In [306]:
```
import itertools
from datetime import datetime

import numpy as np
import scipy as sp
import scipy.stats
import scipy.special
import scipy.integrate
import statsmodels as sm
from statsmodels.distributions.empirical_distribution import ECDF
import plotly
import plotly.graph_objs as go

from numba import njit

np.random.seed(datetime.now().microsecond)
np.set_printoptions(
    precision=5,
    linewidth=120,
)
plotly.offline.init_notebook_mode()
```

Задание №1. Вероятностные распределения

Выбор распределений

- Дискретное: *гипергеометрическое* (https://ru.wikipedia.org/wiki/Гипергеометрическое_распределение)
- Непрерывное: *нормальное* (https://ru.wikipedia.org/wiki/Нормальное_распределение)

Описание основных характеристик распределений

Гипергеометрическое распределение

Гипергеометрическое распределение - дискретное распределение, описывающее вероятность события, при котором ровно k из n случайно выбранн элементов окажутся *помеченными*, при этом выборка осуществляется из множества мощности N, в котором присутствует m помеченных элементов Считается, что каждый из элементов может быть выбран с одинаковой вероятностью $\frac{1}{N}$. Запишем это формально:

$$N \in \mathbb{N},\ m \in \overline{0, N},\ n \in \overline{0, N},$$
$$k \in \overline{\max(0, m + n - N), \min(m, n)}$$

Тогда $HG(N, m, n)$ описывает вероятность события, при котором ровно k из n элементов выборки окажутся *помеченными*:

$$\{\ \xi \sim HG(N, m, n)\ \} \iff \left\{ \mathbb{P}\,(\ \xi = k\ ) = \frac{\binom{m}{k}\binom{N-m}{n-k}}{\binom{N}{n}} \right\}$$

Математическое ожидание

По определению, математическое ожидание случайной величины – это ее $1^{й}$ начальный момент. Для начала, найдем $k^{й}$ начальный момент для ξ (это понадобится для дальнейших выводов):

$$\mathbb{E}\left[\xi^r\right] = \sum_{k=0}^{n} k^r \cdot \mathbb{P}(\xi = k) = \sum_{k=0}^{n} k^r \frac{\binom{m}{k}\binom{N-m}{n-k}}{\binom{N}{n}}$$

Можем считать, что сумма берется при $k = \overline{1, n}$, так как слагаемое при $k = 0$ будет равно 0. Заметим, что

$$\begin{aligned} k\binom{m}{k} &= k\frac{m!}{k!(m-k)!} = \\ &= k\frac{m \cdot (m-1)!}{k \cdot (k-1)! \cdot (m-k)!} = \\ &= m\frac{(m-1)!}{(k-1)! \cdot (m-1-(k-1))!} = \\ &= m\binom{m-1}{k-1} \end{aligned}$$

и, как следствие,

$$\binom{N}{n} = \frac{1}{n} \cdot n \cdot \binom{N}{n} = \frac{1}{n}N\binom{N-1}{n-1}$$

Подставим:

$$\mathbb{E}\left[\xi^r\right] = \frac{n \cdot m}{N} \sum_{k=1}^{r-1} \frac{\binom{m-1}{k-1}\binom{N-m}{n-k}}{\binom{N-1}{n-1}}$$

Положим $j := k - 1$ и изменим индекс суммирования на $j = \overline{0, n-1}$. Заметим, что $n - k = n - (j+1) = (n-1) - j$ и $N - m = (N-1) - (m-1)$:

$$\mathbb{E}\left[\xi^r\right] = \frac{n \cdot m}{N} \sum_{j=0}^{n-1} (j+1)^{r-1} \frac{\binom{m-1}{j}\binom{(N-1)-(m-1)}{(n-1)-j}}{\binom{N-1}{n-1}}$$

Заметим, что выделенная часть выражения может быть записана, как $\mathbb{E}\left[(\theta+1)^{r-1}\right]$, где $\theta \sim HG(N-1, m-1, n-1)$. Следовательно,

$$\mathbb{E}\left[\xi^r\right] = \frac{n \cdot m}{N}\mathbb{E}\left[(\theta+1)^{r-1}\right]$$

Таким образом,

$$\boxed{\mathbb{E}[\xi] = \frac{n \cdot m}{N}}$$

Дисперсия

По определению дисперсии,

$$\mathrm{Var}[\xi] = \mathbb{E}\left[(\xi - \mathbb{E}[\xi])^2\right] = \mathbb{E}\left[\xi^2\right] - (\mathbb{E}[\xi])^2$$

Выведем $2^{й}$ начальный момент:

$$\mathbb{E}\left[\xi^2\right] = \frac{n \cdot m}{N}\mathbb{E}[\theta + 1] = \frac{n \cdot m}{N}\left(\frac{(n-1)(m-1)}{N-1} + 1\right)$$

Подставим:

$$\begin{aligned} \mathrm{Var}[\xi] &= \mathbb{E}\left[\xi^2\right] - (\mathbb{E}[\xi])^2 = \\ &= \frac{n \cdot m}{N}\left(\frac{(n-1)(m-1)}{N-1} + 1\right) - \left(\frac{n \cdot m}{N}\right)^2 = \\ &= \frac{n \cdot m}{N}\left(\frac{(n-1)(m-1)}{N-1} + 1 - \frac{n \cdot m}{N}\right) \end{aligned}$$

Таким образом,

$$\boxed{\mathrm{Var}[\xi] = \frac{n \cdot m}{N}\left(\frac{(n-1)(m-1)}{N-1} + 1 - \frac{n \cdot m}{N}\right)}$$

Производящая функция

По определению, производящая функция вероятностей $G(z, \xi)$ – это математическое ожидание новой случайной величины z^{ξ}. То есть:

$$G_{\xi}(z) = \mathbb{E}\left[z^{\xi}\right]$$

Для $\xi \sim HG(N, m, n)$ производящая функция выглядит так:

$$G_{\xi}(z) = \binom{N-m}{n}\left({}_2F_1(-m, -n; N-m-n+1; z) - 1\right)$$

Здесь ${}_2F_1$ - это гипергеометрическая функция (https://en.wikipedia.org/wiki/Hypergeometric_function), определенная следующим образом:

$${}_2F_1(a, b; c; z) = \sum_{n=0}^{\infty} \frac{a^{(n)}b^{(n)}}{c^{(n)}}\frac{z^n}{n!}$$

, а $x^{(n)}$ - возрастающий факториал (https://en.wikipedia.org/wiki/Falling_and_rising_factorials), определенный как:

$$x^{(n)} = \prod_{k=0}^{n-1}(x + k)$$

Характеристическая функция

По определению, характеристическая функция случайно величины ξ задается следующим образом:

$$\varphi_\xi(t) = \mathbb{E}\left[e^{it\xi}\right]$$

Для $\xi \sim HG(N, m, n)$ характеристическая функция [выглядит (https://ru.wikipedia.org/wiki/Гипергеометрическое_распределение)] так:

$$M_\xi(t) = \frac{\binom{N-D}{n}}{\binom{N}{n}}\ {}_2F_1\left(-n, -D; N-D-n+1; e^{it}\right)$$

Здесь ${}_2F_1$ - это гипергеометрическая функция.

Гистограмма вероятностей

Гистограмма – это графическое представление функции, приближающей плотность вероятности распределения на основе выборки из него.

Чтобы построить гистограмму, сначала нужно разбить множество значений выборки на несколько отрезков. Чаще всего, берут отрезки одинаковой длины, чтобы облегчить восприятие получившегося результата, однако это необязательно. Далее подсчитывается количество вхождений элементов выборки в каждый из отрезков и рисуются прямоугольники, по площади пропорциональные количеству попавших элементов выборки в соответствующий отрезок.

Вообще говоря, гистограмму можно использовать не только для приближения плотности на основе выборки, но и для визуализации самой плотности распределения, зная его плотность.

Мы будем строить гистограмму вероятностей, писать будем на языке Python3 (https://www.python.org) и использовать следующие библиотеки:

- NumPy (https://numpy.org) для работы с массивами
- SciPy (https://www.scipy.org) для комбинаторных и статистических функций
- Plotly (https://plot.ly/python/) для визуализации

Итак, для начала, определим класс гипергеометрического распределения `HG`, который будет содержать в себе информацию о параметрах N, m и n и предоставлять метод `p(k)`, возвращающий вероятность принятия случайной величиной значения `k` при данных параметрах:

In [307]:
```
class HG(object):
    def __init__(self, N: int, m: int, n: int):
        self.N = N
        self.m = m
        self.n = n

    def p(self, k: int) -> float:
        return sp.special.comb(self.m, k) \
            * sp.special.comb(self.N-self.m, self.n-k) \
            / sp.special.comb(self.N, self.n)

    @property
    def expected(self):
        return self.n * self.m / self.N

    @property
    def variance(self):
        return self.expected * ((self.n - 1) * (self.m - 1) / (self.N - 1) + 1 - self.expected)

    @property
    def domain(self):
        return (max(0, self.m + self.n - self.N), min(self.m, self.n))

    def __str__(self) -> str:
        return f'HG({self.N}, {self.m}, {self.n})'
```

Далее создадим объект случайной величины $\xi \sim HG(30, 15, 20)$:

In [308]:
```
xi = HG(80, 15, 40)
```

Следующим шагом, определим интервал $\overline{0, n}$, на котором мы будем рисовать нашу гистограмму:

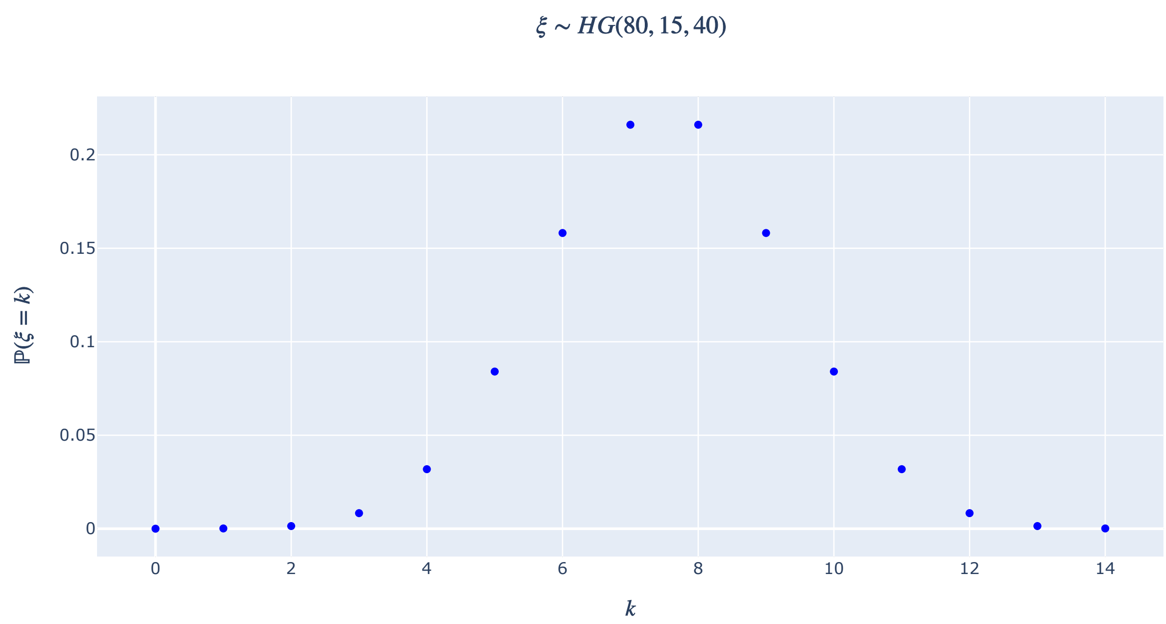
In [309]:
```
hg_data_x = np.arange(*xi.domain)
```

И, наконец, построим гистограмму и выведем ее:

In [310]:
```
theoretical_hg_PDF = np.apply_along_axis(xi.p, 0, hg_data_x,)

theoretical_hg_PDF_trace=go.Scatter(
    name='PDF',
    legendgroup='PDF',
    x=hg_data_x,
    y=theoretical_hg_PDF,
    mode='markers',
    marker_color='blue',
)

hg_hist_fig = go.Figure(
    data=(theoretical_hg_PDF_trace,),
    layout=go.Layout(
        title=go.layout.Title(
            text=r'$\xi \sim ' + str(xi) + '$',
            x=.5,
        ),
        yaxis=go.layout.YAxis(
            title=go.layout.yaxis.Title(
                text=r'$\mathbb{P}(\xi=k)$',
            ),
        ),
        xaxis=go.layout.XAxis(
            title=go.layout.xaxis.Title(
                text=r'$k$',
            ),
        ),
    ),
)
plotly.offline.iplot(hg_hist_fig)
```



Функция распределения

По определению, функция распределения $F_\xi(k) = \mathbb{P}(\xi < k)$. Для дискретной случайной величины событие $\{\xi < k\} = \bigcup\limits_{i=0}^{k-1} \{\xi = i\}$. Каждое из событ $\{\xi = i\}\ \forall i \in \overline{0, k-1}$ являются попарно несовместными. То есть $\forall i, j \in \overline{0, k-1} : i \neq j$ выполняется $\{\xi = i\} \cap \{\xi = j\} = \emptyset$. Из этого следует, что

$$\mathbb{P}(\xi < k) = \sum_{i=0}^{k-1} \mathbb{P}(\xi = i)$$

Подставим и получим:

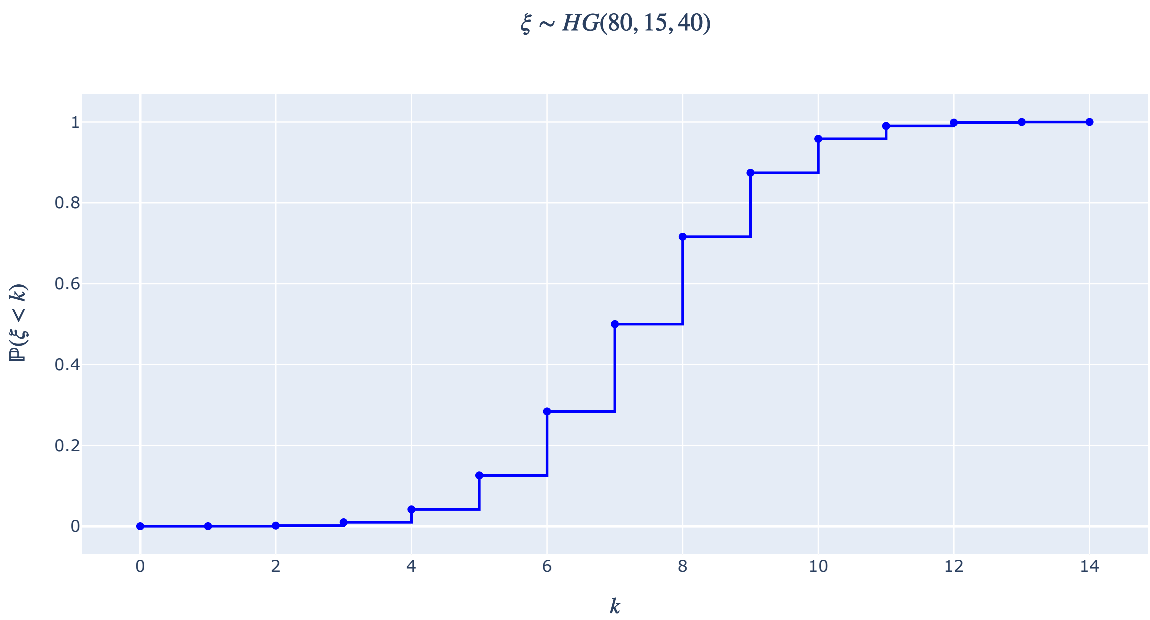
$$F_\xi(k) = \sum_{i=0}^{k-1} \mathbb{P}(\xi = i) = \sum_{i=0}^{k-1} \frac{\binom{m}{i}\binom{N-m}{n-i}}{\binom{N}{n}}$$

Построим график этой функции, учитывая, что аргументом k должно быть натуральное число, не превосходящее n:

In [311]:
```
theoretical_hg_CDF = np.cumsum(np.apply_along_axis(xi.p, 0, hg_data_x))

theoretical_hg_CDF_trace=go.Scatter(
    name='CDF',
    legendgroup='CDF',
    x=hg_data_x,
    y=theoretical_hg_CDF,
    line=go.scatter.Line(
        shape='hv',
        color='blue',
    ),
)

hg_dist_fig = go.Figure(
    data=(theoretical_hg_CDF_trace,),
    layout=go.Layout(
        title=go.layout.Title(
            text=r'$\xi \sim ' + str(xi) + '$',
            x=.5,
        ),
        yaxis=go.layout.YAxis(
            title=go.layout.yaxis.Title(
                text=r'$\mathbb{P}(\xi<k)$',
            ),
        ),
        xaxis=go.layout.XAxis(
            title=go.layout.xaxis.Title(
                text=r'$k$',
            ),
        ),
    ),
)
hg_dist_fig.show()
```



Нормальное распределение

Нормальное распределение - непрерывное распределение, описывающее поведение величины отклонения измеряемого значения x от истинного значения μ (которое является математическим ожиданием) и в рамках некоторого разброса σ (среднеквадратичного отклонения). Запишем это формально:

$$\{\eta \sim N(\mu, \sigma^2)\} \iff \left\{ \begin{array}{c} F_\eta(x) = \mathbb{P}\left(\eta < x\right) = \int_{-\infty}^{x} f_\eta(x)dx, \\ \text{где } f_\eta(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} - \text{плотность вероятности} \end{array} \right\}$$

Математическое ожидание

Найдем математическое ожидание $\eta \sim N(\mu, \sigma^2)$:

$$\begin{aligned}\mathbb{E}[\eta] &= \int_{-\infty}^{+\infty} x \cdot f_\eta(x)dx = \\ &= \int_{-\infty}^{+\infty} xe^{-\frac{(x-\mu)^2}{2\sigma^2}}dx = \\ &= \frac{1}{\sigma\sqrt{2\pi}}\int_{-\infty}^{+\infty} xe^{-\frac{(x-\mu)^2}{2\sigma^2}}dx\end{aligned}$$

Сделаем замену $t = \frac{x-\mu}{\sqrt{2}\sigma}$:

$$\begin{aligned}\mathbb{E}[\eta] &= \frac{1}{\sigma\sqrt{2\pi}}\int_{-\infty}^{+\infty}(\sigma\sqrt{2}t+\mu)e^{-t^2}d\left(\frac{x-\mu}{\sqrt{2}\sigma}\right) = \\ &= \frac{\sigma\sqrt{2}}{\sqrt{\pi}}\int_{-\infty}^{+\infty} te^{-t^2}dt + \frac{\mu}{\sqrt{\pi}}\int_{-\infty}^{+\infty} e^{-t^2}dt = \\ &= \frac{\sigma\sqrt{2}}{\sqrt{\pi}}\left(\int_{-\infty}^{0} te^{-t^2}dt - \int_{-\infty}^{0} te^{-t^2}dt\right) + \frac{\mu}{\sqrt{\pi}}\int_{-\infty}^{+\infty} e^{-t^2}dt = \\ &= \frac{\mu}{\sqrt{\pi}}\int_{-\infty}^{+\infty} e^{-t^2}dt\end{aligned}$$

Заметим, что получившееся выражение содержит интеграл, который может быть сведен к интегралу Эйлера-Пуассона (https://ru.wikipedia.org/wiki/Гауссов_интеграл):

$$\int_{-\infty}^{+\infty} e^{-t^2}dt = 2\int_{0}^{+\infty} e^{-t^2}dt = \sqrt{\pi}$$

Таким образом,

$$\boxed{\mathbb{E}[\eta] = \mu}$$

Дисперсия

$$\begin{aligned}\mathrm{Var}[\eta] &= \mathbb{E}\left[(\eta-\mu)^2\right] = \\ &= \int_{-\infty}^{+\infty}(x-\mu)^2 \cdot f_\eta(x)dx = \\ &= \int_{-\infty}^{+\infty}(x-\mu)^2\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}dx = \\ &= \frac{1}{\sigma\sqrt{2\pi}}\int_{-\infty}^{+\infty}(x-\mu)^2 e^{-\frac{(x-\mu)^2}{2\sigma^2}}dx\end{aligned}$$

Сделаем ту же замену переменной $t = \frac{x-\mu}{\sqrt{2}\sigma}$, тогда $x = t\sqrt{2}\sigma + \mu$ и:

$$\begin{aligned}\mathrm{Var}[\eta] &= \frac{1}{\sigma\sqrt{2\pi}}\int_{-\infty}^{+\infty}(\sqrt{2}\sigma)^2t^2e^{-t^2}d(t\sqrt{2}\sigma+\mu) = \\ &= \frac{2\sigma^2}{\sqrt{\pi}}\int_{-\infty}^{+\infty} t^2e^{-t^2}dt\end{aligned}$$

Проинтегрируем по частям:

$$\begin{aligned}\mathrm{Var}[\eta] &= \frac{\sigma^2}{\sqrt{\pi}}\int_{-\infty}^{+\infty} t2te^{-t^2}dt = \\ &= \frac{\sigma^2}{\sqrt{\pi}}\left(-te^{-t^2}\Big|_{-\infty}^{+\infty} + \int_{-\infty}^{+\infty} e^{-t^2}dt\right)\end{aligned}$$

Здесь снова появляется интеграл Эйлера-Пуассона (https://ru.wikipedia.org/wiki/Гауссов_интеграл) и, в итоге, получаем:

$$\boxed{\mathrm{Var}[\eta] = \sigma^2}$$

То есть, σ является среднеквадратичным отклонением.

Характеристическая функция

Характеристическая функция для $\eta \sim N(\mu, \sigma^2)$ имеет вид:

$$\varphi_\eta(t) = \exp\left(\mu it + \frac{\sigma^2t^2}{2}\right)$$

Определим класс нормального распределения `N`, который будет содержать в себе информацию о параметрах μ и σ и предоставлять следующие методы:

- `f(x)` - возвращает значение плотности в точке `x`
- `p(k)` - возвращает $\mathbb{P}(\eta < x) = \int_{-\infty}^{x} f_\eta(x)dx$

In [312]:
```
class Norm(object):
    def __init__(self, mu: float, sigma: float):
        self.mu = mu
        self.sigma = sigma

    def f(self, x: float) -> float:
        return np.exp(-((x-self.mu)**2)/(2*self.sigma**2))/(self.sigma*(2*np.pi)**.5)

    def p(self, x: float) -> float:
        return sp.integrate.quad(self.f, -np.inf, x)[0]

    @property
    def expected(self):
        return self.mu

    @property
    def variance(self):
        return self.sigma

    def __str__(self):
        return f'N({self.mu}, {self.sigma}^2)'
```

Далее создадим объект случайной величины $\xi \sim HG(30, 15, 20)$:

In [313]:
```
eta = Norm(0, 1)
```

Следующим шагом, руководствуясь правилом трех сигм (https://ru.wikipedia.org/wiki/Среднеквадратическое_отклонение#Правило_трёх_сигм), определим интервал $(-3\sigma, 3\sigma)$, в котором окажутся все значения случайной величины с вероятностью более 0.99:

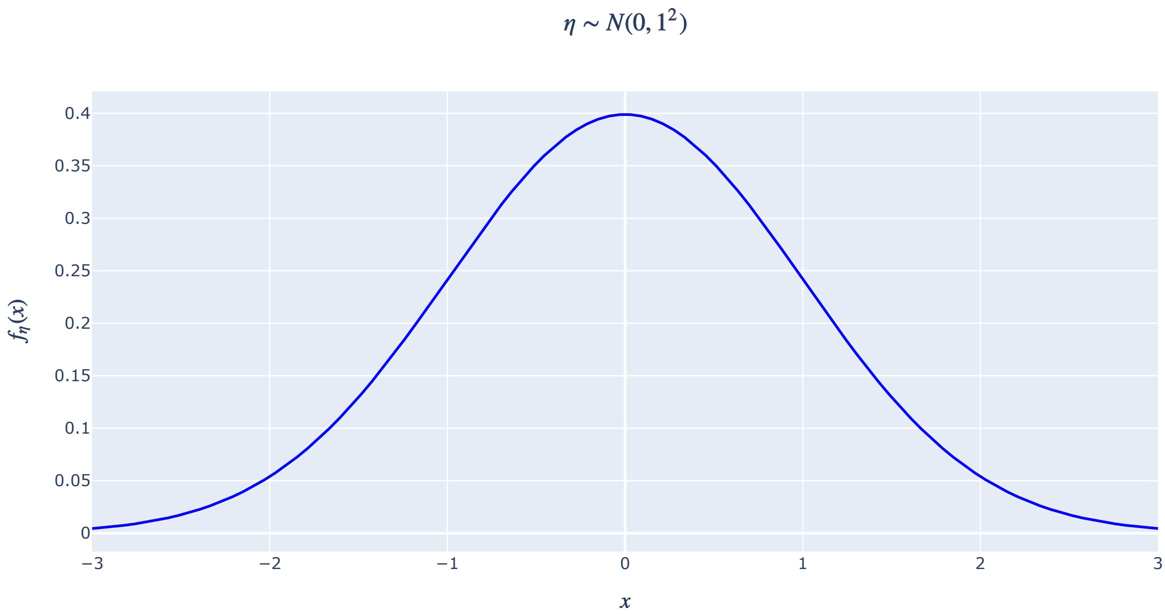
In [314]:
```
norm_data_x = np.linspace(-3*eta.sigma, 3*eta.sigma, 100)
```

И, наконец, построим плотность, используя метод `N.f` нашего класса:

In [315]:
```
theoretical_norm_PDF = np.vectorize(eta.f)(norm_data_x)

theoretical_norm_PDF_trace = go.Scatter(
    name='PDF',
    legendgroup='PDF',
    x=norm_data_x,
    y=theoretical_norm_PDF,
    line=go.scatter.Line(
        color='blue',
    ),
)

norm_dens_fig = go.Figure(
    data=(theoretical_norm_PDF_trace,),
    layout=go.Layout(
        title=go.layout.Title(
            text=r'$\eta \sim ' + str(eta) + '$',
            x=.5,
        ),
        yaxis=go.layout.YAxis(
            title=go.layout.yaxis.Title(
                text=r'$f_\eta(x)$',
            ),
        ),
        xaxis=go.layout.XAxis(
            title=go.layout.xaxis.Title(
                text=r'$x$',
            ),
        ),
    ),
)
plotly.offline.iplot(norm_dens_fig)
```



Функция распределения

По определению, функция распределения $F_\eta(x) = \mathbb{P}\left(\eta < x\right)$. Для непрерывной случайной она определяется как интеграл от функции плотности вероятности:

$$\mathbb{P}\left(\eta < x\right) = \int_{-\infty}^{x} f_\eta(x)dx$$

. Подставим и получим:

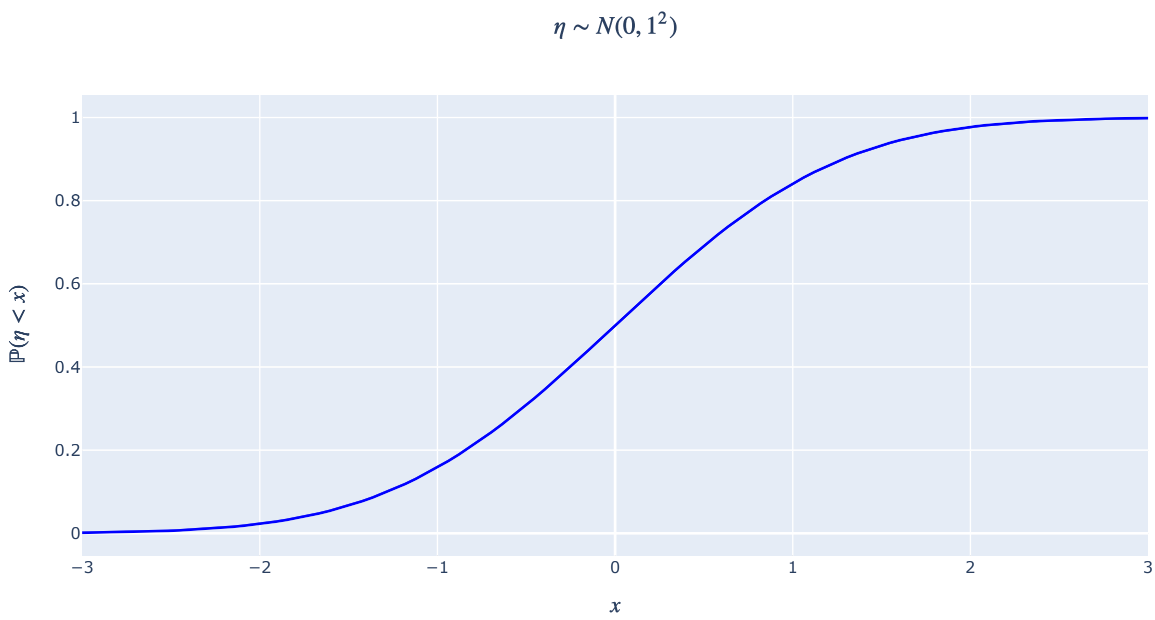
$$F_\xi(k) = \int_{-\infty}^{x} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}\,dx = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{(x-\mu)^2}{2\sigma^2}}\,dx$$

Построим график этой функции, используя метод `Norm.p` :

In [316]:
```
theoretical_norm_CDF = np.vectorize(eta.p)(norm_data_x)

theoretical_norm_CDF_trace = go.Scatter(
    name='CDF',
    legendgroup='CDF',
    x=norm_data_x,
    y=theoretical_norm_CDF,
    line=go.scatter.Line(
        color='blue',
    ),
)

norm_dist_fig = go.Figure(
    data=(theoretical_norm_CDF_trace,),
    layout=go.Layout(
        title=go.layout.Title(
            text=r'$\eta \sim ' + str(eta) + '$',
            x=.5,
        ),
        yaxis=go.layout.YAxis(
            title=go.layout.yaxis.Title(
                text=r'$\mathbb{P}(\eta<x)$',
            ),
        ),
        xaxis=go.layout.XAxis(
            title=go.layout.xaxis.Title(
                text=r'$x$',
            ),
        ),
    ),
)
plotly.offline.iplot(norm_dist_fig)
```



Примеры событий и интерпретации

Гипергеометрическое распределение

Типичная интерпретация

Типичной интерпретацией гипергеометрического распределения является выборка без возвращения из множества элементов, некоторые из которы являются помеченными. Представим, что в нашем распоряжении имеется корзина, наполненная шарами двух цветов: черные и белые. Причём всего корзине находится N шаров, m из которых – белые. Шары в корзине тщательно перемешиваются, чтобы каждый из них мог быть вытащен с одинаковой вероятностью $\frac{1}{N}$. Далее случайно вытаскиваются n шаров без возвращения. Гипергеометрическое распределение описывает вероятнос того, что среди вытащенных шаров ровно k окажутся белыми.

Действительно, всего существует $\binom{N}{n}$ выборок размера n, $\binom{m}{k}$ способов выбрать k помеченных объектов (белых шаров), и $\binom{N-m}{n-k}$ способов заполни оставшиеся $n-k$ *слотов* непомеченными объектами (черными шарами). Таким образом, вероятность того, что среди n вытащенных объектов окаже ровно k помеченных, будет равна $\frac{\binom{m}{k}\binom{N-m}{n-k}}{\binom{N}{n}}$.

Известные соотношения между распределениями

- Случайная величина, имеющая гипергеометрическое распределение с параметром $n = 1$ будет иметь распределение Бернулли с параметром m
$$\{\xi \sim HG(N, m, 1)\} \implies \left\{\xi \sim B\left(\frac{m}{N}\right)\right\}$$
Действительно, при размере выборки равным единице, случайная величина, имеющая гипергеометрическое распределение, может принимать только два значения $k \in \{0, 1\}$. То есть, нам либо попадется помеченный элемент, либо нет. Тогда эти вероятности описывает распределение Бернулли с вероятностью успеха, равной отношению общего количества элементов к количеству помеченных.
- Если зафиксировать размер выборки и количество помеченных элементов, а мощность множества, из которого ведется выборка, устремить к бесконечности, то гипергеометрическое распределение будет сходиться к биномиальному:
$$HG(N, m, n) \underset{N\to\infty}{\longrightarrow} Bi\left(n, \frac{m}{N}\right)$$

Нормальное распределение

Типичная интерпретация

Нормальное распределение описывает нормированную случайную величину, которая является суммой многих случайных слабо взаимосвязанных величин, каждая из которых вносит малый вклад относительно общей суммы. Это вытекает из центральной предельной теоремы (https://ru.wikipedia.org/wiki/Центральная_предельная_теорема).

Известные соотношения между распределениями

- Сумма двух независимых случайных величин, имеющих нормальное распределение, имеет распределение Коши (https://ru.wikipedia.org/wiki/Распределение_Коши):
$$\begin{aligned}\xi &\sim N(\mu_1, {\sigma_1}^2)\\ \eta &\sim N(\mu_2, {\sigma_2}^2)\\ \xi + \eta &\sim C(\mu_1 + \mu_2, \sqrt{{\sigma_1}^2 + {\sigma_2}^2})\end{aligned}$$
- Сумма квадратов k независимых стандартных нормальных случайных величин имеет распределение χ^2 (https://ru.wikipedia.org/wiki/Распределение_хи-квадрат) с k степенями свободы:
$$\forall i \in \overline{1, k} \quad \xi_i \sim N(0, 1)$$
$$\sum_{i=1}^{k} \xi_i \sim \chi^2(k)$$
- Натуральный логарифм логнормального распределения (https://ru.wikipedia.org/wiki/Логнормальное_распределение) имеет нормальное распределение:
$$\xi \sim LogN(\mu, \sigma^2)$$
$$\ln \xi \sim N(\mu, \sigma^2)$$

Задание №2. Моделирование случайных величин

> Основные алгоритмы моделирования брались из книги Р. Н. Вадзинского "Справочник по вероятностным распределениям" (http://zyurvas.narod.ru/knigi/Vadzinski_Ver_raspr.pdf)

Объявим константы:

In [317]:
```
QS = (5, 10, 10**2, 10**3, 10**5)
N = 5
```

Гипергеометрическое распределение

Добавим к нашему классу `HG` метод `gen`, который будет возвращать реализацию из $HG(N, m, n)$:

In [318]:
```
@njit
def gen_hg(N: int, m: int, n: int) -> int:
    x = 0
    for _ in range(n):
        if np.random.rand() < m/N:
            x += 1
            m -= 1
        N -= 1
    return x

HG.gen = lambda self: gen_hg(self.N, self.m, self.n)
```

Поясню происходящее: `x` – это количество помеченных элементов из тех, что мы выбрали.

Изначально `x = 0`, потому что мы еще не выбрали ни одного элемента.

Далее начинаем выбирать `n` элементов. Каждый раз, вероятность того, что мы выберем помеченный элемент равна `m/N`. Мы используем функцию [`numpy.random.rand`](https://docs.scipy.org/doc/numpy/reference/random/index.html) (https://docs.scipy.org/doc/numpy/reference/random/index.html), которая возвращает действительное число в интервале $[0, 1)$, и проверяем, меньше ли значение, чем `m/N`. Если это значение меньше, это означает, что мы выбрали один из помеченных элементов. В этом случае, увеличиваем `x` на единицу и уменьшаем количество оставшихся помеченных элементов на `1`, потому что выборка ведется без возвращения.

Также, вне зависимости от того, был ли выбранный объект помеченным, мы уменьшаем количество объектов, из которых ведется выборка, на едини

Этот процесс повторяется ровно `n` раз, так как всего должно быть выбрано `n` элементов.

Для ускорения работы функции (она будет часто вызываться при построении выборок), мы используем декоратор из прекрасной библиотеки [`numba`](https://numba.pydata.org) (https://numba.pydata.org), который при первом вызове функции скомпилирует нашу функцию в машинный код, и, при последующих вызовах, будет вызываться уже скомпилированная функция, которая выполняется во много раз быстрее (в нашем случае прирост в скорости будет около 2000%).

Генерация выборок

> Объем выборки будем обозначать q, чтобы не путать с параметром распреления n.

Напишем новый метод `HG.sample_hg(q)`, который будет возвращать выборку объема q из $HG(N, m, n)$:

In [319]:
```
def sample_hg(N: int, m: int, n: int, q: int) -> np.ndarray:
    return np.fromiter((gen_hg(N, m, n) for _ in range(q)), np.int)

HG.sample = lambda self, q: sample_hg(self.N, self.m, self.n, q)
```

Для каждого $q \in \{5, 10, 100, 1000, 10^5\}$ сгенерируем по 5 выборок X_q^i объема q, где $i \in \overline{1, 5}$

In [320]:
```
samples_hg = {}

for q in QS:
    samples_hg[q] = np.ndarray(shape=(N, q), dtype=np.int)
    for i in range(N):
        samples_hg[q][i] = xi.sample(q)
```

Теперь `samples_norm` – словарь, где ключом является объемом выборки, а значением по соответствующему ключу – массив из `N = 5` выборок соответствующего объема из распределения $HG(80, 15, 20)$.
Выведем выборки для `q = 5` и `q = 10`:

In [321]:
```
for q in (5, 10):
    print(f'====== q = {q} ======:')
    for i, s in enumerate(samples_hg[q]):
        print(f'#{i+1}: {s}')
```

```
====== q = 5 ======:
#1: [9 6 6 8 7]
#2: [7 6 7 9 8]
#3: [13  7  8  7  7]
#4: [ 7 10  5  6  5]
#5: [8 6 7 6 5]
====== q = 10 ======:
#1: [ 7 10  3  5 12  5  6  6  8 12]
#2: [7 8 8 7 6 5 6 7 6 6]
#3: [ 4 10  8  6  7  7  8  6  6  8]
#4: [ 7  7  8  8 10  9  7  7  6 10]
#5: [ 7  7  4  7  8  6 10  4  9 10]
```

Эмпирическая функция распределения

Для каждой выбоки вычислим эмпирическую функцию распределения (*ECDF*) с помощью функции `statsmodels.distributions.empirical_distribution.ECDF` (http://www.statsmodels.org/stable/generated/statsmodels.distributions.empirical_distribution.ECDF.html):

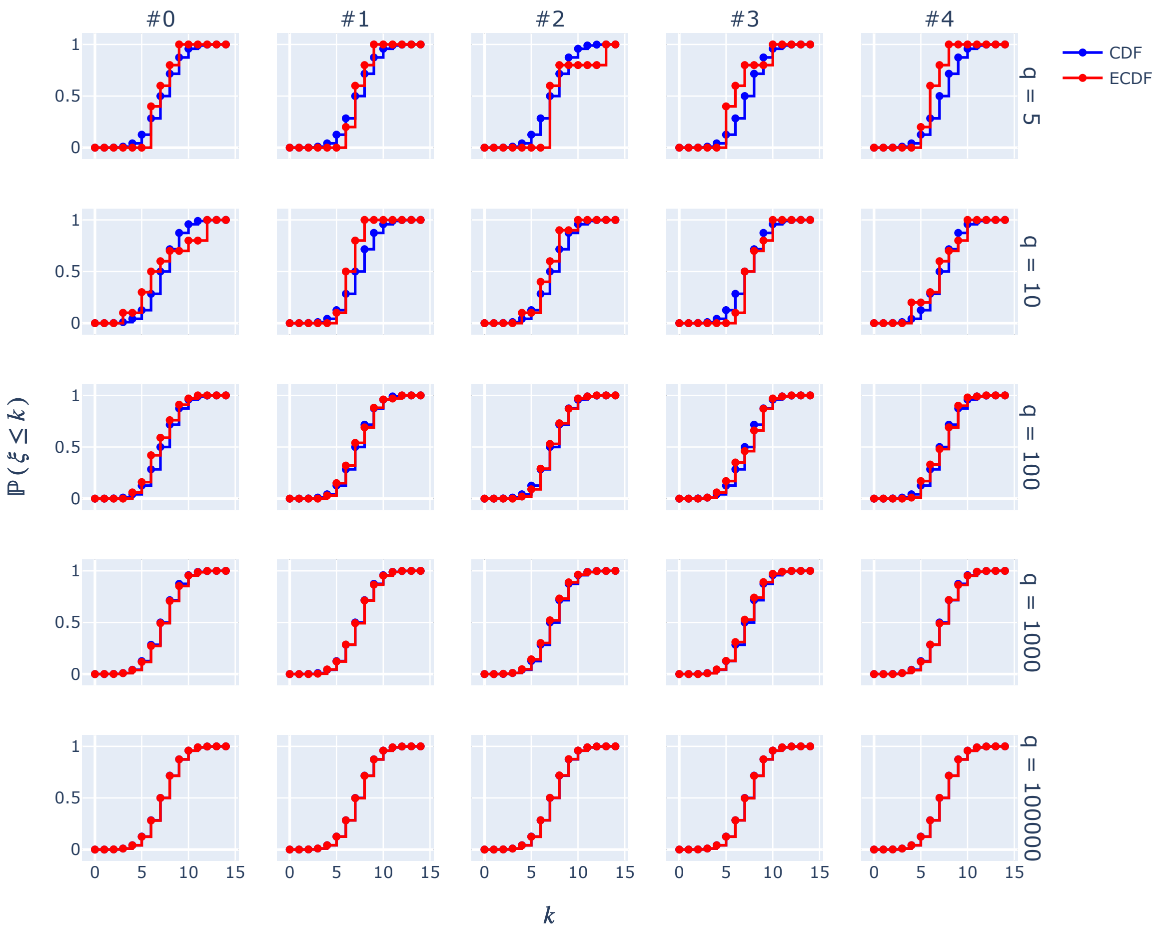
In [322]:
```
HG_ECDFs = {}
for q in samples_hg:
    HG_ECDFs[q] = np.ndarray(shape=(samples_hg[q].shape[0], hg_data_x.shape[0]), dtype=np.float)
    for i, sample in enumerate(samples_hg[q]):
        HG_ECDFs[q][i] = ECDF(sample)(hg_data_x)
```

Теперь для каждой выборки построим эмпирическую функцию распределения, а также сравним получившуюся функцию с теоретической функцией распределения:

```
In [323]: HG_ECDFs_fig = plotly.subplots.make_subplots(rows=len(QS),
                                                      cols=N,
                                                      shared_xaxes='all',
                                                      shared_yaxes='all',
                                                      row_titles=[f'q = {q}' for q in QS],
                                                      column_titles=[f'#{i}' for i in range(N)],
                                                      x_title=r'$k$',
                                                      y_title=r'$\mathbb{P}\left(\,\xi\leq k\,\right)$')

          for i, q in enumerate(HG_ECDFs):
              for j, sample in enumerate(HG_ECDFs[q]):
                  theoretical_hg_CDF_trace.showlegend = (i + j == 0)
                  HG_ECDFs_fig.add_trace(theoretical_hg_CDF_trace, row=i+1, col=j+1)
                  HG_ECDFs_fig.add_trace(
                      go.Scatter(
                          name='ECDF',
                          x=hg_data_x,
                          y=HG_ECDFs[q][j],
                          line=go.scatter.Line(
                              shape='hv',
                              color='red',
                          ),
                          legendgroup='CDF',
                          showlegend=(i + j == 0),
                      ),
                      row=i+1,
                      col=j+1)

          HG_ECDFs_fig.update_layout(height=800, width=900)
          HG_ECDFs_fig.show()
```



Все в порядке, ECDF стремится к CDF при увеличении объема выборки.

Для каждого $q \in \{5, 10, 100, 1000, 10^5\}$ найдем верхнюю границу разности каждой пары эмпирических функций распределения $\Delta_q^{i,j}$:

$$\Delta_q^{i,j} = \sup_{y \in \operatorname{supp} \xi} \left(\hat{F}\left(y, \vec{x_i}\right) - \hat{F}\left(y, \vec{x_j}\right) \right), \quad i,j \in \overline{0,5}$$

In [324]:
```
for q in HG_ECDFs:
    print(f'====== q = {q} ======:')
    for i, ECDF_i in enumerate(HG_ECDFs[q]):
        for j, ECDF_j in enumerate(HG_ECDFs[q][i+1:], start=i+1):
            print(f'i = {i}; j = {j}; max delta = {np.absolute(ECDF_i - ECDF_j).max():.4f}')
```

```
====== q = 5 ======:
i = 0; j = 1; max delta = 0.2000
i = 0; j = 2; max delta = 0.4000
i = 0; j = 3; max delta = 0.4000
i = 0; j = 4; max delta = 0.2000
i = 1; j = 2; max delta = 0.2000
i = 1; j = 3; max delta = 0.4000
i = 1; j = 4; max delta = 0.4000
i = 2; j = 3; max delta = 0.6000
i = 2; j = 4; max delta = 0.6000
i = 3; j = 4; max delta = 0.2000
====== q = 10 ======:
i = 0; j = 1; max delta = 0.3000
i = 0; j = 2; max delta = 0.2000
i = 0; j = 3; max delta = 0.4000
i = 0; j = 4; max delta = 0.2000
i = 1; j = 2; max delta = 0.2000
i = 1; j = 3; max delta = 0.4000
i = 1; j = 4; max delta = 0.3000
i = 2; j = 3; max delta = 0.3000
i = 2; j = 4; max delta = 0.2000
i = 3; j = 4; max delta = 0.2000
====== q = 100 ======:
i = 0; j = 1; max delta = 0.1000
i = 0; j = 2; max delta = 0.1300
i = 0; j = 3; max delta = 0.1300
i = 0; j = 4; max delta = 0.1100
i = 1; j = 2; max delta = 0.0600
i = 1; j = 3; max delta = 0.0800
i = 1; j = 4; max delta = 0.0600
i = 2; j = 3; max delta = 0.0800
i = 2; j = 4; max delta = 0.0800
i = 3; j = 4; max delta = 0.0500
====== q = 1000 ======:
i = 0; j = 1; max delta = 0.0140
i = 0; j = 2; max delta = 0.0360
i = 0; j = 3; max delta = 0.0380
i = 0; j = 4; max delta = 0.0140
i = 1; j = 2; max delta = 0.0280
i = 1; j = 3; max delta = 0.0340
i = 1; j = 4; max delta = 0.0070
i = 2; j = 3; max delta = 0.0150
i = 2; j = 4; max delta = 0.0300
i = 3; j = 4; max delta = 0.0360
====== q = 100000 ======:
i = 0; j = 1; max delta = 0.0019
i = 0; j = 2; max delta = 0.0038
i = 0; j = 3; max delta = 0.0025
i = 0; j = 4; max delta = 0.0018
i = 1; j = 2; max delta = 0.0038
i = 1; j = 3; max delta = 0.0009
i = 1; j = 4; max delta = 0.0027
i = 2; j = 3; max delta = 0.0043
i = 2; j = 4; max delta = 0.0030
i = 3; j = 4; max delta = 0.0033
```

Построение вариационного ряда выборки

Для каждого $q \in \{5, 10, 100, 1000, 10^5\}$ построим вариационный ряд выборки:

In [325]:
```
HG_VAR_ROWS = {}
for q in samples_hg:
    HG_VAR_ROWS[q] = np.ndarray(shape=samples_hg[q].shape, dtype=np.int)
    for i, sample in enumerate(samples_hg[q]):
        HG_VAR_ROWS[q][i] = np.sort(sample)
```

Выведем получившиеся упорядоченные выборки для `q = 5` и `q = 10` :

In [326]:
```
for q in (5, 10):
    print(f'====== q = {q} ======:')
    for i, s in enumerate(HG_VAR_ROWS[q]):
        print(f'#{i+1}: {s}')
```

```
====== q = 5 ======:
#1: [6 6 7 8 9]
#2: [6 7 7 8 9]
#3: [ 7  7  7  8 13]
#4: [ 5  5  6  7 10]
#5: [5 6 6 7 8]
====== q = 10 ======:
#1: [ 3  5  5  6  6  7  8 10 12 12]
#2: [5 6 6 6 6 7 7 7 8 8]
#3: [ 4  6  6  6  7  7  8  8  8 10]
#4: [ 6  7  7  7  7  8  8  9 10 10]
#5: [ 4  4  6  7  7  7  8  9 10 10]
```

Квантили

Квантилью уровня $\alpha \in (0, 1)$ случайной величины ξ называется такое число $x_\alpha \in \mathbb{R}$, что

$$\mathbb{P}(\xi \leq x_\alpha) \geq \alpha$$
$$\mathbb{P}(\xi \geq x_\alpha) \geq 1 - \alpha$$

Найдем квантили уровней $\alpha \in \{0.1, 0.5, 0.7\}$ для гипергеометрического распределения $HG(80, 15, 40)$:

In [327]:
```
hg_quantiles = {}
for a in (.1, .5, .7):
    for i, y in enumerate(theoretical_hg_CDF):
        if y >= a:
            hg_quantiles[a] = hg_data_x[i]
            print(f'a = {a}, quantile = {hg_quantiles[a]}')
            break
```

```
a = 0.1, quantile = 5
a = 0.5, quantile = 7
a = 0.7, quantile = 8
```

Выборочная квантиль уровня $\alpha \in (0, 1)$ выборки $\overrightarrow{X}$ - элемент вариационного ряда этой выборки, стоящий на позиции $\left[\alpha\left|\overrightarrow{X}\right| + 1\right]$.

Для каждой выборки найдем квантили уровней $\alpha \in \{0.1, 0.5, 0.7\}$ и сравним их с соответствующими квантилями данного распределения:

In [328]:
```
for q in QS:
    print(f'======= q = {q} =======:')
    for a in (.1, .5, .7):
        print(f'------ a = {a} ------')
        for i, sample in enumerate(HG_VAR_ROWS[q]):
            quantile = sample[int(a*sample.shape[0])]
            print(f'#{i}; quantile = {quantile:2}, real delta = {quantile - hg_quantiles[a]:+}')
```

```
======= q = 5 =======:
------ a = 0.1 ------
#0; quantile =  6, real delta = +1
#1; quantile =  6, real delta = +1
#2; quantile =  7, real delta = +2
#3; quantile =  5, real delta = +0
#4; quantile =  5, real delta = +0
------ a = 0.5 ------
#0; quantile =  7, real delta = +0
#1; quantile =  7, real delta = +0
#2; quantile =  7, real delta = +0
#3; quantile =  6, real delta = -1
#4; quantile =  6, real delta = -1
------ a = 0.7 ------
#0; quantile =  8, real delta = +0
#1; quantile =  8, real delta = +0
#2; quantile =  8, real delta = +0
#3; quantile =  7, real delta = -1
#4; quantile =  7, real delta = -1
======= q = 10 =======:
------ a = 0.1 ------
#0; quantile =  5, real delta = +0
#1; quantile =  6, real delta = +1
#2; quantile =  6, real delta = +1
#3; quantile =  7, real delta = +2
#4; quantile =  4, real delta = -1
------ a = 0.5 ------
#0; quantile =  7, real delta = +0
#1; quantile =  7, real delta = +0
#2; quantile =  7, real delta = +0
#3; quantile =  8, real delta = +1
#4; quantile =  7, real delta = +0
------ a = 0.7 ------
#0; quantile = 10, real delta = +2
#1; quantile =  7, real delta = -1
#2; quantile =  8, real delta = +0
```

```
#3; quantile =  9, real delta = +1
#4; quantile =  9, real delta = +1
======= q = 100 =======:
------ a = 0.1 ------
#0; quantile =  5, real delta = +0
#1; quantile =  5, real delta = +0
#2; quantile =  6, real delta = +1
#3; quantile =  5, real delta = +0
#4; quantile =  5, real delta = +0
------ a = 0.5 ------
#0; quantile =  7, real delta = +0
#1; quantile =  7, real delta = +0
#2; quantile =  7, real delta = +0
#3; quantile =  8, real delta = +1
#4; quantile =  8, real delta = +1
------ a = 0.7 ------
#0; quantile =  8, real delta = +0
#1; quantile =  9, real delta = +1
#2; quantile =  8, real delta = +0
#3; quantile =  9, real delta = +1
#4; quantile =  9, real delta = +1
======= q = 1000 =======:
------ a = 0.1 ------
#0; quantile =  5, real delta = +0
#1; quantile =  5, real delta = +0
#2; quantile =  5, real delta = +0
#3; quantile =  5, real delta = +0
#4; quantile =  5, real delta = +0
------ a = 0.5 ------
#0; quantile =  8, real delta = +1
#1; quantile =  8, real delta = +1
#2; quantile =  7, real delta = +0
#3; quantile =  7, real delta = +0
#4; quantile =  8, real delta = +1
------ a = 0.7 ------
#0; quantile =  8, real delta = +0
#1; quantile =  8, real delta = +0
#2; quantile =  8, real delta = +0
#3; quantile =  8, real delta = +0
#4; quantile =  8, real delta = +0
======= q = 100000 =======:
------ a = 0.1 ------
#0; quantile =  5, real delta = +0
#1; quantile =  5, real delta = +0
#2; quantile =  5, real delta = +0
#3; quantile =  5, real delta = +0
#4; quantile =  5, real delta = +0
------ a = 0.5 ------
#0; quantile =  8, real delta = +1
#1; quantile =  8, real delta = +1
#2; quantile =  7, real delta = +0
#3; quantile =  8, real delta = +1
#4; quantile =  7, real delta = +0
------ a = 0.7 ------
#0; quantile =  8, real delta = +0
#1; quantile =  8, real delta = +0
#2; quantile =  8, real delta = +0
#3; quantile =  8, real delta = +0
#4; quantile =  8, real delta = +0
```

Как и должно было быть, разность между выборочными квантилями и теоретическими стремится к нулю.

> Вообще говоря, для вычисления квантилей выборок можно было воспользоваться функцией `np.quantile` (https://docs.scipy.org/doc/numpy/reference/generated/numpy.quantile.html), но это было бы слишком просто...

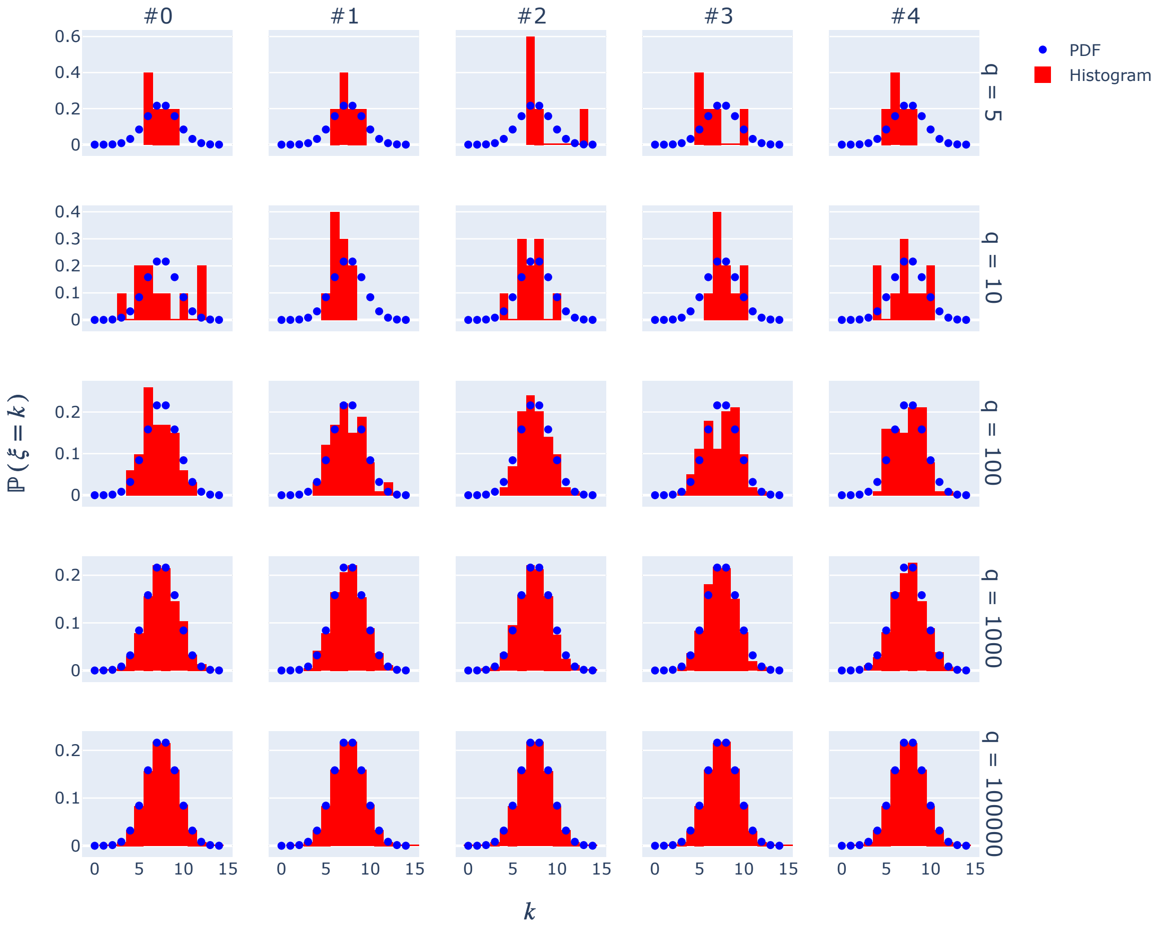
Гистограмма и полигон частот

Для каждого $q \in \{5, 10, 100, 1000, 10^5\}$ построим гистограмму, а также сравним полученные графики с функцией вероятности $\mathbb{P}(\xi = k)$. Полигон частот строить не будем, потому что по сути, это просто альтернативный способ представления гистограммы, который будет нам только мешать анализировать графики.

In [329]:
```
HG_hists_fig = plotly.subplots.make_subplots(rows=len(QS),
                                              cols=N,
                                              shared_xaxes='all',
                                              shared_yaxes=True,
                                              row_titles=[f'q = {q}' for q in QS],
                                              column_titles=[f'#{i}' for i in range(N)],
                                              x_title=r'$k$',
                                              y_title=r'$\mathbb{P}\left(\,\xi = k\,\right)$')

for i, q in enumerate(samples_hg):
    for j, sample in enumerate(samples_hg[q]):
        theoretical_hg_PDF_trace.showlegend = (i + j == 0)
        HG_hists_fig.add_trace(theoretical_hg_PDF_trace, row=i+1, col=j+1)
        HG_hists_fig.add_trace(
            go.Histogram(
                name='Histogram',
                x=sample,
                histnorm='probability',
                marker_color='red',
                legendgroup='PDF',
                showlegend=(i + j == 0),
            ),
            row=i+1,
            col=j+1)

HG_hists_fig.update_layout(height=800, width=900)
HG_hists_fig.show()
```



Как мы видим, закон больших чисел подтверждается графиками: с ростом объема выборки, значения гистограммы выборок все сильнее приближаю
к истинному функции распределения.

Нормальное распределение

С помощью формулы из справочника по вероятностным распределениям (http://zyurvas.narod.ru/knigi/Vadzinski_Ver_raspr.pdf), мы можем получить сра
две реализации x_i и x_{i+1} из стандартного нормального распределения $N(0, 1)$, имея две реализации r_i и r_{i+1} из стандартного равномерного распределения на отрезке $[0, 1]$:

$$x_i = \sqrt{-2\ln r_i}\sin(2\pi r_{i+1})$$
$$x_{i+1} = \sqrt{-2\ln r_i}\cos(2\pi r_{i+1})$$

Чтобы получить реализацию y_i из $N(\mu, \sigma)$:

$$y_i = \mu + \sigma x_i$$

Для простоты, мы будем использовать только одну y_i из двух возможных за раз:

In [330]:
```
@njit
def gen_norm(mu: float, sigma: float) -> float:
    return mu + sigma * \
      np.sqrt(-2 * np.log(np.random.rand())) * \
      np.cos(2 * np.pi * np.random.rand())

Norm.gen = lambda self: gen_norm(self.mu, self.sigma)
```

Генерация выборок

> Как говорилось ранее, объем выборки мы будем обозначать q.

Напишем метод `HG.sample_norm(q)` , которsq будет возвращать выборку объема q из $N(\mu, \sigma)$:

In [331]:
```
def sample_norm(mu: float, sigma: float, q: int) -> np.ndarray:
    return np.fromiter((gen_norm(mu, sigma) for _ in range(q)), np.float)

Norm.sample = lambda self, q: sample_norm(self.mu, self.sigma, q)
```

Для каждого $q \in \{5, 10, 100, 1000, 10^5\}$ сгенерируем по 5 выборок $X_\eta^{q,i}$ объема q, где $i \in \overline{1, 5}$

In [332]:
```
samples_norm = {}

for q in QS:
    samples_norm[q] = np.ndarray(shape=(N, q), dtype=np.float64)
    for i in range(N):
        samples_norm[q][i] = eta.sample(q)
```

Теперь `samples_norm` – словарь, где ключом является объемом выборки, а значением по соответствующему ключу – массив из N = 5 выборок соответствующего объема. Выведем выборки для q = 5 и q = 10:

In [333]:
```
for q in (5, 10):
    print(f'====== q = {q} ======:')
    for i, s in enumerate(samples_norm[q]):
        print(f'#{i+1}: {s}')
```

```
====== q = 5 ======:
#1: [-0.31698 -1.58418 -0.75019  0.02482  1.18862]
#2: [ 1.34723 -0.65077 -1.87313 -1.0329   1.62389]
#3: [-1.46068 -0.16291  0.86098 -0.62396  0.92987]
#4: [-0.19471 -0.09904 -0.59712  0.53845 -0.09015]
#5: [ 1.29836  1.68843 -0.64803  1.10838  0.9246 ]
====== q = 10 ======:
#1: [ 0.75566 -0.13915  1.1515   0.9173   0.05405 -0.39073 -0.36434  0.87139  0.44575  0.28161]
#2: [ 1.37354  0.33713  0.5687  -0.25807  0.23344  0.88738  0.03075  1.40302  0.52142  0.6737 ]
#3: [-1.03787  1.03785 -0.15184 -0.03321  1.51633  1.22456 -0.80803  1.206   -0.8897   0.20689]
#4: [ 0.91409  0.18758 -0.72084 -0.00215  0.63283  0.1114  -1.35266  0.30411  0.7137   0.21824]
#5: [-0.30202  2.49612  0.57886 -0.66788 -1.69219 -0.78671  0.21034 -1.40328  0.12099 -1.11997]
```

Эмпирическая функция распределения

Для каждой выбоки вычислим эмпирическую функцию распределения (*ECDF*) с помощью функции `statsmodels.distributions.empirical_distribution.ECDF` (http://www.statsmodels.org/stable/generated/statsmodels.distributions.empirical_distribution.ECDF.html):

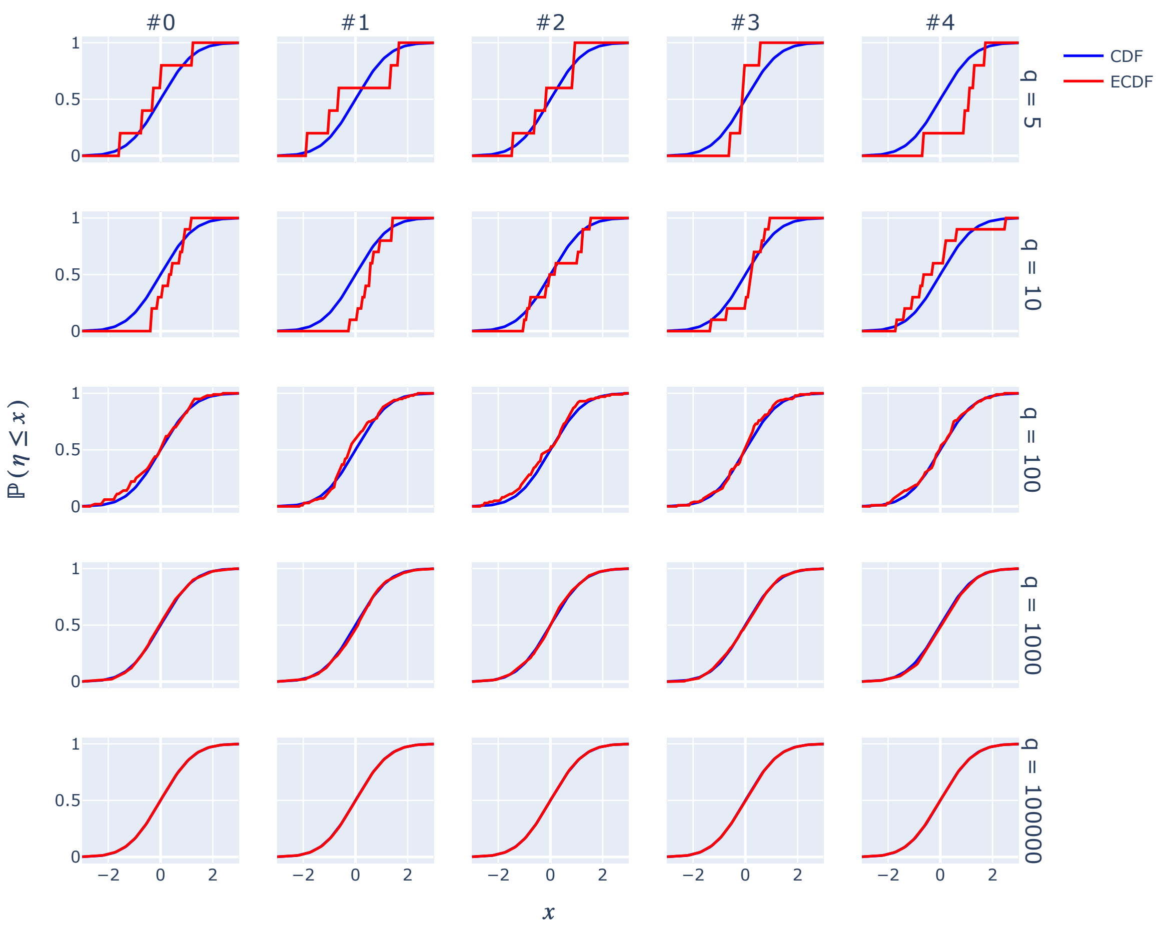
In [334]:
```
Norm_ECDFs = {}
for q in samples_norm:
    Norm_ECDFs[q] = np.ndarray(shape=(samples_norm[q].shape[0], norm_data_x.shape[0]), dtype=np.float)
    for i, sample in enumerate(samples_norm[q]):
        Norm_ECDFs[q][i] = ECDF(sample)(norm_data_x)
```

Теперь для каждой выборки построим эмпирическую функцию распределения, а также сравним получившуюся функцию с теоретической функцией распределения:

In [335]:
```
Norm_ECDFs_fig = plotly.subplots.make_subplots(rows=len(QS),
                                               cols=N,
                                               shared_xaxes='all',
                                               shared_yaxes='all',
                                               row_titles=[f'q = {q}' for q in QS],
                                               column_titles=[f'#{i}' for i in range(N)],
                                               x_title=r'$x$',
                                               y_title=r'$\mathbb{P}\left(\,\eta\leq x\,\right)$')

for i, q in enumerate(Norm_ECDFs):
    for j, sample in enumerate(Norm_ECDFs[q]):
        theoretical_norm_CDF_trace.showlegend = (i + j == 0)
        Norm_ECDFs_fig.add_trace(theoretical_norm_CDF_trace, row=i+1, col=j+1)
        Norm_ECDFs_fig.add_trace(
            go.Scatter(
                name='ECDF',
                x=norm_data_x,
                y=Norm_ECDFs[q][j],
                line=go.scatter.Line(
                    color='red',
                ),
                legendgroup='CDF',
                showlegend=(i + j == 0),
            ),
            row=i+1,
            col=j+1)

Norm_ECDFs_fig.update_layout(height=800, width=900)
Norm_ECDFs_fig.show()
```



Тут опять все хорошо, значения ECDF стремятся к CDF при стремлении объема выборки к бесконечности.

Для каждого $q \in \{5, 10, 100, 1000, 10^5\}$ найдем верхнюю границу разности каждой пары эмпирических функций распределения $\Delta_q^{i,j}$:

$$\Delta_q^{i,j} = \sup_{y \in \operatorname{supp} \eta} \left(\hat{F}\left(y, \vec{x_i}\right) - \hat{F}\left(y, \vec{x_j}\right) \right), \quad i,j \in \overline{0,5}$$

In [336]:
```
for q in Norm_ECDFs:
    print(f'====== q = {q} ======:')
    for i, ECDF_i in enumerate(Norm_ECDFs[q]):
        for j, ECDF_j in enumerate(Norm_ECDFs[q][i+1:], start=i+1):
            print(f'i = {i}; j = {j}; max delta = {np.absolute(ECDF_i - ECDF_j).max():.4f}')
```

```
====== q = 5 ======:
i = 0; j = 1; max delta = 0.4000
i = 0; j = 2; max delta = 0.2000
i = 0; j = 3; max delta = 0.4000
i = 0; j = 4; max delta = 0.6000
i = 1; j = 2; max delta = 0.4000
i = 1; j = 3; max delta = 0.6000
i = 1; j = 4; max delta = 0.4000
i = 2; j = 3; max delta = 0.4000
i = 2; j = 4; max delta = 0.6000
i = 3; j = 4; max delta = 0.8000
====== q = 10 ======:
i = 0; j = 1; max delta = 0.2000
i = 0; j = 2; max delta = 0.3000
i = 0; j = 3; max delta = 0.2000
i = 0; j = 4; max delta = 0.5000
i = 1; j = 2; max delta = 0.4000
i = 1; j = 3; max delta = 0.4000
i = 1; j = 4; max delta = 0.6000
i = 2; j = 3; max delta = 0.4000
i = 2; j = 4; max delta = 0.3000
i = 3; j = 4; max delta = 0.4000
====== q = 100 ======:
i = 0; j = 1; max delta = 0.1200
i = 0; j = 2; max delta = 0.0800
i = 0; j = 3; max delta = 0.1000
i = 0; j = 4; max delta = 0.0700
i = 1; j = 2; max delta = 0.1100
i = 1; j = 3; max delta = 0.1400
i = 1; j = 4; max delta = 0.1000
i = 2; j = 3; max delta = 0.1000
i = 2; j = 4; max delta = 0.1200
i = 3; j = 4; max delta = 0.0800
====== q = 1000 ======:
i = 0; j = 1; max delta = 0.0520
i = 0; j = 2; max delta = 0.0310
i = 0; j = 3; max delta = 0.0250
i = 0; j = 4; max delta = 0.0460
i = 1; j = 2; max delta = 0.0500
i = 1; j = 3; max delta = 0.0380
i = 1; j = 4; max delta = 0.0330
i = 2; j = 3; max delta = 0.0330
i = 2; j = 4; max delta = 0.0540
i = 3; j = 4; max delta = 0.0450
====== q = 100000 ======:
i = 0; j = 1; max delta = 0.0043
i = 0; j = 2; max delta = 0.0032
i = 0; j = 3; max delta = 0.0021
i = 0; j = 4; max delta = 0.0034
i = 1; j = 2; max delta = 0.0033
i = 1; j = 3; max delta = 0.0048
i = 1; j = 4; max delta = 0.0066
i = 2; j = 3; max delta = 0.0037
i = 2; j = 4; max delta = 0.0043
i = 3; j = 4; max delta = 0.0033
```

Построение вариационного ряда выборки

Построение вариационного ряда выборки

Для каждого $q \in \{5, 10, 100, 1000, 10^5\}$ построим вариационный ряд выборки:

In [337]:
```
Norm_VAR_ROWS = {}
for q in samples_norm:
    Norm_VAR_ROWS[q] = np.ndarray(shape=samples_norm[q].shape, dtype=np.float64)
    for i, sample in enumerate(samples_norm[q]):
        Norm_VAR_ROWS[q][i] = np.sort(sample)
```

Выведем получившиеся упорядоченные выборки для `q = 5` и `q = 10` :

In [338]:
```
for q in (5, 10):
    print(f'====== q = {q} ======:')
    for i, s in enumerate(Norm_VAR_ROWS[q]):
        print(f'#{i+1}: {s}')
```

```
====== q = 5 ======:
#1: [-1.58418 -0.75019 -0.31698  0.02482  1.18862]
#2: [-1.87313 -1.0329  -0.65077  1.34723  1.62389]
#3: [-1.46068 -0.62396 -0.16291  0.86098  0.92987]
#4: [-0.59712 -0.19471 -0.09904 -0.09015  0.53845]
#5: [-0.64803  0.9246   1.10838  1.29836  1.68843]
====== q = 10 ======:
#1: [-0.39073 -0.36434 -0.13915  0.05405  0.28161  0.44575  0.75566  0.87139  0.9173   1.1515 ]
#2: [-0.25807  0.03075  0.23344  0.33713  0.52142  0.5687   0.6737   0.88738  1.37354  1.40302]
#3: [-1.03787 -0.8897  -0.80803 -0.15184 -0.03321  0.20689  1.03785  1.206    1.22456  1.51633]
#4: [-1.35266 -0.72084 -0.00215  0.1114   0.18758  0.21824  0.30411  0.63283  0.7137   0.91409]
#5: [-1.69219 -1.40328 -1.11997 -0.78671 -0.66788 -0.30202  0.12099  0.21034  0.57886  2.49612]
```

Квантили

Найдем квантили уровней $\alpha \in \{0.1, 0.5, 0.7\}$ для нормального распределения $N(0, 1^2)$:

In [339]:
```
norm_quantiles = {}
for a in (.1, .5, .7):
    for i, y in enumerate(theoretical_norm_CDF):
        if y >= a:
            norm_quantiles[a] = norm_data_x[i]
            print(f'a = {a}, quantile = {norm_quantiles[a]:+.4f}')
            break
```

```
a = 0.1, quantile = -1.2424
a = 0.5, quantile = +0.0303
a = 0.7, quantile = +0.5758
```

Теперь для каждой выборки найдем *выборочные квантили* уровней $\alpha \in \{0.1, 0.5, 0.7\}$ и сравним их с соответствующими квантилями данного распределения:

In [340]:
```
for q in QS:
    print(f'======= q = {q} =======:')
    for a in (.1, .5, .7):
        print(f'------ a = {a} ------')
        for i, sample in enumerate(Norm_VAR_ROWS[q]):
            quantile = sample[int(a*sample.shape[0])]
            print(f'#{i}; quantile = {quantile:+2.3f}, real delta = {quantile - norm_quantiles[a]:+2.3f}')
```

```
======= q = 5 =======:
------ a = 0.1 ------
#0; quantile = -1.584, real delta = -0.342
#1; quantile = -1.873, real delta = -0.631
#2; quantile = -1.461, real delta = -0.218
#3; quantile = -0.597, real delta = +0.645
#4; quantile = -0.648, real delta = +0.594
------ a = 0.5 ------
#0; quantile = -0.317, real delta = -0.347
#1; quantile = -0.651, real delta = -0.681
#2; quantile = -0.163, real delta = -0.193
#3; quantile = -0.099, real delta = -0.129
#4; quantile = +1.108, real delta = +1.078
------ a = 0.7 ------
#0; quantile = +0.025, real delta = -0.551
#1; quantile = +1.347, real delta = +0.771
#2; quantile = +0.861, real delta = +0.285
#3; quantile = -0.090, real delta = -0.666
#4; quantile = +1.298, real delta = +0.723
======= q = 10 =======:
------ a = 0.1 ------
#0; quantile = -0.364, real delta = +0.878
#1; quantile = +0.031, real delta = +1.273
#2; quantile = -0.890, real delta = +0.353
#3; quantile = -0.721, real delta = +0.522
#4; quantile = -1.403, real delta = -0.161
------ a = 0.5 ------
#0; quantile = +0.446, real delta = +0.415
#1; quantile = +0.569, real delta = +0.538
#2; quantile = +0.207, real delta = +0.177
#3; quantile = +0.218, real delta = +0.188
#4; quantile = -0.302, real delta = -0.332
------ a = 0.7 ------
#0; quantile = +0.871, real delta = +0.296
#1; quantile = +0.887, real delta = +0.312
#2; quantile = +1.206, real delta = +0.630
#3; quantile = +0.633, real delta = +0.057
#4; quantile = +0.210, real delta = -0.365
======= q = 100 =======:
------ a = 0.1 ------
#0; quantile = -1.689, real delta = -0.446
#1; quantile = -1.108, real delta = +0.134
```

```
#2; quantile = -1.631, real delta = -0.388
#3; quantile = -1.325, real delta = -0.082
#4; quantile = -1.570, real delta = -0.328
------ a = 0.5 ------
#0; quantile = -0.001, real delta = -0.031
#1; quantile = -0.260, real delta = -0.291
#2; quantile = -0.025, real delta = -0.056
#3; quantile = -0.028, real delta = -0.058
#4; quantile = -0.028, real delta = -0.059
------ a = 0.7 ------
#0; quantile = +0.592, real delta = +0.016
#1; quantile = +0.339, real delta = -0.237
#2; quantile = +0.449, real delta = -0.126
#3; quantile = +0.318, real delta = -0.258
#4; quantile = +0.460, real delta = -0.116
======= q = 1000 =======:
------ a = 0.1 ------
#0; quantile = -1.270, real delta = -0.027
#1; quantile = -1.234, real delta = +0.009
#2; quantile = -1.347, real delta = -0.104
#3; quantile = -1.309, real delta = -0.067
#4; quantile = -1.226, real delta = +0.017
------ a = 0.5 ------
#0; quantile = -0.033, real delta = -0.064
#1; quantile = +0.093, real delta = +0.063
#2; quantile = +0.004, real delta = -0.027
#3; quantile = +0.023, real delta = -0.007
#4; quantile = +0.058, real delta = +0.027
------ a = 0.7 ------
#0; quantile = +0.477, real delta = -0.099
#1; quantile = +0.557, real delta = -0.019
#2; quantile = +0.472, real delta = -0.103
#3; quantile = +0.551, real delta = -0.025
#4; quantile = +0.603, real delta = +0.027
======= q = 100000 =======:
------ a = 0.1 ------
#0; quantile = -1.293, real delta = -0.050
#1; quantile = -1.271, real delta = -0.029
#2; quantile = -1.279, real delta = -0.037
#3; quantile = -1.287, real delta = -0.045
#4; quantile = -1.289, real delta = -0.047
------ a = 0.5 ------
#0; quantile = -0.001, real delta = -0.031
#1; quantile = +0.003, real delta = -0.027
#2; quantile = -0.003, real delta = -0.033
#3; quantile = +0.001, real delta = -0.029
#4; quantile = -0.003, real delta = -0.033
------ a = 0.7 ------
#0; quantile = +0.524, real delta = -0.052
#1; quantile = +0.527, real delta = -0.049
#2; quantile = +0.517, real delta = -0.059
#3; quantile = +0.527, real delta = -0.049
#4; quantile = +0.521, real delta = -0.054
```

Как и в случае с гипергеометрическим распределением, разность между выборочными квантилями и теоретическими стремится к нулю, что есть правильно.

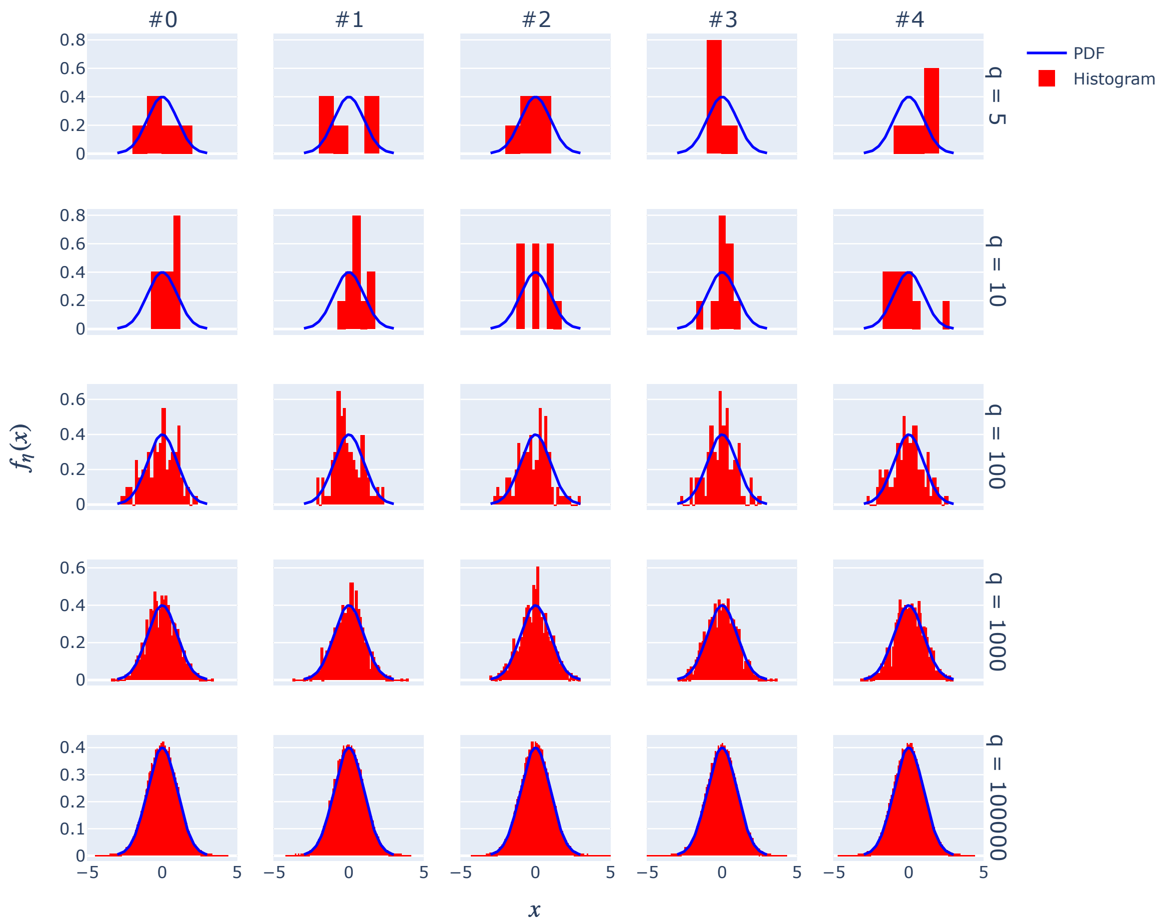
Гистограмма и полигон частот

Для каждого $q \in \{5, 10, 100, 1000, 10^5\}$ построим гистограмму, а также сравним полученные графики с плотностью распределения $f_\eta(x)$. Как и в соответствующем пункте про гипергеометрическое распределение, полигон частот строить не будем.

In [341]:
```
Norm_hists_fig = plotly.subplots.make_subplots(rows=len(QS),
                                    cols=N,
                                    shared_xaxes='all',
                                    shared_yaxes=True,
                                    row_titles=[f'q = {q}' for q in QS],
                                    column_titles=[f'#{i}' for i in range(N)],
                                    x_title=r'$x$',
                                    y_title=r'$f_\eta(x)$')

for i, q in enumerate(samples_norm):
    for j, sample in enumerate(samples_norm[q]):
        theoretical_norm_PDF_trace.showlegend = (i + j == 0)
        Norm_hists_fig.add_trace(theoretical_norm_PDF_trace, row=i+1, col=j+1)
        Norm_hists_fig.add_trace(
            go.Histogram(
                name='Histogram',
                x=sample,
                histnorm='probability density',
                marker_color='red',
                legendgroup='PDF',
                showlegend=(i + j == 0),
            ),
            row=i+1,
            col=j+1)

Norm_hists_fig.update_layout(height=800, width=900)
Norm_hists_fig.show()
```



Как мы видим, закон больших чисел снова подтверждается графиками: с ростом объема выборки, значения гистограммы выборок все сильнее приближаются к истинному значению плотности вероятности.

Задание 3. Оценки

Гипергеометрическое распределение

Выборочные среднее и дисперсия

Для начала, для каждой выборки найдем выборочное среднее $\overline{X}$, выборочную дисперсию S_n^2 и несмещенную выборочную дисперсию S^2 по формул

$$\overline{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$$
$$S_n^2 = \frac{1}{n}\sum_{i=1}^{n}\left(X_i - \overline{X}\right)^2$$
$$S^2 = \frac{n}{n-1}S_n^2$$

Также сравним их с истинным значением среднего (математического ожидания) или дисперсии нашей случайной величины:

In [342]:
```
for q in samples_hg:
    print(f'======= q = {q} =======:')
    for i, sample in enumerate(samples_hg[q]):
        mean = np.sum(sample)/sample.shape[0]
        variance = np.sum((sample-mean)**2)/sample.shape[0]
        unbiased_variance = sample.shape[0] / (sample.shape[0] - 1) * variance
        print(f'''#{i} mean: {mean:20.4f}, real delta: {mean - xi.expected:+.4f}
   variance: {variance:16.4f}, real delta: {variance - (sample.shape[0] - 1) / sample.shape[0] * xi.variance:+.4f}
   unbiased variance: {unbiased_variance:7.4f}, real delta: {unbiased_variance - xi.variance:+.4f}\n''')
```

```
======= q = 5 =======:
#0 mean:               7.2000, real delta: -0.3000
   variance:           1.3600, real delta: -1.1084
   unbiased variance:  1.7000, real delta: -1.3854

#1 mean:               7.4000, real delta: -0.1000
   variance:           1.0400, real delta: -1.4284
   unbiased variance:  1.3000, real delta: -1.7854

#2 mean:               8.4000, real delta: +0.9000
   variance:           5.4400, real delta: +2.9716
   unbiased variance:  6.8000, real delta: +3.7146

#3 mean:               6.6000, real delta: -0.9000
   variance:           3.4400, real delta: +0.9716
   unbiased variance:  4.3000, real delta: +1.2146

#4 mean:               6.4000, real delta: -1.1000
   variance:           1.0400, real delta: -1.4284
   unbiased variance:  1.3000, real delta: -1.7854

======= q = 10 =======:
#0 mean:               7.4000, real delta: -0.1000
   variance:           8.4400, real delta: +5.6631
   unbiased variance:  9.3778, real delta: +6.2923

#1 mean:               6.6000, real delta: -0.9000
   variance:           0.8400, real delta: -1.9369
   unbiased variance:  0.9333, real delta: -2.1521

#2 mean:               7.0000, real delta: -0.5000
   variance:           2.4000, real delta: -0.3769
   unbiased variance:  2.6667, real delta: -0.4188

#3 mean:               7.9000, real delta: +0.4000
   variance:           1.6900, real delta: -1.0869
   unbiased variance:  1.8778, real delta: -1.2077

#4 mean:               7.2000, real delta: -0.3000
   variance:           4.1600, real delta: +1.3831
   unbiased variance:  4.6222, real delta: +1.5368

======= q = 100 =======:
#0 mean:               7.1300, real delta: -0.3700
   variance:           2.9731, real delta: -0.0815
   unbiased variance:  3.0031, real delta: -0.0823

#1 mean:               7.4600, real delta: -0.0400
   variance:           3.2484, real delta: +0.1938
   unbiased variance:  3.2812, real delta: +0.1958

#2 mean:               7.5100, real delta: +0.0100
   variance:           2.6299, real delta: -0.4247
   unbiased variance:  2.6565, real delta: -0.4290

#3 mean:               7.4600, real delta: -0.0400
   variance:           3.5284, real delta: +0.4738
   unbiased variance:  3.5640, real delta: +0.4786
```

```
#4 mean:                7.4500, real delta: -0.0500
   variance:            2.8675, real delta: -0.1871
   unbiased variance:   2.8965, real delta: -0.1890

======= q = 1000 =======:
#0 mean:                7.5660, real delta: +0.0660
   variance:            3.1416, real delta: +0.0593
   unbiased variance:   3.1448, real delta: +0.0593

#1 mean:                7.5290, real delta: +0.0290
   variance:            3.0952, real delta: +0.0128
   unbiased variance:   3.0983, real delta: +0.0128

#2 mean:                7.4050, real delta: -0.0950
   variance:            3.1450, real delta: +0.0626
   unbiased variance:   3.1481, real delta: +0.0627

#3 mean:                7.3860, real delta: -0.1140
   variance:            2.8890, real delta: -0.1934
   unbiased variance:   2.8919, real delta: -0.1935

#4 mean:                7.5300, real delta: +0.0300
   variance:            3.0771, real delta: -0.0053
   unbiased variance:   3.0802, real delta: -0.0053

======= q = 100000 =======:
#0 mean:                7.5069, real delta: +0.0069
   variance:            3.0678, real delta: -0.0176
   unbiased variance:   3.0678, real delta: -0.0176

#1 mean:                7.5047, real delta: +0.0047
   variance:            3.0866, real delta: +0.0012
   unbiased variance:   3.0866, real delta: +0.0012

#2 mean:                7.4968, real delta: -0.0032
   variance:            3.0489, real delta: -0.0365
   unbiased variance:   3.0489, real delta: -0.0365

#3 mean:                7.5063, real delta: +0.0063
   variance:            3.0777, real delta: -0.0077
   unbiased variance:   3.0778, real delta: -0.0077

#4 mean:                7.5054, real delta: +0.0054
   variance:            3.0861, real delta: +0.0006
   unbiased variance:   3.0861, real delta: +0.0006
```

Нахождение параметров распределения

Мы будем оценивать параметр N для $HG(N, m, n)$. Это может быть полезно, если мы хотим оценить мощность множетсва, из которого ведется выборка, зная количество помеченных элементов m и объем выборки n. Для нахождения оценки мы будем использовать статистику $T\left(\overrightarrow{X}\right)$, определенную как выборочное среднее:

$$T\left(\overrightarrow{X}\right) = \overline{X} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Посчитаем математическое ожидание $T\left(\overrightarrow{X}\right)$, учитывая, что $\overrightarrow{X} = (X_1, X_2, \ldots, X_n)$ - назависимые одинаково распределенные случайные величины:

$$\begin{aligned}
\mathbb{E}\left[T\left(\overrightarrow{X}\right)\right] = \mathbb{E}\left[\overline{X}\right] = \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n} x_i\right] = \\
= \frac{1}{n}\mathbb{E}\left[\sum_{i=1}^{n} x_i\right] = \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}\left[\,x_i\,\right] = \\
= \frac{n}{n}\mathbb{E}\left[\,x_1\,\right] = \mathbb{E}\left[\,x_1\,\right] = \\
= \frac{m \cdot n}{N} \neq N
\end{aligned}$$

Таким образом, статистика $T\left(\overrightarrow{X}\right)$ является смещенной оценкой для параметра N.

С помощью метода моментов находим оценку $\hat{N} = \frac{m \cdot n}{\overline{X}}$. Оценка является состоятельной, так как:

$$\hat{N} \xrightarrow{\mathbb{P}} N$$

.

Теперь найдем значения оценки $\hat{N}$ от постоенных выборок и сравним эти значения с истинным значением N:

In [343]:
```
for q in samples_hg:
    print(f'======= q = {q} =======:')
    for i, sample in enumerate(samples_hg[q]):
        mean = np.sum(sample)/sample.shape[0]
        N_hat = xi.m * xi.n / mean
        print(f'''#{i} N_hat: {N_hat:7.4f},\treal delta: {N_hat - xi.N:+10.4f}''')
```

```
======= q = 5 =======:
#0 N_hat: 83.3333,      real delta:    +3.3333
#1 N_hat: 81.0811,      real delta:    +1.0811
#2 N_hat: 71.4286,      real delta:    -8.5714
#3 N_hat: 90.9091,      real delta:   +10.9091
#4 N_hat: 93.7500,      real delta:   +13.7500
======= q = 10 =======:
#0 N_hat: 81.0811,      real delta:    +1.0811
#1 N_hat: 90.9091,      real delta:   +10.9091
#2 N_hat: 85.7143,      real delta:    +5.7143
#3 N_hat: 75.9494,      real delta:    -4.0506
#4 N_hat: 83.3333,      real delta:    +3.3333
======= q = 100 =======:
#0 N_hat: 84.1515,      real delta:    +4.1515
#1 N_hat: 80.4290,      real delta:    +0.4290
#2 N_hat: 79.8935,      real delta:    -0.1065
#3 N_hat: 80.4290,      real delta:    +0.4290
#4 N_hat: 80.5369,      real delta:    +0.5369
======= q = 1000 =======:
#0 N_hat: 79.3021,      real delta:    -0.6979
#1 N_hat: 79.6919,      real delta:    -0.3081
#2 N_hat: 81.0263,      real delta:    +1.0263
#3 N_hat: 81.2348,      real delta:    +1.2348
#4 N_hat: 79.6813,      real delta:    -0.3187
======= q = 100000 =======:
#0 N_hat: 79.9268,      real delta:    -0.0732
#1 N_hat: 79.9496,      real delta:    -0.0504
#2 N_hat: 80.0346,      real delta:    +0.0346
#3 N_hat: 79.9325,      real delta:    -0.0675
#4 N_hat: 79.9419,      real delta:    -0.0581
```

Как можно заметить, при увеличении объема выборки, разность между значением оценки и оцениваемым параметром $\hat{N} - N$ стремится к нулю, что подтверждает то, что нашал оценка $\hat{N}$ является состоятельной.

Нормальное распределение

Выборочные среднее и дисперсия

Для каждой выборки найдем выборочное среднее, выборочную дисперсию и несмещенную выборочную дисперсию и сравним их с истинными значениями для нашео распределения:

In [344]:
```
for q in samples_norm:
    print(f'======= q = {q} =======:')
    for i, sample in enumerate(samples_norm[q]):
        mean = np.sum(sample)/sample.shape[0]
        variance = np.sum((sample - mean) ** 2) / sample.shape[0]
        unbiased_variance = sample.shape[0] / (sample.shape[0] - 1) * variance
        print(f'''#{i} mean: {mean:20.4f}, real delta: {mean - eta.expected:+.4f}
   variance: {variance:16.4f}, real delta: {variance - (sample.shape[0] - 1) / sample.shape[0] * eta.variance:+.4f}
   unbiased variance: {unbiased_variance:7.4f}, real delta: {unbiased_variance - eta.variance:+.4f}\n''')
```

```
======= q = 5 =======:
#0 mean:               -0.2876, real delta: -0.2876
   variance:            0.8346, real delta: +0.0346
   unbiased variance:  1.0432, real delta: +0.0432

#1 mean:               -0.1171, real delta: -0.1171
   variance:            1.8765, real delta: +1.0765
   unbiased variance:  2.3456, real delta: +1.3456

#2 mean:               -0.0913, real delta: -0.0913
   variance:            0.8227, real delta: +0.0227
   unbiased variance:  1.0284, real delta: +0.0284

#3 mean:               -0.0885, real delta: -0.0885
   variance:            0.1326, real delta: -0.6674
   unbiased variance:  0.1658, real delta: -0.8342

#4 mean:                0.8743, real delta: +0.8743
   variance:            0.6435, real delta: -0.1565
   unbiased variance:  0.8044, real delta: -0.1956

======= q = 10 =======:
#0 mean:                0.3583, real delta: +0.3583
   variance:            0.2800, real delta: -0.6200
   unbiased variance:  0.3111, real delta: -0.6889

#1 mean:                0.5771, real delta: +0.5771
```

```
   variance:           0.2597, real delta: -0.6403
   unbiased variance:  0.2885, real delta: -0.7115

#2 mean:               0.2271, real delta: +0.2271
   variance:           0.8403, real delta: -0.0597
   unbiased variance:  0.9337, real delta: -0.0663

#3 mean:               0.1006, real delta: +0.1006
   variance:           0.4181, real delta: -0.4819
   unbiased variance:  0.4646, real delta: -0.5354

#4 mean:              -0.2566, real delta: -0.2566
   variance:           1.3210, real delta: +0.4210
   unbiased variance:  1.4677, real delta: +0.4677

======= q = 100 =======:
#0 mean:              -0.1325, real delta: -0.1325
   variance:           1.1600, real delta: +0.1700
   unbiased variance:  1.1717, real delta: +0.1717

#1 mean:              -0.0838, real delta: -0.0838
   variance:           0.9054, real delta: -0.0846
   unbiased variance:  0.9146, real delta: -0.0854

#2 mean:              -0.1571, real delta: -0.1571
   variance:           1.1745, real delta: +0.1845
   unbiased variance:  1.1864, real delta: +0.1864

#3 mean:              -0.0649, real delta: -0.0649
   variance:           0.9272, real delta: -0.0628
   unbiased variance:  0.9365, real delta: -0.0635

#4 mean:              -0.0533, real delta: -0.0533
   variance:           1.0913, real delta: +0.1013
   unbiased variance:  1.1023, real delta: +0.1023

======= q = 1000 =======:
#0 mean:              -0.0082, real delta: -0.0082
   variance:           0.9697, real delta: -0.0293
   unbiased variance:  0.9707, real delta: -0.0293

#1 mean:               0.0355, real delta: +0.0355
   variance:           0.9926, real delta: -0.0064
   unbiased variance:  0.9936, real delta: -0.0064

#2 mean:              -0.0106, real delta: -0.0106
   variance:           0.9713, real delta: -0.0277
   unbiased variance:  0.9723, real delta: -0.0277

#3 mean:              -0.0068, real delta: -0.0068
   variance:           0.9967, real delta: -0.0023
   unbiased variance:  0.9977, real delta: -0.0023

#4 mean:               0.0564, real delta: +0.0564
   variance:           0.9588, real delta: -0.0402
   unbiased variance:  0.9598, real delta: -0.0402

======= q = 100000 =======:
#0 mean:              -0.0031, real delta: -0.0031
   variance:           0.9997, real delta: -0.0003
   unbiased variance:  0.9997, real delta: -0.0003

#1 mean:               0.0038, real delta: +0.0038
   variance:           0.9992, real delta: -0.0008
   unbiased variance:  0.9992, real delta: -0.0008

#2 mean:              -0.0021, real delta: -0.0021
   variance:           0.9937, real delta: -0.0063
   unbiased variance:  0.9937, real delta: -0.0063

#3 mean:              -0.0002, real delta: -0.0002
   variance:           1.0071, real delta: +0.0071
   unbiased variance:  1.0071, real delta: +0.0071

#4 mean:              -0.0048, real delta: -0.0048
   variance:           1.0085, real delta: +0.0085
   unbiased variance:  1.0085, real delta: +0.0085
```

Нахождение параметров распределения

Будем оценивать параметр μ для $N(\mu, \sigma^2)$. Здесь интуиция сразу подсказывает опять брать статистику $T\left(\overrightarrow{X}\right) = \overline{X}$, потому что математическое ожидание нормального рспределения равно μ.

$$\mathbb{E}\left[\overline{X}\right] = \mathbb{E}\left[X_1\right] = \mu$$

Как и ожидалось, статистика $T\left(\overrightarrow{X}\right)$ является несмещенной оценкой для параметра μ. Также она является состоятельной и эффективной.

Теперь найдем значения оценки $\hat{\mu}$ от постоенных выборок и сравним эти значения с истинным значением μ:

In [345]:
```
for q in samples_norm:
    print(f'======= q = {q} =======:')
    for i, sample in enumerate(samples_norm[q]):
        mu_hat = np.sum(sample) / sample.shape[0]
        print(f'''#{i} mu_hat: {mu_hat:7.4f},\treal delta: {mu_hat - eta.mu:+10.4f}''')
```

```
======= q = 5 =======:
#0 mu_hat: -0.2876,     real delta:    -0.2876
#1 mu_hat: -0.1171,     real delta:    -0.1171
#2 mu_hat: -0.0913,     real delta:    -0.0913
#3 mu_hat: -0.0885,     real delta:    -0.0885
#4 mu_hat:  0.8743,     real delta:    +0.8743
======= q = 10 =======:
#0 mu_hat:  0.3583,     real delta:    +0.3583
#1 mu_hat:  0.5771,     real delta:    +0.5771
#2 mu_hat:  0.2271,     real delta:    +0.2271
#3 mu_hat:  0.1006,     real delta:    +0.1006
#4 mu_hat: -0.2566,     real delta:    -0.2566
======= q = 100 =======:
#0 mu_hat: -0.1325,     real delta:    -0.1325
#1 mu_hat: -0.0838,     real delta:    -0.0838
#2 mu_hat: -0.1571,     real delta:    -0.1571
#3 mu_hat: -0.0649,     real delta:    -0.0649
#4 mu_hat: -0.0533,     real delta:    -0.0533
======= q = 1000 =======:
#0 mu_hat: -0.0082,     real delta:    -0.0082
#1 mu_hat:  0.0355,     real delta:    +0.0355
#2 mu_hat: -0.0106,     real delta:    -0.0106
#3 mu_hat: -0.0068,     real delta:    -0.0068
#4 mu_hat:  0.0564,     real delta:    +0.0564
======= q = 100000 =======:
#0 mu_hat: -0.0031,     real delta:    -0.0031
#1 mu_hat:  0.0038,     real delta:    +0.0038
#2 mu_hat: -0.0021,     real delta:    -0.0021
#3 mu_hat: -0.0002,     real delta:    -0.0002
#4 mu_hat: -0.0048,     real delta:    -0.0048
```

Как можно заметить, при увеличении объема выборки, разность между значением оценки и оцениваемым параметром $\hat{\mu} - \mu$ стремится к нулю, что подтверждает то, что нашал оценка $\hat{\mu}$ является состоятельной.

Теперь попытаемся оценить параметр σ^2 для $N(\mu, \sigma^2)$. Интуиция снова нас не подводит и предлагает использовать выборочную дисперсию:

$$T\left(\overrightarrow{X}\right) = S_n^2 = \frac{1}{n}\sum_{i=1}^{n}\left(X_i - \overline{X}\right)^2$$

Судя по названию, *выборочная дисперсия* должна оказаться *несмещенной* оценкой для нашего параметра σ^2, который как раз и является дисперси нашего распределения. Проверим это:

$$\mathbb{E}\left[S^2\right] = \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}\left(X_i - \overline{X}\right)^2\right] = \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}\left[\left(X_i - \overline{X}\right)^2\right] =$$
$$= \mathbb{E}\left[\left(X_i - \overline{X}\right)^2\right] = \sigma^2$$

In [346]:
```
for q in samples_norm:
    print(f'======= q = {q} =======:')
    for i, sample in enumerate(samples_norm[q]):
        mean = np.sum(sample)/sample.shape[0]
        sigma_2_hat = np.sum((sample - mean) ** 2) / sample.shape[0]
        print(f'''#{i} sigma_2_hat: {sigma_2_hat:7.4f},\treal delta: {sigma_2_hat - eta.sigma ** 2:+10.4f}''')
```

```
======= q = 5 =======:
#0 sigma_2_hat:  0.8346,        real delta:    -0.1654
#1 sigma_2_hat:  1.8765,        real delta:    +0.8765
#2 sigma_2_hat:  0.8227,        real delta:    -0.1773
#3 sigma_2_hat:  0.1326,        real delta:    -0.8674
#4 sigma_2_hat:  0.6435,        real delta:    -0.3565
======= q = 10 =======:
#0 sigma_2_hat:  0.2800,        real delta:    -0.7200
#1 sigma_2_hat:  0.2597,        real delta:    -0.7403
#2 sigma_2_hat:  0.8403,        real delta:    -0.1597
#3 sigma_2_hat:  0.4181,        real delta:    -0.5819
#4 sigma_2_hat:  1.3210,        real delta:    +0.3210
======= q = 100 =======:
#0 sigma_2_hat:  1.1600,        real delta:    +0.1600
#1 sigma_2_hat:  0.9054,        real delta:    -0.0946
#2 sigma_2_hat:  1.1745,        real delta:    +0.1745
#3 sigma_2_hat:  0.9272,        real delta:    -0.0728
#4 sigma_2_hat:  1.0913,        real delta:    +0.0913
======= q = 1000 =======:
#0 sigma_2_hat:  0.9697,        real delta:    -0.0303
#1 sigma_2_hat:  0.9926,        real delta:    -0.0074
#2 sigma_2_hat:  0.9713,        real delta:    -0.0287
#3 sigma_2_hat:  0.9967,        real delta:    -0.0033
#4 sigma_2_hat:  0.9588,        real delta:    -0.0412
======= q = 100000 =======:
#0 sigma_2_hat:  0.9997,        real delta:    -0.0003
#1 sigma_2_hat:  0.9992,        real delta:    -0.0008
#2 sigma_2_hat:  0.9937,        real delta:    -0.0063
#3 sigma_2_hat:  1.0071,        real delta:    +0.0071
#4 sigma_2_hat:  1.0085,        real delta:    +0.0085
```

Как можно заметить, при увеличении объема выборки, разность между значением оценки и оцениваемым параметром $\hat{\sigma^2} - \sigma^2$ стремится к нулю, что подтверждает то, что наша оценка $\hat{\sigma^2}$ является состоятельной.

Задание 4. Проверка гипотиз о виде распределения

4.1. Проверка гипотез о виде распределения

Определим уровни значимости $\alpha \in \{0.1, 0.05\}$ и объемы выборок для наших критериев:

In [347]:
```
sign_levels = (0.1, 0.05)
test_volumes = [q for q in QS if q >= 10**3]
```

Критерий согласия Колмогорова для простой гипотезы

Сам *критерий Колмогорова* формулируется следующим образом:

> Если объем $n \geq 20$ выборки X_n и при выбранном уровне значимости α число λ_α определено соотношением $K(\lambda_\alpha) = 1 - \alpha$, то
> $$\boxed{\{H_0 \text{ отвергается } \} \iff \left\{\sqrt{n} \cdot D_n(X) \geq \lambda_\alpha\right\}}$$
> , где:
>
> - H_0 - гипотеза, утверждающая, что распредление ξ имеет функцию распределения $F_\xi(x) = F(x)$, где $F(x)$ полностью задана
> - $K(x)$ - распределение Колмогорова, имеющее функцию распределения $F_K(x)$:
> $$F_K(x) = \begin{cases} \sum\limits_{k \in \mathbb{Z}} (-1)^k e^{-2k^2x^2}, & x > 0; \\ 0, & x \leq 0. \end{cases}$$
> - $D_n(X)$ определяется как максимум (по $x \in \mathbb{R}$) квадрата отклонения эмпирической функции распределения $\hat{F_X}(x)$, построенной по выборке X из распределения ξ, от теоретической функции распределения $F_\xi(x)$:
> $$D_n(X) = \sup_{x \in \mathbb{R}} \left|\hat{F_X}(x) - F_\xi(x)\right|$$

В практических целях, статистику $D_n(X)$ бывает удобнее вычислять по формуле $D_n(X) = \max(D_n^+(X), D_n^-(X))$, где
$$D_n^+(X) = \max_{k \in \overline{1,n}} \left(\frac{k}{n} - F(X_{(k)})\right)$$
$$D_n^-(X) = \max_{k \in \overline{1,n}} \left(F(X_{(k)}) - \frac{k-1}{n}\right)$$
, а $X_{(1)} \leq X_{(2)} \leq \cdots \leq X_{(n)}$ - вариационный ряд выборки.

Также, предлагается вместо статистики $D_n(X)$ использовать следующий вид статистики с поправкой Большева, который сходится к распределению Колмогорова со скоростью $O\left(\frac{1}{n^2}\right)$:
$$S_n(X) = \frac{6n \cdot D_n(X) + 1}{6\sqrt{n}}$$
Это позволит использовать данный критерий при меньших объемах выборки.

Преимущества

- Невысокая нижняя граница для объема выборки $n \geq 20$

Недостатки

- Применение данного критерия возможно только для непрерывных распределений (о множестве значений случайной величины обычно судят, исходя из её физической природы)

Реализация для непрерывного распределения

Для начала, определим функцию $D_n(X)$:

In [348]:
```
@njit
def Dn(cdf_data: np.ndarray, ecdf_data: np.ndarray) -> np.float64:
    return np.absolute(ecdf_data - cdf_data).max()
```

Будем использовать функцию `cdf(x)` класса распределения Колмогорова `scipy.stats.kstwobign` (https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.kstwobign.html#scipy.stats.kstwobign), которая возвращает значение вероятности Колмогорова для:
$$P\left(\sqrt{n} \cdot D_n(X) \leq x\right)$$

Истинная гипотеза

Возьмем выбоки X_η^q объемов $q \in \{10^3, 10^5\}$ из нормального распределения $N(0, 1^2)$ и для каждого уровня значимости $\alpha \in \{0.1, 0.05\}$ посчитаем, выполняется ли критерий Колмогорова для гипотезы H_0, утверждающей, имеет ли исследуемая случайная величина нормальное распределение $N(0, 1^2)$. Если критерий Колмогорова выполняется, то мы принимаем гипотезу H_0, иначе - отвергаем.

In [381]:
```
for sign_level in sign_levels:
    print(f'\n=================== Уровень значимости: {sign_level:4} ====================')
    for q in test_volumes:
        print(f'--------------------- Объем выборки: {q:6} ---------------------')
        for i, ecdf in enumerate(Norm_ECDFs[q]):
            D = Dn(theoretical_norm_CDF, ecdf)
            Kn = np.sqrt(q)*D
            prob_deny = sp.stats.kstwobign.cdf(Kn)
            deny = (prob_deny >= 1 - sign_level)
            print(f'#{i+1}: P( sqrt(n)*Dn <= {Kn:.2f} ) = {prob_deny:.4f} {"≤" if deny else ">"} ' \
            f'{sign_level:.4f} → H0 {"не" if not deny else "  "} отвергается')
```

```
=================== Уровень значимости:  0.1 ====================
--------------------- Объем выборки:   1000 ---------------------
#1: P( sqrt(n)*Dn <= 0.62 ) = 0.1599 > 0.1000 → H0 не отвергается
#2: P( sqrt(n)*Dn <= 1.24 ) = 0.9058 ≤ 0.1000 → H0    отвергается
#3: P( sqrt(n)*Dn <= 0.89 ) = 0.5871 > 0.1000 → H0 не отвергается
#4: P( sqrt(n)*Dn <= 0.56 ) = 0.0883 > 0.1000 → H0 не отвергается
#5: P( sqrt(n)*Dn <= 1.04 ) = 0.7665 > 0.1000 → H0 не отвергается
--------------------- Объем выборки: 100000 ---------------------
#1: P( sqrt(n)*Dn <= 0.61 ) = 0.1444 > 0.1000 → H0 не отвергается
#2: P( sqrt(n)*Dn <= 0.97 ) = 0.6983 > 0.1000 → H0 не отвергается
#3: P( sqrt(n)*Dn <= 0.80 ) = 0.4526 > 0.1000 → H0 не отвергается
#4: P( sqrt(n)*Dn <= 0.74 ) = 0.3614 > 0.1000 → H0 не отвергается
#5: P( sqrt(n)*Dn <= 1.27 ) = 0.9209 ≤ 0.1000 → H0    отвергается

=================== Уровень значимости: 0.05 ====================
--------------------- Объем выборки:   1000 ---------------------
#1: P( sqrt(n)*Dn <= 0.62 ) = 0.1599 > 0.0500 → H0 не отвергается
#2: P( sqrt(n)*Dn <= 1.24 ) = 0.9058 > 0.0500 → H0 не отвергается
#3: P( sqrt(n)*Dn <= 0.89 ) = 0.5871 > 0.0500 → H0 не отвергается
#4: P( sqrt(n)*Dn <= 0.56 ) = 0.0883 > 0.0500 → H0 не отвергается
#5: P( sqrt(n)*Dn <= 1.04 ) = 0.7665 > 0.0500 → H0 не отвергается
--------------------- Объем выборки: 100000 ---------------------
#1: P( sqrt(n)*Dn <= 0.61 ) = 0.1444 > 0.0500 → H0 не отвергается
#2: P( sqrt(n)*Dn <= 0.97 ) = 0.6983 > 0.0500 → H0 не отвергается
#3: P( sqrt(n)*Dn <= 0.80 ) = 0.4526 > 0.0500 → H0 не отвергается
#4: P( sqrt(n)*Dn <= 0.74 ) = 0.3614 > 0.0500 → H0 не отвергается
#5: P( sqrt(n)*Dn <= 1.27 ) = 0.9209 > 0.0500 → H0 не отвергается
```

Как мы видим, чем больше уровень значимости, тем больше вероятность, что $\sqrt{n}D_n(X)$ попадет в критическую область, и, соответственно, тем боль вероятность, что истинная гипотеза будет отвергнута.

Заранее ложная гипотеза

Пусть гипотеза H_0 утверждает, что наблюдаемая случайная величина имеет распределение $N(0, 1.1)$, что достаточно близко к истинному распределению $N(0, 1)$.

Размерность уже посчитанных массивов `Norm_ECDFs` для выборок из $N(0, 1)$ будет отличаться от размерности `CDF` для $N(0, 1.1)$, так как доверительные интервалы у них отличаются. Так что определим новый интервал, в который точно попадут оба:

$$(\min(\mu_1, \mu_2) - 3\max(\sigma_1, \sigma_2), \max(\mu_1, \mu_2) + 3\max(\sigma_1, \sigma_2))$$

In [419]:
```
false_eta = Norm(0, 1.1)
false_norm_data_x = np.linspace(
    min(eta.mu, false_eta.mu) - 3 * max(eta.sigma, false_eta.sigma),
    max(eta.mu, false_eta.mu) + 3 * max(eta.sigma, false_eta.sigma),
    1000,
)
```

И вычислим CDF для $N(10, 5)$

In [420]:
```
false_norm_cdf = np.apply_along_axis(
    sp.stats.norm(loc=false_eta.mu, scale=false_eta.sigma).cdf,
    0, false_norm_data_x)
```

In [423]:
```
for sign_level in sign_levels:
    print(f'\n================== Уровень значимости: {sign_level:4} ===================')
    for q in test_volumes:
        print(f'--------------------- Объем выборки: {q:6} ---------------------')
        for i, sample in enumerate(samples_norm[q]):
            ecdf = ECDF(sample)(false_norm_data_x)
            D = Dn(false_norm_cdf, ecdf)
            Kn = np.sqrt(q)*D
            prob_deny = sp.stats.kstwobign.cdf(Kn)
            deny = (prob_deny >= 1 - sign_level)
            print(f'#{i+1}: P( sqrt(n)*Dn <= {Kn:3.2f} ) = {prob_deny:.4f} {"≥" if deny else "<"} ' \
            f'{1 - sign_level:.4f} → H0 {"не" if not deny else "  "} отвергается')
```

```
================== Уровень значимости:  0.1 ===================
--------------------- Объем выборки:   1000 ---------------------
#1: P( sqrt(n)*Dn <= 1.17 ) = 0.8728 < 0.9000 → H0 не отвергается
#2: P( sqrt(n)*Dn <= 1.33 ) = 0.9405 ≥ 0.9000 → H0    отвергается
#3: P( sqrt(n)*Dn <= 1.25 ) = 0.9105 ≥ 0.9000 → H0    отвергается
#4: P( sqrt(n)*Dn <= 1.11 ) = 0.8299 < 0.9000 → H0 не отвергается
#5: P( sqrt(n)*Dn <= 1.75 ) = 0.9956 ≥ 0.9000 → H0    отвергается
--------------------- Объем выборки: 100000 ---------------------
#1: P( sqrt(n)*Dn <= 7.72 ) = 1.0000 ≥ 0.9000 → H0    отвергается
#2: P( sqrt(n)*Dn <= 7.99 ) = 1.0000 ≥ 0.9000 → H0    отвергается
#3: P( sqrt(n)*Dn <= 7.85 ) = 1.0000 ≥ 0.9000 → H0    отвергается
#4: P( sqrt(n)*Dn <= 7.20 ) = 1.0000 ≥ 0.9000 → H0    отвергается
#5: P( sqrt(n)*Dn <= 7.54 ) = 1.0000 ≥ 0.9000 → H0    отвергается

================== Уровень значимости: 0.05 ===================
--------------------- Объем выборки:   1000 ---------------------
#1: P( sqrt(n)*Dn <= 1.17 ) = 0.8728 < 0.9500 → H0 не отвергается
#2: P( sqrt(n)*Dn <= 1.33 ) = 0.9405 < 0.9500 → H0 не отвергается
#3: P( sqrt(n)*Dn <= 1.25 ) = 0.9105 < 0.9500 → H0 не отвергается
#4: P( sqrt(n)*Dn <= 1.11 ) = 0.8299 < 0.9500 → H0 не отвергается
#5: P( sqrt(n)*Dn <= 1.75 ) = 0.9956 ≥ 0.9500 → H0    отвергается
--------------------- Объем выборки: 100000 ---------------------
#1: P( sqrt(n)*Dn <= 7.72 ) = 1.0000 ≥ 0.9500 → H0    отвергается
#2: P( sqrt(n)*Dn <= 7.99 ) = 1.0000 ≥ 0.9500 → H0    отвергается
#3: P( sqrt(n)*Dn <= 7.85 ) = 1.0000 ≥ 0.9500 → H0    отвергается
#4: P( sqrt(n)*Dn <= 7.20 ) = 1.0000 ≥ 0.9500 → H0    отвергается
#5: P( sqrt(n)*Dn <= 7.54 ) = 1.0000 ≥ 0.9500 → H0    отвергается
```

Действительно, чем больше уровень значимости, тем больше вероятность $\mathbb{P}\left(\overline{H}_1 \middle| H_0\right)$, что ложная гипотеза будет отвергнута. Однако, с ростом объема выборки эта вероятность приближается к единице и в случае с малым уровнем значимости.

Критерий согласия Колмогорома для сложной гипотезы

TODO

Критерий согласия χ^2 для простой гипотезы

TODO

Критерий согласия χ^2 для сложной гипотезы

TODO

Критерий однородности Смирнова

Сам критерий формулируется следующим образом:

> Пусть в экперименте наблюдается *дискретная* случайная величина ξ, принимающая конечное число N различных значений $\{j_i\}_{i=1}^N$ с вероятностями $\{\mathbb{P}\,(\,\xi = j_i\,) = p_i\}_{i=1}^N$. Естесственно, должно выполняться условие нормировки $\sum_{i=1}^N p_i = 1$. Если произвеено n испытаний над ξ, т.е. имеется выборка $\overrightarrow{\xi} = (\xi_1, \dots, \xi_n)$, то пусть
> $$\nu_i = \sum_{i=1}^n I(\xi_i = j_i) \qquad \forall i \in \overline{1, N}$$
> - соответствующие частоты исходов. Конечно же, сумма этих частот должна быть равна количеству испытаний: $\sum_{i=1}^N \nu_i = n$. Тогда вектор $\overrightarrow{\nu} = (\nu_1, \dots, \nu_N)$ имеет полиномиальное распределение $M(n; p_1, \dots, p_N)$. Итак, пусть по наблюдению вектора частот $\overrightarrow{\nu}$ требуется проверить простую гипотезу $H_0 : \overrightarrow{p} = \overrightarrow{p^\circ}$

TODO

In [ ]: