И.Х. Хуснуллин

ОСНОВНЫЕ МЕТОДЫ ПРОЕКТНО-ТЕХНОЛОГИЧЕСКОЙ РАБОТЫ ПО МАТЕМАТИКЕ

Учебное пособие

Уфа 2026

УДК 51-7

Хуснуллин И.Х. Основные методы проектно-технологической работы по математике: учеб. пособие [Текст]. – Уфа: Изд-во БГПУ, 2025. – 95с.

Излагаются основы научных исследований по математике с применением языка программирования Python. Приводятся и демонстрируются основные возможности библиотек NumPy, Matplotlib, Seaborn, SciPy, SymPy для визуализации данных, для научных расчётов и символьных вычислений. Рассматриваются примеры решения учебных, научно-исследовательских и профессиональных задач по математике и смежным дисциплинам с помощью этих библиотек.

Предназначено для практических занятий и самостоятельной работы для студентов направления 03.03.01 Прикладные математика и физика Направленность (профиль) Компьютерные технологии и интеллектуальный анализ данных. После каждой главы приведены индивидуальные задания для самостоятельного решения.

Рецензенты:

Е.Г. Кудашева, *к.ф.-м.н., доцент (БГПУ им.М.Акмуллы);*

В.Ф. Вильданова, *к.ф.-м.н., доцент (ИМВЦ УФИЦ РАН).*

ISBN

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Python — один из популярных языков для научных вычислений. Он предоставляет инструменты для решения сложных математических, научных и инженерных задач.

Некоторые области применения Python для научных вычислений:

- **Наука о данных и аналитика**. Python используют для очистки, обработки и визуализации данных, а также для применения моделей машинного обучения.
- **Физика и инженерия**. Численные возможности Python позволяют решать сложные физические и инженерные задачи. Например, для моделирования, решения дифференциальных уравнений и выполнения задач линейной алгебры используют SciPy и NumPy.
- **Машинное обучение и искусственный интеллект**. Такие библиотеки, как TensorFlow, Keras и PyTorch, позволяют учёным создавать и обучать сложные модели для таких задач, как обработка естественного языка, компьютерное зрение и прогнозная аналитика.
- **Биоинформатика**. Библиотеки Python, такие как BioPython, используют в биоинформатике для таких задач, как анализ последовательности, предсказание структуры белка и анализ экспрессии генов.
- **Геопространственный анализ**. Python применяют для анализа и визуализации географических данных. Библиотеки, такие как GeoPandas и Shapely, помогают с манипулированием и анализом данных географических информационных систем (ГИС).

Для научных вычислений на Python используют различные библиотеки, например: NumPy, SciPy, Matplotlib, Pandas и другие.

NumPy (Numerical Python) — это библиотека Python с открытым исходным кодом, которая используется практически во всех областях науки и техники. Это универсальный стандарт для работы с числовыми данными в Python, лежащий в основе научных экосистем Python и PyData. Пользователи NumPy включают всех, от начинающих программистов до опытных исследователей, занимающихся современными научными и промышленными исследованиями и разработками.

Библиотека NumPy содержит многомерные массивы и матричные структуры данных. NumPy можно использовать для выполнения множества математических операций с массивами. Он добавляет в Python мощные структуры данных, гарантирующие эффективные вычисления с массивами и матрицами, и предоставляет огромную библиотеку высокоуровневых математических функций, которые работают с этими массивами и матрицами.

Вот несколько областей, где NumPy активно используется:

- Научные и инженерные вычисления;
- Машинное обучение и искусственный интеллект;
- Обработка данных;
- Визуализация данных;
- Высокопроизводительные вычисления;

SciPy (сокращение от Scientific Python, или научный Python) – библиотека, которая является расширением библиотеки **NumPy** и предназначена для выполнения сложных инженерных, статистических и научных расчетов, а так же для анализа данных и построения графиков. **NumPy** выполняет стандартные операции, такие как сортировка, индексирование, а так же элементарные операции, связанные с типом данных массива. **SciPy** используется для выполнения сложных операций, например, расчет алгебраических функций или различных числовых алгоритмов.

Matplotlib — мультиплатформенная библиотека для визуализации данных, основанная на массивах библиотеки **NumPy** и спроектированная в расчете на работу с обширным стеком **SciPy**.

Библиотека Matplotlib зарекомендовала себя как невероятно удобный и популярный инструмент визуализации, но даже заядлые ее пользователи признают, что она зачастую оставляет желать лучшего. Библиотека **Seaborn** — решение этих проблем. **Seaborn** предоставляет API поверх библиотеки Matplotlib, обеспечивающий разумные варианты стилей графиков и цветов по умолчанию, определяющий простые высокоуровневые функции для часто встречающихся типов графиков и хорошо интегрирующийся с функциональностью, предоставляемой объектами DataFrame библиотеки Pandas.

SymPy — это библиотека Python для выполнения символьных вычислений. Это система компьютерной алгебры, которая может выступать как отдельное приложение, так и в качестве библиотеки для других приложений. Поскольку это чистая библиотека Python, ее можно использовать даже в интерактивном режиме. В **SymPy** есть разные функции, которые применяются в сфере символьных вычислений,

математического анализа, алгебры, дискретной математики, квантовой физики и так далее. **SymPy** может представлять результат в разных форматах: **LaTeX, MathML** и так далее. Вот где применяется **SymPy**:

- Многочлены
- Математический анализ
- Дискретная математика
- Матрицы
- Геометрия
- Построение графиков
- Физика
- Статистика
- Комбинаторика

Символьные вычисления — это разработка алгоритмов для управления математическими выражениями и другими объектами. Такие вычисления объединяют математику и компьютерные науки для решения математических выражений с помощью математических символов. Система компьютерной алгебры же, такая как **SymP**y, оценивает алгебраические выражения с помощью тех же символов, которые используются в традиционных ручных методах.

Проведение как теоретических, так и практических научных исследований по математике студентами, аспирантами включает в себя решение различных математических задач, таких как: нахождение корней алгебраических уравнений, задачи линейной алгебры, вычисление интегралов, решение систем обыкновенных дифференциальных уравнений, аппроксимация и интерполяция данных, оптимизация и т. д. Важным элементом анализа научных результатов является визуализация данных. Найти точное решение математических задач в реальных приложениях зачастую не удаётся, что обусловливает необходимость использования алгоритмов численного решения. Иногда необходимо символьное решение задачи — с помощью тех же символов, которые используются в традиционных ручных методах.

Наиболее известными коммерческими пакетами для проведения научных и инженерных расчётов являются пакет программ MATLAB, системы компьютерной алгебры Mathcad, Maple и Mathematica. Указанные программные пакеты имеют продвинутые и удобные графические интерфейсы, собственные высокоуровневые языки программирования и позволяют решать большое количество математических и инженерных

задач, а также визуализировать данные и получать рисунки высокого качества. Однако возможность использования данных пакетов студентами, индивидуальными исследователями, небольшими организациями и учебными заведениями затруднена по причине высокой стоимости их лицензий, а иногда невозможностью использования в связи с санкциями.

Существует ряд бесплатных программных пакетов и библиотек для проведения научных расчётов, анализа и визуализации данных. Например, описанные выше библиотеки для языка программирования Python NumPy, SciPy, SymPy, Matplotlib, Seaborn итд. Данные библиотеки в настоящее время активно развиваются и являются одним из лучших бесплатных инструментов для решения стандартных математических задач.

Целью данного учебного пособия является ознакомление студентов – будущих учителей математики с основными возможностями библиотек NumPy, SciPy, SymPy, Matplotlib, Seaborn для решения учебных, профессиональных и научно исследовательских задач по математике и смежным дисциплинам. Для лучшего освоения материала в конце каждой главы приводятся индивидуальные задания для самостоятельного решения. Сами задания не приводятся в данном пособии, а доступны по ссылке или QR коду, достаточно просто навести камеру телефона на QR код для получения заданий.

Структура пособия следующая. В первой главе описываются основные возможности библиотеки NumPy для решения задач линейной алгебры: операции над матрицами, определитель матрицы, системы линейных уравнений, собственные значения матрицы, норма, комплексные числа, а также приложения матриц в аналитической геометрии. Во второй главе описываются основные возможности библиотеки SciPy для решения задач интегрального исчисления: вычисление одномерных, двумерных и трехмерных интегралов. В третей и четвертой главах описываются основные возможности библиотеки Matplotlib и Seaborn для построения всевозможных графиков функций, диаграмм, гистограмм, контурных графиков итд. В пятой главе описываются основные возможности библиотеки SymPy для символьных преобразований: сокращение дробей, раскрытие скобок, разложение на множители. В шестой главе — основные возможности библиотеки SymPy для символьного решения уравнений: линейных, нелинейных, систем линейных и нелинейных уравнений, дифференциальных уравнений. Содержание всех глав полезны для решения учебных и научно-исследовательских задач, а пятая и шестая

главы и для решения профессиональных задач студентов педагогических направлений.

Установка и настройка **Jupiter Notebook**:

Начало работы с **Google Colab**:

Линейная алгебра в NumPy

Что же такое NumPy?

NumPy (Numerical Python) — это библиотека Python с открытым исходным кодом, которая используется практически во всех областях науки и техники. Это универсальный стандарт для работы с числовыми данными в Python, лежащий в основе научных экосистем Python и PyData. Пользователи NumPy включают всех, от начинающих программистов до опытных исследователей, занимающихся современными научными и промышленными исследованиями и разработками.

Библиотека NumPy содержит многомерные массивы и матричные структуры данных. NumPy можно использовать для выполнения множества математических операций с массивами. Он добавляет в Python мощные структуры данных, гарантирующие эффективные вычисления с массивами и матрицами, и предоставляет огромную библиотеку высокоуровневых математических функций, которые работают с этими массивами и матрицами.

Вот несколько областей, где NumPy активно используется:

- Научные и инженерные вычисления;
- Машинное обучение и искусственный интеллект;
- Обработка данных;
- Визуализация данных;
- Высокопроизводительные вычисления.

Линейная алгебра в NumPy

Создание матриц

Самый простой способ — с помощью функции numpy.array(list, dtype=None, …).

В качестве первого аргумента ей надо передать итерируемый объект, элементами которого являются другие итерируемые объекты одинаковой длины и содержащие данные одинакового типа.

Второй аргумент является опциональным и определяет тип данных матрицы. Его можно не задавать, тогда тип данных будет определен из типа элементов первого аргумента. При задании этого параметра будет произведена попытка приведения типов.

```
import numpy as np # подключение библиотеки NumPy

a = np.array([1, 2, 3])   # Создаем одномерный массив
print(type(a))            # Prints "<class 'numpy.ndarray'>"
print(a.shape)            # Prints "(3,)" - кортеж с размерностями
print(a[0], a[1], a[2])   # Prints "1 2 3"
a[0] = 5                  # Изменяем значение элемента массива
print(a)                  # Prints "[5, 2, 3]"

b = np.array([[1,2,3],[4,5,6]])    # Создаем двухмерный массив
print(b.shape)                     # Prints "(2, 3)"
print(b[0, 0], b[0, 1], b[1, 0])   # Prints "1 2 4"
print(np.arange(1, 5)) #Создает вектор с элементами от 1 до 4
```

```
<class 'numpy.ndarray'>
(3,)
1 2 3
[5 2 3]
(2, 3)
1 2 4
[1 2 3 4]
```

Второй способ создания — с помощью встроенных функций numpy.eye(N, M=None, …), numpy.zeros(shape, …), numpy.ones(shape, …).

Первая функция создает единичную матрицу размера N×M; если M не задан, то M = N.

Вторая и третья функции создают матрицы, состоящие целиком из нулей или единиц соответственно. В качестве первого аргумента необходимо задать размерность массива — кортеж целых чисел. В двумерном случае это набор из двух чисел: количество строк и столбцов матрицы.

```
b = np.eye(5) # Единичная матрица размера 5 на 5
print ("Единичная матрица:\n", b)
```

Единичная матрица:
[[1. 0. 0. 0. 0.]
[0. 1. 0. 0. 0.]
[0. 0. 1. 0. 0.]
[0. 0. 0. 1. 0.]
[0. 0. 0. 0. 1.]]

```
c = np.ones((7, 5))
print ("Матрица, состоящая из одних единиц:\n", c)
```

Матрица, состоящая из одних единиц:
[[1. 1. 1. 1. 1.]
[1. 1. 1. 1. 1.]
[1. 1. 1. 1. 1.]
[1. 1. 1. 1. 1.]
[1. 1. 1. 1. 1.]
[1. 1. 1. 1. 1.]
[1. 1. 1. 1. 1.]]

```
d = np.full((2,2), 7)  # Создает матрицу (2, 2) заполненую заданным значением 7
print(d)
```

[[7 7]
[7 7]]

Третий способ — с помощью функции numpy.arange([start,]stop, [step,], …), которая создает одномерный массив последовательных чисел из промежутка [start, stop) с заданным шагом step, и метода array.reshape(shape).

Параметр shape, как и в предыдущем примере, задает размерность матрицы (кортеж чисел). Логика работы метода ясна из следующего примера:

```
v = np.arange(0, 24, 2)
print ("Вектор-столбец:\n", v)
```

Вектор-столбец:
[0 2 4 6 8 10 12 14 16 18 20 22]

```
d = v.reshape((3, 4))
print ("Матрица:\n", d)
```

```
Матрица:
 [[ 0  2  4  6]
 [ 8 10 12 14]
 [16 18 20 22]]
```

Математические операции над матрицами

К массивам (матрицам) можно применять известные вам математические операции. Следут понимать, что при этом у элементов должны быть схожие размерности. Поведение в случае не совпадения размерностей хорошо описанно в документации numpy.

```
x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)
arr = np.array([1, 2])
```

Сложение матриц

Операция сложения определена для двух матриц, таких что число столбцов и строк первой равно числу столбцов и строк второй.

Пусть матрицы A и B таковы, что $A \in \mathbb{R}^{n \times m}$ и $B \in \mathbb{R}^{n \times m}$. Произведением матриц A и B называется матрица C, такая что $c_{i,\mathrm{j}} = a_{ij} + b_{ij}$, где $c_{i,\mathrm{j}}$ — элемент матрицы A, стоящий на пересечении строки с номером i и столбца с номером j.

```
print(x)
print(y)
```

```
[[1. 2.]
 [3. 4.]]
[[5. 6.]
 [7. 8.]]
```

```
# Сложение происходит поэлементно
print(x + y)
```

```
[[ 6.  8.]
 [10. 12.]]
```

```
print(np.add(x, y)) # Та же операция, что и x + y
```

```
[[ 6.  8.]
 [10. 12.]]
```

```
# Вычитание
print(x - y) #первый способ
print(np.subtract(x, y)) # второй способ
```

```
[[-4. -4.]
 [-4. -4.]]
[[-4. -4.]
 [-4. -4.]]
```

```
# Деление
print('x = ', x)
print('y = ', y)
print('x/y = ', x / y) # первый способ
print('x/y = ', np.divide(x, y)) # второй способ
```

```
x =  [[1. 2.]
 [3. 4.]]
y =  [[5. 6.]
 [7. 8.]]
x/y =  [[0.2        0.33333333]
 [0.42857143 0.5       ]]
x/y =  [[0.2        0.33333333]
 [0.42857143 0.5       ]]
```

Умножение матриц и столбцов

Операция умножения определена для двух матриц, таких что число столбцов первой равно числу строк второй.

Пусть матрицы A и B таковы, что $A \in \mathbb{R}^{n \times m}$ и $B \in \mathbb{R}^{m \times k}$. Произведением матриц A и B называется матрица C, такая что $c_{i,k} = \sum_{j=1}^{mk} a_{ij}\, b_{jk}$, где $c_{i,k}$ — элемент матрицы C, стоящий на пересечении строки с номером i и столбца с номером j.

В NumPy произведение матриц вычисляется с помощью функции numpy.dot(a, b, …) или с помощью метода array1.dot(array2), где array1 и array2 — перемножаемые матрицы.

```
a = np.array([[1, 0], [0, 1]])
b = np.array([[4, 1], [2, 2]])
r1 = np.dot(a, b)
r2 = a.dot(b)

print ("Матрица A:\n", a)
print ("Матрица B:\n", b)
print ("Результат умножения функцией:\n", r1)
print ("Результат умножения методом:\n", r2)
```

```
Матрица A:
 [[1 0]
 [0 1]]
Матрица B:
 [[4 1]
 [2 2]]
Результат умножения функцией:
 [[4 1]
 [2 2]]
Результат умножения методом:
 [[4 1]
 [2 2]]
```

Обратите внимание: операция ∗ производит над матрицами покоординатное умножение, а не матричное!

```
r = a * b
print ("Матрица A:\n", a)
print ("Матрица B:\n", b)
print ("Результат покоординатного умножения через операцию умножения:\n", r)
```

```
Матрица A:
 [[1 0]
 [0 1]]
Матрица B:
 [[4 1]
```

[2 2]]
Результат покоординатного умножения через операцию умножения:
[[4 0]
[0 2]]

Транспонирование матриц

Транспонированной матрицей A^T называется матрица, полученная из исходной матрицы A заменой строк на столбцы. Формально: элементы матрицы A^T определяются как $a_{ij}^{\mathrm{T}} = aji$, где a_{ij}^T — элемент матрицы A^T, стоящий на пересечении строки с номером i и столбца с номером j.

В NumPy транспонированная матрица вычисляется с помощью функции numpy.transpose() или с помощью метода array.T, где array — нужный двумерный массив.

```
a = np.array([[1, 2], [3, 4]])
b = np.transpose(a)
c = a.T

print ("Матрица:\n", a)
print ("Транспонирование функцией:\n", b)
print ("Транспонирование методом:\n",  c)
```

Матрица:
[[1 2]
[3 4]]
Транспонирование функцией:
[[1 3]
[2 4]]
Транспонирование методом:
[[1 3]
[2 4]]

Определитель матрицы

Для квадратных матриц существует понятие определителя. Пусть A — квадратная матрица. Определителем (или детерминантом) матрицы $A \in \mathbb{R}^{n\times n}$ назовем число $detA = \sum_{\alpha 1,\alpha 2,\ldots,\alpha n} (-1)^{N(\alpha 1,\alpha 2,\ldots,\alpha n)} \cdot a_{\alpha_{11}} \cdots a_{\alpha_n n}$,

где $\alpha_1, \alpha_2, \ldots, \alpha_n$ — перестановка чисел от 1 до n, $N(\alpha 1, \alpha 2, \ldots, \alpha n)$ — число инверсий в перестановке, суммирование ведется по всем возможным перестановкам длины n. Например, для матрицы размера 2×2 получается: $det\begin{vmatrix} a_{11}a_{12}a_{21}a_{22} \end{vmatrix} = a_{11}a_{22} - a_{12}a_{21}$.

Вычисление определителя матрицы по определению требует порядка $n!$ операций, поэтому разработаны методы, которые позволяют вычислять его быстро и эффективно.

В NumPy определитель матрицы вычисляется с помощью функции numpy.linalg.det(a), где a — исходная матрица.

```
a = np.array([[1, 2, 1], [1, 1, 4], [2, 3, 6]], dtype=np.float32)
det = np.linalg.det(a)

print ("Матрица:\n", a)
print ("Определитель:\n", det)
```

```
Матрица:
 [[1. 2. 1.]
 [1. 1. 4.]
 [2. 3. 6.]]
Определитель:
 -1.0
```

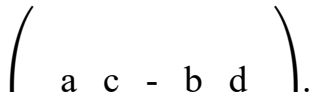
```
a = np.array([[1, 2], [3, 4]] )
det = np.linalg.det(a)
det
```

```
-2.0000000000000004
```

```
a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]], dtype=np.float32)
det = np.linalg.det(a)
det
```

```
6.661338e-16
```

Рассмотрим одно интересное свойство определителя. Пусть у нас есть параллелограмм с углами в точках (0, 0), (c, d), (a + c, b + d), (a, b) (углы даны в порядке обхода по часовой стрелке). Тогда площадь этого параллелограмма можно вычислить как модуль определителя матрицы

 . Похожим образом можно выразить и объем параллелепипеда через определитель матрицы размера 3×3.

Ранг матрицы

Напоминание теории. Рангом матрицы A называется максимальное число линейно независимых строк (столбцов) этой матрицы.

В NumPy ранг матрицы вычисляется с помощью функции numpy.linalg.matrix_rank(M, tol=None), где M — матрица, tol — параметр, отвечающий за некоторую точность вычисления. В простом случае можно его не задавать, и функция сама определит подходящее значение этого параметра.

```
a = np.array([[1, 2, 3], [1, 1, 1], [2, 2, 2]])
r = np.linalg.matrix_rank(a)

print ("Матрица:\n", a)
print ("Ранг матрицы:", r)
```

```
Матрица:
 [[1 2 3]
 [1 1 1]
 [2 2 2]]
Ранг матрицы: 2
```

С помощью вычисления ранга матрицы можно проверять линейную независимость системы векторов. Допустим, у нас есть несколько векторов. Составим из них матрицу, где наши векторы будут являться строками. Понятно, что векторы линейно независимы тогда и только тогда, когда ранг полученной матрицы совпадает с числом векторов. Приведем пример:

```
a = np.array([1, 2, 3])
b = np.array([1, 1, 1])
c = np.array([2, 3, 5])
m = np.array([a, b, c])
m
```

```
array([[1, 2, 3],
       [1, 1, 1],
       [2, 3, 5]])
```

```
print (np.linalg.matrix_rank(m) == m.shape[0])
```

True

```
np.linalg.matrix_rank(m)
```

3

```
m.shape[0]
```

3

Системы линейных уравнений

Системой линейных алгебраических уравнений называется система вида $Ax = b$, где $A \in \mathbb{R}^{n \times m}, x \in \mathbb{R}^{m \times 1}, b \in \mathbb{R}^{n \times 1}$. В случае квадратной невырожденной матрицы A решение системы единственно. В NumPy решение такой системы можно найти с помощью функции numpy.linalg.solve(a, b), где первый аргумент — матрица A, второй — столбец b.

```
a = np.array([[3, 1], [1, 2]]) # матрица A
b = np.array([9, 8]) # Матрица B
x = np.linalg.solve(a, b) # решение уравнения AX=B

print ("Матрица A:\n", a)
print ("Вектор b:\n", b)
print ("Решение системы:\n", x)
```

```
Матрица A:
 [[3 1]
 [1 2]]
Вектор b:
 [9 8]
Решение системы:
 [2. 3.]
```

Убедимся, что вектор x действительно является решением системы:

```
print (a.dot(x)) # проверка
```

```
[9. 8.]
```

Бывают случаи, когда решение системы не существует. Но хотелось бы все равно “решить” такую систему. Логичным кажется искать такой вектор x, который минимизирует выражение $\|Ax - b\|^2$ — так мы приблизим выражение Ax к b. В NumPy такое псевдорешение можно искать с помощью функции numpy.linalg.lstsq(a, b, …), где первые два аргумента такие же, как и для функции numpy.linalg.solve(). Помимо решения функция возвращает еще три значения, которые нам сейчас не понадобятся.

```
a = np.array([[0, 1], [1, 1], [2, 1], [3, 1]])
b = np.array([-1, 0.2, 0.9, 2.1])
x, res, r, s = np.linalg.lstsq(a, b, rcond=None)

print ("Матрица A:\n", a)
print ("Вектор b:\n", b)
print ("Псевдорешение системы:\n", x)
```

```
Матрица A:
 [[0 1]
 [1 1]
 [2 1]
 [3 1]]
Вектор b:
 [-1.   0.2  0.9  2.1]
Псевдорешение системы:
 [ 1.   -0.95]
```

Обратная матрица

Для квадратных невырожденных матриц определено понятие обратной матрицы. ПустьA — квадратная невырожденная матрица. Матрица A^{-1} называется обратной матрицей к A, если $AA^{-1} = A^{-1}A = I$,

где I — единичная матрица.

В NumPy обратные матрицы вычисляются с помощью функции numpy.linalg.inv(a), где a — исходная матрица.

```
a = np.array([[1, 2, 1], [1, 1, 4], [2, 3, 6]], dtype=np.float32)
b = np.linalg.inv(a)
```

```
print ("Матрица A:\n", a)
print ("Обратная матрица к A:\n", b)
print('Проверка:')
print ("Произведение A на обратную должна быть единичной:\n", a.dot(b))
```

```
Матрица A:
 [[1. 2. 1.]
 [1. 1. 4.]
 [2. 3. 6.]]
Обратная матрица к A:
 [[ 6.  9. -7.]
 [-2. -4.  3.]
 [-1. -1.  1.]]
Проверка:
Произведение A на обратную должна быть единичной:
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

Собственные числа и собственные вектора матрицы

Для квадратных матриц определены понятия собственного вектора и собственного числа. Пусть A — квадратная матрица и $A \in \mathbb{R}^{n \times n}$. Собственным вектором матрицы A называется такой ненулевой вектор $x \in \mathbb{R}^n$, что для некоторого $\lambda \in \mathbb{R}$ выполняется равенство $Ax = \lambda x$. При этом λ называется собственным числом матрицы A. Собственные числа и собственные векторы матрицы играют важную роль в теории линейной алгебры и ее практических приложениях.

В NumPy собственные числа и собственные векторы матрицы вычисляются с помощью функции numpy.linalg.eig(a), где a — исходная матрица. В качестве результата эта функция выдает одномерный массив w собственных чисел и двумерный массив v, в котором по столбцам записаны собственные вектора, так что вектор v[:, i] соотвествует собственному числу w[i].

```
a = np.array([[-1, -6], [2, 6]])
w, v = np.linalg.eig(a)
```

```
a = np.array([[1, 2], [3, 4]])
w, v = np.linalg.eig(a)
```

Расстояния между векторами

p-норма

p-норма (норма Гёльдера) для вектора $x = (x_1, \dots, x_n) \in \mathbb{R}^n$ вычисляется по формуле:

$\|x\|_p = (\sum_{i=1}^{n} |x_i|^p)^{1/p}, p \geq 1$. В частных случаях при: $p = 1$ получаем ℓ_1 норму, $p = 2$ получаем ℓ_2 норму.

Далее нам понабится модуль numpy.linalg, реализующий некоторые приложения линейной алгебры. Для вычисления различных норм мы используем функцию numpy.linalg.norm(x, ord=None, …), где x — исходный вектор, ord — параметр, определяющий норму (мы рассмотрим два варианта его значений — 1 и 2). Импортируем эту функцию:

```
from numpy.linalg import norm
```

l_1 норма

ℓ_1 норма (также известная как манхэттенское расстояние) для вектора $x = (x_1, \dots, x_n) \in \mathbb{R}^n$ вычисляется по формуле:

$\|x\|_1 = \sum_{i=1}^{n} |x_i|$. Ей в функции numpy.linalg.norm(x, ord=None, …) соответствует параметр ord=1.

```
a = np.array([1, 2, -3])
print('Вектор a:', a)
```

```
Вектор a: [ 1  2 -3]
```

```
print('L1 норма вектора a:\n', norm(a, ord=1))
```

```
L1 норма вектора a:
 6.0
```

ℓ_2 норма

ℓ_2 норма (также известная как евклидова норма) для вектора $x = (x_1, \dots, x_n) \in \mathbb{R}^n$ вычисляется по формуле:

$$\|x\|_2 = \sqrt{\sum_{i=1}^{n} (x_i)^2}.$$

Ей в функции numpy.linalg.norm(x, ord=None, …) соответствует параметр ord=2.

```
print('L2 норма вектора a:\n', norm(a, ord=2))
```

```
L2 норма вектора a:
 3.7416573867739413
```

Расстояния между векторами

Для двух векторов $x = (x_1, \dots, x_n) \in \mathbb{R}^n$ и $y = (y_1, \dots, y_n) \in \mathbb{R}^n$ ℓ_1 и ℓ_2 расстояния вычисляются по следующим формулам соответственно:

$$\rho_1(x, y) = \|x - y\|_1 = \sum_{i=1}^{n} |x_i - y_i|$$

$$\rho_2(x, y) = \|x - y\|_2 = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}.$$

```
a = np.array([1, 2, -3])
b = np.array([-4, 3, 8])
print ('Вектор a:', a)
print ('Вектор b:', b)
```

```
Вектор a: [ 1  2 -3]
Вектор b: [-4  3  8]
```

```
print ('L1 расстояние между векторами a и b:\n', norm(a - b, ord = 1))
print ('L2 расстояние между векторами a и b:\n', norm(a - b, ord = 2))
```

```
L1 расстояние между векторами a и b:
 17.0
L2 расстояние между векторами a и b:
 12.12435565298214
```

Скалярное произведение и угол между векторами

Скалярное произведение в пространстве $\mathbb{R}^n$ для двух векторов $x = (x_1, \dots, x_n)$ и $y = (y_1, \dots, y_n)$ определяется как:

$\langle x, y \rangle = \sum_{i=1}^{n} x_i\, y_i$. Длиной вектора $x = (x_1, \dots, x_n) \in \mathbb{R}^n$ называется квадратный корень из скалярного произведения, то есть длина равна евклидовой норме вектора:

$|x| = \sqrt{(\langle x, x \rangle)} = \sqrt{\left(\sum_{i=1}^{n} x_i^2\right)} = \|x\|_2$. Теперь, когда мы знаем расстояние между двумя ненулевыми векторами и их длины, мы можем вычислить угол между ними через скалярное произведение:

$\langle x, y \rangle = |x||y| cos(\alpha) \Longrightarrow cos(\alpha) = (\langle x, y \rangle)/(|x||y|)$, где $\alpha \in [0, \pi]$ — угол между векторами x и y.

```
a = np.array([0, 5, -1])
b = np.array([-4, 9, 3])
print ('Вектор a:', a)
print ('Вектор b:', b)
```

```
Вектор a: [ 0  5 -1]
Вектор b: [-4  9  3]
```

```
cos_angle = np.dot(a, b) / norm(a) / norm(b)
print ('Косинус угла между a и b:', cos_angle)
print ('Сам угол:', np.arccos(cos_angle))
```

```
Косинус угла между a и b: 0.8000362836474323
Сам угол: 0.6434406336093618
```

Комплексные числа

Комплексными числами называются числа вида $x + iy$, где x и y — вещественные числа, а i — мнимая единица (величина, для которой выполняется равенство $i^2 = -1$). Множество всех комплексных чисел обозначается буквой $\mathbb{C}$. В питоне комплексные числа можно задать следующим образом (j обозначает мнимую единицу):

```
a = 3 + 2j
b = 1j
```

```
print ("Комплексное число a:\n", a)
print ("Комплексное число b:\n", b)
```

```
Комплексное число a:
 (3+2j)
Комплексное число b:
 1j
```

С комплексными числами в питоне можно производить базовые арифметические операции так же, как и с вещественными числами:

```
c = a * a
d = a / (4 - 5j)

print ("Комплексное число c:\n", c)
print ("Комплексное число d:\n", d)
```

```
Комплексное число c:
 (5+12j)
Комплексное число d:
 (0.0487804878048781+0.5609756097560976j)
```

Задания

1) Выполнить все примеры выше в среде Google Colab.
2) Вычислить определитель матрицы. Номер варианта соответствует порядковому номеру в списке. Данные можно скачать по ссылке: https://drive.google.com/file/d/1uXReH3kAcOUxqGTDPZzWlIv3hqZkMxov/view?usp=sharing

или по QR коду:

3) Выполнить действия над матрицами. Номер варианта соответствует порядковому номеру в списке. Данные можно скачать по ссылке: https://drive.google.com/file/d/1zxAhtPeO8d2Hhd0rB3u9JQbtHRSj-T-U/view?usp=sharing

или по QR коду:

4) Найти ранги матриц A и B (матрицы брать из задания №3).

5) Найти матрицу обратную данной и выполнить проверку умножением. Номер варианта соответствует порядковому номеру в списке. Данные можно скачать по ссылке: https://drive.google.com/file/d/1udrfIBj6jtWJlxSRAq2J_-fPmByzOig8/view?usp=sharing

или по QR коду:

6) Система линейных уравнений задана в матричном виде AX=B . Решить ситему уравнений двумя разными способами и выполнить проверку. Номер варианта соответствует порядковому номеру в списке. Данные можно скачать по ссылке: https://drive.google.com/file/d/1G5U0qtr3a7QECy54Su3pRfx7434x8PCd/view?usp=sharing

или по QR коду:

7) Выполнить действия над векторами. Номер варианта соответствует порядковому номеру в списке. Данные можно скачать по ссылке: https://drive.google.com/file/d/1AvSqXo3wfJHYitZmGgNWge2djRN0YuFf/view?usp=sharing

или по QR коду:

8) Проверить лежат ли в одной плоскости четыре точки A,B,C,D. Номер варианта соответствует порядковому номеру в списке. Данные можно скачать по ссылке: https://drive.google.com/file/d/19zIRf_TLX9TIL-yCkWss78IVV8shFfOA/view?usp=sharing

или по QR коду:

9) Даны вершины тетрайдера A,B,C,D . Найти объем тетрайдера. Номер варианта соответствует порядковому номеру в списке. Данные можно скачать по ссылке: https://drive.google.com/file/d/1_jD8dAX3DwNTi-4baLmTHKSTxJUraSP0/view?usp=sharing

или по QR коду:

10) Вычислить собственные векторы и собственные значения матрицы A (матрицу взять из задания №2)

Полноценную версию блокнота Google Colab в pdf формате можно посмотреть по QR коду:

Библиотека SciPy. Вычисление интегралов.

SciPy (сокращение от Scientific Python, или научный Python) – библиотека, которая является расширением библиотеки NumPy и предназначена для выполнения сложных инженерных, статистических и научных расчетов, а так же для анализа данных и построения графиков.

Чем SciPy отличается от NumPy?

Первое отличие – по типу выполняемых операций. NumPy выполняет стандартные операции, такие как сортировка, индексирование, а так же элементарные операции, связанные с типом данных массива. SciPy используется для выполнения сложных операций, например, расчет алгебраических функций или различных числовых алгоритмов.

Второе отличие – методы и функции, которые содержатся в библиотеках. NumPy содержит немалое количество функций и методов для решения задач линейной алгебры, а также расчета преобразования Фурье и обработки сигналов и изображений, однако они неполноценные. В SciPy, наоборот, эти функции и методы реализованы в полном объеме, и их гораздо больше.

Третье отличие – концепция массивов. Массивы из библиотеки NumPy — это многомерные массивы объектов, которые имеют одинаковый тип, то есть они однородны или гомогенны. В SciPy концепция массивов немного другая, поскольку она более функциональна, и не имеет ограничений на однородность. То есть массивы SciPy могут быть как гомогенными, так и гетерогенными.

Четвертое отличие – язык, написания библиотек и скорость. Библиотека NumPy написана на языке C и как следствие имеет более высокую скорость вычислений, однако за нее приходится платить функциональностью библиотеки. Библиотека SciPy написана на языке Python и поэтому имеет более низкую скорость выполнения, однако низкая скорость компенсируется предоставляемой функциональностью.

Вычисление интегралов в SciPy

Integrate.

Данный пакет, позволяет проводить численное интегрирование: рассчитывать определенные интегралы, решать обыкновенные дифференциальные уравнения и пр.

Доступные методы:

quad(func, a, b[, args, full_output, ...])	Вычислите определенный интеграл.
quad_vec(f, a, b[, эпсабс, эпсрел, норма, ...])	Адаптивное интегрирование векторной функции.
dblquad(func, a, b, gfun, hfun[, args, ...])	Вычислите двойной интеграл.
tplquad(func, a, b, gfun, hfun, qfun, rfun)	Вычислите тройной (определенный) интеграл.
nquad(func, диапазоны[, args, opts, full_output])	Интегрирование по нескольким переменным.
fixed_quad(func, a, b[, args, n])	Вычислите определенный интеграл, используя гауссову квадратуру фиксированного порядка.
quadrature(func, a, b[, args, tol, rtol, ...])	Вычислите определенный интеграл, используя квадратуру Гаусса с фиксированным допуском.
romberg(функция, a, b[, args, tol, rtol, ...])	Интеграция Ромберга вызываемой функции или метода.
newton_cotes(rn[, равно])	Возвращаемые веса и коэффициент ошибки для интегрирования Ньютона-Котеса.
qmc_quad(func, a, b, *[, n_estimates, ...])	Вычислите интеграл в N-мерностях, используя квадратуру Квази-Монте-Карло.

Определенный интеграл scipy.integrate.quad

scipy.integrate. quad (func , a , b , args = () , full_output = 0 , epsabs = 1.49e-08 , epsrel = 1.49e-08 , предел = 50 , точки = нет , вес = нет , wvar = нет , wopts = нет , maxp1 = 50 , limlst = 50 , complex_func = False)

Интегрирует функцию от a до b (возможно, с бесконечным интервалом), используя технику из библиотеки Фортрана QUADPACK.

Параметры:

- func Функция или метод Python для интеграции. Если функция func принимает много аргументов, она интегрируется по оси, соответствующей первому аргументу.
- a:float Нижний предел интегрирования.

- b:float Верхний предел интегрирования.
- args: tuple, optional Дополнительные аргументы для передачи в func
- full_output: int, optional Ненулевое значение, чтобы вернуть словарь интеграционной информации. Если значение не равно нулю, предупреждающие сообщения также подавляются, и сообщение добавляется к выходному кортежу.
- complex_func: bool, optional Укажите, является ли тип возвращаемого значения функции (func) вещественным (complex_func=False: default) или сложным (complex_func=True). В обоих случаях аргумент функции действительный.

Возвращает: y: float Интеграл функции от a до b .

abserr: float Оценка абсолютной ошибки результата.

infodict: dict Словарь, содержащий дополнительную информацию.

Примеры:

Вычислить интеграл $\int_0^4 x^2\,dx$

```
from scipy import integrate
import numpy as np
f = lambda x: x**2
integrate.quad(f, 0, 4)
```

(21.333333333333336, 2.368475785867001e-13)

Первое значение в кортеже – значение интеграла, второе – верхняя оценка погрешности вычисленного значения интеграла

Вычислить $\int_0^\infty e^{-x}\,dx$

```
invexp = lambda x: np.exp(-x)
integrate.quad(invexp, 0, np.inf)
```

(1.0000000000000002, 5.842606703608969e-11)

Вычислить $\int_0^1 \alpha\, xdx$ для праметра $\alpha = 1{,}3$

```
f = lambda x, a: a*x
y, err = integrate.quad(f, 0, 1, args=(1,))
print(y)
y, err = integrate.quad(f, 0, 1, args=(3,))
print(y)
```

0.5
1.5

Вычисление двойного интеграла scipy.integrate.dblquad

scipy.integrate. dblquad (func , a , b , gfun , hfun , args = () , epsabs = 1.49e-08 , epsrel = 1.49e-08)

Параметры: * func: callable Функция или метод Python как минимум с двумя переменными: y должен быть первым аргументом, а x — вторым аргументом.

- a, b: float Пределы интегрирования по x: $a < b$
- gfun: callable or float Нижняя граничная кривая в y, которая представляет собой функцию, принимающую один аргумент с плавающей запятой (x) и возвращающую результат с плавающей запятой или число с плавающей запятой, указывающее постоянную граничную кривую.
- hfun: callable or float Верхняя граничная кривая по y (те же требования, что и у gfun).
- args: sequence, optional Дополнительные аргументы для передачи в func ..
- epsabs: float, optional Абсолютный допуск. По умолчанию — 1.49e-8.

epsrel: float, optional Относительный допуск внутренних одномерных интегралов. По умолчанию — 1.49e-8.

Возвращает: * y: float Результирующий интеграл.

- abserr: float Оценка ошибки.

Примеры:

Вычислить интеграл $\int_{x=0}^{x=2}\int_{y=0}^{y=1} x\,y^2 dydx$

```
f = lambda y, x: x*y**2
integrate.dblquad(f, 0, 2, 0, 1)
```

(0.6666666666666667, 7.401486830834377e-15)

Вычислить $\int_{x=0}^{x=\pi/4}\int_{y=\sin(x)}^{y=\cos(x)} 1\,dydx$

```
f = lambda y, x: 1
integrate.dblquad(f, 0, np.pi/4, np.sin, np.cos)
```

(0.41421356237309503, 1.1083280054755938e-14)

Вычислить $\int_{x=0}^{x=1}\int_{y=x}^{y=2-x} \pi xydydx$ для $\alpha = 1{,}3$

```
f = lambda y, x, a: a*x*y
integrate.dblquad(f, 0, 1, lambda x: x, lambda x: 2-x, args=(1,))
```

(0.33333333333333337, 5.551115123125783e-15)

```
integrate.dblquad(f, 0, 1, lambda x: x, lambda x: 2-x, args=(3,))
```

(1.0, 1.6653345369377348e-14)

Вычислите двумерный интеграл Гаусса $\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-(x^2+y^2)}\,dydx$

```
f = lambda x, y: np.exp(-(x ** 2 + y ** 2))
integrate.dblquad(f, -np.inf, np.inf, -np.inf, np.inf)
```

(3.141592653589777, 2.5173086737434902e-08)

Вычисление тройного интеграла scipy.integrate.tplquad

scipy.integrate. tplquad (func , a , b , gfun , hfun , qfun , rfun , args = () , epsabs = 1.49e-08 , epsrel = 1.49e-08)

Параметры: * funcfunction Функция или метод Python как минимум с тремя переменными в порядке (z, y, x).

- a, b: float Пределы интегрирования по x: a < b
- gfun: function or float Нижняя граничная кривая в y, которая представляет собой функцию, принимающую один аргумент с плавающей запятой (x) и возвращающую результат с плавающей запятой или число с плавающей запятой, указывающее постоянную граничную кривую.
- hfun: function or float Верхняя граничная кривая по y (те же требования, что и у gfun).
- qfun: function or float Нижняя граничная поверхность в z. Это должна быть функция, которая принимает два числа с плавающей запятой в порядке (x, y) и возвращает число с плавающей запятой или число с плавающей запятой, указывающее постоянную граничную поверхность.
- rfun: function or float Верхняя граничная поверхность в z. (Те же требования, что и у qfun .)
- args: tuple, optional Дополнительные аргументы для передачи в func .
- epsabs: float, optional Абсолютный допуск перешел непосредственно к самому внутреннему одномерному квадратурному интегрированию. По умолчанию — 1.49e-8.
- epsrel: float, optional Относительная толерантность к самым внутренним одномерным интегралам. По умолчанию — 1.49e-8.

Возвращает: * y: float Результирующий интеграл.

- abserr: float Оценка ошибки.

Для получения достоверных результатов интеграл должен сходиться; поведение расходящихся интегралов не гарантируется.

Примеры

Вычислить тройной интеграл $\int_{x=1}^{x=2}\int_{y=2}^{y=3}\int_{z=0}^{z=1} xyzdzdydx$

```
f = lambda z, y, x: x*y*z
integrate.tplquad(f, 1, 2, 2, 3, 0, 1)
```

(1.8750000000000002, 3.324644794257407e-14)

Вычислить тройной интеграл $\int_{x=0}^{x=1}\int_{y=0}^{y=1-2x}\int_{z=0}^{z=1-x-2y}$ xyzdzdydx

```
f = lambda z, y, x: x*y*z
integrate.tplquad(f, 0, 1, 0, lambda x: 1-2*x, 0, lambda x, y: 1-x-2*y)
```

(0.05416666666666667, 2.1774196738157757e-14)

Вычислить $\int_{x=0}^{x=1}\int_{y=0}^{y=1}\int_{z=0}^{z=1}\alpha\, xyzdzdydx$ для $\alpha = 1{,}3$

```
f = lambda z, y, x, a: a*x*y*z
integrate.tplquad(f, 0, 1, 0, 1, 0, 1, args=(1,))
```

(0.12499999999999999, 5.527033708952211e-15)

```
integrate.tplquad(f, 0, 1, 0, 1, 0, 1, args=(3,))
```

(0.375, 1.6581101126856638e-14)

Вычислите трехмерный интеграл Гаусса $\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-(x^2+y^2+z^2)}\, dzdydx$

```
f = lambda x, y, z: np.exp(-(x ** 2 + y ** 2 + z ** 2))
integrate.tplquad(f, -np.inf, np.inf, -np.inf, np.inf, -np.inf, np.inf)
```

(5.568327996830833, 4.4619079938249415e-08)

Задания

1) Выполнить все примеры выше в среде Google Colab.
2) Вычислить определенные интегралы. Номер варианта соответствует порядковому номеру в списке. Данные можно скачать по ссылке: https://drive.google.com/file/d/1aQVEIfT6CxFo1aQLaxnEpFBm-JDC1sgS/view?usp=sharing

или по QR коду:

3) Вычислить несобственные интегралы. Номер варианта соответствует порядковому номеру в списке. Данные можно скачать по ссылке: https://drive.google.com/file/d/1LtQuuhqFcETZZNnDLbA5OiQ2VVbIHuYG/view?usp=sharing

или по QR коду:

4) Вычислить двойные интегралы. Номер варианта соответствует порядковому номеру в списке. Данные можно скачать по ссылке: https://drive.google.com/file/d/1FAE9ZlGQyBApg9ndRBNzA4tUJsI8ftXN/view?usp=sharing

или по QR коду:

5) Вычислить тройные интегралы. Номер варианта соответствует порядковому номеру в списке. Данные можно скачать по ссылке:

https://drive.google.com/file/d/1BFgOfY0E3fJ68C-NsIYb6X50LSx2wamT/view?usp=sharing

или по QR коду:

Полноценную версию блокнота Google Colab в pdf формате можно посмотреть по QR коду:

Построение графиков с помощью библиотеки Matplotlib

Matplotlib — мультиплатформенная библиотека для визуализации данных, основанная на массивах библиотеки NumPy и спроектированная в расчете на работу с обширным стеком SciPy. Она была задумана Джоном Хантером в 2002 году и изначально представляла собой патч к оболочке IPython, предназначенный для реализации возможности интерактивного построения с помощью утилиты gnuplot графиков в стиле MATLAB из командной строки IPython. Создатель оболочки IPython Фернандо Перес в этот момент был занят завершением написания диссертации, он сообщил Джону, что в ближайшие несколько месяцев у него не будет времени на анализ патча. Хантер принял это как благословение на самостоятельную разработку — так родился пакет Matplotlib, версия 0.1 которого была выпущена в 2003 году. Институт исследований космоса с помощью космического телескопа (Space Telescope Science Institute, занимающийся управлением телескопом «Хаббл») финансово поддержал разработку пакета Matplotlib и обеспечил расширение его возможностей, избрав в качестве пакета для формирования графических изображений.

Простые линейные графики

Вероятно, простейшим из всех графиков является визуализация отдельной функции y = f(x). В этом разделе мы рассмотрим создание простого графика такого типа. Как и во всех следующих разделах, начнем с настройки блокнота для построения графиков и импорта функций, которые будем использовать:

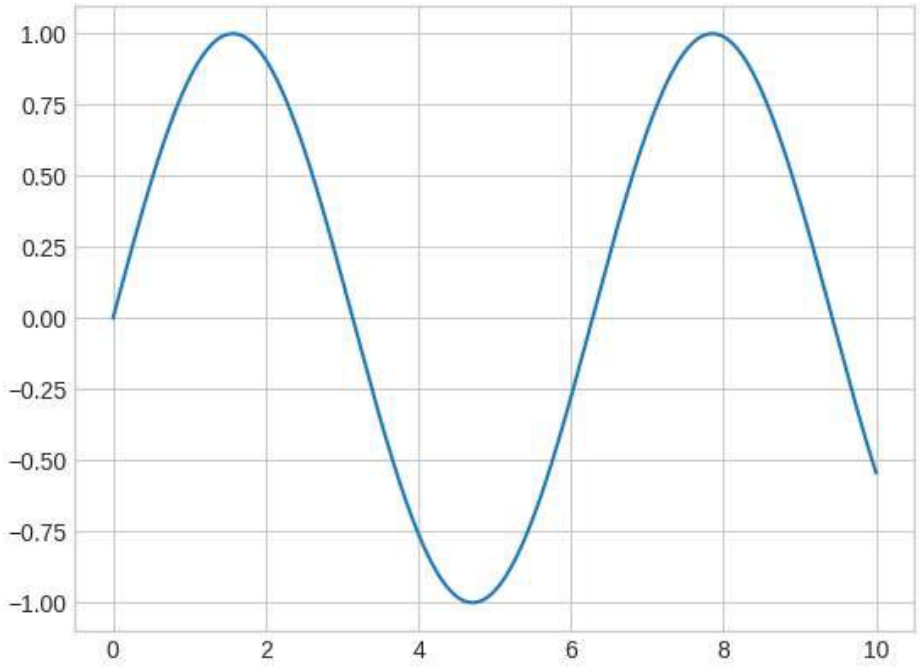
```
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import numpy as np
```

```
plt.style.use('seaborn-whitegrid')
```

Начнем с простой синусоиды

```
fig = plt.figure()
ax = plt.axes()
```

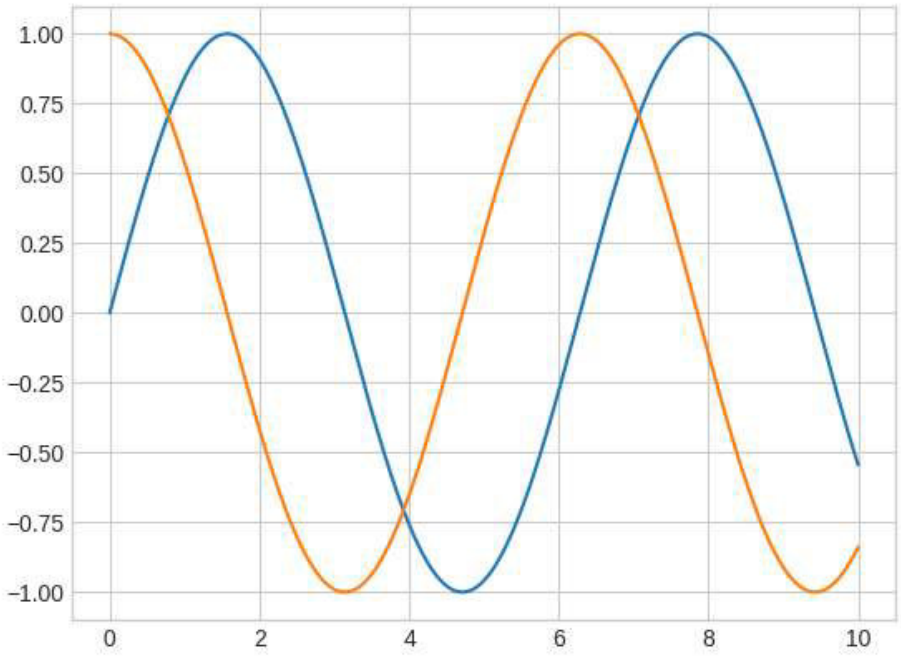
```
x = np.linspace(0, 10, 1000)
ax.plot(x, np.sin(x));
```



Если нужно создать простой рисунок с несколькими линиями, можно вызвать функцию plot несколько раз

```
plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))
```

```
[<matplotlib.lines.Line2D at 0x7855ac5e8ee0>]
```

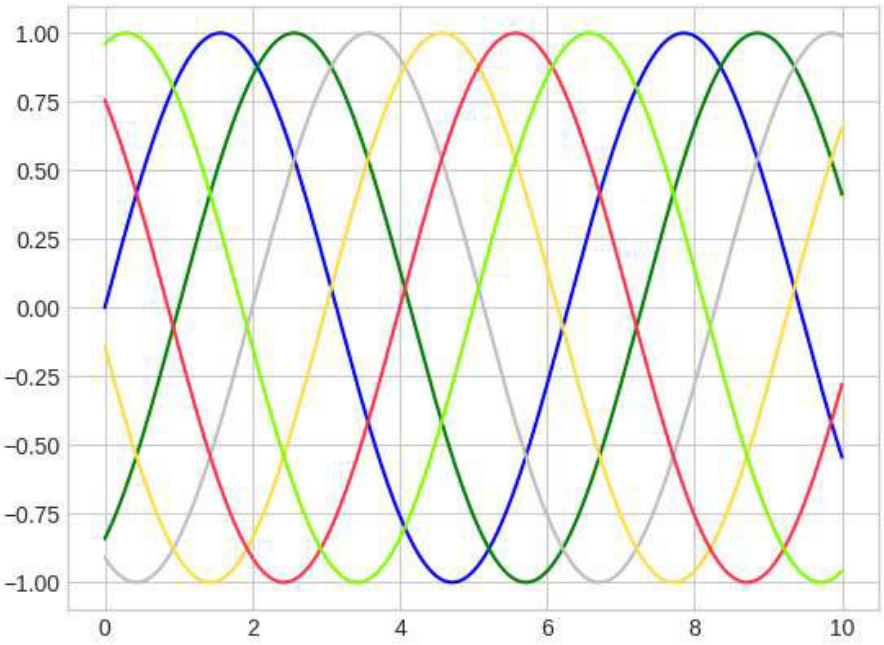


Настройка графика: цвета и стили линий

Первое, что вы можете захотеть сделать с графиком, — научиться управлять цветами и стилями линий. Функция plt.plot принимает дополнительные аргументы, которыми можно воспользоваться для этой цели. Для настройки цвета используйте ключевое слово color, которому ставится в соответствие строковый аргумент, задающий практически любой цвет. Задать цвет можно разными способами

```
plt.plot(x, np.sin(x - 0), color='blue')
plt.plot(x, np.sin(x - 1), color='g')
plt.plot(x, np.sin(x - 2), color='0.75')
plt.plot(x, np.sin(x - 3), color='#FFDD44')
plt.plot(x, np.sin(x - 4), color=(1.0,0.2,0.3))
plt.plot(x, np.sin(x - 5), color='chartreuse')
```

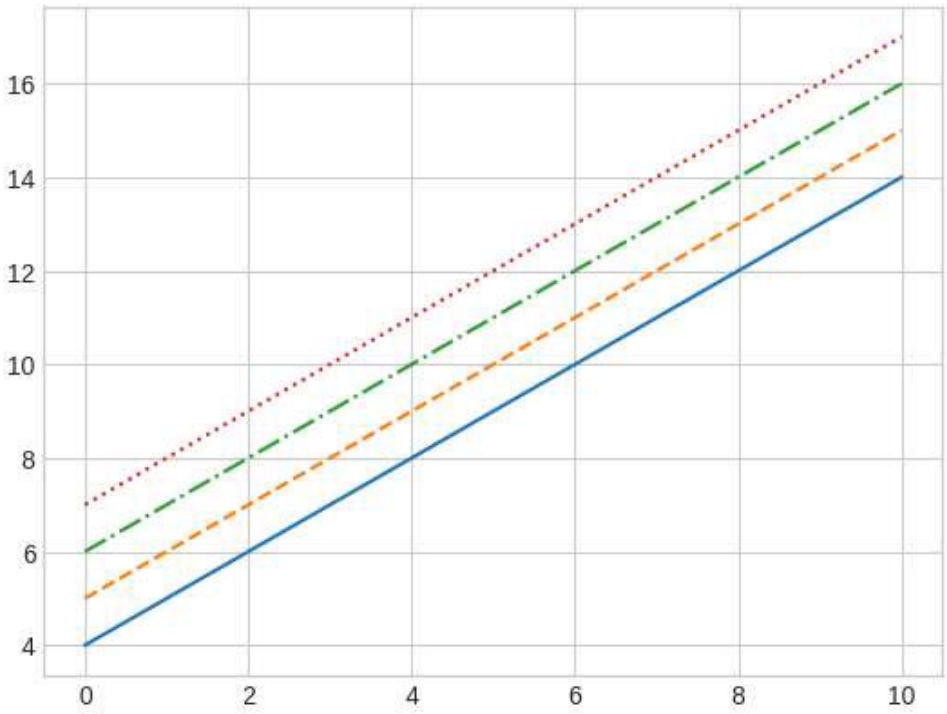
```
[<matplotlib.lines.Line2D at 0x7855ac1796c0>]
```



Если цвет не задан, библиотека Matplotlib будет автоматически перебирать по циклу набор цветов по умолчанию при наличии на графике нескольких линий. Стиль линий можно настраивать и с помощью ключевого слова linestyle

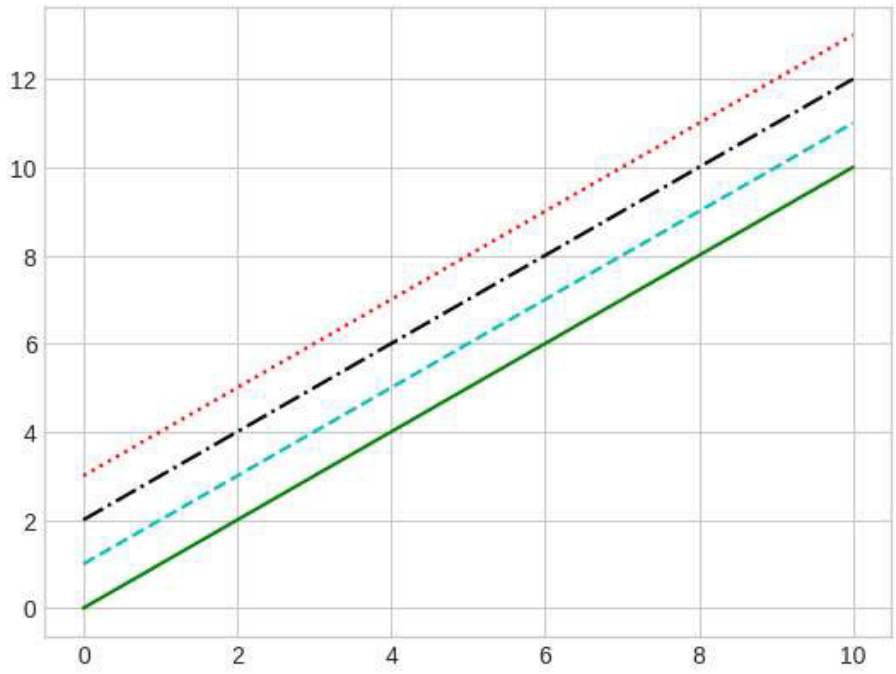
```
plt.plot(x, x + 4, linestyle='-')  # solid
plt.plot(x, x + 5, linestyle='--') # dashed
```

```
plt.plot(x, x + 6, linestyle='-.') # dashdot
plt.plot(x, x + 7, linestyle=':');  # dotted
```



Если вы предпочитаете максимально сжатый синтаксис, можно объединить задание кодов linestyle и color в одном неключевом аргументе функции plt.plot

```
plt.plot(x, x + 0, '-g')  # solid green
plt.plot(x, x + 1, '--c') # dashed cyan
plt.plot(x, x + 2, '-.k') # dashdot black
plt.plot(x, x + 3, ':r');  # dotted red
```

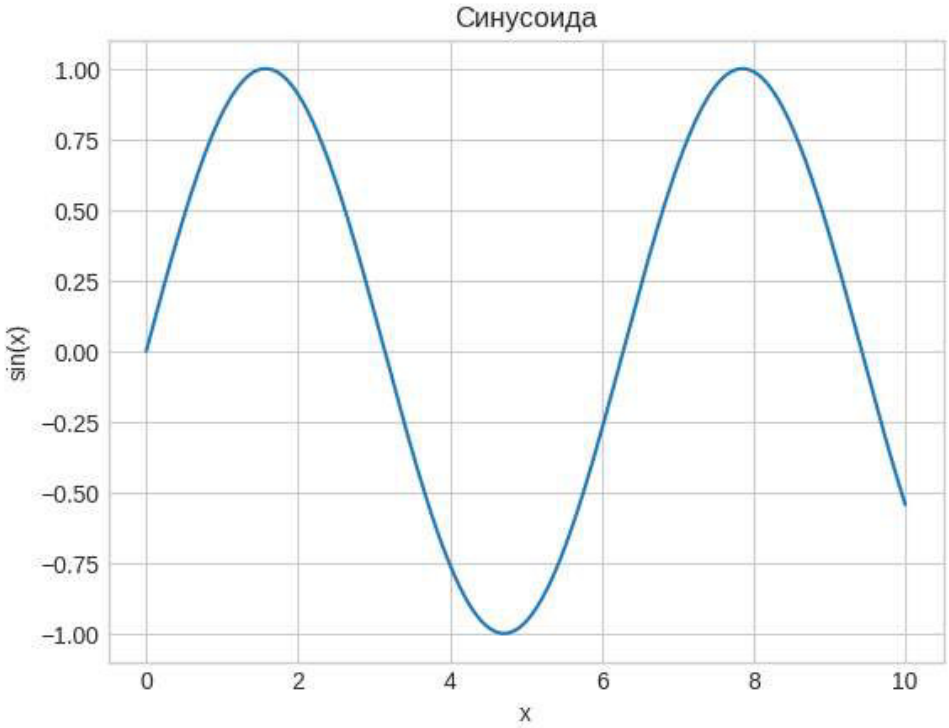


Метки на графиках

Рассмотрим маркирование графиков: названия, метки осей координат и простые легенды. Названия и метки осей — простейшие из подобных меток. Существуют методы, позволяющие быстро задать их значения

```
plt.plot(x, np.sin(x))
plt.title("Синусоида")
plt.xlabel("x")
plt.ylabel("sin(x)")
```

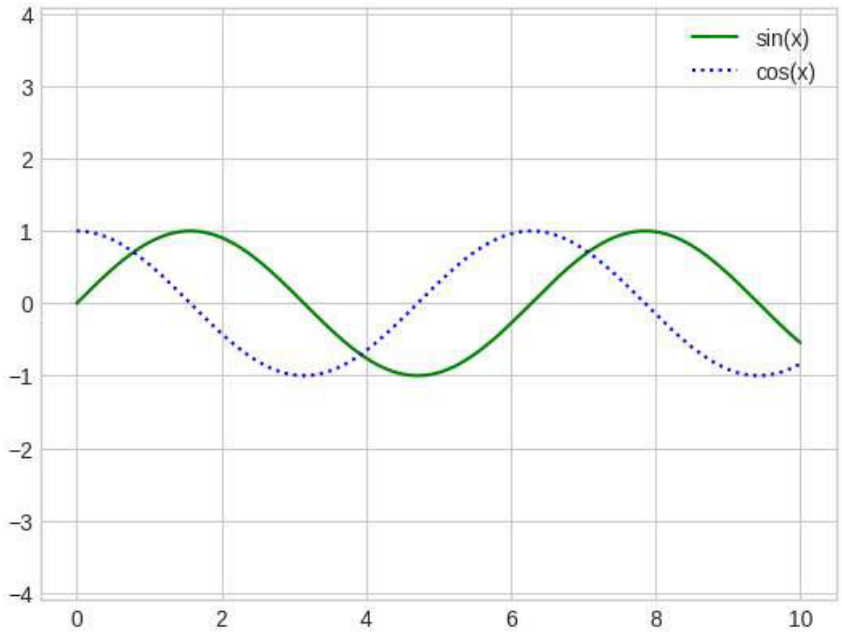
```
Text(0, 0.5, 'sin(x)')
```



В случае отображения нескольких линий в одной координатной сетке удобно создать легенду для графика, на которой бы отмечался каждый тип линии. В библиотеке Matplotlib для быстрого создания такой легенды имеется встроенный метод plt.legend(). Хотя существует несколько возможных способов, проще всего, как мне кажется, задать метку каждой линии с помощью ключевого слова label функции plot

```
plt.plot(x, np.sin(x), '-g', label='sin(x)')
plt.plot(x, np.cos(x), ':b', label='cos(x)')
plt.axis('equal')

plt.legend()
```

```
<matplotlib.legend.Legend at 0x7855ac1c6170>
```



Простые диаграммы рассеяния

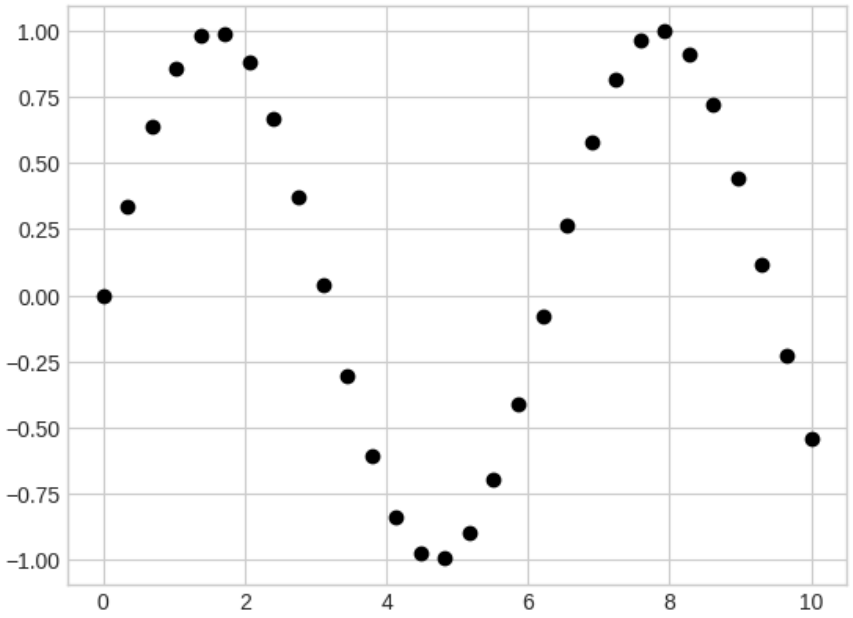
Еще один часто используемый тип графиков — диаграммы рассеяния, родственные линейным графикам. В них точки не соединяются отрезками линий, а представлены по отдельности точками, кругами или другими фигурами на графике.

Построение диаграмм рассеяния с помощью функции plt.plot

В предыдущем разделе мы рассмотрели построение линейных графиков посредством функции plt.plot/ax.plot. С ее помощью можно строить и диаграммы рассеяния

```
x = np.linspace(0, 10, 30)
y = np.sin(x)

plt.plot(x, y, 'o', color='black')
```

```
[<matplotlib.lines.Line2D at 0x7855907c9990>]
```

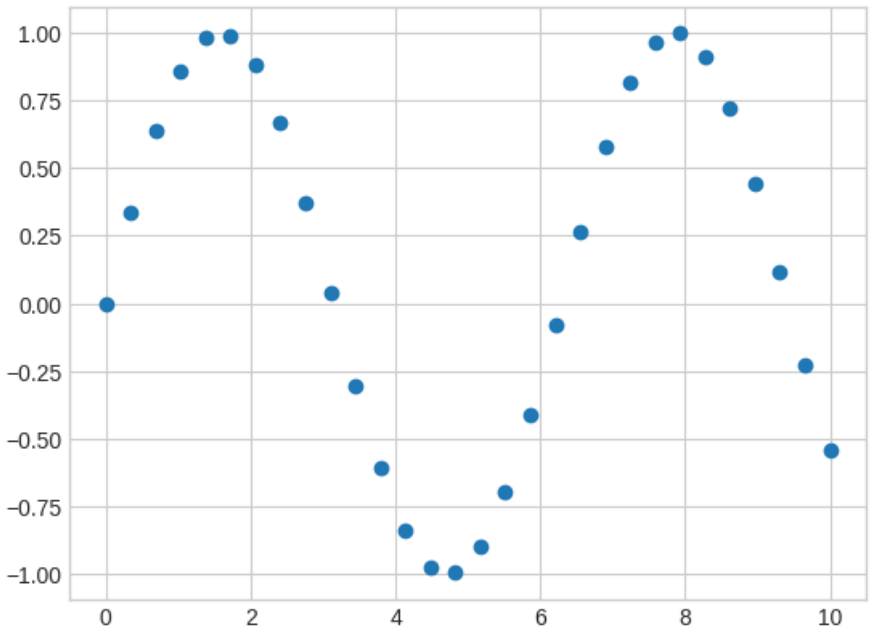


Построение диаграмм рассеяния с помощью функции plt.scatter

Еще большими возможностями обладает метод построения диаграмм рассеяния с помощью функции plt.scatter, во многом напоминающей функцию plt.plot

```
plt.scatter(x, y, marker='o')
```

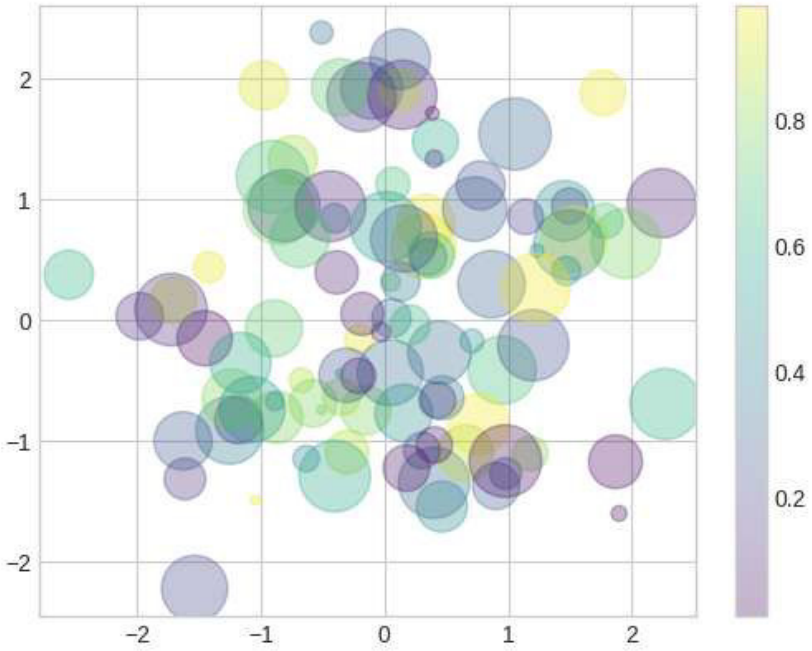
```
<matplotlib.collections.PathCollection at 0x78559065cd00>
```



Продемонстрируем это, создав случайную диаграмму рассеяния с точками различных цветов и размеров. Чтобы лучше видеть

перекрывающиеся результаты, воспользуемся ключевым словом alpha для настройки уровня прозрачности

```
rng = np.random.RandomState(0)
x = rng.randn(100)
y = rng.randn(100)
colors = rng.rand(100)
sizes = 1000 * rng.rand(100)

plt.scatter(x, y, c=colors, s=sizes, alpha=0.3, cmap='viridis')
plt.colorbar();
```



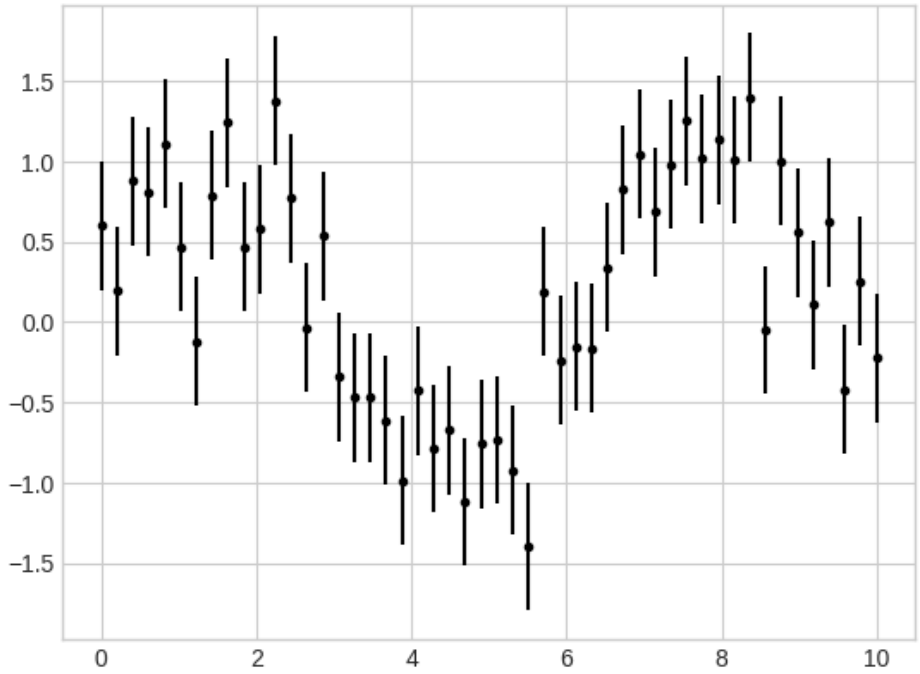
Визуализация погрешностей

Простые планки погрешностей

Точный учет погрешностей, как и точная информация о самих значениях важен для любых научных измерений. Например, представьте, что я использую некоторые астрофизические наблюдения для оценки постоянной Хаббла, то есть измеряю скорость расширения Вселенной в данной точке. Мне известно, что в современных источниках по этому вопросу указывается значение около 71 (км/с)/Мпк, а я с помощью моего метода получил значение 74 (км/с)/Мпк. Не противоречат ли значения друг другу? По вышеприведенной информации ответить на этот вопрос невозможно.

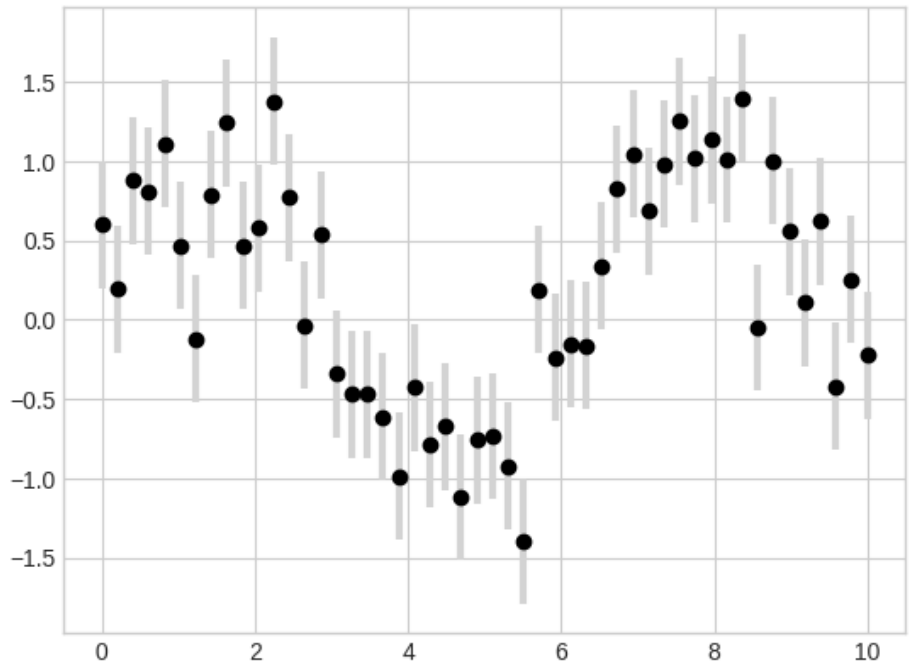
```
x = np.linspace(0, 10, 50)
dy = 0.4
y = np.sin(x) + dy * np.random.randn(50)

plt.errorbar(x, y, yerr=dy, fmt='.k')
```

<ErrorbarContainer object of 3 artists>



Здесь fmt — код форматирования, управляющий внешним видом линий и точек, его синтаксис совпадает с сокращенным синтаксисом, используемым в функции plt.plot, описываемой в разделах «Простые линейные графики» и «Простые диаграммы рассеяния» данной главы. Помимо этих простейших, у функции errorbar есть множество параметров для более тонкой настройки выводимых данных. С помощью этих дополнительных параметров вы можете легко настроить в соответствии со своими требованиями внешний вид графика планок погрешностей. Планки погрешностей удобно делать более светлыми, чем точки, особенно на насыщенных графиках.

```
plt.errorbar(x, y, yerr=dy, fmt='o', color='black',
             ecolor='lightgray', elinewidth=3, capsize=0);
```



Графики плотности и контурные графики

Иногда удобно отображать трехмерные данные в двумерной плоскости с помощью контуров или областей, кодированных различными цветами. Существует три предназначенные для этой цели функции библиотеки Matplotlib:

- plt.contour — для контурных графиков;
- plt.contourf — для контурных графиков с заполнением;
- plt.imshow — для отображения изображений. В этом разделе мы рассмотрим несколько примеров использования данных функций.

Визуализация трехмерной функции.

Начнем с демонстрации контурного графика для функции вида z = f (x, y), воспользовавшись для этой цели функцией f

```
def f(x, y):
    return np.sin(x) ** 10 + np.cos(10 + y * x) * np.cos(x)
```

Создать контурный график можно с помощью функции plt.contour. У нее имеется три аргумента: координатная сетка значений x, координатная сетка значений y и координатная сетка значений z. Значения x и y представлены точками на графике, а значения z будут представлены контурами уровней. Вероятно, наиболее простой способ подготовить такие

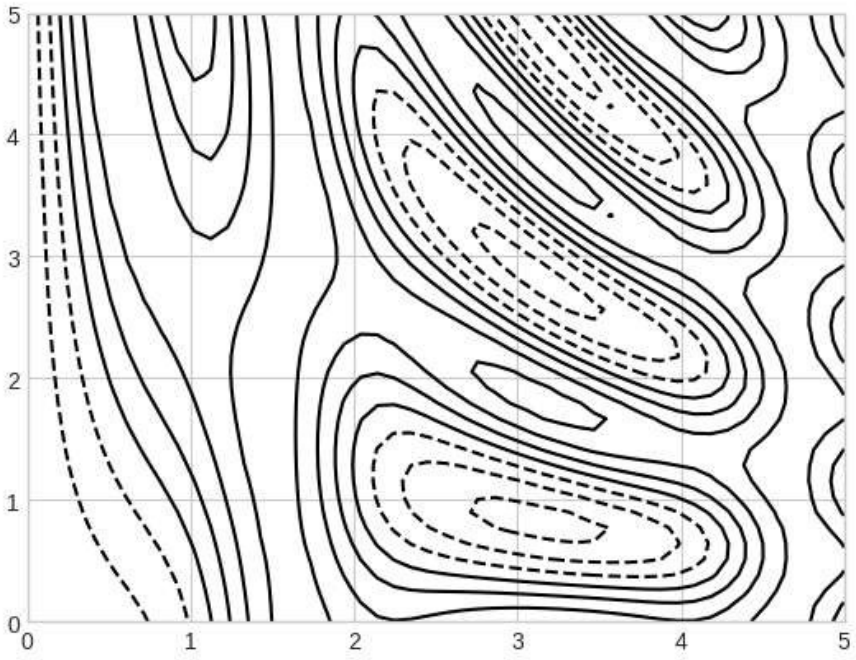
данные — воспользоваться функцией np.meshgrid, формирующей двумерные координатные сетки из одномерных массивов:

```
x = np.linspace(0, 5, 50)
y = np.linspace(0, 5, 40)

X, Y = np.meshgrid(x, y)
Z = f(X, Y)
```

Теперь посмотрим, как это отображается на обычном линейном контурном графике

```
plt.contour(X, Y, Z, colors='black')
```

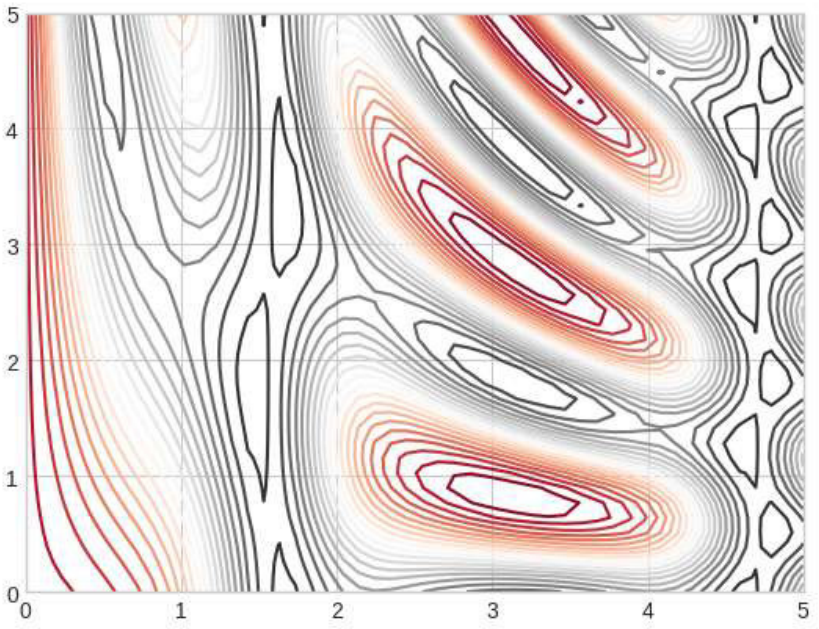
```
<matplotlib.contour.QuadContourSet at 0x785590453730>
```



Обратите внимание, что по умолчанию при использовании одного цвета отрицательные значения обозначаются штриховыми линиями, а положительные — сплошными. Можно также кодировать линии различными цветами, задав карты цветов с помощью аргумента cmap.

```
plt.contour(X, Y, Z, 20, cmap='RdGy')
```

```
<matplotlib.contour.QuadContourSet at 0x7855904f7e80>
```

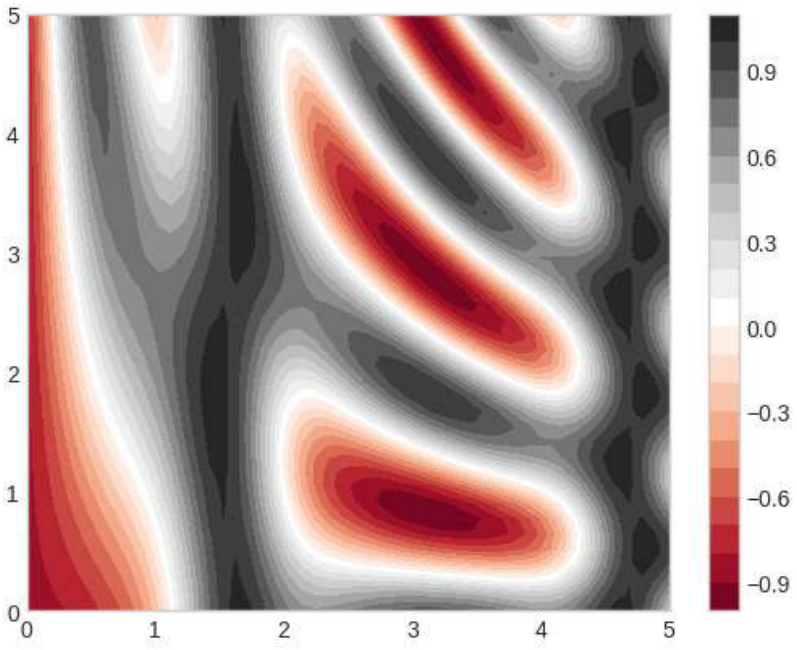


Наш график приобрел более приятный глазу вид, но промежутки между линиями несколько отвлекают внимание. Изменить это можно, воспользовавшись контурным графиком с заполнением, доступным посредством функции plt.contourf() (обратите внимание на букву f в конце ее названия), синтаксис которой не отличается от синтаксиса функции plt.contour().

Вдобавок мы воспользуемся командой plt.colorbar(), автоматически создающей для графика дополнительную ось с маркированной информацией о цвете:

```
plt.contourf(X, Y, Z, 20, cmap='RdGy')
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x78559021b9a0>
```

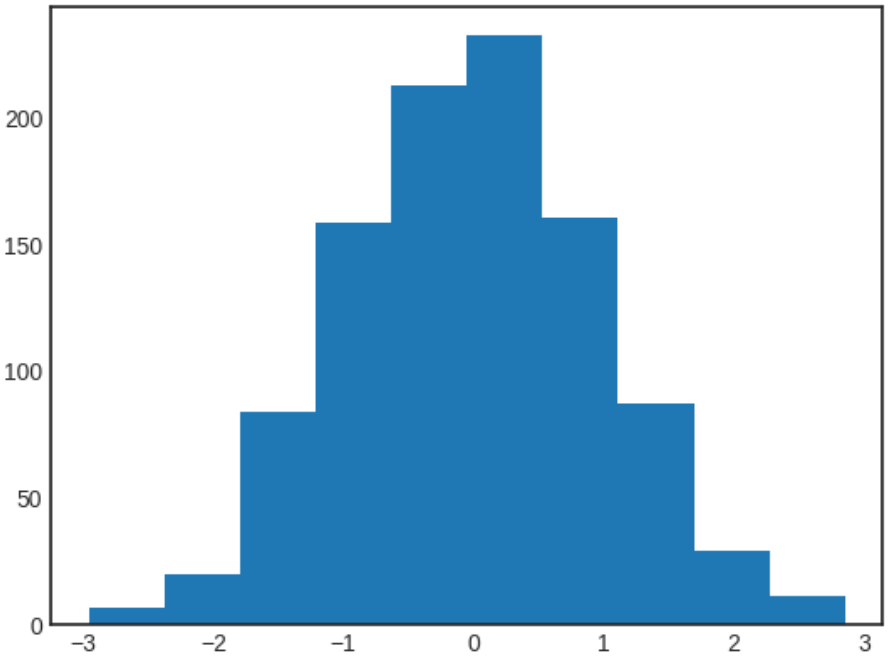


Гистограммы, разбиения по интервалам и плотность

Простая гистограмма может принести огромную пользу при первичном анализе набора данных. Ранее мы видели пример использования функции библиотеки Matplotlib для создания простой гистограммы в одну строку после выполнения всех обычных импортов

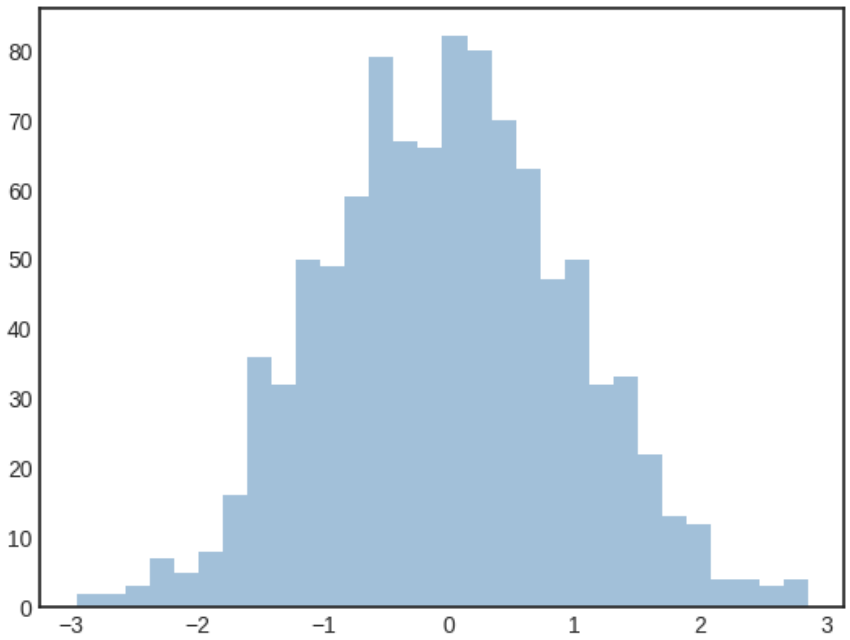
```
plt.style.use('seaborn-white')
data = np.random.randn(1000)
plt.hist(data);
```

```
plt.style.use('seaborn-white')
```



У функции hist() имеется множество параметров для настройки как вычисления, так и отображения. Вот пример гистограммы с детальными пользовательскими настройками

```
plt.hist(data, bins=30, alpha=0.5,
         histtype='stepfilled', color='steelblue',
         edgecolor='none');
```



Двумерные гистограммы и разбиения по интервалам

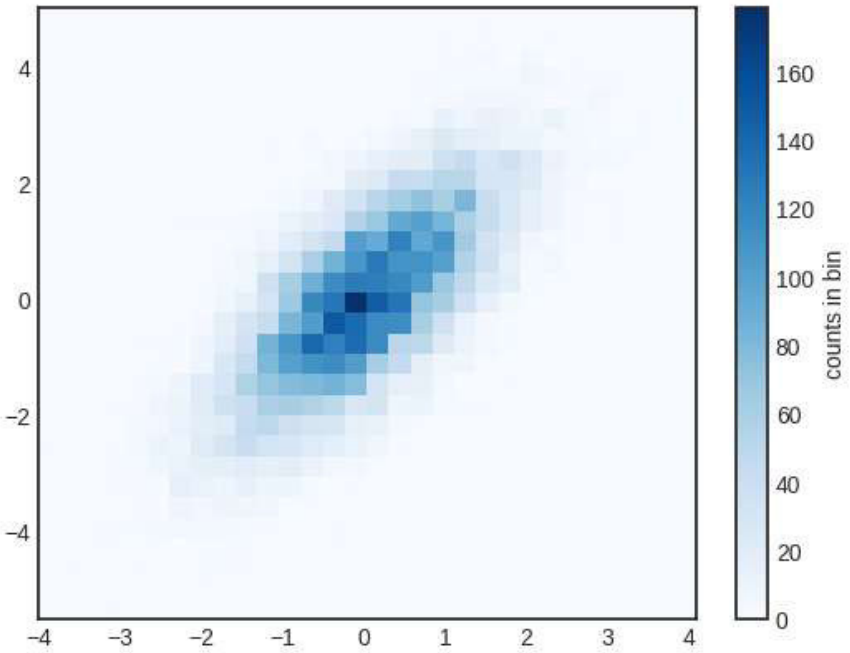
Аналогично тому, как мы создавали одномерные гистограммы, разбивая последовательность чисел по интервалам, можно создавать и двумерные гистограммы, распределяя точки по двумерным интервалам. Рассмотрим несколько способов выполнения. Начнем с описания данных массивов x и y, полученных из многомерного Гауссова распределения:

```
mean = [0, 0]
cov = [[1, 1], [1, 2]]
x, y = np.random.multivariate_normal(mean, cov, 10000).T
```

Функция plt.hist2d: двумерная гистограмма

Один из простых способов нарисовать двумерную гистограмму — воспользоваться функцией plt.hist2d библиотеки Matplotlib

```
plt.hist2d(x, y, bins=30, cmap='Blues')
cb = plt.colorbar()
cb.set_label('counts in bin')
```

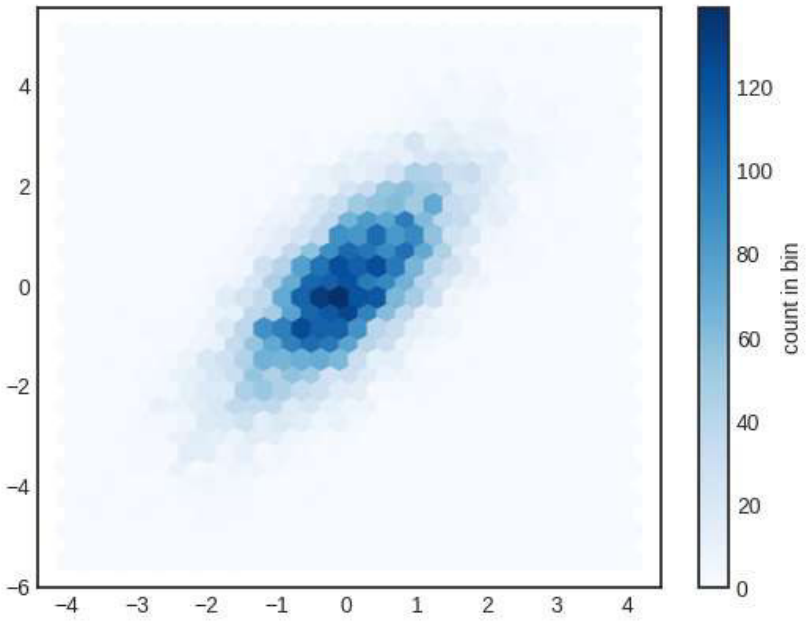


Функция plt.hexbin: гексагональное разбиение по интервалам

Двумерная гистограмма создает мозаичное представление квадратами вдоль координатных осей. Другая геометрическая фигура для подобного мозаичного представления — правильный шестиугольник. Для этих целей библиотека Matplotlib предоставляет функцию plt.hexbin —

двумерный набор данных, разбитых по интервалам на сетке из шестиугольников

```
plt.hexbin(x, y, gridsize=30, cmap='Blues')
cb = plt.colorbar(label='count in bin')
```



Задания

1) Выполнить все примеры выше в среде Google Colab.
2) Построить гистограмму для по данным своего варианта. Номер варианта соответствует порядковому номеру в списке. Данные можно скачать по ссылке: https://drive.google.com/file/d/1FQy2FVNlph8dy889Gvf8yVKMNC3XLUoH/view?usp=drive_link

или по QR коду:

3) Построить график функции в указанном промежутке. Цвет линии – красный, тип линии – сплошной. Ось абсцисс подписать “x”, ось

ординат подписать “y”. Задать метку построенной линии. Задания можно скачать по ссылке: https://drive.google.com/file/d/1yb6rjIYma9c7v09GihhTIrnMEkQJP7AT/view?usp=sharing

или по QR коду:

Полноценную версию блокнота Google Colab в pdf формате можно посмотреть по QR коду:

Построение графиков с помощью библиотеки Seaborn

Библиотека Matplotlib зарекомендовала себя как невероятно удобный и популярный инструмент визуализации, но даже заядлые ее пользователи признают, что она зачастую оставляет желать лучшего.

Библиотека Seaborn — решение этих проблем. Seaborn предоставляет API поверх библиотеки Matplotlib, обеспечивающий разумные варианты стилей графиков и цветов по умолчанию, определяющий простые высокоуровневые функции для часто встречающихся типов графиков и хорошо интегрирующийся с функциональностью, предоставляемой объектами DataFrame библиотеки Pandas.

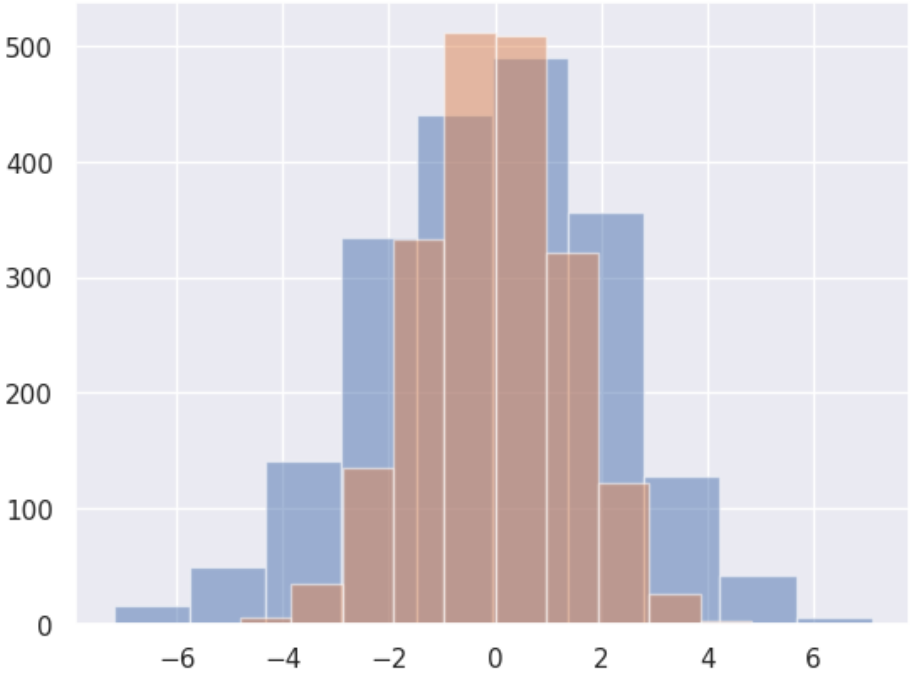
По принятым соглашениям Seaborn импортируется под именем sns:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
```

Гистограммы, KDE и плотности

Зачастую все, что нужно сделать при визуализации статистических данных, — это построить гистограмму и график совместного распределения переменных. Мы уже видели, что в библиотеке Matplotlib сделать это относительно несложно:

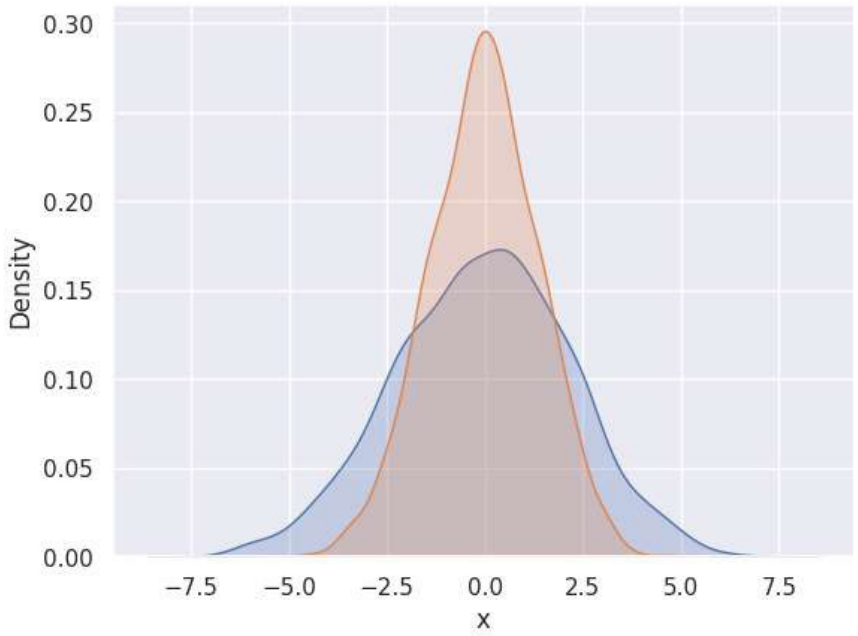
```
import pandas as pd
data = np.random.multivariate_normal([0, 0], [[5, 2], [2, 2]], size=2000)
data = pd.DataFrame(data, columns=['x', 'y'])

for col in 'xy':
    plt.hist(data[col], alpha=0.5)
```



Вместо гистограммы можно получить гладкую оценку распределения путем ядерной оценки плотности распределения, которую Seaborn выполняет с помощью функции sns.kdeplot

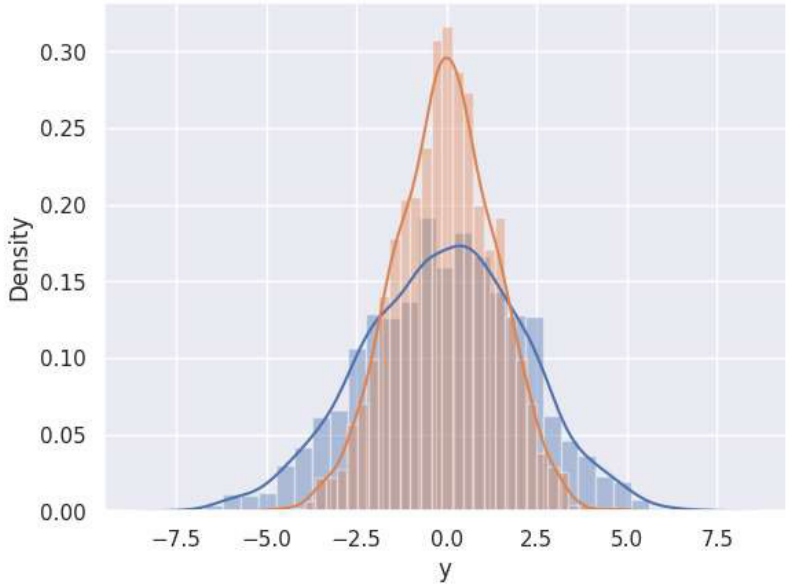
```
for col in 'xy':
    sns.kdeplot(data[col], shade=True)
```

```
sns.kdeplot(data[col], shade=True)
```



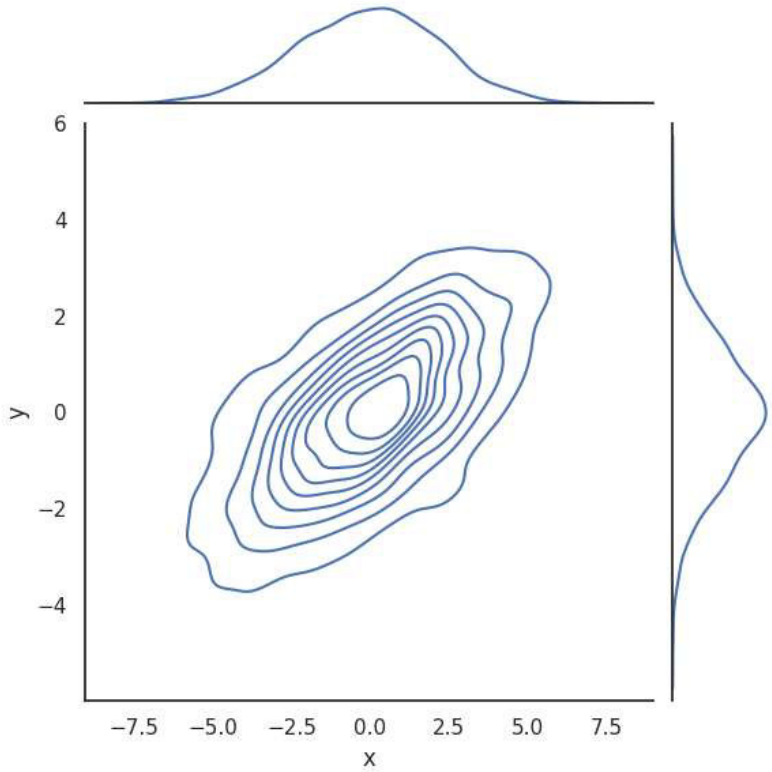
С помощью функции distplot можно сочетать гистограммы и KDE

```
sns.distplot(data['x'])
sns.distplot(data['y']);
```



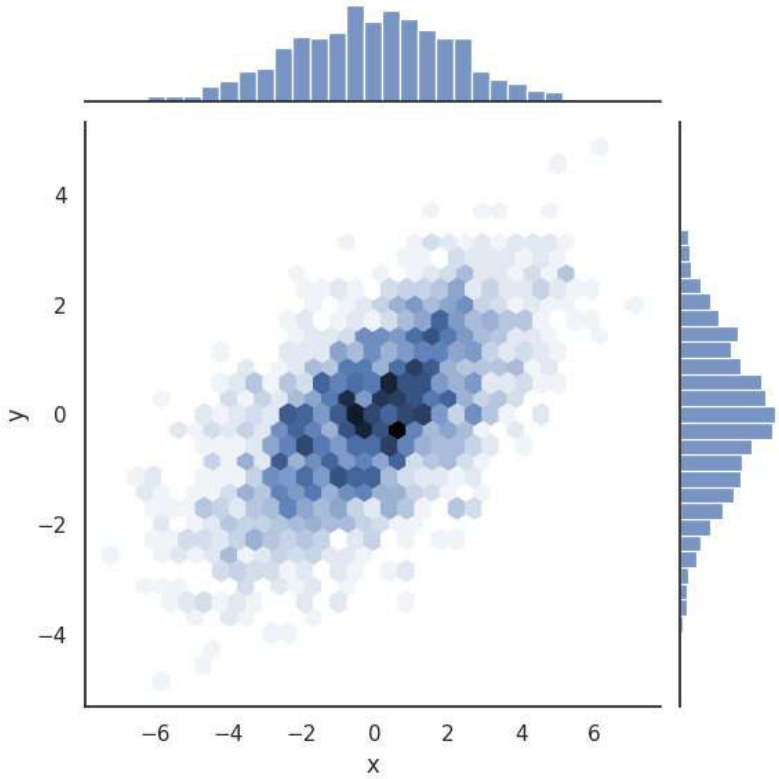
Посмотреть на совместное распределение и частные распределения можно, воспользовавшись функцией sns.jointplot.

```
with sns.axes_style('white'):
    sns.jointplot(data = data, x = "x", y = "y", kind='kde');
```



Функции jointplot можно передавать и другие параметры, например можно воспользоваться гистограммой на базе шестиугольников

```
with sns.axes_style('white'):
    sns.jointplot(data = data, x = "x", y = "y", kind='hex')
```

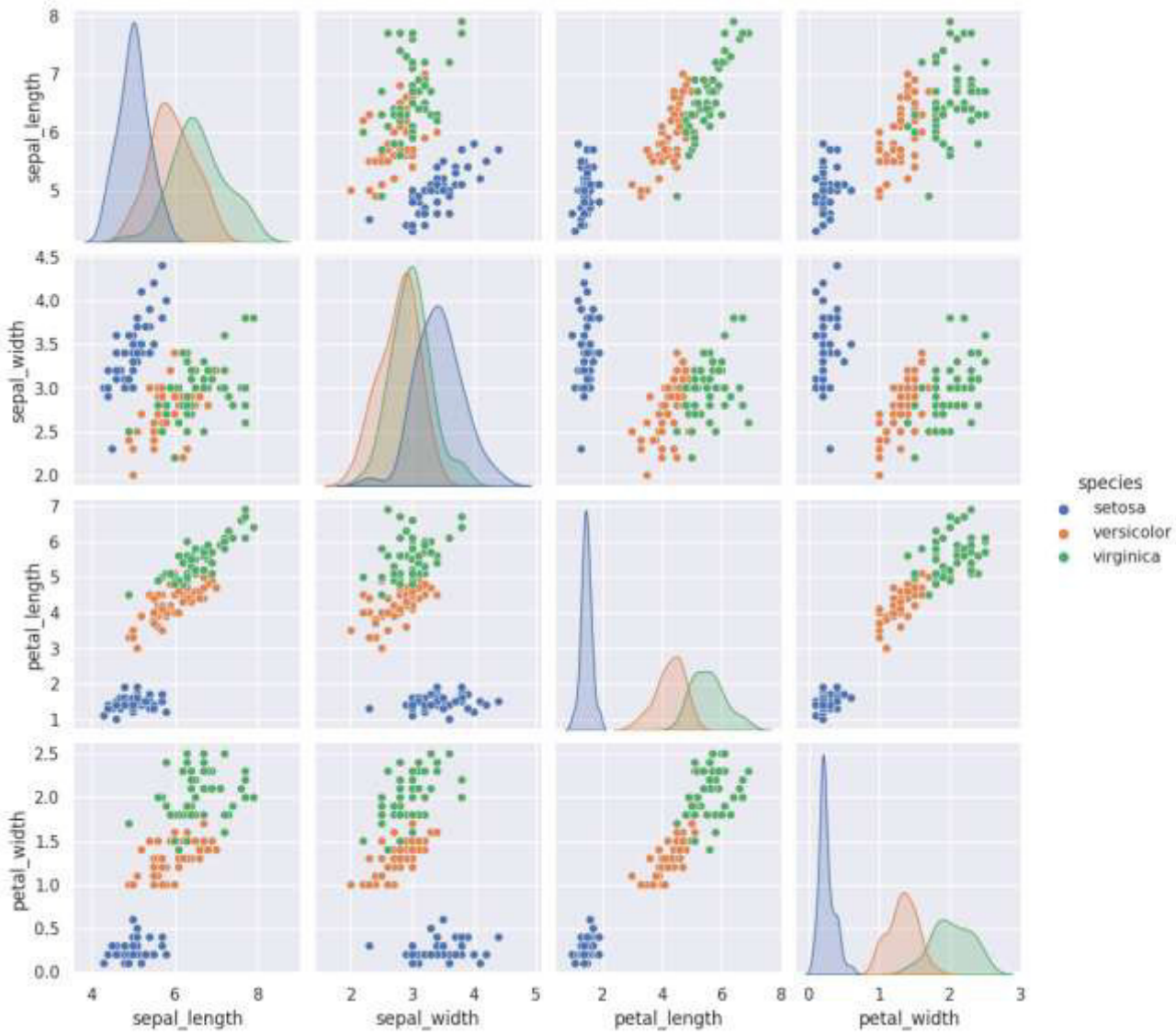


При обобщении графиков совместных распределений на наборы данных более высоких размерностей мы постепенно приходим к графикам пар (pair plots). Они очень удобны для изучения зависимостей между многомерными данными, когда необходимо построить график всех пар значений. Продемонстрируем это на наборе данных Iris, содержащем измерения лепестков и чашелистиков трех видов ирисов:

```
iris = sns.load_dataset("iris")
iris.head()
```

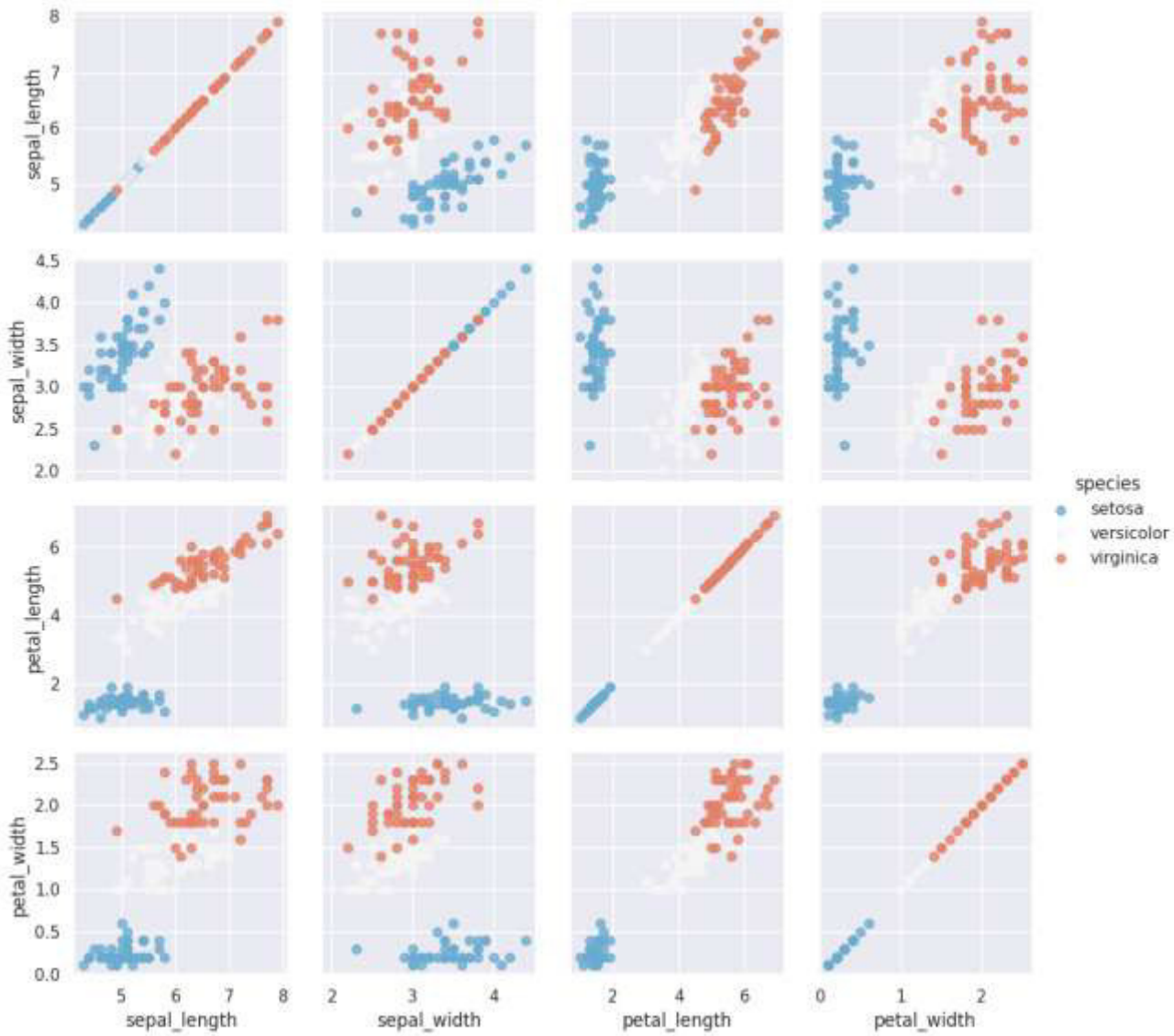
Визуализация многомерных зависимостей между выборками сводится к вызову функции sns.pairplot:

```
sns.pairplot(iris, hue='species', size=2.5);
```



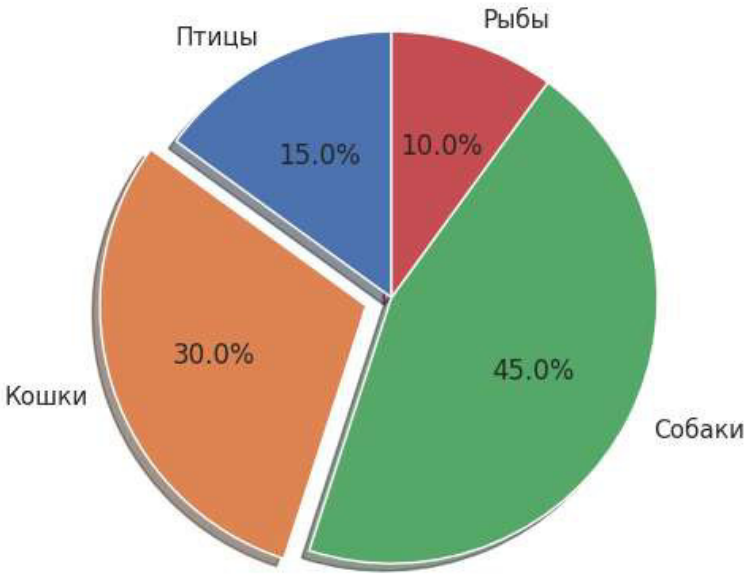
Выясним, существует ли какая-либо корреляция между переменными. Для построения графиков всех этих корреляций мы воспользуемся функцией pairgrid:

```
iris.head(3)

g = sns.PairGrid(iris, vars=['sepal_length', 'sepal_width', 'petal_length',
'petal_width'],
                 hue='species', palette='RdBu_r')
g.map(plt.scatter, alpha=0.8)
g.add_legend();
```



Круговые диаграммы

```
labels = 'Птицы', 'Кошки', 'Собаки', 'Рыбы'
sizes = [15, 30, 45, 10]
explode = (0, 0.1, 0, 0)  # Выделяем отдельно второй раздел

fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)
ax1.axis('equal')  # Equal - Равное соотношение сторон гарантирует, что пирог будет нарисован в виде круга.
```



Загрузка рисунков

```
image = plt.imread('/content/sample_data/sibir.jpg') # указать путь к файлу рисунка

fig, ax = plt.subplots()
ax.imshow(image)
ax.axis('off')  # отключить оси координат x и y
plt.show()
```

Задания

1) Выполнить все примеры выше в среде Google Colab.

2) Скачать фотографию по ссылке:
https://drive.google.com/file/d/1vi3oD8R62cal6FfxjJuupCvrBPUUPjk3/view?usp=drive_link
или по QR коду:

и загрузить в блокнот Google Colab. Отключить отображение осей координат. Подписать фотографию.

3) Построить круговую диаграмму по данным своего варианта. Отделить один из разделов построенной диаграммы. Данные можно скачать по ссылке:
https://drive.google.com/file/d/1rvVc6kEqpSzk9yKuJSpRGg6WXPFFIstS/view?usp=sharing
или по QR коду:

Полноценную версию блокнота Google Colab в pdf формате можно посмотреть по QR коду:

Математическая библиотека SymPy

SymPy — это библиотека Python для выполнения символьных вычислений. Это система компьютерной алгебры, которая может выступать как отдельное приложение, так и в качестве библиотеки для других приложений. Поработать с ней онлайн можно на https://live.sympy.org/. Поскольку это чистая библиотека Python, ее можно использовать даже в интерактивном режиме.

В SymPy есть разные функции, которые применяются в сфере символьных вычислений, математического анализа, алгебры, дискретной математики, квантовой физики и так далее. SymPy может представлять результат в разных форматах: LaTeX, MathML и так далее. Распространяется библиотека по лицензии New BSD. Первыми эту библиотеку выпустили разработчики Ondřej Čertík и Aaron Meurer в 2007 году. Текущая актуальная версия библиотеки — 1.6.2.

Вот где применяется SymPy:

- Многочлены
- Математический анализ
- Дискретная математика
- Матрицы
- Геометрия
- Построение графиков
- Физика
- Статистика
- Комбинаторика

Символьные вычисления в SymPy

Символьные вычисления — это разработка алгоритмов для управления математическими выражениями и другими объектами. Такие вычисления объединяют математику и компьютерные науки для решения математических выражений с помощью математических символов.

Система компьютерной алгебры же, такая как SymPy, оценивает алгебраические выражения с помощью тех же символов, которые используются в традиционных ручных методах. Например, квадратный корень числа с помощью модуля math в Python вычисляется вот так:

```
import math
print(math.sqrt(25)) # квадратный корень из 25
print(math.sqrt(7))  # квадратный корень из 7
```

```
5.0
2.6457513110645907
```

Как можно увидеть, квадратный корень числа 7 вычисляется приблизительно. Но в SymPy квадратные корни чисел, которые не являются идеальными квадратами, просто не вычисляются:

```
import sympy
print(sympy.sqrt(7))
```

```
sqrt(7)
```

Это можно упростить и показать результат выражения символически таким вот образом:

```
print(math.sqrt(12))
print(sympy.sqrt(12))
```

```
3.4641016151377544
2*sqrt(3)
```

В случае с модулем math вернется число, а вот в SymPy — формула.

```
from sympy import *
x = Symbol('x')
expr = integrate(x**x, x)
expr
```

$$\int x^x \, dx$$

Квадратный корень неидеального корня также может быть представлен в формате **LaTeX** с помощью привычных символов:

```
x = 7
sqrt(x)
```

$$\sqrt{7}$$

Символьные вычисления с помощью таких систем, как **SymPy**, помогают выполнять вычисления самого разного рода (производные, интегралы, пределы, решение уравнений, работа с матрицами) в

символьном виде. В пакете **SymPy** есть разные модули, которые помогают строить графики, выводить результат (**LaTeX**), заниматься физикой, статистикой, комбинаторикой, числовой теорией, геометрией, логикой и так далее.

Числа

Основной модуль в **SymPy** включает класс **Number**, представляющий атомарные числа. У него есть пара подклассов: **Float** и **Rational**. В **Rational** также входит **Integer**.

Класс Float

Float представляет числа с плавающей точкой произвольной точности:

```
from sympy import Float
Float(6.32)
```

6.32

SymPy может конвертировать целое число или строку в число с плавающей точкой:

```
Float(10)
```

10.0

Представить число дробью можно с помощью объекта класса **Rational**, где знаменатель — не 0:

```
Rational(3/4)
```

$$\frac{3}{4}$$

Если число с плавающей точкой передать в конструктор Rational(), то он вернет дробь:

```
Rational(3.65)
```

$$\frac{8219069319951155}{2251799813685248}$$

Для упрощения можно указать ограничение знаменателя:

```
Rational(0.2).limit_denominator(100)
```

$$\frac{1}{5}$$

Выведется дробь 1/5 вместо 3602879701896397/18014398509481984.

Если же в конструктор передать строку, то вернется рациональное число произвольной точности:

```
Rational("3.65")
```

$$\frac{73}{20}$$

Также рациональное число можно получить, если в качестве аргументов передать два числа. Числитель и знаменатель доступны в виде свойств:

```
a = Rational(3, 5)
print(a)
print("числитель:{}, знаменатель:{}".format(a.p, a.q)) # числитель:3, знаменатель:5
```

```
3/5
числитель:3, знаменатель:5
```

Класс Integer

Класс **Integer** в **SymPy** представляет целое число любого размера. Конструктор принимает рациональные и числа с плавающей точкой. В результате он откидывает дробную часть:

```
Integer(10)
```

10

```
Integer(3.4)
```

3

```
Integer(2/7)
```

0

Также есть класс **RealNumber**, который является алиасом для Float. В **SymPy** есть классы-одиночки **Zero** и **One**, доступные через **S.Zero** и **S.One** соответственно.

```
S.Zero
```

0

S.One

1

Другие числовые объекты-одиночки — **Half**, **NaN**, **Infinity** и **ImaginaryUnit**.

```
print(S.Half)
```

1/2

```
print(S.NaN)
```

nan

Бесконечность представлена в виде объекта-символа oo или как S.**Infinity**:

```
from sympy import oo
oo
```

∞

ImaginaryUnit можно импортировать как символ I, а получить к нему доступ — через **S.ImaginaryUnit**.

```
from sympy import I
I    # мнимая единица
```

i

Символы

Symbol — самый важный класс в библиотеке **SymPy**. Как уже упоминалось ранее, символьные вычисления выполняются с помощью символов. И переменные **SymPy** являются объектами класса **Symbol**.

Аргумент функции **Symbol()** — это строка, содержащая символ, который можно присвоить переменной.

```
from sympy import Symbol
x = Symbol('x')
y = Symbol('y')
expr = x**2 + y**2
expr
```

$$x^2 + y^2$$

Символ может включать больше одной буквы:

```
from sympy import Symbol
s = Symbol('side')
s**3
```

$$side^3$$

Также в SymPy есть функция Symbols(), с помощью которой можно определить несколько символов за раз. Строка содержит названия переменных, разделенные запятыми или пробелами.

```
from sympy import symbols
x, y, z = symbols("x, y, z")
y
```

$$y$$

В модуле abc можно найти элементы латинского и греческого алфавитов в виде символов. Таким образом вместо создания экземпляра Symbol можно использовать метод:

```
from sympy.abc import x, z
x
```

$$x$$

Индексированные символы (последовательность слов с цифрами) можно определить с помощью синтаксиса, напоминающего функцию range(). Диапазоны обозначаются двоеточием. Тип диапазона определяется символом справа от двоеточия. Если это цифра, то все смежные цифры слева воспринимаются как неотрицательное начальное значение.

Смежные цифры справа берутся на 1 больше конечного значения.

```
from sympy import symbols
symbols('a:5')
```

(a0, a1, a2, a3, a4)

```
symbols('mark(1:4)')
```

(mark1, mark2, mark3)

Одна из базовых операций в математических выражениях — подстановка. Функция **subs()** заменяет все случаи первого параметра на второй.

```
from sympy.abc import x, a
expr = sin(x) * sin(x) + cos(x) * cos(x)
expr
```

$$\sin^2(x) + \cos^2(x)$$

А кодом expr.subs(x,a) мы получим туже формулу, но с a вместо x.

```
expr.subs(x,a)
```

$$\sin^2(a) + \cos^2(\mathrm{a})$$

Эта функция полезна, когда требуется вычислить определенное выражение. Например, нужно посчитать значения выражения, заменив a на 0:

```
expr.subs(x,0)
```

1

```
from sympy.abc import x
from sympy import sin, pi
expr = sin(x)
expr1 = expr.subs(x, pi)
expr1
```

0

Также функция используется для замены подвыражения другим подвыражением. В следующем примере b заменяется на $a + b$.

```
from sympy.abc import a, b
expr = (a + b)**2
expr
```

$(a + \mathrm{b})^2$

```
expr1 = expr.subs(b, a + b) # подставляем вместо b  a + b
expr1
```

$(2a + b)^2$

Функция simplify()

Функция **simplify()** используется для преобразования любого произвольного выражения, чтобы его можно было использовать как выражение **SymPy**. Обычные объекты **Python**, такие как целые числа, конвертируются в **SymPy.Integer** и так далее. Строки также конвертируются в выражения **SymPy**:

```
expr = "x**2 + 3*x +  2"
expr1 = sympify(expr)
expr1.subs(x, 2)
```

12

Функция simplify() принимает следующий аргумент: strict=False. Если установить True, то преобразованы будут только те типы, для которых определено явное преобразование. В противном случае также возникнет ошибка SimplifyError. Если же поставить False, то арифметические выражения и операторы будут конвертированы в их эквиваленты SumPy без вычисления выражения.

```
simplify("10/5 + 4/2")
```

4

Функция evalf()

Функция вычисляет данное числовое выражение с точностью до 100 цифр после плавающей точки. Она также принимает параметр subs, как объект словаря с числовыми значениями для символов. Например такое выражение:

```
from sympy.abc import r
expr = pi * r**2
expr
```

$$\pi r^2$$

```
expr.subs({r: 5})
```

$$25\pi$$

```
expr.evalf(subs={r: 5})
```

78.5398163397448

Функция lambdify()

Функция **lambdify()** переводит выражения **SymPy** в функции **Python**. Если выражение, которое нужно вычислить, затрагивает диапазон значений, то функция **evalf()** становится неэффективной. Функция lambdify действует как лямбда-функция с тем исключением, что она конвертирует **SymPy** в имена данной числовой библиотеки, обычно **NumPy**. По умолчанию же она реализована на основе стандартной библиотеки math.

```
expr =1 / sin(x)
f = lambdify(x, expr)
f(3.14)
```

627.8831939138764

У выражения может быть больше одной переменной. В таком случае первым аргументом функции является список переменных, а после него — само выражение:

```
expr = a**2 + b**2
f = lambdify([a, b], expr)
f(2, 3)
```

13

Но чтобы использовать numpy в качестве основной библиотеки, ее нужно передать в качестве аргумента функции lambdify().

```
f = lambdify([a, b], expr, "numpy")
```

Логические выражения

Булевы функции расположены в модуле sympy.basic.booleanarg. Их можно создать и с помощью стандартных операторов Python: & (And), | (Or), ~ (Not), а также >> и <<. Булевы выражения наследуются от класса Basic.

BooleanTrue.

Эта функция является эквивалентом True из Python. Она возвращает объект-одиночку, доступ к которому можно получить и с помощью S.true.

```
from sympy import *
x = sympify(true)
x, S.true
```

(True, True)

BooleanFalse.

А эта функция является эквивалентом False. Ее можно достать с помощью S.False.

```
x = sympify(false)
x, S.false
```

(False, False)

And.

Функция логического **AND** оценивает два аргумента и возвращает False, если хотя бы один из них является False. Эта функция заменяет оператор &.

```
from sympy.logic.boolalg import And
x, y = symbols('x y')
x = True
y = True
And(x, y), x & y
```

(True, True)

Or.

Оценивает два выражения и возвращает True, если хотя бы одно из них является True. Это же поведение можно получить с помощью оператора |.

```
from sympy.logic.boolalg import Or
x, y = symbols('x y')
x = True
y = False
Or(x, x|y)
```

True

Not.

Результат этой функции — отрицание булево аргумента. True, если аргумент является False, и False в противном случае. В Python за это отвечает оператор ~

```
from sympy.logic.boolalg import Or,And,Not
x, y = symbols('x y')
x = True
y = False
Not(x), Not(y)
```

(False, True)

Xor.

Логический **XOR** (исключающий **OR**) возвращает True, если нечетное количество аргументов равняется True, а остальные — False. False же вернется в том случае, если четное количество аргументов True, а остальные — False. То же поведение работает в случае оператора ^.

```
from sympy.logic.boolalg import Xor
x, y = symbols('x y')
x = True
y = False
Xor(x, y)
```

True

В предыдущем примере один(нечетное число) аргумент является True, поэтому Xor вернет True. Если же количество истинных аргументов будет четным, результатом будет False, как показано дальше.

```
Xor(x, x^y)
```

False

Nand.

Выполняет логическую операцию **NAND**. Оценивает аргументы и возвращает True, если хотя бы один из них равен False, и False — если они истинные.

```
from sympy.logic.boolalg import Nand
a, b, c = (True, False, True)
Nand(a, c), Nand(a, b)
```

(False, True)

Nor.

Выполняет логическую операцию **NOR**. Оценивает аргументы и возвращает False, если один из них True, или же True, если все — False.

```
from sympy.logic.boolalg import Nor
a, b = False, True
Nor(a), Nor(a, b)
```

(True, False)

Equivalent.

Эта функция возвращает отношение эквивалентности. **Equivalent(A, B)** будет равно True тогда и только тогда, когда A и B оба будут True или False. Функция вернет True, если все аргументы являются логически эквивалентными. В противном случае — False.

```
from sympy.logic.boolalg import Equivalent
a, b = True, False
Equivalent(a, b), Equivalent(a, True)
```

(False, True)

Функции упрощения

SymPy умеет упрощать математические выражения. Для этого есть множество функций. Основная называется **simplify()**, и ее основная задача — представить выражение в максимально простом виде.

simplify

Это функция объявлена в модуле **sympy.simplify**. Она пытается применить методы интеллектуальной эвристики, чтобы сделать входящее выражение «проще». Следующий код упрощает такое выражение:
$\sin^2(x) + \cos^2(x)$

```
x = Symbol('x')
expr = sin(x)**2 + cos(x)**2
expr  # выражение до упрощения
```

$$\sin^2(x) + \cos^2(x)$$

```
simplify(expr) # после упрощения
```

$$1$$

expand

Одна из самых распространенных функций упрощения в **SymPy**. Она используется для разложения полиномиальных выражений. Например:

```
a, b = symbols('a b')
(a+b)**2
```

$$(a + b)^2$$

```
expand((a+b)**2) # после оамкрытия скобок
```

$$a^2 + 2ab + b^2$$

```
(a+b)*(a-b)
```

$$(a - b)(a + b)$$

```
expand((a+b)*(a-b))
```

$$a^2 - b^2$$

Функция **expand()** делает выражение больше, а не меньше. Обычно это так и работает, но часто получается так, что выражение становится меньше после использования функции:

```
(x + 1)*(x - 2) - (x - 1)*x
```

$$-x(x - 1) + (x - 2)(x + 1)$$

```
expand((x + 1)*(x - 2) - (x - 1)*x)
```

$$-2$$

factor

Эта функция берет многочлен и раскладывает его на неприводимые множители по рациональным числам.

```
x, y, z = symbols('x y z')
expr = (x**2*z + 4*x*y*z + 4*y**2*z)
expr
```

$$x^2z + 4xyz + 4y^2z$$

```
factor(expr)
```

$$z(x + 2y)^2$$

Функция **factor()** — это противоположность expand(). Каждый делитель, возвращаемый factor(), будет несокращаемым. Функция **factor_list()** предоставляет более структурированный вывод:

```
expr=(x**2*z + 4*x*y*z + 4*y**2*z)
factor_list(expr)
```

```
(1, [(z, 1), (x + 2*y, 2)])
```

collect

Эта функция собирает дополнительные члены выражения относительно списка выражений с точностью до степеней с рациональными показателями.

```
expr = y**2*x + 4*x*y*z + 4*y**2*z + y**3 + 2*x*y
expr
```

$$xy^2 + 4xyz + 2xy + y^3 + 4y^2z$$

Результат работы collect():

```
expr = y**2*x + 4*x*y*z + 4*y**2*z + y**3 + 2*x*y
collect(expr, y)
```

$$y^3 + y^2(x + 4z) + y(4xz + 2x)$$

cancel

Эта функция берет любую рациональную функцию и приводит ее в каноническую форму p/q, где p и q — это разложенные полиномы без общих множителей. Старшие коэффициенты p и q не имеют знаменателей, то есть, являются целыми числами.

```
expr1=x**2+2*x+1
expr2=x+1
expr1/expr2
```

$$\frac{x^2 + 2x + 1}{x + 1}$$

```
cancel(expr1/expr2)
```

$$x + 1$$

Другие примеры:

```
expr = 1/x + (3*x / 2 - 2)/(x - 4)
expr
```

$$\frac{\frac{3x}{2} - 2}{x - 4} + \frac{1}{x}$$

```
cancel(expr)
```

$$\frac{3x^2 - 2x - 8}{2x^2 - 8x}$$

trigsimp

Эта функция используется для упрощения тригонометрических тождеств. Стоит отметить, что традиционные названия обратных тригонометрических функций добавляются в название функции в начале. Например, обратный косинус или арккосинус называется **acos()**:

```
from sympy import trigsimp, sin, cos
from sympy.abc import x, y
expr = 2*sin(x)**2 + 2*cos(x)**2
expr
```

$$2\sin^2(x) + 2\cos^2(x)$$

```
trigsimp(expr)
```

$$2$$

Функция **trigsimp** использует эвристику для применения наиболее подходящего тригонометрического тождества.

powersimp

Эта функция сокращает выражения, объединяя степени с аналогичными основаниями и значениями степеней.

```
expr = x**y*x**z*y**z
expr
```

$$x^y x^z y^z$$

Можно сделать так, чтоб **powsimp()** объединяла только основания или степени, указав **combine='base'** или **combine='exp'**. По умолчанию это значение равно **combine='all'**. Также можно задать параметр force. Если он будет равен True, то основания объединятся без проверок.

```
powsimp(expr, combine='base', force=True)
```

$$x^y (xy)^z$$

combsimp

Комбинаторные выражения, включающие факториал и биномы, можно упростить с помощью функции combsimp(). В SymPy есть функция factorial().

```
expr = factorial(x) / factorial(x - 1)
expr
```

$$\frac{x!}{(x-1)!}$$

```
combsimp(expr) # после упрощения
```

$$x$$

binomial

binomial(x, y) — это количество способов, какими можно выбрать элементы y из множества элементов x. Его же можно записать и как C_x^y.

```
binomial(x, y)
```

$$\binom{x}{y}$$

```
binomial(5, 2)
```

```
binomial(x + 1, y + 1) / binomial(x, y)
```

$$\frac{\binom{x+1}{y+1}}{\binom{x}{y}}$$

```
combsimp(binomial(x + 1, y + 1) / binomial(x, y))
```

$$\frac{x+1}{y+1}$$

logcombine

Эта функция принимает логарифмы и объединяет их с помощью следующих правил:

- $log(x) + log(y) == log(x * y)$ — оба положительные.
- $a * log(x) == log(x^a)$ если x является положительным и вещественным.

```
a*log(x) + log(y) - log(z)
```

$$a\log(x) + \log(y) - \log(z)$$

```
logcombine(a*log(x) + log(y) - log(z))
```

$$a\log(x) + \log(y) - \log(z)$$

Если здесь задать значение параметра force равным True, то указанные выше предположения будут считаться выполненными, если нет предположений о величине.

```
logcombine(a*log(x) + log(y) - log(z), force = True)
```

$$\log\left(\frac{x^a y}{z}\right)$$

Задания

1) Выполнить все примеры выше в среде Google Colab.
2) Упростить выражение. Номер варианта соответствует порядковому номеру в списке. Данные можно скачать по ссылке:

https://drive.google.com/file/d/15g6O5gwiDn85rMJESC_rzowCVMsysrgz/view?usp=sharing

или по QR коду:

3) Разложить на множители. Номер варианта соответствует порядковому номеру в списке. Данные можно скачать по ссылке: https://drive.google.com/file/d/1CNd3FEUc8P5-tP9neZuicc3bpVSysQh2/view?usp=sharing

или по QR коду:

4) Упростить тригонометрическое выражение. Номер варианта соответствует порядковому номеру в списке. Данные можно скачать по ссылке: https://drive.google.com/file/d/1WFJcTtghcet0XT0SWBUFMA3Wfo0FFZwn/view?usp=sharing

или по QR коду:

Полноценную версию блокнота Google Colab в pdf формате можно посмотреть по QR коду:

Символьное решение уравнений с помощью библиотеки SymPy

Этот набор инструментов включает поддержку символьных вычислений, высокоточных вычислений, сопоставления с образцом, рисования, решения уравнений, исчисления, комбинаторики, дискретной математики, геометрии, вероятности и статистики, физики и т. д. Функция более мощная, и производительность при решении уравнений.

Поскольку символы = и == определены как символ присваивания и равенства в Python, их нельзя использовать для создания символьных уравнений. Для этого в **SymPy** есть функция **Eq()**.

```
from sympy import *
x, y = symbols('x y')
Eq(x, y)
```

$$x = y$$

Поскольку $x = y$возможно только в случае $x - y = 0$, уравнение выше можно записать как:

```
Eq(x-y, 0)
```

$$x - y = 0$$

Модуль **solver** из **SymPy** предлагает функцию **solveset()**:

solveset(equation,variable,domain)

Параметр **domain** по умолчанию равен S.Complexes. С помощью функции **solveset()** можно решить алгебраическое уравнение.

Решение уравнения $x^2 - 9 = 0$

```
solveset(Eq(x**2-9, 0), x)
```

$$\{-3,3\}$$

Решение уравнения $x^2 - 3x = -2$

```
solveset(Eq(x**2-3*x, -2), x)
```

$$\{1,2\}$$

Линейное уравнение

Для решения линейных уравнений нужно использовать функцию **linsolve()**.

Например, уравнения могут быть такими:

- $x - y = 4$
- $x + y = 1$

```
x = Symbol('x')
y = Symbol('y')
linsolve([Eq(x-y, 4), Eq(x+y, 1)], [x, y])
```

$$\left\{\left(\frac{5}{2}, -\frac{3}{2}\right)\right\}$$

Функция **linsolve()** также может решать линейные уравнения в матричной форме:

```
a, b = symbols('a b')
a = Matrix([[1, -1], [1, 1]])
b = Matrix([4, 1])
linsolve([a, b], [x, y])
```

$$\left\{\left(\frac{5}{2}, -\frac{3}{2}\right)\right\}$$

Вывод будет тот же.

Нелинейное уравнение

Для таких уравнений используется функция **nonlinsolve()**. Пример такого уравнения:

```
a, b = symbols('a b')
nonlinsolve([a**2 + a, a - b], [a, b])
```

$$\{(-1,\ -1), (0,\ 0)\}$$

Дифференциальное уравнение

Для начала создайте функцию, передав **cls=Function** в функцию symbols. Для решения дифференциальных уравнений используйте dsolve.

```
x = symbols('x')
f = symbols('f', cls=Function)
f(x)
```

$$f(x)$$

Здесь $f(x)$ — это невычисленная функция. Ее производная:

```
f(x).diff(x)
```

$$\frac{d}{dx}f(x)$$

Сначала создается объект **Eq**, соответствующий следующему дифференциальному уравнению,

```
eqn = Eq(f(x).diff(x) - f(x), sin(x)) # создаем дифференциальное уравнение
eqn
```

$$-f(x)+\frac{d}{dx}f(x)=\sin(x)$$

затем решает дифференциальное уравнение $f'(x)-f(x)=sin(x)$

```
dsolve(eqn, f(x)) # решаем дифференциальное уравнение
```

$$f(x)=\left(C_1-\frac{e^{-x}\sin(x)}{2}-\frac{e^{-x}\cos(x)}{2}\right)e^{x}$$

Решим дфференциальное уравнение

$$f''(x)-f'(x)+f(x)=cos(x)$$

```
eqn = Eq(f(x).diff(x).diff(x) - f(x).diff(x) + f(x), cos(x)) # создаем дифференциальное уравнение
eqn
```

$$f(x)-\frac{d}{dx}f(x)+\frac{d^2}{dx^2}f(x)=\cos(x)$$

```
dsolve(eqn, f(x)) # решаем дифференциальное уравнение
```

$$f(x)=\left(C_1\sin\left(\frac{\sqrt{3}x}{2}\right)+C_2\cos\left(\frac{\sqrt{3}x}{2}\right)\right)e^{\frac{x}{2}}-\sin(x)$$

Решим уравнение

$$(x^2 - 1)f'(x) + 2xf^2(x) = 0$$

```
eqn = Eq((x ** 2 - 1) * f(x).diff(x) + 2*x*f(x) ** 2, 0) # создаем дифференциальное уравнение
eqn
```

$$2xf^2(x) + (x^2 - 1)\frac{d}{dx}f(x) = 0$$

```
dsolve(eqn, f(x)) # решаем дифференциальное уравнение
```

$$f(x) = -\frac{1}{C_1 - \log(x^2 - 1)}$$

```
eqn = Eq(f(x).diff(x) - 2 * x * f(x) + f(x) ** 2, 5 - x ** 2) # создаем дифференциальное уравнение
eqn
```

$$-2xf(x) + f^2(x) + \frac{d}{dx}f(x) = 5 - x^2$$

Решим уравнение Риккати

$$f'(x) - 2xf(x) + f^2(x) = 5 - x^2$$

```
dsolve(eqn, f(x)) # решаем дифференциальное уравнение
```

$$f(x) = \frac{C_1 x - 2C_1 - xe^{4x} - 2e^{4x}}{C_1 - e^{4x}}$$

Примеры решения уравнений.

Пример №1. Решить систему уравнений $2x + y = 1, x - 2y = 0$.

```
# Двоичное линейное уравнение
x = Symbol('x')
y = Symbol('y')
solved_value=solve([2*x+y-1, x-2*y], [x, y])
print(solved_value)
```

```
{x: 2/5, y: 1/5}
```

Пример №2. Решить квадратное уравнение $x^2 - 9 = 0$.

```
x = Symbol('x')
solved_value=solve([x**2-9], [x])
print(solved_value) # Эта библиотека находит все корни уравнения
```

```
[(-3,), (3,)]
```

Пример №3. Решить уравнение $x^2 + 9 = 0$. Это урвнение имеет два комплексных корня $\pm 3i$.

```
x = Symbol('x')
solved_value = solve([x ** 2 + 9], [x])
solved_value
```

```
[(-3*I,), (3*I,)]
```

```
simplify(solved_value[0]) # первый корень
```

$$(-3i)$$

```
simplify(solved_value[1]) # второй корень
```

$$(3i)$$

Пример №4. Решить уравнение $x^4 - 9 = 0$. Это урвнение имеет два комплексных корня $\pm\sqrt{3}i$ и два действительных корня $\pm\sqrt{3}$.

```
x = Symbol('x')
solved_value = solve([x ** 4 - 9], [x])
print(solved_value)
```

```
[(-sqrt(3),), (sqrt(3),), (-sqrt(3)*I,), (sqrt(3)*I,)]
```

```
simplify(solved_value[0]) # первый корень
```

$$\left(-\sqrt{3}\right)$$

```
simplify(solved_value[1]) # второй корень
```

$$\left(\sqrt{3}\right)$$

```
simplify(solved_value[2]) # третий корень
```

$$\left(-\sqrt{3}i\right)$$

```
simplify(solved_value[3]) # четвертый корень
```

$$(\sqrt{3}i)$$

Пример №5. Решить уравнение $\sin(x) = \frac{1}{2}$. Метод вычисляет все корни в промежутке $[0; 2\pi]$. Это уравнение имеет два корня $\frac{\pi}{6}, \frac{5\pi}{6}$.

```
solved_value = solve([sin(x) - 1/2], [x])
print(solved_value)
```

```
[(0.523598775598299,), (2.61799387799149,)]
```

Пример №6. Решить уравнение $\sin(x) = 1$.

```
x = Symbol('x')
solved_value = solve([sin(x) - 1], [x])
print(solved_value)
```

```
[(pi/2,)]
```

```
simplify(solved_value[0])
```

$$\left(\frac{\pi}{2}\right)$$

Пример №7. Решить систему

$$x^2 + 2xy = 6,$$

$$2xy - 2y^2 = -3.$$

Это сложная задача, как бы люди его ни делали, его сложно понять и итеративно невозможно решить с помощью **SciPy**. Но мощная функция **Sympy** может очень хорошо решить это уравнение.

```
x = Symbol('x')
y = Symbol('y')
solved_value=solve([x**2+2*x*y-6,2*x*y-2*y**2+3], [x, y])
print(solved_value)
```

```
[(-(-3 + sqrt(13))*sqrt(sqrt(13)/2 + 2), -sqrt(sqrt(13)/2 + 2)), ((-3 + sqrt(13))*sqrt(sqrt(13)/2 + 2), sqrt(sqrt(13)/2 + 2)), (-sqrt(2 - sqrt(13)/2)*(-sqrt(13) - 3), -sqrt(2 - sqrt(13)/2)), (sqrt(2 - sqrt(13)/2)*(-sqrt(13) - 3), sqrt(2 - sqrt(13)/2))]
```

```
simplify(solved_value[0]) # 1-я пара решения x и y
```

$$\left(-(-3+\sqrt{13})\sqrt{\frac{\sqrt{13}}{2}+2},\ -\sqrt{\frac{\sqrt{13}}{2}+2}\right)$$

simplify(solved_value[1]) *# 2-я пара решения x и y*

$$\left((-3+\sqrt{13})\sqrt{\frac{\sqrt{13}}{2}+2},\ \sqrt{\frac{\sqrt{13}}{2}+2}\right)$$

simplify(solved_value[2]) *# 3-я пара решения x и y*

$$\left(-\sqrt{2-\frac{\sqrt{13}}{2}}(-\sqrt{13}-3),\ -\sqrt{2-\frac{\sqrt{13}}{2}}\right)$$

simplify(solved_value[3]) *# 4-я пара решения x и y*

$$\left(\sqrt{2-\frac{\sqrt{13}}{2}}(-\sqrt{13}-3),\ \sqrt{2-\frac{\sqrt{13}}{2}}\right)$$

Задания

1) Выполнить все примеры выше в среде Google Colab.
2) Решить уравнение f(x) = 0. Номер варианта соответствует порядковому номеру в списке. Данные можно скачать по ссылке: https://drive.google.com/file/d/1diAnq3GeOcMXPOqZitHiP9cm5kVMs0or/view?usp=sharing

или по QR коду:

3) Решить нелинейную систему. Номер варианта соответствует порядковому номеру в списке. Данные можно скачать по ссылке: https://drive.google.com/file/d/1OGgB2918oMtroWcB_4OmgrSBrB4v0Ziv/view?usp=sharing

или по QR коду:

4) Решить дифференциальное уравнение. Номер варианта соответствует порядковому номеру в списке. Данные можно скачать по ссылке: https://drive.google.com/file/d/1c7i9486Vj3TZ9NHOAhMlETkAchbyLbHQ/view?usp=sharing

или по QR коду:

Полноценную версию блокнота Google Colab в pdf формате можно посмотреть по QR коду:

ЗАКЛЮЧЕНИЕ

В пособии изложены лишь базовые возможности Python и библиотек NumPy, Matplotlib, SciPy и SymPy. Оно может быть использовано для первоначального знакомства с языком Python и данными библиотеками с целью дальнейшего более глубокого самостоятельного изучения их возможностей.

Удобство применения Python для научных расчётов связано, в частности, с возможностью интерактивного режима работы, когда вводимые команды сразу же исполняются интерпретатором. Интерактивность позволяет быстро производить отдельные виды вычислений и визуализировать результаты расчётов без необходимости тратить время на написание отдельных программ. В пособии рассматривается интерактивный режим работы в оболочке IPython. В настоящее время данный подход развивается

в проекте Jupiter и Google Colab (URL: https://jupyter.org/, colab.research.google.com).

Jupiter представляет собой интерактивную оболочку с веб-интерфейсом для разработки программ. Веб-приложение Jupiter Notebook позволяет создавать документы, содержащие интерактивный программный код, уравнения, рисунки и обычный текст. Имеется возможность совместной работы над такими документами. Блокноты Jupiter могут использоваться для дальнейшего изучения возможностей Python для научных расчётов.

Google Colab — это бесплатная облачная платформа для создания и выполнения кода на Python с возможностью использования мощных центральных и графических процессоров. Сервис разработан на базе Google Research и предназначен для образования и научных исследований в области машинного обучения.

Особенности Google Colab**:**

- **Не нуждается в настройке**. Для работы сервису достаточно браузера и подключения к интернету.

- **Все данные и код хранятся в облаке Google**. Поэтому можно совместно работать над проектами с другими людьми, делиться кодом и результатами.
- **Поддерживает интерактивное программирование**. Код можно писать в отдельных ячейках и выполнять их по порядку, просматривая результаты выполнения после блока с кодом.
- **Позволяет использовать различные библиотеки Python**, включая популярные библиотеки для анализа данных и машинного обучения: NumPy, Pandas и TensorFlow.

Google Colab используется для различных целей, включая машинное обучение, обработку и анализ больших объёмов данных, выполнение крупномасштабных вычислений и другие.

ЛИТЕРАТУРА

Плас Дж. Вандер. Python для сложных задач: наука о данных и машинное обучение [Текст] / Плас Дж. Вандер; СПб.: Питер, 2018. – 576 с.: ил. – (Серия «Бестселлеры O'Reilly»). – ISBN 978-5-496-03068-7 (в пер.).

Грас Д. Data Science. Наука о данных с нуля. [Текст] / Грас Д. – 2-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2021. - 416 с.: ил. – ISBN 978-5-9775-6731-2 (в пер.).

Хайбрахманов, С. А. Основы научных расчётов на языке программирования Python: учеб. Пособие [Текст] / С. А. Хайбрахманов. — Челябинск: Изд-во Челяб. гос. ун-та, 2019. — 96 с.

База научных статей, переводов, таблиц, графиков, иллюстраций, примеров кода по темам - анализ данных, машинное обучение, нейронные сети, искусственный интеллект [Электронный ресурс] – Режим доступа: https://machinelearningmastery.ru/

PythonRu — образовательный блог о языке программирования Python [Электронный ресурс] – Режим доступа: https://pythonru.com/biblioteki/sympy-v-python

Лабораторная работа №1.

Верные и значащие цифры числа, сложение и умножение приближенных чисел

Цель работы: изучить понятие верных и значащих цифр числа, абсолютной и относительной погрешностей; освоить правила сложения и умножения приближенных чисел.

ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Задание 1. Выполнить пункты а)-в) согласно выданному варианту (варианты заданий приведены)

а) Определить, какое равенство точнее.

б) Округлить сомнительные цифры числа, оставив верные знаки. Определить абсолютную погрешность результата.

в) Найти предельные абсолютную и относительную погрешности приближенного числа, все цифры которого по умолчанию верные.

Рекомендации: предварительно разобрать материал лекций по теме «Погрешности».

Варианты индивидуальных заданий:

1. а) 14/17 = 0,824, $\sqrt{53} = 7{,}28$; б) 23.3748, δ = 0.27%; в) 0.645.

2. а) 7/3 =2,33, $\sqrt{58} = 7{,}62$; б) 13.5726 ± 0.0072; в) 4.8556.

3. а) 27/31 = 0,871, $\sqrt{42} = 6{,}48$, б) 0.088748, δ = 0.56%; в) 71.385.

4. а) 23/9= 2,56, $\sqrt{87} = 9{,}33$; б) 4,57633 ± 0.00042; в) 6.8346.

5. а) 6/7 =0,857, $\sqrt{41} = 6{,}40$; б) 46.7843, δ = 0.32%; в) 7.38.

6. а) 12/7 = 1,71, $\sqrt{47} = 6{,}86$; б) 0.38725 ± 0.00112; в) 0.006 46.

7. а) 21/13 = 1,62, $\sqrt{63} = 7{,}94$; б) 45.7832, δ = 0.18%; в) 3.6765.

8. а) 16/7= 2,29, $\sqrt{11} = 3{,}32$; б) 0.752 44 ± 0.00013; в) 5.374.

9. а) 18/7=2,57, $\sqrt{22} = 4{,}69$; б) 46.453, δ = 0.15%; в) 6.125.

10. а) 17/9=1,89, $\sqrt{17} = 4{,}12$; б) 0.66385 ± 0.000 42; в) 24.6.

11. а) 51/11= 4,64, $\sqrt{35} = 5{,}92$; б) 0.66385, δ = 0.34%; в) 0.543.

12. а) 19/12 = 1,58, $\sqrt{12} = 3{,}46$; б) 4.884 45 ± 0.000 52; в) 4.633.

13. а) 13/7=1,857, $\sqrt{7} = 2{,}65$; б) 2.8867, δ = 0.43%; в) 63.749.

14. а) 49/13=3,77, $\sqrt{14} = 3{,}74$; б) 5.6483± 0.0017; в) 0.008 58.

15. а) 5/3=1,667, $\sqrt{38} = 6{,}16$; б) 3.7542, δ = 0.32%; в) 0.389.

16. а) 17/11=1,545, $\sqrt{18} = 4{,}243$; б) 0.8647 ± 0.0013; в) 0.864.

17. а) 7/22= 0,318, $\sqrt{13} = 3{,}61$; б) 0.3944, δ = 0.15%; в) 21.7.

18. а) 13/17 = 0,765, $\sqrt{31} = 5{,}57$; б) 3.6878 ± 0.0013; в) 8.74.

19. а) 50/19= 2,63, $\sqrt{27} = 5{,}20$; б) 0.856 38, δ = 0.22%; в) 231.57.

20. а) 21/29= 0,724, $\sqrt{44} = 6{,}63$; б) 13.6853± 0.0023; в) 2.16.

21. а) 17/19 = 0,895, $\sqrt{52} = 7{,}21$; б) 7.521, δ = 0.12%; в) 0.5748.

22. а) 6/11=0,545, $\sqrt{83} = 9{,}11$; б) 3.7832 ± 0.0043; в) 2.678.

23. а) 16/19 = 0,842, $\sqrt{55} = 7{,}416$; б) 17.356, δ = 0.11%; в) 0.5718.

24. а) 23/15 =1,53, $\sqrt{98} = 9{,}899$; б) 8.7432 ± 0.0023; в) 0.578.

25. а) 2/21=0,095, $\sqrt{22} = 4{,}69$; б) 24.5641, δ = 0.09%; в) 4.478.

Пример решения варианта

а) 9/19=0,474, $\sqrt{103} = 10{,}149$. Найдем значения этих выражений с бóльшим числом десятичных знаков: a = 9/19 = 0,473 68..., b = $\sqrt{103}$= 10.14889...

Вычислим предельные абсолютные погрешности, округляя их с избытком:

$$\Delta_{\boldsymbol{a}} = |0{,}47368 - \ 0{,}474| = 3{,}2 \cdot 10^{-4} \leq 0{,}0004,$$

$$\Delta_{\boldsymbol{b}} = |10{,}14889 - \ 10{,}149| = 1{,}1 \cdot 10^{-4} \leq 0{,}0002.$$

Предельные относительные погрешности составляют:

$$\delta_a = \frac{\Delta_{\boldsymbol{a}}}{|a|} = \frac{0{,}0004}{0{,}474} \approx 8 \cdot 10^{-4} = 0{,}08\%,$$

$$\delta_b = \frac{\Delta_{\boldsymbol{b}}}{|b|} = \frac{0{,}0002}{10{,}149} \approx 2 \cdot 10^{-5} = 0{,}002\%.$$

Так как $\delta_b < \delta_a$ (причем намного меньше), то равенство $\sqrt{103} = 10{,}149$ является более точным.

б) Пусть приближенное число с = 72,354± 0,021. Поскольку предельная абсолютная погрешность $\Delta_c = 0{,}021 < 0{,}1$, число с придется округлить до десятых. Тогда окончательно только с верными цифрами его запишем в виде: 72,4.

Пусть приближенное число d = 5,272 известно с относительной погрешностью $\delta_d = 0{,}1\%$. Поскольку $\Delta_{\boldsymbol{d}} = |d| \cdot \delta_d = 5{,}272 \cdot 0{,}001 = 0{,}005272 < 0{,}01$, число d придется округлить до сотых; окончательно, только с верными цифрами его запишем в виде: 5,27.

в) Пусть приближенное число f =14.278. Поскольку все его цифры верные, предельная абсолютная погрешность $\Delta_{\boldsymbol{f}} \leq 0{,}001$. Тогда предельная относительная погрешность

$$\delta_f = \frac{\Delta_{\boldsymbol{f}}}{|f|} = \frac{0{,}001}{14.278} \approx 7 \cdot 10^{-5} \leq 10^{-4} = 0{,}01\%.$$

Задание 2. Выполнить пункты а)-б) согласно выданному варианту (варианты заданий приведены)

а) Сложите следующие пары чисел, округляя результат до правильного количества знаков после запятой. Объясните, почему важно учитывать количество верных цифр при сложении приближенных чисел.

б) Умножьте следующие пары чисел, округляя результат до правильного количества значащих цифр. Объясните, почему важно учитывать количество верных цифр при умножении приближенных чисел.

Варианты индивидуальных заданий:

1. а) 12,345 и 6.789; б) 1, 3105 и 0,0082
2. а) 0.00456 и 789.01; б) 32,470 и 0,003
3. а) 123.45 и 678.9; б) 6,50567 и 0,0021
4. а) 225.301 и 678.9; б) 2,30 и 1,00324
5. а) 73.45 и 72.9110; б) 7,03 и 1,0000
6. а) 123.0 и 678.912; б) 0,559 и 3,0001
7. а) 3.045 и 800.9; б) 1,90 и 4,0000
8. а) 521.7912 и 11.81; б) 2,70 и 0,0004
9. а) 3.518 и 6.90000; б) 8,05 и 5,0003
10. а) 1.0001 и 678.9; б) 19,7 и 6,00002

3. Проведите анализ результатов. Согласны ли вы с выводами:

- результаты сложения показывают, что некоторые цифры могут быть неверными из-за ограниченной точности слагаемых.
- результаты умножения показывают, что все цифры могут быть верными, если произведение достаточно велико относительно множителей.

Лабораторная работа №2.

ИНТЕРПОЛИРОВАНИЕ ФУНКЦИЙ ПОЛИНОМОМ ЛАГРАНЖА

Цель: формирование навыков интерполирования таблично заданных функций полиномом Лагранжа; оценка погрешности полинома Лагранжа

Задание на лабораторную работу:

1) из каждого блока (1 и 2) выполнить по одному заданию согласно своему варианту (выдается преподавателем);
2) оформить отчет, включающий описание задания, краткое описание метода, скриншот выполнения задания в Excel, описание и анализ результата (что получено, для чего, какой можно сделать вывод).

Варианты заданий

Блок 1. Пусть функция $f(x)$ задана в виде таблицы ее значений (узлы интерполяции). С помощью полинома Лагранжа найти значение функции $f(x)$ в заданной точке x:

1) x=1.52;

x	1.50	1.54	1.56	1.60	1.63	1.70
$f(x)$	3.873	3.924	3.950	4.000	4.037	4.123

2) x=2.22;

x	2.0	2.3	2.5	3.0	3.5	3.8	4.0
$f(x)$	5.848	6.127	6.300	6.694	7.047	7.243	7.368

3) $x=1.68$;

x	1.60	1.64	1.66	1.70	1.73	1.80
$f(x)$	2.873	2.924	2.950	3.000	3.037	3.123

4) $x=3.0$

x	-3.2	-0.8	0.4	2.8	4.0	6.4
$f(x)$	-1.9449	-0.6126	0.3097	1.8068	2.0913	1.4673

5) $x=2.78$

x	1.2	1.9	3.3	4.7	5.4	6.8
$f(x)$	0.3486	1.0537	1.7844	2.2103	2.3712	2.6322

6) $x=5.21$;

x	1.3	2.1	3.7	4.5	6.1	7.7	8.5
$f(x)$	1.777	4.5634	13.8436	20.3952	37.3397	59.4051	72.3593

7) $x=27$;

x	14	17	31	35
$f(x)$	-1.9449	-0.6126	0.3097	1.8068

8) $x=2.22$;

x	2.0	2.3	2.5	3.0	3.5	3.8	4.0
$f(x)$	5.848	6.127	6.300	6.694	7.047	7.243	7.368

9) $x=102$;

x	93.0	96.2	100.0	104.2	108.7
$f(x)$	-1.9449	-0.6126	0.3097	1.8068	2.0913

10) $x=0.5$;

x	-2	-1	0	1	2
$f(x)$	-1.9449	-0.6126	0.3097	1.8068	2.0913

11) $x=5$;

x	0	2	3	6	7	9
$f(x)$	658503	704969	729000	804357	830584	884736

Блок 2. *Теперь пусть функция задана аналитически. Но мы хотим проверить, насколько хорошо многочленом Лагранжа ее можно приблизить. Для этого*

по табличным точкам этой функции построим многочлен Лагранжа $L_n(x)$, затем вычислим его значение в заданной точке x и сравним с тем, чему равна была бы сама заданная функция ***в этой точке x.***

1) Зная значения функции $\sin x$ при $x = 0, \frac{\pi}{6}, \frac{\pi}{3}, \frac{\pi}{2}$, вычислить приближенно с помощью полинома Лагранжа значение $\sin\frac{\pi}{5}$ и оценить погрешность (*таким образом, здесь* $x = \frac{\pi}{5}$)
2) Зная значения функции $cos\, x$ при $x = 0, \frac{\pi}{6}, \frac{\pi}{3}, \frac{\pi}{2}$, вычислить приближенно с помощью полинома Лагранжа значение $cos\frac{\pi}{5}$ и оценить погрешность.
3) Функция $y = \sqrt[3]{x}$ задана таблично. Найти с помощью полинома Лагранжа значение $\sqrt[3]{1{,}15}$ и оценить погрешность.

x	1.0	1.1	1.3	1.5
y	1.000	1.032	1.091	1.145

4) Зная значения функции $y = \sqrt{x}$ при $x = 49, 64, 81, 100,$ найти с помощью полинома Лагранжа $\sqrt{77}$ и оценить погрешность.

5) Функция $y = e^x$ задана таблично. Найти с помощью полинома Лагранжа значение $e^{0{,}506}$ и оценить погрешность.

x	0.50	0.51	0.52
y	1.6487	1.6653	1.6852

6) Функция $y = \frac{1}{x}$ задана таблично. Найти с помощью полинома Лагранжа значение $\frac{1}{2{,}726}$ и оценить погрешность.

x	2.70	2.72	2.74
y	1.6487	1.6653	1.6852

7) Зная значения функции $cos\, x$ при $x = \frac{\pi}{6}, \frac{\pi}{4}, \frac{\pi}{3}, \frac{\pi}{2}$, вычислить приближенно с помощью полинома Лагранжа значение $cos\frac{\pi}{5}$ и оценить погрешность.

Краткие теоретические сведения

Пусть значения некоторой функции $f(x)$ заданы в виде таблицы:

x	x_0	x_1	x_2	…	x_n
y	y_0	y_1	y_2	⋯	y_n

Необходимо найти значение функции для такого значения аргумента, которое принадлежит отрезку $[x_0, x_n]$, но не совпадает ни с одним из значений $x_i, i = 0, \dots, n$. Очевидный способ решения поставленной задачи – вычислить значение $f(x)$ с помощью аналитического выражения функции $f(x)$. Но это возможно только в том случае, когда аналитическое выражение функции $f(x)$ пригодно для вычислений.

Часто аналитическое выражение функции $f(x)$ вообще неизвестно. В таких случаях применяют особый способ – **построение** по заданным значениям x_i, y_i $(i = \overline{0,n})$ **интерполирующей функции $F(x)$,** которая в некотором смысле близка к $f(x)$.

Пусть функция $f(x)$ определена на отрезке $[a, b]$,

$x_0, x_1, \dots, x_n -$ различные точки этого отрезка (узлы интерполяции). Пусть известны значения данной функции в узлах $x_i, i = 0, \dots, n$: $f(x_i) = y_i$.

Задача интерполирования функции $f(x)$ на $[a, b]$ заключается в том, чтобы **подобрать полином с вещественными коэффициентами степени не выше n вида:**

$$P_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n,$$

принимающий в узлах x_i, те же значения y_i, что и функция $f(x)$, т.е.

$$P_n(x_i) = a_0 + a_1 x_i + a_2 {x_i}^2 + \cdots + a_n {x_i}^n = f(x_i), \qquad i = 0,1,2, \dots, n.$$

Построенный таким образом полином $P_n(x)$ называют интерполяционным, а узлы $x_0, x_1, \dots, x_n -$ узлами интерполирования.

Рассмотрим узел x_i.

Коэффициент Лагранжа $L_n^i(x)$, соответствующий этому узлу, представляет собой многочлен и определяются следующими условиями:

а) степень его равна n;

б) он обращается в единицу при $x = x_i$ и в 0 – во всех других узлах x_j $(j \neq k)$.

$$L_n^i(x) = \frac{(x - x_0) \cdot (x - x_1) \cdot \dots \cdot (x - x_{i-1}) \cdot (x - x_{i+1}) \cdot \dots (x - x_n)}{(x_i - x_0) \cdot (x_i - x_1) \cdot \dots \cdot (x_i - x_{i-1}) \cdot (x_i - x_{i+1}) \cdot \dots \cdot (x_i - x_n)}.$$

Число узлов x_i $(i \neq k)$ равно n (а это и есть различные корни полинома).

Итак, полином Лагранжа, имеющий вид

$$\boldsymbol{L_n(x) = \sum_{i=0}^{n} f(x_i) \cdot L_n^i(x) = f(x_0) \cdot L_n^0(x) + f(x_1) \cdot L_n^1(x) + \cdots + f(x_n) \cdot L_n^n(x)}$$

удовлетворяет условиям: $L_n(x_i) = f(x_i),\ i = 0,1,2,\dots,n$ и имеет степень n.

Пример. Для функции $\boldsymbol{y = \sin(\pi x)}$ построить интерполяционный полином Лагранжа, выбрав узлы $x_0 = 0,\ x_1 = \frac{1}{6},\ x_2 = \frac{1}{2}$.

Решение. Вычислим соответствующие значения функции в узлах:

$$y_0 = 0, \quad y_1 = \sin\frac{\pi}{6} = \frac{1}{2}, \quad y_2 = \sin\frac{\pi}{2} = 1.$$

Для удобства запишем данные в таблицу:

x_i	0	1/6	1/2
y_i	0	1/2	1

$$\boldsymbol{L_2^0(x)} = \frac{(x - x_1)\cdot(x - x_2)}{(x_0 - x_1)\cdot(x_0 - x_2)} = \frac{(x - 1/6)\cdot(x - 1/2)}{(0 - 1/6)\cdot(0 - 1/2)} = 12\cdot(x - 1/6)\cdot(x - 1/2);$$

$$\boldsymbol{L_2^1(x)} = \frac{(x - x_0)\cdot(x - x_2)}{(x_1 - x_0)\cdot(x_1 - x_2)} = \frac{(x - 0)\cdot(x - 1/2)}{(1/6 - 0)\cdot(1/6 - 1/2)} = -1/18\cdot x\cdot(x - 1/2)$$

$$\boldsymbol{L_2^2(x)} = \frac{(x - x_0)\cdot(x - x_1)}{(x_2 - x_0)\cdot(x_2 - x_1)} = \frac{(x - 0)\cdot(x - 1/6)}{(1/2 - 0)\cdot(1/2 - 1/6)} = 6\cdot(x^2 - 1/6\cdot x)$$

$$\boldsymbol{L_2(x) = \sum_{i=0}^{2} f(x_i) \cdot L_2^i(x) = y_0 \cdot L_2^0(x) + y_1 \cdot L_2^1(x) + y_2 \cdot L_2^2(x) =}$$

$$= \mathbf{0}\cdot 12\cdot\left(x - \frac{1}{6}\right)\cdot\left(x - \frac{1}{2}\right) + \frac{1}{2} * \left(-\frac{1}{18}\cdot x\cdot\left(x - \frac{1}{2}\right)\right) +$$

$$+1\cdot 6\left(x^2 - \frac{1}{6}\cdot x\right) = -\frac{1}{36}\cdot\left(x^2 - \frac{1}{2}x\right) + 6x^2 - x = \cdots$$

Получаем многочлен 2-й степени.

Формула Лагранжа удобна для использования в задачах интерполирования многих функций в точке x, поскольку все значения множителей $L_n^i(x)$ можно вычислить сразу для всех функций. Однако

формула Лагранжа имеет существенный недостаток: иногда заданное число узлов недостаточно для достижения заданной точности. Тогда к заданным узлам добавляют ещё один или несколько и проводят вычисления заново.

Интерполяционный полином Лагранжа принимает в узлах $x_0, x_1, \dots, x_n$ значения $y_0, y_1, \dots, y_n$. Возникает вопрос: насколько построенный полином близок к функции $f(x)$ в других точках, т.е. насколько велик остаточный член $R_n(x) = |f(x) - L_n(x)|$?

Для абсолютной погрешности интерполяционной формулы Лагранжа можно получить следующую оценку:

$$R_n(x) \le \frac{M_{n+1}}{(n+1)!}\left|(x-x_0)(x-x_1)\cdot\ldots\cdot(x-x_n)\right|.$$

где

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

Пример. **С какой точностью** с помощью интерполяционной формулы Лагранжа можно вычислить $\sqrt{115}$ для функции $y = \sqrt{x}$, если выбрать узлы интерполирования $x_0 = 100,\ x_1 = 121,\ x_2 = 144$?

Решение. По условию имеем три узла, следовательно, $n + 1 = 3$, откуда $n = 2$. Тогда $y' = \frac{1}{2\sqrt{x}},\ y'' = -\frac{1}{4\sqrt{x^3}},\ y''' = \frac{3}{8}x^{-\frac{5}{2}}$.

Найдем максимум производной:

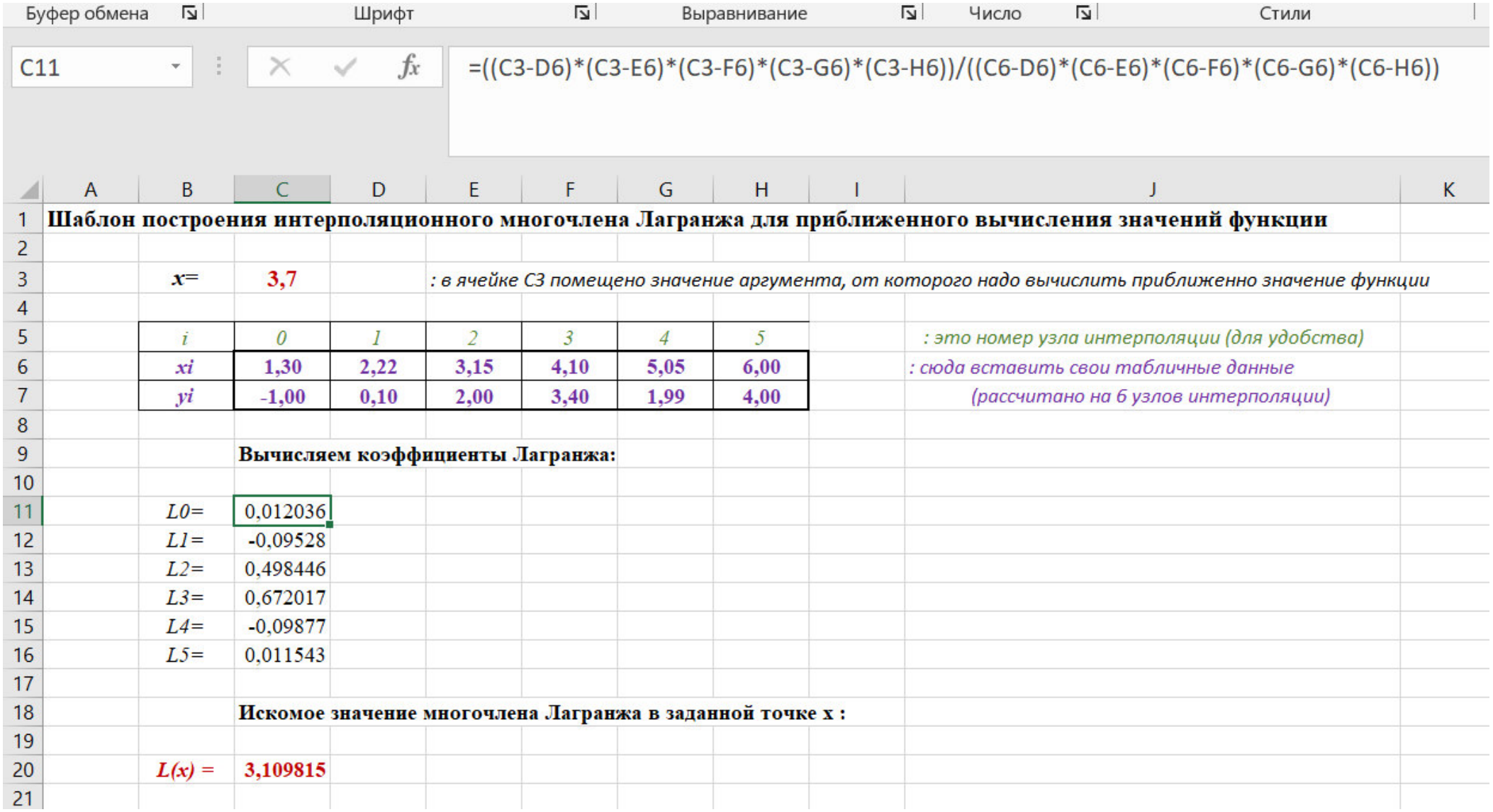
$M_3 = \max|y'''| = \frac{3}{8}\cdot 10^{-5}$ при $100 \le x \le 144$.

Оцениваем погрешность $R_2(x)$:

$$|R_2(115)| \le \frac{3}{8}\cdot 10^{-5}\cdot\frac{1}{3!}\cdot|(115-100)(115-112)(115-144)| =$$

$$= \frac{3}{8\cdot 6}\cdot 10^{-5}\cdot 15\cdot 3\cdot 29 = \frac{1}{16}\cdot 10^{-5}\cdot 15\cdot 6\cdot 29 \approx \mathbf{1.6\cdot 10^{-3}}.$$

Образец оформления задания блока 1 (рекомендуемый)



	A	B	C	D	E	F	G	H	I	J	K
1	**Шаблон построения интерполяционного многочлена Лагранжа для приближенного вычисления значений функции**										
2											
3		***x*=**	**3,7**		*: в ячейке C3 помещено значение аргумента, от которого надо вычислить приближенно значение функции*						
4											
5		*i*	*0*	*1*	*2*	*3*	*4*	*5*		*: это номер узла интерполяции (для удобства)*	
6		***xi***	**1,30**	**2,22**	**3,15**	**4,10**	**5,05**	**6,00**		*: сюда вставить свои табличные данные*	
7		***yi***	**-1,00**	**0,10**	**2,00**	**3,40**	**1,99**	**4,00**		*(рассчитано на 6 узлов интерполяции)*	
8											
9			**Вычисляем коэффициенты Лагранжа:**								
10											
11		*L0=*	0,012036								
12		*L1=*	-0,09528								
13		*L2=*	0,498446								
14		*L3=*	0,672017								
15		*L4=*	-0,09877								
16		*L5=*	0,011543								
17											
18			**Искомое значение многочлена Лагранжа в заданной точке x :**								
19											
20		***L(x) =***	**3,109815**								
21											

C20 | =C11*C7+C12*D7+C13*E7+C14*F7+C15*G7+C16*H7

Шаблон построения интерполяционного многочлена Лагранжа для приближенного вычисления значений функции

x= **3,7** : *в ячейке C3 помещено значение аргумента, от которого надо вычислить приближенно значение функции*

i	*0*	*1*	*2*	*3*	*4*	*5*	: это номер узла интерполяции (для удобства)
xi	**1,30**	**2,22**	**3,15**	**4,10**	**5,05**	**6,00**	: сюда вставить свои табличные данные
yi	**-1,00**	**0,10**	**2,00**	**3,40**	**1,99**	**4,00**	(рассчитано на 6 узлов интерполяции)

Вычисляем коэффициенты Лагранжа:

L0=	0,012036
L1=	-0,09528
L2=	0,498446
L3=	0,672017
L4=	-0,09877
L5=	0,011543

Искомое значение многочлена Лагранжа в заданной точке x :

L(x) = **3,109815**

ЛАБОРАТОРНАЯ РАБОТА №3

ЧИСЛЕННЫЕ МЕТОДЫ РЕШЕНИЯ НЕЛИНЕЙНЫХ УРАВНЕНИЙ: МЕТОД ДИХОТОМИИ (ПОЛОВИННОГО ДЕЛЕНИЯ) И МЕТОД НЬЮТОНА (МЕТОД КАСАТЕЛЬНЫХ)

Цель работы: изучить численные методы решения нелинейных уравнений с одним неизвестным; научиться программировать алгоритмы итерационных методов уточнения корней – метод дихотомии и метод Ньютона; ***сравнить скорость сходимости различных методов***.

Описание методов см. в приложении 1 и лекциях.

Задание (порядок выполнения):

1. Графическим методом определить наличие, приблизительное расположение и количество корней заданного уравнения (вариант назначается преподавателем).

Задать границы отрезка отделения корня/корней (интервала поиска): взять отрезок длиной 1 (т.е. если графически видно, что корень уравнения находится между 1 и 2, то начальный отрезок взять [1; 2]) и записать в виде таблицы:

корень	Интервал примерного расположе
№1	$[a_1, b_1]$
№2 (при наличии)	$[a_2, b_2]$

<u>Замечание:</u> *в некоторых вариантах уравнение имеет ровно 1 корень, в некоторых – 2.* ***<u>Длину</u>*** *отрезка изоляции рекомендуется брать от 0,5 до 1.*

2. Уточнить корни уравнений по методу половинного деления (дихотомии). Для каждого корня: в качестве начального отрезка использовать найденные в п.1 интервалы; **определить приближенные значения корня с разной точностью ε (например, $10^{-3}, 10^{-5}, 10^{-8}$).**

Результаты представьте в виде таблицы:

Корень	Приближенное значение	Точность	**Количество итераций**
№1		10^{-3}	

		10^{-5}	
		10^{-8}	
№2 (при наличии)		10^{-3}	
		10^{-5}	
		10^{-8}	

Замечание. ***Напомню, что вычисление приближенного числа с точностью 10^{-k} означает, что необходимо сохранить верной значащую цифру, стоящую на k -м разряде после запятой.***

3. Найти корень по методу Ньютона (методу касательных). В качестве начального приближения к корню x_0 выберите середину отрезка изоляции корня. Для правильного выбора начального приближения используйте критерий $f(x_0) \cdot f''(x_0) > 0$.

Определите приближенные значения корней с разной точностью ε (например, $10^{-3}, 10^{-5}, 10^{-8}$). Результаты представьте в виде таблицы:

корень	Начальное приближение	Приближенное значение	точность	Количество итераций
№1			10^{-3}	
			10^{-5}	
			10^{-8}	
№2 (при наличии)			10^{-3}	
			10^{-5}	
			10^{-8}	

5. Составить отчет по лабораторной работе. Сформулировать выводы *о скорости* сходимости различных методов (сколько итераций необходимо сделать каждым методом для достижения указанных в таблицах точностей).

Варианты заданий:

1. $3x + cos\, x + 1 = 0;$	2. $3x + 2\, cos\, x + 1{,}5 = 0;$
3. $x^4 + 5x - 7 = 0$	4. $3\, sin\, x = x^2 + 1;$
5. $x^4 + x^3 - 17x^2 - 45x = 0$	6. $2x^2 - 2\, cos\, 2\, x = 1;$
7. $x^4 + 5x^3 + 7x^2 - 1 = 0$	8. $2^x - 2\, cos\, x = 1;$
9. $x^3 - 2x + 3 = 0$	10. $ln\, x + 2x^2 - x = 0;$
11. $x^4 + 5x - 7 = 0$	12. $e^x + 2x^2 - 2 = 0;$
13. $ln\, x + 2x - 3 = 0;$	14. $ln\, x + 3x^2 - 3 = 0;$
15. $arctgx - x^2 + 1 = 0;$	16. $arctg2x + x^2 - 1 = 0;$

17.	$(x+1)^2 - sin\, x = 3.$	18.	$(x+1)^2 - sin\, 2x = 1.$
19.	$\ln 2x - 4 + x = 0.$	20.	$\ln 3x + 3 + x = 0.$
21.	$x^3 - 2x - 7 = 0$	22.	$x^3 - 2x^2 - 5 = 0$
23.	$x^4 + 3x - 20 = 0$	24.	$x^4 + x - 10 = 0$

Приложение 1.

ПРИБЛИЖЕННОЕ РЕШЕНИЕ УРАВНЕНИЙ

Существуют аналитические и численные методы нахождения корней уравнения. Аналитические методы позволяют получить алгебраическое выражение для вычисления корня, что бывает невозможно для ряда уравнений, которые называются трансцендентными. Численные методы позволяют находить сами корни для любых уравнений с заданной точностью, но они часто требуют *большого объёма вычислений*. Развитие вычислительной техники позволило в настоящее время широко использовать численные методы.

Все численные методы предполагают определённый алгоритм решения поставленной задачи. Обычно решение сводится к выбору оптимального набора дискретных точек, которые ведут к решению. Чаще всего все численные методы реализуются через итерационные процедуры, которые выполняются до тех пор, пока не достигнута заданная точность решения.

Все эти методы ***не позволяют*** получить ***абсолютно точное*** решение, но всегда можно задаться *какой-либо точностью* решения.

Первое, что надо сделать для применения приближенных (численных) методов – определить количество корней заданного уравнения и отделить каждый корень (т.е. найти примерное место его нахождения на числовой прямой).

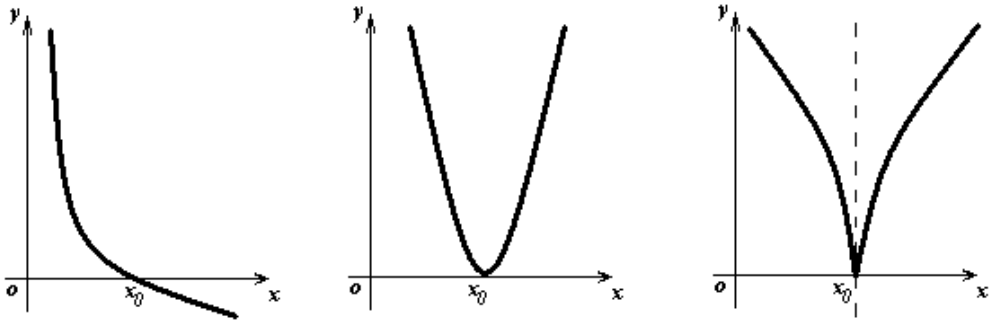
§1. Отделение корней уравнения

Корнем уравнения

$$f(x) = 0 \qquad (1)$$

называется такое значение $x = x_0$, при котором уравнение (1) превращается в верное тождество, т.е. выполняется $f(x_0) = 0$.

Корень уравнения ***геометрически*** представляет собой ***абсциссу точки пересечения***, касания или другой общей точки графика функции $y = f(x)$ и оси OX



Отделить корень уравнения – значит **найти такой конечный промежуток**, внутри которого имеется ***единственный корень*** данного уравнения.

Для *отделения корня* (или *нахождения отрезка изоляции корня*) можно применить графический метод.

Графический метод отделения корней

Отделение корней уравнения (1) можно выполнить графически, построив график функции $y = f(x)$, по которому можно судить о том, в каких промежутках находится точка пересечения его с осью *OX*.

В некоторых случаях целесообразно представить уравнение $f(x) = 0$ в виде:

$$f_1(x) = f_2(x) \qquad \textbf{(2)}$$

с таким расчетом, чтобы графики функций $y = f_1(x)$ и $y = f_2(x)$ строились по возможности проще. Корень уравнения (1) представляет собой абсциссу точки пересечения графиков функций $y = f_1(x)$ и $y = f_2(x)$.

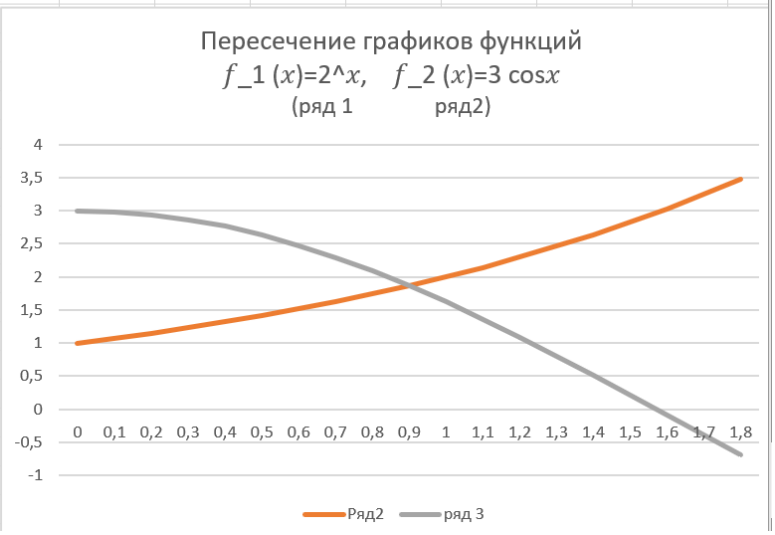
Например, для решения уравнения

$$2^x - 3\cos x = 0 \qquad (\text{здесь} \quad f(x) = 2^x - 3\cos x)$$

можно представить уравнение в виде $2^x = 3\cos x$ и, построив отдельно графики функций $f_1(x) = 2^x$, $f_2(x) = 3\cos x$, найти точки их пересечения.

Точка пересечения этих графиков – примерное значение корня (точное мы не знаем, так как корень этого уравнения – иррациональное число). Поэтому выделяем небольшой отрезок $[a; b]$, где очевидно лежит наш корень, и дальше применяем какой-либо численный метод, чтобы уточнить значение корня (или *найти приближение к корню с заданной точностью*).

x	y=2^x	x	y=3*cos(x)
0	1	0	3
0,1	1,071773	0,1	2,9850125
0,2	1,148698	0,2	2,94019973
0,3	1,231144	0,3	2,86600947
0,4	1,319508	0,4	2,76318298
0,5	1,414214	0,5	2,63274769
0,6	1,515717	0,6	2,47600684
0,7	1,624505	0,7	2,29452656
0,8	1,741101	0,8	2,09012013
0,9	1,866066	0,9	1,8648299
1	2	1	1,62090692
1,1	2,143547	1,1	1,36078836
1,2	2,297397	1,2	1,08707326
1,3	2,462289	1,3	0,80249649
1,4	2,639016	1,4	0,50990143
1,5	2,828427	1,5	0,21221161
1,6	3,031433	1,6	-0,08759857
1,7	3,24901	1,7	-0,38653348
1,8	3,482202	1,8	-0,68160628



Рассмотрим 2 численных метода решения нелинейного уравнения: метод половинного деления (или дихотомии) и метод Ньютона (или метод касательных).

§2. Метод половинного деления (дихотомия)

Метод непосредственно следует из аналитического способа отделения корней. Пусть для уравнения $f(x) = 0$ найден первичный отрезок $[a; b]$ изоляции корня. Далее приближение к корню производится интуитивно понятной процедурой: делим отрезок пополам; определяем, в какой половине отрезка находится корень, и работаем уже только с этой половиной, т.е. все то же самое выполняем с «ополовиненным» отрезком, и т.д.; после каждого шага длина отрезка, в котором находится корень, уменьшается в 2 раза.

Таким образом, цель – на каждой итерации вдвое уменьшать отрезок, где находится корень, тем самым приближаясь к корню.

Замечание. На отрезке $[a; b]$ есть корень означает, что график функции $f(x)$ **пересекает** ось между точками a и b *(т.е. функция принимает значение, равное 0)*, значит, значения функции в точках a и b **– разных знаков**, а это математически означает $\boldsymbol{f(a) \cdot f(b) < 0}$.

Итак, вычислим середину отрезка $c = \frac{a+b}{2}$.

Если *случайно* окажется, что $f(c) = 0$, то x_2 является корнем уравнения $f(x) = 0$.

Если же $f(c) \neq 0$, то **из двух половин** $[a,\ c]$ **и** $[c,\ b]$ первичного отрезка **выберем** для дальнейшего деления пополам ту, на концах которой функция $y = f(x)$ принимает **значения противоположных знаков**.

Значит, нам надо вычислить $f(a), f(b), f(c)$ и:

- Выбрать отрезок $[a,\ c]$, если $\boldsymbol{f(a) \cdot f(c) < 0}$;
- Выбрать отрезок $[c,\ b]$, если $\boldsymbol{f(c) \cdot f(b) < 0}$.

Выбранный отрезок снова разделим пополам и выберем половину отрезка с противоположными знаками $f(x)$ на концах, и т. д.

Критерий достижения требуемой точности (критерий обрыва счета): если корень надо вычислить **с точностью ε**, то **деление пополам следует продолжать до тех пор, пока длина очередного отрезка не станет меньше 2·ε**; тогда середина этого отрезка даст значение корня с точностью **ε**.

Итак, как только $|\boldsymbol{b - a}| < \boldsymbol{\varepsilon} \Rightarrow$

деление отрезка прекращаем,

наш корень $\boldsymbol{c = \frac{a + b}{2}}$ найден с точностью $\boldsymbol{\varepsilon}$.

!!! В таблице результатов надо вписать значение найденного с определенной точностью корня (для нескольких ε, как указано) и количество итераций, которые вы сделали для получения этого значения.

§3. Метод касательных (Ньютона)

Пусть действительный корень уравнения $f(x) = 0$ изолирован на отрезке $[a, b]$. Для выбора начального приближения учитываем следующий алгоритм (следствие теоремы – см. лекции)

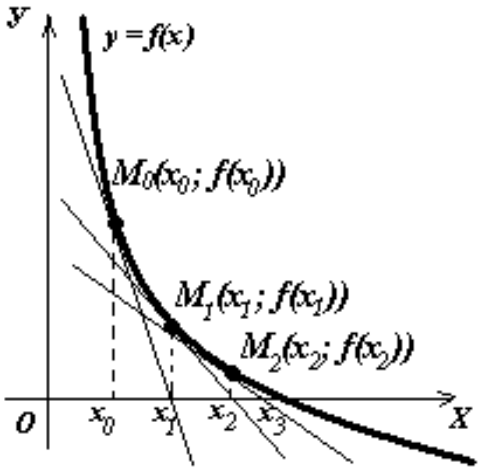


рис 1.4

Алгоритм выбора x_0:

1. Убедиться, что $f'(x)$ не меняет знак на отрезке изоляции корня $[a, b]$ (т.е. функция монотонна на $[a, b]$) и не равна нулю.
2. Убедиться, что вторая производная $f''(x)$ не меняет знак.
3. Вычислить значения $f(a), f(b)$ и знак $f''(x)$ на отрезке.
4. Выбор x_0:

✓ если $f(a) \cdot f''(x) > 0$ (для всего отрезка, например, в точке a), то $x_0 = a$.

✓ если $f(b) \cdot f''(x) > 0$, то $x_0 = b$.

Итак, проверим, что на отрезке $[a, b]$ функция монотонна и сохраняет определенное направление выпуклости, т.е. ее первая и вторая производные не меняют знак на $[a, b]$. Если это выполнено, то выбираем начальную точку $x_0 \in [a, b]$ (зачастую это может быть произвольная точка отрезка изоляции, если условия выполнены всюду на $[a, b]$).

Проведем в точке $M_0(x_0, f(x_0))$ касательную к кривой $y = f(x)$.

За приближенное значение корня примем *абсциссу точки пересечения* этой касательной с осью *OX*. Это приближенное значение корня найдется по формуле:

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

Применив этот метод вторично в точке $M_1(x_1, f(x_1))$, получим следующее приближение: $x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$

и т.д.

Полученная таким образом последовательность $x_0, x_1, x_2, \ldots$ имеет своим пределом **искомый корень.**

Таким образом, чтобы найти корень уравнения $f(x) = 0$ по методу Ньютона, надо:

1) Выбрать начальное приближение – любую точку отрезка изоляции $x_0 \in [a, b]$, проверив предварительно выполнение условий сохранения знаков $f'(x)$ и $f''(x)$ на отрезке $[a, b]$.
2) Для дальнейшего уточнения корня выполнить итерации согласно формуле:

$$\boldsymbol{x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}} \qquad (*)$$

3) Выполнять итерации до тех пор, пока не будет достигнута заданная степень точности (см. в лекции оценку погрешности):
   - ✓ как только $\left|\frac{f(x_n)}{m}\right| < \varepsilon$ (где $m = \min\limits_{[a,b]} |f'(x)|$), то итерации можно прекратить, решение с требуемой точностью ε получено;

- ✓ для проверки можно ориентироваться на получаемые в результате выполнения итераций значения $x_0, x_1, x_2, \ldots$: как только необходимая цифра в разряде не станет постоянной, вычисления можно прекратить)

Пример.

Методом касательных (Ньютона) найти положительный корень уравнения $x^4 - 2x - 4 = 0$ **с точностью до 0,01**.

Решение. Здесь $f(x) = x^4 - 2x - 4$,

$f'(x) = 4x^3 - 2$, $f''(x) = 12x^2$.

Графическим методом выберем отрезок изоляции корня $[1, 2]$ и возьмем начальное приближение, например, 1,7.

Проверка условий применимости метода:

так как $f(x)$ и $f''(x)$ при $x_0 = 1{,}7$ имеют один и тот же знак, а именно: $f(1{,}7) = 0{,}952 > 0$ и $f''(1{,}7) = 34{,}68 > 0$,

то применяем формулу (*) при $n = 0$:

вычисляем $f'(1{,}7) = 17{,}652$,

находим $\boldsymbol{x_1} = 1{,}7 - \frac{0{,}952}{17{,}652} = \mathbf{1{,}646}$.

Применяя второй раз формулу (*) при $n = 1$, получим:

$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$, где $f(x_1) = f(1{,}646) = 0{,}048$, $f'(1{,}646) = 15{,}838$.

$\boldsymbol{x_2} = 1{,}646 - \frac{0{,}048}{15{,}838} = \mathbf{1{,}643}$.

Аналогично получим третье приближение:

$x_3 = x_2 - \frac{f(x_2)}{f'(x_2)}$,

$f(1{,}643) = 0{,}004$, $f'(1{,}643) = 15{,}740$,

следовательно, $\boldsymbol{x_3} = 1{,}643 - \frac{0{,}004}{15{,}740} = \mathbf{1{,}6427}$.

Искомый корень **с точностью до 0,01** (по правилам округления) **равен 1,64.**

Замечание. Ясно, что данные вычисления удобно проводить в Excel, где по находимым приближениям можно оценивать, как меняются цифры после

запятой при дальнейших итерациях. Если нужна степень точности 0,001, то необходимо продолжить итерации до тех пор, пока третья цифра не станет неизменной после запятой.

Вообще критерий обрыва счета по методу Ньютона при достижении заданной степени точности таков: как только

$|x_{n+1} - x_n| < \varepsilon,$ вычисления можно прекратить.

В ответе оставить столько цифр после запятой, сколько соответствуют заданной степени точности.

В таблице результатов надо вписать значение найденного с определенной точностью корня (для нескольких ε, как указано) и количество итераций, которые вы сделали для получения этого значения.

Лабораторная работа № 4

ЧИСЛЕННЫЕ МЕТОДЫ РЕШЕНИЯ НЕЛИНЕЙНЫХ УРАВНЕНИЙ МЕТОДОМ ХОРД И МЕТОДОМ ИТЕРАЦИЙ

Цель работы: изучить численные методы решения нелинейных уравнений с одним неизвестным (см. соответствующие лекции и приложение 1); научиться программировать алгоритмы итерационных методов уточнения корней; сравнить скорость сходимости различных методов.

Задание:

1. Графическим методом определить наличие, приблизительное расположение и количество корней заданных уравнений.

2. Задать границы отрезка изоляции корня: $[\boldsymbol{a}, \boldsymbol{b}]$.

3. Найти корень **по методу итераций** (см. теоретические сведения).

Приведите заданное уравнение к виду, удобному для итераций:

$x = \varphi(x)$ таким образом, чтобы $|\varphi'(x)| \leq r$ <1 выполнялось всюду на отрезке $[a, b]$, содержащем единственный корень x_0.

В качестве начального приближения к корню x_0 выберите одну из границ интервала, найденного в ходе выполнения второго задания.

Определите приближенные значения корней с разной точностью ε (например, $10^{-3}, 10^{-5}, 10^{-8}$). Результаты представьте в виде таблицы:

корень	Начальное приближение	Приближенное значение	точность	Количество итераций
1			10^{-3}	
			10^{-5}	
			10^{-8}	
2 (при наличии)			10^{-3}	
			10^{-5}	
			10^{-8}	

4. Уточнить корни уравнения **по методу хорд** (см. теоретические сведения; в качестве начального отрезка использовать интервал, найденный при выполнении второго задания). Определить приближенные значения корней с разной точностью ε (например, $10^{-3}, 10^{-5}, 10^{-8}$). Результаты представьте в виде таблицы:

корень	Приближенное значение	точность	Количество итераций
1		10^{-3}	
		10^{-5}	
		10^{-8}	
2 (при наличии)		10^{-3}	
		10^{-5}	
		10^{-8}	

5. Сформулировать выводы **о скорости сходимости различных методов** (сколько итераций требуется сделать для достижения одной и той же точности разными методами).

Варианты заданий:

1. $3x + cos\, x + 1 = 0;$
2. $3\, sin\, x = x^2 - 2;$
3. $x^2 - cos\, 2\, x = -2;$
4. $2^x - 2\, cos\, x = 0;$
5. $ln\, x + 2(x - 3) = 0;$
6. $e^x + x^2 - 2 = 0;$
7. $ln\, x + 2x - 3 = 0;$

8. $arctgx - x^2 + 1 = 0;$
9. $(x+1)^2 - sin\, x = 3.$
10. $\ln 2x - 4 + x = 0.$
11. $x^3 - 2x - 7 = 0$
12. $x^4 + 3x - 20 = 0$
13. $x^4 - 3x + 1 = 0$
14. $x^4 + 5x - 7 = 0$

Приложение 1

МЕТОД ИТЕРАЦИЙ

Если каким-нибудь способом получено приближенное значение x_0 корня уравнения (например, это середина отрезка изоляции искомого корня), то *уточнение приближения* можно осуществить **методом итераций** (*методом последовательных приближений*).

Идея метода простой итерации состоит в том, чтобы уравнение $f(x) = 0$ привести к эквивалентному уравнению $\boldsymbol{x = \varphi(x)}$ так, чтобы отображение $\boldsymbol{\varphi(x)}$ было сжимающим.

Итак, пусть задано уравнение $\boldsymbol{f(x) = 0}$.

Пусть $[a; b]$ –отрезок изоляции корня.

Представим уравнение в виде $\boldsymbol{x = \varphi(x)}$, где

$$\boldsymbol{|\varphi'(x)| \le r < 1} \qquad \textbf{(*)}$$

всюду на отрезке $\boldsymbol{[a, b]}$***, содержащем единственный корень*** $\boldsymbol{\xi}$.

Выбрав любое начальное значение $x_0 \in [a; b]$, можно построить последовательность уточненных значений корня:

$$x_1 = \varphi(x_0),\ .,\ x_3 = \varphi(x_2), \ldots\ldots, x_n = \varphi(x_{n-1})\ .$$

Пределом последовательности $x_0, x_1, \ldots, x_n, \ldots$ является единственный корень $\bar{x}$ уравнения $f(x) = 0$ на отрезке $[a, b]$.

Замечание. Напоминаю, что для применения метода итераций можно по-разному приводить исходное уравнение $f(x) = 0$ к виду $x = \varphi(x)$. При этом, *чтобы получить приближение к корню* (т.е. **чтобы последовательность** получаемых по алгоритму значений $\boldsymbol{x_0, x_1, \ldots, x_n, \ldots}$ **имела предел** – искомый корень уравнения), *надо, чтобы выполнялось условие* (*).

Как привести уравнение $f(x) = 0$ к виду $x = \varphi(x)$, удобному для итераций, и обеспечить благоприятное значение q:

1) $f(x) = 0 \Leftrightarrow x + f(x) = x$, обозн. $\varphi(x) = x + f(x)$. При этом *чтобы получить приближение к корню* (т.е. чтобы последовательность получаемых по алгоритму значений $x_0, x_1, \ldots, x_n, \ldots$ имела предел – искомый корень уравнения), *надо, чтобы выполнялось условие* (*). Если это условие не выполняется, то можно попробовать по-другому выразить *x. Либо поступить следующим образом:*

2) Если $\max\limits_{[a,b]}|\varphi'(x)|$ недостаточно мало на $[a,b]$, то преобразуем уравнение:

$$f(x)=0 \Leftrightarrow \lambda\cdot f(x)=0\ (\lambda\neq 0)$$

$$\Leftrightarrow x+\lambda\cdot f(x)=x;$$

$$\varphi(x):=x+\lambda\cdot f(x) \quad \Longrightarrow \quad x=\varphi(x).$$

Для нахождения λ:

- выберем $x_0\in[a,b]$ (например, середину отрезка);
- потребуем выполнения: $\varphi'(x_0)=\lambda\cdot f'(x_0)+1=0\Longrightarrow$

$$\boldsymbol{\lambda=-1/f'(x_0)}$$

Тогда $\boldsymbol{\varphi(x)=x+\lambda\cdot f(x)}$ будет обеспечивать сходящийся итерационный процесс для нахождения корня.

Пример 1.5.

Методом итераций найти приближенное значение корня уравнения

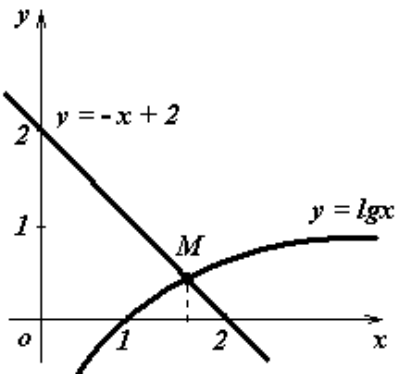
$2-\lg x-x=0$ **с точностью до *0,0001.***

Решение.

Найдем отрезок изоляции действительного корня уравнения. Для этого представим уравнение в виде:

$$lg\,x=-x+2$$

Построим графики функций $y=\lg x$ и $y=-x+2$. Точка M пересечения графиков имеет абсциссу *в промежутке* [1; 2].



Итак, отрезок изоляции - [1; 2].

Возьмем в качестве начального приближения точку $x_0=1$.

Запишем исходное уравнение в виде $x=2-lg\,x$ (и проверим, подойдет ли такой вид представления уравнения для проведения итераций – проверим условие (*)):

$$\varphi(x)=2-lg\,x\,,\quad \varphi'(x)=-\frac{1}{x\cdot ln\,10}$$

Так как $|\varphi'(x)|\le 0{,}42<1$ на всем промежутке [1; 2], следовательно, способ итераций применим.

Коэффициент сжатия $q = \max\limits_{[1,2]}|\varphi'(x)| = 0{,}42.$

Найдем приближения:

$$x_1 = 2 - lg\, x_0 = 2 - lg\, 1 = 2$$

$$x_2 = 2 - lg\, x_1 = 2 - lg\, 2 = 2 - 0{,}3010 = 1{,}6990$$

$$x_3 = 2 - lg\, x_2 = 2 - lg\, 1{,}6990 = 2 - 0{,}2302 = 1{,}7698$$

$$x_4 = 2 - lg\, x_3 = 2 - lg\, 1{,}7698 = 2 - 0{,}2480 = 1{,}7520$$

$$x_5 = 2 - lg\, x_4 = 2 - lg\, 1{,}7520 = 2 - 0{,}2435 = 1{,}7565$$

$$x_6 = 2 - lg\, x_5 = 2 - lg\, 1{,}7565 = 2 - 0{,}2445 = 1{,}7555$$

$$x_7 = 2 - lg\, x_6 = 2 - lg\, 1{,}7555 = 2 - 0{,}2444 = 1{,}7556$$

Критерий останова: $\boldsymbol{\Delta = |\overline{x} - x_n| \le \frac{q}{1-q}|x_n - x_{n-1}| < \varepsilon = 0{,}0001}$

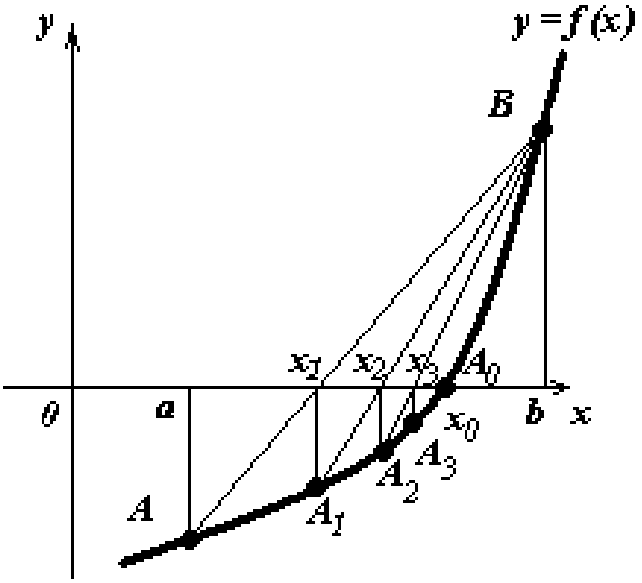
$\boldsymbol{\frac{0{,}42}{1-0{,}42} \cdot |x_7 - x_6| \approx 7{,}2 * 10^{-5} = 0{,}000072 < 0{,}0001.}$

Таким образом, искомый корень с точностью до 0,0001 равен **1,7556.**

МЕТОД ХОРД (ПРАВИЛО ПРОПОРЦИОНАЛЬНЫХ ЧАСТЕЙ).

Метод хорд приближенного решения уравнения

$$f(x) = 0 \qquad \textbf{(1)}$$



имеет следующую геометрическую иллюстрацию: вместо точки пересечения оси OX и графика функции $y = f(x)$, входящей в это уравнение, ***рассматривается точка пересечения данной оси и отрезка прямой***, соединяющей концы дуги графика.

Пусть требуется вычислить действительный корень уравнения (1), изолированный на отрезке $[a, b]$.

Рассмотрим график функции $y = f(x)$. Пусть $f(a) < 0$, а $f(b) > 0$. Точки $A(a, f(a))$ и $B(b, f(b))$ соединим хордой. Найдем точку x_1:

$$\boldsymbol{x_1 = a - \frac{(b-a)f(a)}{f(b)-f(a)}} \qquad \textbf{(2)}$$

Если $f(x_1) < 0$, то за новый, более узкий, интервал изоляции можно взять отрезок $[x_1; b]$. Соединив точки $A_1(x_1, f(x_1))$ и $B(b, f(b))$, получим в точке пересечения хорды с осью второе приближение x_2, которое вычислим по формуле:

$$\boldsymbol{x_2 = x_1 - \frac{(b-x_1)\cdot f(x_1)}{f(b)-f(x_1)}} \qquad \textbf{(3)}$$

и т. д. Последовательность чисел $a, x_1, x_2, \ldots$ стремится к искомому корню x_0; процесс продолжаем, пока не будет достигнута заданная степень точности.

Пример 1.2:

Методом хорд найти положительный корень уравнения

$x^4 - 2x - 4 = 0$ с точностью до 0,01.

Решение. Положительный корень будет находиться в промежутке (1; 1,7), т.к. $f(1) = -5 < 0$, а $f(17) = 0{,}952 > 0$.

Найдем первое приближенное значение корня по формуле (2):

$$\boldsymbol{x_1 = a - \frac{(b-a)f(a)}{f(b)-f(a)}},\ \text{где}\ \boldsymbol{a = 1},\ b = \boldsymbol{1,7}$$

Получим $x_1 = 1 - \frac{(1,7-1)\cdot f(1)}{f(1,7)-f(1)}$=1,588.

Так как $f(1{,}588) = -0{,}817 < 0$, то применяя вторично способ хорд к промежутку (1,588; 1,7), получим:

$x_2 = 1{,}588 - \frac{(1,7-1{,}588)\cdot f(1{,}588)}{f(1,7)-f(1{,}588)}$= 1,639; $f(1{,}639) = -0{,}051 < 0$.

Найдем третье приближенное значение на промежутке (1,639; 1,7),

получим:

$x_3 = 1{,}639 - \frac{(1,7-1{,}639)\cdot f(1{,}639)}{f(1,7)-f(1{,}639)}$= 1,642; $f(1{,}642) = -0{,}016 < 0$.

Найдем четвертое приближенное значение на отрезке (1,642; 1,7),

получим: $x_4 = 1{,}642 - \frac{(1,7-1{,}642)f(1{,}642)}{f(1,7)-f(1{,}642)}$= 1,643; $f(1{,}643) = -0{,}004 > 0$.

Следовательно, искомый корень с точностью до 0,01 равен 1,64.

Лабораторная работа №5

ПРИБЛИЖЕННОЕ ВЫЧИСЛЕНИЕ ОПРЕДЕЛЕННЫХ ИНТЕГРАЛОВ

1. Цель работы

Ознакомление с методами численного интегрирования – методами прямоугольников, трапеций, парабол (Симпсона), с понятием порядка точности (погрешности) численного метода.

2. Порядок выполнения работы

1) Разработать алгоритм и программу вычисления интеграла от функции

$f(x) = \sqrt{x^2 + m}$ (на отрезке $[1, 2]$, где m - номер по списку группы) ***методами средних прямоугольников и трапеций*** с шагами разбиения:

$$a) \quad h = 0{,}1; \quad b) \quad h = 0{,}01.$$

2) Вычислить для каждого метода полученную погрешность.

3) Сравнить результаты, объяснить их.

4) Разработать алгоритм и программу вычисления интеграла от функции

$f(x) = \sqrt{x^2 + m}$ (на отрезке $[1, 2]$, где m - номер по списку группы) ***методом Симпсона*** с шагом разбиения $2h = 0{,}1$. Вычислить полученную погрешность, сравнить результат с предыдущими методами.

5) Оформить отчет. В отчете представить:

- ✓ Постановку задачи;
- ✓ пояснение сути методов;
- ✓ проводимые расчеты;
- ✓ Листинг программы, скриншоты выполнения программы
- ✓ Сделать вывод: объяснение результатов, сравнение полученных погрешностей, сравнение точности метода по сравнению с методами прямоугольников и трапеций при одном и том же количестве точек разбиения.

3. Описание методов

Пусть требуется найти определенный интеграл $\int_a^b f(x)dx$ от непрерывной функции $f(x)$. Если можно найти первообразную $F(x)$ функции $f(x)$, то интеграл вычисляется по формуле Ньютона – Лейбница. Но не всегда первообразная функции выражается через элементарные функции. В этих и других случаях используют приближенные формулы, с помощью которых определенный интеграл находится с любой степенью точности.

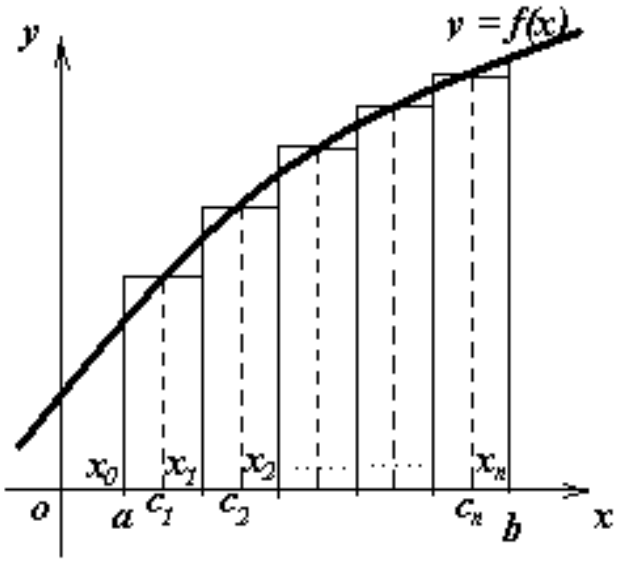
Наиболее употребимые формулы – формула прямоугольников, формула трапеции и формула парабол (Симпсона), основанные на ***геометрическом смысле определенного интеграла.***

§1. Метод прямоугольников

Пусть на отрезке $[a,b]$, где $a < b$, задана непрерывная функция $f(x)$. Требуется вычислить интеграл $\int_a^b f(x)dx$, численно равный площади соответствующей криволинейной трапеции. Разобьем основание этой трапеции на n равных частей длины

$$h = \frac{b-a}{n} = x_i - x_{i-1}.$$

Тогда $x_i = x_0 + ih$.



В точке $c_i = \frac{x_{i-1}+x_i}{2}$ (это **середина** каждого полученного частичного отрезка $[x_{i-1},x_i]$) отметим соответствующую ординату: $\tilde{y}_i = f(c_i)$. Приняв эту ординату за высоту, построим прямоугольник с площадью $S_i = h \cdot \tilde{y}_i$.

Тогда сумма площадей всех n прямоугольников (при достаточно большом n) дает площадь, приближенно равную площади трапеции, т.е.

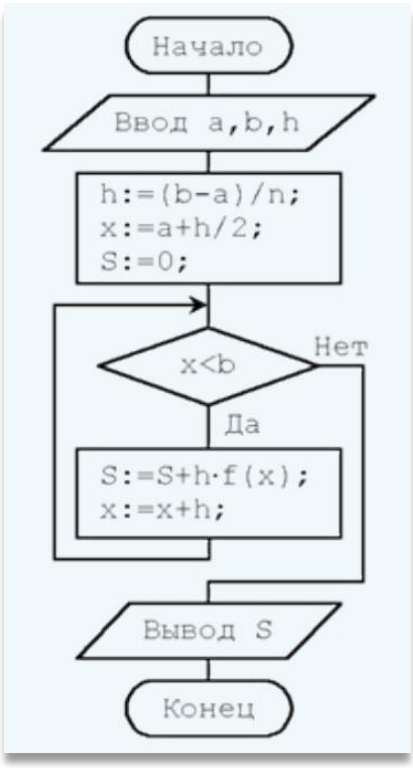
$$\int_a^b f(x)dx = S_1 + S_2 + S_3 + \cdots \ldots \ldots S_n = h \cdot (\tilde{y}_1 + \tilde{y}_2 + \tilde{y}_3 + \cdots \ldots . \tilde{y}_n) =$$

$$= h \cdot \sum_{i=1}^{n} \tilde{y}_i = h \cdot \sum_{i=1}^{n} f(C_i)$$

т.е.

$$\int_a^b f(x)dx \approx \frac{b-a}{n} \cdot \sum_{i=1}^{n} f\left(\frac{x_{i-1}+x_i}{2}\right) \qquad (1)$$

формула средних прямоугольников



Абсолютная погрешность этого метода определяется неравенством:

$$|R_n(f)| \le \frac{M_2 \cdot h^3 \cdot n}{24}, \qquad (2)$$

где $M_2 = \max_{x \in [a,b]} |f''(x)|$ (3)

Пример: Вычислить интеграл $\int_0^2 x^3 dx$ при n=4, используя метод средних прямоугольников.

Решение.

$$\int_a^b f(x)dx \approx \frac{b-a}{n}\cdot\sum_{i=1}^{n} f\left(\frac{x_{i-1}+x_i}{2}\right) \pm |R_n(f)| \Rightarrow$$

$\int_0^2 x^3dx \approx \frac{2-0}{4}\cdot\sum_{i=1}^{4} f\left(\frac{x_{i-1}+x_i}{2}\right) \pm |R_n(f)|=$

$=\frac{1}{2}\sum_{i=1}^{4} f\left(\frac{x_{i-1}+x_i}{2}\right) + |R_n(f)| = \frac{1}{2}\left(f\left(\frac{x_0+x_1}{2}\right) + f\left(\frac{x_1+x_2}{2}\right) + f\left(\frac{x_2+x_3}{2}\right) + f\left(\frac{x_3+x4}{2}\right)\right) \pm |R_n(f)|$

Так как $x_i = x_0 + ih$ и $h = \frac{1}{2}$, получим:

$x_0 = 0$	$f\left(\frac{x_0+x_1}{2}\right) = f\left(\frac{0+1/2}{2}\right) = f\left(\frac{1}{4}\right) = \left(\frac{1}{4}\right)^3 = \frac{1}{64}$
$x_1 = x_0 + 1\cdot h = 0 + 1\cdot\frac{1}{2} =$	
$x_2 = x_0 + 2\cdot h = 0 + 2\cdot\frac{1}{2} =$	$f\left(\frac{x_1+x_2}{2}\right) = f\left(\frac{1/2+1}{2}\right) = f\left(\frac{3}{4}\right) = \left(\frac{3}{4}\right)^3 = \frac{27}{64}$
$x_3 = x_0 + 3\cdot h = 0 + 3\cdot\frac{1}{2} =$	
$x_4 = x_0 + 4\cdot h = 0 + 4\cdot\frac{1}{2} =$	$f\left(\frac{x_2+x_3}{2}\right) = f\left(\frac{1+3/2}{2}\right) = f\left(\frac{5}{4}\right) = \left(\frac{5}{4}\right)^3 = \frac{125}{64}$
	$f\left(\frac{x_3+x_4}{2}\right) = f\left(\frac{3/2+2}{2}\right) = f\left(\frac{7}{4}\right) = \left(\frac{7}{4}\right)^3 = \frac{343}{64}$

$|R_n(f)| \le \frac{M_2\cdot h^3\cdot n}{24}$, где $M_2 = \max\limits_{x\in[a,b]}|f''(x)|$

$f''(x) = 6x, \Rightarrow\ M_1 = \max\limits_{[0;\,2]}|6x| = 12, \Rightarrow\ |R_4(f)| \le \frac{12\cdot\left(\frac{1}{2}\right)^3\cdot 4}{24} = \frac{1}{4}$

Следовательно: $\int_0^2 x^3dx \approx \frac{1}{2}\cdot\left(\frac{1}{64}+\frac{27}{64}+\frac{125}{64}+\frac{343}{64}\right) \pm 0,25 \approx 3,875 \pm 0,25.$

§2. Метод трапеций

Формулу трапеций получают аналогично формуле прямоугольников: на каждом частичном отрезке криволинейная трапеция заменяется обычной.

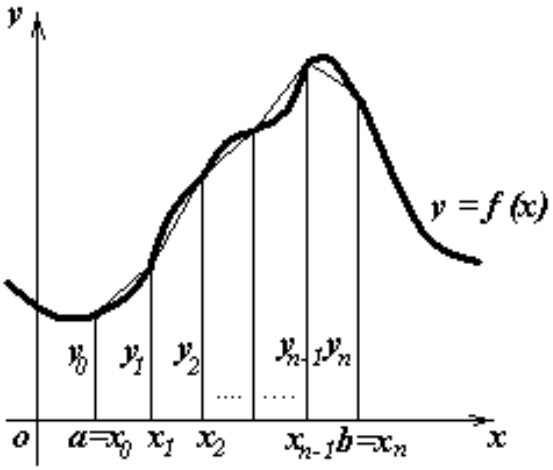


рис 3.2

Пусть на отрезке $[a,b]$, где $a<b$, задана непрерывная функция $f(x)$. Требуется вычислить интеграл $\int_a^b f(x)dx$, численно равный площади соответствующей криволинейной трапеции.

Разобьем основание этой трапеции на n равных частей длины

$$h=\frac{b-a}{n}=x_i-x_{i-1}.$$

Тогда $x_i=x_0+ih, \quad y_i=f(x_i)$.

На каждом частичном промежутке тогда построится трапеция вида ABCD: $A(x_{i-1},0), B(0,y_{i-1}), C(0,y_i), D(x_i,0), i=1,2,\dots,n$. Площадь каждой такой трапеции равна (полусумма оснований, умноженная на высоту):

$$S_{ABCD}=S_i=\frac{y_{i-1}+y_i}{2}\cdot(x_i-x_{i-1})=\frac{y_{i-1}+y_i}{2}\cdot h.$$

Тогда площадь всей криволинейной трапеции приблизительно равна сумме площадей трапеций S_i, высота каждой из которых равна h, то есть:

$$\int_a^b f(x)dx \approx S_1+S_2+S_3+\dots S_n$$

$$=h\cdot\frac{y_0+y_1}{2}+h\cdot\frac{y_1+y_2}{2}+\dots+h\cdot\frac{y_{n-1}+y_n}{2}=$$

$$=h\cdot\left(\frac{y_0+2y_1+2y_2+\dots+2y_{i-1}+2y_i+\dots+2y_{n-1}+y_n}{2}\right)=$$

$$=\frac{b-a}{n}\cdot\left(\frac{y_0+y_n}{2}+y_1+y_2+\dots+y_{n-1}\right)$$

Итак,

$$\int_a^b f(x)dx \approx \frac{b-a}{n}\cdot\left(\frac{y_0+y_n}{2}+y_1+y_2+\dots\dots+y_{n-1}\right)\pm|R_n(f)| \qquad (5)$$ –

формула трапеций, здесь

$R_n(f)$ –**абсолютная погрешность метода** (аналогично методу прямоугольников), которая составляет:

$$|\boldsymbol{R_n(f)}| \le \frac{\boldsymbol{M_2 \cdot h^3 \cdot n}}{\boldsymbol{12}}, \text{ где } \boldsymbol{M_2 = \max_{x \in [a,b]} |f''(x)|} \qquad \textbf{(4)}$$

Пример: Вычислить интеграл $\int_0^2 x^3 dx$ при $n = 4$, используя метод трапеций.

Решение. По формуле трапеций:

$$\int_0^2 x^3 dx \approx \frac{\boldsymbol{2-0}}{\boldsymbol{4}} \cdot \left(\frac{\boldsymbol{y_0+y_4}}{\boldsymbol{2}} + \boldsymbol{y_1 + y_2 + y_3}\right) \pm |R_n(f)|,$$

т.к. $\boldsymbol{x_i = x_0 + i \cdot h}$, $h = \frac{1}{2}$, то

$\boldsymbol{x_0 = 0, x_1 = x_0 + 1 \cdot h = 0 + 1 \cdot \frac{1}{2} = \frac{1}{2},}$

$\boldsymbol{x_2 = x_0 + 2 \cdot h = 0 + 2 \cdot \frac{1}{2} = 1,}$

$\boldsymbol{x_3 = x_0 + 3 \cdot h = 0 + 3 \cdot \frac{1}{2} = \frac{3}{2},}$

$x_4 = x_0 + 4 \cdot h = 0 + 4 \cdot \frac{1}{2} = 2.$

Тогда $y_0 = f(0) = 0^3 = 0$, $y_1 = f\left(\frac{1}{2}\right) = \left(\frac{1}{2}\right)^3 = \frac{1}{8}$,

$y_2 = f(1) = 1^3 = 1$, $y_3 = f\left(\frac{3}{2}\right) = \left(\frac{3}{2}\right)^3 = \frac{27}{8}$, $y_4 = f(2) = 2^3 = 8.$

Найдем погрешность:

$|R_n(f)| \le \frac{\boldsymbol{M_2 \cdot h^3 \cdot n}}{\boldsymbol{12}}$, где $M_2 = \max_{[a;\, b]} |f''(x)|$

$\boldsymbol{f''(x) = 6x,}$ $M_2 = \max_{[0;\, 2]} |6x| = 12,$ $|R_4(f)| \le \frac{12 \cdot \left(\frac{1}{2}\right)^3 \cdot 4}{12} = \frac{1}{2} = 0{,}5$

Следовательно,

$$\int_0^2 \boldsymbol{x^3 dx} \approx \frac{\boldsymbol{1}}{\boldsymbol{2}} \cdot \left(\frac{\boldsymbol{0+8}}{\boldsymbol{2}} + \frac{\boldsymbol{1}}{\boldsymbol{8}} + \boldsymbol{1} + \frac{\boldsymbol{27}}{\boldsymbol{8}}\right) \pm \boldsymbol{0,5} = 4{,}25 \pm 0{,}5.$$

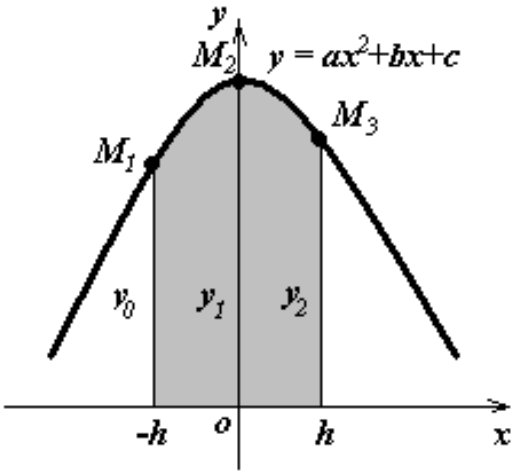
Замечание. Обратите внимание, что погрешности как метода трапеций, так и метода средних прямоугольников, имеют второй порядок. Если подынтегральная функция задана аналитически, то предпочтительнее использовать формулу средних прямоугольников, так как ее погрешность в 2 раза меньше по абсолютной величине по сравнению с методом трапеций. Если же функция задана таблично и нет возможности вычислить значения функции в серединных точках интервалов, то применение формулы левых/правых прямоугольников менее предпочтительно, так как погрешность этого метода

имеет первый порядок, т.е. намного менее точные по сравнению с формулой трапеций (см. лекции).

§3. Метод парабол (метод Симпсона)

Если заменить график функции на каждом отрезке $[x_{i-1}, x_i]$ не отрезками прямых, как в методах прямоугольников и трапеций, а дугами парабол, то получим более точную формулу приближенного значения интеграла $\int_a^b f(x)dx$.

Предварительно найдем вспомогательную площадь S криволинейной трапеции, ограниченной сверху параболой $y = ax^2 + bx + c$, прямыми $x = -h$, $x = h$ и отрезком $[-h, h]$.



Пусть парабола проходит через точки

$M_1(-h; y_0),\ M_2(0; y_1),\ M_3(h; y_2)$.

Значит, ординаты указанных точек вычисляются так:

$$\begin{cases} y_0 = a(-h)^2 + b(-h) + c = ah^2 - bh + c \\ y_1 = a \cdot 0^2 + b \cdot 0 + c = c \\ y_2 = ah^2 + bh + c \end{cases}$$

(6)

Тогда полученная площадь на участке $[-h, h]$ равна:

$$\boldsymbol{S = \int_{-h}^{h}(ax^2 + bx + c)dx = \left(a\frac{x^3}{3} + b\frac{x^2}{2} + cx\right)|_{-h}^{h} =}$$

$$\boldsymbol{= a\frac{h^3}{3} + b\frac{h^2}{2} + ch - \left(a\frac{(-h)^3}{3} + b\frac{(-h)^2}{2} + c(-h)\right) = \frac{2}{3}ah^3 + 2ch}$$

(7)

Выразим полученное в (7) значение через y_0, y_1, y_2.

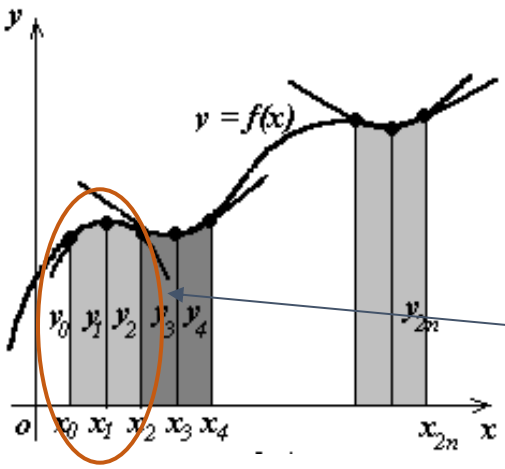
Из формул (6) имеем: $c = y_1,\ a = \frac{1}{2h^2}(y_0 - 2y_1 + y_2)$.

Подставляем полученные значения в (7):

$$S = \frac{h}{3}(y_0 + 4y_1 + y_2) \qquad \textbf{(8)}$$

Вывод формулы парабол (Симпсона)

Пусть дана криволинейная трапеция, ограниченная функциями $y = f(x),\ x = a,\ x = b, y = 0.$



1. Разобьем отрезок $[a, b]$ (где $a < b$) на ***2n*** равных частей. Получим отрезки длиной

$$h = \frac{b-a}{2n} \qquad \textbf{(1)}$$

2. В точках деления вычислим значения функции $y = f(x)$, т.е. вычислим

$$y_0, y_1, y_2, \dots, y_{2n-2}, y_{2n-1}, y_{2n}.$$

Заменим каждую пару соседних криволинейных трапеций **параболической трапецией** с основанием, равным $2h$.

На отрезке $[x_0, x_2]$ парабола проходит через точки (x_0, y_0), (x_1, y_1), (x_2, y_2).

Используя формулу (8), получим: $S_1 = \int_{x_0}^{x_2} f(x)dx = \frac{h}{3}(y_0 + 4y_1 + y_2)$

Аналогично на отрезке

$[x_2, x_4]$: $\quad S_2 = \int_{x_2}^{x_4} f(x)dx = \frac{h}{3}(y_2 + 4y_3 + y_4)$ и т. д.

$[x_{2n-2}, x_{2n}]$: $\quad S_n = \int_{x_{2n-2}}^{x_{2n}} f(x)dx = \frac{h}{3}\left(y_{2n-2} + 4y_{2n-1} + y_{2n}\right)$

Следовательно:

$$\int_a^b \boldsymbol{f(x)dx = S_1 + S_2 + \dots + S_n =}$$

$$= \frac{\boldsymbol{h}}{\boldsymbol{3}}(\boldsymbol{y_0 + 4y_1 + y_2})$$

$$+ \frac{\boldsymbol{h}}{\boldsymbol{3}}(\boldsymbol{y_2 + 4y_3 + y_4}) + \dots + \frac{\boldsymbol{h}}{\boldsymbol{3}}(\boldsymbol{y_{2n-2} + 4y_{2n-1} + y_{2n}}) =$$

$$= \frac{\boldsymbol{h}}{\boldsymbol{3}}[(\boldsymbol{y_0 + y_{2n}}) + \boldsymbol{4}(\boldsymbol{y_1 + y_3 + \dots + y_{2n-1}})$$

$$+ \boldsymbol{2}(\boldsymbol{y_2 + y_4 + \dots + y_{2n}})]$$

Учитывая погрешность вычислений $|R_n|$ и подставляя $h = \frac{b-a}{2n}$, получим **формулу Симпсона:**

$$\int_a^b f(x)dx = \frac{b-a}{6n}[(y_0 + y_{2n}) + 4(y_1 + y_3 + \ldots + y_{2n-1}) + 2(y_2 + y_4 + \ldots + y_{2n-2})] \pm |R_n| \qquad (10)$$

Замечание. При практическом применении формулы (10) удобно запомнить обозначения следующим образом:

- ✓ y_0 и y_{2n} –значения функции, соответствующие концам промежутка интегрирования a и b;
- ✓ $y_1, y_3, \ldots, y_{2n-1}$ –значения функции, соответствующие точкам деления отрезка $[a, b]$ с шагом h;
- ✓ $y_2, y_4, \ldots, y_{2n-2}$ – значения функции, соответствующие серединам отрезков, полученных при делении отрезка $[a, b]$ с шагом h.

Абсолютная погрешность метода оценивается соотношением:

$$|R_n(f)| \le \frac{M_3 \cdot h^3}{192} \cdot (b-a), \text{ где } M_3 = \max_{[a;\,b]} |f'''(x)| \qquad (11)$$

Пример:

Вычислить интеграл $\int_0^2 x^3 dx$, используя метод парабол при $n = 4$.

Решение.

Количество разбиений $2n = 8$, $h = \frac{2-0}{2 \cdot 4} = \frac{1}{4}$, $f(x) = x^3$

Составим таблицу:

x	y_0, y_8	$y_{четное}$	$y_{нечетное}$
$x_0 = 0$	$y_0 = f(0) = 0$		
$x_1 = 0 + \frac{1}{4} = \frac{1}{4}$			$y_1 = f\left(\frac{1}{4}\right) = \left(\frac{1}{4}\right)^3 = \frac{1}{64}$
$x_2 = 0 + 2 \cdot \frac{1}{4} = \frac{2}{4} = \frac{1}{2}$		$y_2 = f\left(\frac{1}{2}\right) = \left(\frac{1}{2}\right)^3 = \frac{1}{8}$	
$x_3 = 0 + 3 \cdot \frac{1}{4} = \frac{3}{4}$			$y_3 = f\left(\frac{3}{4}\right) = \left(\frac{3}{4}\right)^3 = \frac{27}{64}$
$x_4 = 0 + 4 \cdot \frac{1}{4} = 1$		$y_4 = f(1) = 1^3 = 1$	

$x_5 = 0 + 5 \cdot \frac{1}{4} = \frac{5}{4}$			$y_5 = f\left(\frac{5}{4}\right) = \left(\frac{5}{4}\right)^3 = \frac{125}{64}$
$x_6 = 0 + 6 \cdot \frac{1}{4} = \frac{3}{2}$		$y_6 = f\left(\frac{3}{2}\right) = \left(\frac{3}{2}\right)^3 = \frac{27}{8}$	
$x_7 = 0 + 7 \cdot \frac{1}{4} = \frac{7}{4}$			$y_7 = f\left(\frac{7}{4}\right) = \left(\frac{7}{4}\right)^3 = \frac{343}{64}$
$x_8 = 2$	$x_8 = f(2) = 2^3 = 8$		

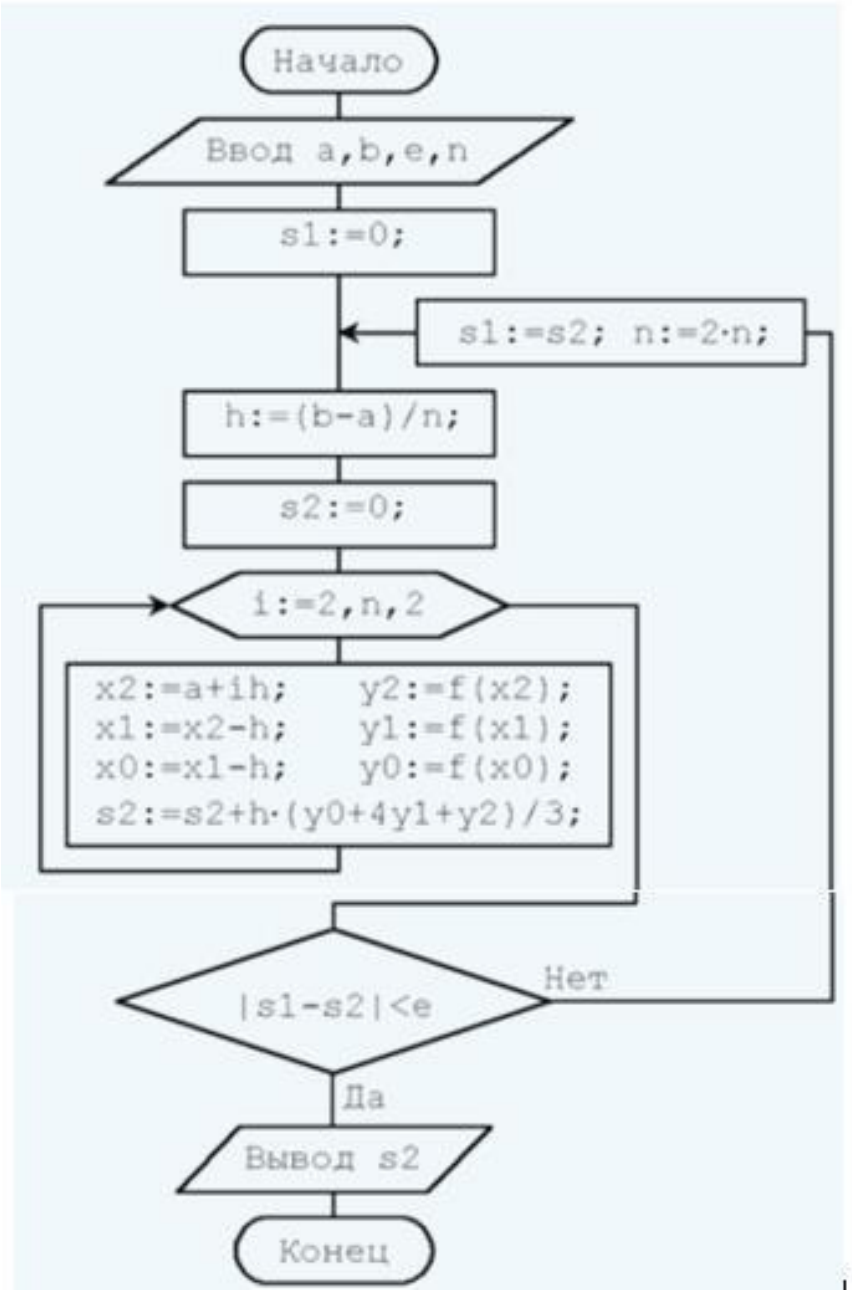
Рассмотрим погрешность метода:

$$|R_n(f)| \leq \frac{M_3 \cdot h^3}{192} \cdot (b - a)$$

$$|R_4(f)| \leq \frac{M_3 \cdot h^3}{192} = \frac{6 \cdot \left(\frac{1}{2}\right)^3 \cdot 2}{192} = 0{,}0078125$$

Итак, результат по формуле Симпсона (парабол):

$\int_0^2 x^3 dx = \frac{2-0}{6 \cdot 4}\left[(0+8) + 4\left(\frac{1}{64} + \frac{27}{64} + \frac{125}{64} + \frac{343}{64}\right) + 2\left(\frac{1}{8} + \frac{8}{8} + \frac{27}{8}\right)\right] \pm$
$\pm 0{,}0078125 = \mathbf{4 \pm 0{,}0078125}$.



ЛАБОРАТОРНАЯ РАБОТА №6

Тема лабораторной работы:
численное решение задачи Коши
для обыкновенного дифференциального уравнения
первого порядка

Цель работы – ознакомление с методами численного решения задачи Коши для обыкновенного дифференциального уравнения (ОДУ), их сходство и разницу с методами численного интегрирования функций.

Описание методов

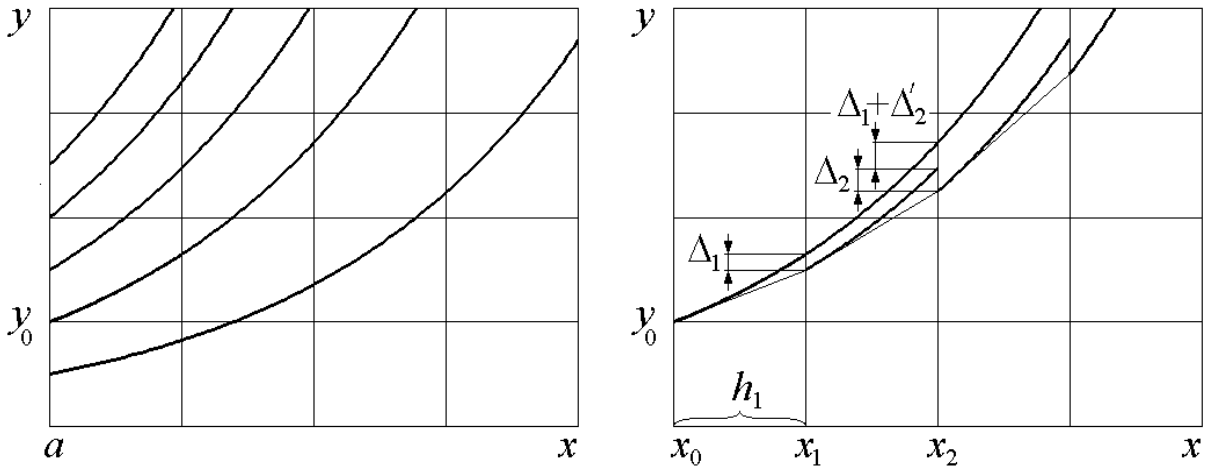
Задача Коши для ОДУ первого порядка формулируется следующим образом: необходимо решить ОДУ

$$\frac{dy}{dx} = f(x, y) \tag{1}$$

при следующем начальном условии

$$y(x_0) = y_0. \tag{2}$$

Известно, что решение дифференциального уравнения первого порядка **графически** представляется в виде однопараметрического семейства кривых, т.е. зависимостей $y(x, c)$, где c – константа интегрирования (см. например, рис. 1, *а*).



а) *б)*

Рис. 1. Решение задачи Коши для обыкновенного дифференциального уравнения

Задание начального значения позволяет **выбрать** из этого семейства **одну кривую, которая является решением задачи Коши.**

Разобьем отрезок интегрирования на n частей.

Введем в рассмотрение последовательность узловых точек $x \in [a,\ b]$:

$x_0 = a,\ x_1 = a + h,\ x_2 = a + 2h, \ldots, x_n = a + nh,$

где $h = \frac{b-a}{n} = x_{i+1} - x_i$ – шаг разбиения.

Допустим, интегрирование проводится при постоянном значении $f(x, y)$, аналогично методу левых прямоугольников численного интегрирования функций

$$\boldsymbol{y_{j+1} = y_j + h \cdot f(x_j, y_j).} \quad (3)$$

Поскольку значение функции $f(x, y)$ равно производной искомой функции $y(x)$, из этого следует, что происходит сдвиг вдоль касательной, проведенной к графику функции $y(x)$ из начальной точки. Далее определяется угол наклона касательной в полученной точке и проводится новый шаг.

Такой метод решения задачи Коши называют методом Эйлера. Результаты применения этого метода показаны на рис. 1, *б)*.

Отметим только, что вследствие погрешности Δ_1, появившейся на первом шаге, второй шаг совершается относительно кривой, которая не совпадает с исходной. Таким образом, **погрешность** решения при совершении второго шага складывается из двух частей: погрешности Δ_j, вызванной сдвигом вдоль касательной, и погрешностью Δ'_j, обусловленной погрешностью определения тангенса угла наклона касательной.

Метод Рунге-Кутты дает более точный результат по сравнению с методом Эйлера. В методе Рунге-Кутты используется большее количество оценок функции, чтобы гарантировать, что ее погрешность пропорциональна четвертой степени размера шага: для нахождения каждого следующего y_{i+1} предварительно надо вычислить коэффициенты $\boldsymbol{k_1, k_2, k_3, k_4}$ (формулы приведены в таблице)

Формулы методов Эйлера и Рунге-Кутты 4-го порядка представлены в таблице 1:

Таблица 1

№	Название метода	Формула метода
1	Метод Эйлера	$\boldsymbol{y_{j+1} = y_j + h \cdot f(x_j, y_j)},$ $j = 0,1,\dots, n-1$
2	Метод Рунге-Кутты 4-го порядка	$\boldsymbol{k_1} = h \cdot f(x_j, y_j),$ $\boldsymbol{k_2} = h \cdot f(x_{j+1/2}, y_j + k_1/2),$ где $x_{j+1/2} = x_j + h/2,$ $\boldsymbol{k_3} = h \cdot f(x_{j+1/2}, y_j + k_2/2),$ $\boldsymbol{k_4} = h \cdot f(x_{j+1}, y_j + k_3),$ $\boldsymbol{y_{j+1} = y_j + \frac{1}{6}[k_1 + 2k_2 + 2k_3 + k_4]},$ $j = 0,1,\dots, n-1.$

Таблица 2: варианты заданий

1. $y' = x - y^2,\ y(1) = 3$	6. $y' = y + sin\, x\,,\ y(0) = 1$
2. $y' = 2x - y,\ y(2) = 0$	7. $y' = x^3 + 2,\ y(1) = 1$
3. $y' = y^2 - 3x,\ y(1) = 2$	8. $y' = x^3 + y,\ y(0) = 1$
4. $y' = \frac{y^2}{x},\ y(1) = -1$	9. $y' = tgx - y,\ y(0) = 0$
5. $y' = x + 1 + \frac{y}{x},$ $y(1) = 0$	10. $y' = 3x^2 - y^2,\ y(0) = 1$

Задание на лабораторную работу:

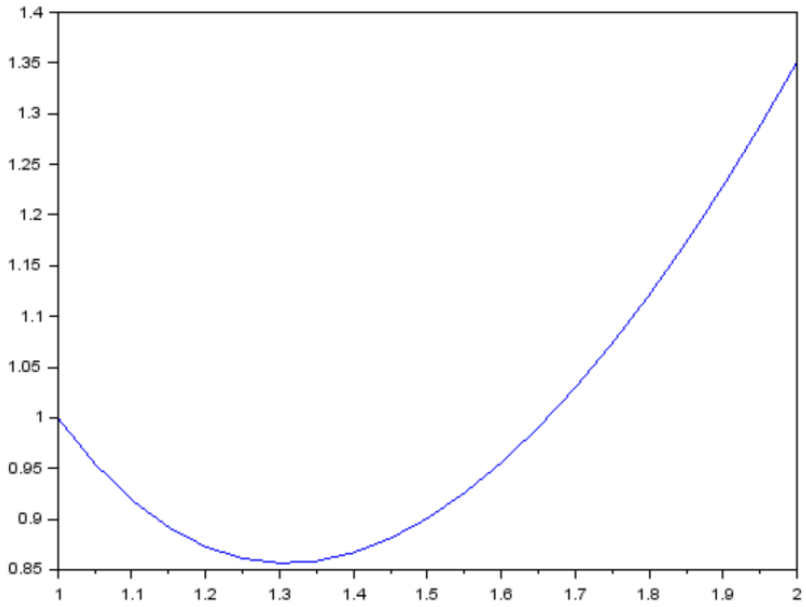
1. Решить дифференциальное уравнение (для заданного варианта из табл.2) методом Эйлера с шагами $h = 0{,}1$ и $h = 0{,}05$ на промежутке изменения аргумента длиной 2, т.е. $[x_0,\ x_0 + 2]$. Построить графики полученного решения для каждого шага.
2. Решить то же дифференциальное уравнение методом Рунге-Кутты с шагом $h = 0{,}1$ на том же промежутке изменения аргумента. Построить график полученного решения и сравнить результат с предыдущим методом.

3. Найти решение того же дифференциального уравнения с помощью scilab (функция ode) (см. пример) с шагом 0,1; 0,01.
4. Сделать сравнительные выводы о результатах (погрешности), полученных при решении дифференциального уравнениями двумя методами.

Пример программы для вычисления в scilab

```
function yd=f (t, x), yd=t^2-2*x, endfunction;
x0=1; //значение функции в начальной точке
t0=1; //начальной точке (абсцисса)
t=1:0.05:2; //интервал, на котором строим кривую, шаг 0.05
y=ode(x0, t0, t, f);
plot(t, y)
```

Полученная часть интегральной кривой:



Лабораторная работа.

Минимизация функции одной переменной

Задание:

1. для заданной функции (см. варианты заданий) определить интервал, на котором она унимодальна; // это можно сделать, исследуя график

функции с использованием производной или построением в excel, в некоторых вариантах промежутки нахождения минимума указаны);

2. методом золотого сечения найти минимум функции и сравнить его с точным значением ($\varepsilon = 10^{-3}$);
3. методом дихотомии найти минимум функции и сравнить его с точным значением ($\varepsilon = 10^{-3}$);
4. проанализировать полученные результаты: сделать вывод, как изменится результат при изменении первоначальной длины интервала и более высокой точности ($\varepsilon = 10^{-5}$).

Варианты заданий

№ варианта	***Задание***
1	$y = 24 - \frac{2}{3}x + \frac{1}{30}x^2$ (рассмотреть промежуток от [7; 12)
2	$y = -x\sqrt{1 - x^2}$
3	$y = (x - 5)e^x$ [3,5; 4,5]
4	$y = x^4 - 6x^2 + 10$ [-2,5; 0]
5	$y = -\frac{1}{3}x^3 + x^2 + \frac{1}{3}$
6	$y = -(x - 1)^2(x + 2)^3$ [0; 2]
7	$y = x^3 - 3x + 3$
8	$y = x + \frac{1}{\sqrt{x+1}} - 6$ [-0,9; 1]
9	$y = 3x^4 - 3x^3$ [0,1; 1,1]
10	$y = \frac{1}{x} + 4\sqrt{x}$
11	$y = 0{,}1x^4 - 8x$
12	$y = x^4 - 6x^2 + 10$

Оптимизация – процесс выбора наилучшего варианта из всех возможных. В процессе решения задачи оптимизации обычно необходимо найти оптимальные значения параметров, определяющих данную задачу. Их называют проектными параметрами. Выбор оптимального решения или сравнения двух альтернативных решений проводится с помощью некоторой зависимой величины, определяемой проектными параметрами. Эта величина называется целевой функцией. Если целевая функция зависит от одного проектного параметра, то мы имеем одномерную задачу оптимизации.

Цель одномерной оптимизации – определение ***минимума*** (или ***максимума***) функции одной переменной, заданной на интервале изменения аргумента. Предположим, что функция на отрезке $[a, b]$ ***унимодальна***, т. е. на данном отрезке она имеет только один минимум.

Процесс решения задачи состоит в последовательном сужении интервала изменения параметра, называемого интервалом неопределенности. В начале процесса оптимизации его длина равна $b - a$, а к концу она должна стать менее заданного допустимого значения ε.

1. Метод дихотомии

Метод дихотомии несколько схож с методом половинного деления для решения уравнений, однако отличается от него критерием отбрасывания концов.

Пусть задана функция $f(x): [a, b] \to \mathbb{R}$, $f(x) \in C([a, b])$, являющаяся на отрезке $[a, b]$ унимодальной.

Разобьём мысленно заданный отрезок пополам и возьмём две симметричные относительно центра точки x_1 и x_2 так, что:

$$x_1 = \frac{a+b}{2} - \delta, \qquad x_2 = \frac{a+b}{2} + \delta$$

где δ – некоторое число в интервале (0; (*b*–*a*)/2).

Вычислим значения функции $f(x)$ в двух новых точках. Сравнением определим, в какой из двух новых точек значение функции $f(x)$ максимально. Отбросим тот из концов изначального отрезка, к которому точка с максимальным значением функции оказалась ближе (напомним, мы ищем ***минимум***), то есть:

- Если $f(x_1) > f(x_2)$, то берётся отрезок $[x_1, b]$, а отрезок $[a, x_1]$ отбрасывается.

- Иначе берётся зеркальный относительно середины отрезок$[a, x_2]$, а отбрасывается $[x_2, b]$.

Процедура повторяется, пока не будет достигнута заданная точность: если $|b - a| < 2\varepsilon$, то $x = \frac{a+b}{2}$, решение найдено (остановка процесса).

2. Метод золотого сечения

При построении процесса оптимизации стараются сократить объем вычислений и время поиска. Одним из наиболее эффективных методов является метод золотого сечения – это численный метод, используемый для нахождения минимума (или максимума) unimodal функции на заданном интервале. Он основан на свойствах золотого сечения, которое делит отрезок на две части так, что отношение длины большей части к меньшей равно золотому сечению (приблизительно 1.618). Метод состоит в построении последовательности отрезков$[a_0, b_0]$, $[a_1, b_1]$, …, стягивающихся к точке минимума функции. На каждом шаге, за исключением первого, вычисление значения функции производится один раз в точке, называемой золотым сечением. Золотое сечение интервала выбирается так, чтобы отношение длины большего отрезка l_1 к длине всего интервала l равнялось отношению длины меньшего отрезка l_2 к длине большего отрезка l_1:

$$l_1/l = l_2/l_1.$$

Из этого соотношения можно найти точку деления:

$$l_1{}^2 = l_2 l = l_2(l_1 + l_2),$$

$$l_2{}^2 + l_1 l_2 - l_1{}^2 = 0,$$

$$(l_2/l_1)^2 + l_2/l_1 - 1 = 0,$$

$$l_2/l_1 = (-1 \pm \sqrt{5})/2.$$

Так как нас интересует только положительное решение, то

$$l_2/l_1 = l_1/l = (-1 + \sqrt{5})/2 \approx 0.618.$$

(0,618 – пропорция «золотого сечения»)

Отсюда

$$l_1 \approx 0.618l,$$

$$l_2 = 0.382l.$$

Алгоритм поиска минимума функции методом золотого сечения

1. Заданный отрезок унимодальности функции делится двумя симметричными (относительно его центра) точками и рассчитываются значения в этих точках.
2. Тот из концов отрезка, к которому среди двух вновь поставленных точек ближе оказалась та, значение в которой **максимально** (для случая поиска минимума), отбрасывают.
3. Ищем (одну) новую точку.
4. Процедура продолжается до тех пор, пока не будет достигнута заданная точность.

Формальное описание алгоритма:

1. Задаются начальные границы отрезка $[a,\ b]$ и точность ε.
2. Вычислить коэффициент золотого сечения: $\lambda = \left(1+\sqrt{5}\right)/2 \approx 1.618$.
3. Вычислить точки деления:
$$x_1 = b - (b-a)/\lambda \quad \text{и} \quad x_2 = a + (b-a)/\lambda$$
4. Вычислить значения функции $y_1 = f(x_1)$ и $y_2 = f(x_2)$.
5. Выбираем отрезок, в котором локализован минимум:
   если $y_1 < y_2$, то
   - ✓ установить $b = x_2$
   - ✓ пересчитать $x_2 = x_1$
   - ✓ пересчитать $x_1 = b - (b-a)/\lambda$.

   иначе
   - ✓ установить $a = x_1$
   - ✓ пересчитать $x_1 = x_2$

   пересчитать $x_2 = a + (b-a)/\lambda$

6. Если $|b-a|<\varepsilon$, то решение найдено: $\boldsymbol{x=\frac{a+b}{2}} \Longrightarrow$ завершение итерационного процесса. Иначе – шаг 4.

	a	b	x1	x2	f(x1)	f(x2)	длина промежутка [a,b]		расчеты промеж x1	расчеты промеж x2	$\lambda = (1+\sqrt{5})/2 \approx$ **1.618**
2	-2	-0,9	-1,57985167	-1,320148	1,25408526	2,580572919	1,1				1,618
3	-2	-1,320148331	-1,7403286	-1,579852	1,0008262	1,254085259	0,679851669		-1,740328596	-1,579819735	
4	-2	-1,579851669	-1,83952307	-1,740329	1,14733709	1,000826196	0,420148331		-1,839523072	-1,740328596	
5	-1,83952307	-1,579851669	-1,7403286	-1,679034	1,0008262	1,032704915	0,259671404		-1,740340793	-1,679033948	
6	-1,83952307	-1,679033948	-1,77822377	-1,740329	1,02626985	1,000826196	0,160489125		-1,778223765	-1,740333255	
7	-1,77822377	-1,679033948	-1,7403286	-1,71692	1,0008262	1,002723421	0,099189817		-1,740337914	-1,716919799	
8	-1,77822377	-1,716919799	-1,75480853	-1,740329	1,00629689	1,000826196	0,061303966		-1,75480853	-1,740335035	
9	-1,75480853	-1,716919799	-1,7403286	-1,731392	1,0008262	1,000005214	0,037888731		-1,740336814	-1,731391514	
10	-1,7403286	-1,716919799	-1,73139151	-1,725861	1,00000521	1,000458144	0,023408797		-1,731387535	-1,72586086	
11	-1,7403286	-1,72586086	-1,7348026	-1,731392	1,00009101	1,000005214	0,014467736		-1,734802601	-1,731386856	
12	-1,7348026	-1,72586086	-1,73139151	-1,729276	1,00000521	1,000092234	0,008941741		-1,731387276	-1,729276185	
13	-1,7348026	-1,729276185	-1,73269177	-1,731392	1,00000493	1,000005214	0,005526416		-1,732691769	-1,731387016	
14	-1,7348026	-1,731391514	-1,73349973	-1,732692	1,00002521	1,000004932	0,003411086		-1,733499726	-1,732694389	
15	-1,73349973	-1,731391514	-1,73269177	-1,732197	1,00000493	1,000000256	0,002108211		-1,732694488	-1,732196752	
16	-1,73269177	-1,731391514	-1,73219675	-1,731888	1,00000026	1,000000317	0,001300255		-1,732195133	-1,731888151	
17	-1,73269177	-1,731888151	-1,73238482	-1,732197	1,00000134	1,000000256	0,000803619		-1,732384825	-1,732195095	
18	-1,73238482	-1,731888151	-1,73219675	-1,732078	1,00000026	1,000000009	0,000496674		-1,732195119	-1,732077857	
19	-1,73219675	-1,731888151	-1,73207786	-1,732006	1,00000001	1,000000024	0,000308601		-1,732078881	-1,732006022	
20	-1,73219675	-1,732006022	-1,7321239	-1,732078	1,00000006	1,000000009	0,00019073		-1,732123902	-1,732078872	
21											
22											
23		Минимум функции с точностью до 10^(-3			-1,732						
24		количество итераций:			16						
25											