Введение в анализ данных

Домашнее задание 1. `Numpy`, `matplotlib`, `scipy.stats`

Правила:

- Дедлайн **25 марта 23:59**. После дедлайна работы не принимаются кроме случаев наличия уважительной причины.
- Выполненную работу нужно отправить на почту `mipt.stats@yandex.ru`, указав тему письма `"[номер группы] Фамилия Имя - Задание 1"`. Квадратные скобки обязательны.
- Прислать нужно ноутбук и его pdf-версию (без архивов). Названия файлов должны быть такими: `1.N.ipynb и 1.N.pdf`, где `N` -- ваш номер из таблицы с оценками. *pdf-версию можно сделать с помощью Ctrl+P. Пожалуйста, посмотрите ее полностью перед отправкой. Если что-то существенное не напечатается в pdf, то баллы могут быть снижены.*
- Решения, размещенные на каких-либо интернет-ресурсах, не принимаются. Кроме того, публикация решения в открытом доступе может быть приравнена к предоставлении возможности списать.
- Для выполнения задания используйте этот ноутбук в качестве основы, ничего не удаляя из него.
- Пропущенные описания принимаемых аргументов дописать на русском.
- Если код будет не понятен проверяющему, оценка может быть снижена.

Баллы за задание:

Легкая часть (достаточно на "хор"):

- Задача 1.1 -- 3 балла
- Задача 1.2 -- 3 балла
- Задача 2 -- 3 балла

Сложная часть (необходимо на "отл"):

- Задача 1.3 -- 3 балла
- Задача 3.1 -- 3 балла
- Задача 3.2 -- 3 балла
- Задача 3.3 -- 3 балла
- Задача 4 -- 4 балла

Баллы за разные части суммируются отдельно, нормируются впоследствии также отдельно. Иначе говоря, 1 балл за легкую часть может быть не равен 1 баллу за сложную часть.

In [1]:
```
import numpy as np
import scipy.stats as sps

import matplotlib.pyplot as plt
import matplotlib.cm as cm
from mpl_toolkits.mplot3d import Axes3D
import ipywidgets as widgets

import typing

%matplotlib inline
```

Легкая часть: генерация

В этой части другие библиотеки использовать запрещено. Шаблоны кода ниже менять нельзя.

Задача 1

Имеется симметричная монета. Напишите функцию генерации независимых случайных величин из нормального и экспоненциального распределений с заданными параметрами.

In [2]:
```
import seaborn as sns
sns.set(palette='Set2')
```

С этой библиотекой правда лучше гистограмма выглядит

In [3]:
```
# Эта ячейка -- единственная в задаче 1, в которой нужно испо
# библиотечную функцию для генерации случайных чисел.
# В других ячейках данной задачи используйте функцию coin.

# симметричная монета
coin = sps.bernoulli(p=0.5).rvs
```

Проверьте работоспособность функции, сгенерировав 10 бросков симметричной монеты.

In [4]:
```
coin(size=10)
```

Out[4]:
```
array([0, 0, 0, 0, 0, 0, 1, 1, 1, 0])
```

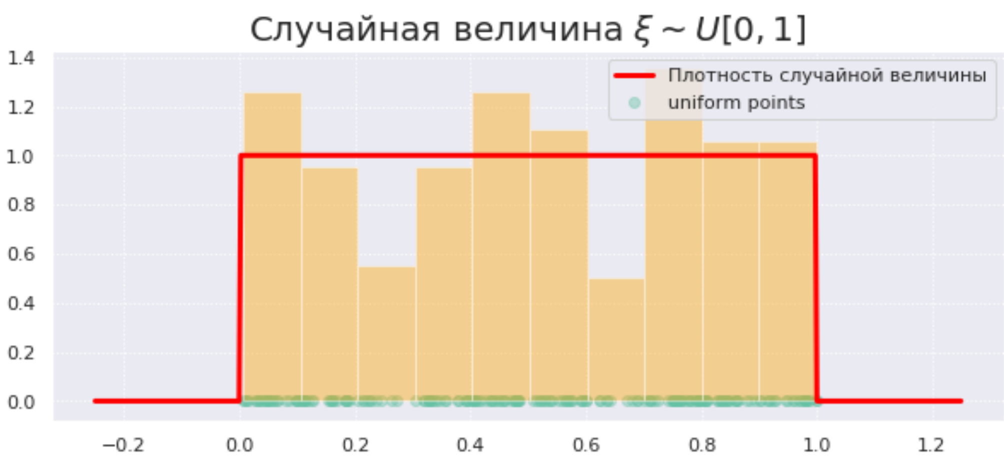
Часть 1. Напишите сначала функцию генерации случайных величин из равномерного распределения на отрезке $[0, 1]$ с заданной точностью. Это можно сделать, записав случайную величину $\xi \sim U[0, 1]$ в двоичной системе системе счисления $\xi = 0, \xi_1\xi_2\xi_3\ldots$. Тогда $\xi_i \sim Bern(1/2)$ и независимы в совокупности. Приближение заключается в том, что вместо генерации бесконечного количества ξ_i мы полагаем $\xi = 0, \xi_1\xi_2\xi_3\ldots\xi_n$.

Нужно реализовать функцию так, чтобы она могла принимать на вход в качестве параметра `size` как число, так и объект `tuple` любой размерности, и возвращать объект `numpy.array` соответствующей размерности. Например, если `size=(10, 1, 5)`, то функция должна вернуть объект размера $10 \times 1 \times 5$. Кроме того, функцию `coin` можно вызвать только один раз, и, конечно же, не использовать какие-либо циклы. Аргумент `precision` отвечает за число n.

In [5]:
```
def uniform(size=1, precision=30):
    A = coin(precision * np.prod(size)).reshape(np.prod(size)
    B = 2. ** np.arange(-1, - precision - 1, -1)
    return (A @ B).reshape(size)
```

Для $U[0, 1]$ сгенерируйте 200 независимых случайных величин, постройте график плотности на отрезке $[-0.25, 1.25]$, а также гистограмму по сгенерированным случайным величинам.

In [6]:
```
size = 200
grid = np.linspace(-0.25, 1.25, 500)

sample = uniform(size, 50)

# Отрисовка графика
plt.figure(figsize=(10, 4))

# отображаем значения случайных величин полупрозрачными точка
plt.scatter(
    sample,
    np.zeros(size),
    alpha=0.4,
    label='uniform points'
)

# по точкам строим нормированную полупрозрачную гистограмму
plt.hist(
    sample,
    bins=10,
    density=True,
    alpha=0.4,
    color='orange'
)

# рисуем график плотности
plt.plot(
    grid,
    sps.uniform.pdf(grid),
    color='red',
    lw=3,
    label='Плотность случайной величины'
)
plt.legend()
plt.title(r'Случайная величина $\xi\sim U[0, 1]$', fontsize=2
plt.grid(ls=':')
plt.show()
```



Исследуйте, как меняются значения случайных величин в зависимости от `precision` .

In [7]:
```
size = 100

plt.figure(figsize=(15, 3))

for i, precision in enumerate([1, 2, 3, 5, 10, 30]):
    plt.subplot(3, 2, i + 1)
    plt.scatter(
        uniform(size, precision),
        np.zeros(size),
        alpha=0.4
    )
    plt.yticks([])
    if i < 4: plt.xticks([])

plt.show()
```

Вывод: Видно, что с ростом точности, с которой мы генерируем числа в промежутке $[0, 1]$, плотность точек стремится к плотности стандартного непрерывного равномерного распределения.

Часть 2. Напишите функцию генерации случайных величин в количестве `size` штук (как и раньше, тут может быть `tuple`) из распределения $\mathcal{N}(loc, scale^2)$ с помощью преобразования Бокса-Мюллера, которое заключается в следующем. Пусть ξ и η -- независимые случайные величины, равномерно распределенные на (0, 1]. Тогда случайные величины $X = cos(2\pi\xi)\sqrt{-2\ln\eta}, Y = sin(2\pi\xi)\sqrt{-2\ln\eta}$ являются независимыми нормальными $\mathcal{N}(0, 1)$.

Реализация должна быть без циклов. Желательно использовать как можно меньше бросков монеты.

In [8]:
```
def normal(size=1, loc=0, scale=1, precision=30):
    A = uniform(size, precision)
    B = uniform(size, precision)
    return np.sin(2 * np.pi * A) * np.sqrt(-2 * np.log(B))
```

Для $\mathcal{N}(0, 1)$ сгенерируйте 200 независимых случайных величин, постройте график плотности на отрезке $[-3, 3]$, а также гистограмму по сгенерированным случайным величинам.

In [9]:
```
size = 200
grid = np.linspace(-3, 3, 1000)

sample = normal(size, loc=0, scale=1, precision=50)

# Отрисовка графика
plt.figure(figsize=(16, 7))

# по точкам строим нормированную полупрозрачную гистограмму
plt.hist(
    sample,
    bins=20,
    density=True,
    alpha=0.4,
    label='Гистограмма случайной величины'
)

# рисуем график плотности
plt.plot(
    grid,
    sps.norm.pdf(grid),
    color='red',
    lw=3,
    label='Плотность случайной величины'
)
plt.title(r'Случайная величина $\xi \sim \mathcal{N}$(0, 1)',
plt.legend(fontsize=14, loc=1)
#plt.grid(ls=':')
plt.show()
```

Сложная часть: генерация

Часть 3. Вы уже научились генерировать выборку из равномерного распределения. Напишите функцию генерации выборки из экспоненциального распределения, используя из теории вероятностей:

Если ξ --- случайная величина, имеющая абсолютно непрерывное распределение, и F --- ее функция распределения, то случайная величина $F(\xi)$ имеет равномерное распределение на $[0, 1]$.

Какое преобразование над равномерной случайной величиной необходимо совершить?

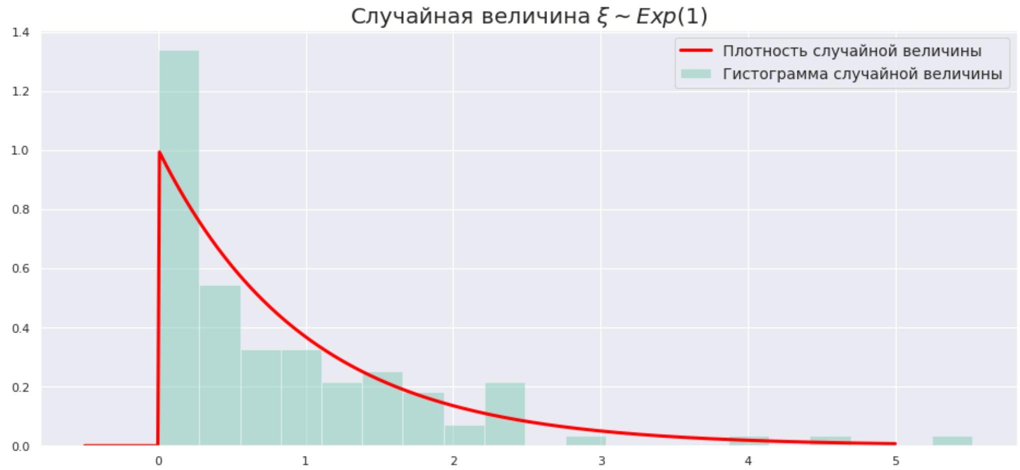
Воспользуемся методом обратного преобразования.

Для получения полного балла реализация должна быть без циклов, а параметр `size` может быть типа `tuple` .

In [10]:
```
def expon(size=1, lambd=1, precision=30):
    return - (1 / lambd) * np.log(uniform(size, precision))
```

Для $Exp(1)$ сгенерируйте выборку размера 100 и постройте график плотности этого распределения на отрезке $[-0.5, 5]$.

In [11]:
```
size = 100
grid = np.linspace(-0.5, 5, 500)

sample = expon(size, lambd=1, precision=50)

# Отрисовка графика
plt.figure(figsize=(16, 7))

# по точкам строим нормированную полупрозрачную гистограмму
plt.hist(
    sample,
    bins=20,
    density=True,
    alpha=0.4,
    label='Гистограмма случайной величины'
)

# рисуем график плотности
plt.plot(
    grid,
    sps.expon.pdf(grid),
    color='red',
    lw=3,
    label='Плотность случайной величины'
)
plt.title(r'Случайная величина $\xi \sim Exp(1)$', fontsize=2
plt.legend(fontsize=14, loc=1)
#plt.grid(ls=':')
plt.show()
```



Вывод по задаче:

С помощью генератора выборки из стандартного непрерывного равномерного распределения можно получить генератор выборки любого непрерывного распределения.

Легкая часть: матричное умножение

Задача 2

Напишите функцию, реализующую матричное умножение. При вычислении разрешается создавать объекты размерности три. Запрещается пользоваться функциями, реализующими матричное умножение (`numpy.dot` , операция `@` , операция умножения в классе `numpy.matrix`). Разрешено пользоваться только простыми векторно-арифметическими операциями над `numpy.array` , а также преобразованиями осей. *Авторское решение занимает одну строчку.*

In [12]:
```
def matrix_multiplication(A, B):
    return np.array([a_row * b_row for a_row in A for b_row i
```

Проверьте правильность реализации на случайных матрицах. Должен получится ноль.

In [13]:
```
A = sps.uniform.rvs(size=(10, 20))
B = sps.uniform.rvs(size=(20, 30))
np.abs(matrix_multiplication(A, B) - A @ B).sum()
```

Out[13]:
```
1.2567724638756772e-13
```

На основе опыта: вот в таком стиле многие из вас присылали бы нам свои работы, если не стали бы делать это задание :)

In [14]:
```
def stupid_matrix_multiplication(A, B):
    C = [[0 for j in range(len(B[0]))] for i in range(len(A))
    for i in range(len(A)):
        for j in range(len(B[0])):
            for k in range(len(B)):
                C[i][j] += A[i][k] * B[k][j]
    return C
```

Проверьте, насколько быстрее работает ваш код по сравнению с неэффективной реализацией `stupid_matrix_multiplication` . Эффективный код должен работать почти в 200 раз быстрее. Для примера посмотрите также, насколько быстрее работают встроенные `numpy` -функции.

In [15]:
```
A = sps.uniform.rvs(size=(400, 200))
B = sps.uniform.rvs(size=(200, 300))

%time C1 = matrix_multiplication(A, B)
%time C2 = A @ B  # python 3.5
%time C3 = np.matrix(A) * np.matrix(B)
%time C4 = stupid_matrix_multiplication(A, B)
%time C5 = np.einsum('ij,jk->ik', A, B)
```

```
CPU times: user 268 ms, sys: 959 ms, total: 1.23 s
Wall time: 1.23 s
CPU times: user 2.4 ms, sys: 0 ns, total: 2.4 ms
Wall time: 1.18 ms
CPU times: user 1.11 ms, sys: 395 µs, total: 1.51 ms
Wall time: 999 µs
CPU times: user 18.2 s, sys: 27.7 ms, total: 18.2 s
Wall time: 18 s
CPU times: user 8.65 ms, sys: 37 µs, total: 8.69 ms
Wall time: 8.5 ms
```

Ниже для примера приведена полная реализация функции. Вас мы, конечно, не будем требовать проверять входные данные на корректность, но документации к функциям нужно писать.

In [17]:
```
def matrix_multiplication(A, B):
    '''Возвращает матрицу, которая является результатом
    матричного умножения матриц A и B.

    '''

    # Если A или B имеют другой тип, нужно выполнить преобраз
    A = np.array(A)
    B = np.array(B)

    # Проверка данных входных данных на корректность
    assert A.ndim == 2 and B.ndim == 2, 'Размер матриц не рав
    assert A.shape[1] == B.shape[0], \
        ('Матрицы размерностей {} и {} неперемножаемы'.format

    C = np.array([a_row * b_row for a_row in A for b_row in B

    return C
```

Сложная часть: броуновское движение

Задача 3

Познавательная часть задачи (не пригодится для решения задачи)

Абсолютное значение скорости движения частиц идеального газа, находящегося в состоянии ТД-равновесия, есть случайная величина, имеющая распределение Максвелла и зависящая только от одного термодинамического параметра — температуры T.

В общем случае плотность вероятности распределения Максвелла для n-мерного пространства имеет вид:

$$p(v) = \mathrm{C}\, e^{-\frac{mv^2}{2kT}}\, v^{n-1},$$

где $v \in [0, +\infty)$, а константа C находится из условия нормировки $\int\limits_0^{+\infty} p(v)\mathrm{d}v = 1.$

Физический смысл этой функции таков: вероятность того, что скорость частицы входит в промежуток $[v_0, v_0 + \mathrm{d}v]$, приближённо равна $p(v_0)\mathrm{d}v$ при достаточно малом $\mathrm{d}v$. Тут надо оговориться, что математически корректное утверждение таково:

$$\lim_{\mathrm{d}v \to 0} \frac{\mathsf{P}\{v \mid v \in [v_0, v_0 + \mathrm{d}v]\}}{\mathrm{d}v} = p(v_0).$$

Поскольку это распределение не ограничено справа, определённая доля частиц среды приобратает настолько высокие скорости, что при столкновении с макрообъектом может происходить заметное отклонение как траектории, так и скорости его движения.

Мы предполагаем идеальность газа, поэтому компоненты вектора скорости частиц среды v_i можно считать независимыми нормально распределёнными случайными величинами, т.е.

$$v_i \sim \mathcal{N}(0, s^2),$$

где s зависит от температуры и массы частиц и одинаково для всех направлений движения.

При столкновении макрообъекта с частицами среды происходит перераспределение импульса в соответствии с законами сохранения энергии и импульса, но в силу большого числа подобных событий за единицу времени, моделировать их напрямую достаточно затруднительно. Поэтому для выполнения этого ноутбука сделаем следующие предположения:

- Приращение компоненты координаты броуновской частицы за фиксированный промежуток времени (или за шаг) Δt имеет вид $\Delta x_i \sim \mathcal{N}(0, \sigma^2)$.
- σ является конкретным числом, зависящим как от Δt, так и от параметров броуновской частицы и среды.

Задание

1. Разработать функцию симуляции броуновского движения

Функция должна вычислять приращение координаты частицы на каждом шаге как $\Delta x_{ijk} \sim \mathcal{N}(0, \sigma^2)\ \forall i, j, k$, где i — номер частицы, j — номер координаты, а k — номер шага. Функция принимает в качестве аргументов:

- Параметр σ;
- Количество последовательных изменений координат (шагов), приходящихся на один процесс;
- Число процессов для генерации (количество различных частиц);
- Количество пространственных измерений для генерации процесса.

Возвращаемое значение:

- 3-х мерный массив `result`, где `result[i,j,k]` — значение j-й координаты i-й частицы на k-м шаге.

Общее требование

- Считать, что все частицы в начальный момент времени находятся в начале координат.

Что нужно сделать

- Реализовать функцию для произвольной размерности, не используя циклы.
- Дописать проверки типов для остальных аргументов.

Обратите внимание на использование аннотаций для типов аргументов и возвращаемого значения функции. В новых версиях Питона подобные возможности синтаксиса используются в качестве подсказок для программистов и статических анализаторов кода, и никакой дополнительной функциональности не добавляют.

Например, `typing.Union[int, float]` означает `"или int, или float"`.

Что может оказаться полезным

- Генерация нормальной выборки: `scipy.stats.norm`. [Ссылка (https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.norm.html)](https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.norm.html)
- Кумулятивная сумма: метод `cumsum` у `np.ndarray`. [Ссылка (https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.cumsum.html)](https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.cumsum.html)

In [ ]:
```
def generate_brownian(sigma: typing.Union[int, float] = 1,
                      *,
                      n_proc: int = 10,
                      n_dims: int = 2,
                      n_steps: int = 100) -> np.ndarray:
    """
    :param sigma:    стандартное отклонение нормального распр
                     генерирующего пошаговые смещения координ
    :param n_proc:   <ДОПИСАТЬ>
    :param n_dims:   <ДОПИСАТЬ>
    :param n_steps:  <ДОПИСАТЬ>

    :return:         np.ndarray размера (n_proc, n_dims, n_st
                     на позиции [i,j,k] значение j-й координа
                     на k-м шаге.
    """
    if not np.issubdtype(type(sigma), np.number):
        raise TypeError("Параметр 'sigma' должен быть числом"
    # <ДОПИСАТЬ ПРОВЕРКИ ТИПОВ>

    return <...>
```

Символ `*` в заголовке означает, что все аргументы, объявленные после него, необходимо определять только по имени.

Например,

```
generate_brownian(323, 3)           # Ошибка
generate_brownian(323, n_steps=3)   # OK
```

При проверке типов остальных аргументов, по аналогии с `np.number`, можно использовать `np.integer`. Конструкция `np.issubdtype(type(param), np.number)` используется по причине того, что стандартная питоновская проверка `isinstance(sigma, (int, float))` не будет работать для `numpy`-чисел `int64, int32, float64` и т.д.

In [ ]:
```
brownian_2d = generate_brownian(2, n_steps=12000, n_proc=500,
assert brownian_2d.shape == (500, 2, 12000)
```

2. Визуализируйте траектории для 9-ти первых броуновских частиц

Что нужно сделать

- Нарисовать 2D-графики для `brownian_2d` .
- Нарисовать 3D-графики для `brownian_3d = generate_brownian(2, n_steps=12000, n_proc=500, n_dims=3)` .

Общие требования

- Установить соотношение масштабов осей, равное 1, для каждого из подграфиков.

Что может оказаться полезным

- [Туториал (https://matplotlib.org/devdocs/gallery/subplots_axes_and_figures/subplots_demo.html)](https://matplotlib.org/devdocs/gallery/subplots_axes_and_figures/subplots_demo.html) по построению нескольких графиков на одной странице.
- Метод `plot` у `AxesSubplot` (переменная `ax` в цикле ниже).
- Метод `set_aspect` у `AxesSubplot` .

In [ ]:
```
fig, axes = plt.subplots(3, 3, figsize=(18, 10))
fig.suptitle('Траектории броуновского движения', fontsize=20)

for ax, (xs, ys) in zip(axes.flat, brownian_2d):
    <...>
    pass
```

3. Постройте график среднего расстояния частицы от начала координат в зависимости от времени (шага)

- Постройте для `n_dims` от 1 до 5 включительно.
- Кривые должны быть отрисованы на одном графике. Каждая кривая должна иметь легенду.
- Для графиков подписи к осям обязательны.

Вопросы

- Как вы думаете, какой функцией может описываться данная зависимость?
- Сильно ли её вид зависит от размерности пространства?
- Можно ли её линеаризовать? Если да, нарисуйте график с такими же требованиями.

In [ ]:
```
plt.figure(figsize=(12, 6))

for n_dims in range(1, 6):
    plt.plot(
        <...>
        label=f'Размерность: {n_dims}'
    )

plt.ylabel('Ср. раст. частицы от нач. координат')
plt.xlabel('Шаг')
plt.legend(loc='best')
plt.tight_layout()
plt.show()
```

Сложная часть: визуализация распределений

Задача 4

В этой задаче вам нужно исследовать свойства дискретных распределений и абсолютно непрерывных распределений.

Для перечисленных ниже распределений нужно
1) На основе графиков дискретной плотности (функции массы) для различных параметров пояснить, за что отвечает каждый параметр.
2) Сгенерировать набор независимых случайных величин из этого распределения и построить по ним гистограмму.
3) Сделать выводы о свойтсвах каждого из распределений.

Распределения:

- Бернулли
- Биномиальное
- Равномерное
- Геометрическое

Для выполнения данного задания можно использовать код с лекции.

In [ ]:
```
<...>
```