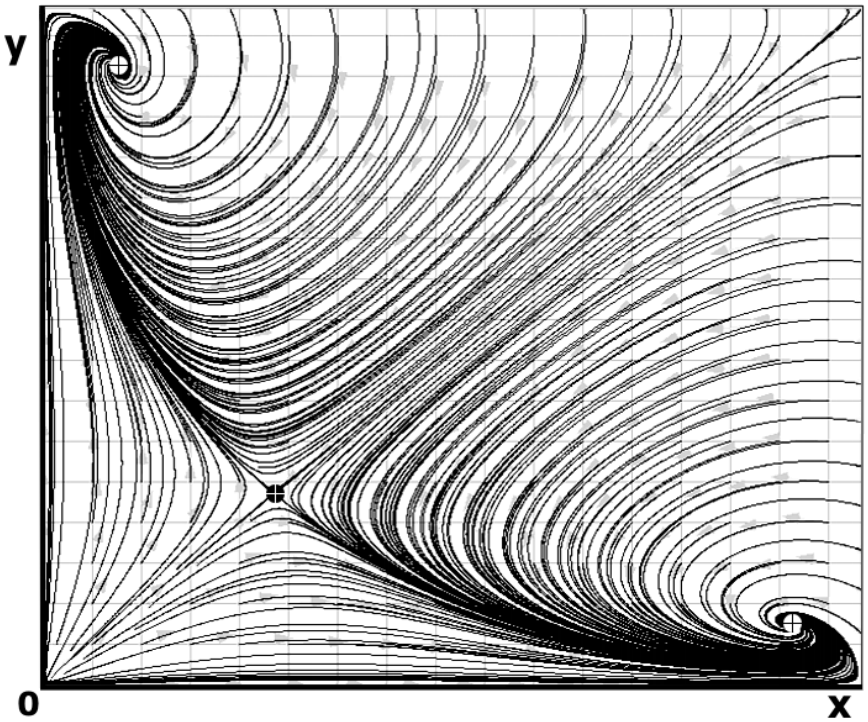
Представление любого характеристического свойства решения как функции критического параметра, называется бифуркационной диаграммой.

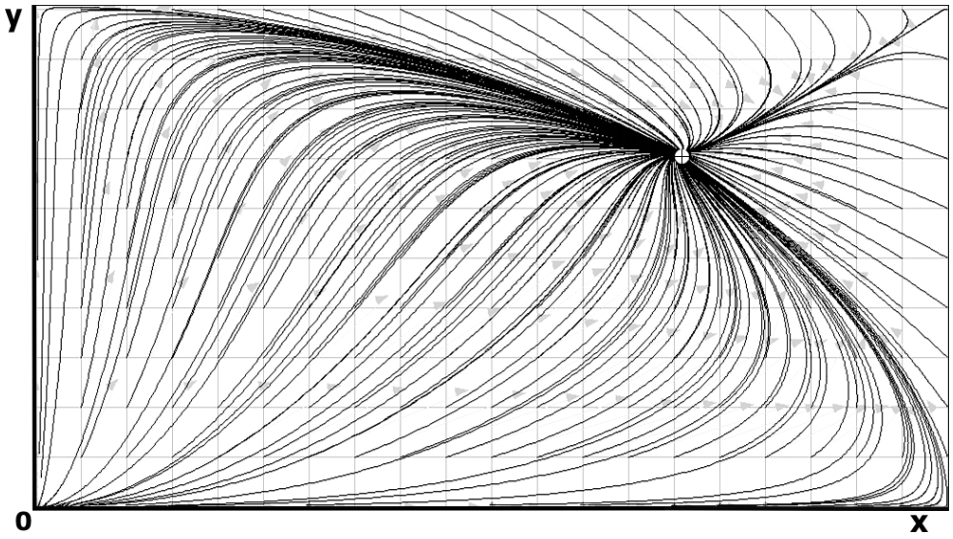
Суперкритической (нормальной, надкритической) называется бифуркация, при которой изменение системы происходит без скачка.

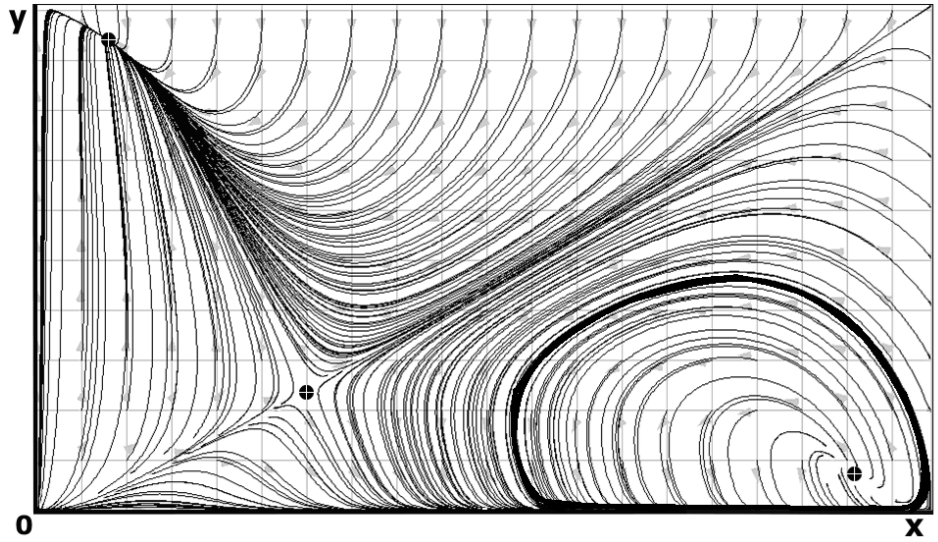
Субкритической (обратной) называется бифуркация, при которой изменение системы происходит скачком.

Последовательность бифуркаций, качественно меняющих свойства системы, называется сценарием.

Фазовый портрет

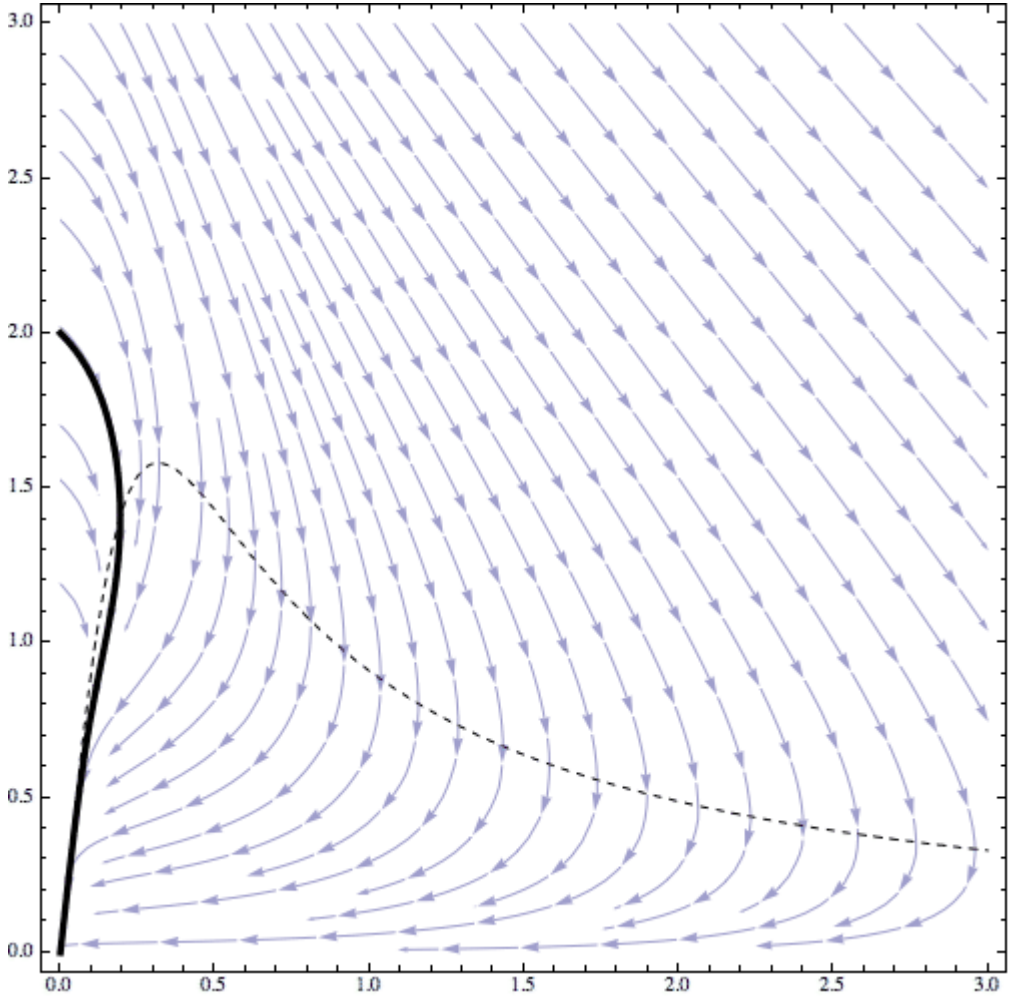




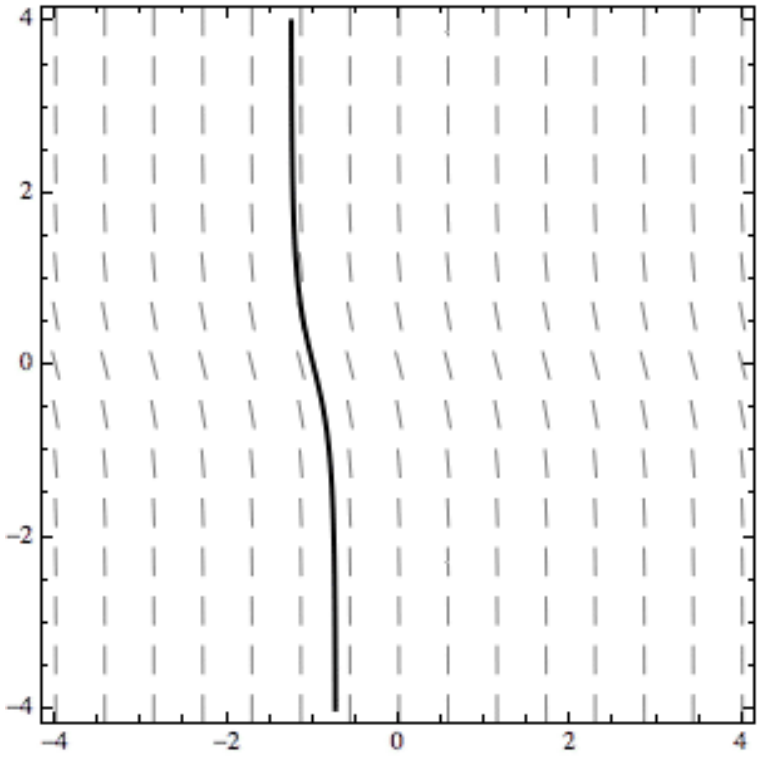


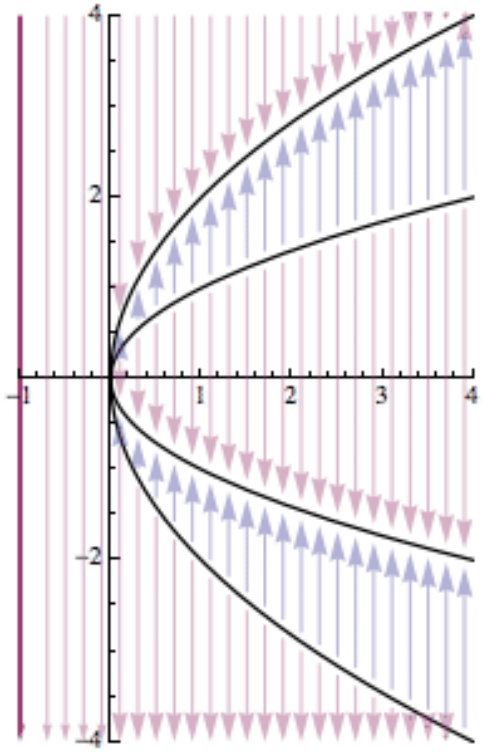
$$x' = -x + ay + x^2 y$$

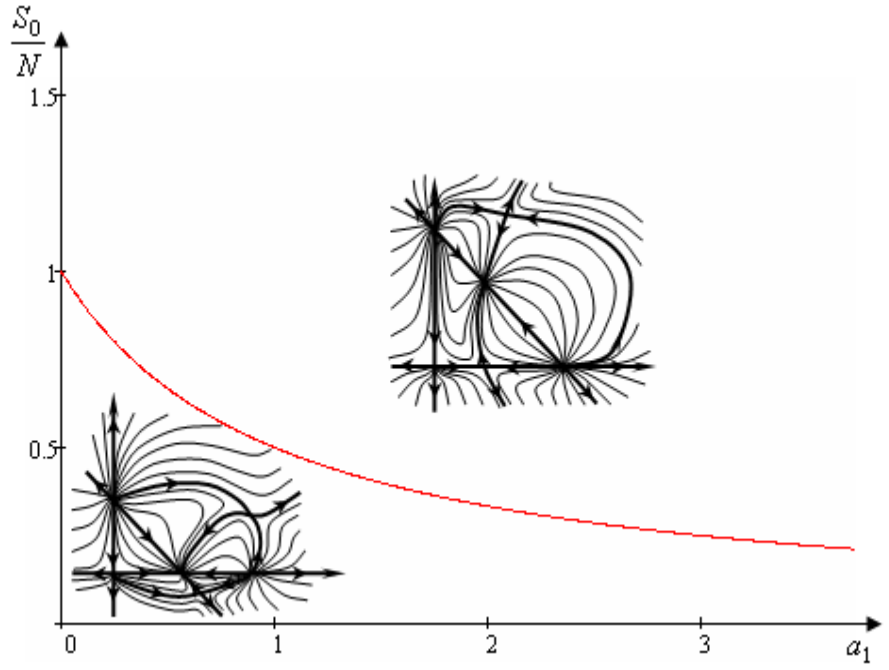
$$y' = b - ay - x^2 y$$

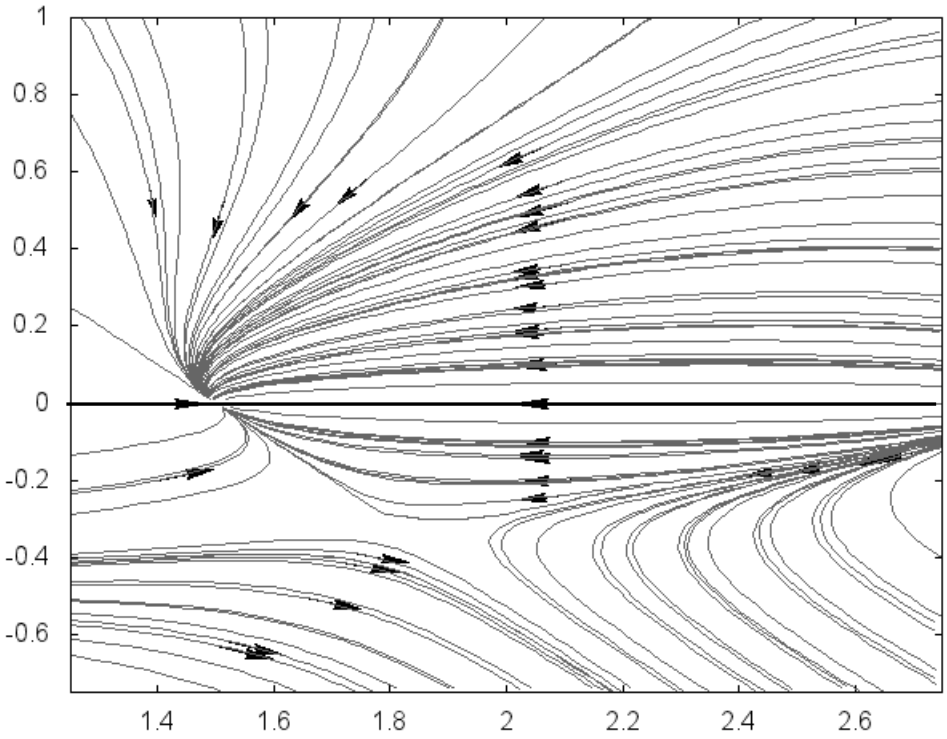


$$x'(y) = -(x(y))^4 + 5a(x(y))^2 - 4a^2$$






$\frac{S_0}{N}$
1.5
1
0.5
0
1
2
3
a_1

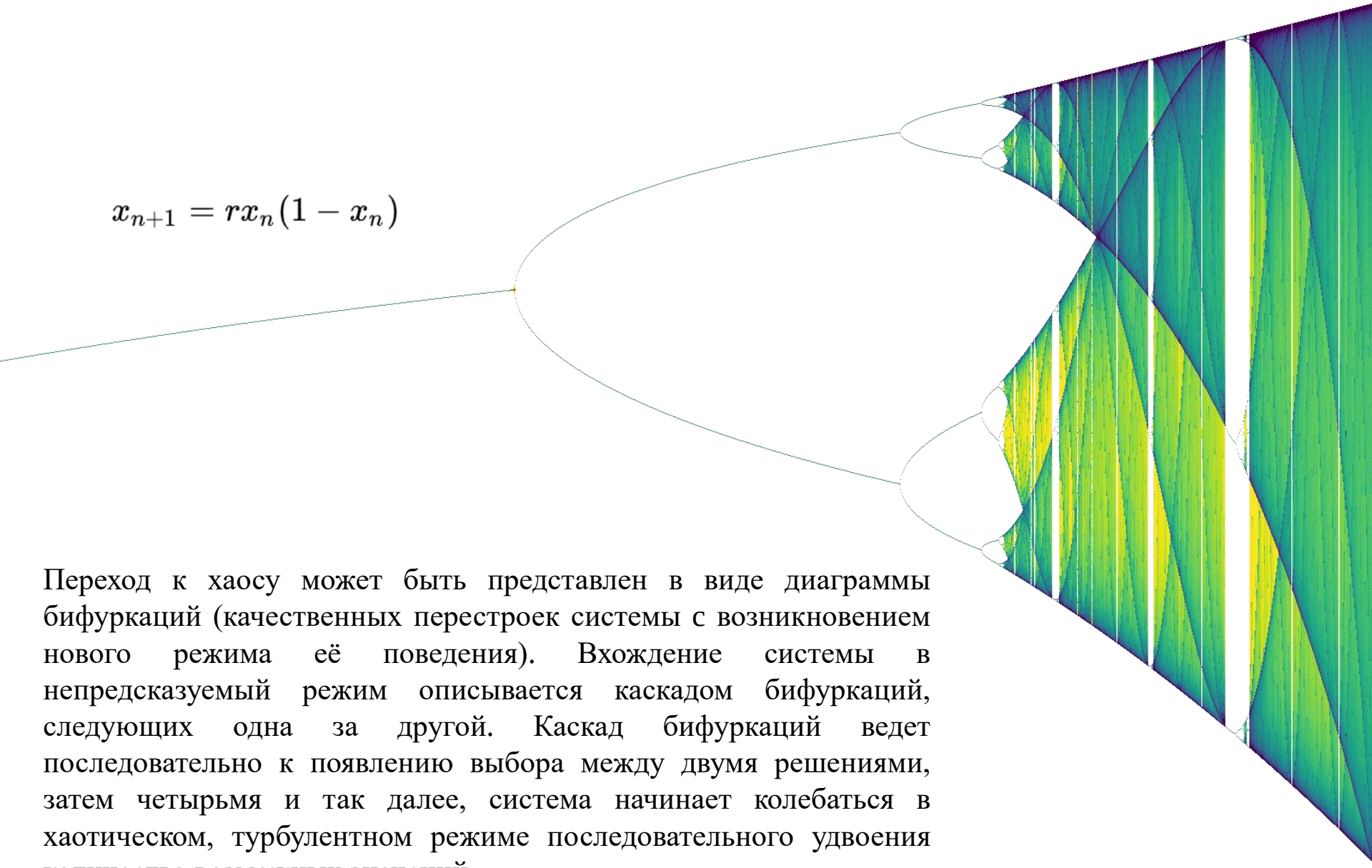

1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
1.4
1.6
1.8
2
2.2
2.4
2.6

Странный аттрактор — это притягивающее множество неустойчивых траекторий в фазовом пространстве диссипативной динамической системы.

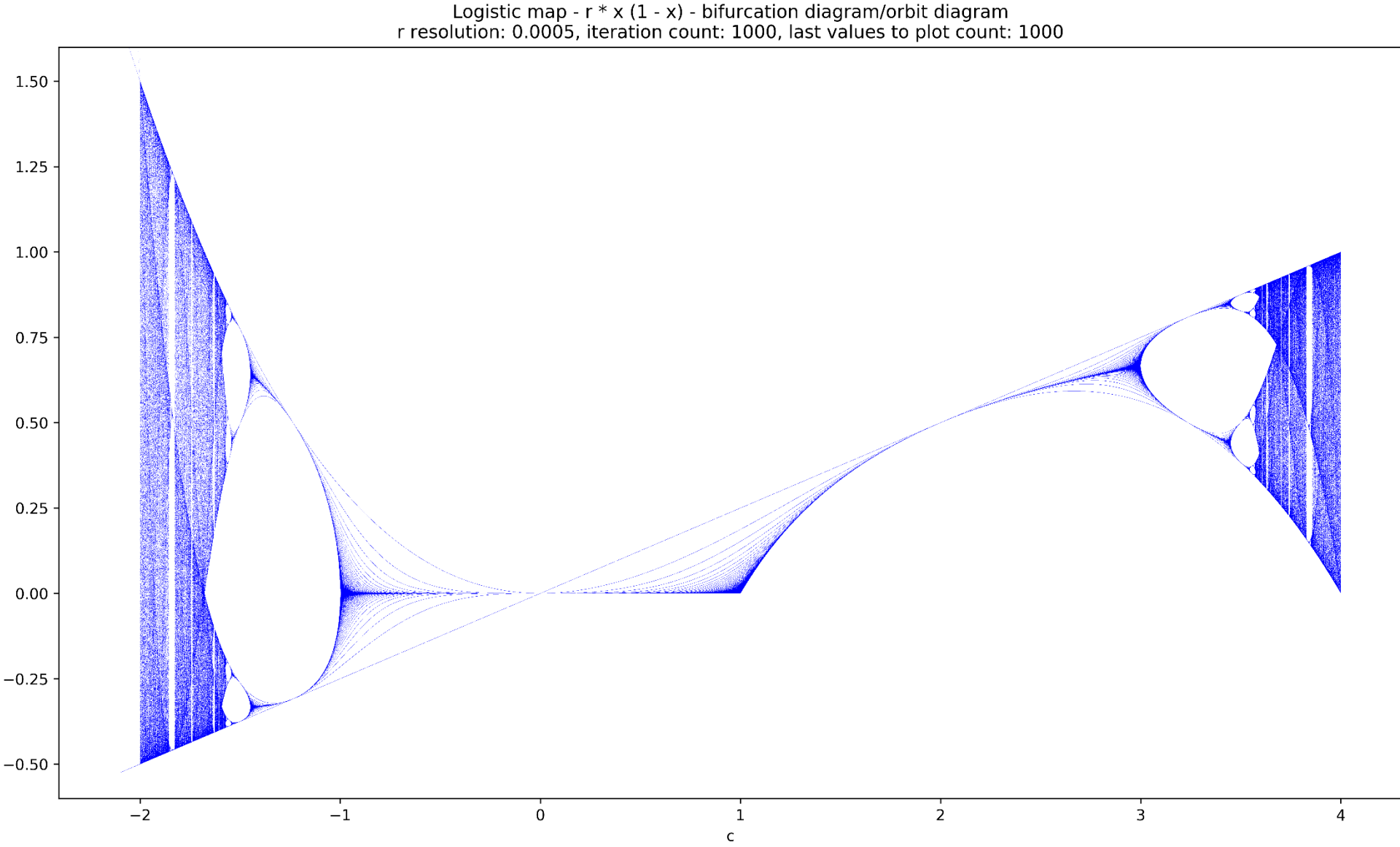
В отличие от аттрактора, не является многообразием, то есть не является кривой или поверхностью. Структура странного аттрактора фрактальна. Траектория такого аттрактора непериодическая (она не замыкается) и режим функционирования неустойчив (малые отклонения от режима нарастают).

Основным критерием хаотичности аттрактора является экспоненциальное нарастание во времени малых возмущений. Следствием этого является «перемешивание» в системе, непериодичность во времени любой из координат системы, сплошной спектр мощности и убывающая во времени автокорреляционная функция.

$$x_{n+1} = r x_n (1 - x_n)$$



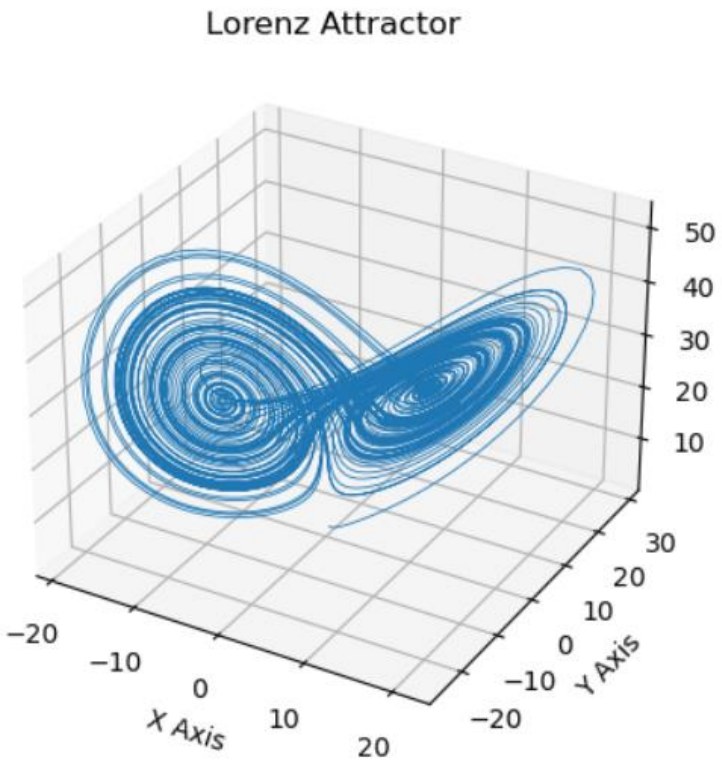
Переход к хаосу может быть представлен в виде диаграммы бифуркаций (качественных перестроек системы с возникновением нового режима её поведения). Вхождение системы в непредсказуемый режим описывается каскадом бифуркаций, следующих одна за другой. Каскад бифуркаций ведет последовательно к появлению выбора между двумя решениями, затем четырьмя и так далее, система начинает колебаться в хаотическом, турбулентном режиме последовательного удвоения количества возможных значений.


Logistic map - r * x (1 - x) - bifurcation diagram/orbit diagram
r resolution: 0.0005, iteration count: 1000, last values to plot count: 1000
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−2
−1
0
1
2
3
4
c

Аттрактор Лоренца

$$\begin{cases} \dot{x} = \sigma(y - x) \\ \dot{y} = x(r - z) - y \\ \dot{z} = xy - bz \end{cases}$$

```
plt.show()
```

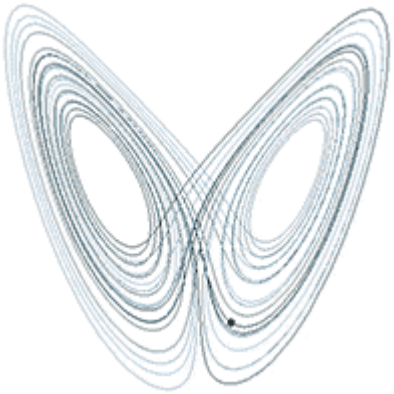


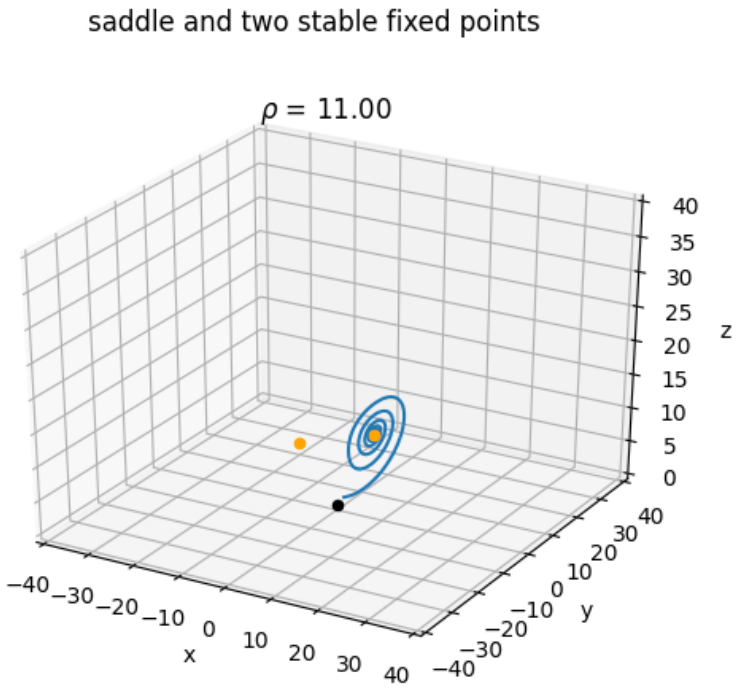
In [1]:

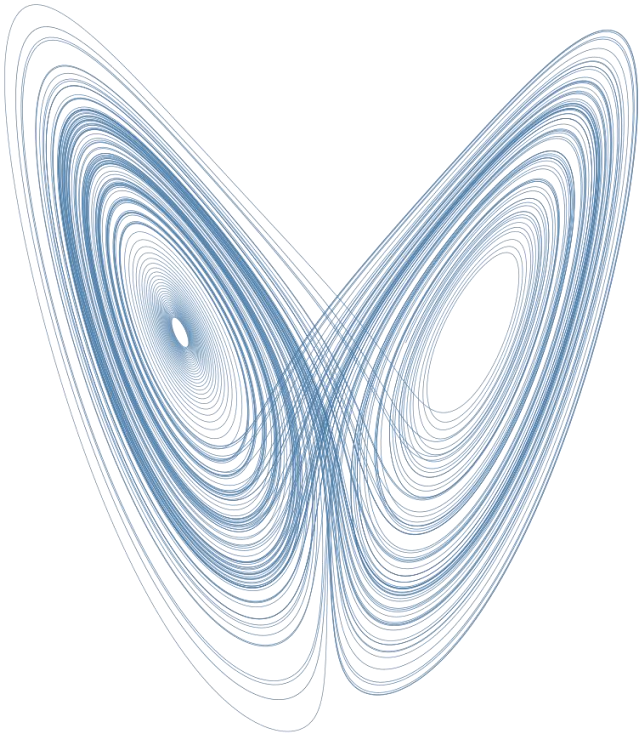
```python
import matplotlib.pyplot as plt
import numpy as np

def lorenz(xyz, *, s=10, r=28, b=2.667):
    x, y, z = xyz
    x_dot = s*(y - x)
    y_dot = r*x - y - x*z
    z_dot = x*y - b*z
    return np.array([x_dot, y_dot, z_dot])

dt = 0.01
num_steps = 10000

xyzs = np.empty((num_steps + 1, 3))
xyzs[0] = (0., 1., 1.05)
for i in range(num_steps):
    xyzs[i + 1] = xyzs[i] + lorenz(xyzs[i]) * dt

ax = plt.figure().add_subplot(projection='3d')

ax.plot(*xyzs.T, lw=0.6)
ax.set_xlabel("X Axis")
ax.set_ylabel("Y Axis")
ax.set_zlabel("Z Axis")
ax.set_title("Lorenz Attractor")

plt.show()
```

Аттрактор Лоренца

$$\begin{cases} \dot{x} = \sigma(y - x) \\ \dot{y} = x(r - z) - y \\ \dot{z} = xy - bz \end{cases}$$

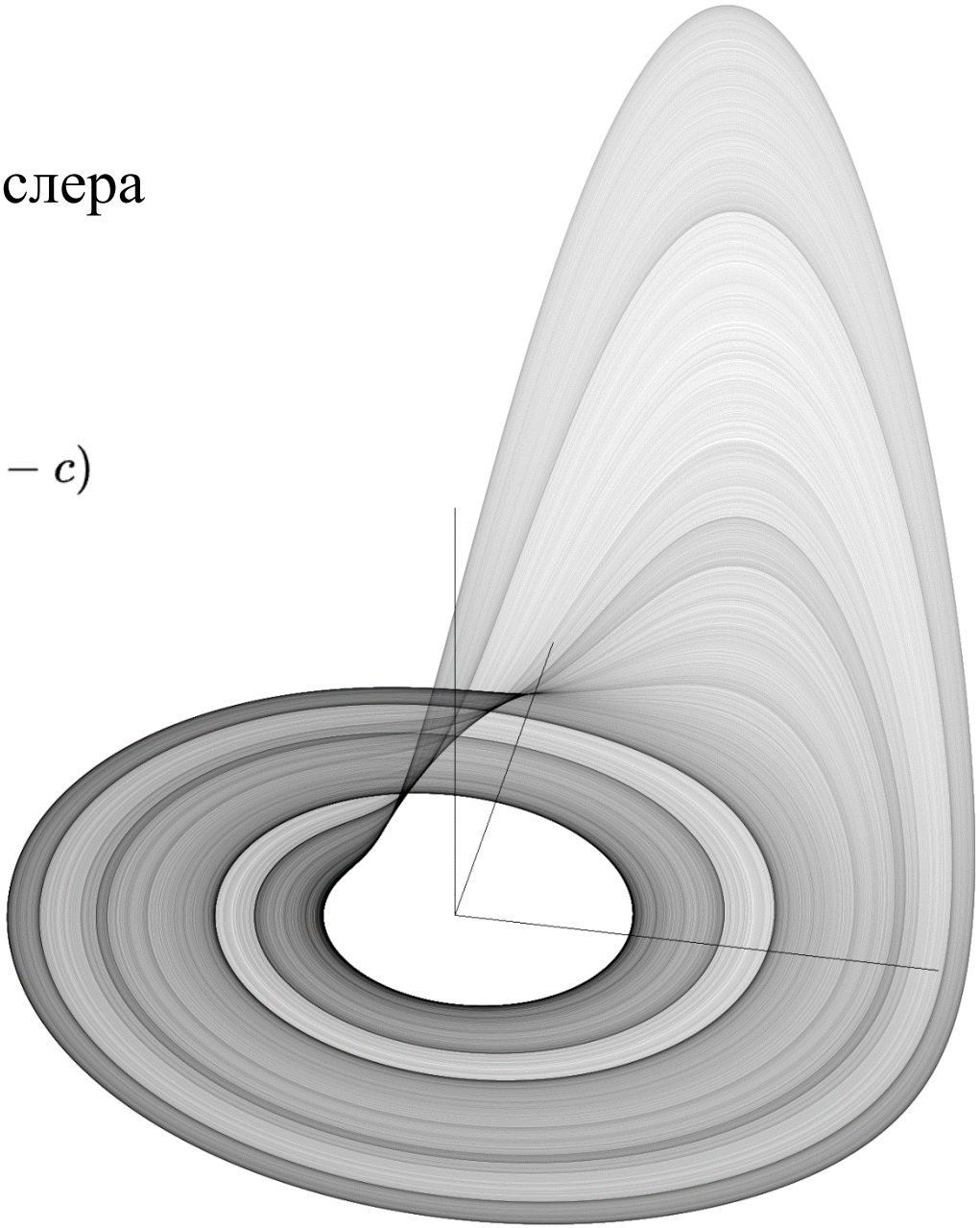






Аттрактор Рёсслера

$$\begin{cases} \frac{dx}{dt} = -y - z \\ \frac{dy}{dt} = x + ay \\ \frac{dz}{dt} = b + z(x - c) \end{cases}$$



РЯДЫ ФУРЬЕ

Сложный сигнал может быть представлен в виде некоторой комбинации компонентов – более простых колебаний (сигналов).

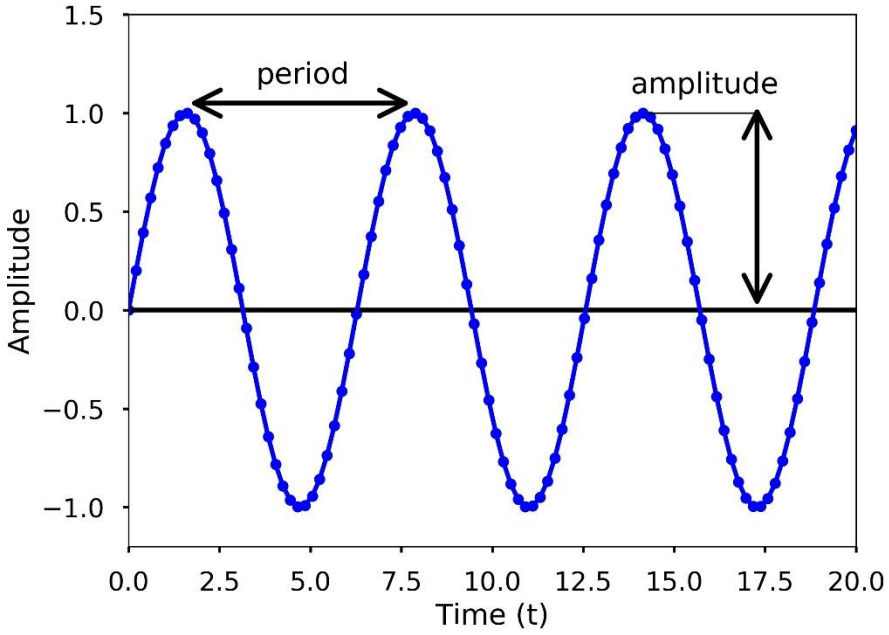
Если эти колебания имеют ясный физический смысл, то свойства сигнала могут быть объяснены в терминах самих колебаний.

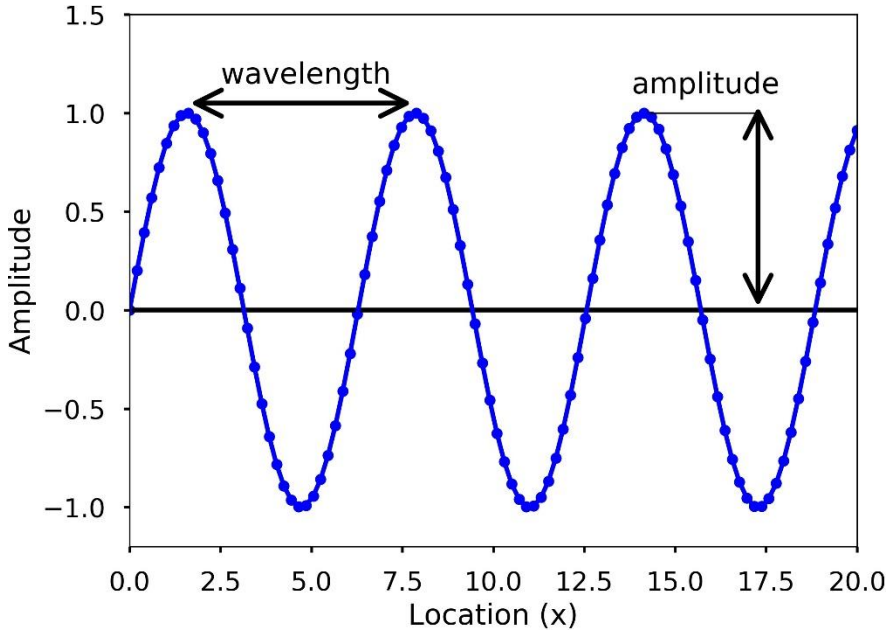
Анализом сигналов называется процесс определения и оценки величины компонентов, осуществляемый некоторыми техническими средствами по определенным формулам.

Произвольные периодические функции представляют собой суммы простейших гармонических функций – синусов и косинусов кратных частот.

Эти суммы получили название рядов Фурье.

Разложение периодического сигнала в ряд Фурье и проведение преобразования Фурье непериодических сигналов – являются основными методов исследования их свойств и характеристик





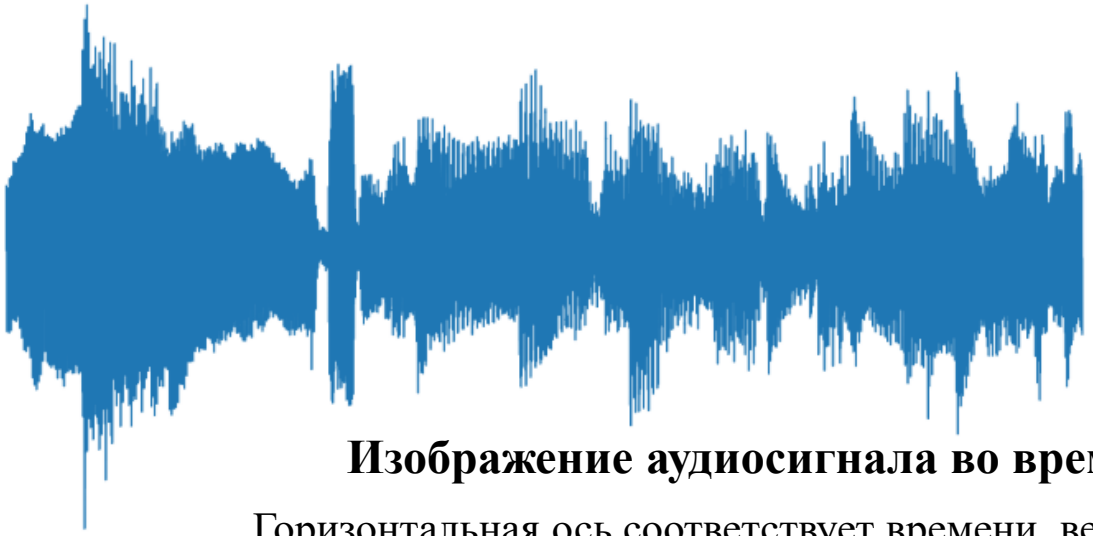
https://weitz.de/fourier/

https://prajwalsouza.github.io/Experiments/Fourier-Transform-Visualization.html

https://dave.whipp.name/tutorial/fourier.html

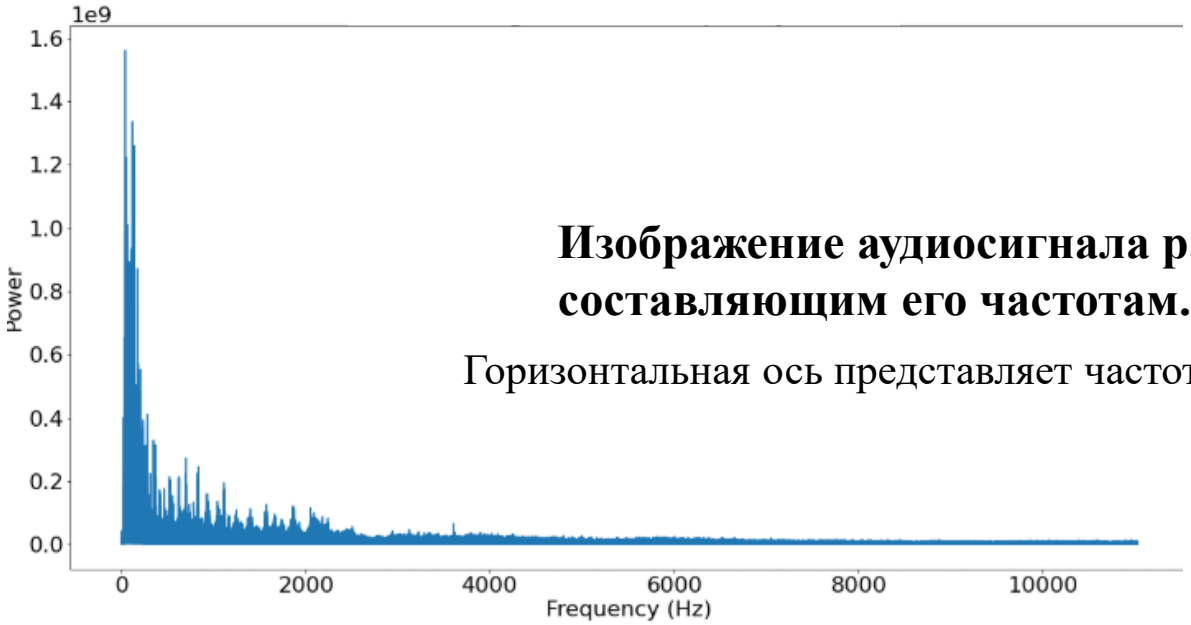
В зависимости от конкретной формы базисных функций различают несколько форм записи ряда Фурье:

- Синусно-косинусная форма
- Вещественная форма
- Комплексная форма



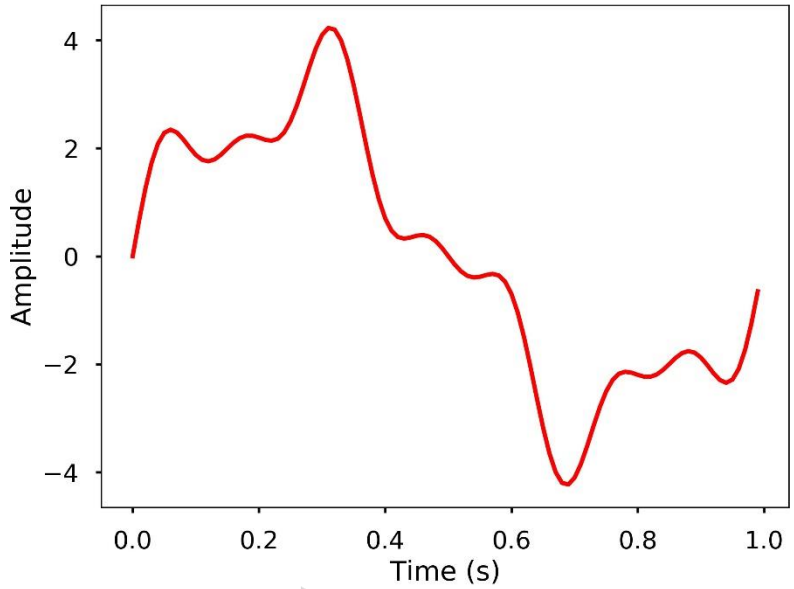
Изображение аудиосигнала во временной области.

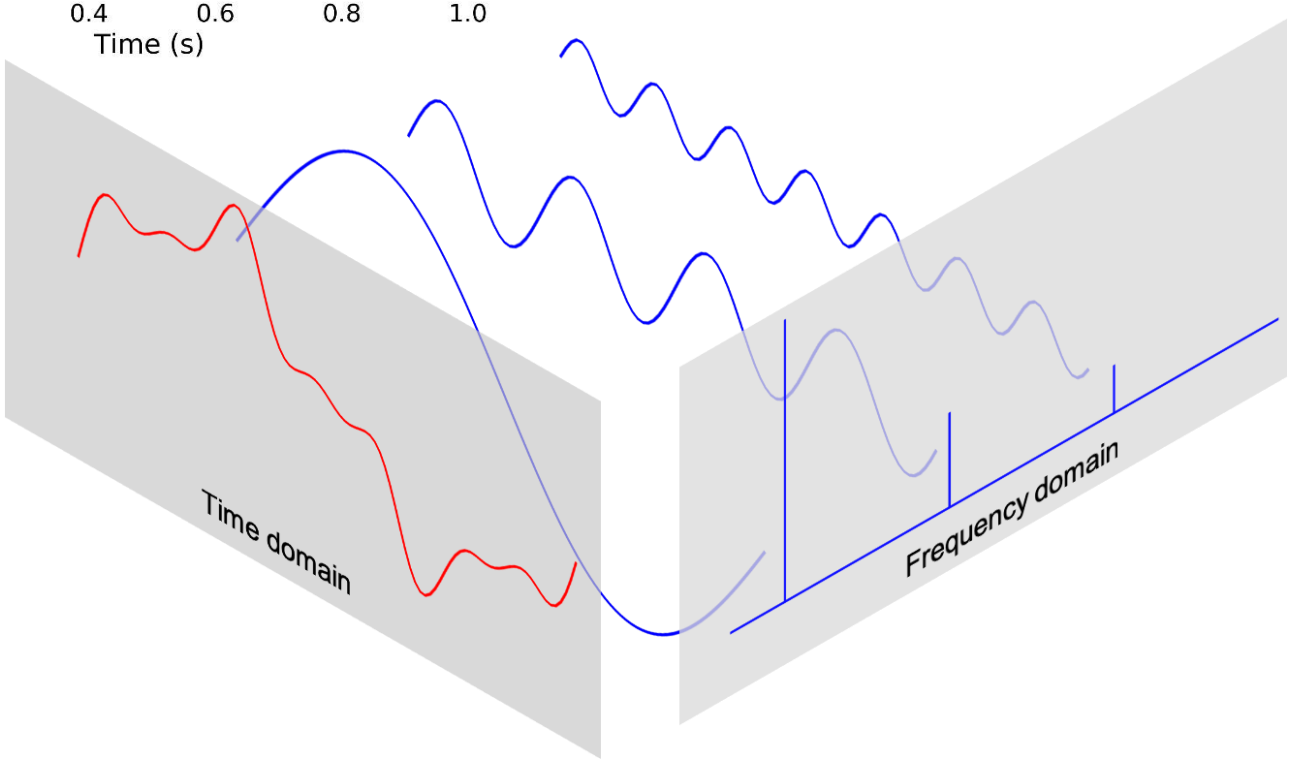
Горизонтальная ось соответствует времени, вертикальная ось – амплитуде.



Изображение аудиосигнала разложенным по составляющим его частотам.

Горизонтальная ось представляет частоту, вертикальная ось – мощность.


Amplitude
4
2
0
−2
−4
0.0
0.2
0.4
0.6
0.8
1.0
Time (s)


Time domain
Frequency domain

Имеет следующий вид:

$$s(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty} (a_k \cos(\mathrm{k}\omega_1 t) + b_k \sin(\mathrm{k}\omega_1 t))$$

где $\omega_1 = {}^{2\pi}/_{T}$ – круговая частота, кратные ей частоты $\mathrm{k}\omega_1$ – гармоники сигнала, коэффициенты a_k и b_k рассчитываются по формулам:

- $a_k = \frac{2}{T} \int_{-T/2}^{T/2} s(t) \cos(k\omega_1 t)\, dt$ и
- $b_k = \frac{2}{T} \int_{-T/2}^{T/2} s(t) \sin(k\omega_1 t)\, dt$
- $\frac{a_0}{2} = \frac{1}{T} \int_{-T/2}^{T/2} s(t) dt$ – среднее значение сигнала за период

Ряд по тригонометрической системе функций называется тригонометрическим рядом:

$$\frac{a_0}{2} + a_1 \cos\frac{\pi x}{l} + b_1 \sin\frac{\pi x}{l} + a_2 \cos\frac{2\pi x}{l} + b_2 \sin\frac{2\pi x}{l} + \cdots + a_n \cos\frac{n\pi x}{l} + b_n \sin\frac{n\pi x}{l} + \cdots =$$

$$= \frac{a_0}{2} + \sum_{n=1}^{\infty} \left(a_n \cos\frac{n\pi x}{l} + b_n \sin\frac{n\pi x}{l} \right)$$

Если этот ряд сходится к некоторой функции $f(x)$ в каждой точке непрерывности этой функции, то говорят, что $f(x)$ разлагается в ряд по тригонометрической системе функций.

$$f(x) = \frac{a_0}{2} + a_1 \cos\frac{\pi x}{l} + b_1 \sin\frac{\pi x}{l} + a_2 \cos\frac{2\pi x}{l} + b_2 \sin\frac{2\pi x}{l} + \cdots + a_n \cos\frac{n\pi x}{l} + b_n \sin\frac{n\pi x}{l} + \cdots$$

Разложение функции в тригонометрический ряд

Тригонометрическим рядом называется функциональный ряд вида

$$\frac{a_0}{2} + (a_1 \cos \omega_1 x + b_1 \sin \omega_1 x) + (a_2 \cos \omega_2 x + b_2 \sin \omega_2 x) + \ldots =$$

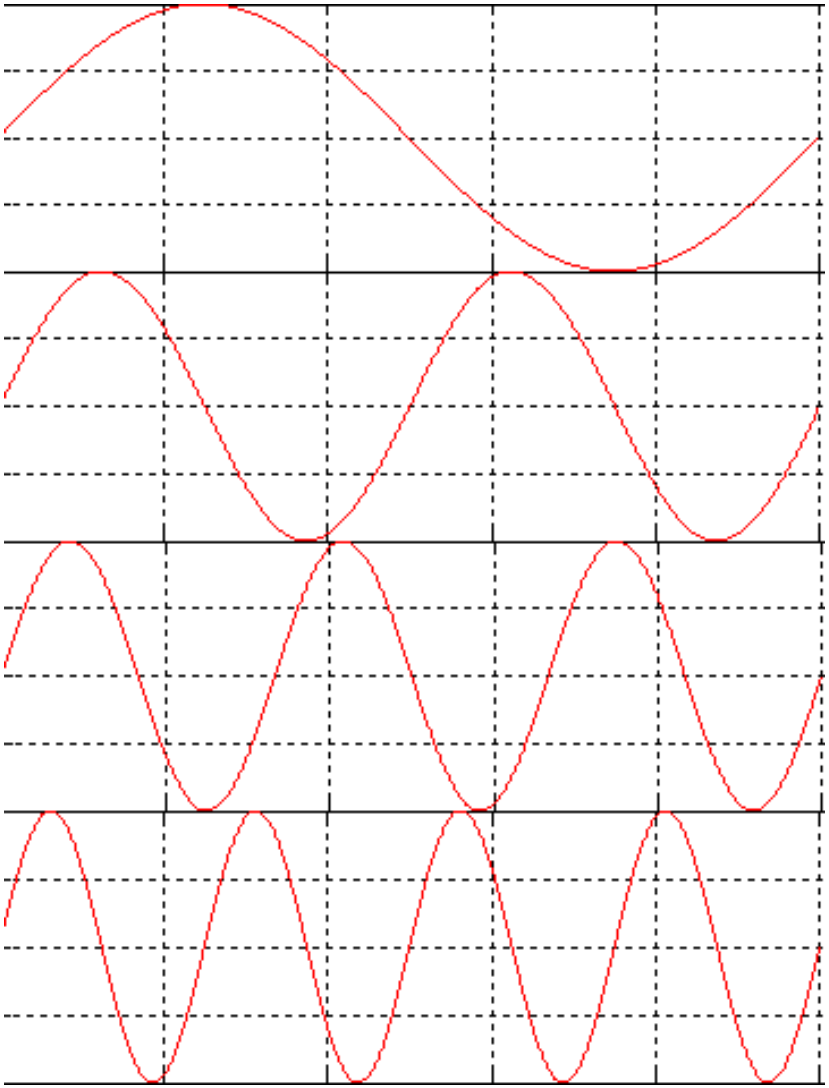
$$= \frac{a_0}{2} + \sum_{n=1}^{\infty} (a_n \cos \omega_n x + b_n \sin \omega_n x)$$

где a_0 , a_n , b_n $(n = 1,2,\ldots)$ – *числа* (***коэффициенты ряда***),

$$\omega_n = \frac{n\pi}{\ell} \quad (n \in \mathbb{N},\ \ell > 0)$$

Свободный член ряда записан в виде дроби для единообразия последующих формул.

Функции разложения в ряд Фурье



Справедливы утверждения:

1. Если $S(x)$ – сумма тригонометрического ряда, то $S(x)$ – периодическая, с периодом $T = 2\ell$.
2. Если ряд сходится на отрезке, длиной 2ℓ , то он сходится на всей числовой оси.

   В общем случае область сходимости ряда – объединение интервалов вида $(a + 2\ell k\ ;\ b + 2\ell k)$, где a, b – некоторые числа, $k \in \mathbb{Z}$

Пусть $f(x)$ – периодическая функция $(T = 2\ell)$.

1. Разложима ли $f(x)$ в тригонометрический ряд?
2. Если $f(x)$ разложима в тригонометрический ряд, то как найти его коэффициенты?

Замечания.

1) Периодическую функцию достаточно рассматривать на любом отрезке длиной T .

   Удобнее всего брать отрезок $[-\ell\,;\,\ell]$ (где $T = 2\ell$).

2) Имеют место следующие равенства:

$$\int_{-\ell}^{\ell} \sin \omega_n x\,dx = 0, \qquad \int_{-\ell}^{\ell} \cos \omega_n x\,dx = 0. \qquad \int_{-\ell}^{\ell} \sin \omega_n x \cdot \cos \omega_k x\,dx = 0.$$

$$\int_{-\ell}^{\ell} \sin \omega_n x \cdot \sin \omega_k x\,dx = \begin{cases} 0, & \text{если } n \neq k; \\ \ell, & \text{если } n = k. \end{cases}$$

$$\int_{-\ell}^{\ell} \cos \omega_n x \cdot \cos \omega_k x\,dx = \begin{cases} 0, & \text{если } n \neq k; \\ \ell, & \text{если } n = k. \end{cases}$$

Пусть f(x) – периодическая функция (T = 2ℓ),

f(x) – сумма тригонометрического ряда на [–ℓ ; ℓ], т.е.

$$f(x)=\frac{a_0}{2}+\sum_{n=1}^{\infty}(a_n\cos\omega_n x+b_n\sin\omega_n x).$$

Пусть ряд сходится к f(x) на [–ℓ ; ℓ] равномерно.

Тогда: 1. f(x) – непрерывна на [–ℓ ; ℓ];

2. f(x) – интегрируема на [–ℓ ; ℓ], причем

$$a_0=\frac{1}{\ell}\int_{-\ell}^{\ell}f(x)dx,$$

$$a_n=\frac{1}{\ell}\cdot\int_{-\ell}^{\ell}f(x)\cdot\cos\omega_n x\,dx\quad(n=1,2,3,\ldots),$$

$$b_n=\frac{1}{\ell}\cdot\int_{-\ell}^{\ell}f(x)\cdot\sin\omega_n x\,dx\quad(n=1,2,3,\ldots).$$

Ряды Фурье позволяют изучать периодические (и непериодические) функции при помощи представления их функциональными рядами.

Переменные токи и напряжения, смещения, скорость и ускорение кривошипно-шатунных механизмов и акустические волны - это типичные практические примеры применения периодических функций в инженерных расчетах.

Разложение в ряд Фурье основывается на предположении, что все имеющие практическое значение функции в интервале, равном периоду этих функций, можно выразить в виде сходящихся тригонометрических рядов.

ТЕОРЕМА о разложении функции в тригонометрический ряд.

Если функция разлагается в тригонометрический ряд, то этот ряд является ее тригонометрическим рядом Фурье.

Чтобы в формуле ряда Фурье остались только слагаемые одного вида, например косинусные, применяют тригонометрические преобразования и получают *вещественную* форму представления разложения:

$$s(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty} A_k \cos(k\omega_1 t + \varphi_k),$$

При этом изменяется амплитуда и возникает начальная фаза, а частота остается прежней.

Если s(t) четная функция, то фазы могут принимать только значения 0 и π, если нечетная – фазы могут быть равны ±π/2.

Является наиболее употребимой в технике обработки сигналов.

Получается из вещественной формы применением формулы Эйлера

$$e^{jx} = cosx + jsinx$$

в виде

$$cosx = \frac{1}{2}\left(e^{jx} + e^{-jx}\right).$$

Получим:

$$s(t) = \sum_{-\infty}^{\infty} C_k \, e^{jk\omega_1 t}$$

Формула расчета коэффициентов ряда Фурье в комплексной форме:

$$C_k = \frac{1}{T}\int_{-T/2}^{T/2} s(t)\exp(-jk\omega_1 t)dt$$

Если s(t) является *четной* функцией, коэффициенты ряда будут *вещественными*, если *нечетной – мнимыми*.

Совокупность амплитуд гармоник ряда Фурье называют **амплитудным спектром**, совокупность их фаз – **фазовым спектром**.

Рассмотрим формулу: $a\cos t + b\sin t = r\sin(t+\varphi) = r\cos(t-\psi)$, где $r = \sqrt{a^2+b^2}$, $\mathrm{tg}\varphi = \frac{a}{b}$, $\mathrm{ctg}\psi = \frac{b}{a}$, подставляя в тригонометрический ряд Фурье, получаем:

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} A_n \sin\left(\frac{\pi n x}{l} + \varphi_n\right),$$

Где $A_0 = \left|\frac{a_0}{2}\right|$, $A_n = \sqrt{a_n^2 + b_n^2}$ – ***амплитудный спектр***

$\varphi_n = arctg\frac{a_n}{b_n}$ – ***фазовый спектр***

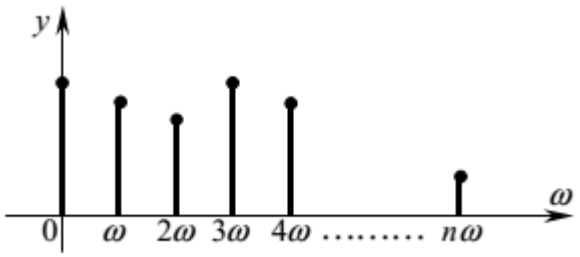
Следует отличать спектры от характеристик. Последние относятся не к сигналам (как спектры), а к системам обработки сигналов.

Первое слагаемое $A_1 \sin\left(\frac{\pi x}{l} + \varphi_1\right)$ называют ***основной гармоникой***,

остальные слагаемые $A_n \sin\left(\frac{\pi n x}{l} + \varphi_n\right)$ – ***верхними гармониками*** $(n = 1,2,3 \ldots)$;

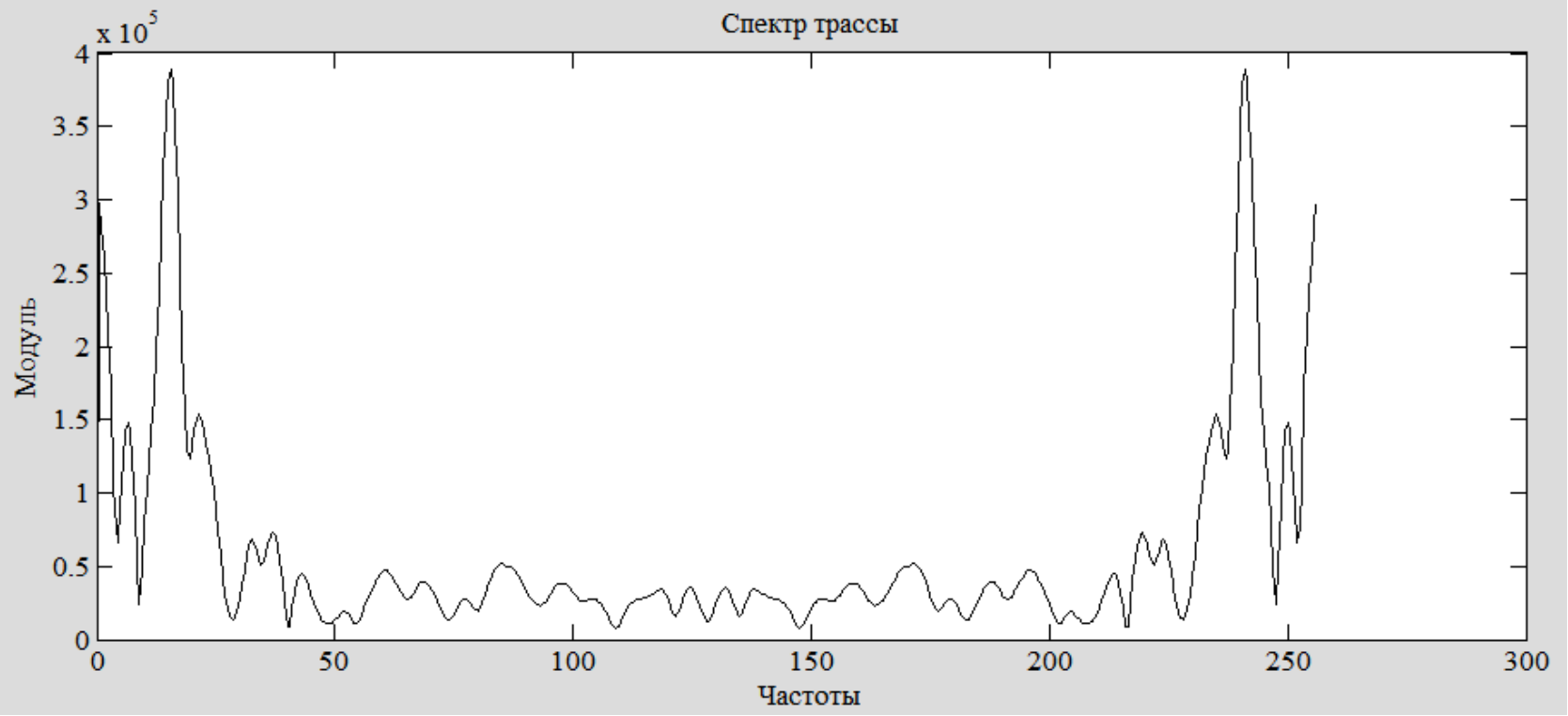
A_n – амплитуда n -гармоники, φ_n – фаза n –гармоники, $\omega = \frac{\pi}{l}$ называется ***основной частотой***;

$\omega_n = \frac{n\pi}{l}$ называются ***спектральными или волновыми*** числами функции $f(x)$;

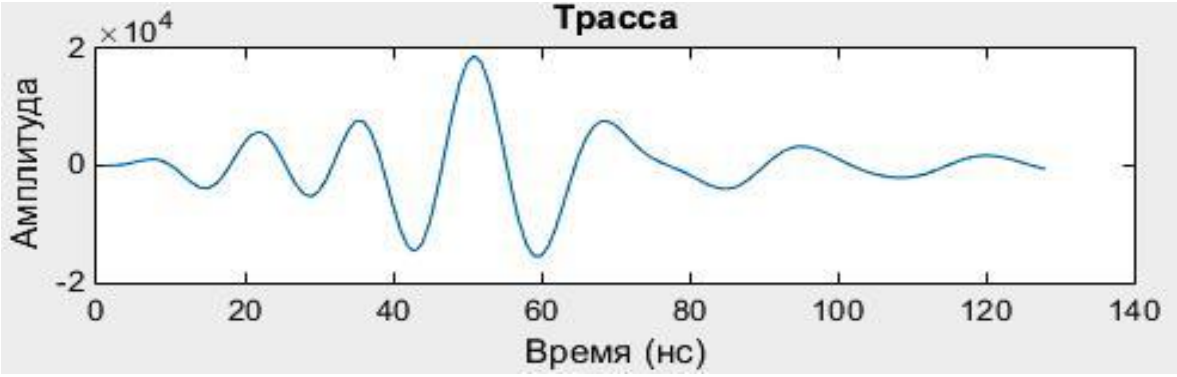


Большинство фильтров работают не с самим сигналом, а с его спектром. Если вычислить спектр сигнала, удалить из него (или существенно уменьшить) определенные частоты, а затем выполнить обратное преобразование Фурье, то результатом будет фильтрованный сигнал. Эта процедура производится в тех случаях, когда частотный спектр помехи или шума занимают на оси частот интервал, отличный или лишь частично перекрывающийся с частотным диапазоном сигнала.

График модуля спектра трассы радарограммы полученное БПФ.



Полученный комплексный спектр трассы умножается на комплексный спектр фильтра и результат подвергается обратному преобразованию Фурье. На рисунке представлен график фильтрованный трассы радарограммы.



Любой периодический сигнал бесконечен во времени, что на практике неосуществимо, поэтому периодический сигнал - математическая абстракция, и все рассмотренное выше не применимо к реальным сигналам.

Реальный сигнал ограничен во времени и, следовательно, является непериодическим. Однако, условно его можно рассматривать как периодический с периодом $T \to \infty$. Тогда $\omega_0 = 2\pi/T \to 0$, а спектры амплитуд и фаз становятся непрерывными (сплошными), сумма в разложении Фурье превращается в интеграл.

В результате переходим к интегралу Фурье

Ряд по тригонометрической системе функций называется рядом Фурье, если его коэффициенты находят по формулам:

$$a_0 = \frac{1}{l}\int_{-l}^{l} f(x)dx$$

$$a_n = \frac{1}{l}\int_{-l}^{l} f(x)\cdot\cos\frac{n\pi x}{l}dx$$

$$b_n = \frac{1}{l}\int_{-l}^{l} f(x)\cdot\sin\frac{n\pi x}{l}dx$$

$$f(x) = \frac{a_0}{2} + a_1\cos\frac{\pi x}{l} + b_1\sin\frac{\pi x}{l} + a_2\cos\frac{2\pi x}{l} + b_2\sin\frac{2\pi x}{l} + \cdots + a_n\cos\frac{n\pi x}{l} + b_n\sin\frac{n\pi x}{l} + \cdots$$

Разложению в ряд Фурье могут подвергаться периодические сигналы. При этом они представляются в виде суммы гармонических функций либо комплексных экспонент ($e^{2\pi\nu t}$) с частотами, образующими арифметическую прогрессию.

Чтобы такое разложение существовало, фрагмент сигнала длительностью в один период должен удовлетворять условиям Дирихле

Является инструментом спектрального анализа *непериодических* сигналов.

Прямое преобразование Фурье (Фурье-образ сигнала):

$$\boldsymbol{S}(\omega) = \int_{-\infty}^{\infty} s(t) e^{-j\omega t} dt$$

Обратное преобразование Фурье:

$$s(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \boldsymbol{S}(\omega) e^{j\omega t} d\omega$$

Если использовать не круговую частоту, а обычную $f = {}^{\omega}/_{2\pi}$, то формулы прямого и обратного преобразований Фурье будут отличаться только знаком в показателе экспоненты:

$$\boldsymbol{S}(\omega) = \int_{-\infty}^{\infty} s(t) e^{-j2\pi f t} dt$$

$$s(t) = \int_{-\infty}^{\infty} \boldsymbol{S}(\omega) e^{j2\pi f t} d\omega$$

Прямое преобразование Фурье – это разложение сигнала на гармонические функции (в спектр).

Обратное преобразование Фурье – это синтез сигнала по гармоникам.

Во фрагменте сигнала длительностью
в один период

- Не должно быть разрывов второго рода (с уходящими в бесконечность ветвями функции);
- Число разрывов первого рода (скачков) должно быть конечным;
- Число экстремумов должно быть конечным;
- В любой точке периода первая производная должна быть конечной (или конечной является левая или правая производная – условие Дини).

Достаточные условия сходимости ряда к функции $f(x)$ на отрезке $[-l; l]$)

Пусть на отрезке $[-l; l]$ функция $f(x)$ удовлетворяет условиям:

1. $f(x)$ непрерывна или имеет конечное число точек разрыва первого рода (иными словами, кусочно-непрерывна);
2. $f(x)$ монотонна или имеет конечное число интервалов монотонности (иными словами, кусочно-монотонна).

Тригонометрический ряд Фурье функции f(x) сходится во всех точках $[-l; l]$,

и его суммой является функция S(x), определенная на $[-l; l]$ следующим образом:

a) S(x)=f(x), если $x \in (-l; l)$, x – точка непрерывности функции $f(x)$;

b) $S(x) = \dfrac{f(x-0)+f(x+0)}{2}$, если $x \in (-l; l)$, x – точка разрыва функции $f(x)$;

c) $S(-l) = S(l) = \dfrac{f(-l+0)+f(l-0)}{2}$ на границе промежутка $[-l; l]$.

Причем на любом отрезке, не содержащем точек разрыва функции, сходимость тригонометрического ряда Фурье равномерная.

Условия 1 и 2 называются условиями Дирихле

ТЕОРЕМА – 2-е достаточное условие разложения функции в тригонометрический ряд.

Пусть f(x) – периодическая функция (T = 2ℓ).

Если на [–ℓ ; ℓ] функция f(x) и ее производная f’(x) непрерывны или имеют конечное число точек разрыва первого рода, то тригонометрический ряд Фурье функции f(x) сходится на всей числовой прямой и его суммой будет функция S(x), определенная следующим образом:

a) S(x) = f(x), если x – точка непрерывности f(x);

b) $S(x) = \dfrac{f(x-0) + f(x+0)}{2}$, если x – точка разрыва f(x).

При этом тригонометрический ряд Фурье функции f(x) сходится равномерно на любом отрезке, целиком лежащем в интервале непрерывности функции f(x).

Функция, удовлетворяющая условию данной теоремы, называется кусочно-гладкой.

Чтобы преобразование Фурье было применимо, сигнал должен удовлетворять не только тем же, что и в случае разложения в ряд Фурье, условиям, но и еще одному – сигнал должен быть абсолютно интегрируемым, т.е. интеграл его модуля должен быть конечной величиной:

$$\int_{-\infty}^{\infty} |s(t)|\, dt < \infty$$

Модуль спектральной функции **S**(ω) называют амплитудным спектром, а ее аргумент – фазовым спектром.

Преобразование Фурье ставит в соответствие сигналу, **заданному во времени**, его **спектральную функцию**, т.е. осуществляется **переход из временной области в частотную**.

Преобразование Фурье является взаимно-однозначным, поэтому представление сигнала в частотной области (т.е. спектральная функция) содержит **столько же информации**, сколько и исходный сигнал, заданный во временной области.

Все функции тригонометрической системы периодические, их общий период T = 2l.

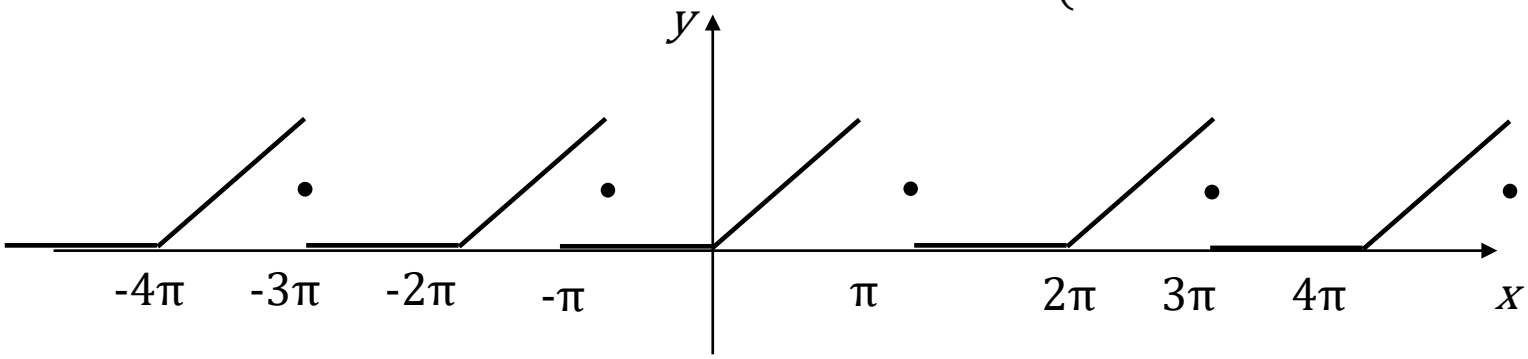
Следовательно, если ряд Фурье сходится на промежутке [-*l*; *l*], то он сходится и на всей числовой прямой.

Его сумма периодически повторяет значения функции, которые она принимает на [-*l*; *l*].

Поэтому говорят о разложении в ряд Фурье функции f(x) не только на промежутке [-*l*; *l*], но и функции, которая является периодическим продолжением f(x) с периодом 2*l* на всю числовую ось.

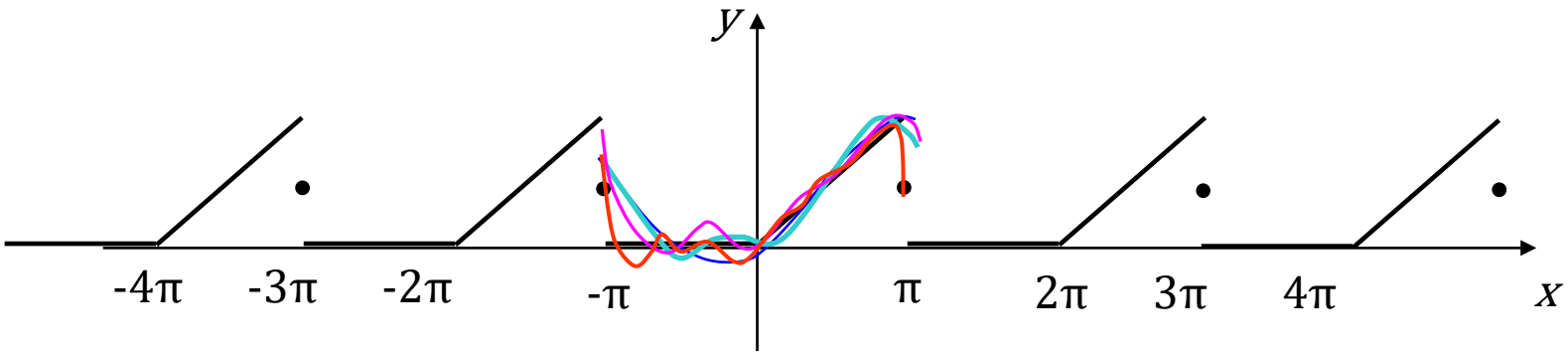
Пример

$$f(x)=\begin{cases}0, & \text{если} \quad -\pi \le x \le 0,\\ x, & \text{если} \quad 0 < x \le \pi,\end{cases}$$



ПРИМЕР

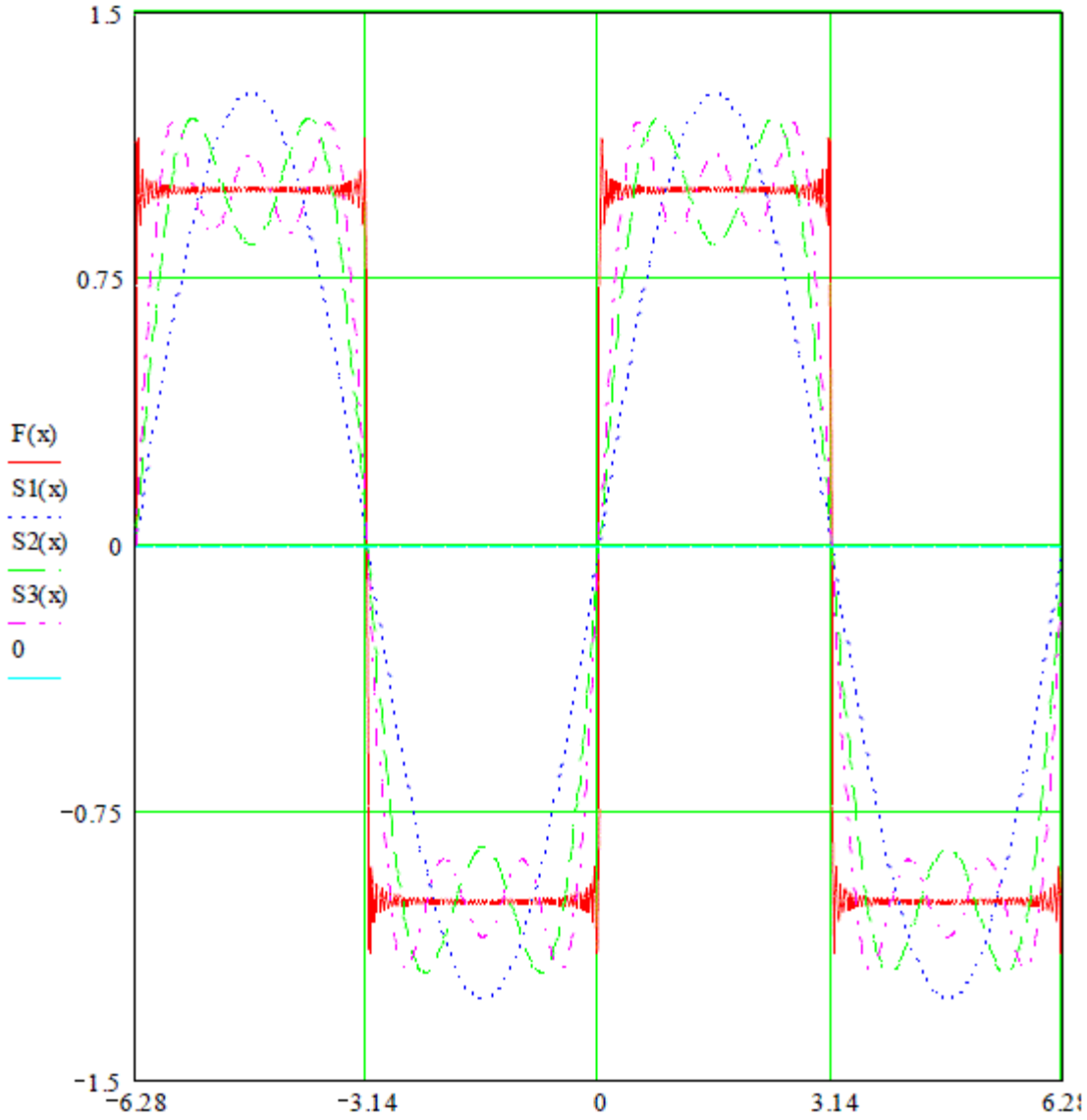
$$f(x)=\begin{cases}0, & \text{если} \quad -\pi \le x \le 0,\\ x, & \text{если} \quad 0 < x \le \pi,\end{cases}$$

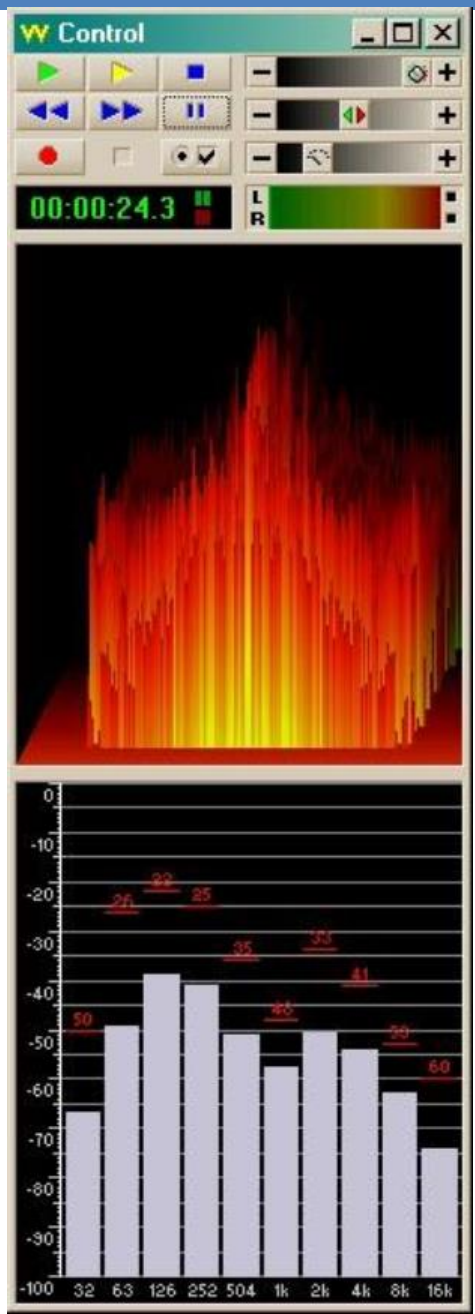


$$f(x)=\frac{\pi}{2}+\sum_{n=1}^{\infty}\left(\frac{(-1)^n-1}{\pi n^2}\cos n\,x+\frac{(-1)^{n+1}}{n}\sin n\,x\right)$$

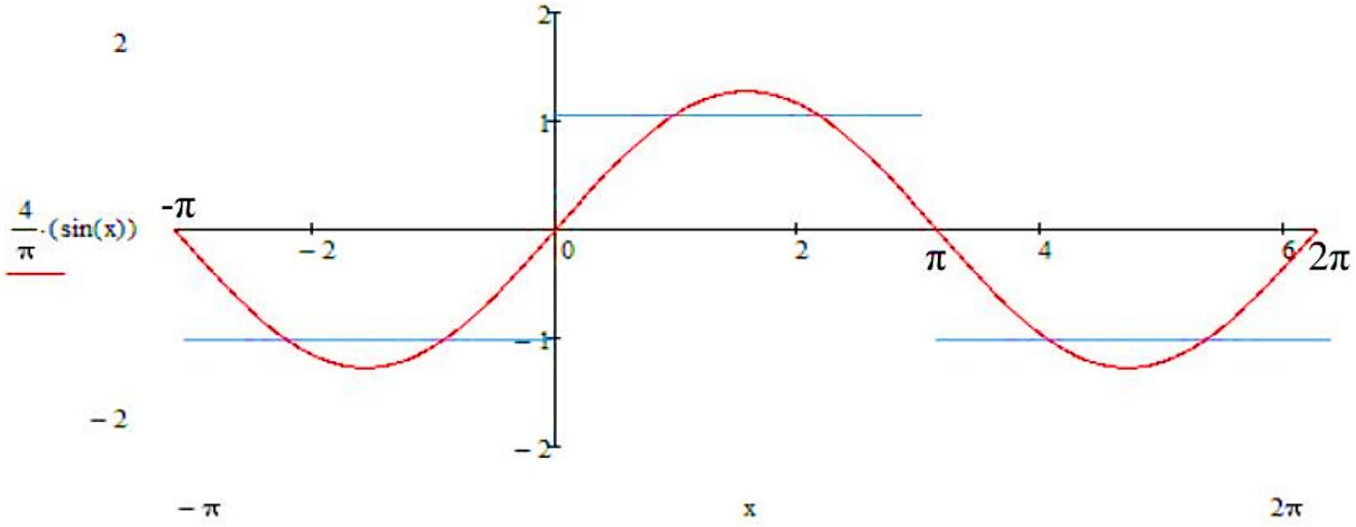
$$=\frac{\pi}{2}-\frac{2}{\pi}\cos x+\sin x-\frac{1}{2}\sin 2\,x-\frac{2}{\pi\cdot 9}\cos 3\,x+\frac{1}{3}\sin 3\,x+\frac{1}{4}\sin 4\,x-\frac{2}{\pi\cdot 25}\cos 5\,x+\frac{1}{5}\sin 5\,x$$

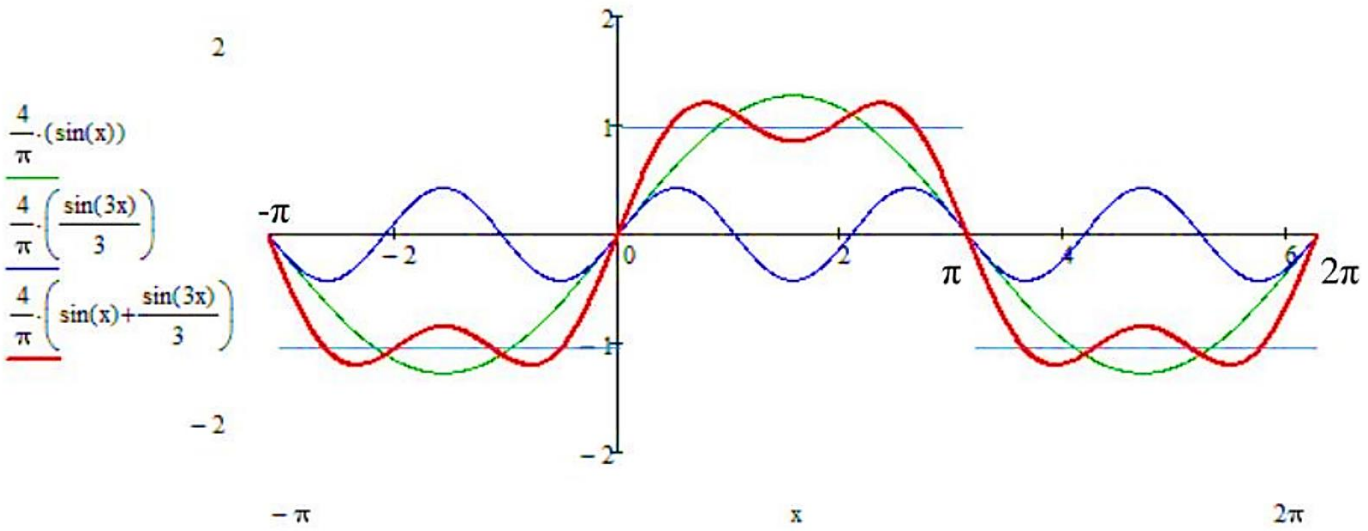
ПРИМЕР


1.5
0.75
0
−0.75
−1.5
−6.28
−3.14
0
3.14
6.2
F(x)
S1(x)
S2(x)
S3(x)
0

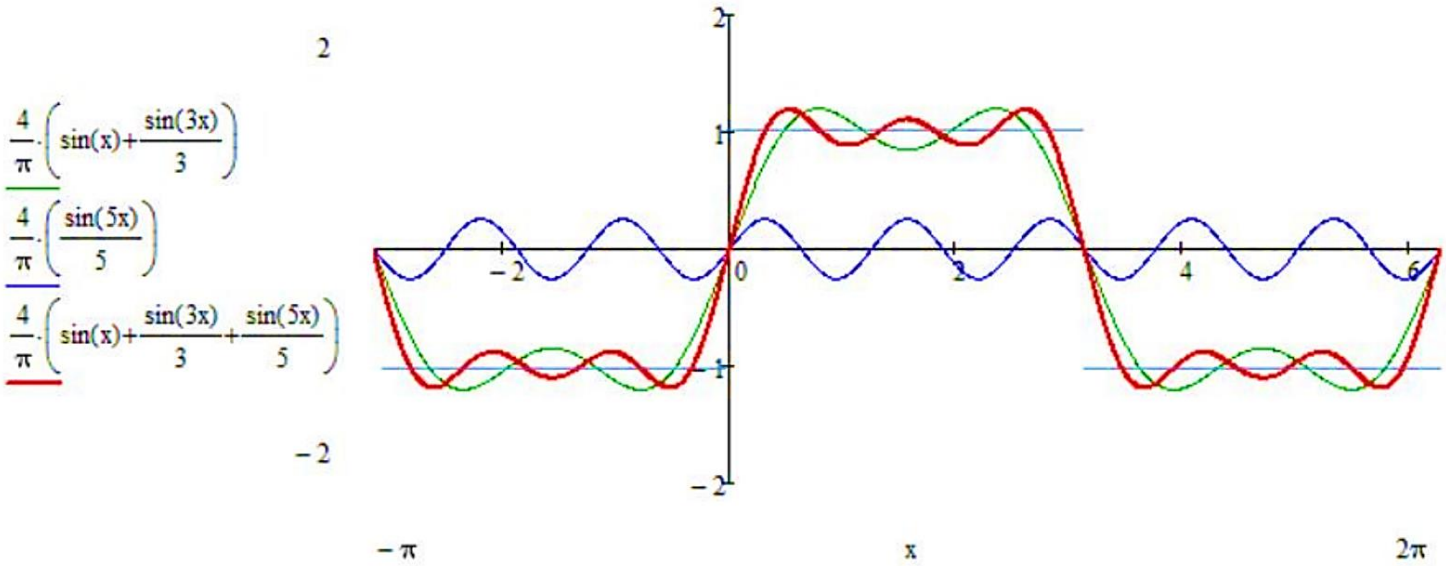

Control
00:00:24.3
L
R
0
-10
-20
-30
-40
-50
-60
-70
-80
-90
-100
32
63
126
252
504
1k
2k
4k
8k
16k

ПРИМЕР #2


$\frac{4}{\pi}\cdot(\sin(x))$
2
-2
-π
-2
0
2
π
4
6
2π
1
-1
-2
-π
x
2π


$\frac{4}{\pi}\cdot(\sin(x))$
$\frac{4}{\pi}\cdot\left(\frac{\sin(3x)}{3}\right)$
$\frac{4}{\pi}\cdot\left(\sin(x)+\frac{\sin(3x)}{3}\right)$
2
-2
-π
-2
0
2
π
4
6
2π
1
-1
-2
-π
x
2π

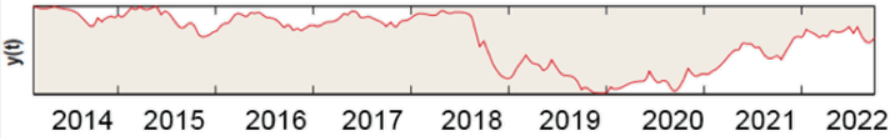
ПРИМЕР #3



Чем больше сложим простых гармоник, тем точнее результирующая функция будет представлять функцию f(x).

ПРИМЕР #4

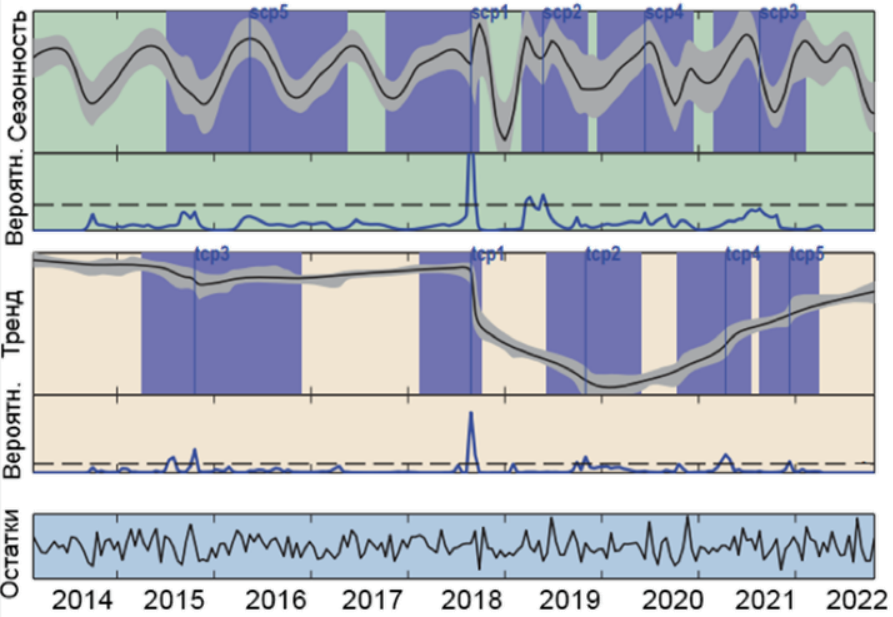
Пример временного ряда NDVI



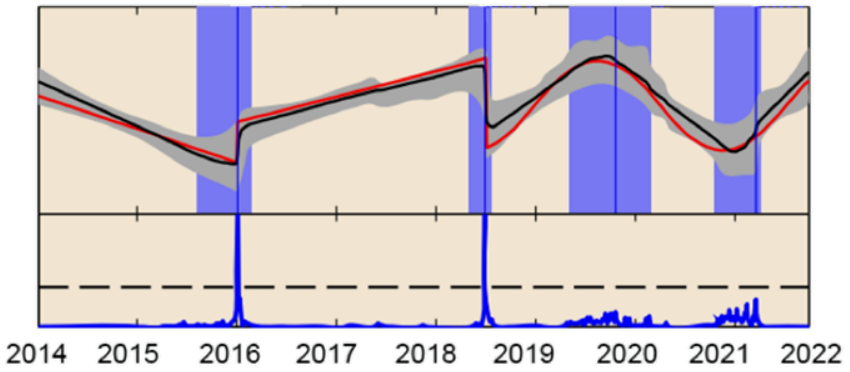
$$S(t) = \sum_{j=1}^{k} \left[\alpha_{k,l} \cdot \sin\left(\frac{2\pi j t}{f} + \delta_j\right) + b_{k,l} \cdot \cos\left(\frac{2\pi j t}{f} + \delta_j\right) \right] \gamma_j$$

$$\xi_k \le t < \xi_{k+1}, \ k = 0, \ldots, p.$$

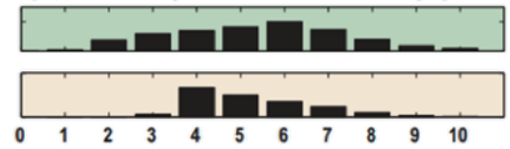
Разложение ряда на компоненты:



Аппроксимация, разложение на сегменты:

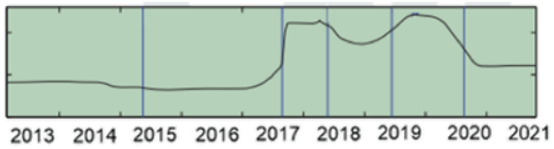


Распределение вероятности изменения прироста NDVI

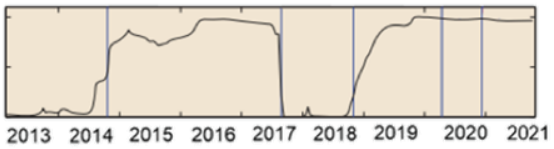


Количество изменений "сигналов" в сезоности или тренде

Динамика изменения среднесезонного NDVI



Ожидаемая вероятность прироста биомассы



Тригонометрические ряды Фурье для четных и нечетных функций:

$$a_0 = \frac{1}{l}\int_{-l}^{l} f(x)dx \qquad a_n = \frac{1}{l}\int_{-l}^{l} f(x)\cdot\cos\frac{n\pi x}{l}dx \qquad b_n = \frac{1}{l}\int_{-l}^{l} f(x)\cdot\sin\frac{n\pi x}{l}dx$$

f(x) – четная => $a_0 = \frac{2}{l}\int_{0}^{l} f(x)dx$ $a_n = \frac{2}{l}\int_{0}^{l} f(x)\cdot\cos\frac{n\pi x}{l}dx$

все $b_n = 0$

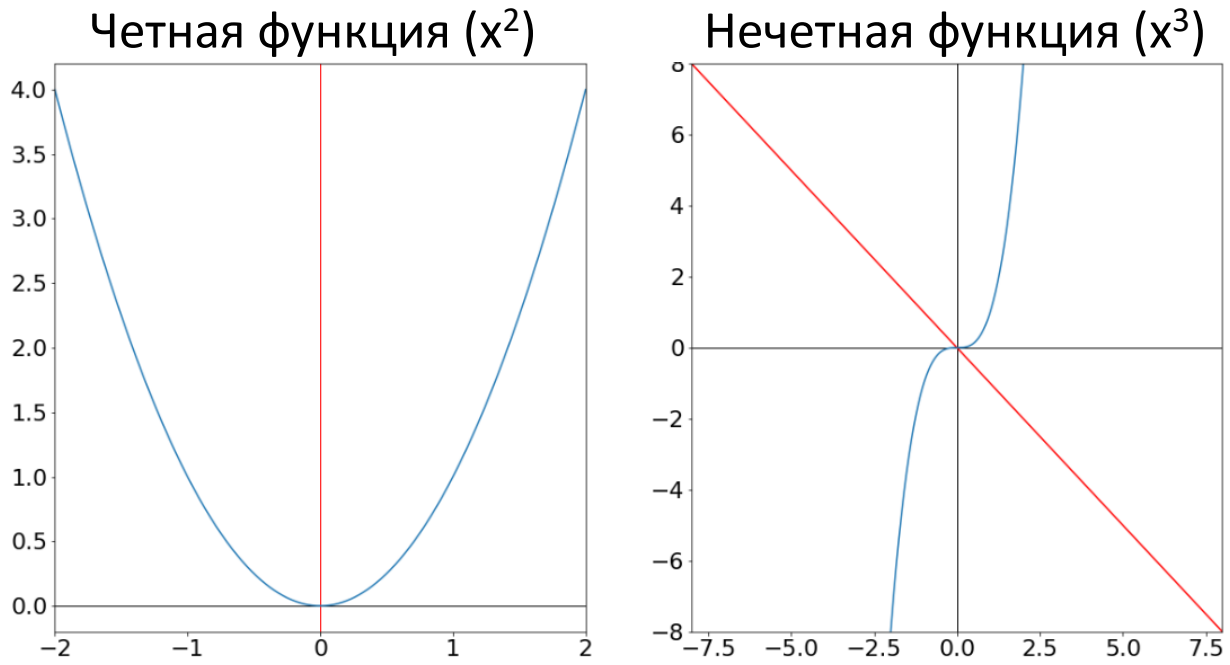
Ряд по косинусам: $S(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos\frac{n\pi x}{l},$

f(x) – нечетная => $b_n = \frac{2}{l}\int_{0}^{l} f(x)\cdot\sin\frac{n\pi x}{l}dx$

$a_0 = 0$, все $a_n = 0$

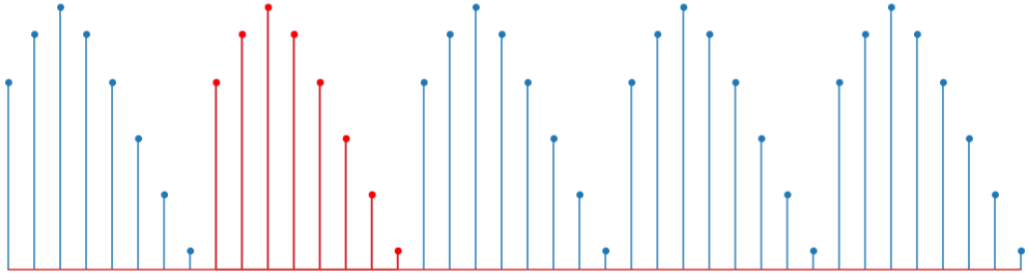
Ряд по синусам: $S(x) = \sum_{n=1}^{\infty} b_n \sin\frac{n\pi x}{l}$

Ряды называются неполными тригонометрическими рядами Фурье



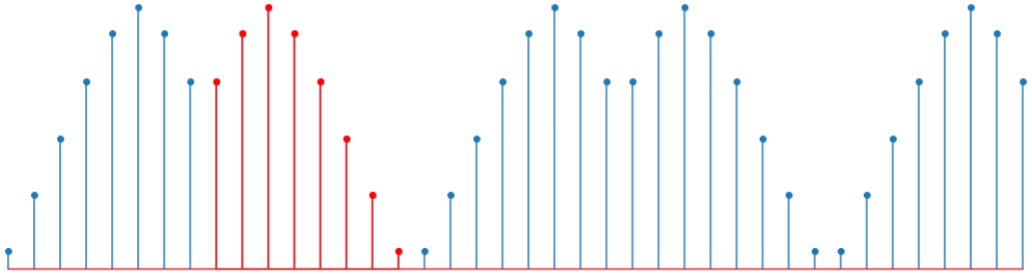
Четные функции симметричны относительно оси y, а нечетные – относительно начала координат.

При расчете полного преобразования Фурье предполагается, что функция, по которой происходит вычисление, повторяется бесконечно. Однако преобразования для четных и нечетных функций позволяют учесть симметрию сигнала. Косинусное преобразование предполагает, что функция продлевается за счет четной симметрии, а для синусоидального – за счет нечетной симметрии.



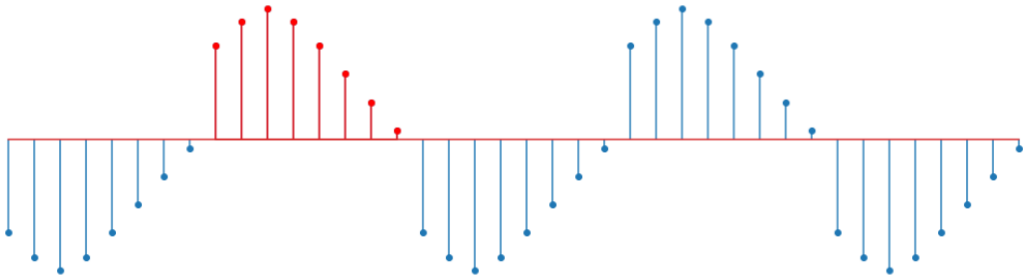
Полное преобразование

повторяет функцию как есть



Косинусное преобразование

отражает функцию по вертикали



Синусоидальное преобразование

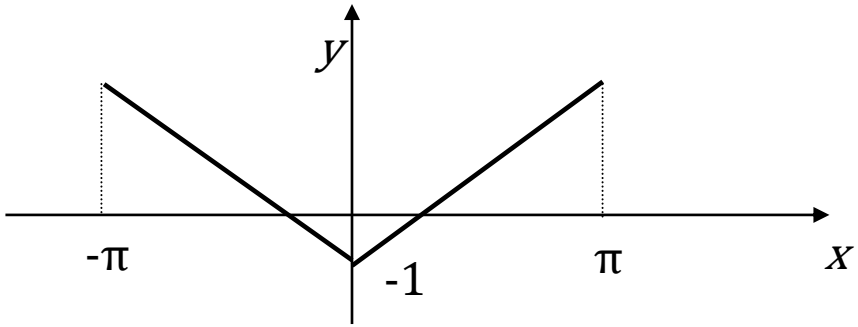
отражает функцию по горизонтали

Пусть f(x) – задана на полуинтервале [0 ; l]
Продолжить f(x) на промежуток [– l ; 0] можно произвольным образом, но принято это делать четным или нечетным образом.

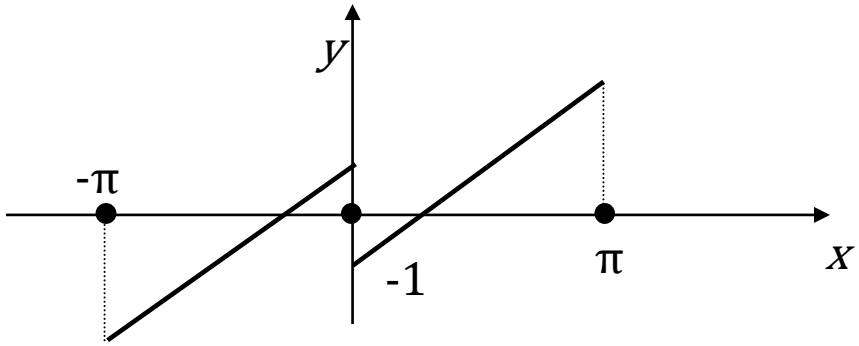
Для функций, отличных от нуля в начале координат, применяют четное продолжение, чтобы избежать разрыва функции в точке $x = 0$.

Для функций, равных нулю в начале координат, променяют нечетное продолжение, т.к. при этом непрерывной оказывается и производная функции в точке $x = 0$.

пример: f(x) = x - 1 задана на [0, π]



Продолжение четным образом

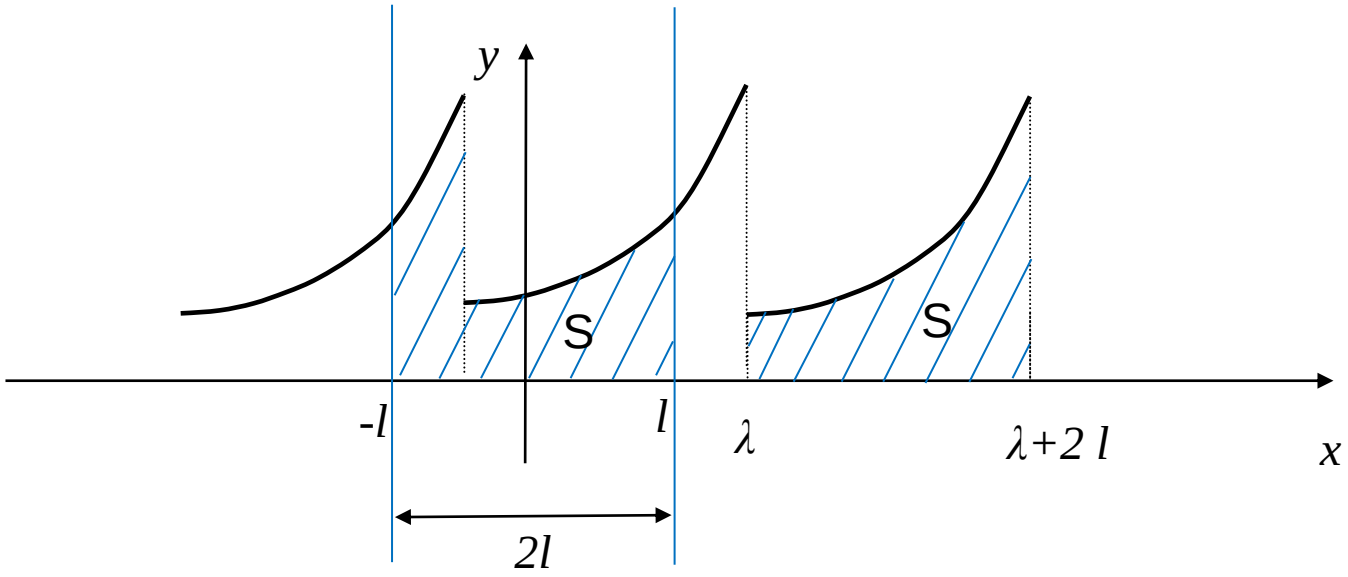


Продолжение нечетным образом

Разложение функций в ряд Фурье на интервале [0, l]

Функцию, заданную на интервале [0, l] доопределяют на интервал [-l, 0] чётным образом, если $f(0) \neq 0$
и нечётным образом, если $f(0) = 0$
Вне интервала [-l, l] продолжают периодическим образом.
Проверяют выполнение условий Дирихле для полученной функции, вычисляют коэффициенты Фурье по соответствующим формулам и записывают ряд Фурье.

Функция задана на произвольном промежутке.
Пусть задана периодическая функция f(x) с периодом T = 2l
на интервале [l ; l + 2l].



$$\int_{-l}^{l} f(x)dx = \int_{\lambda}^{\lambda+2l} f(x)dx$$

При вычислении коэффициентов Фурье для периодической функции f(x), заданной на интервале [l ; l + 2l], в силу геометрического смысла определенного интеграла можно пользоваться формулами:

$$a_0 = \frac{1}{l}\int_{\lambda}^{\lambda+2l} f(x)dx$$

$$a_n = \frac{1}{l}\int_{\lambda}^{\lambda+2l} f(x)\cdot\cos\frac{n\pi x}{l}dx$$

$$b_n = \frac{1}{l}\int_{\lambda}^{\lambda+2l} f(x)\cdot\sin\frac{n\pi x}{l}dx$$

Разложение непериодических функций в ряд Фурье	Функцию, заданную на интервале [a, a + 2l], вне интервала [a, a + 2l] продолжают периодическим образом. Проверяют выполнение условий Дирихле для полученной функции, вычисляют коэффициенты Фурье по соответствующим формулам и записывают ряд Фурье.

Замечания

1. Доопределение функции на f(x) четным или нечетным образом позволяет избежать нахождения аналитического выражения функции $\overline{f}(x)$ на $(-\ell\,;\,0)$.

2. Если $f(0) \neq 0$, то f(x) лучше продолжать на $(-\ell\,;\,0)$ четным образом (т.к. 0 в этом случае будет точкой непрерывности функции $\overline{f}(x)$). Если $f(0) = 0$, то продолжать f(x) на $(-\ell\,;\,0)$ можно как четным, так и нечетным образом.

3. Если функция задана на $[0\,;\,\ell)$, то она разлагается в ряд Фурье не единственным образом.

$$f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n + \ldots$$

$$\varphi(x) = b_0 + b_1 x + b_2 x^2 + \ldots + b_n x^n + \ldots$$

$$f(x)\varphi(x) = a_0 b_0 + (a_0 b_1 + a_1 b_0)x + (a_0 b_2 + a_1 b_1 + a_2 b_0)x^2 + \ldots +$$

$$(a_0 b_n + a_1 b_{n-1} + \ldots + a_n b_0)x^n + \ldots$$

Значение sin и cos:

$$\cos n\pi = (-1)^n;$$

$$\cos\left(n\frac{\pi}{2}\right) = \begin{cases} 0, & n = 2k+1, \\ (-1)^{\frac{n}{2}}, & n = 2k; \end{cases}$$

$$\sin\left(n\frac{\pi}{2}\right) = \begin{cases} 0, & n = 2k, \\ (-1)^{\frac{n-1}{2}}, & n = 2k+1. \end{cases}$$

Ряды Фурье часто применяются в комплексной форме записи.
Преобразуем ряд

$$\frac{a_0}{2}+\sum_{n=1}^{\infty}(a_n\cos nx+b_n\sin nx)$$

и его коэффициенты

$$a_n=\frac{1}{\pi}\int_{-\pi}^{\pi}f(x)\cos nxdx,\quad n=0,1,2,\dots,$$

$$b_n=\frac{1}{\pi}\int_{-\pi}^{\pi}f(x)\sin nxdx,\quad n=1,2,\dots$$

к комплексной форме. Для этого используем формулы Эйлера, выражающие косинус и синус через показательную функцию:

$$\cos nx=\frac{e^{inx}+e^{-inx}}{2},\quad \sin nx=\frac{e^{inx}-e^{-inx}}{2i}.$$

Подставив эти выражения в обычный тригонометрический ряд, находим

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cdot \frac{e^{inx} + e^{-inx}}{2} + b_n \cdot \frac{e^{inx} - e^{-inx}}{2i} =$$

$$= \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cdot \frac{e^{inx} + e^{-inx}}{2} - ib_n \cdot \frac{e^{inx} - e^{-inx}}{2} =$$

$$= \frac{a_0}{2} + \sum_{n=1}^{\infty} \frac{(a_n - ib_n)e^{inx}}{2} + \frac{(a_n + ib_n)e^{-inx}}{2} =$$

$$= \frac{a_0}{2} + \sum_{n=1}^{\infty} c_n e^{inx} + c_{-n} e^{-inx}$$

где обозначено

$$c_n = \frac{a_n - ib_n}{2},\ c_{-n} = \frac{a_n + ib_n}{2}.$$

Найдем выражения для комплексных коэффициентов c_n и c_{-n}.
Используя выражения для c_n и c_{-n} и формулы для коэффициентов a_n, b_n, получим

$$c_n = \frac{1}{2}\left(\frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos nx\,dx - i\frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\sin nx\,dx\right) =$$

$$= \frac{1}{2\pi}\int_{-\pi}^{\pi} f(x)(\cos nx - i\sin nx)\,dx = \frac{1}{2\pi}\int_{-\pi}^{\pi} f(x)e^{-inx}dx$$

где $n = \pm 1, \pm 2, \pm 3, \ldots$.

Вычисление c_0 производится по формуле

$$c_n = \frac{a_n - ib_n}{2},\ c_{-n} = \frac{a_n + ib_n}{2}.$$

Формулу для разложения функции в ряд Фурье можно переписать в виде

$$f(x) = c_0 + \sum_{n=1}^{\infty} c_n e^{inx} + c_{-n} e^{-inx}$$

или

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{inx}$$

Это равенство называется комплексной формой ряда Фурье функции f(x)

Коэффициенты такого ряда можно записать в виде

$$c_n = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(x) e^{-inx} dx \quad (n = 0, \pm 1, \pm 2, \ldots)$$

числа c_n - комплексные коэффициенты ряда Фурье.

Заметим,

$$a_n = c_n + c_{-n}, \quad b_n = i\,(c_n - c_{-n})$$

На случай комплексной формы переносятся все свойства, полученные для обычных рядов.

Прежде всего это: Теорема Дирихле. Если 2π - периодическая функция $f(x)$ на интервале $(-\pi; \pi)$ удовлетворяет условиям Дирихле:

1. $f(x)$ кусочно-непрерывна, т. е. непрерывна или имеет конечное число точек разрыва I рода;
2. $f(x)$ кусочно-монотонна, т. е. монотонна на всем отрезке, либо этот отрезок можно разбить на конечное число интервалов так, что на каждом из них функция монотонна,

то соответствующий функции $f(x)$ ряд Фурье сходится на этом отрезке.

Отметим, что при разложении четных и нечетных 2π – периодических функций удобно использовать действительную форму ряда Фурье.
Если функция f(x) - четная, то ее ряд Фурье имеет вид

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos nx$$

где

$$a_0 = \frac{2}{\pi} \int_0^{\pi} f(x)\, dx, \quad a_n = \frac{2}{\pi} \int_0^{\pi} f(x) \cos nx\, dx, \quad n \in \mathbb{N}$$

Если функция f(x) - нечетная, то ее ряд Фурье имеет вид

$$f(x) = \sum_{n=1}^{\infty} b_n \sin nx$$

где

$$b_n = \frac{2}{\pi} \int_0^{\pi} f(x) \sin nx\, dx, \quad n \in \mathbb{N}$$

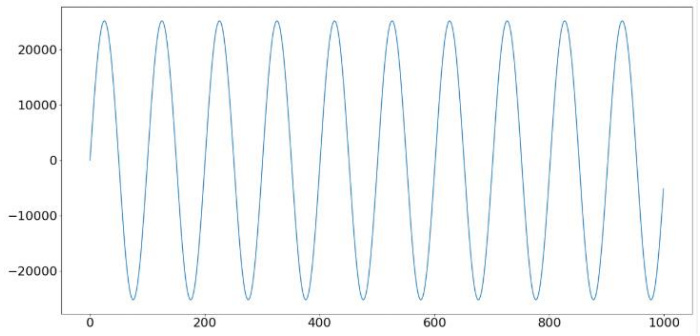
Система функций e^{-inx} $(n = 0, \pm 1, \pm 2, \ldots)$ является полной ортонормированной системой в пространстве функций удовлетворяющих условиям Дирихле на интервале (-π,π]. Это вытекает из равенства

$$\frac{1}{2\pi}\int_{-\pi}^{\pi} e^{inx}e^{-imx}dx = \frac{1}{2\pi}\int_{-\pi}^{\pi} e^{i(n-m)x}dx = \delta_{nm},$$

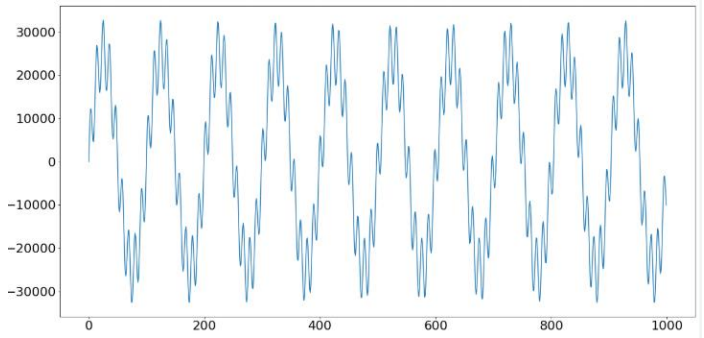
где δ_{nm} – символ Кронекера

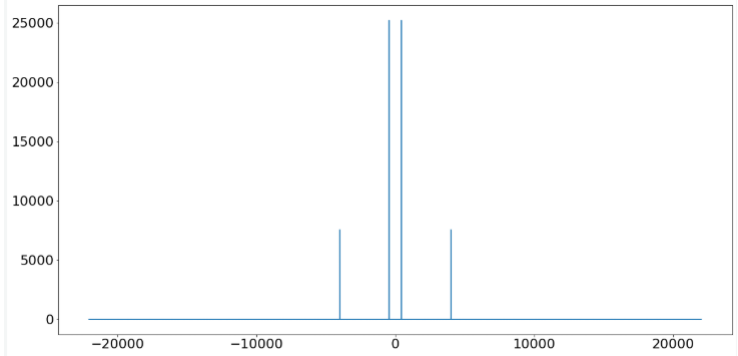
$$\delta_{nm} = \begin{cases} 0 & n \neq m, \\ 1 & n = m \end{cases}$$

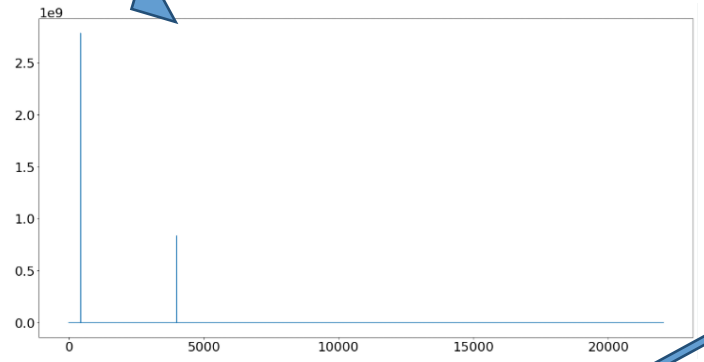
и теоремы Дирихле.



+ Шум

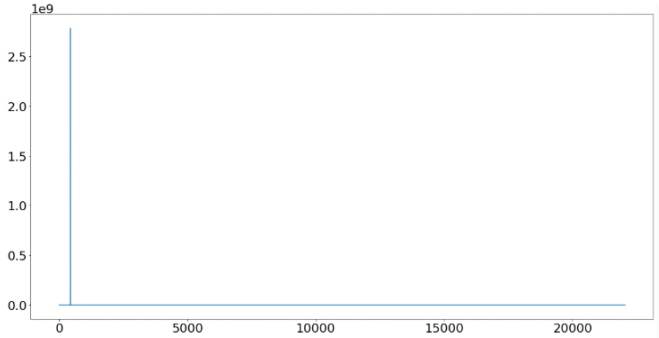


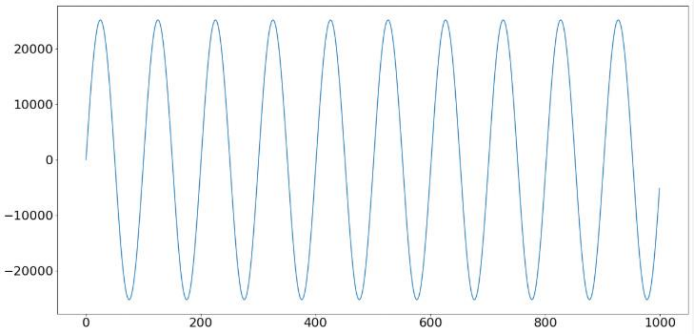




+ Фильтрация

Дискретное или быстрое преобразование Фурье





Домашняя работа #8

Обработать два зашумленных аудиосигнала, удалив в них шум путём применения дискретного преобразования Фурье и сохранив в виде новых wav-файлов. Ссылка на исходные wav-файлы для обработки: https://cloud.mail.ru/public/LVfj/bwpMydCv7

Срок сдачи работ: **до 1-го июня**.

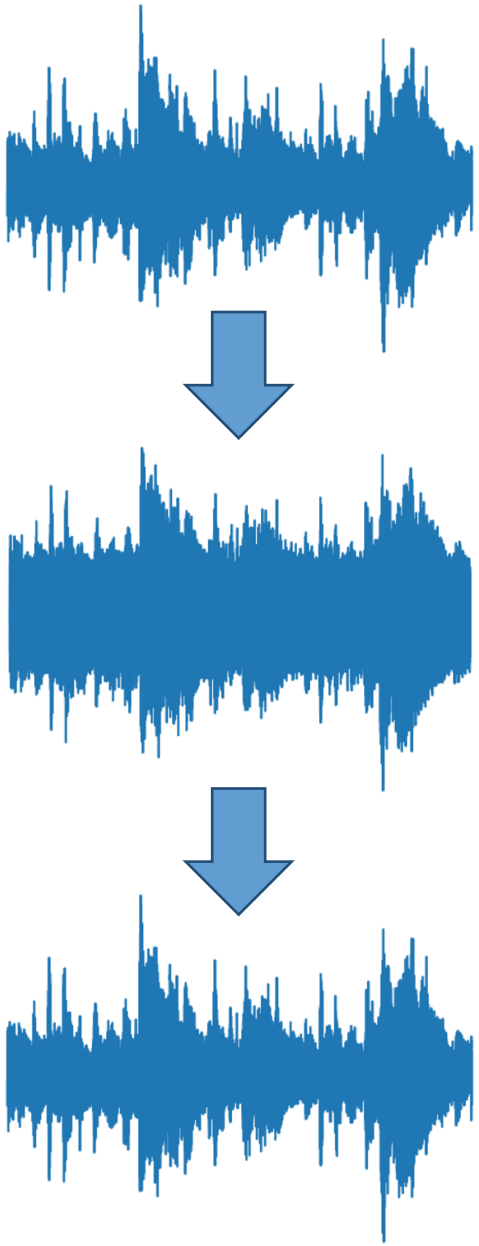
Работы принимаются на электронный адрес VSChernyshenko@fa.ru в виде **двух файлов .wav**, а также самого ноутбука – в формате **html** или **pdf**. Название файла: “Фамилия_Имя_группа_ДЗ8.*”

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

Домашняя работа #8

В рамках домашней работы, для каждого из двух corrupt_sound wav-файлов ожидается:

- Разбитие зашумленных аудиосигналов на фрагменты
- Вычисление преобразование Фурье для каждого фрагмента
- Проверка того, какие гармоники отвечают за шум
- Удаление этих гармоник из аудиосигнала
- Применение обратного преобразования Фурье для обновлённых аудиофрагментов
- Объединение всех фрагментов вместе, сохранение их в новый wav-файл

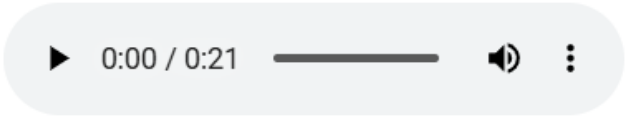


Ожидается, что зашумление для corrupt_sound1 снизится значительно; шум в corrupt_sound2 будет удалён полностью. Качество полученных результатов будет определять оценивание работы.

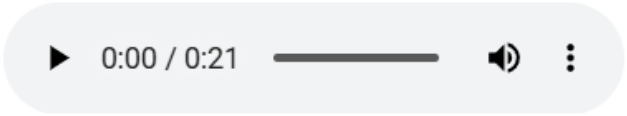
```
import scipy.io.wavfile as siowav
import IPython.display as ipd

# Чтение сохранённых WAV-файлов
sr1, corrupt_sound1 = siowav.read("corrupt_sound1.wav")
print(f"Частота дискретизации: {sr1}, Форма массива: {corrupt_sound1.shape}")

sr2, corrupt_sound2 = siowav.read("corrupt_sound2.wav")
print(f"Частота дискретизации: {sr1}, Форма массива: {corrupt_sound2.shape}")
```

```
Частота дискретизации: 48000, Форма массива: (1048576,)
Частота дискретизации: 48000, Форма массива: (1048576,)
```

```
# Воспроизведение аудио
audio = ipd.Audio(corrupt_sound1, rate=sr1)
display(audio)
```

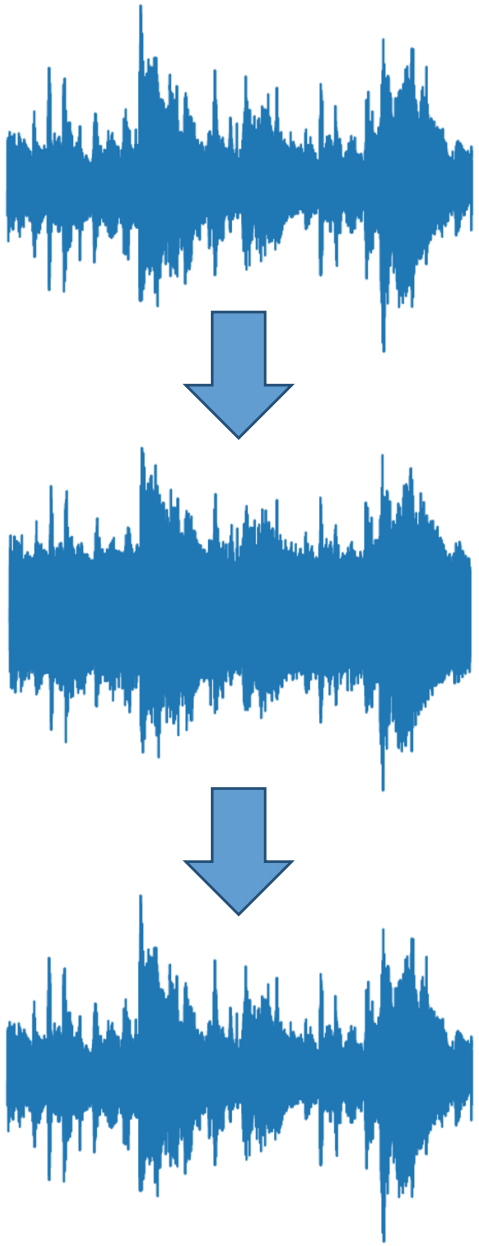



```
audio = ipd.Audio(corrupt_sound2, rate=sr2)
display(audio)
```




```
# При помощи разложения Фурье, разбивая исходные corrupt_sound на небольшие части,
# попытайтесь уменьшить уровень шуми в corrupt_sound1 и
# полностью удалить шум в corrupt_sound2.
```

```
# Сохранение аудиопотока на диск
siowav.write("output_new.wav", sr_new, data_new)
print("Аудиофайл сохранён как 'output_new.wav'")
```



Домашняя работа #8

Обязательные условия :

- Разрешается использование методов пакета matplotlib, math, следующих методов пакета Numpy: array, linspace, meshgrid, zeros, zeros_like, shape, random, sqrt, log, exp, sin, cos, tan, scipy.io.wavfile, IPython.display. Наличие иных методов приводит к аннулированию оценки работы.
- В ноутбуке укажите, как вы обнаружили частоты, отвечающие за шум, приведите краткое обоснование.

Для произвольной непрерывной, периодической с периодом 2π функции имеет место неравенство Бесселя

$$\sum_{n=-\infty}^{+\infty} |c_n|^2 \le \frac{1}{2\pi} \int_0^{2\pi} |f(x)|^2 \, dx.$$

Если функция разложима в ряд Фурье, то имеет место равенство Парсеваля:

$$\sum_{n=-\infty}^{+\infty} |c_n|^2 = \frac{1}{2\pi} \int_0^{2\pi} |f(x)|^2 \, dx.$$

Длительность сигнала и ширина его спектра подчиняются соотношению неопределенности: произведение этих параметров (база спектра) не может быть меньше единицы.

Ограничений максимального значения базы сигнала не существует, поэтому можно сформировать сигнал большой длительности с широким спектром, а короткий сигнал с узким спектром существовать не может.

Линейность – преобразование Фурье является линейным интегральным преобразованием, спектр суммы сигналов равен сумме их спектров:

$$s(t) = \alpha f(t) + \beta g(t);$$

$$\boldsymbol{S}(\omega) = \alpha \boldsymbol{F}(\omega) + \beta \boldsymbol{G}(\omega).$$

Задержка сигнала во времени на величину τ не влияет на амплитудный спектр сигнала, добавляя в фазовый спектр слагаемое -ωτ, линейно зависящее от частоты:

$$s(t) = f(t - \tau);$$

$$\boldsymbol{S}(\omega) = \boldsymbol{F}(\omega)e^{-j\omega\tau}$$

Изменение масштаба оси времени (длительности сигнала) – приводит к изменению ширины спектра в обратную сторону с одновременным изменением уровня спектральных составляющих:

$$s(t) = f(at)$$

$$\boldsymbol{S}(\omega) = \frac{1}{|a|}\boldsymbol{F}\left(\frac{\omega}{a}\right), \qquad a \neq 0$$

Изменение длительности сигнала *не может* быть осуществлено *линейной системой* с постоянными параметрами.

Дифференцирование сигнала во временной области приводит к умножению спектра исходного сигнала на $j\omega$, т.о. низкие частоты ослабляются, а высокие усиливаются, фазовый спектр сдвигается на $\pm\pi$ для положительных и отрицательных частот соответственно.

$$s(t) = \frac{df}{dt},$$

$$\boldsymbol{S}(\omega) = j\omega \boldsymbol{F}(\omega)$$

Множитель $j\omega$ называется оператором дифференцирования сигнала в частотной области.

Интегрирование сигнала ведет к ослаблению высоких и усилению низких частот, фазовый спектр смещается на -/+π для положительных и отрицательных частот соответственно:

$$s(t) = \int_{-\infty}^{\infty} f(t)dt\,,$$

$$\boldsymbol{S}(\omega) = \frac{\boldsymbol{F}(\omega)}{j\omega} + \pi \boldsymbol{F}(0)\delta(\omega)$$

Дополнительное слагаемое – дельта-функция на нулевой частоте, умноженная на постоянную составляющую сигнала *s(t)*.

Спектр свертки сигналов равен произведению спектров:

$$s(t) = \int_{-\infty}^{\infty} f(t')g(t-t')dt$$

$$\boldsymbol{S}(\omega) = \boldsymbol{F}(\omega)\boldsymbol{G}(\omega)$$

Свертка сигнала описывает прохождение сигнала через линейную систему с постоянными параметрами.

- **Умножение сигнала на гармоническую функцию** приводит к раздвоению спектра – он распадается на два сигнала вдвое меньшего уровня каждый, смещенных вправо и влево на ω_0 по оси частот:

$$s(t) = f(t)\cos(\omega_0 t + \varphi_0)$$

$$\boldsymbol{S}(\omega) = \frac{1}{2}e^{j\varphi_0}\boldsymbol{F}(\omega - \omega_0) + \frac{1}{2}e^{-j\varphi_0}\boldsymbol{F}(\omega + \omega_0)$$

- Связь преобразования Фурье и коэффициентов ряда Фурье:

$$C_k = \frac{1}{T} \boldsymbol{S}\left(\frac{2\pi k}{T}\right)$$

Преобразование Фурье применяется для вычисления спектра сигнала, являющегося функцией времени или пространственных координат.

Дискретный сигнал представляет собой последовательность чисел, которую для проведения Фурье-преобразований необходимо представить в виде некоторой функции.

Обычно отсчеты представляют в виде дельта-функций с определенными множителями и задержками, что для последовательности отсчетов {x(k)} можно представить в виде:

$$s(t) = \sum_{k=-\infty}^{\infty} x(k)\delta(t-k)$$

$$\boldsymbol{S}(\omega) = \sum_{k=-\infty}^{\infty} x(k)e^{-j\omega k}$$

Главное свойство спектра любого дискретного сигнала – его периодичность.

Характер спектра дискретизированного сигнала демонстрирует частотно-временную дуальность преобразования Фурье:

- Периодический сигнал имеет дискретный спектр;
- Дискретный сигнал имеет периодический спектр.

ДПФ специально предназначено для работы с дискретными сигналами.

- *Лежит в основе различных технологий спектрального анализа для исследования случайных процессов.*

В результате вычисления ДПФ случайного процесса (сигнала) получается лишь спектр его единственной (одной из возможных) реализаций, что обычно не представляет большого интереса.

Для спектрального анализа случайных сигналов необходимо использовать **усреднение спектра**.

Методы спектрального анализа, в которых после усреднения сигнала используется только информация, извлеченная из самого входного сигнала, называются **непараметрическими**.

Если при проведении усреднения случайного сигнала определена некоторая его статистическая модель, спектральный анализ будет также решать задачи определения параметров этой модели. Такие методы называются **параметрическими**.

Периодический дискретный сигнал, описываемый конечным набором из N чисел, имеет дискретный периодический спектр, один период спектра имеет N гармоник.

Рассмотрим периодическую последовательность отсчетов {x(k)} с периодом N:

$$x(k+N) = x(k)$$

для любого k.

Поставим в соответствие этой последовательности сигнал из смещенных по времени дельта-функций:

$$s(t) = \sum_{k=-\infty}^{\infty} x(k)\delta(t-kT),$$

также периодический с периодом NT.

Спектр дискретного сигнала s(t) периодический с периодом $2\pi/T$, а поскольку и сигнал периодический, то его спектр дискретен с расстояниями между гармониками $2\pi/NT$.

Применив к дискретному периодическому сигналу s(t) разложение в ряд Фурье, получим дискретное преобразование Фурье, т.е. разложение сигнала по гармоникам:

$$\boldsymbol{X}(n) = \sum_{k=0}^{N-1} x(k)\exp(-j\frac{2\pi nk}{N})$$

Обратное дискретное преобразование Фурье:

$$x(k) = \frac{1}{N}\sum_{n=0}^{N-1} \boldsymbol{X}(n)\exp(j\frac{2\pi nk}{N})$$

В целом аналогичны свойствам непрерывного преобразования Фурье, например линейность, задержка (сдвиг) сигнала, симметрия, произведение последовательностей. Но есть и некоторые нюансы, возникающие вследствие дискретности, например:

- при перемножении сигналов их длины должны быть одинаковыми (N);
- суммирование элементов произведения должно производиться по одному периоду (полученный результат называется круговой сверткой спектров исходных сигналов).

Дискретное преобразование Фурье является спектром дискретного периодического сигнала и позволяет восстановить непрерывный периодический сигнал, занимающий некоторую ограниченную полосу частот. Заменив дискретный параметр k (номер отсчета) в формуле обратного ДПФ на непрерывный t/T, где T – период дискретизации, получим непрерывный сигнал:

$$x(t) = \frac{1}{N} \sum_{n=-N/2}^{N/2-1} \boldsymbol{X}(n) \exp\left(j \frac{2\pi n t}{TN}\right)$$

Такой аналоговый сигнал занимает полосу частот от 0 до π/T.

$$\boldsymbol{X}(n) = \sum_{k=0}^{N-1} x(k)\exp(-j\frac{2\pi nk}{N})$$

Из выражений ДПФ можно видеть, что для вычисления каждой гармоники нужно N операций комплексного умножения и сложения и соответственно N^2 операций на полное выполнение ДПФ.

При больших объемах массивов данных это может приводить к существенным временным затратам.

Ускорение вычислений достигается при использовании **быстрого преобразования Фурье** (БПФ).

Домашняя работа #9

Проанализировать изменение трендов потребления энергопотребления на основе «десезонированных» данных путём применения быстрого преобразования Фурье.

Ссылка на датасет для обработки: https://cloud.mail.ru/public/mnqr/wCE6wkzXj

Разрешается использование базовых методов numpy, matplotlib, IPython.display, sklearn.linear_model, sklearn.preprocessing не более.

Срок сдачи работ: **до 8-го июня**.

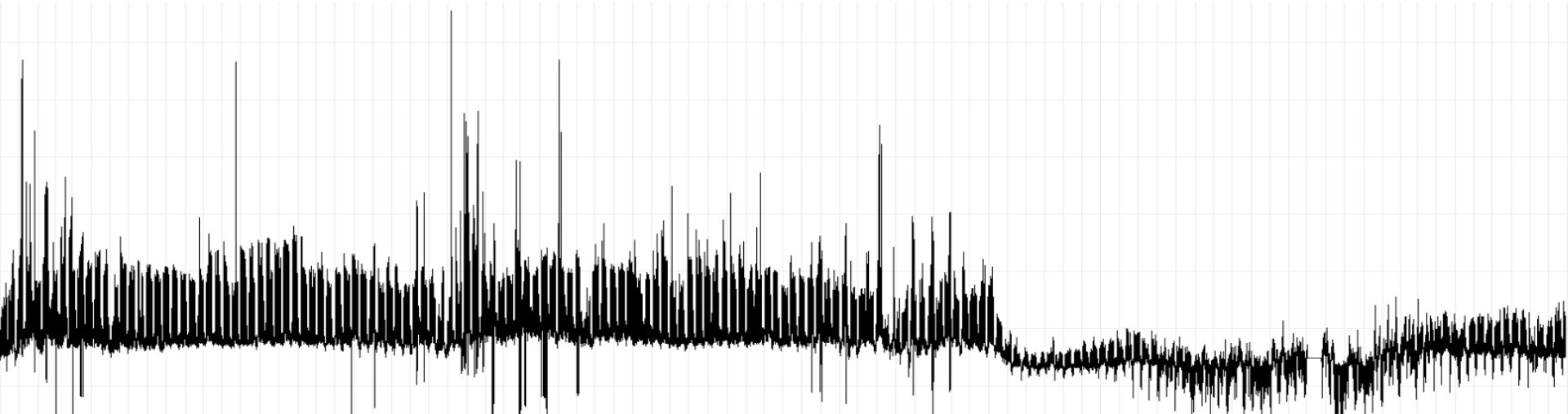
Работы принимаются на электронный адрес VSChernyshenko@fa.ru в виде ноутбука – в формате html или pdf. Название файла: “Фамилия_Имя_группа_ДЗ9.*”

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

Домашняя работа #9

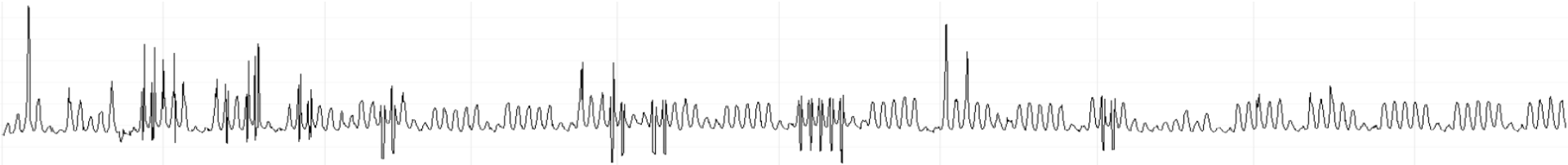
Список признаков в датасете Electricity_consumption.csv.

1. Timestamp – метка времени
2. **demand_kW** - потребляемая мощность (кВт), *экзогенная переменная*
3. apparent_temperature – температура по ощущениям
4. air_temperature - температура воздуха
5. dew_point_temperature - температура точки росы
6. relative_humidity - относительная влажность
7. wind_speed - скорость ветра



Домашняя работа #9

1. Загрузка и предобработка данных из Electricity_consumption.csv
2. Выполнить быстрое преобразование Фурье для удаления сезонных компонентов (суточных, недельных и годовых циклов) в отношении экзогенной переменной – demand_kW по подобранному threshold на базе анализа частотной спектрограммы.
3. Выполнить обратное преобразование.
4. Построить многомерную модель методами sklearn, описывающую полученные на предыдущем шаге данные с использованием признаков о погоде и т.п.
5. Сгладить остатки.
6. Построить графики исходного временного ряда, «десезонированного» ряда, остатков модели, сглаженных остатков. Наложить сглаженные остатки на исходный ряд после приведения их к единому масштабу.
7. Написать отчёт в отдельной ячейке Markdown,



Домашняя работа #9

Обязательные условия :

- Разрешается использование методов пакета matplotlib, math, следующих методов пакета Numpy: array, linspace, meshgrid, zeros, zeros_like, shape, sqrt, log, exp, sin, cos, tan. Наличие иных методов приводит к аннулированию оценки работы.
- В ноутбуке напишите отчёт, включающий в себя:
  - описание того как вы обнаружили частоты, отвечающие за сезонность;
  - описание построенной модели;
  - описание метода сглаживания временного ряда остатков модели;
  - характеристику природы изменений трендов;
  - описательный прогноз дальнейшего движения спроса на электричество.