In [ ]:
```
!pip install \
    numpy \
    scipy \
    statsmodels \
    plotly \
    numba
```

In [1]:
```
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

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*:

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

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,

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

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:

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

Таким образом,

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

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.

    @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:

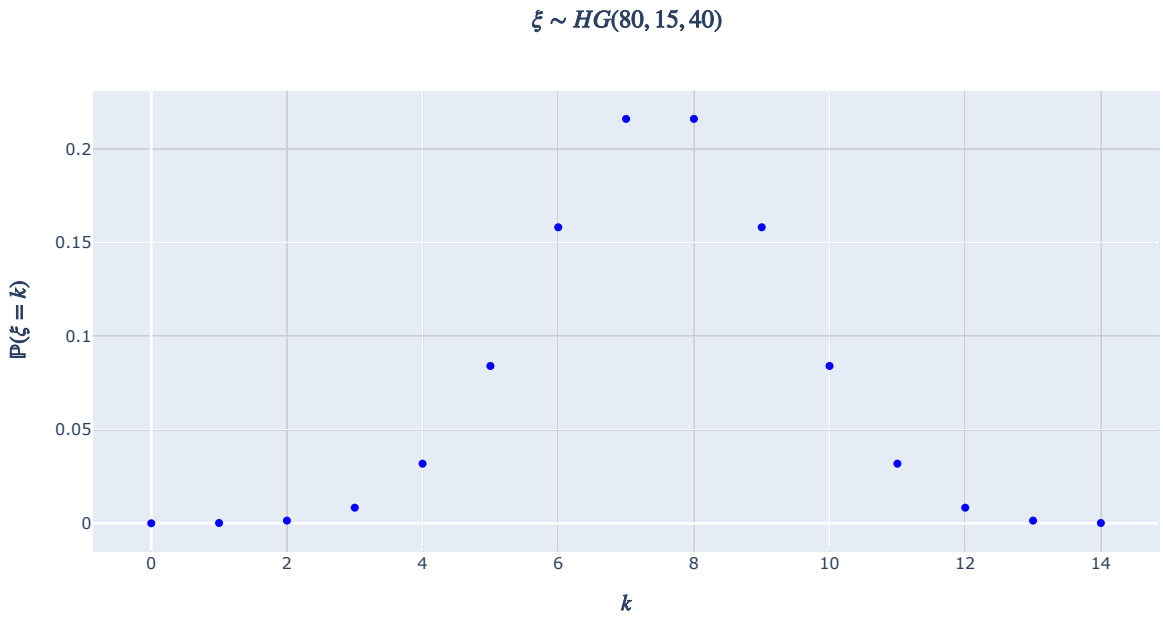
In [4]:
```
hg_data_x = np.arange(*xi.domain)
```

И, наконец, построим гистограмму и выведем ее:

In [5]:
```
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 $\{\xi < k\} = \bigcup_{i=0}^{k-1} \{\xi = i\}$. 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:

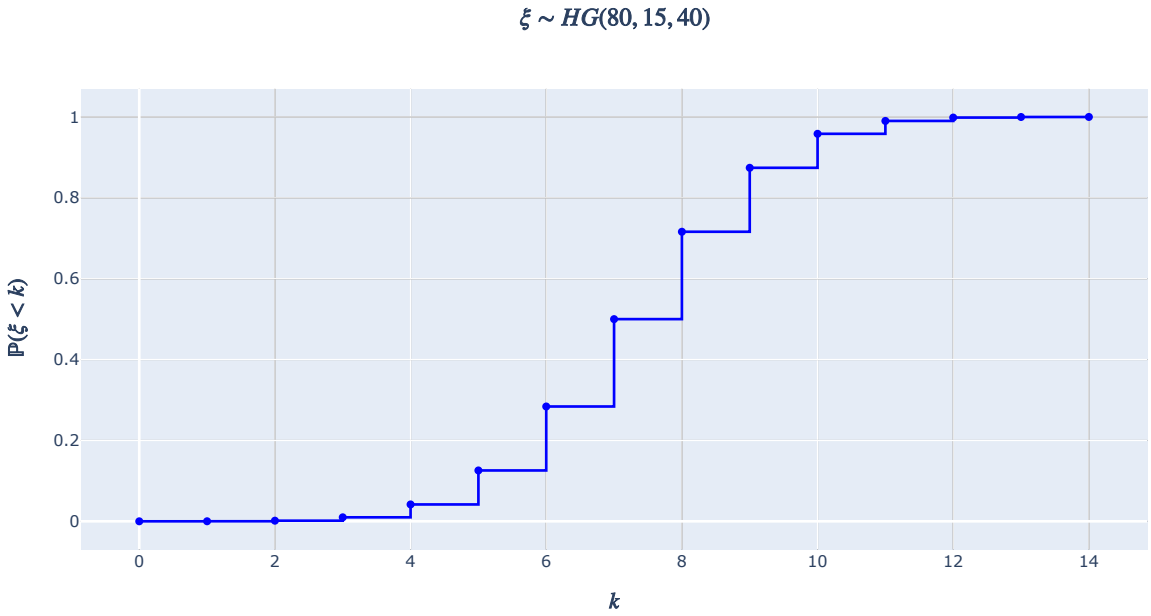
$$F_\xi(k) = \sum_{i=0}^{k-1} \mathbb{P}(\xi = i) = \sum_{i=0}^{k-1} \frac{\binom{m}{i}\binom{N-m}{n-i}}{\binom{N}{n}}$$

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):

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

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:

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

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:

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

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]
    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)
```

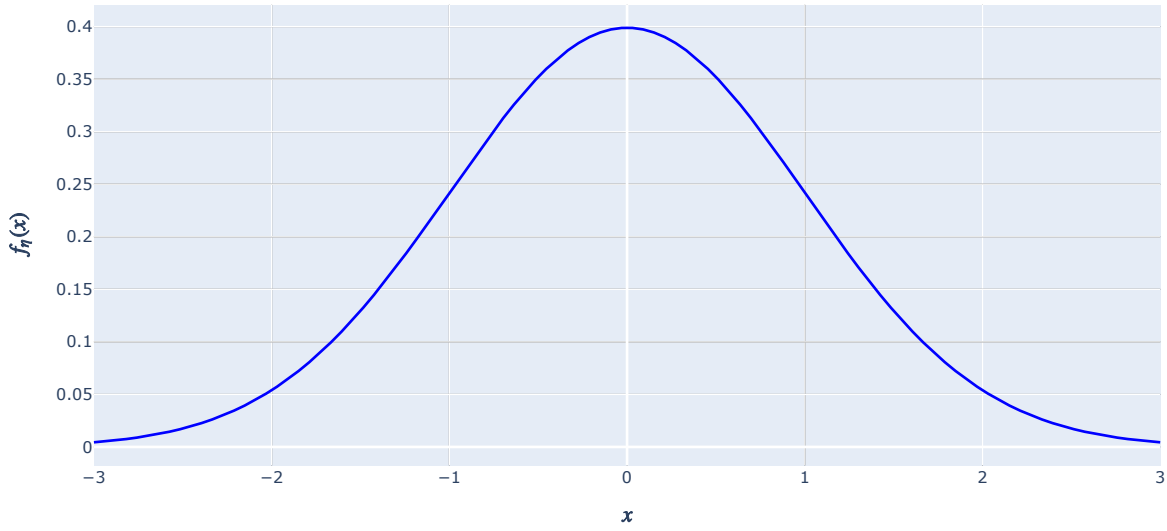
И, наконец, построим плотность, используя метод `N.f` нашего класса:

In [11]:
```
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)
```

$$\eta \sim N(0, 1^2)$$



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:

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

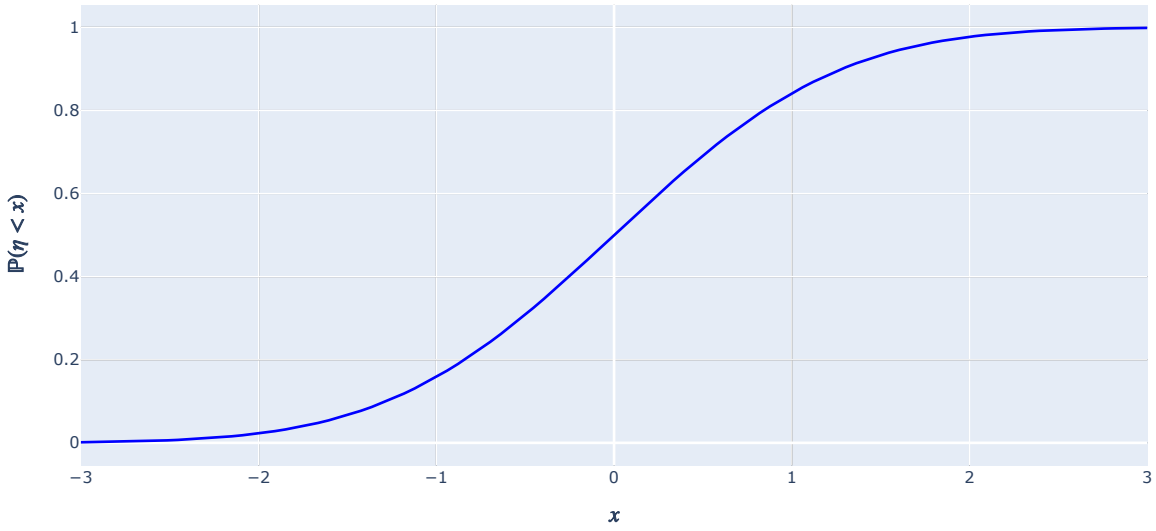
Построим график этой функции, используя метод `Norm.p` :

In [12]:
```
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)
```

$$\eta \sim N(0, 1^2)$$



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 [13]:
```
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 [14]:
```
@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 распределения n.

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

In [15]:
```
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 [16]:
```
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` выборо соответствующего объема из распределения $HG(80, 15, 20)$.
Выведем выборки для `q = 5` и `q = 10`:

In [17]:
```
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: [8 5 8 5 6]
#2: [ 9 10  8  7  6]
#3: [9 4 9 9 6]
#4: [6 5 8 6 7]
#5: [7 7 5 6 6]
====== q = 10 ======:
#1: [11  7  9  5  8  7  6 10  7  8]
#2: [11  8  7  9  9  6 11  7  7  8]
#3: [11  7  5 10  8  9 10  6  8  8]
#4: [11  4  7  8  7  6  4  7 11  5]
#5: [5 8 8 7 8 9 7 7 6 5]
```

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):

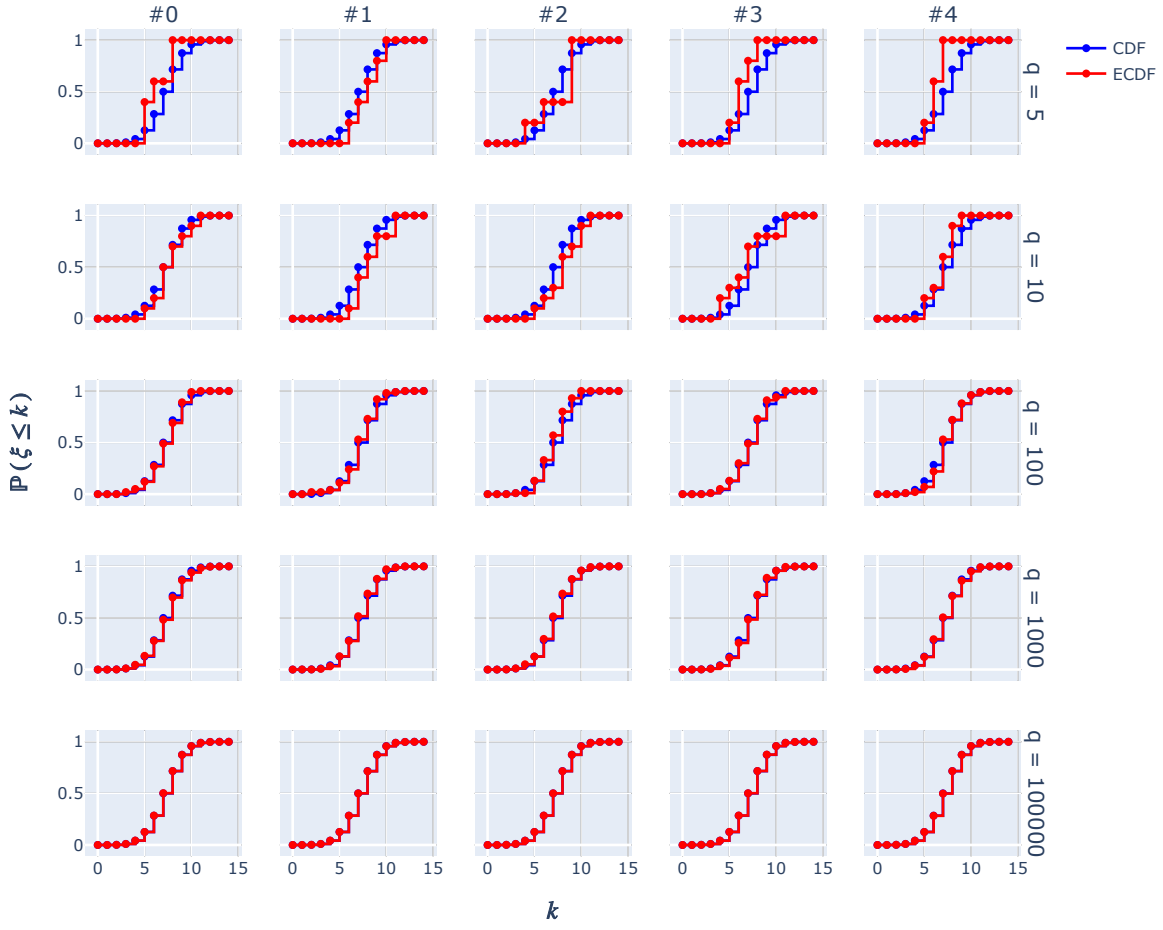
In [18]:
```
HG_ECDFs = {}
for q in samples_hg:
    HG_ECDFs[q] = np.ndarray(shape=(samples_hg[q].shape[0], hg_data_x.shape[0]), dtype=np.float)
    for i, sample in enumerate(samples_hg[q]):
        HG_ECDFs[q][i] = ECDF(sample)(hg_data_x)
```

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

In [19]:
```
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 $\Delta_q^{i,j}$:

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

In [20]:
```
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()}')
```

```
====== q = 5 ======:
i = 0; j = 1; max delta = 0.4000000000000001
i = 0; j = 2; max delta = 0.6
i = 0; j = 3; max delta = 0.2
i = 0; j = 4; max delta = 0.3999999999999999
i = 1; j = 2; max delta = 0.20000000000000007
i = 1; j = 3; max delta = 0.4000000000000001
i = 1; j = 4; max delta = 0.6
i = 2; j = 3; max delta = 0.6
i = 2; j = 4; max delta = 0.6
i = 3; j = 4; max delta = 0.19999999999999996
====== q = 10 ======:
i = 0; j = 1; max delta = 0.10000000000000009
i = 0; j = 2; max delta = 0.19999999999999996
i = 0; j = 3; max delta = 0.20000000000000007
i = 0; j = 4; max delta = 0.19999999999999996
i = 1; j = 2; max delta = 0.1
i = 1; j = 3; max delta = 0.30000000000000004
i = 1; j = 4; max delta = 0.30000000000000004
i = 2; j = 3; max delta = 0.4
i = 2; j = 4; max delta = 0.30000000000000004
i = 3; j = 4; max delta = 0.2
====== q = 100 ======:
i = 0; j = 1; max delta = 0.040000000000000036
i = 0; j = 2; max delta = 0.10999999999999999
i = 0; j = 3; max delta = 0.04999999999999993
i = 0; j = 4; max delta = 0.05000000000000002
i = 1; j = 2; max delta = 0.09
i = 1; j = 3; max delta = 0.05999999999999997
i = 1; j = 4; max delta = 0.040000000000000036
i = 2; j = 3; max delta = 0.08000000000000007
i = 2; j = 4; max delta = 0.11000000000000001
i = 3; j = 4; max delta = 0.07999999999999999
====== q = 1000 ======:
i = 0; j = 1; max delta = 0.03799999999999992
i = 0; j = 2; max delta = 0.03799999999999992
i = 0; j = 3; max delta = 0.025999999999999912
i = 0; j = 4; max delta = 0.02300000000000002
i = 1; j = 2; max delta = 0.019999999999999962
i = 1; j = 3; max delta = 0.03200000000000003
i = 1; j = 4; max delta = 0.02400000000000002
i = 2; j = 3; max delta = 0.03899999999999998
i = 2; j = 4; max delta = 0.02400000000000002
i = 3; j = 4; max delta = 0.035999999999999976
====== q = 100000 ======:
i = 0; j = 1; max delta = 0.0019400000000000528
i = 0; j = 2; max delta = 0.0014899999999999913
i = 0; j = 3; max delta = 0.0020499999999999963
i = 0; j = 4; max delta = 0.00377999999999995
i = 1; j = 2; max delta = 0.0009899999999999909
i = 1; j = 3; max delta = 0.0023400000000000087
i = 1; j = 4; max delta = 0.0023099999999999787
i = 2; j = 3; max delta = 0.0015199999999999658
i = 2; j = 4; max delta = 0.0022899999999999587
i = 3; j = 4; max delta = 0.0032899999999999596
```

Построение вариационного ряда выборки

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

In [21]:
```
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 [22]:
```
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 5 6 8 8]
#2: [ 6  7  8  9 10]
#3: [4 6 9 9 9]
#4: [5 6 6 7 8]
#5: [5 6 6 7 7]
====== q = 10 ======:
#1: [ 5  6  7  7  7  8  8  9 10 11]
#2: [ 6  7  7  7  8  8  9  9 11 11]
#3: [ 5  6  7  8  8  8  9 10 10 11]
#4: [ 4  4  5  6  7  7  7  8 11 11]
#5: [5 5 6 7 7 7 8 8 8 9]
```

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 [23]:
```
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 $\vec{X}$ - элемент вариационного ряда этой выборки стоящий на позиции $\left[\alpha \left|\vec{X}\right| + 1\right]$.

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

In [24]:
```
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]:2}')
```

```
======= q = 5 =======:
------ a = 0.1 ------
#0; quantile =  5, real delta =  0
#1; quantile =  6, real delta =  1
#2; quantile =  4, real delta = -1
#3; quantile =  5, real delta =  0
#4; quantile =  5, real delta =  0
------ a = 0.5 ------
#0; quantile =  6, real delta = -1
#1; quantile =  8, real delta =  1
#2; quantile =  9, real delta =  2
#3; quantile =  6, real delta = -1
#4; quantile =  6, 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 =  7, real delta = -1
#4; quantile =  7, real delta = -1
======= q = 10 =======:
------ a = 0.1 ------
#0; quantile =  6, real delta =  1
#1; quantile =  7, real delta =  2
#2; quantile =  6, real delta =  1
#3; quantile =  4, real delta = -1
#4; quantile =  5, real delta =  0
------ a = 0.5 ------
#0; quantile =  8, real delta =  1
#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 =  9, real delta =  1
#1; quantile =  9, real delta =  1
#2; quantile = 10, real delta =  2
```

```
#3; quantile =  8, real delta =  0
#4; quantile =  8, real delta =  0
======= 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 =  6, real delta =  1
------ a = 0.5 ------
#0; quantile =  8, real delta =  1
#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 =  9, real delta =  1
#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 = 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 =  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 =  9, real delta =  1
#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 =  7, real delta =  0
#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
```

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...

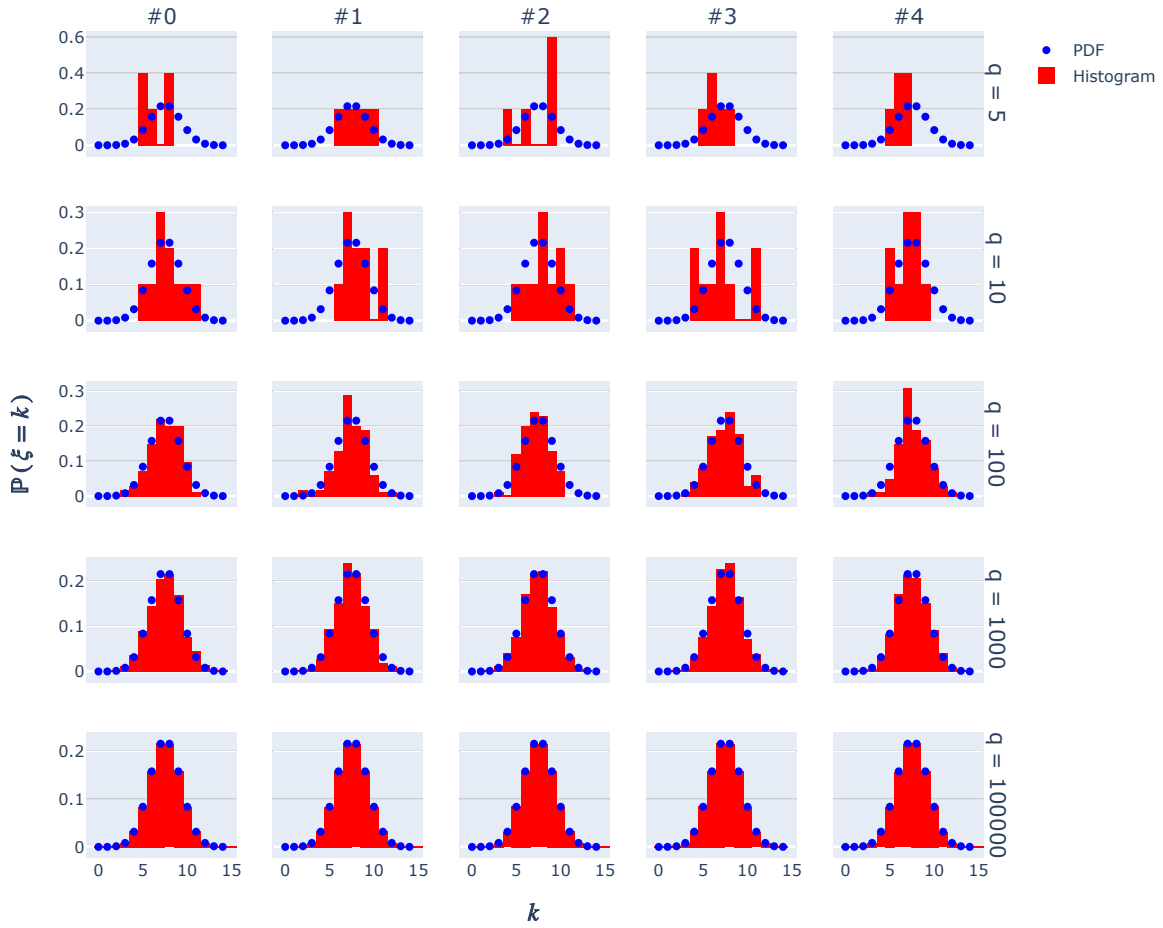
Гистограмма и полигон частот

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

In [25]:
```
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 [26]:
```
@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 [27]:
```
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 [28]:
```
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 [29]:
```
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: [ 0.38355074  0.59864973 -0.94441869  0.45670365 -0.69624523]
#2: [-1.05264043  0.21162356 -0.44096205 -1.29705104  2.03914677]
#3: [-0.88190105  0.67341396 -1.45364005  0.34844937  0.11932992]
#4: [ 0.12450243 -0.40167478  0.12017491 -1.64496674  0.8686007 ]
#5: [-1.39305143  1.05806701  0.20210074  2.17812796  2.20750802]
====== q = 10 ======:
#1: [-1.27057673 -0.11156886  3.0094608   0.82423796 -1.07507477 -0.09805348
  0.03886828 -0.90111851 -3.24439435 -0.53209576]
#2: [ 0.84127522  0.09483126  0.84264207 -2.33533499  0.24972638  0.25361827
  0.86115008 -1.23328725  0.70884954  1.60496484]
#3: [ 0.85825636  1.20123068  1.14856207  1.05561005 -0.47531636  1.92480635
 -0.93075042  0.64522756 -1.32650606 -0.48631989]
#4: [ 1.21376594  0.2277509   0.6617112  -0.32800958  2.08490629 -0.18307111
 -1.04879371  1.11346897  0.51526264  0.56837484]
#5: [-1.95333426 -1.23941243 -0.71998845  0.75785751 -1.59169291  1.27870523
 -0.43075285 -0.3987848  -0.34393893  0.61940757]
```

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):

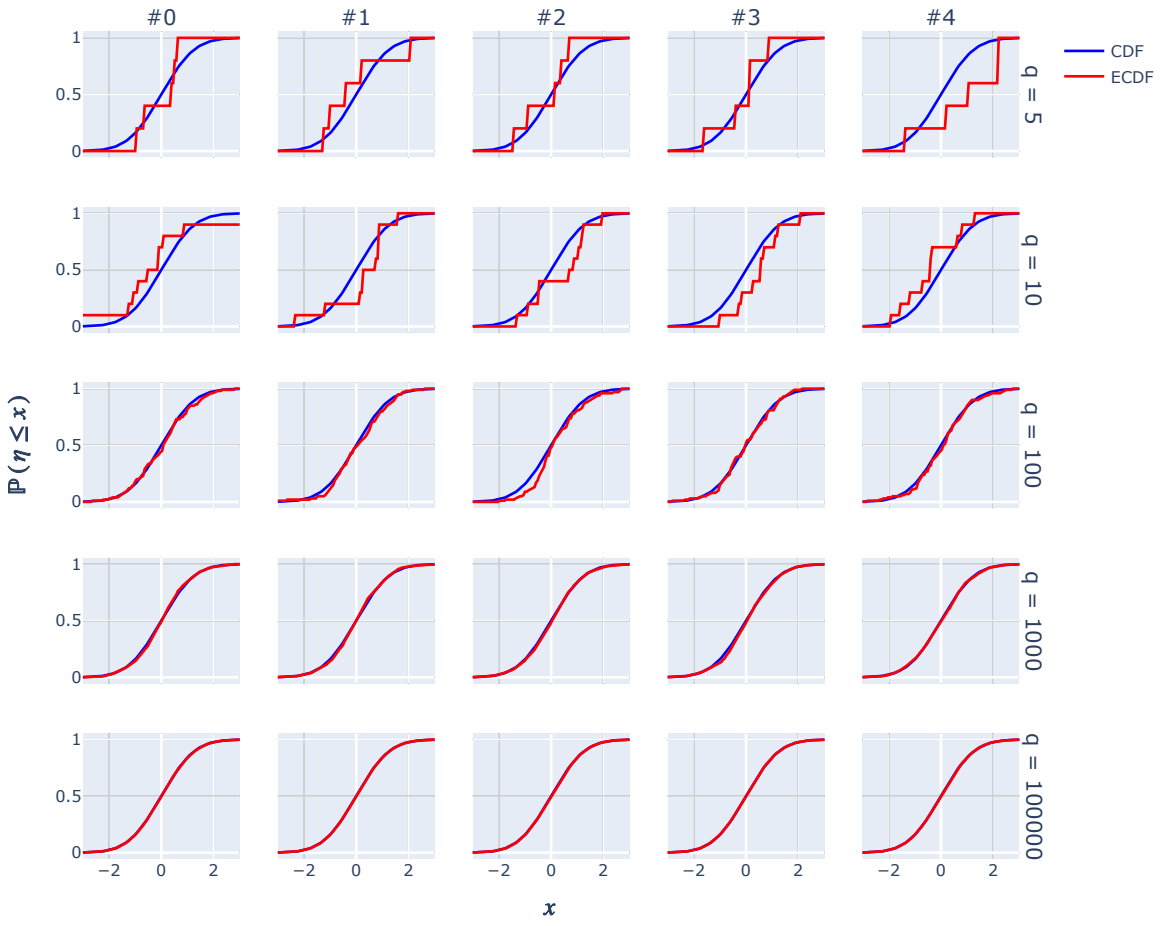
In [30]:
```
Norm_ECDFs = {}
for q in samples_norm:
    Norm_ECDFs[q] = np.ndarray(shape=(samples_norm[q].shape[0], norm_data_x.shape[0]), dtype=np.float)
    for i, sample in enumerate(samples_norm[q]):
        Norm_ECDFs[q][i] = ECDF(sample)(norm_data_x)
```

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

In [31]:
```
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 [32]:
```
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()}')
```

```
====== q = 5 ======:
i = 0; j = 1; max delta = 0.4
i = 0; j = 2; max delta = 0.20000000000000007
i = 0; j = 3; max delta = 0.4
i = 0; j = 4; max delta = 0.6
i = 1; j = 2; max delta = 0.20000000000000007
i = 1; j = 3; max delta = 0.20000000000000007
i = 1; j = 4; max delta = 0.4000000000000001
i = 2; j = 3; max delta = 0.2
i = 2; j = 4; max delta = 0.6
i = 3; j = 4; max delta = 0.6000000000000001
====== q = 10 ======:
i = 0; j = 1; max delta = 0.6000000000000001
i = 0; j = 2; max delta = 0.4
i = 0; j = 3; max delta = 0.5
i = 0; j = 4; max delta = 0.20000000000000007
i = 1; j = 2; max delta = 0.30000000000000004
i = 1; j = 3; max delta = 0.20000000000000007
i = 1; j = 4; max delta = 0.5
i = 2; j = 3; max delta = 0.30000000000000004
i = 2; j = 4; max delta = 0.4
i = 3; j = 4; max delta = 0.6000000000000001
====== q = 100 ======:
i = 0; j = 1; max delta = 0.07000000000000002
i = 0; j = 2; max delta = 0.16000000000000003
i = 0; j = 3; max delta = 0.09000000000000002
i = 0; j = 4; max delta = 0.08000000000000002
i = 1; j = 2; max delta = 0.12000000000000002
i = 1; j = 3; max delta = 0.06000000000000005
i = 1; j = 4; max delta = 0.07000000000000006
i = 2; j = 3; max delta = 0.14
i = 2; j = 4; max delta = 0.11000000000000001
i = 3; j = 4; max delta = 0.08000000000000002
====== q = 1000 ======:
i = 0; j = 1; max delta = 0.01899999999999996
i = 0; j = 2; max delta = 0.028000000000000025
i = 0; j = 3; max delta = 0.033
i = 0; j = 4; max delta = 0.03300000000000003
i = 1; j = 2; max delta = 0.03600000000000003
i = 1; j = 3; max delta = 0.029000000000000026
i = 1; j = 4; max delta = 0.04300000000000004
i = 2; j = 3; max delta = 0.033
i = 2; j = 4; max delta = 0.02899999999999997
i = 3; j = 4; max delta = 0.04899999999999999
====== q = 100000 ======:
i = 0; j = 1; max delta = 0.003569999999999962
i = 0; j = 2; max delta = 0.0033599999999999186
i = 0; j = 3; max delta = 0.005149999999999988
i = 0; j = 4; max delta = 0.0042200000000000015
i = 1; j = 2; max delta = 0.0024999999999999467
i = 1; j = 3; max delta = 0.0031799999999999606
i = 1; j = 4; max delta = 0.0022100000000000453
i = 2; j = 3; max delta = 0.005079999999999973
i = 2; j = 4; max delta = 0.002950000000000008
i = 3; j = 4; max delta = 0.0023099999999999232
```

Построение вариационного ряда выборки

Построение вариационного ряда выборки

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

In [33]:
```
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 [34]:
```
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: [-0.94441869 -0.69624523  0.38355074  0.45670365  0.59864973]
#2: [-1.29705104 -1.05264043 -0.44096205  0.21162356  2.03914677]
#3: [-1.45364005 -0.88190105  0.11932992  0.34844937  0.67341396]
#4: [-1.64496674 -0.40167478  0.12017491  0.12450243  0.8686007 ]
#5: [-1.39305143  0.20210074  1.05806701  2.17812796  2.20750802]
====== q = 10 ======:
#1: [-3.24439435 -1.27057673 -1.07507477 -0.90111851 -0.53209576 -0.11156886
 -0.09805348  0.03886828  0.82423796  3.0094608 ]
#2: [-2.33533499 -1.23328725  0.09483126  0.24972638  0.25361827  0.70884954
  0.84127522  0.84264207  0.86115008  1.60496484]
#3: [-1.32650606 -0.93075042 -0.48631989 -0.47531636  0.64522756  0.85825636
  1.05561005  1.14856207  1.20123068  1.92480635]
#4: [-1.04879371 -0.32800958 -0.18307111  0.2277509   0.51526264  0.56837484
  0.6617112   1.11346897  1.21376594  2.08490629]
#5: [-1.95333426 -1.59169291 -1.23941243 -0.71998845 -0.43075285 -0.3987848
 -0.34393893  0.61940757  0.75785751  1.27870523]
```

Квантили

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

In [35]:
```
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]}')
            break
```

```
a = 0.1, quantile = -1.2424242424242424
a = 0.5, quantile = 0.030303030303030276
a = 0.7, quantile = 0.5757575757575757
```

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 [36]:
```
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}, real delta = {quantile - norm_quantiles[a]:2}')
```

```
======= q = 5 =======:
------ a = 0.1 ------
#0; quantile = -0.9444186920823205, real delta = 0.2980055503419219
#1; quantile = -1.2970510443433028, real delta = -0.05462680191906033
#2; quantile = -1.4536400488199095, real delta = -0.21121580639566706
#3; quantile = -1.644966739388878, real delta = -0.4025424969646356
#4; quantile = -1.3930514289798257, real delta = -0.15062718655558327
------ a = 0.5 ------
#0; quantile = 0.3835507440881929, real delta = 0.3532477137851626
#1; quantile = -0.440962048182873, real delta = -0.47126507848590327
#2; quantile = 0.11932992256959918, real delta = 0.0890268922665689
#3; quantile = 0.12017491057403645, real delta = 0.08987188027100618
#4; quantile = 1.0580670109641357, real delta = 1.0277639806611054
------ a = 0.7 ------
#0; quantile = 0.4567036478602458, real delta = -0.11905392789732988
#1; quantile = 0.21162356198807286, real delta = -0.36413401376950283
#2; quantile = 0.3484493748934855, real delta = -0.2273082008640902
#3; quantile = 0.12450243347550126, real delta = -0.4512551422820744
#4; quantile = 2.1781279630496755, real delta = 1.6023703872920998
======= q = 10 =======:
------ a = 0.1 ------
#0; quantile = -1.2705767253871838, real delta = -0.028152482962941372
#1; quantile = -1.2332872510140196, real delta = 0.009136991410222839
#2; quantile = -0.9307504246906443, real delta = 0.31167381773359815
#3; quantile = -0.3280095843846181, real delta = 0.9144146580396244
#4; quantile = -1.5916929057395788, real delta = -0.34926866331533635
------ a = 0.5 ------
#0; quantile = -0.11156885582138329, real delta = -0.14187188612441357
#1; quantile = 0.7088495433349801, real delta = 0.6785465130319498
#2; quantile = 0.8582563581988674, real delta = 0.8279533278958371
#3; quantile = 0.568374838693011, real delta = 0.5380718083899807
#4; quantile = -0.398784797666701, real delta = -0.4290878279697313
------ a = 0.7 ------
#0; quantile = 0.03886828146691953, real delta = -0.5368892942906561
#1; quantile = 0.8426420712897361, real delta = 0.2668844955321604
#2; quantile = 1.1485620652283899, real delta = 0.5728044894708142
#3; quantile = 1.113468973125666, real delta = 0.5377113973680903
```

```
#4; quantile = 0.6194075705016663, real delta = 0.04364999474409059
======= q = 100 =======:
------ a = 0.1 ------
#0; quantile = -1.2923165821510065, real delta = -0.0498923397267641
#1; quantile = -1.0039358553942959, real delta = 0.23848838702994657
#2; quantile = -0.8323143783412852, real delta = 0.41010986408295724
#3; quantile = -1.0359454219850521, real delta = 0.2064788204391903
#4; quantile = -1.1365602529801084, real delta = 0.105863989444134
------ a = 0.5 ------
#0; quantile = 0.06871859055144777, real delta = 0.038415560248417496
#1; quantile = 0.08517757651209072, real delta = 0.054874546209060446
#2; quantile = 0.05111149996030969, real delta = 0.020808469657279414
#3; quantile = 0.0021270793140251, real delta = -0.028175950989005175
#4; quantile = 0.11465108496956569, real delta = 0.08434805466653542
------ a = 0.7 ------
#0; quantile = 0.5558949819086519, real delta = -0.01986259384892375
#1; quantile = 0.6304441421825149, real delta = 0.05468656642493919
#2; quantile = 0.6387497807301447, real delta = 0.06299220497256897
#3; quantile = 0.6127796034745421, real delta = 0.03702202771696639
#4; quantile = 0.5712388702947007, real delta = -0.004518705462875028
======= q = 1000 =======:
------ a = 0.1 ------
#0; quantile = -1.2510637906829822, real delta = -0.008639548258739804
#1; quantile = -1.2405883731463827, real delta = 0.0018358692778597252
#2; quantile = -1.252624670098675, real delta = -0.010200427674432522
#3; quantile = -1.203297348908296, real delta = 0.03912689351594634
#4; quantile = -1.3343878829753422, real delta = -0.09196364055109973
------ a = 0.5 ------
#0; quantile = 0.018516057517086227, real delta = -0.01178697278594405
#1; quantile = -0.009763252736279211, real delta = -0.04006628303930949
#2; quantile = 0.03600154301929937, real delta = 0.0056985127162690954
#3; quantile = 0.03523699220301441, real delta = 0.004933961899984136
#4; quantile = 0.037449572689190996, real delta = 0.00714654238616072
------ a = 0.7 ------
#0; quantile = 0.5090798997475553, real delta = -0.06667767601002039
#1; quantile = 0.45972219093018785, real delta = -0.11603538482738784
#2; quantile = 0.5310064611304669, real delta = -0.04475111462710879
#3; quantile = 0.5515114147187896, real delta = -0.024246161038786074
#4; quantile = 0.5635502893874376, real delta = -0.012207286370138126
======= q = 100000 =======:
------ a = 0.1 ------
#0; quantile = -1.2739127953178442, real delta = -0.03148855289360175
#1; quantile = -1.2755553897620373, real delta = -0.033131147337794875
#2; quantile = -1.2732290706321252, real delta = -0.030804828207882773
#3; quantile = -1.2865949159934214, real delta = -0.04417067356917892
#4; quantile = -1.2766589077688586, real delta = -0.034234665344616166
------ a = 0.5 ------
#0; quantile = 0.0041160953731742365, real delta = -0.02618693492985604
#1; quantile = 0.0006008456932860534, real delta = -0.029702184609744223
#2; quantile = 0.004836512943998832, real delta = -0.025466517359031444
#3; quantile = -0.004116524846478955, real delta = -0.03441955514950923
#4; quantile = -0.0006548296734215674, real delta = -0.030957859976451844
------ a = 0.7 ------
#0; quantile = 0.530850437907307, real delta = -0.044907137850268675
#1; quantile = 0.5258873053423633, real delta = -0.049870270415212414
#2; quantile = 0.5298218174558559, real delta = -0.04593575830171981
#3; quantile = 0.5190988958206868, real delta = -0.05665867993688889
#4; quantile = 0.5248943839096208, real delta = -0.05086319184795485
```

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

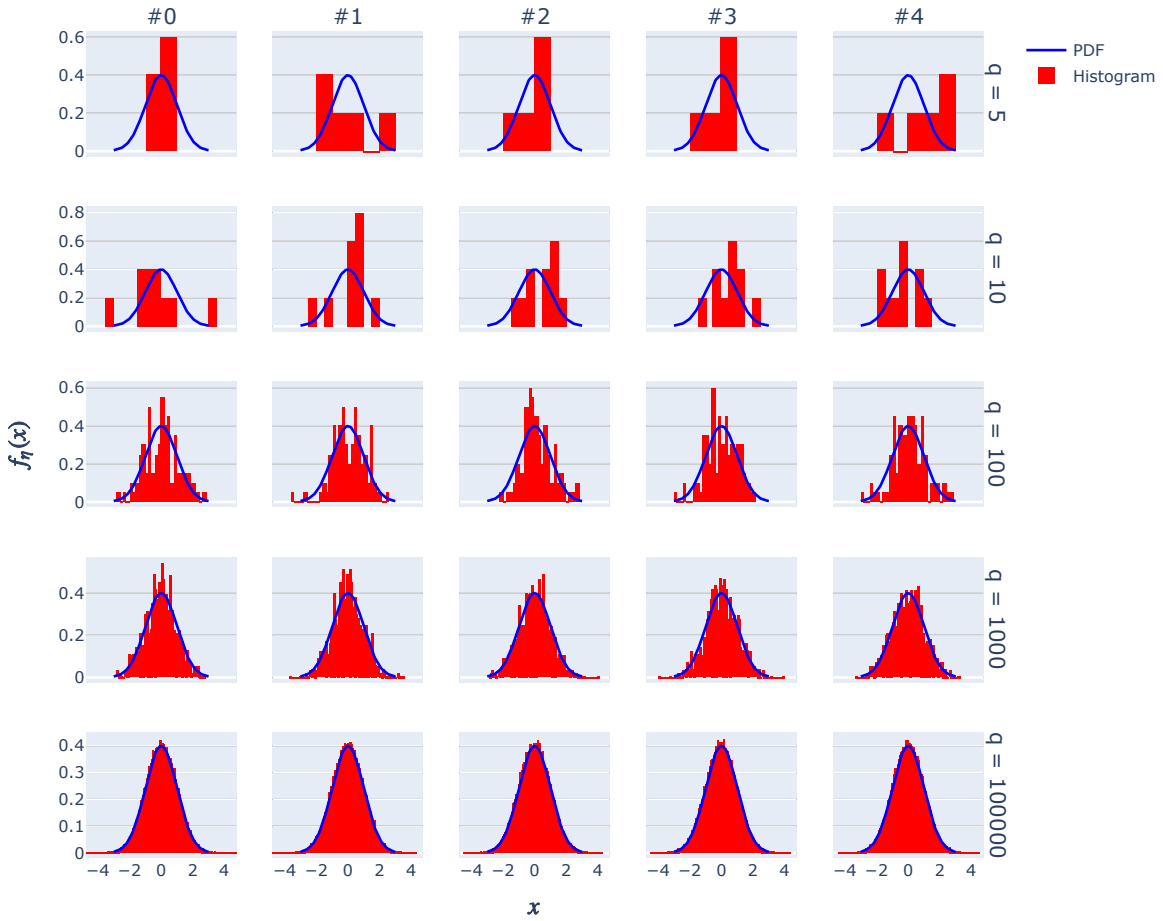
Гистограмма и полигон частот

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 [37]:
```
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.