

# Математическая статистика

## Практическое задание 1

В данном задании рассматриваются различные способы генерации выборки из некоторых стандартных распределений, а так же рассматриваются некоторые свойства эмпирической функции распределения и ядерной оценки плотности.

### Правила:

- Баллы за каждую задачу указаны далее. Если сумма баллов за задание меньше 25%, то все задание оценивается в 0 баллов.
- Выполненную работу нужно отправить на почту `probability.diht@yandex.ru`, указав тему письма "[номер группы] Фамилия Имя - Задание 1". Квадратные скобки обязательны.
- Прислать нужно ноутбук и его pdf-версию. Названия файлов должны быть такими: `1.N.ipynb` и `1.N.pdf`, где N - ваш номер из таблицы с оценками.
- Никакой код из данного задания при проверке запускаться не будет.

### Баллы за задание:

- Задача 1 - 13 баллов
- Задача 2 - 3 балла
- Задача 3 - 5 баллов
- Задача 4 - 3 балла
- Задача 5 - 2 балла
- Задача 6 - 1 балл
- Задача 7 - 3 балла

```
In [2]: import numpy as np
import scipy.stats as sps
import matplotlib.pyplot as plt

%matplotlib inline
```

**Задача 1.** Имеется симметричная монета. С ее помощью напишите функцию генерации выборки из многомерного нормального распределения с заданными параметрами.

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

Для получения максимального балла реализовать функцию нужно так, чтобы она могла принимать на вход в качестве параметра `size` 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.

```
In [3]: coin = sps.bernoulli(0.5).rvs # симметричная монета
# coin(size=10) --- реализация 10 бросков монеты

def uniform(size=1, precision=30):
    return np.apply_along_axis( # Применяем вдоль оси
        # Берём наше двоичное представление, домножаем на нужные степени, а потом суммируем, чтобы получить
        # число
        lambda x: np.sum(x*np.power(0.5, np.arange(1, precision+1, 1))),
        # Номер оси, вдоль которой суммировать
        1 if type(size) is int else len(size),
        coin(
            size=(
                # делаем tuple из элемента, если int, и просто возвращаем, если tuple
                ((size,) if type(size) is int else size)
                # Добавляем новую размерность
                + (precision,)
            )
        )
    ) # В одну строчку
```

```
In [4]: # Тестируем для tuple
uniform((3, 2, 5), 5)
```

```
Out[4]: array([[[ 0.65625,  0.59375,  0.0625 ,  0.        ,  0.21875],
                 [ 0.        ,  0.8125 ,  0.46875,  0.84375,  0.21875]],

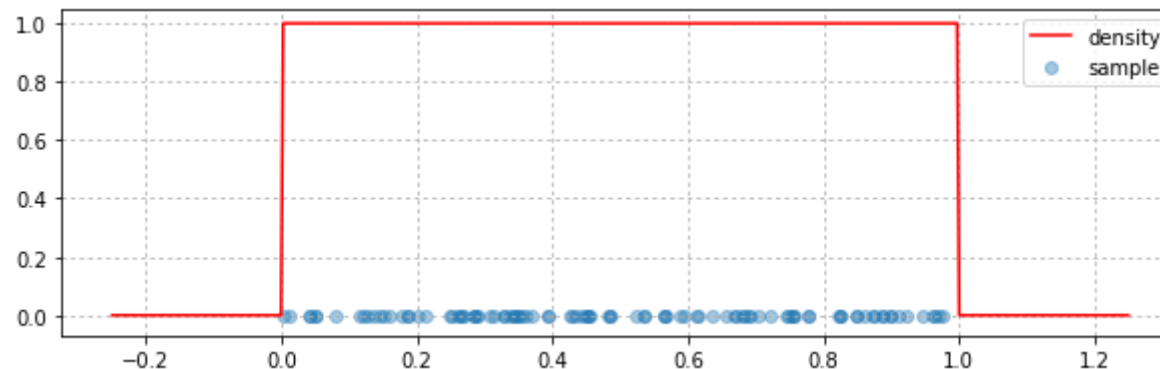
               [[ 0.34375,  0.71875,  0.75    ,  0.5      ,  0.28125],
                 [ 0.84375,  0.375   ,  0.0625 ,  0.09375,  0.        ]],

               [[ 0.65625,  0.125   ,  0.5625 ,  0.9375 ,  0.75    ],
                 [ 0.46875,  0.65625,  0.5     ,  0.53125,  0.75    ]]])
```

Для  $U[0, 1]$  сгенерируйте выборку и постройте график плотности.

```
In [5]: size = 100
grid = np.linspace(-0.25, 1.25, 500)

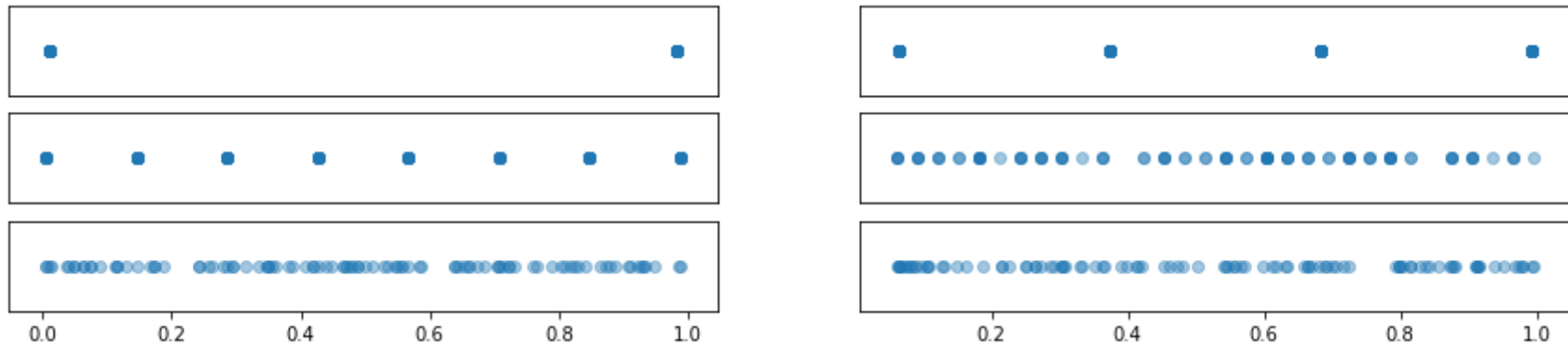
plt.figure(figsize=(10, 3))
plt.scatter(uniform(size, 50),
            np.zeros(size), alpha=0.4, label='sample')
plt.plot(grid,
         sps.uniform.pdf(grid),
         color='red', label='density')
plt.legend()
plt.grid(ls=':')
plt.show()
```



Исследуйте, как меняется выборка в зависимости от precision.

```
In [6]: 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:** При помощи случайных величин с бернуллиевским распределением можно приблизить случайную величину с равномерным распределением, и при больших значениях precision это приближение будет хорошим

Часть 2. Напишите функцию генерации выборки размера size (как и раньше, тут может быть tuple) из распределения  $\mathcal{N}(loc, scale^2)$  с помощью преобразования Бокса-Мюллера (задача 7.12 из книги по теории вероятностей).

Для получения полного балла реализация должна быть без циклов.

```
In [7]: def normal(size=1, loc=0, scale=1, precision=30):
        size = ((size,) if type(size) is int else size)
        x = uniform(size, precision)
        y = uniform(size, precision)
        # 1-у, чтобы не было логарифмов от 0 (наше uniform никогда не возвращает 1, но может вернуть 0)
        standard_normal = np.cos(2*np.pi*x)*np.sqrt(-2*np.log(1-y))
        return scale*standard_normal-loc
```

```
In [8]: # Тестируем для tuple
        normal((3, 2, 5), 2, 2, 5)
```

```
Out[8]: array([[[-1.15716927, -3.42873704, -2.59744906, -7.16436187, -2.79570865],
                [ 1.07709188, -4.5363512 , -1.04511289, -5.12439625, -3.56083865]],

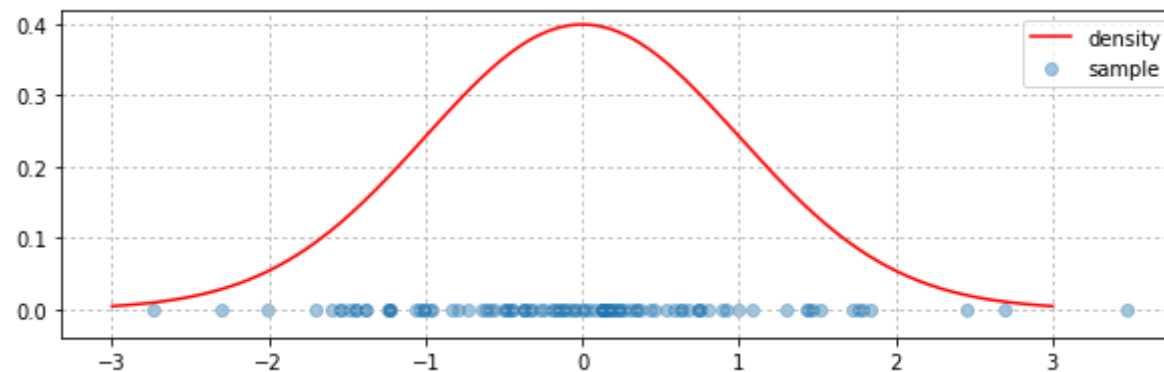
               [[-2.          , -2.73786573, -1.82687225, -0.04203869, -0.25909713],
                [-1.7258387 ,  0.27459922, -1.08865075, -2.          , -4.88405377]],

               [[-2.          , -0.73861467, -3.69594382, -3.7914758 , -1.26916105],
                [-2.94216984, -1.35229176, -2.          , -2.37829563, -5.07672066]])])
```

Для  $\mathcal{N}(0, 1)$  сгенерируйте выборку и постройте график плотности этого распределения на отрезке  $[-3, 3]$ .

```
In [9]: size = 100
grid = np.linspace(-3, 3, 500)

plt.figure(figsize=(10, 3))
plt.scatter(normal(size, precision=50),
            np.zeros(size), alpha=0.4, label='sample')
plt.plot(grid,
         sps.norm.pdf(grid),
         color='red', label='density')
plt.legend()
plt.grid(ls=':')
plt.show()
```



Пусть  $P$  — некоторое распределение на  $(\mathbb{R}, \mathcal{B}(\mathbb{R}))$ . Числа  $a$  и  $\sigma$  называются параметрами сдвига и масштаба соответственно для семейства распределений  $\{P_{a,\sigma} \mid a \in \mathbb{R}, \sigma \in \mathbb{R}_+\}$ , где  $P_{a,\sigma}(B) = P\left(\frac{B-a}{\sigma}\right)$  и  $\frac{B-a}{\sigma} = \left\{\frac{x-a}{\sigma} \mid x \in B\right\}$

**Вопрос:** Найдите плотность  $P_{a,\sigma}$ , если  $P$  имеет плотность  $p(x)$ .

**Решение:**  $F_{a,\sigma}(x) = P_{a,\sigma}(t)$

Здесь  $P\left(\frac{t-a}{\sigma}\right)$

**Вопрос:** Пусть  $P$  --- стандартное нормальное распределение. Выпишите параметрическое семейство распределений, параметризованное параметрами сдвига и масштаба по отношению к распределению  $P$ . Какая связь между параметрами и характеристиками распределения (например, математическое ожидание)?

Имеем:  $p_{a,\sigma}(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-a)^2}{2\sigma^2}} = \frac{p(\sigma x - a)}{\sigma}$  (из теории вероятностей прошлого семестра).

Параметрическое семейство распределений — это  $P_{a,\sigma}(B) = \{P(\frac{x-a}{\sigma}) \mid x \in B\}$ .

Матожидание нормального распределения — это  $a$ , а дисперсия —  $\sigma^2$ . Поэтому матожидание меняется при сдвиге, а дисперсия — при масштабировании.

Постройте на одном графике разными цветами плотности стандартного нормального распределения, а так же для параметров  $a = 3, \sigma = 1$  и  $a = 0, \sigma = 2$ . Интервал по оси икс  $[-7, 7]$ .

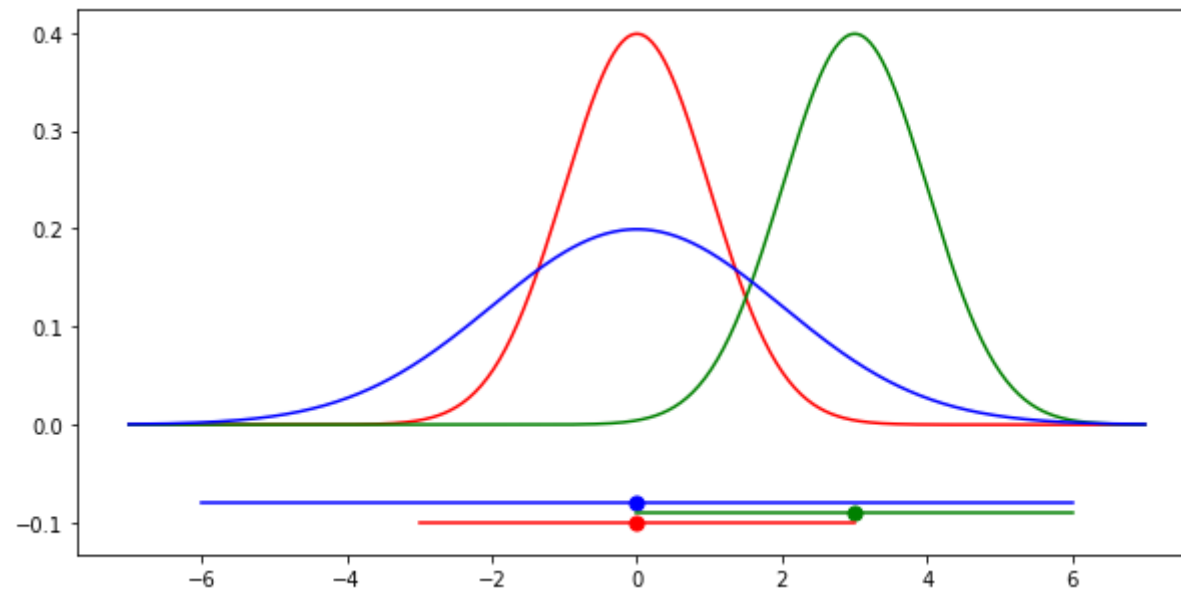
Ниже графика теми же цветами изобразите также точку  $a$  и  $3\sigma$ -интервал, используя шаблон, приведенный ниже.

```

In [10]: grid = np.linspace(-7, 7, 500)
plt.figure(figsize=(10, 5))

params = ((0,1,'red',0.00), (3,1,'green', 0.01), (0,2,'blue', 0.02))
for tup in params:
    a, sigma, color, shift = tup
    plt.plot(grid,
              sps.norm.pdf(grid, a, sigma),
              color=color, label='density')
    plt.plot([a - 3 * sigma, a + 3 * sigma], [-0.1+shift, -0.1+shift], color=color)
    plt.scatter(a, -0.1+shift, color=color, s=50)

```



**Вывод:** При помощи преобразования Бокса-Мюллера из равномерно распределённых случайных величин можно получить величину со стандартным нормальным распределением, а из неё — нормально распределённую с любыми параметрами.

*Часть 3.* Теперь напишите функцию генерации выборки из многомерного нормального распределения с заданным вектором средних `mean` и матрицей ковариаций `cov_matrix`. Помочь в этом может теорема об эквивалентных определениях гауссовского вектора. Для извлечения квадратного корня из матрицы может пригодиться следующая функция, которая вычисляет собственные значения и векторы матрицы.



```
In [11]: from scipy.linalg import eigh
```

На этот раз достаточно, чтобы функция корректно работала в случае, когда size является числом.

```
In [12]: def gauss(mean, cov_matrix, size=1, precision=30):  
    # Преобразование типов  
    mean = np.array(mean)  
    cov_matrix = np.array(cov_matrix)  
  
    # Проверка на корректность входа  
    assert mean.ndim == 1 and cov_matrix.ndim == 2  
    assert mean.shape[0] == cov_matrix.shape[0]  
    assert cov_matrix.shape[0] == cov_matrix.shape[1]  
  
    norm = normal((len(mean), size), precision=precision)  
    D, R = eigh(cov_matrix)  
    D = np.diagflat(D)  
  
    return (R.transpose()@np.sqrt(D)@norm).T+mean
```

Сгенерируйте выборку размера size из двумерного нормального распределения с нулевым вектором средних и матрицей ковариаций

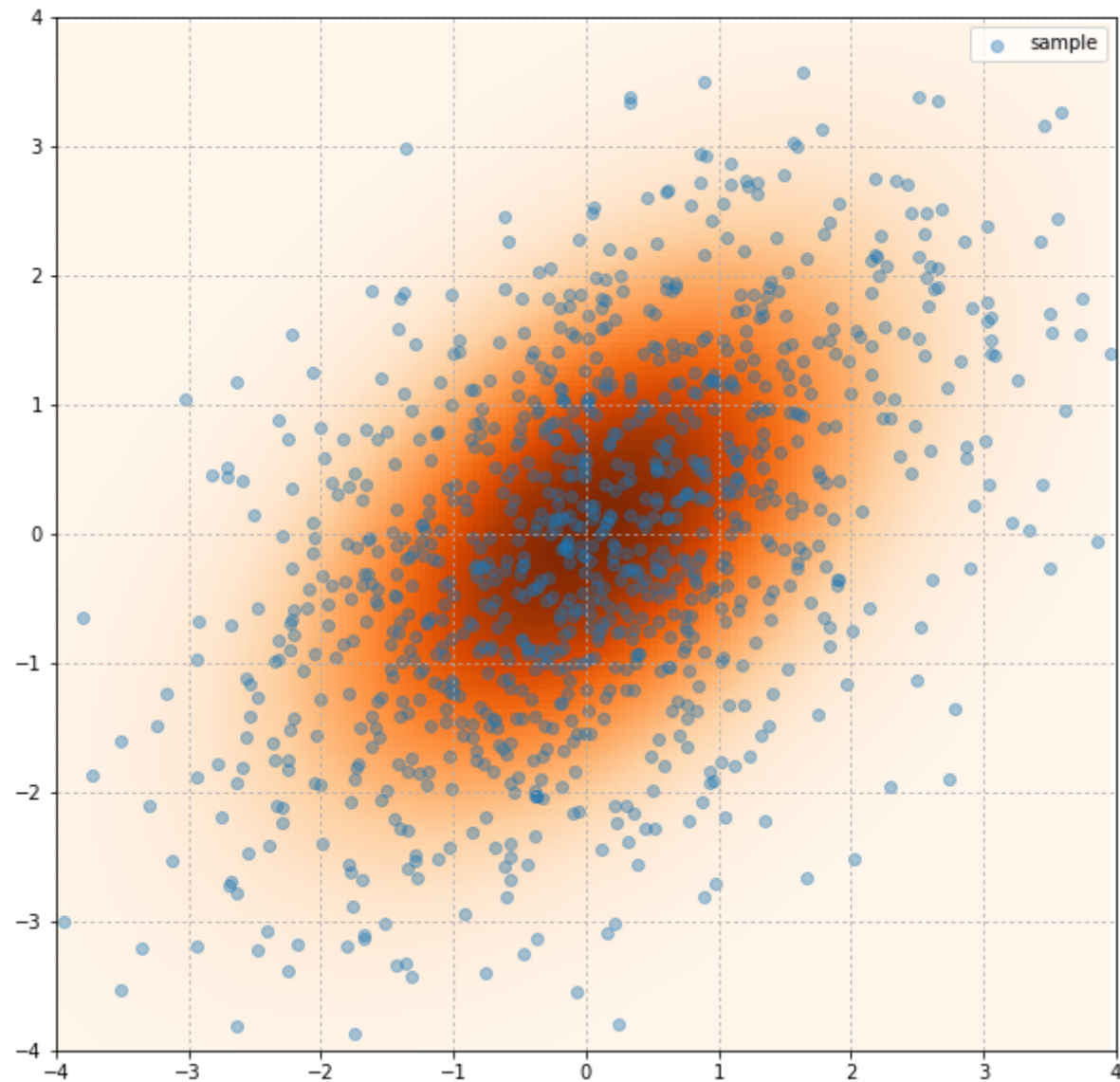
$\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$ . Нанесите точки выборки на график и отметьте цветом значение плотности.

В инструкциях по Питону плотность вычислялась с помощью неэффективного кода. Подумайте, как можно написать эффективный короткий код, не использующий циклы.

```
In [13]: size = 1000
mean = [0, 0]
cov = [[2, 1], [1, 2]]
sample = gauss(mean, cov, size) # Генерация выборки

grid = np.mgrid[-4:4:0.05, -4:4:0.05]
density = sps.multivariate_normal.pdf(np.dstack((grid[0], grid[1])), mean, cov) # Вычисление плотности

plt.figure(figsize=(10, 10))
plt.pcolormesh(grid[0], grid[1], density, cmap='Oranges')
plt.scatter(sample[:, 0], sample[:, 1], alpha=0.4, label='sample')
plt.legend()
plt.grid(ls=':')
plt.xlim((-4, 4))
plt.ylim((-4, 4))
plt.show()
```



**Вывод:** С помощью стандартно нормально распределённых величин можно получить многомерно нормально распределённую величину, т. е. гауссовский вектор. Опять же, чем лучше мы приближаем нормальные, тем лучше мы приближаем гауссовские векторы.

**Задача 2.** Вы уже научились генерировать выборку из равномерного распределения. Напишите функцию генерации выборки из экспоненциального распределения, используя результат задачи 6.9 из книги по теории вероятностей.

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

```
In [14]: def expon(size=1, lambd=1, precision=30):
         unif = uniform(size, precision)
         # 1-unif, чтобы не было логарифмов от 0 (наше uniform никогда не возвращает 1, но может вернуть 0)
         # Распределение от этого остаётся равномерным
         return -np.log(1-unif)/lambd
```

```
In [15]: # Тестируем для tuple
         expon((3, 2, 5), 4, 5)
```

```
Out[15]: array([[ 0.09367336,  0.14384104,  0.51986039,  0.31712783,  0.41849411],
                [ 0.04247476,  0.00793717,  0.00793717,  0.14384104,  0.01613463]],

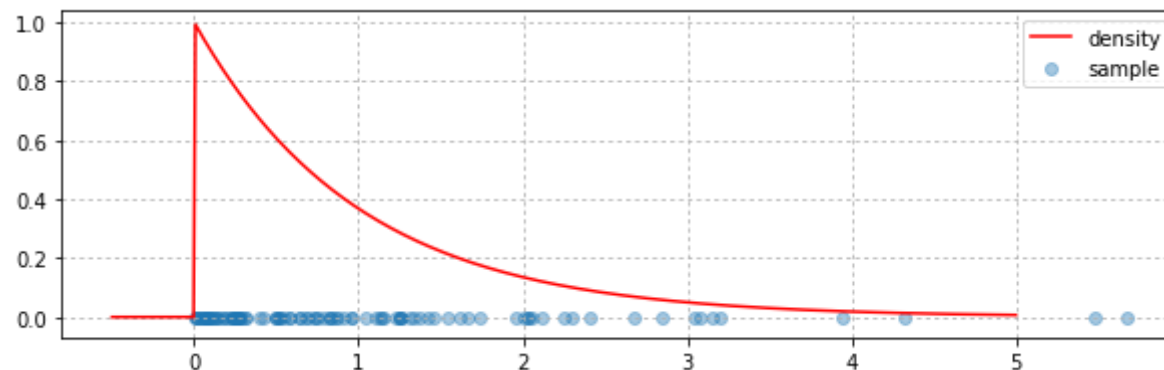
               [[ 0.18942143,  0.13032423,  0.2907877 ,  0.04247476,  0.03338285],
                [ 0.34657359,  0.05190984,  0.04247476,  0.03338285,  0.01613463]],

               [[ 0.02461002,  0.34657359,  0.14384104,  0.01613463,  0.37995644],
                [ 0.14384104,  0.26696016,  0.18942143,  0.41849411,  0.14384104]])
```

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

```
In [16]: size = 100
grid = np.linspace(-0.5, 5, 500)

plt.figure(figsize=(10, 3))
plt.scatter(expon(size, precision=50),
            np.zeros(size), alpha=0.4, label='sample')
plt.plot(grid,
         sps.expon.pdf(grid),
         color='red', label='density')
plt.legend()
plt.grid(ls=':')
plt.show()
```



**Вывод:** Из нормально распределённых величин можно сделать экспоненциально распределённые. Точность приближения — аналогично.

**Задача 3.** Для каждого распределения постройте эмпирическую функцию распределения (ЭФР), гистограмму и ядерную оценку плотности. Сделать это помогут следующие функции.

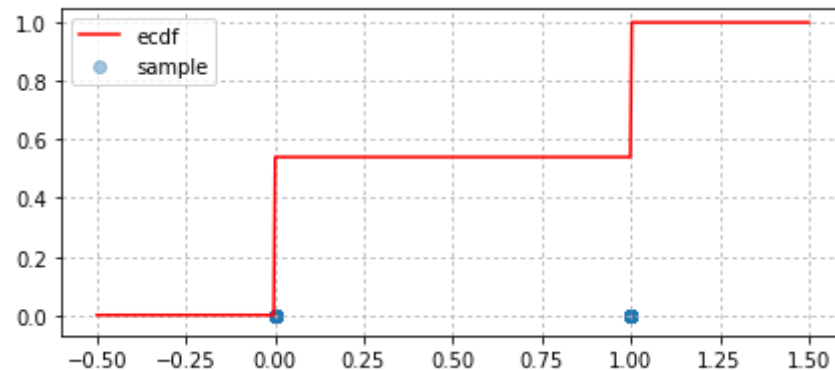
```
In [17]: from statsmodels.distributions.empirical_distribution import ECDF
         from statsmodels.nonparametric.kde import KDEUnivariate
```

## 1. Бернуллиевское.

Тут приведен пример построения ЭФР, просто запустите эту ячейку.

```
In [18]: sample = coin(size=100)
ecdf = ECDF(sample)
grid = np.linspace(-0.5, 1.5, 500)

plt.figure(figsize=(7, 3))
plt.scatter(sample, np.zeros(size), alpha=0.4, label='sample')
plt.plot(grid, ecdf(grid), color='red', label='ecdf')
plt.legend()
plt.grid(ls=':')
plt.show()
```



Далее, чтобы не копировать несколько раз один и тот же код, напомним некоторую функцию.

В третьей функции нужно построить ядерную оценку плотности, о которой будет рассказано на лекциях. В частности, формула была на презентации на первой лекции. Пример построения можно посмотреть тут

<http://statsmodels.sourceforge.net/0.6.0/generated/statsmodels.nonparametric.kde.KDEUnivariate.html>  
(<http://statsmodels.sourceforge.net/0.6.0/generated/statsmodels.nonparametric.kde.KDEUnivariate.html>)

```
In [19]: def draw_ecdf(sample, grid, cdf=None):  
        ''' По сетке grid строит графики эмпирической функции распределения  
        и истинной (если она задана) для всей выборки и для 1/10 ее части.
```

```

'''
plt.figure(figsize=(16, 3))
for i, size in enumerate([len(sample) // 10, len(sample)]):
    plt.subplot(1, 2, i + 1)

    plt.scatter(sample[:size], np.zeros(size),
                alpha=0.4, label='sample')

    if cdf is not None:
        plt.plot(grid,
                 cdf(grid),
                 color='green', alpha=0.3, lw=2, label='true cdf')

    plt.plot(grid,
             ECDF(sample[:size])(grid),
             color='red', label='ecdf')

    plt.legend()
    plt.grid(ls=':')
    plt.title('sample size = {}'.format(size))
plt.show()

def draw_hist(sample, grid, pdf=None):
    ''' Строит гистограмму и по сетке grid график истинной плотности
    (если она задана) для всей выборки и для 1/10 ее части.
    '''

    plt.figure(figsize=(16, 3))
    for i, size in enumerate([len(sample) // 10, len(sample)]):
        plt.subplot(1, 2, i + 1)
        plt.hist(sample[:size],
                20,
                range=(grid.min(), grid.max()),
                normed=True)

        if pdf is not None:
            plt.plot(grid,
                     pdf(grid),
                     color='green', alpha=0.3, lw=2)

    plt.show()

```



```

def draw_pdf(sample, grid, pdf=None):
    ''' По сетке grid строит графики ядерной оценки плотности
    и истинной плотности (если она задана) для всей выборки и для 1/10 ее части.
    '''

    plt.figure(figsize=(16, 3))
    for i, size in enumerate([len(sample) // 10, len(sample)]):
        plt.subplot(1, 2, i + 1)
        kernel_density = KDEUnivariate(sample[:size])
        kernel_density.fit()

        plt.scatter(sample[:size], np.zeros(size),
                     alpha=0.4, label='sample')

        if pdf is not None:
            plt.plot(grid,
                     pdf(grid),
                     color='green', alpha=0.3, lw=2, label='true pdf')

        plt.plot(grid,
                 kernel_density.evaluate(grid),
                 color='red', label='kde')

        plt.legend()
        plt.grid(ls=':')
    plt.show()

```

При использовании KDEUnivariate могут возникать разные проблемы. Можно попробовать их решить следующими способами:

**1.** В режиме суперюзера в файле `/usr/local/lib/python3.5/dist-packages/statsmodels/nonparametric/kdetools.py` замените строку 20 на

```
y = X[:int(m/2+1)] + np.r_[0,X[int(m/2+1):],0]*1j
```

В файле `/usr/local/lib/python3.5/dist-packages/statsmodels/nonparametric/kde.py` замените строку 327 на

```
nobs = len(X) # after trim
```

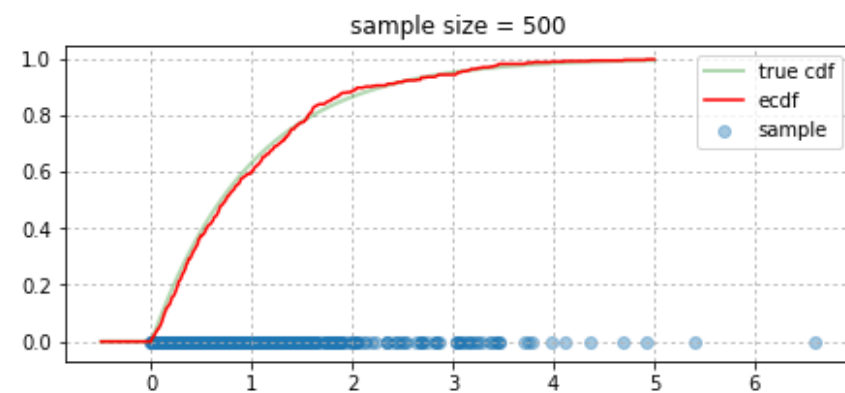
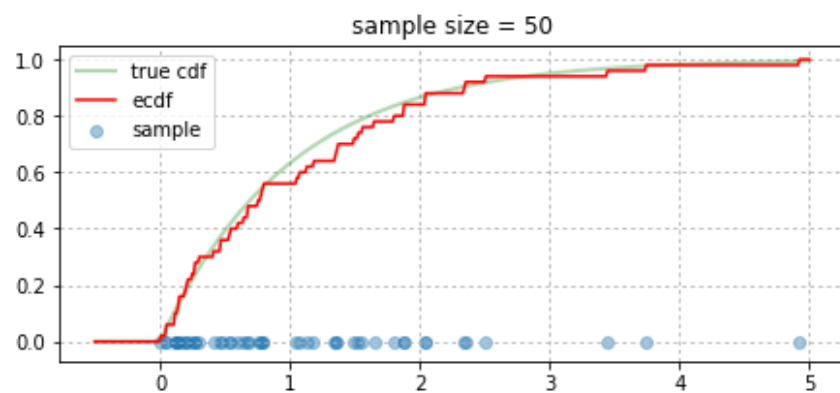
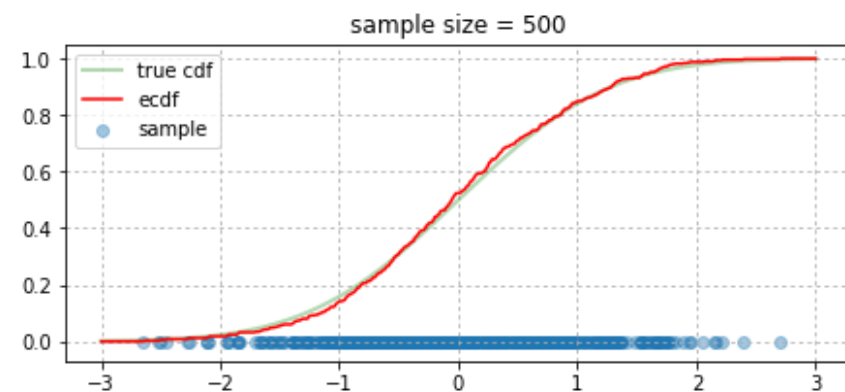
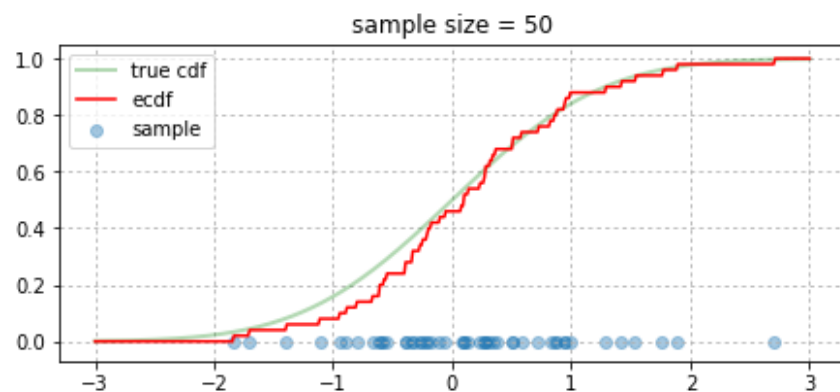
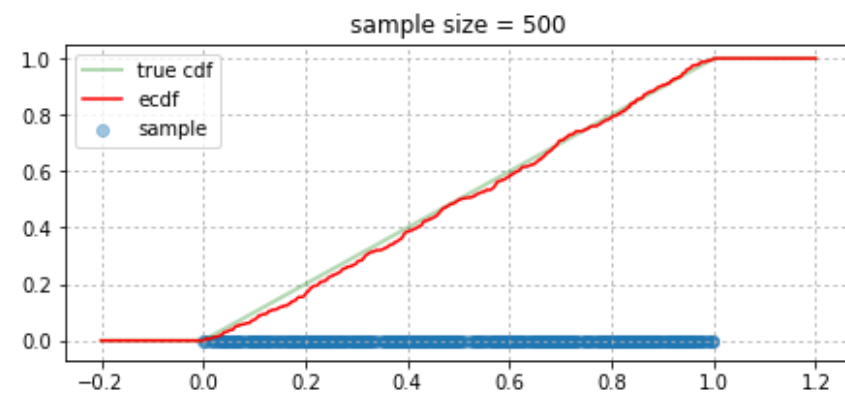
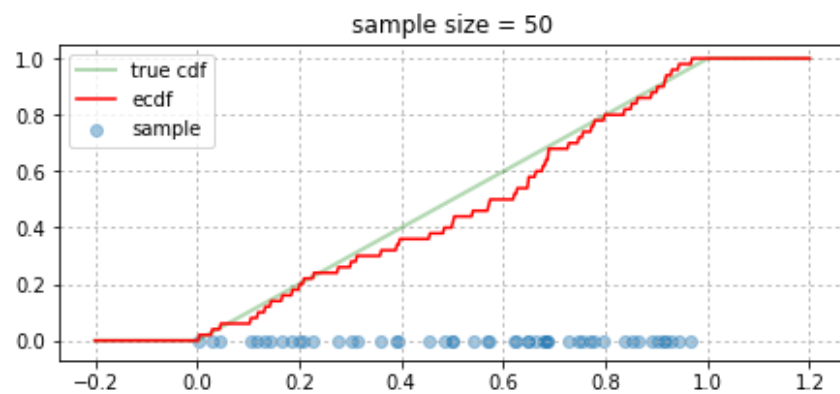
**2.** Попробуйте скачать с гитхаба <https://github.com/statsmodels/statsmodels/> (<https://github.com/statsmodels/statsmodels/>), установить руками. При этом должен быть установлен cython.

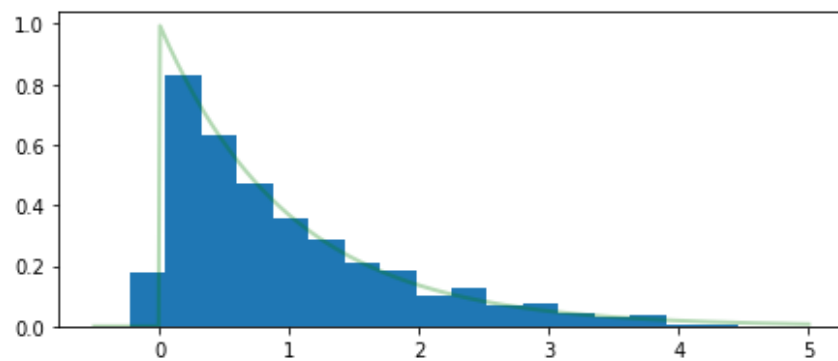
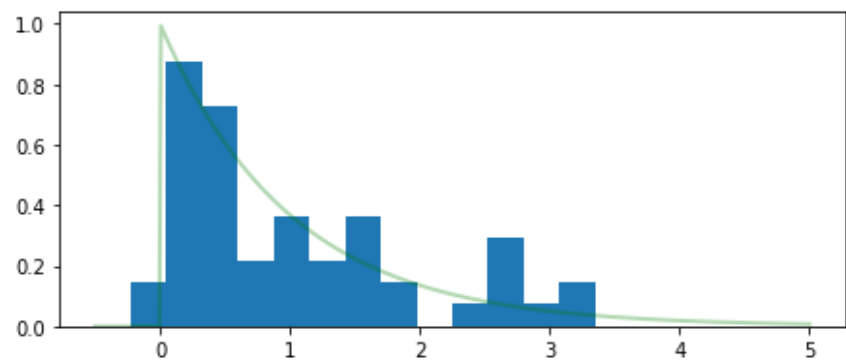
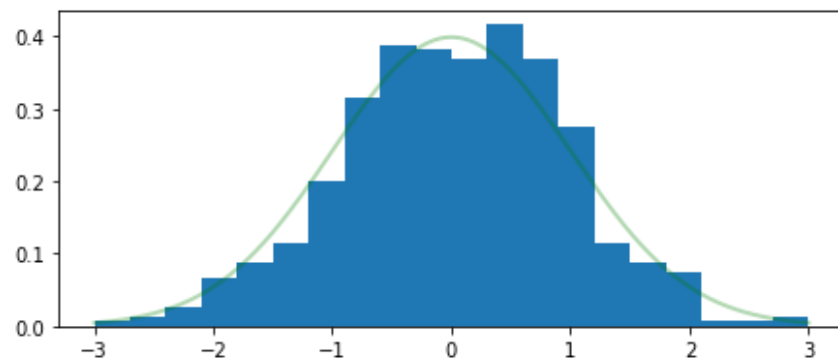
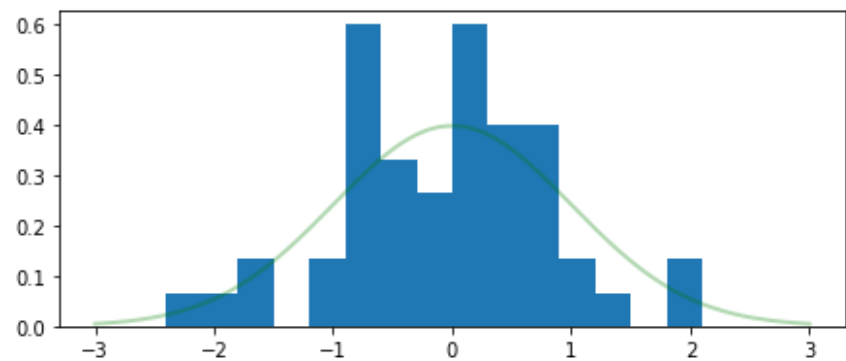
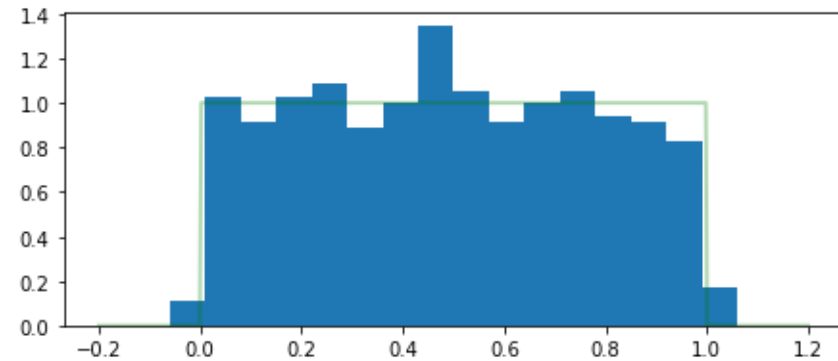
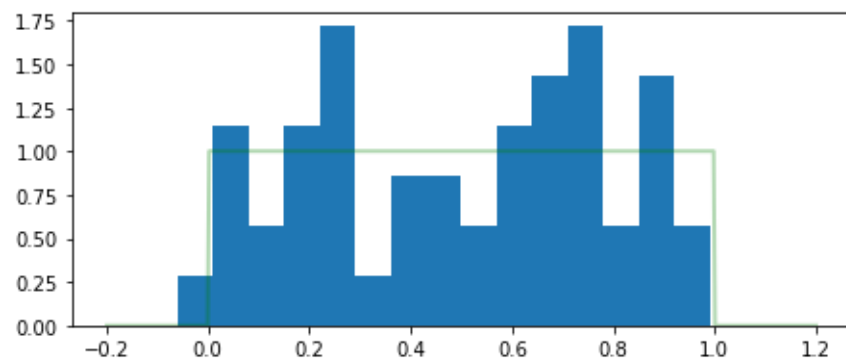
Можно также воспользоваться другой реализацией <http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KernelDensity.html#sklearn.neighbors.KernelDensity> (<http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KernelDensity.html#sklearn.neighbors.KernelDensity>)

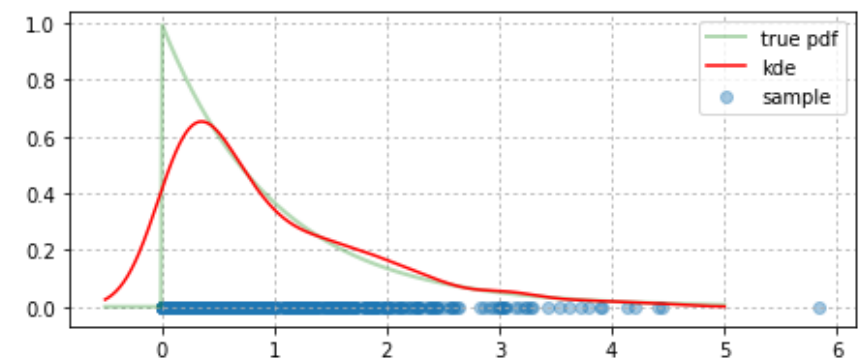
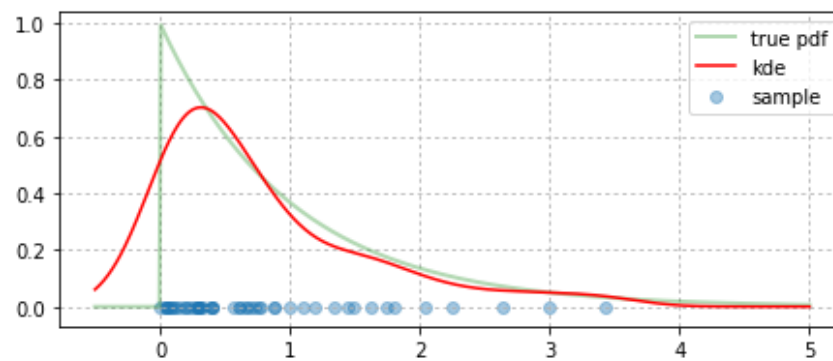
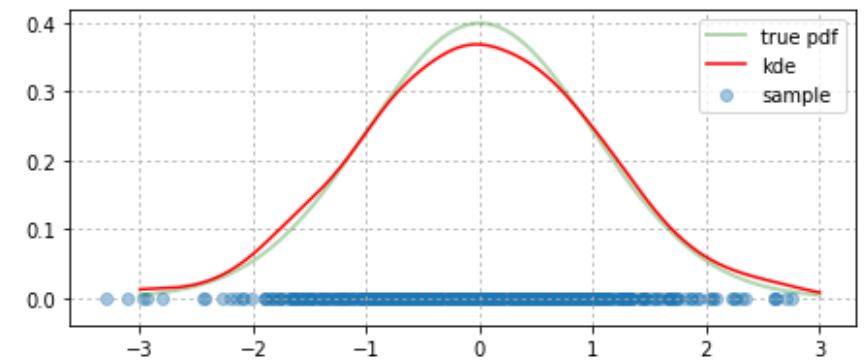
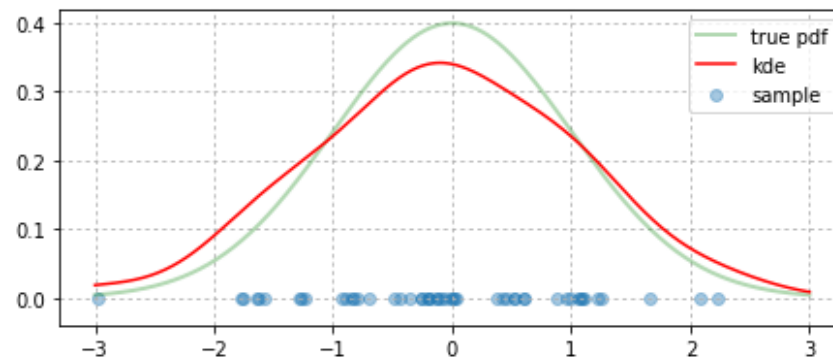
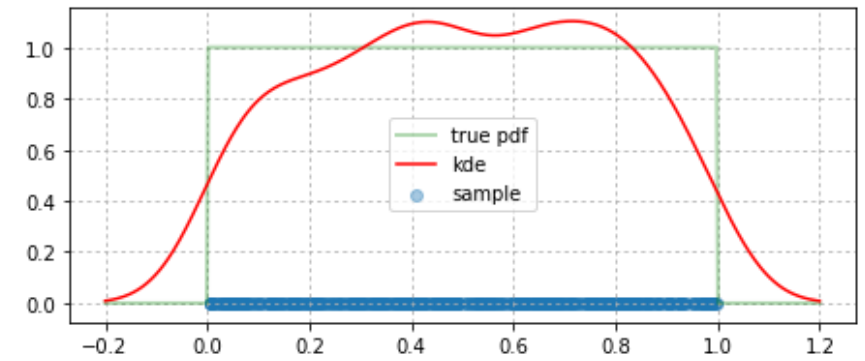
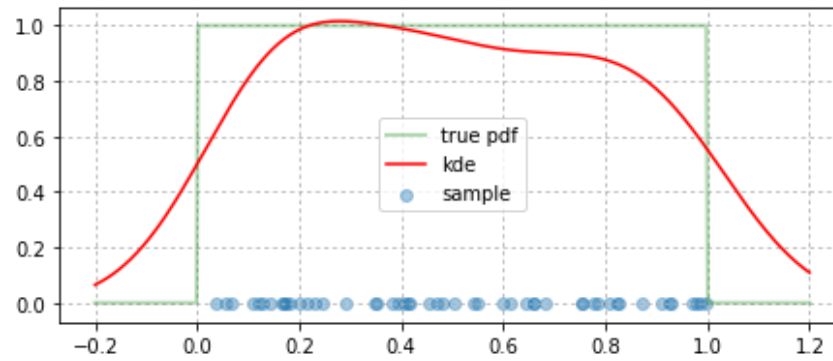
---

Теперь примените реализованные выше функции к выборкам размера 500 из распределений  $U[0, 1]$ ,  $\mathcal{N}(0, 1)$  и  $Exp(1)$ . Графики (ф.р., плотностей) стройте на интервалах  $(-0.2, 1.2)$ ,  $(-3, 3)$  и  $(-0.5, 5)$  соответственно.

```
In [20]: draw_ecdf(uniform(500, precision=100), np.linspace(-0.2, 1.2, 500), cdf=sps.uniform.cdf)
draw_ecdf(normal(500, precision=100), np.linspace(-3, 3, 500), cdf=sps.norm.cdf)
draw_ecdf(expon(500, precision=100), np.linspace(-0.5, 5, 500), cdf=sps.expon.cdf)
draw_hist(uniform(500, precision=100), np.linspace(-0.2, 1.2, 500), pdf=sps.uniform.pdf)
draw_hist(normal(500, precision=100), np.linspace(-3, 3, 500), pdf=sps.norm.pdf)
draw_hist(expon(500, precision=100), np.linspace(-0.5, 5, 500), pdf=sps.expon.pdf)
draw_pdf(uniform(500, precision=100), np.linspace(-0.2, 1.2, 500), pdf=sps.uniform.pdf)
draw_pdf(normal(500, precision=100), np.linspace(-3, 3, 500), pdf=sps.norm.pdf)
draw_pdf(expon(500, precision=100), np.linspace(-0.5, 5, 500), pdf=sps.expon.pdf)
```







**Вывод:** По выборке можно построить эмпирическую функцию распределения и ядерную оценку плотности. Чем больше выборка, тем приближение точнее.

**Задача 4.** Сгенерируйте выборку  $X_1, \dots, X_{10000}$  из стандартного нормального распределения. Для каждого  $n \leq 10000$  постройте эмпирическую функцию распределения  $F_n^*$  и посчитайте **точное** значение статистики

$$D_n = \sup_{x \in \mathbb{R}} |F_n^*(x) - F(x)|.$$

Постройте график зависимости статистики  $D_n$  от  $n$ . Верно ли, что  $D_n \rightarrow 0$  и в каком смысле? Не забудьте сделать вывод.

```
In [195]: N = 10000
sample = sps.norm.rvs(size=N)
Dn = np.zeros(N)
# У нас эмпирическая функция распределения кусочно-постоянна, а функция распределения монотонна,
# поэтому достаточно смотреть разность в точках разрыва эмпирической функции, а разрывы у неё в точках sample.
for size in range(1, N):
    discontinuities = sorted(sample[:size])
    # То есть для каждого промежутка постоянства надо посмотреть левую и правую границу
    ecdf = ECDF(discontinuities)
    values_right = ecdf(discontinuities)
    values_left = np.insert(values_right[:-1], 0, [0])

    real_cdf = sps.norm.cdf

    # Делаем это через массивы numpy, потому что иначе очень медленно
    max_right = np.max(np.abs(real(discontinuities)-values_right))
    max_left = np.max(np.abs(real(discontinuities)-values_left))

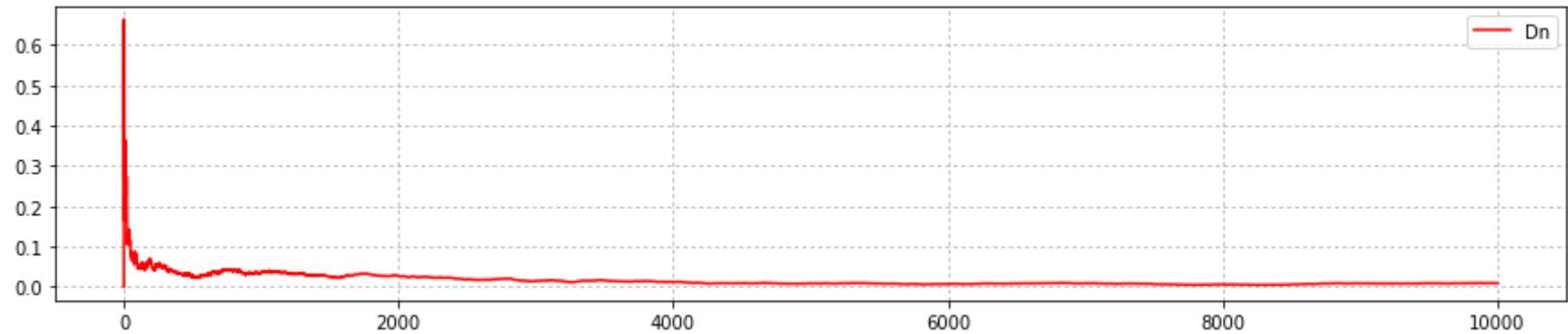
    Dn[size] = max(max_left, max_right)
    if size % 100 == 0:
        print(size//100, end=' ')

grid = np.linspace(0, N, N)

plt.figure(figsize=(15, 3))
plt.plot(grid, Dn, color='red', label='Dn')
plt.legend()
plt.grid(ls=':')
plt.show()
```



1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39  
 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75  
 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99



Сходимость происходит почти наверное по теореме Гливенко-Кантелли.

**Вывод:** По теореме сходимость почти наверное, а по графику можно однозначно сказать лишь о сходимости по вероятности.

**Задача 5.** Исследуйте вид ядерной оценки плотности в зависимости от вида ядра и его ширины.

Для этого сгенерируйте выборку  $X_1, \dots, X_{200}$  из распределения  $U[0, 1]$  и постройте серию графиков для различной ширины гауссовского ядра, а затем другую серию графиков для различных типов ядер при фиксированной ширине. На каждом графике на отрезке  $[-0.2, 1.2]$  должны быть изображены истинная плотность (полупрозрачным цветом) и ее ядерная оценка, а так же с нулевой  $y$ -координатой должны быть нанесены точки выборки. Для экономии места стройте графики в два столбца.

Не забудьте сделать вывод.

```
In [22]: # Не-гауссовские типы ядер не работают в statsmodels, поэтому будет использована sklearn.
from sklearn.neighbors.kde import KernelDensity
```

```

In [23]: N = 200
sample = sps.uniform.rvs(size=N)
pdf = sps.uniform.pdf
size = len(sample)

kernels = ['gaussian', 'tophat', 'epanechnikov', 'exponential', 'linear', 'cosine']
bandwidths = [0.01, 0.05, 0.1, 0.25, 0.5, 1.0]

plt.figure(figsize=(16, 20))

# Строим для разных типов ядер
for i, kernel in enumerate(kernels):
    plt.subplot(len(kernels), 2, 2*i+1)
    grid = np.linspace(-0.2, 1.2, size)
    kernel_density = KernelDensity(kernel=kernel, bandwidth=0.2).fit(sample[:size].reshape(-1, 1))

    plt.scatter(sample[:size], np.zeros(size), alpha=0.4, label='sample')
    plt.plot(grid, pdf(grid), color='green', alpha=0.3, lw=2, label='true pdf')
    plt.plot(grid, np.exp(kernel_density.score_samples(grid.reshape(-1, 1))),
              color='red', label='kde for %s kernel'%kernel)

    plt.legend()
    plt.grid(ls=':')

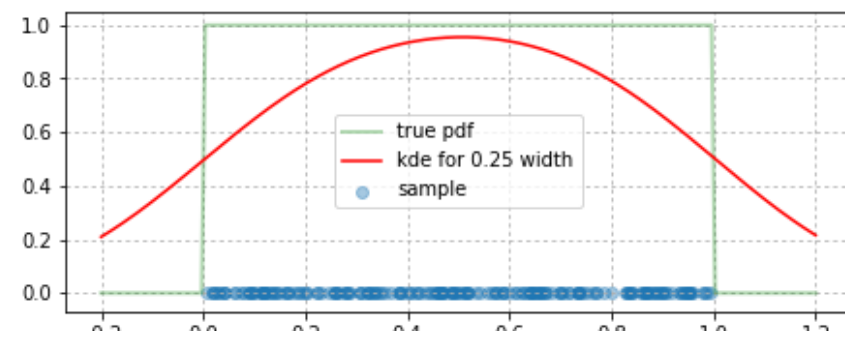
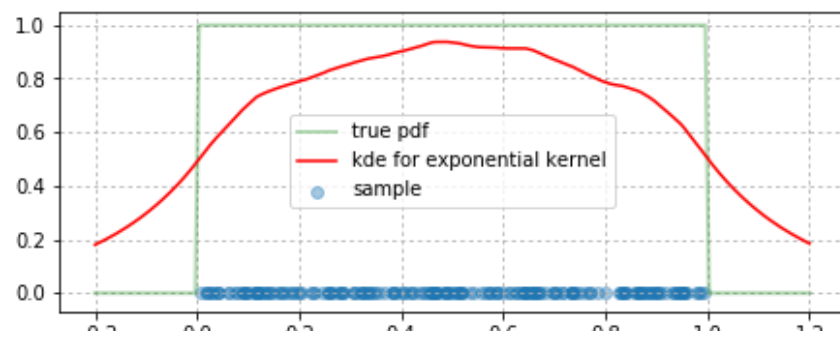
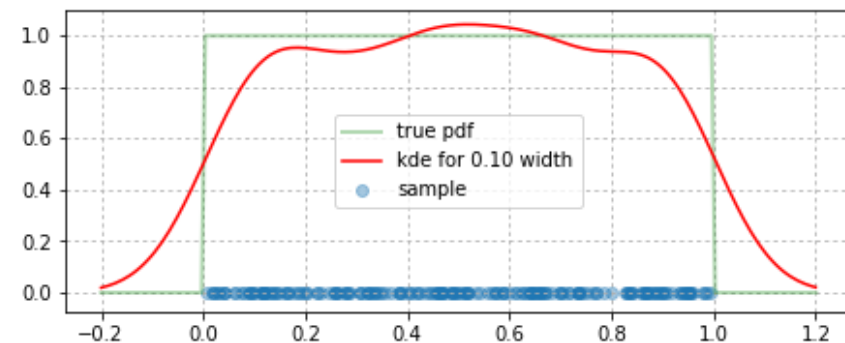
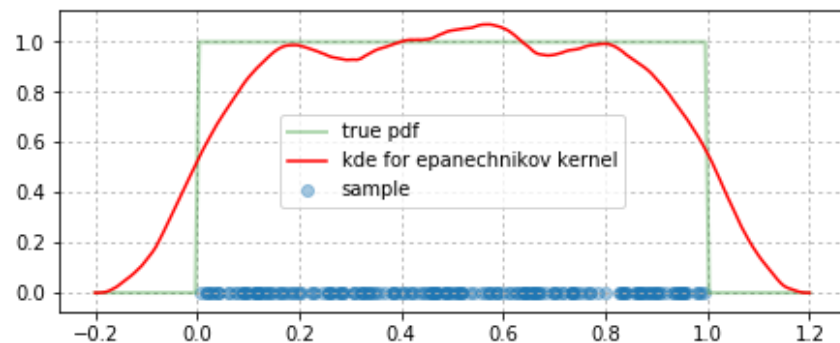
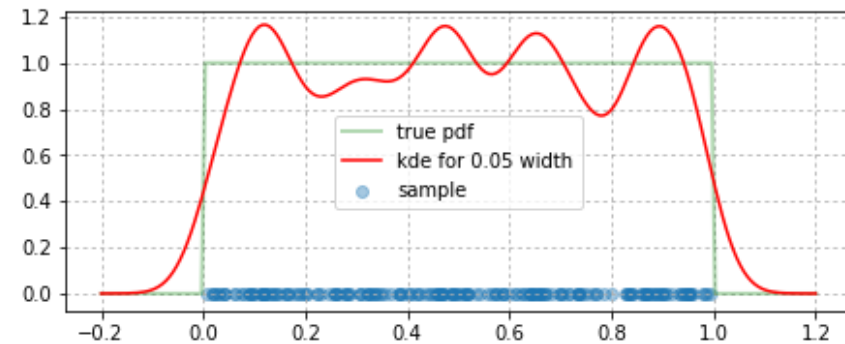
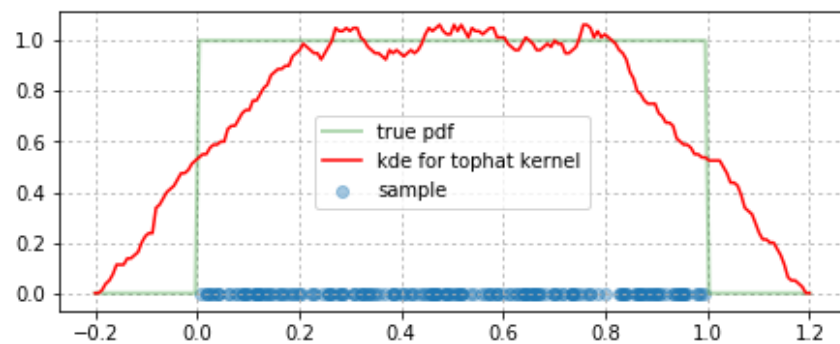
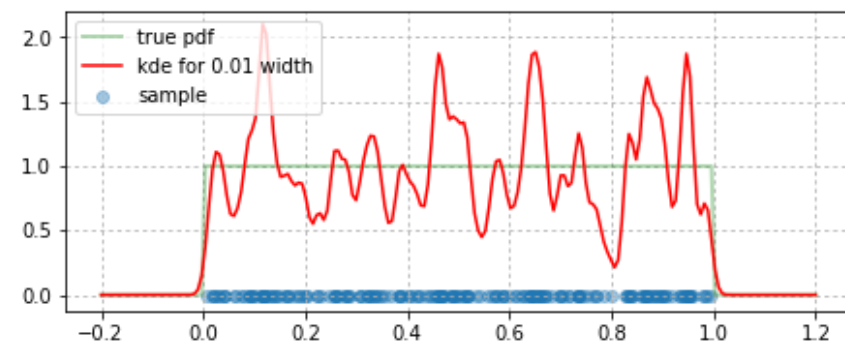
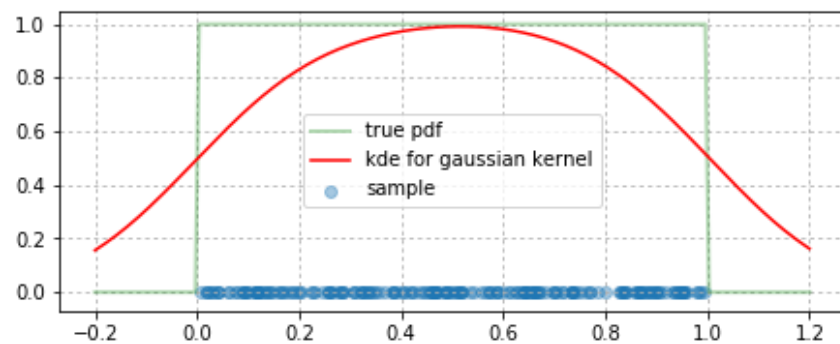
# Строим для разных ширин
for i, bandwidth in enumerate(bandwidths):
    plt.subplot(len(kernels), 2, 2*i+2)
    grid = np.linspace(-0.2, 1.2, size)
    kernel_density = KernelDensity(kernel='gaussian', bandwidth=bandwidth).fit(sample[:size].reshape(-1, 1))

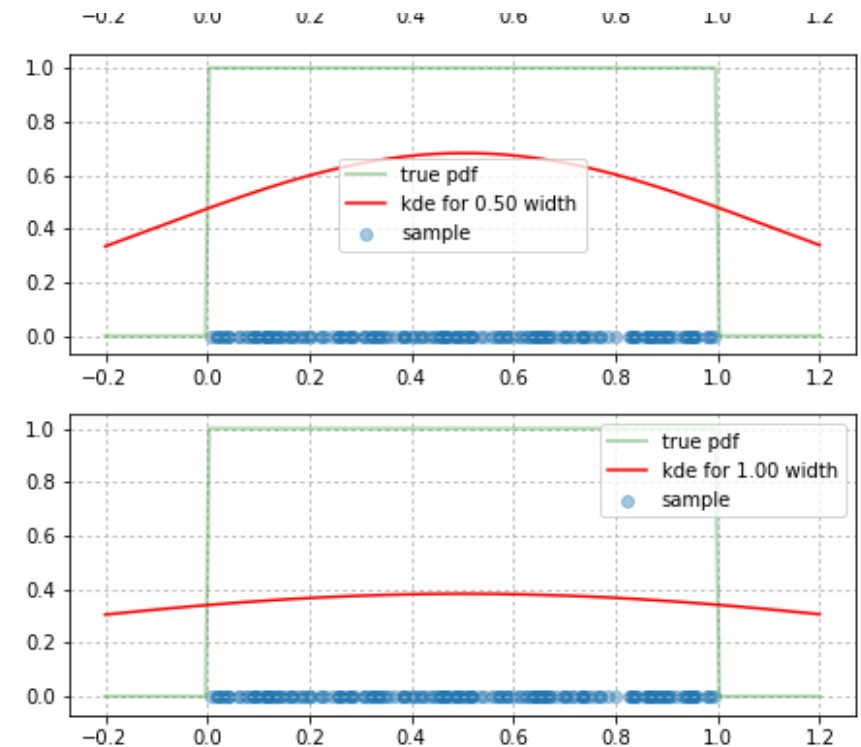
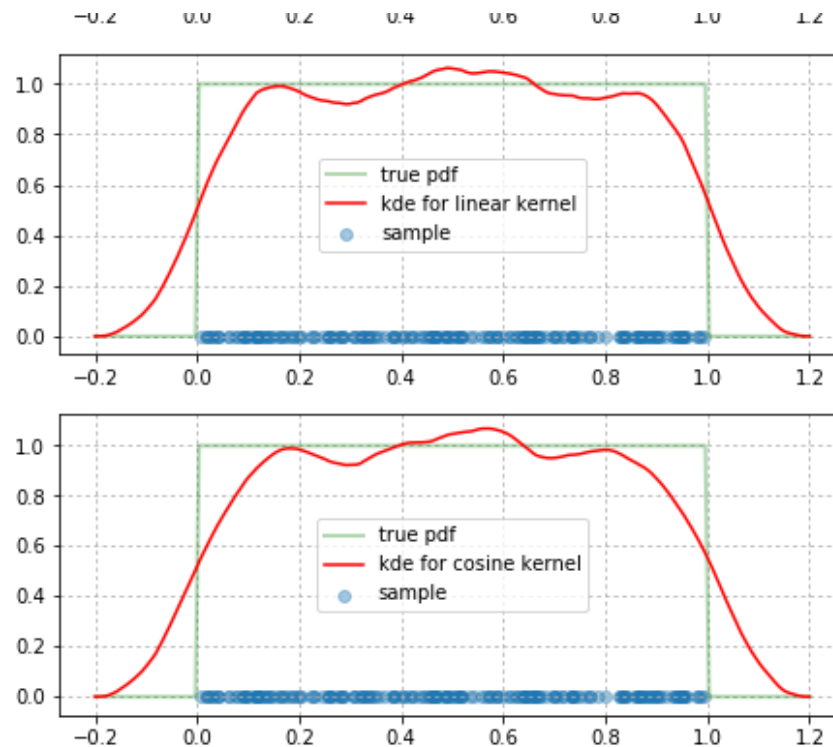
    plt.scatter(sample[:size], np.zeros(size), alpha=0.4, label='sample')
    plt.plot(grid, pdf(grid), color='green', alpha=0.3, lw=2, label='true pdf')
    plt.plot(grid, np.exp(kernel_density.score_samples(grid.reshape(-1, 1))),
              color='red', label='kde for %.2f width'%bandwidth)

    plt.legend()
    plt.grid(ls=':')

plt.show()

```





**Вывод:** Для различных типов ядер и для различных ширин ядер получаются различные ядерные оценки. Какие-то из них приближают лучше, какие-то хуже. Визуально наблюдается, что для какой-то ширины (в нашем случае — 0.10) и для каких-то типов ядер (у нас — Епанечникова, линейного и cosine ядер) график похож на требуемый больше всего.

**Задача 6.** В файле `countries.csv` дан список стран и территорий с указанием их площади. Нанести значения площади на график и постройте эмпирическую функцию распределения и ядерную оценку плотности. Поскольку некоторые страны слишком большие, ограничьте график по оси икс. Не забудьте сделать вывод.

```
In [196]: import csv
```

```

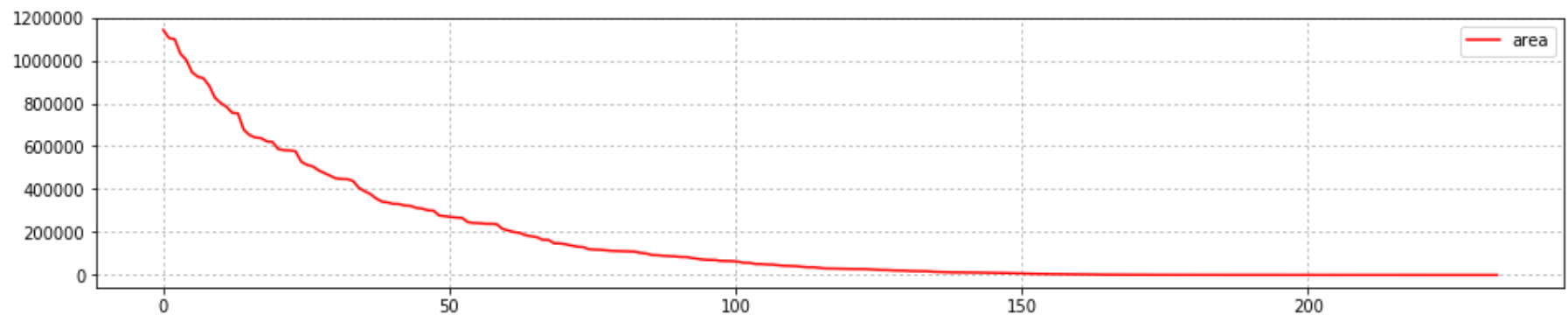
In [197]: countries_arr = []
with open('countries.csv') as csvfile:
    reader = csv.reader(csvfile, delimiter=' ')
    for row in reader:
        countries_arr.append(float(row[2]))

# Выкидываем 10% наибольших, чтобы ограничить график
graphic_bound = 0.1

countries_nparr = np.array(countries_arr)
countries_nparr_cutted = np.array(countries_arr)[int(len(countries_arr)*graphic_bound):]

# Рисуем график
grid = np.linspace(0, len(countries_nparr_cutted), len(countries_nparr_cutted))
plt.figure(figsize=(16, 3))
plt.plot(grid, countries_nparr_cutted, color='red', label='area')
plt.legend()
plt.grid(ls=':')
plt.show()

```



```
In [198]: # Строим ЭФР
plt.figure(figsize=(16, 3))
plt.scatter(countries_nparr_cutted, np.zeros(len(countries_nparr_cutted)),
            alpha=0.4, label='sample')

plt.plot(countries_nparr_cutted,
         ECDF(countries_nparr)(countries_nparr_cutted),
         color='red', label='ecdf')

plt.legend()
plt.grid(ls=':')
plt.title('sample size = {}'.format(size))
plt.show()

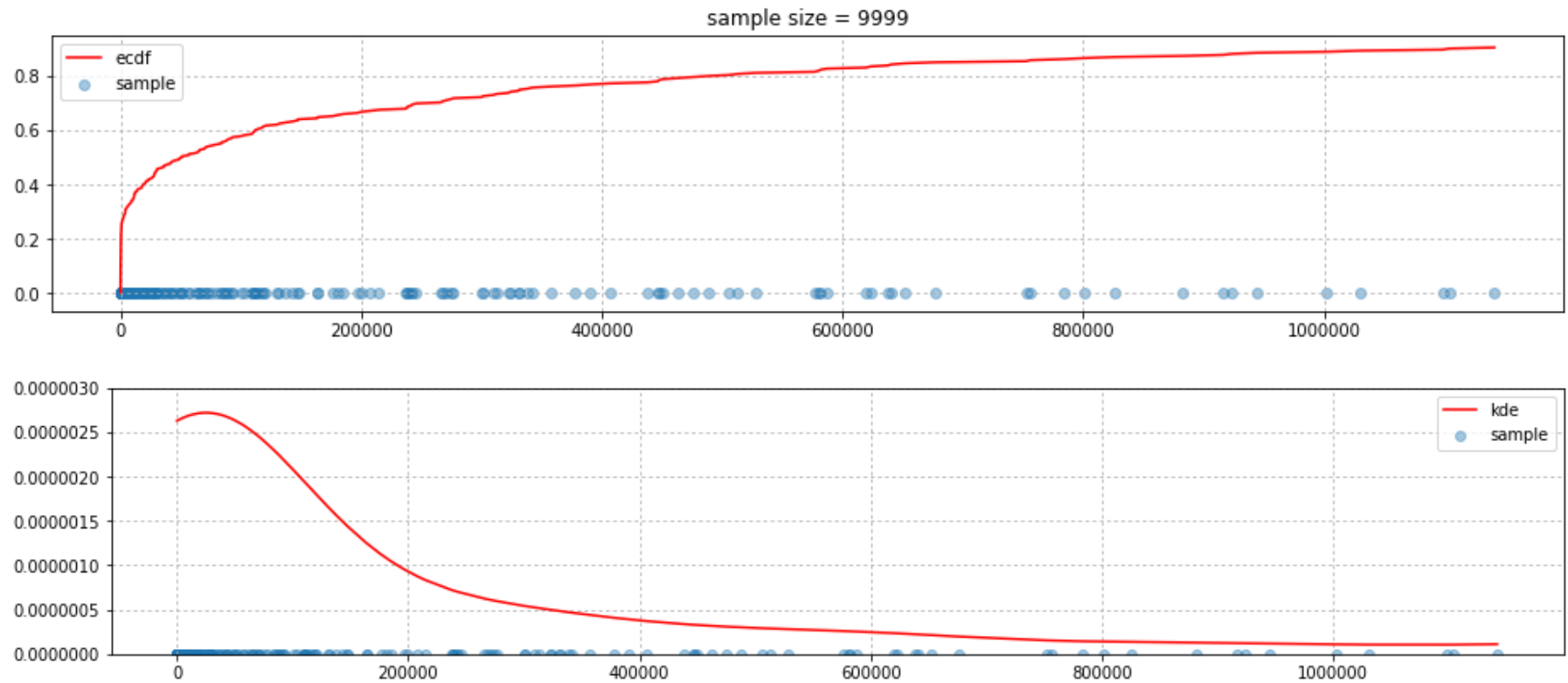
# Строим ядерную оценку плотности
plt.figure(figsize=(16, 3))
plt.ylim(0, 3e-6)

plt.scatter(countries_nparr_cutted, np.zeros(len(countries_nparr_cutted)),
            alpha=0.4, label='sample')

kernel_density = KDEUnivariate(countries_nparr)
kernel_density.fit()

plt.plot(countries_nparr_cutted, kernel_density.evaluate(countries_nparr_cutted),
         color='red', label='kde')

plt.legend()
plt.grid(ls=':')
plt.show()
```



**Вывод:** Распределение площади стран после отсечения 10% самых больших приобрело вид половинки нормального, но одновременно стало похоже на множество других распределений. Нельзя сразу сказать, какое это распределение.

**Задача 7.** Проведите небольшое исследование. Выберите случайных  $n$  человек в социальной сети. Вы можете выбирать их случайно из всех зарегистрированных в этой социальной сети, либо по какому-то *одному* критерию (укажите его). Составьте выборку  $X_1, \dots, X_n$ , где  $X_i$  --- количество друзей у  $i$ -го человека. Постройте по этой выборке эмпирическую функцию распределения. Можете ли вы сказать, какому закону подчиняется распределение количества друзей?

Выборка должна быть из не менее 30 человек, ограничений сверху нет. Вы можете также написать программу, которая будет автоматически собирать данные. Не забудьте сделать вывод.

```
In [69]: # Пользуемся VK API
import vk
session = vk.Session(access_token='secret')
api = vk.API(session)
```

```
In [70]: # Члены группы ФИБТ 2 курс 2016
response = api.groups.getMembers(group_id='diht2015')
```

```
In [71]: import time

members = response['users']
print("Have %d users"%len(members))

friends_count = []
for id in members:
    try:
        person_friends = api.friends.get(user_id=id)
        friends_count.append(len(person_friends))
        print(len(friends_count), end=', ')
    except:
        print('deleted', end=', ')
        time.sleep(0.4)

print("Done")
```

Have 251 users

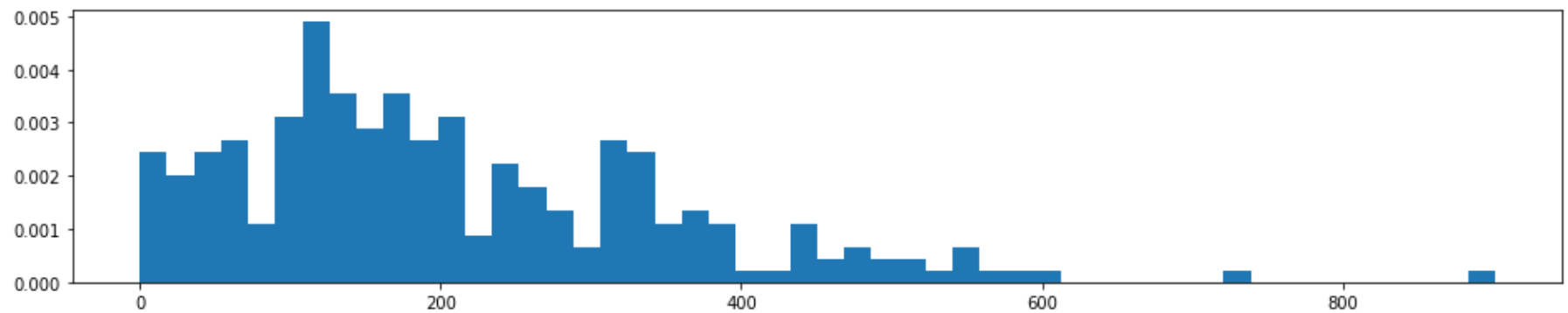
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, deleted, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, Done

```
In [203]: # Делаем выборку
sample = sorted(np.array(friends_count))
sample = np.array(sample).astype(float)
```



```
In [204]: # Строим гистограмму
plt.figure(figsize=(16, 3))
size = len(sample)
grid = np.linspace(0, np.max(sample), 1000)
plt.hist(sample[:size],
          50,
          range=(grid.min(), grid.max()),
          normed=True)

plt.show()
```



```

In [205]: # Сторим ядерную оценку плотности и пытаемся подобрать распределение
plt.figure(figsize=(16, 3))
size = len(sample)

kernel_density = KDEUnivariate(sample[:size])
kernel_density.fit()

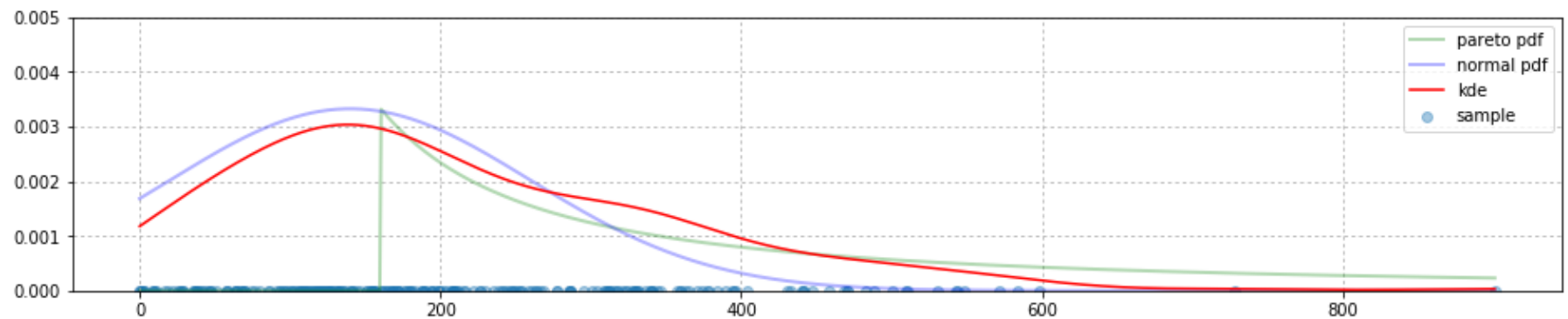
plt.scatter(sample[:size], np.zeros(size),
            alpha=0.4, label='sample')

plt.plot(grid, sps.pareto.pdf(grid, 0.5, loc=10, scale=150), color='green', alpha=0.3, lw=2, label='pareto
pdf')
plt.plot(grid, sps.norm.pdf(grid, loc=140, scale=120), color='blue', alpha=0.3, lw=2, label='normal pdf')

plt.ylim(0, 0.005)
plt.plot(grid, kernel_density.evaluate(grid), color='red', label='kde')

plt.legend()
plt.grid(ls=':')
plt.show()

```



**Вывод:** Получившееся распределение напоминает распределение Парето, потому что "хвост" идёт довольно высоко, но у Парето нет симметрии, а у эмпирической функции она в некоторой степени есть. Оно напоминает нормальное распределение, но только вблизи центра, а "хвосты" у нормального идут гораздо ниже. Поэтому, хорошо определить, какое это распределение, не получается.

In [ ]: