Визуализация и обработка данных. Семинар 1.

Введение в учебный курс.

Вычислительная математика - это область математики, изучающая методы доведения математических задач до результата (то есть вычисления) и методы использования вычислительных средств (в наше время основном изучают компьютерные вычисления).

Изложение вычислительной математики зависит от того, для решения каких задач она предназначена. Обычно университетские курсы вычислительной математики для физиков ориентированы на численные методы, которые связаны с математической физикой. Например - вычисление интегралов или решение уравнений в частных производных.

Наш курс отличается прежде всего тем, что он рассматривает и два других вида задач. Это анализ экспериментальных данных и символьные вычисления. При этом визуализация и обработка данных является приоритетной задачей курса, с этой же целью по сравнению с традиционным курсом увеличен объём материала, посвящённый вычислительной линейной алгебре (добавили больше про матричные разложения).

Первый семестр состоит из анализа данных и численных методов линейной алгебры. Большое внимание будет уделено практическому использованию Python и его библиотек для решения задач.

На Яндекс-диске размещена библиотека материалов, которая использовалась при создании учебного курса. Здесь https://disk.yandex.ru/d/jd-3ijEOpwohmg надо смотреть папку "библиотека вычматов".

Введение в анализ и визуализацию данных.

Основам анализа и визуализации данных вас учили на кафедре общей физики, в ходе учебных экспериментов https://mipt.ru/education/chair/physics/laboratornyy-praktikum/. Там же есть отдельный учебный курс, который можно считать вводным к данному курсу.
Ссылки: https://github.com/mipt-genphys/mipt-stat-book
https://mipt.ru/education/chair/physics/records/obrabotka-rezultatov-eksperimenta.php
http://npm.mipt.ru/books/lab-intro/
https://mipt-npm.github.io/python-scientific-book/
https://mipt.ru/upload/medialibrary/111/main.pdf

Учебные курсы, связанные с обработкой учебных экспериментов, являются недостаточными в подготовке физиков, так как современные эксперименты очень

сильно отличаются от обычных учебных экспериментов. Если в обычном учебном эксперименте требуется сделать максимум несколько десятков измерений, то есть за всё время лабораторной работы получается меньше килобайта данных, то на LHC каждую секунду с детекторов поступает сотни гигабайт данных. Там, где данных довольно много и их сложно обрабатывать, всегда используется специализированное программное обеспечение, нередко позволяющее также моделировать сам эксперимент и управлять оборудованием (например, LabVIEW , ROOT). Однако многие задачи анализа и обработки данных, моделирования, можно решать с помощью библиотек Python.

Язык Python разрабатывался так, чтобы на нем можно было легко писать код. Вы потратите намного больше времени на код на C++, чем на питоновский код. Правда, за это приходится платить низкой производительностью — программа на C++ будет в десятки раз быстрее, чем программа на Питоне. Но если вы пишете какой-нибудь простой исследовательский код, то тут вы потратите намного меньше времени и нервов, если напишете код на Питоне.

Также в Питоне есть огромное количество библиотек для анализа данных (описание некоторых ниже) с очень хорошей документацией.

NumPy — библиотека для работы с массивами и матрицами, в том числе матричные операции.

SciPy — библиотека для выполнения научных и инженерных расчётов.

ScikitLearn (sklearn) — библиотека алгоритмов машинного обучения с множеством примеров и наборами данных.

Matplotlib — библиотека для визуализации данных.

SymPy - библиотека для символьных вычислений.

Желающие могут изучить пакет Pandas для работы с табличными данными, но нам он скорее всего не понадобится.

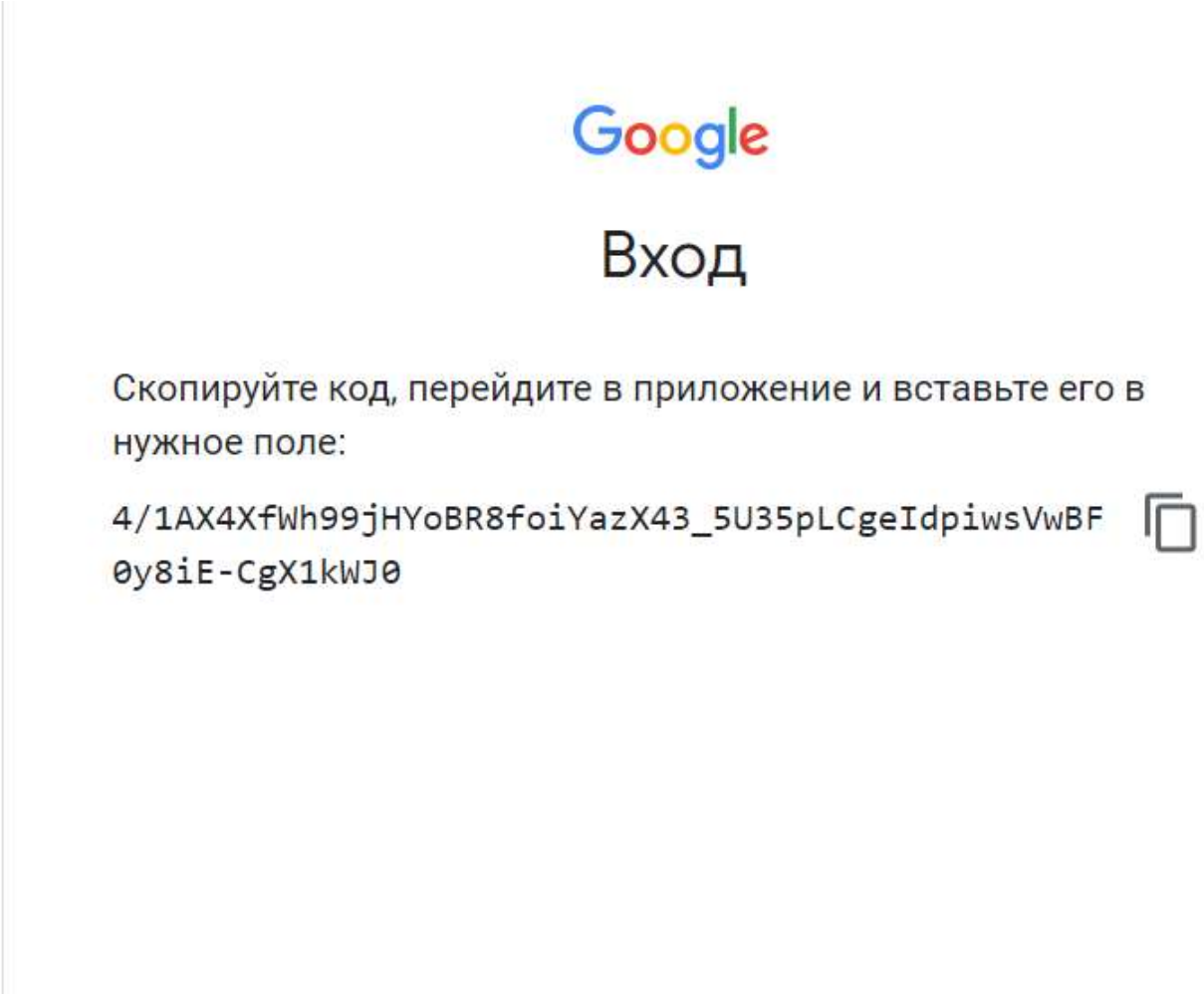
Рекомендуется установить Jupyter Notebook (инструкция есть в приложенных файлах), чтобы просматривать документы и писать код, но можно ничего не устанавливать и использовать онлайн-инструменты. Среди онлайн инструментов в первую очередь рекомендуется Google Colab. https://colab.research.google.com/notebooks/intro.ipynb

Чтобы там работать, нужно перейти по ссылке, дать разрешение на доступ к своему гугл-диску, ввести код:

from google.colab import drive

drive.mount('/content/drive/')

После этого войти с помощью гугл-аккаунта, появится код доступа, его надо ввести в появившуюся строку ввода



Если всё сделать правильно, то появится сообщение "Mounted at /content/drive/"

```
from google.colab import drive

drive.mount('/content/drive/')
```

```
Mounted at /content/drive/
```

Файлы кафедры дискретной математики МФТИ можно использовать как справочник по Python и его библиотекам. При желании найти дополнительные возможности - посмотреть официальную документацию библиотек.

Введение в предмет вычислительной математики.

Вычислительная математика является частью высшей математики, компьютерных наук и науки о математическом моделировании. Однако она имеет свою специфику (пункты по книге И.Б. Петрова "Вычислительная математика для физиков" (2021), пример взят из

интернета https://3ys.ru/chislennye-metody-v-zadachakh-matematicheskogo-analiza/o-reshenii-kvadratnogo-uravneniya.html):

1) дискретизация непрерывных областей;

2) в расчётах всегда присутствует машинная погрешность округления, так как количество знаков после запятой ограничено;

3) выбор численного метода влияет на результат решения задачи;

4) исследуется экономичность вычислений;

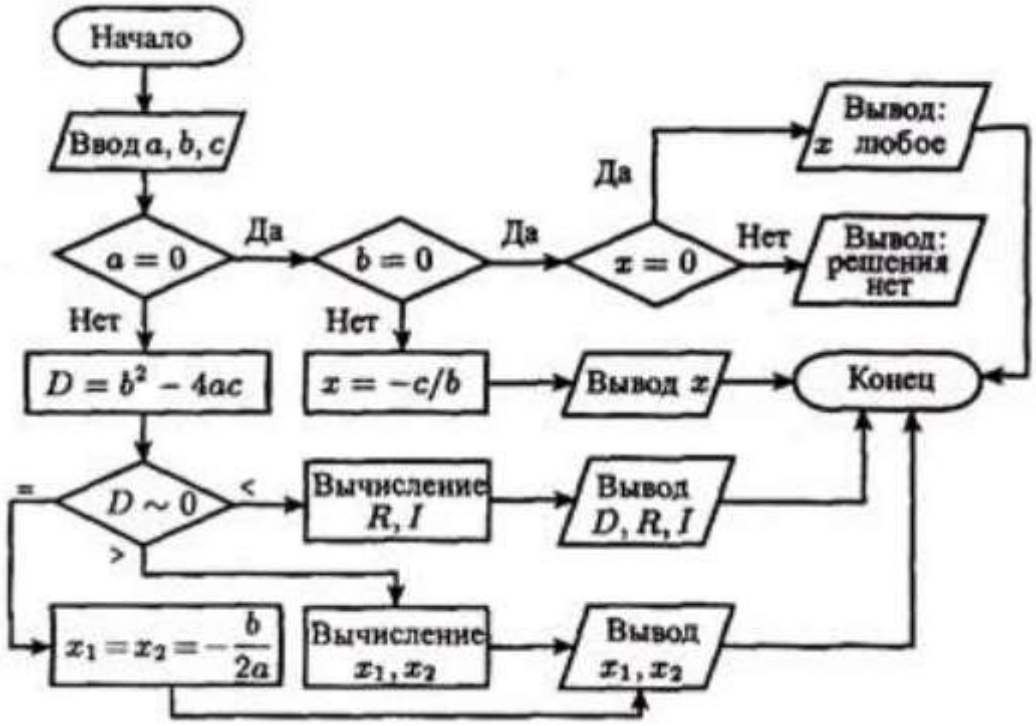
5) важна обусловленность задачи - чувствительность её решения к малым изменениям входных данных;

6) устойчивость численных методов – это чувствительность методов к малым изменениям входных данных.

В качестве примера специфики вычислительной математики приведём задачу решения квадратного уравнения. В курсе школьной алгебры все квадратные уравнения решают просто через дискриминант, но в вычислительной математике такой метод работает далеко не для всех квадратных уравнений.

Необходимо исследовать возможность появления погрешностей в зависимости от соотношения между коэффициентами уравнения. Рассмотрим один из важнейших случаев, когда коэффициент b (при x) значительно превышает по абсолютной величине остальные (a и c). В этом случае корень из дискриминанта очень близок |b| и возникает проблема вычитания друг из друга близких значений. Минимальный корень очень маленький и для получения его точного значения необходимо либо повысить количество используемых разрядов в записи вещественного числа, либо вычислять его иначе (например, больший корень по обычной формуле, а меньший вычислять на основе большего по теореме Виета).

Если написать алгоритм вычисления корня квадратного уравнения с учётом этих особенностей, то его блок-схема выглядит так (R - это вещественная часть корня, I - мнимая).



Если подумать, какие могут быть проблемы с этим алгоритмом, то можно обнаружить как минимум три ситуации, когда он не работает.

Пример 1. a = 10^(-200), b=-3•10^(-200), c = 2•(10^-200). При вычислении произведений b^2 и 4ac получается машинный нуль, т.е. D = 0; решение пойдет по ветви равных корней: x1= x2 = 1.5. Точные значения корней, как нетрудно видеть, x1 = 1, x2 = 2.

Пример 2. a=10^200, b = -3•10^200, c=2•10^(-200). Этот вариант аналогичен предыдущему случаю с той лишь разницей, что вместо получения машинного нуля произойдет переполнение и прерывание счета.

Пример 3. a = 10^(-200), b =10^200, c=-10^200. Это трудный для реализации на компьютере случай. В практических расчетах встречаются уравнения с малым коэффициентом при x^2. В этом случае b^2>> 4ac, но при вычислении b^2 произойдет переполнение. Простейшим выходом из этого положения может быть сведение к случаю a = 0 с обязательной проверкой других коэффициентов.

Есть современные программные пакеты, которые умеют квадратные уравнения решать по куда более сложным алгоритмам. Например, Вольфрам умеет сам определять, сколько разрядов нужно, и автоматически подключает длинную арифметику. Он также умеет использовать разные формулы (он их сам выводит с помощью алгоритмов символьной арифметики), выбирая среди них те, которые считают точнее (умеет это анализировать и сравнивать, там очень нетривиальный алгоритм) и даже самостоятельно определяет погрешность вычислений. Однако, и для Вольфрама можно найти квадратные уравнения, которые он решает неправильно.

https://www.wolframalpha.com/input/?i=%2810%5E%28-200%29%29*x%5E2+%2B+%2810%5E%28200%29%29*x+-+10%5E%28200%29+%3D+0

$$\frac{x^2}{10^{200}} + 10^{200}\,x - 10^{200} = 0$$

Меньший корень здесь очень близок к 1, а Вольфрам выдаёт, что он очень близок к нулю.

Основные виды задач анализа данных.

Основным признаком классификации задач анализа данных (и также машинного обучения, которое для работы с данными предназначено) является классификация по откликам (видам выходных данных). Поэтому, хотя часто выделяют много видов задач, их все можно свести к трём категориям (по критерию разного типа отклика):

1) Задачи классификации и распознавания образов — множество возможных ответов конечно. Их называют идентификаторами классов или метками.

2) Задачи регрессии и задачи аппроксимации — ответы являются действительными числами или векторами из действительных чисел.

3) Задачи прогнозирования – ответы характеризуют будущее поведение процесса или явления.

Иногда даже выделяют только две категории видов задач (регрессию и классификацию), не выделяя отдельно задачи прогнозирования, так как задачи прогнозирования сами делятся на прогнозирование дискретных меток и прогнозирование непрерывных данных.

Более распространённой является классификация на 5 видов (основана на совмещении классификации по виду выходных данных с разделением задач по тому, какие методы используются для их решения):

1) Задача регрессии – прогноз или аппроксимация на основе выборки объектов с различными признаками. На выходе должно получиться вещественное число (2, 35, 76.454 и др.), к примеру цена квартиры, стоимость ценной бумаги по прошествии полугода, ожидаемый доход магазина на следующий месяц, качество вина при слепом тестировании. В физике эта задача нужна для обнаружения функциональных зависимостей между переменными (регрессионный анализ).

2) Задача классификации – получение категориального ответа на основе набора признаков. Имеет конечное количество ответов: есть ли на фотографии кот, является ли изображение человеческим лицом.

3) Задача кластеризации – распределение данных на группы: разделение всех клиентов мобильного оператора по уровню платёжеспособности, отнесение космических

объектов к той или иной категории (планета, звёзда, чёрная дыра и т. п.). По сути это тоже разновидность классификации, но главное отличие кластеризации от классификации состоит в том, что перечень групп четко не задан и определяется в процессе работы алгоритма, поэтому для решения задач кластеризации используются совсем другие методы решения, чем в пункте (2).

4) Задача уменьшения размерности – сведение большого числа признаков к меньшему (обычно 2–3 или несколько) для удобства их последующей визуализации (например, сжатие данных). Для этой задачи часто используют матричные и тензорные разложения, которые в физике твёрдого тела и квантовой физике обычно называют Matrix product states (MPS).

5) Задача выявления аномалий – отделение аномалий от стандартных случаев. На первый взгляд она совпадает с задачей классификации, но есть одно существенное отличие: аномалии – явление редкое, поэтому многие методы классификации здесь не работают. На практике такой задачей является, например, выявление мошеннических действий с банковскими картами. В обработке данных физических экспериментов это нужно для того, чтобы отделять промахи в измерениях от прочих погрешностей.

Проблема выбора модели.

Как видно из вышеизложенного, целью обработки и анализа данных является создание моделей, которые описывают эти данные, либо проверка уже имеющихся моделей. Существует много разных классификаций моделей в зависимости от того, какими свойствами они обладают и для какой цели предназначены. Здесь мы обсудим проблему выбора модели, то есть вопрос о том, какую модель данных считать хорошей, а какую плохой.

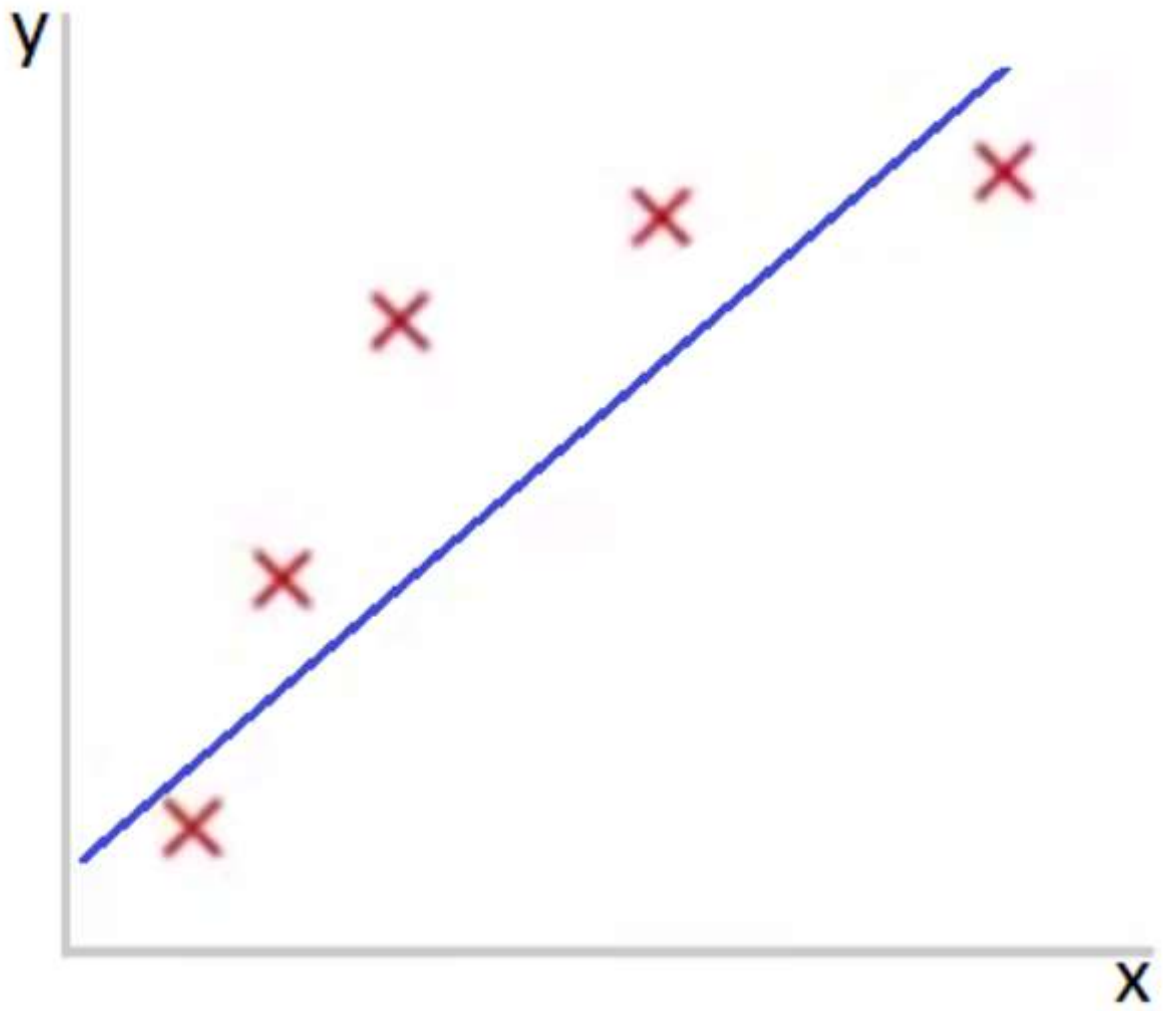
Для этого любую модель можно рассматривать в терминах машинного обучения как алгоритм, который обучается на входных данных, которые должен описывать (машинное обучение - это наука, которая изучает обучающиеся алгоритмы, которые должны определять и находить закономерности в данных). В связи с этим можно выделить две противоположные проблемы, успешное решение которых характеризует хорошую модель:

1) Переобучение (англ. overfitting) — негативное явление, возникающее, когда алгоритм обучения вырабатывает предсказания, которые слишком близко или точно соответствуют конкретному набору данных и поэтому не подходят для применения алгоритма к дополнительным данным или будущим наблюдениям.

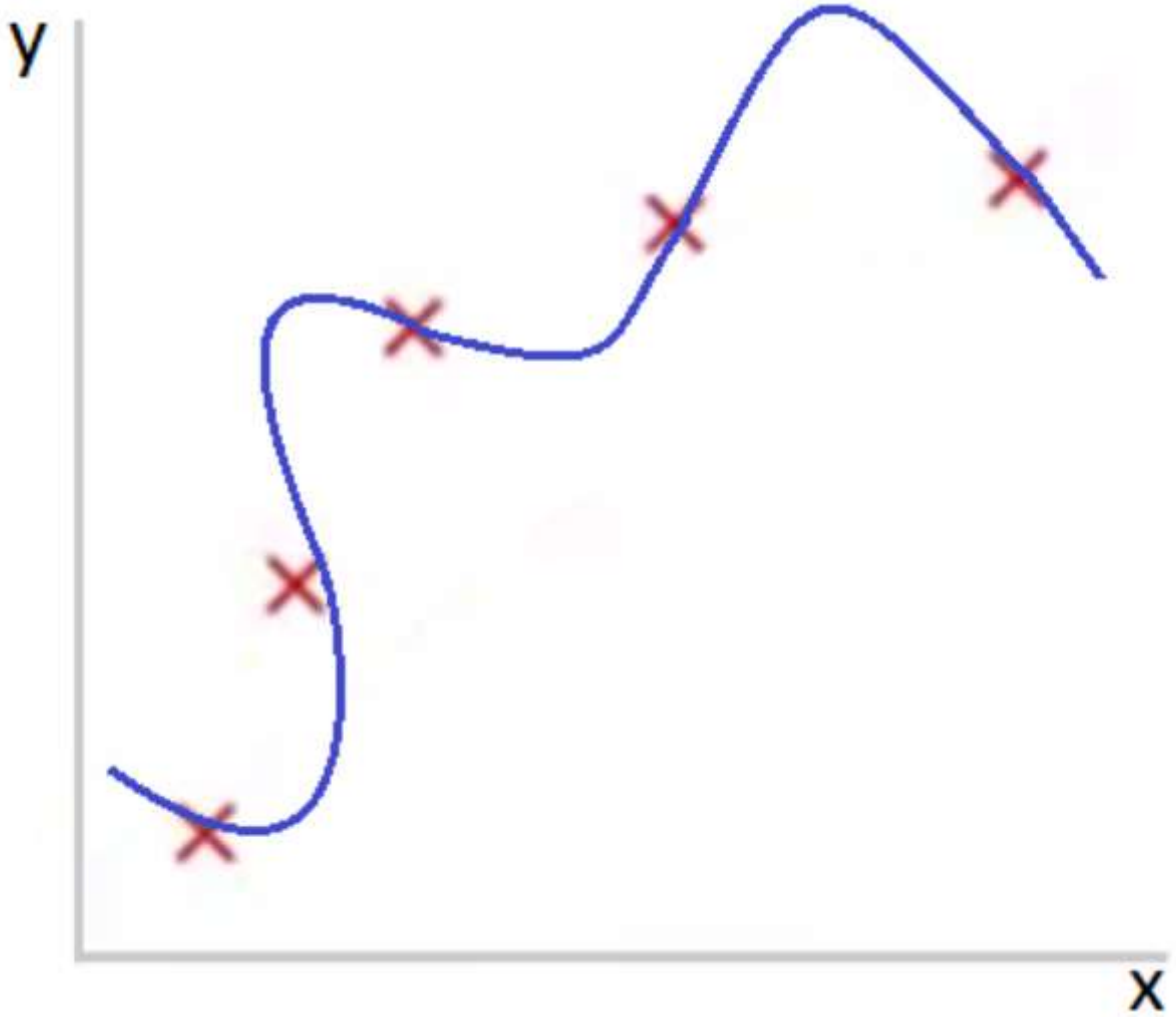
2) Недообучение (англ. underfitting) — негативное явление, при котором алгоритм обучения не обеспечивает достаточно малой величины средней ошибки на обучающей выборке. Недообучение возникает при использовании недостаточно сложных моделей.

Для объяснения сути проблемы проще всего проиллюстрировать конкретными примерами.

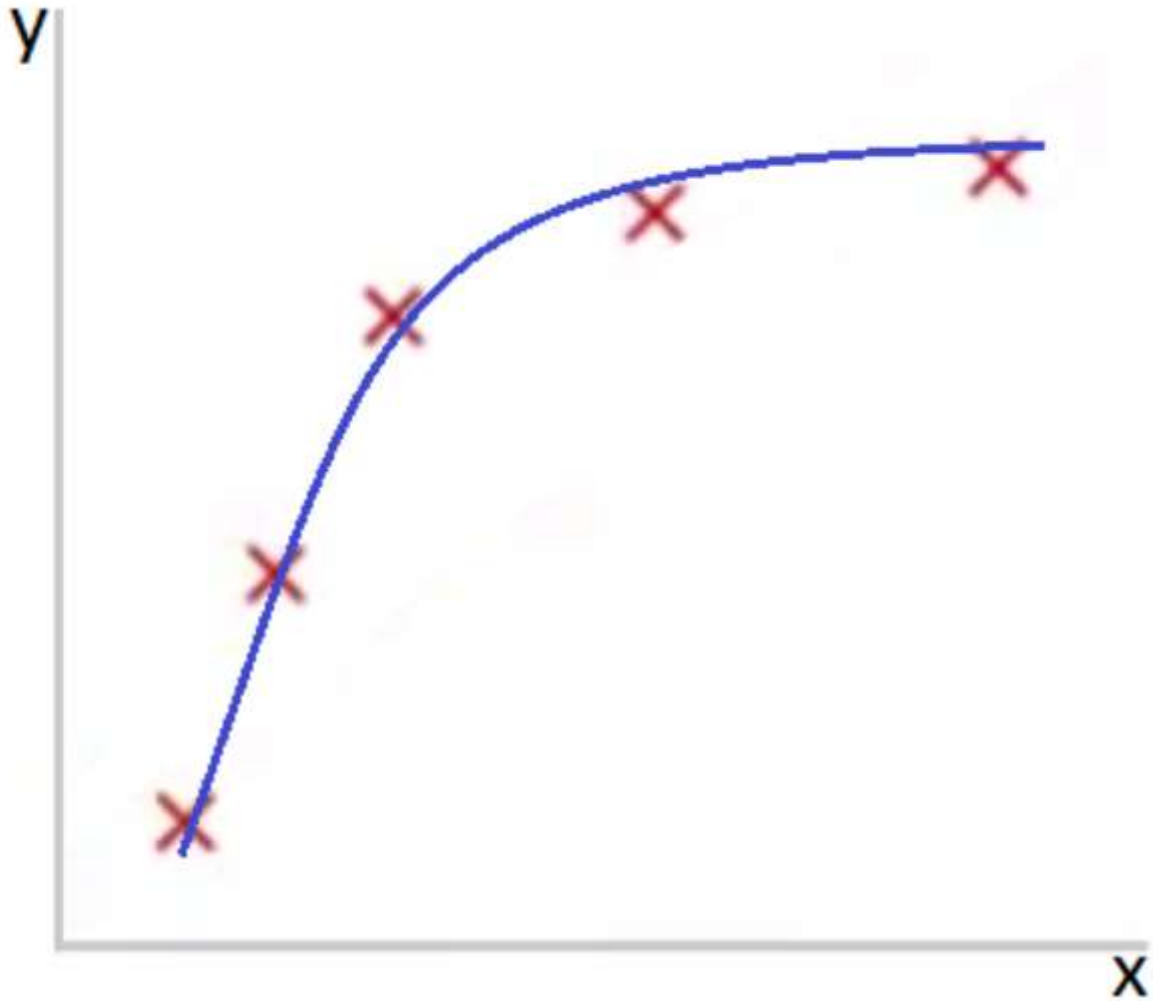
Недообучение



Переобучение



Хорошая модель.



Для оценки качества моделей используется кроссвалидация (Cross-validation, перекрёстная прове́рка, скользящий контроль) — метод оценки модели и её поведения на независимых данных. При оценке модели имеющиеся в наличии данные разбиваются на k частей. Затем на k−1 частях данных производится обучение модели, а оставшаяся часть данных используется для тестирования. Перекрёстная проверка используется как в data science, так и в анализе обычных экспериментов в естественных науках.

Основы теории погрешностей.

Любая вычисляемая или измеряемая величина определяется с погрешностью. Это значит, что приближённое значение величины, вычисляемое на компьютере, отличается от истинного.

Погрешности бывают абсолютные и относительные. Абсолютная погрешность - это максимальное отклонение от истинного значения, взятое по модулю. Относительная погрешность - вычисленное в процентах.

Погрешности прежде всего разделяют на устранимые и неустранимые. Неустранимыми называют погрешности, вызванные неопределённостью входных данных. Устранимые

погрешности бывают двух видов: погрешности округлений при вычислениях на компьютере и погрешности вычислительных методов.

Погрешности округлений при компьютерных вычислениях.

Представление вещественных чисел в памяти ПК

Вещественные числа обычно представляются в виде чисел с плавающей запятой. Числа с плавающей запятой — один из возможных способов представления действительных чисел, который является компромиссом между точностью и диапазоном принимаемых значений, его можно считать аналогом экспоненциальной записи чисел. Число с плавающей запятой состоит из набора отдельных двоичных разрядов, условно разделенных на так называемые знак (англ. sign), порядок (англ. exponent) и мантиссу (англ. mantis). В стандарте IEEE 754 число с плавающей запятой представляется в виде набора битов, часть из которых кодирует собой мантиссу числа, другая часть — показатель степени, и ещё один бит используется для указания знака числа (0 — если число положительное, 1 — если число отрицательное). При этом порядок записывается как целое число в коде со сдвигом, а мантисса — в нормализованном виде своей дробной частью в двоичной системе счисления. Порядок и мантисса — целые числа, которые вместе со знаком дают представление числа с плавающей запятой в виде:

(−1)^ S × M × B^E,

где S — знак, B — основание, E — порядок, а M — мантисса.

Нормальной формой (англ. normal form) числа с плавающей запятой называется такая форма, в которой мантисса (без учёта знака) в десятичной системе находится на полуинтервале [0;1). Эта форма неоднозначна, поэтому используют нормализованную (англ. normalized), в которой мантисса десятичного числа принимает значения от 1 (включительно) до 10 (не включительно), а мантисса двоичного числа принимает значения от 1 (включительно) до 2 (не включительно). То есть в мантиссе слева от запятой до применения порядка находится ровно один знак. В такой форме любое число (кроме 0) записывается единственным образом. Ноль же представить таким образом невозможно, поэтому стандарт предусматривает специальную последовательность битов для задания числа 00 (а заодно и некоторых других полезных чисел, таких как −∞ и +∞).

IEEE 754-2008 Standard for Floating-Point Arithmetic определяет представление и операции для чисел с плавающей точкой в компьютерных системах. Рассматривает форматы хранения, правила арифметики (в том числе и правила округления), стандартные и расширенные функции для типов одинарной (single), двойной (double), расширенной (extended) и расширяемой (extendable) точности, а также рекомендует форматы для обмена данными. В рамках используемых форматов определяет:

как представлять нормализованные и денормализованные положительные и отрицательные числа с плавающей запятой;

как представлять специальные величины «плюс бесконечность» и «минус бесконечность» (±Infinity, ±∞), ноль;

исключительные ситуации: деление на ноль, переполнение, потеря значимости - результат операции становится настолько близким к нулю, что порядок числа выходит за пределы разрядной сетки, работа с денормализованными числами и другие;

как представлять специальные величины «Не число» (NaN, not a number).

Неопределенность или NaN (от not a number) – это представление, придуманное для того, чтобы арифметическая операция могла всегда вернуть какое-то не бессмысленное значение. В IEEE754 NaN представлен как число, в котором E=Emax+1, а мантисса не нулевая. Любая операция с NaN возвращает NaN. При желании в мантиссу можно записывать информацию, которую программа сможет интерпретировать. Стандартом это не оговорено и мантисса чаще всего игнорируется.

Как можно получить NaN? Одним из следующих способов:

∞+(- ∞);

0 × ∞ ;

0/0;

∞/∞ ;

sqrt(x), где x<0.

По определению NaN ≠ NaN, поэтому, для проверки значения переменной нужно просто сравнить ее с собой. Два вида NaN: тихий NaN (qNaN) и сигнализационный NaN (sNaN). NaN может нести полезную нагрузку, предназначенный для диагностической информации, указывающей источник, вызвавший NaN. Знак NaN не имеет никакого значения, но может быть предсказуемым в некоторых случаях.

Денормализованные числа (англ. denormalized/subnormal numbers) - это способ увеличить количество представимых числом с плавающей запятой значений около нуля, дабы повысить точность вычислений. Каждое значение денормализованного числа меньше самого маленького нормализованного ("обычного") значения числа с плавающей запятой. Согласно стандарту, если порядок равен своему минимальному значению (все его биты — нули, а истинное значение порядка равно его сдвигу) и все биты мантиссы равны нулю, то это ±0. Если же мантисса не равна нулю, то это число с порядком, на единицу большим минимального (все биты порядка, кроме младшего — нули) и данной мантиссой, целая часть которой считается равной нулю, а не единице.

В IEEE 754-2008 денормализованные числа (denormal или denormalized numbers) были переименованы в subnormal numbers, то есть в числа, меньшие "нормальных". Поэтому их еще называют "субнормальными".

Погрешность округлений.

В системе счисления с основанием p (обычно p=2) и числом разрядов t число x можно представить в виде

$$x = \pm p^s \cdot \left(\frac{a_1}{p} + \cdots + \frac{a_t}{p^t}\right) = \pm p^s \cdot M$$

M – **мантисса** , s – **порядок**. Тогда

$\delta(x) = p^{1-t}$ – погрешность округления.

1) Ошибки, связанные с точностью представления вещественных чисел в формате IEEE754.

2) Ошибки, связанные с неправильным приведением типов данных.

3) Ошибки, вызванные сдвигом мантисс. Циклические дыры. Эти ошибки связаны с потерей точности результата при неполном пересечении мантисс чисел на числовой оси. Если мантиссы чисел не пересекаются на числовой оси, то операции сложения и вычитания между ними невозможны. Правило (a+b) + c = (a+c) + b обычно не выполняется в машинной арифметике, так как погрешность зависит от того, в каком порядке складывать.

4) Ошибки, вызванные округлением. Грязный ноль – когда переменная, которая должна быть равна нулю, не равна нулю из-за округления, которое делает процессор – вследствие этого относительная погрешность результата может достигать бесконечности.

5) Ошибки на границе норма/денорма числа (числа убийцы). Эти ошибки возникают при работе с числами находящимися на границе нормализованного/денормализованного представления чисел. Они связаны с различием в представления чисел в формате IEEE754 и в различии формул перевода формата IEEE754 в вещественные числа. То есть устройства (или программы) должны применять различные алгоритмы в зависимости от положения вещественного числа на числовой оси формата.

Кроме того, что это приводит к усложнению устройств и алгоритмов, ещё возникает неопределённость переходной зоны. Неопределённость переходной зоны заключается в том, что стандарт не определяет конкретного значения границы перехода. По сути

дела граница перехода находится между двумя вещественными числами: последним денормализованным числом 000FFFFFFFFFFFF и первым нормализованным числом 0010000000000000.

Так как граница является вещественным числом, то её точность можно задавать до бесконечности и цифровому устройству или программе может не хватить разрядности для принятия решения, к какому диапазону отнести число. Для примера можно привести баг https://bugs.php.net/bug.php?id=53632 , который вызвал панику в 2011-м.

Ввод числа 2.2250738585072011e-308 вызывал зависание процесса со 100 % загрузкой процессора. Сообщение о баге поступило 30.12.2010, исправлено было 10.01.2011.

Так как PHP препроцессор используют большинство серверов, то у любого пользователя сети в эти 10 дней была возможность вырубить большую часть информационные ресурсов планеты, используя данную уязвимость.

С ошибками округления связано огромное количество крупных аварий, например:

– Взрыв ракеты "Пэтриот» в Саудовской Аравии 25 февраля 1991, который привел к гибели 28 человек, связан с ошибками округления.

– Взрыв ракеты Ариан-5 сразу после старта при ее первом испытании во Французской Гвиане 4 июня 1996 был следствием переполнения числовой сетки компьютера.

– 23 августа 1991 в Гандсфиорде в Норвегии затонула нефтяная платформа, что привело к убытку почти в один миллиард долларов – из-за ошибок округления при выполнении конечно-элементного анализа при моделировании.

Погрешности вычислительных методов.

В вычислительной математике изучают погрешности используемых методов. Многие из них сводятся к оценкам остаточного члена в ряде Тейлора. Например:

Погрешность метода

$$\Delta f(x_1, x_2, \dots, x_n) \le \sum_{i=1}^{n} \left|\frac{\partial f(x_1, x_2, \dots, x_n)}{\partial x_i}\right| \Delta x_i$$

$$f'(x) \approx \frac{f(x+h)-f(x)}{h} \to \Delta f' = \frac{h \cdot f''}{2} + o(h)$$

$$f'(x) \approx \frac{f(x+h)-f(x-h)}{2 \cdot h} \to \Delta f' = \frac{h^2 \cdot f'''}{6} + o(h^2)$$

$$f(x_0, a_1, \dots, a_n) = 0 \to \frac{\partial f}{\partial x_0} \Delta x_0 \le \sum_{i=1}^{n} \left|\frac{\partial f}{\partial a_i}\right| \Delta a_i$$

Примеры решения теоретических задач.

Пример 1

Пусть x_0 — однократный действительный корень полинома $P_n(x) = a_n x^n + \dots + a_0$, у которого все коэффициенты a_j известны с достаточно малой погрешностью Δ. Какова погрешность корня x_0?

Решение

Так как корень x_0 изолированный, при небольшом приращении коэффициентов $a_j \to a_j + \Delta a_j$ корень $x_0 \to x_0 + \Delta x_0$. Используя уравнения $P_n(x_0, \{a_j\}) = 0$ и $P_n(x_0 + \Delta x_0, \{a_j + \Delta a_j\}) = 0$ и малость приращений $\Delta a_j \le \Delta$, $j = 0, \dots, n$ и Δx_0, имеем $0 = \Delta P \approx (n a_n x_0^{n-1} + \dots + a_1)\Delta x_0 + \Delta a_n x_0^n + \dots + \Delta a_0$. Отсюда

$$|\Delta x_0| \le \frac{|\Delta a_n x_0^n| + \dots + |\Delta a_0|}{|(n a_n x_0^{n-1} + \dots + a_1)|} \le \frac{\left(|x_0|^n + \dots + 1\right)\Delta}{|(n a_n x_0^{n-1} + \dots + a_1)|}.$$

Пример 2

Пусть производная функции $f'(x)$ приближенно вычисляется в точке x_0 на ЭВМ по формуле: $f'(x_0) \approx (f(x_0+h)-f(x_0))/h$, причем значения функции $f(x)$ сами имеют погрешности округления, связанные с конечностью мантиссы. Почему в этом случае не следует брать слишком малый h? Оцените оптимальный шаг h, предположив, что $f''(x)$ существует и непрерывна, $|f''(x)| \le M_2$ на отрезке $[x_0, x_0+h]$ и длина мантиссы равна t.

Решение

Применяя формулу Тейлора, получим

$$f'(x_0)-(f(x_0+h)-f(x_0))/h = -hf''(\xi)/2,$$

где $\xi \in [x_0, x_0+h]$. Погрешность округления при вычислении $f(x)$ можно грубо оценить как $2^{1-t}|f(x_0)|$. Таким образом, полная погрешность по модулю не превосходит $hM_2/2 + 2^{-t+2} f(x_0)/h \ge 2\sqrt{M_2 2^{1-t}|f(x_0)|}$, и оптимальный шаг $h_0 = 2\sqrt{2^{1-t}|f(x_0)|/M_2}$.

Пример 3

Пусть задана последовательность чисел x_n, $n = 0, 1, 2, \ldots$, причем $x_{n+1} + 100x_n = 101$. При $x_0 = 1$ все $x_n = 1$. Если допустить маленькую погрешность в задании x_0, например, $x_0 = 1 + 10^{-6}$, то погрешность x_n будет быстро возрастать с ростом n: $x_1 = 1 - 10^{-4}$, $x_2 = 1 + 10^{-2}$, $x_3 = 1 - 1 = 0$, $x_2 = 1 + 10^2$, … При выборе $x_0 = -1$ имеем: $x_1 = 201$, $x_2 = -19999$, … Но теперь погрешность в начальных данных уже не будет так сильно портить результат вычислений: $x_1 = 201 - 10^{-4}$, $x_2 = -19999 + 10^{-2}$, ... Объясните кажущийся парадокс.

Решение

Для соотношения $x_{n+1} + 100x_n = 101$ нетрудно записать решение для произвольного n в общем виде: $x_n = C(-100)^n + 1$, где константа C определяется из начальных данных. Теперь видно, что $x_0 = 1$ — это специальный случай, когда x_n не растут с ростом n. Небольшое отклонение приводит к быстро возрастающей относительной погрешности, тогда как для другого случая относительная погрешность почти не меняется.

Теоретическая задача для самостоятельного решения (10 вариантов).

В этой задаче требуется найти аналитическое решение и проверить его с помощью вычислений на Python.

1.1. Чему равна погрешность в определении действительного корня $x=1$ уравнения $ax^4+bx^3+dx+e=0$, если $a=1\pm10^{-3}$, $b=1\pm10^{-3}$, $d=-1\pm10^{-3}$, $e=-1\pm10^{-3}$?

1.2. Чему равна погрешность в определении корней уравнения $ax^3+bx^2=0$, если $a=1\pm10^{-3}$, $b=-4\pm10^{-3}$?

1.3. С каким числом верных знаков (или относительной погрешностью) должен быть известен свободный член в уравнении $x^2-2x+0.999993751=0$, чтобы корни имели четыре верных знака?

1.4. С каким числом верных знаков (или относительной погрешностью) должен быть известен свободный член в уравнении $x^2-4x+3.999901=0$, чтобы корни имели четыре верных знака?

1.5. Определить оптимальный шаг $h=\text{const}$ формулы численного дифференцирования $f'(x-h)\approx\big(f(x)-f(x-h)\big)/h$, $\max\limits_{[x-h,x]}\left|f''(x)\right|\le100$, если абсолютная погрешность при задании $f(x)$, $f(x-h)$ не превосходит $\Delta=0.1$.

1.6. Определить оптимальный шаг $h=\text{const}$ формулы численного дифференцирования $f'(x)\approx\big(f(x+h)-f(x-h)\big)/2h$, $\max\limits_{[x-h,x+h]}\left|f'''(x)\right|\le100$, если абсолютная погрешность при задании, $f(x\pm h)$ не превосходит $\Delta=0.1$.

1.7. Определить оптимальный шаг $h = \text{const}$ формулы численного дифференцирования $f'(x) \approx \left(3f(x) - 4f(x-h) + f(x-2h)\right)/2h$, $\max\limits_{[x-2h,x]} \left|f'''(x)\right| \le 100$, если абсолютная погрешность при задании $f(x)$, $f(x-h)$, $f(x-2h)$ не превосходит $\Delta = 0.1$.

1.8. Пусть приближенное значение первой производной функции $f(x)$ определяется при $h \ll 1$ по формуле $f'(x) \approx \left(3f(x) - 4f(x-h) + f(x-2h)\right)/2h$, а сами значения $f(x), f(x-h), f(x-2h)$ вычисляются с абсолютной погрешностью Δ. Какую погрешность можно ожидать при вычислении производной, если $\left|f^{(k)}(x)\right| \le M_k$, $k = 1, 2, \ldots$?

1.9. Пусть задана последовательность чисел x_n, $n = 0,\ 1,\ 2,\ \ldots$, причем $x_{n+1} - 5x_n = 4$, а x_0 известно с относительной погрешностью 10^{-6}. При каких значениях x_0 относительная погрешность при вычислении x_n будет быстро возрастать с ростом n?

1.10. Пусть задана последовательность чисел x_n, $n = 0,\ 1,\ 2,\ \ldots$, причем $5x_{n+1} - x_n = 4$, а x_0 известно с относительной погрешностью 10^{-6}. При каких значениях x_0 относительная погрешность при вычислении x_n будет быстро возрастать с ростом n?

Задача визуализации данных на Python для самостоятельного решения (10 вариантов).

Для выполнения этой задачи можно использовать файл номер 6 по основам Python от кафедры дискретной математики МФТИ или галерею примеров Matplotlib. https://matplotlib.org/2.0.2/gallery.html

Для начала сгенерируем какую-нибудь случайную выборку. Например, нормальное распределение, среднее значение равно 0, среднеквадратичное отклонение равно 3.

```
import numpy as np
import scipy.stats as sps
sample = sps.norm.rvs(size=200, loc=0, scale=3)
print('Первые 10 значений выборки:\n', sample[:10])
print('Выборочное среденее: %.3f' % sample.mean())
print('Выборочная дисперсия: %.3f' % sample.var())
```

```
Первые 10 значений выборки:
 [ 5.10622092 -5.06145095  4.08193931 -0.26208612  2.85289238  2.69353436
 -1.43923249  8.46684354 -0.43383022 -1.98011577]
Выборочное среденее: -0.052
Выборочная дисперсия: 9.076
```

Пример генерации биноминального распределения с параметрами n и p

```
sample = sps.binom.rvs(size=200, n=10, p=0.6)
print('Первые 10 значений выборки:\n', sample[:10])
print('Выборочное среденее: %.3f' % sample.mean())
print('Выборочная дисперсия: %.3f' % sample.var())
```

```
Первые 10 значений выборки:
 [6 7 9 6 7 7 4 7 7 5]
Выборочное среденее: 5.730
Выборочная дисперсия: 2.317
```

Описание работы с numpy и scipy есть в файле номер 5, а также можно смотреть документацию https://docs.scipy.org/doc/scipy/reference/stats.html

Пусть X — некоторое распределение с параметрами params

`X.rvs(size=N, params)` — генерация выборки размера (Random VariateS). Возвращает `numpy.array`

`X.cdf(x, params)` — значение функции распределения в точке (Cumulative Distribution Function)

`X.logcdf(x, params)` — значение логарифма функции распределения в точке

`X.ppf(q, params)` — -квантиль (Percent Point Function)

`X.mean(params)` — математическое ожидание

`X.median(params)` — медиана

`X.var(params)` — дисперсия (Variance)

`X.std(params)` — стандартное отклонение = корень из дисперсии (Standard Deviation)

Кроме того для непрерывных распределений определены функции

`X.pdf(x, params)` — значение плотности в точке (Probability Density Function)

`X.logpdf(x, params)` — значение логарифма плотности в точке

А для дискретных

`X.pmf(k, params)` — значение дискретной плотности в точке (Probability Mass Function)

`X.logpdf(k, params)` — значение логарифма дискретной плотности в точке

Параметры могут быть следующими:

`loc` — параметр сдвига

`scale` — параметр масштаба

и другие параметры (например, n и p для биномиального)

Предлагается взять какую-нибудь функцию (например, синус, но можно взять любую другую) и добавить к ней это распределение как небольшой шум. После этого нарисовать график этой суммы заданной функции и шума. Перед началом выполнения задания напишите

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
%matplotlib inline
```

Последняя строчка нужна для встраивания графиков в блокноты Jupyter notebook.

Варианты заданий:

Для шума с нормальным распределением:

2.1) нарисовать график в логарифмическом масштабе по обеим осям

2.2) нарисовать график в полярных координатах

2.3) нарисовать гистограмму

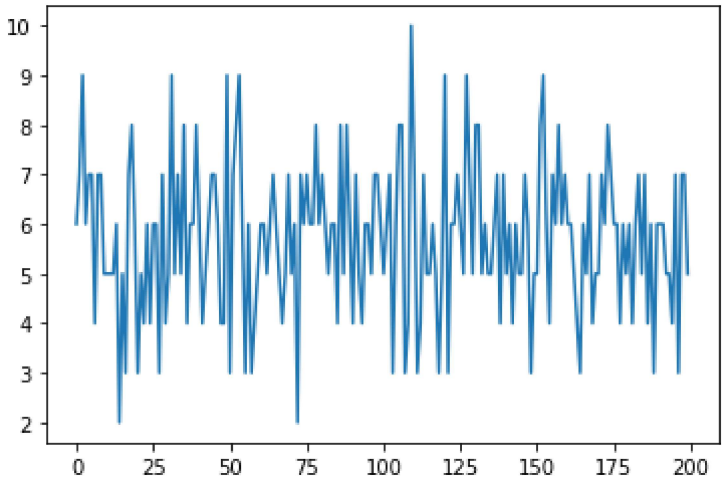
2.4) нарисовать поверхность (функция двух переменных)

2.5) нарисовать трёхмерную линию (независимая переменная - параметр t, а координаты функции от неё).

То же самое для шума с биноминальным распределением - варианты 2.6-2.10 соответственно.

Пример графика шума (если координаты x не указывать, то они образуют последовательность 0, 1, 2, ...).

```
plt.figure()
plt.plot(sample)
plt.show()
```



✓ 0 сек. выполнено в 06:58