In [ ]:
```
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.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 выбранных 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 $1^{\text{й}}$ начальный момент. Для начала, найдем $k^{\text{й}}$ начальный момент для ξ (это понадобится для дальнейших выводов):

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

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 $N - m = (N-1) - (m -$

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

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,

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

Дисперсия

По определению дисперсии,

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

Выведем $2^{\text{й}}$ начальный момент:

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

Подставим:

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

Таким образом,

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

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 одинаков длины, чтобы облегчить восприятие получившегося результата, однако это необязательно. Далее подсчитывается количество вхождений элемент 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 предоставлять метод `p(k)`, возвращающий вероятность принятия случайной величиной значения `k` при данных параметрах:

In [2]:
```
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 [3]:
```
xi = HG(80, 15, 40)
```

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

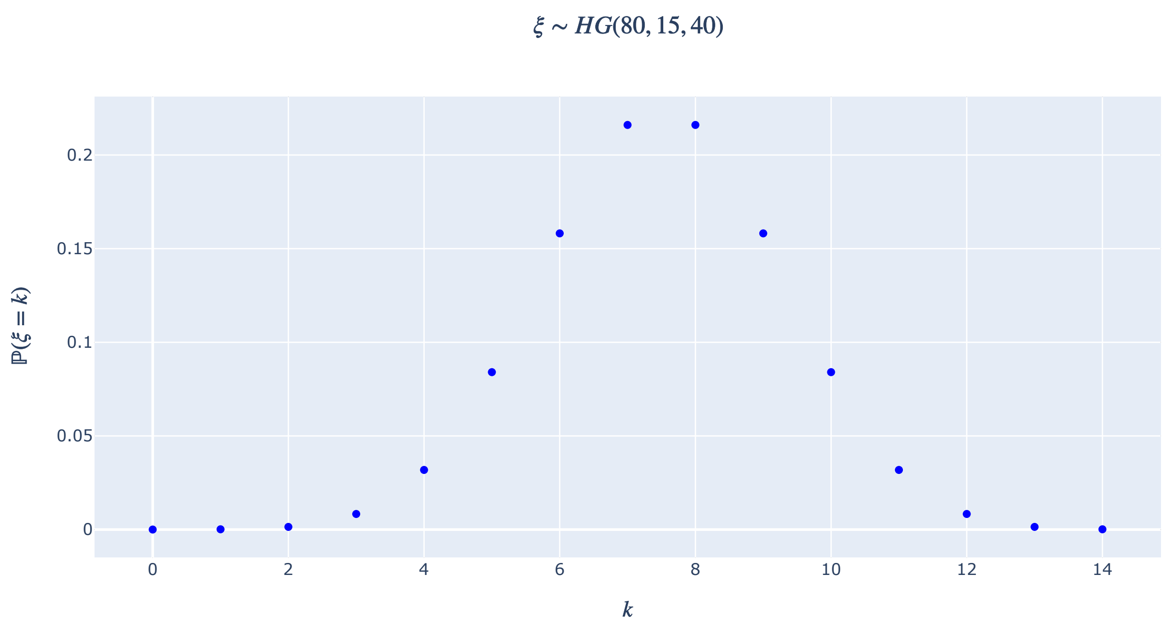
In [4]:
```
hg_data_x = np.arange(*xi.domain)
```

И, наконец, построим гистограмму и выведем ее:

In [5]:
```python
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 событий $\{\xi = i\}$ $\forall i \in \overline{0, k-1}$ 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:

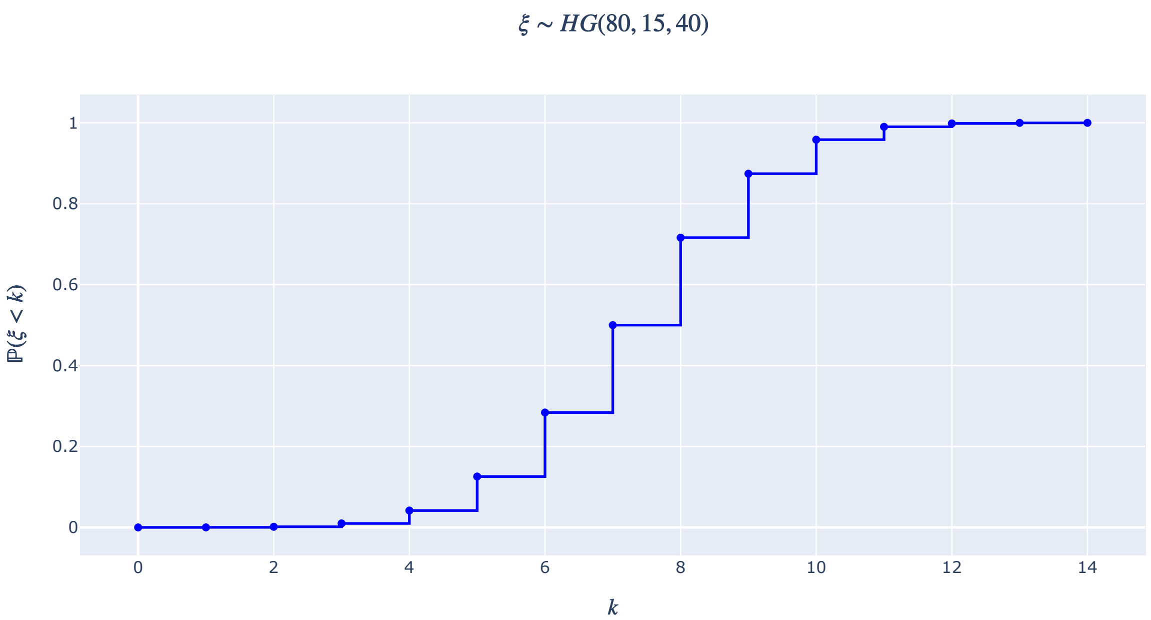
$$F_\xi(k) = \sum_{i=0}^{k-1} \mathbb{P}\,(\,\xi = i\,) = \sum_{i=0}^{k-1} \frac{\binom{m}{i}\binom{N-m}{n-i}}{\binom{N}{n}}$$

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

In [6]:
```
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:

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

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)](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

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

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)](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 [7]:
```
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 [8]:
```
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:

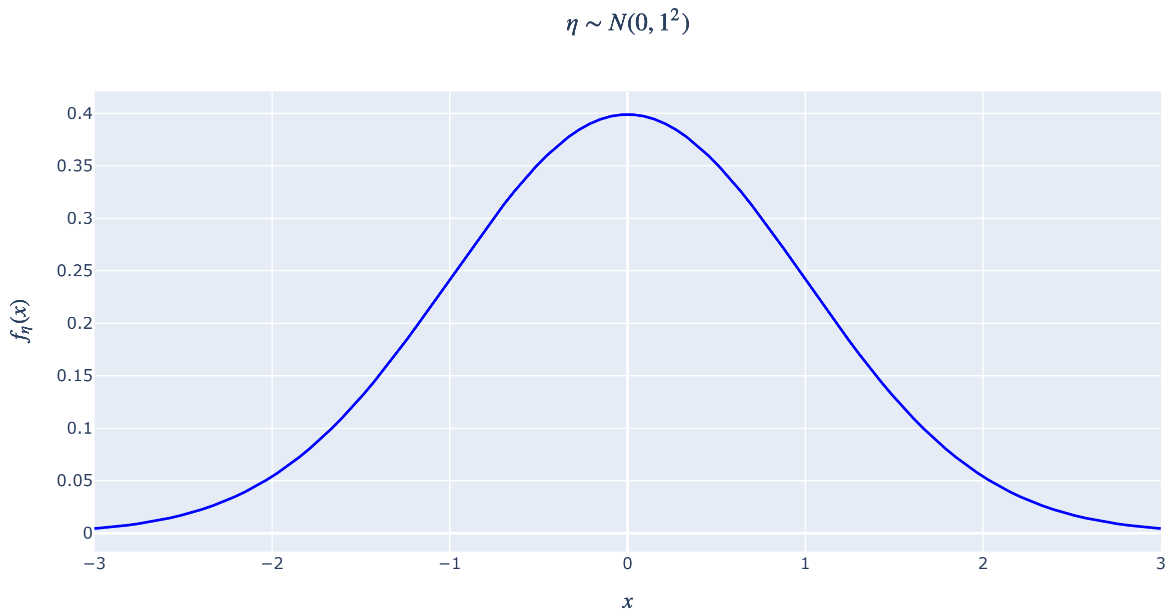
In [9]:
```
norm_data_x = np.linspace(-3*eta.sigma, 3*eta.sigma, 100)
```

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

In [10]:
```
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 $F_\eta(x) = \mathbb{P}\,(\,\eta < x\,)$. Для непрерывной случайной она определяется как интеграл от функции плотности вероятности:

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

. Подставим и получим:

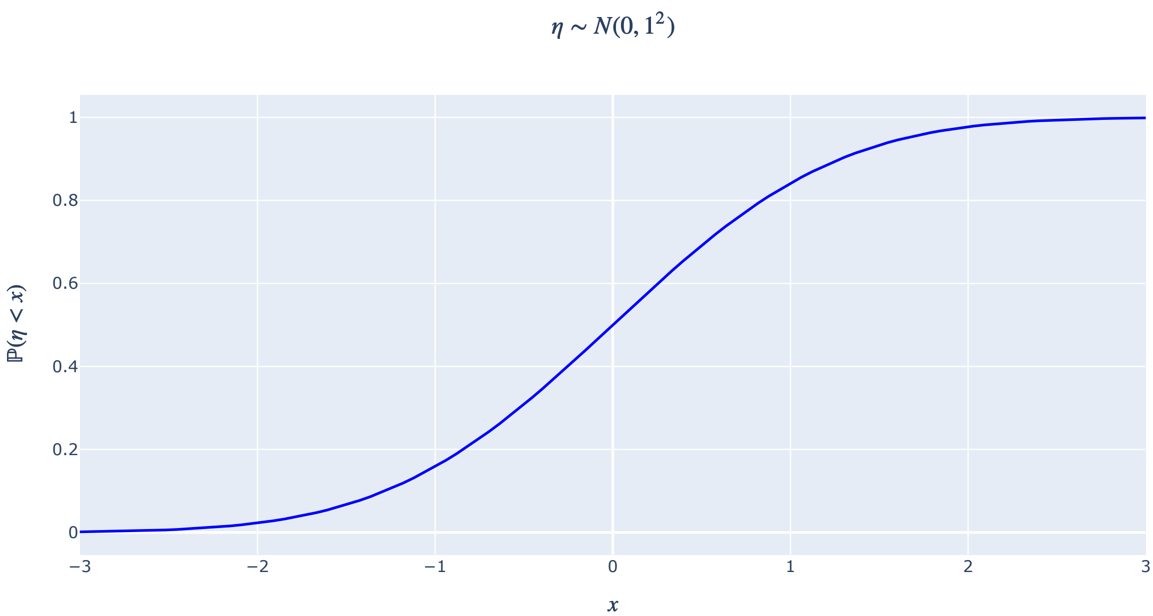
$$F_\xi(k) = \int_{-\infty}^{x} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}\,dx = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{(x-\mu)^2}{2\sigma^2}}\,dx$$

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

In [11]:
```
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 которых являются помеченными. Представим, что в нашем распоряжении имеется корзина, наполненная шарами двух цветов: черные и белые. Причём всего в 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 описыв вероятность 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 заполнить оставшиеся $n-k$ *слотов* непомеченными объектами (черными шарами). Таким образом, вероятность того, что среди n вытащенных объектов окажется 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/N:
$$\{\xi \sim HG(N, m, 1)\} \implies \left\{\xi \sim B\left(\frac{m}{N}\right)\right\}$$
Действительно, при размере выборки равным единице, случайная величина, имеющая гипергеометрическое распределение, может принимат только два значения $k \in \{0, 1\}$. То есть, нам либо попадется помеченный элемент, либо нет. Тогда эти вероятности описывает распределени 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 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)](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)](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)](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)](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)](http://zyurvas.narod.ru/knigi/Vadzinski_Ver_raspr.pdf)

Объявим константы:

In [12]:
```
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 [13]:
```
@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 функц `numpy.random.rand` (https://docs.scipy.org/doc/numpy/reference/random/index.html), которая возвращает действительное число в интервале $[0, 1$ проверяем, меньше ли значение, чем `m/N`. Если это значение меньше, это означает, что мы выбрали один из помеченных элементов. В этом случ мы 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 единицу.

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), который при первом вызове функции скомпилирует нашу функцию в машинный код, и, при последующих вызов 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 [14]:
```
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 [15]:
```
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` выборс соответствующего объема из распределения $HG(80, 15, 20)$.
Выведем выборки для `q = 5` и `q = 10`:

In [16]:
```
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: [6 7 5 7 8]
#2: [ 8  7 11  8  7]
#3: [7 9 4 6 7]
#4: [ 8 10  8  6  3]
#5: [9 7 7 7 7]
====== q = 10 ======:
#1: [ 8  9 11  9  9  8  7  9  6 10]
#2: [ 7  5  6 10  8  7  8  7  7  7]
#3: [8 8 5 9 5 9 8 7 5 8]
#4: [9 8 7 8 7 7 7 6 6 8]
#5: [ 8  7  4  7  6  8  9 10 10  4]
```

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):

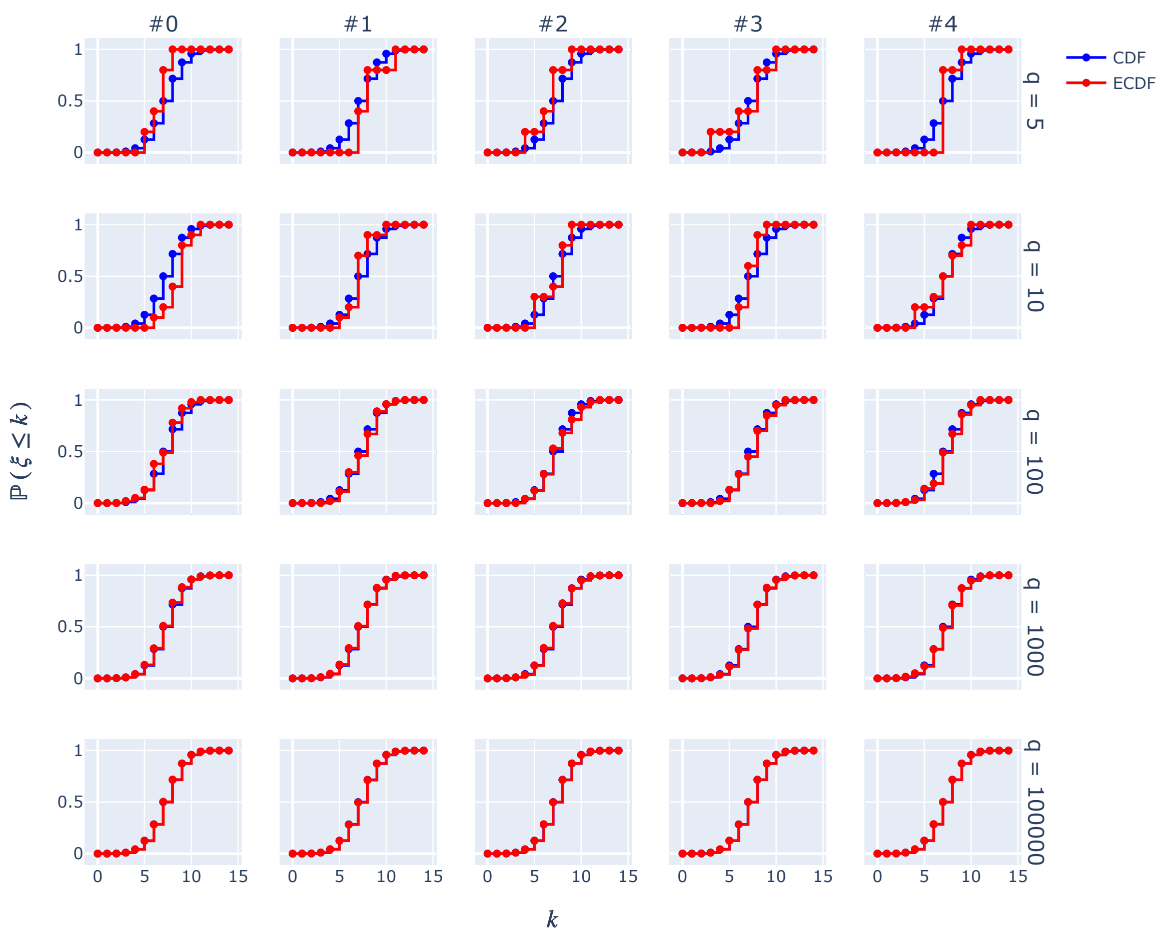
In [17]:
```
HG_ECDFs = {}
for q in samples_hg:
    HG_ECDFs[q] = np.ndarray(shape=(samples_hg[q].shape[0], hg_data_x.shape[0]), dtype=np.float)
    for i, sample in enumerate(samples_hg[q]):
        HG_ECDFs[q][i] = ECDF(sample)(hg_data_x)
```

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

```
In [18]: 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 [19]: 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 = 0.4000
i = 0; j = 2; max delta = 0.2000
i = 0; j = 3; max delta = 0.4000
i = 0; j = 4; max delta = 0.4000
i = 1; j = 2; max delta = 0.4000
i = 1; j = 3; max delta = 0.4000
i = 1; j = 4; max delta = 0.4000
i = 2; j = 3; max delta = 0.4000
i = 2; j = 4; max delta = 0.4000
i = 3; j = 4; max delta = 0.4000
====== q = 10 ======:
i = 0; j = 1; max delta = 0.5000
i = 0; j = 2; max delta = 0.4000
i = 0; j = 3; max delta = 0.5000
i = 0; j = 4; max delta = 0.3000
i = 1; j = 2; max delta = 0.3000
i = 1; j = 3; max delta = 0.1000
i = 1; j = 4; max delta = 0.2000
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 = 0.1100
i = 0; j = 2; max delta = 0.1100
i = 0; j = 3; max delta = 0.1000
i = 0; j = 4; max delta = 0.1900
i = 1; j = 2; max delta = 0.0800
i = 1; j = 3; max delta = 0.0400
i = 1; j = 4; max delta = 0.1100
i = 2; j = 3; max delta = 0.0800
i = 2; j = 4; max delta = 0.0900
i = 3; j = 4; max delta = 0.0900
====== q = 1000 ======:
i = 0; j = 1; max delta = 0.0210
i = 0; j = 2; max delta = 0.0120
i = 0; j = 3; max delta = 0.0260
i = 0; j = 4; max delta = 0.0280
i = 1; j = 2; max delta = 0.0150
i = 1; j = 3; max delta = 0.0250
i = 1; j = 4; max delta = 0.0190
i = 2; j = 3; max delta = 0.0260
i = 2; j = 4; max delta = 0.0220
i = 3; j = 4; max delta = 0.0140
====== q = 100000 ======:
i = 0; j = 1; max delta = 0.0065
i = 0; j = 2; max delta = 0.0047
i = 0; j = 3; max delta = 0.0020
i = 0; j = 4; max delta = 0.0018
i = 1; j = 2; max delta = 0.0025
i = 1; j = 3; max delta = 0.0044
i = 1; j = 4; max delta = 0.0047
i = 2; j = 3; max delta = 0.0027
i = 2; j = 4; max delta = 0.0030
i = 3; j = 4; max delta = 0.0026
```

Построение вариационного ряда выборки

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

```
In [20]: 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 [21]:
```python
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: [5 6 7 7 8]
#2: [ 7  7  8  8 11]
#3: [4 6 7 7 9]
#4: [ 3  6  8  8 10]
#5: [7 7 7 7 9]
====== q = 10 ======:
#1: [ 6  7  8  8  9  9  9  9 10 11]
#2: [ 5  6  7  7  7  7  7  8  8 10]
#3: [5 5 5 7 8 8 8 8 9 9]
#4: [6 6 7 7 7 7 8 8 8 9]
#5: [ 4  4  6  7  7  8  8  9 10 10]
```

Квантили

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

$$\mathbb{P}(\,\xi \le x_\alpha\,) \ge \alpha$$
$$\mathbb{P}(\,\xi \ge x_\alpha\,) \ge 1 - \alpha$$

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

In [22]:
```python
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 [23]:
```python
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 =  5, real delta = +0
#1; quantile =  7, real delta = +2
#2; quantile =  4, real delta = -1
#3; quantile =  3, real delta = -2
#4; quantile =  7, real delta = +2
------ a = 0.5 ------
#0; quantile =  7, real delta = +0
#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 =  7, real delta = -1
#1; quantile =  8, real delta = +0
#2; quantile =  7, real delta = -1
#3; quantile =  8, real delta = +0
#4; quantile =  7, real delta = -1
======= q = 10 =======:
------ a = 0.1 ------
#0; quantile =  7, real delta = +2
#1; quantile =  6, real delta = +1
#2; quantile =  5, real delta = +0
#3; quantile =  6, real delta = +1
#4; quantile =  4, real delta = -1
------ a = 0.5 ------
#0; quantile =  9, real delta = +2
#1; quantile =  7, real delta = +0
#2; quantile =  8, real delta = +1
#3; quantile =  7, real delta = +0
#4; quantile =  8, real delta = +1
------ a = 0.7 ------
#0; quantile =  9, real delta = +1
#1; quantile =  8, real delta = +0
#2; quantile =  8, real delta = +0
```

```
#3; quantile =  8, real delta = +0
#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 =  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 =  8, real delta = +1
------ a = 0.7 ------
#0; quantile =  8, real delta = +0
#1; quantile =  9, real delta = +1
#2; quantile =  9, real delta = +1
#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 =  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 =  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 =  7, real delta = +0
#1; quantile =  8, real delta = +1
#2; quantile =  8, real delta = +1
#3; quantile =  7, real delta = +0
#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...

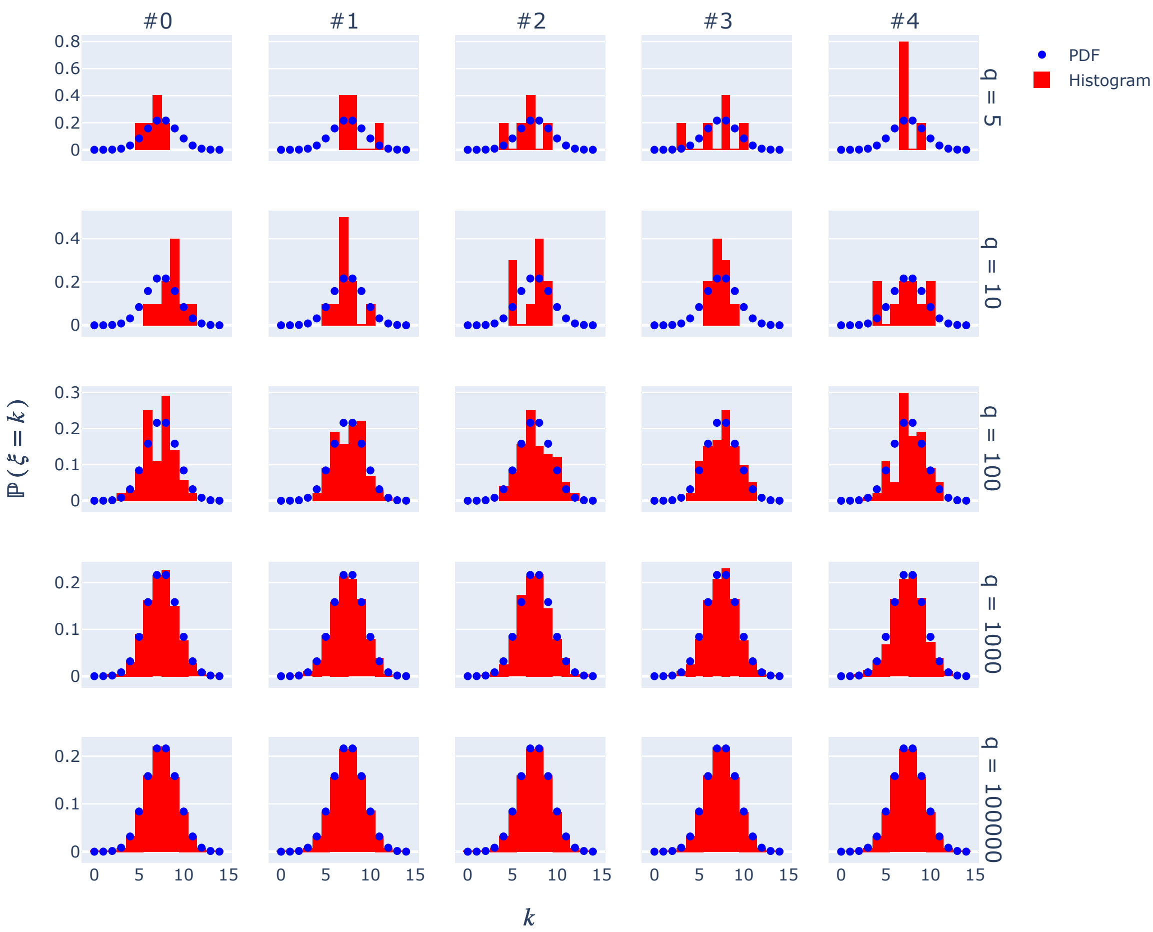
Гистограмма и полигон частот

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

In [24]:
```
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 приближаются 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 сразу 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 равномерног 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 [25]:
```
@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 [26]:
```
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 [27]:
```
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 [28]:
```
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: [ 0.69198 -0.69316  1.22509  1.27796  2.49675]
#2: [-0.40527 -0.47611 -1.30229  0.25197  0.53894]
#3: [ 1.15419  0.07907  0.04235  1.4413  -0.25576]
#4: [-1.43322 -0.00307 -0.06868  1.66663 -0.21912]
#5: [ 1.3651  -0.08005  1.66889 -1.81044 -0.25549]
====== q = 10 ======:
#1: [-0.16506  1.92954 -0.03464  0.91636  0.33117 -0.59503  1.83182  1.2187  -0.62711 -1.58499]
#2: [ 0.51679  0.73893 -0.37844  0.51192  1.00576 -0.1105   0.16059  0.09324  0.34661  0.61831]
#3: [ 0.01647  0.50394  1.19981 -1.29306  0.48681 -0.78282  0.71671  0.5499   1.19453  0.09099]
#4: [ 1.43406 -1.24942  0.21205  0.58018  0.36254 -1.65598  0.71907 -0.57109 -0.53523 -0.30954]
#5: [-0.84441  0.73921 -0.25078 -0.28855  0.58544  1.05388 -0.44175 -1.64611  0.93266  0.40869]
```

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):

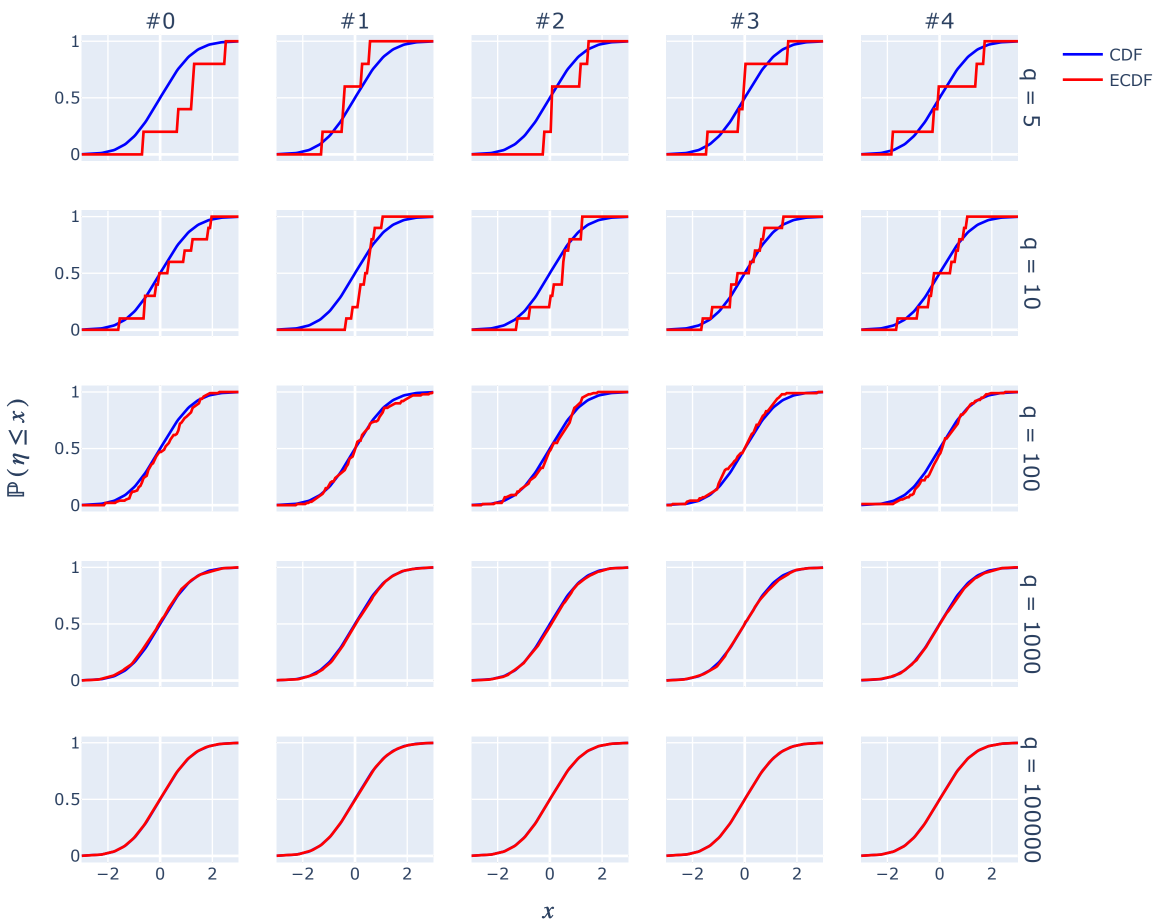
In [29]:
```
Norm_ECDFs = {}
for q in samples_norm:
    Norm_ECDFs[q] = np.ndarray(shape=(samples_norm[q].shape[0], norm_data_x.shape[0]), dtype=np.float)
    for i, sample in enumerate(samples_norm[q]):
        Norm_ECDFs[q][i] = ECDF(sample)(norm_data_x)
```

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

In [30]:
```python
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 [31]:
```
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 = 0.8000
i = 0; j = 2; max delta = 0.4000
i = 0; j = 3; max delta = 0.6000
i = 0; j = 4; max delta = 0.4000
i = 1; j = 2; max delta = 0.6000
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 = 0.4000
i = 3; j = 4; max delta = 0.2000
====== q = 10 ======:
i = 0; j = 1; max delta = 0.3000
i = 0; j = 2; max delta = 0.3000
i = 0; j = 3; max delta = 0.3000
i = 0; j = 4; max delta = 0.3000
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 = 0.3000
i = 2; j = 4; max delta = 0.3000
i = 3; j = 4; max delta = 0.2000
====== q = 100 ======:
i = 0; j = 1; max delta = 0.1000
i = 0; j = 2; max delta = 0.1000
i = 0; j = 3; max delta = 0.1400
i = 0; j = 4; max delta = 0.0800
i = 1; j = 2; max delta = 0.1000
i = 1; j = 3; max delta = 0.1000
i = 1; j = 4; max delta = 0.0800
i = 2; j = 3; max delta = 0.1000
i = 2; j = 4; max delta = 0.0600
i = 3; j = 4; max delta = 0.1200
====== q = 1000 ======:
i = 0; j = 1; max delta = 0.0430
i = 0; j = 2; max delta = 0.0440
i = 0; j = 3; max delta = 0.0340
i = 0; j = 4; max delta = 0.0400
i = 1; j = 2; max delta = 0.0180
i = 1; j = 3; max delta = 0.0230
i = 1; j = 4; max delta = 0.0200
i = 2; j = 3; max delta = 0.0250
i = 2; j = 4; max delta = 0.0180
i = 3; j = 4; max delta = 0.0200
====== q = 100000 ======:
i = 0; j = 1; max delta = 0.0058
i = 0; j = 2; max delta = 0.0070
i = 0; j = 3; max delta = 0.0055
i = 0; j = 4; max delta = 0.0060
i = 1; j = 2; max delta = 0.0047
i = 1; j = 3; max delta = 0.0025
i = 1; j = 4; max delta = 0.0031
i = 2; j = 3; max delta = 0.0029
i = 2; j = 4; max delta = 0.0035
i = 3; j = 4; max delta = 0.0021
```

Построение вариационного ряда выборки

Построение вариационного ряда выборки

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

In [32]:
```
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 [33]:
```
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: [-0.69316  0.69198  1.22509  1.27796  2.49675]
#2: [-1.30229 -0.47611 -0.40527  0.25197  0.53894]
#3: [-0.25576  0.04235  0.07907  1.15419  1.4413 ]
#4: [-1.43322 -0.21912 -0.06868 -0.00307  1.66663]
#5: [-1.81044 -0.25549 -0.08005  1.3651   1.66889]
====== q = 10 ======:
#1: [-1.58499 -0.62711 -0.59503 -0.16506 -0.03464  0.33117  0.91636  1.2187   1.83182  1.92954]
#2: [-0.37844 -0.1105   0.09324  0.16059  0.34661  0.51192  0.51679  0.61831  0.73893  1.00576]
#3: [-1.29306 -0.78282  0.01647  0.09099  0.48681  0.50394  0.5499   0.71671  1.19453  1.19981]
#4: [-1.65598 -1.24942 -0.57109 -0.53523 -0.30954  0.21205  0.36254  0.58018  0.71907  1.43406]
#5: [-1.64611 -0.84441 -0.44175 -0.28855 -0.25078  0.40869  0.58544  0.73921  0.93266  1.05388]
```

Квантили

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

In [34]:
```
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 [35]:
```
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 = -0.693, real delta = +0.549
#1; quantile = -1.302, real delta = -0.060
#2; quantile = -0.256, real delta = +0.987
#3; quantile = -1.433, real delta = -0.191
#4; quantile = -1.810, real delta = -0.568
------ a = 0.5 ------
#0; quantile = +1.225, real delta = +1.195
#1; quantile = -0.405, real delta = -0.436
#2; quantile = +0.079, real delta = +0.049
#3; quantile = -0.069, real delta = -0.099
#4; quantile = -0.080, real delta = -0.110
------ a = 0.7 ------
#0; quantile = +1.278, real delta = +0.702
#1; quantile = +0.252, real delta = -0.324
#2; quantile = +1.154, real delta = +0.578
#3; quantile = -0.003, real delta = -0.579
#4; quantile = +1.365, real delta = +0.789
======= q = 10 =======:
------ a = 0.1 ------
#0; quantile = -0.627, real delta = +0.615
#1; quantile = -0.111, real delta = +1.132
#2; quantile = -0.783, real delta = +0.460
#3; quantile = -1.249, real delta = -0.007
#4; quantile = -0.844, real delta = +0.398
------ a = 0.5 ------
#0; quantile = +0.331, real delta = +0.301
#1; quantile = +0.512, real delta = +0.482
#2; quantile = +0.504, real delta = +0.474
#3; quantile = +0.212, real delta = +0.182
#4; quantile = +0.409, real delta = +0.378
------ a = 0.7 ------
#0; quantile = +1.219, real delta = +0.643
#1; quantile = +0.618, real delta = +0.043
#2; quantile = +0.717, real delta = +0.141
#3; quantile = +0.580, real delta = +0.004
#4; quantile = +0.739, real delta = +0.163
======= q = 100 =======:
------ a = 0.1 ------
#0; quantile = -1.091, real delta = +0.152
#1; quantile = -1.239, real delta = +0.004
```

```
#2; quantile = −1.267, real delta = −0.025
#3; quantile = −1.325, real delta = −0.082
#4; quantile = −0.987, real delta = +0.256
------ a = 0.5 ------
#0; quantile = +0.169, real delta = +0.139
#1; quantile = +0.026, real delta = −0.005
#2; quantile = +0.071, real delta = +0.040
#3; quantile = +0.006, real delta = −0.024
#4; quantile = +0.093, real delta = +0.063
------ a = 0.7 ------
#0; quantile = +0.743, real delta = +0.168
#1; quantile = +0.562, real delta = −0.014
#2; quantile = +0.702, real delta = +0.127
#3; quantile = +0.453, real delta = −0.123
#4; quantile = +0.639, real delta = +0.063
======= q = 1000 =======:
------ a = 0.1 ------
#0; quantile = −1.375, real delta = −0.133
#1; quantile = −1.228, real delta = +0.015
#2; quantile = −1.312, real delta = −0.069
#3; quantile = −1.231, real delta = +0.011
#4; quantile = −1.294, real delta = −0.052
------ a = 0.5 ------
#0; quantile = −0.044, real delta = −0.074
#1; quantile = +0.037, real delta = +0.006
#2; quantile = +0.042, real delta = +0.011
#3; quantile = −0.002, real delta = −0.033
#4; quantile = +0.004, real delta = −0.026
------ a = 0.7 ------
#0; quantile = +0.496, real delta = −0.080
#1; quantile = +0.582, real delta = +0.006
#2; quantile = +0.610, real delta = +0.034
#3; quantile = +0.549, real delta = −0.027
#4; quantile = +0.589, real delta = +0.013
======= q = 100000 =======:
------ a = 0.1 ------
#0; quantile = −1.288, real delta = −0.045
#1; quantile = −1.280, real delta = −0.037
#2; quantile = −1.276, real delta = −0.034
#3; quantile = −1.276, real delta = −0.033
#4; quantile = −1.278, real delta = −0.035
------ a = 0.5 ------
#0; quantile = −0.010, real delta = −0.040
#1; quantile = +0.000, real delta = −0.030
#2; quantile = +0.002, real delta = −0.028
#3; quantile = +0.003, real delta = −0.028
#4; quantile = +0.002, real delta = −0.029
------ a = 0.7 ------
#0; quantile = +0.521, real delta = −0.055
#1; quantile = +0.527, real delta = −0.048
#2; quantile = +0.526, real delta = −0.050
#3; quantile = +0.526, real delta = −0.050
#4; quantile = +0.533, real delta = −0.042
```

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

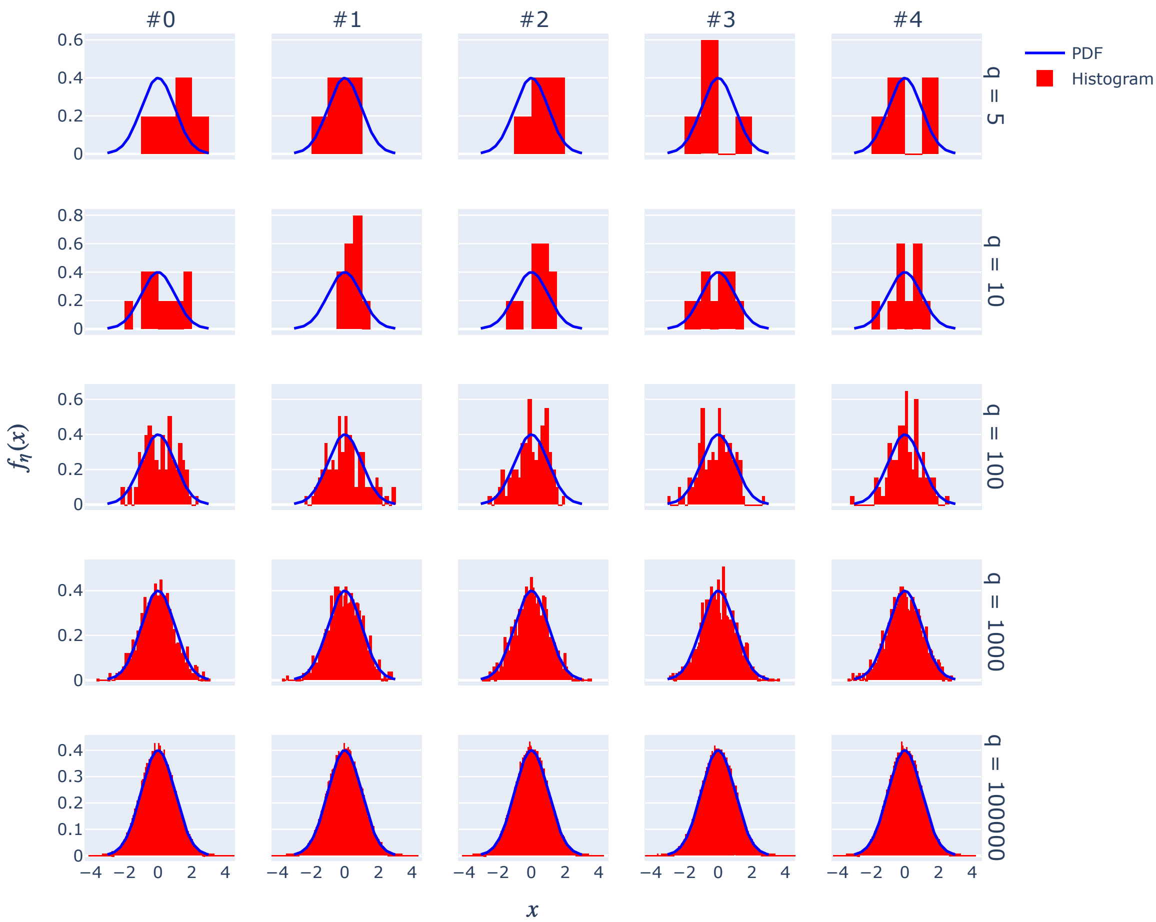
Гистограмма и полигон частот

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 [36]:
```
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 формулам:

$$\overline{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$$

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

$$S^2 = \frac{n}{n-1}S_n^2$$

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

In [37]:

```python
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:                6.6000, real delta: -0.9000
   variance:            1.0400, real delta: -1.4284
   unbiased variance:  1.3000, real delta: -1.7854

#1 mean:                8.2000, real delta: +0.7000
   variance:            2.1600, real delta: -0.3084
   unbiased variance:  2.7000, real delta: -0.3854

#2 mean:                6.6000, real delta: -0.9000
   variance:            2.6400, real delta: +0.1716
   unbiased variance:  3.3000, real delta: +0.2146

#3 mean:                7.0000, real delta: -0.5000
   variance:            5.6000, real delta: +3.1316
   unbiased variance:  7.0000, real delta: +3.9146

#4 mean:                7.4000, real delta: -0.1000
   variance:            0.6400, real delta: -1.8284
   unbiased variance:  0.8000, real delta: -2.2854

======= q = 10 =======:
#0 mean:                8.6000, real delta: +1.1000
   variance:            1.8400, real delta: -0.9369
   unbiased variance:  2.0444, real delta: -1.0410

#1 mean:                7.2000, real delta: -0.3000
   variance:            1.5600, real delta: -1.2169
   unbiased variance:  1.7333, real delta: -1.3521

#2 mean:                7.2000, real delta: -0.3000
   variance:            2.3600, real delta: -0.4169
   unbiased variance:  2.6222, real delta: -0.4632

#3 mean:                7.3000, real delta: -0.2000
   variance:            0.8100, real delta: -1.9669
   unbiased variance:  0.9000, real delta: -2.1854

#4 mean:                7.3000, real delta: -0.2000
   variance:            4.2100, real delta: +1.4331
   unbiased variance:  4.6778, real delta: +1.5923

======= q = 100 =======:
#0 mean:                7.2500, real delta: -0.2500
   variance:            2.8075, real delta: -0.2471
   unbiased variance:  2.8359, real delta: -0.2496

#1 mean:                7.6000, real delta: +0.1000
   variance:            2.8200, real delta: -0.2346
   unbiased variance:  2.8485, real delta: -0.2370

#2 mean:                7.6300, real delta: +0.1300
   variance:            3.4931, real delta: +0.4385
   unbiased variance:  3.5284, real delta: +0.4429

#3 mean:                7.6200, real delta: +0.1200
   variance:            2.9356, real delta: -0.1190
   unbiased variance:  2.9653, real delta: -0.1202
```

```
#4 mean:               7.6600, real delta: +0.1600
   variance:           2.9444, real delta: −0.1102
   unbiased variance:  2.9741, real delta: −0.1113

======= q = 1000 =======:
#0 mean:               7.4490, real delta: −0.0510
   variance:           3.0494, real delta: −0.0330
   unbiased variance:  3.0525, real delta: −0.0330

#1 mean:               7.4630, real delta: −0.0370
   variance:           3.1126, real delta: +0.0303
   unbiased variance:  3.1157, real delta: +0.0303

#2 mean:               7.4850, real delta: −0.0150
   variance:           3.0878, real delta: +0.0054
   unbiased variance:  3.0909, real delta: +0.0054

#3 mean:               7.5470, real delta: +0.0470
   variance:           3.0578, real delta: −0.0246
   unbiased variance:  3.0609, real delta: −0.0246

#4 mean:               7.5250, real delta: +0.0250
   variance:           3.2334, real delta: +0.1510
   unbiased variance:  3.2366, real delta: +0.1512

======= q = 100000 =======:
#0 mean:               7.4969, real delta: −0.0031
   variance:           3.0698, real delta: −0.0156
   unbiased variance:  3.0699, real delta: −0.0156

#1 mean:               7.5075, real delta: +0.0075
   variance:           3.0994, real delta: +0.0140
   unbiased variance:  3.0994, real delta: +0.0140

#2 mean:               7.5039, real delta: +0.0039
   variance:           3.0807, real delta: −0.0047
   unbiased variance:  3.0807, real delta: −0.0047

#3 mean:               7.5007, real delta: +0.0007
   variance:           3.0858, real delta: +0.0004
   unbiased variance:  3.0858, real delta: +0.0004

#4 mean:               7.4967, real delta: −0.0033
   variance:           3.0924, real delta: +0.0070
   unbiased variance:  3.0925, real delta: +0.0070
```

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 величин

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

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 [38]:
```
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: 90.9091,      real delta:   +10.9091
#1 N_hat: 73.1707,      real delta:    -6.8293
#2 N_hat: 90.9091,      real delta:   +10.9091
#3 N_hat: 85.7143,      real delta:    +5.7143
#4 N_hat: 81.0811,      real delta:    +1.0811
======= q = 10 =======:
#0 N_hat: 69.7674,      real delta:   -10.2326
#1 N_hat: 83.3333,      real delta:    +3.3333
#2 N_hat: 83.3333,      real delta:    +3.3333
#3 N_hat: 82.1918,      real delta:    +2.1918
#4 N_hat: 82.1918,      real delta:    +2.1918
======= q = 100 =======:
#0 N_hat: 82.7586,      real delta:    +2.7586
#1 N_hat: 78.9474,      real delta:    -1.0526
#2 N_hat: 78.6370,      real delta:    -1.3630
#3 N_hat: 78.7402,      real delta:    -1.2598
#4 N_hat: 78.3290,      real delta:    -1.6710
======= q = 1000 =======:
#0 N_hat: 80.5477,      real delta:    +0.5477
#1 N_hat: 80.3966,      real delta:    +0.3966
#2 N_hat: 80.1603,      real delta:    +0.1603
#3 N_hat: 79.5018,      real delta:    -0.4982
#4 N_hat: 79.7342,      real delta:    -0.2658
======= q = 100000 =======:
#0 N_hat: 80.0330,      real delta:    +0.0330
#1 N_hat: 79.9202,      real delta:    -0.0798
#2 N_hat: 79.9580,      real delta:    -0.0420
#3 N_hat: 79.9926,      real delta:    -0.0074
#4 N_hat: 80.0353,      real delta:    +0.0353
```

Как можно заметить, при увеличении объема выборки, разность между значением оценки и оцениваемым параметром $\hat{N} - N$ стремится к нулю, ч 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 [39]:
```
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:                0.9997, real delta: +0.9997
   variance:            1.0660, real delta: +0.2660
   unbiased variance:  1.3325, real delta: +0.3325

#1 mean:               -0.2786, real delta: -0.2786
   variance:            0.4106, real delta: -0.3894
   unbiased variance:  0.5132, real delta: -0.4868

#2 mean:                0.4922, real delta: +0.4922
   variance:            0.4543, real delta: -0.3457
   unbiased variance:  0.5679, real delta: -0.4321

#3 mean:               -0.0115, real delta: -0.0115
   variance:            0.9768, real delta: +0.1768
   unbiased variance:  1.2210, real delta: +0.2210

#4 mean:                0.1776, real delta: +0.1776
   variance:            1.5681, real delta: +0.7681
   unbiased variance:  1.9601, real delta: +0.9601

======= q = 10 =======:
#0 mean:                0.3221, real delta: +0.3221
   variance:            1.1764, real delta: +0.2764
   unbiased variance:  1.3071, real delta: +0.3071

#1 mean:                0.3503, real delta: +0.3503
```

```
   variance:           0.1552, real delta: −0.7448
   unbiased variance:  0.1724, real delta: −0.8276

#2 mean:               0.2683, real delta: +0.2683
   variance:           0.5747, real delta: −0.3253
   unbiased variance:  0.6385, real delta: −0.3615

#3 mean:              −0.1013, real delta: −0.1013
   variance:           0.7996, real delta: −0.1004
   unbiased variance:  0.8884, real delta: −0.1116

#4 mean:               0.0248, real delta: +0.0248
   variance:           0.6795, real delta: −0.2205
   unbiased variance:  0.7550, real delta: −0.2450

======= q = 100 =======:
#0 mean:               0.1459, real delta: +0.1459
   variance:           0.9156, real delta: −0.0744
   unbiased variance:  0.9249, real delta: −0.0751

#1 mean:               0.0947, real delta: +0.0947
   variance:           1.1570, real delta: +0.1670
   unbiased variance:  1.1687, real delta: +0.1687

#2 mean:               0.0041, real delta: +0.0041
   variance:           0.8653, real delta: −0.1247
   unbiased variance:  0.8741, real delta: −0.1259

#3 mean:              −0.1218, real delta: −0.1218
   variance:           0.9629, real delta: −0.0271
   unbiased variance:  0.9726, real delta: −0.0274

#4 mean:               0.0898, real delta: +0.0898
   variance:           0.8604, real delta: −0.1296
   unbiased variance:  0.8691, real delta: −0.1309

======= q = 1000 =======:
#0 mean:              −0.0372, real delta: −0.0372
   variance:           1.0480, real delta: +0.0490
   unbiased variance:  1.0490, real delta: +0.0490

#1 mean:               0.0337, real delta: +0.0337
   variance:           0.9930, real delta: −0.0060
   unbiased variance:  0.9940, real delta: −0.0060

#2 mean:               0.0400, real delta: +0.0400
   variance:           1.0250, real delta: +0.0260
   unbiased variance:  1.0261, real delta: +0.0261

#3 mean:               0.0286, real delta: +0.0286
   variance:           1.0350, real delta: +0.0360
   unbiased variance:  1.0360, real delta: +0.0360

#4 mean:               0.0263, real delta: +0.0263
   variance:           1.0496, real delta: +0.0506
   unbiased variance:  1.0506, real delta: +0.0506

======= q = 100000 =======:
#0 mean:              −0.0081, real delta: −0.0081
   variance:           0.9996, real delta: −0.0004
   unbiased variance:  0.9996, real delta: −0.0004

#1 mean:               0.0007, real delta: +0.0007
   variance:           1.0027, real delta: +0.0028
   unbiased variance:  1.0028, real delta: +0.0028

#2 mean:               0.0025, real delta: +0.0025
   variance:           0.9898, real delta: −0.0102
   unbiased variance:  0.9898, real delta: −0.0102

#3 mean:               0.0012, real delta: +0.0012
   variance:           0.9951, real delta: −0.0049
   unbiased variance:  0.9951, real delta: −0.0049

#4 mean:               0.0040, real delta: +0.0040
   variance:           0.9987, real delta: −0.0013
   unbiased variance:  0.9987, real delta: −0.0013
```

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 [40]:
```python
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:  0.9997,     real delta:    +0.9997
#1 mu_hat: -0.2786,     real delta:    -0.2786
#2 mu_hat:  0.4922,     real delta:    +0.4922
#3 mu_hat: -0.0115,     real delta:    -0.0115
#4 mu_hat:  0.1776,     real delta:    +0.1776
======= q = 10 =======:
#0 mu_hat:  0.3221,     real delta:    +0.3221
#1 mu_hat:  0.3503,     real delta:    +0.3503
#2 mu_hat:  0.2683,     real delta:    +0.2683
#3 mu_hat: -0.1013,     real delta:    -0.1013
#4 mu_hat:  0.0248,     real delta:    +0.0248
======= q = 100 =======:
#0 mu_hat:  0.1459,     real delta:    +0.1459
#1 mu_hat:  0.0947,     real delta:    +0.0947
#2 mu_hat:  0.0041,     real delta:    +0.0041
#3 mu_hat: -0.1218,     real delta:    -0.1218
#4 mu_hat:  0.0898,     real delta:    +0.0898
======= q = 1000 =======:
#0 mu_hat: -0.0372,     real delta:    -0.0372
#1 mu_hat:  0.0337,     real delta:    +0.0337
#2 mu_hat:  0.0400,     real delta:    +0.0400
#3 mu_hat:  0.0286,     real delta:    +0.0286
#4 mu_hat:  0.0263,     real delta:    +0.0263
======= q = 100000 =======:
#0 mu_hat: -0.0081,     real delta:    -0.0081
#1 mu_hat:  0.0007,     real delta:    +0.0007
#2 mu_hat:  0.0025,     real delta:    +0.0025
#3 mu_hat:  0.0012,     real delta:    +0.0012
#4 mu_hat:  0.0040,     real delta:    +0.0040
```

Как можно заметить, при увеличении объема выборки, разность между значением оценки и оцениваемым параметром $\hat{\mu} - \mu$ стремится к нулю, чт 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 дисперсией 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 [41]:
```python
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:  1.0660,        real delta:    +0.0660
#1 sigma_2_hat:  0.4106,        real delta:    -0.5894
#2 sigma_2_hat:  0.4543,        real delta:    -0.5457
#3 sigma_2_hat:  0.9768,        real delta:    -0.0232
#4 sigma_2_hat:  1.5681,        real delta:    +0.5681
======= q = 10 =======:
#0 sigma_2_hat:  1.1764,        real delta:    +0.1764
#1 sigma_2_hat:  0.1552,        real delta:    -0.8448
#2 sigma_2_hat:  0.5747,        real delta:    -0.4253
#3 sigma_2_hat:  0.7996,        real delta:    -0.2004
#4 sigma_2_hat:  0.6795,        real delta:    -0.3205
======= q = 100 =======:
#0 sigma_2_hat:  0.9156,        real delta:    -0.0844
#1 sigma_2_hat:  1.1570,        real delta:    +0.1570
#2 sigma_2_hat:  0.8653,        real delta:    -0.1347
#3 sigma_2_hat:  0.9629,        real delta:    -0.0371
#4 sigma_2_hat:  0.8604,        real delta:    -0.1396
======= q = 1000 =======:
#0 sigma_2_hat:  1.0480,        real delta:    +0.0480
#1 sigma_2_hat:  0.9930,        real delta:    -0.0070
#2 sigma_2_hat:  1.0250,        real delta:    +0.0250
#3 sigma_2_hat:  1.0350,        real delta:    +0.0350
#4 sigma_2_hat:  1.0496,        real delta:    +0.0496
======= q = 100000 =======:
#0 sigma_2_hat:  0.9996,        real delta:    -0.0004
#1 sigma_2_hat:  1.0027,        real delta:    +0.0027
#2 sigma_2_hat:  0.9898,        real delta:    -0.0102
#3 sigma_2_hat:  0.9951,        real delta:    -0.0049
#4 sigma_2_hat:  0.9987,        real delta:    -0.0013
```

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 нашал оценка $\hat{\sigma^2}$ является состоятельной.