In [1]:
```python
import random as rd  # Генерация псевдослучайных чисел
import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt  # Графики
from scipy.stats import norm  # Для теоретической функции распределения
from scipy.stats import chi2  # Проверка гипотез
import scipy.integrate as spi  # Поиск интегралов
```

In [2]:
```python
class Worker:
    def __init__(self, N_: int, A_: float, B_: float, size_: int):
        self.N = N_
        self.A = A_
        self.B = B_
        self.size = size_
        self.borders = {"left": 2 * self.A - (math.sqrt(12 * self.B) + 2 * self.A) / 2,  # Границы распределени
                        "right": (math.sqrt(12 * self.B) + 2 * self.A) / 2}
        self.intervals = []
        self.mu = A_ * N_  # Теоретическое мат. ожидание
        self.sigma = math.sqrt(B_ * N_)  # Теоретическое СКО
        self.intervals_q = []  # Равновероятные интервалы для проверки гипотезы распределения

    # Обратная функция распределения, равномерное распределение
    def inverse_distribution_function(self, y: float) -> float:  # y ∈ [0; 1)
        x = y * (self.borders['right'] - self.borders['left']) + self.borders['left']  # x ∈ [left: right)
        return x

    # Получение равномерно распределенной случайной величины [left: right)
    def get_uniform_distribution(self) -> list:
        result = [0] * self.N
        for i in range(len(range(self.N))):
            y = rd.uniform(0 , 1)  # пПсевдослучайное значение от 0 до 1
            x = self.inverse_distribution_function(y)  # Случайное значение от left до right
            result[i] = x
        return result

    # Случайная величина - общая сумма вклада
    def total_deposit_amount(self, contributions: list) -> float:  #(Contributions - вклады)
        return np.sum(contributions)

     # Нормальное распределение по центральной предельной теореме
# Часть 1
    # Розыгрыш случайной величины
    def gauss_distribution(self) -> list:
        theta = [0] * int(self.size)  # N равномерно распределенных с.в.
        for i in range(self.size):
            temp = self.get_uniform_distribution()
            theta[i] = self.total_deposit_amount(temp)
        theta.sort()
        #theta = [i / self.N for i in theta]
        self.gauss_distribution = theta
        return theta

# Часть 2
# Определение теоретических и выборочных числовых характеристикик
    def statistical_characteristics(self) -> pd.DataFrame:
        E_theta = self.A * self.N  # +
        x_ =  np.sum(self.gauss_distribution) / self.size
        E_minus_x = abs(E_theta - x_)

        D_theta = self.B * self.N
        S_2 = (np.sum(pow(self.gauss_distribution - x_, 2) / self.size))
        D_minus_S = abs(D_theta - S_2)

        R = self.gauss_distribution[-1] - self.gauss_distribution[0]
        Me = 0
        if len(self.gauss_distribution) % 2:
            Me = self.gauss_distribution[int(len(self.gauss_distribution) / 2)]
        else:
            Me = (self.gauss_distribution[int(len(self.gauss_distribution) / 2) - 1]\
            + self.gauss_distribution[int(len(self.gauss_distribution) / 2)]) / 2

        F_teor = [norm.cdf((self.gauss_distribution[i] - self.A * self.N) \
                           / math.sqrt(self.B * self.N)) for i in range(self.size)]
        F_selective = [i / self.size for i in range(self.size)]
        D = np.max(np.abs(np.subtract(F_teor, F_selective)))

        df = pd.DataFrame(
        {
            "Eη": E_theta ,  # Теоричетическое математическое ожидание
            "$\overline{x}$": x_,  # выборочное среднее
            "|Eη - $\overline{x}$|": E_minus_x,

            "Dη": D_theta, # Теоретическая диспресия
            "$S^2$": S_2,  # Выборочная дисперсия
            "|Dη - $S^2$|": D_minus_S,
```

```
            "$\hat{Me}$": Me,  # Выборочная медиана
            "$\hat{R}$": R,  # Размах выборки
            "D": D,  # Мера расхождения

            "-": ['_','__']
        })
        df.drop('-', axis = 1,inplace = True)
        df.drop(0, inplace = True)
        return df

    def get_D(self) -> pd.DataFrame:
        F_teor = [norm.cdf((self.gauss_distribution[i] - self.A * self.N) \
                           / float(math.sqrt(self.B * float(self.N)))) for i in range(self.size)]
        F_selective = [i / self.size for i in range(self.size)]
        abs_variance = [abs(F_teor[i] - F_selective[i]) for i in range(self.size)]
        df = pd.DataFrame({"Выборочная ф-я": F_selective, "Теоретическая ф-я": F_teor, "Расхождение": abs_varia
        return df

# Построение графиков теоретической Fη(x) и выборочной F'η(x) функций распределения.
    def distr_function(self):
        figure = plt.figure(figsize = (16, 12))
        Oy = [i / self.size for i in range(self.size)]
#         self.mu = self.A * self.N
#         self.sigma = math.sqrt(self.B * self.N)
        x = np.arange(np.min(self.gauss_distribution), np.max(self.gauss_distribution))

        plt.xlabel("η")
        plt.ylabel("Fη", rotation = 0)
        plt.plot(self.gauss_distribution, Oy, label = f'Выборочная функция распределения, N = {self.N}')
        plt.plot(x,norm.cdf(x,self.mu,self.sigma), label=f'Теоретическая функция распределения, μ: {self.mu}, σ
        plt.legend(loc="upper left", prop={'size': 15})
        plt.grid()
        plt.show()

    def check_distr(self):
        figure = plt.figure(figsize = (20, 12))
        Oy = [i / self.size for i in range(self.size)]
#         self.mu = self.A * self.N
#         self.sigma = math.sqrt(self.B * self.N)
        selectiveFunction = pd.DataFrame({'Fη':[], 'η':[]})
        counter = 2
        selectiveFunction.loc[0] = (self.gauss_distribution[0] - 3 * self.sigma, 0)
        selectiveFunction.loc[1] = (self.gauss_distribution[0], 0)
        for i in range(self.size - 1):
            selectiveFunction.loc[counter] = (self.gauss_distribution[i], (i + 1) / self.size)
            counter+=1
            selectiveFunction.loc[counter] = (self.gauss_distribution[i + 1], (i + 1) / self.size)
            counter+=1
        selectiveFunction.loc[counter] = (self.gauss_distribution[-1], 1)
        selectiveFunction.loc[counter + 1] = (self.gauss_distribution[-1] +  3 * self.sigma, 1)
        max_ =  np.max(self.gauss_distribution)
        min_ = np.min(self.gauss_distribution)

        x = np.arange(min_ - 3 * self.sigma, max_ + 3 * self.sigma, (max_ - min_ + 1) / 50000)
        plt.xlabel("η")
        plt.ylabel("Fη", rotation = 0)
        plt.plot(selectiveFunction['Fη'], selectiveFunction['η'], \
                 label = f'Выборочная функция распределения, N = {self.N}')
        plt.plot(x, norm.cdf(x,self.mu,self.sigma),label=f'Теоретическая функция распределения, μ: {self.mu}, σ
        plt.legend(loc="upper left", prop={'size': 15})
        plt.grid()
        plt.show()

# Часть 3
# Таблица с интервалами
    def set_intervals(self, manual_input = False) -> list:
        """manual_input - ручной ввод числа интервалов, выбор границ интервалов."""
        if not manual_input:
            column = 1 + int(np.log2(self.size))#количество столбцов по формуле Стёрджеса
            h = (max(self.gauss_distribution) - min(self.gauss_distribution)) / column
            self.intervals = [min(self.gauss_distribution) + i * h for i in range(column)]
        else:
            column = int(input('Число интервалов:'))
            self.intervals = [0] * max((column - 1), 1)
            for i in range(len(self.intervals)):
                self.intervals[i] = float(input(f'z{i+1}:'))

        max_ =  np.max(self.gauss_distribution) + 3 * self.sigma  # Границы
       # min_ = np.min(self.gauss_distribution) - 3 * self.sigma
       # if min(self.intervals) > min(self.gauss_distribution): #[min(self.gauss_distribution) - 10]
        #    self.intervals = [min_] + self.intervals
        if max(self.intervals) < max(self.gauss_distribution):
            self.intervals = self.intervals + [max_]# [max(self.gauss_distribution) + 10]
        return self.intervals

    def __get_params__(self):
        """
        Вычисление z_i по заданным границам промежутков (z)
```

```
        Вычисление теоретической плотности распределения в точках z_i (fn)
        Вычисление высоты прямоугольников для гистограммы и таблицы (h)
        Считается, что интервалы уже заданы
        """
        if len(self.intervals) == 0:
            print('Интервалы не заданы!')
            return

        # Середины интервалов (z)
        self.z = [float('-inf')] + \
        [(self.intervals[k] + self.intervals[k + 1]) / 2 for k in range(len(self.intervals)-1)] + \
        [float('inf')]
        # Теоретическая плотность в точке z[k] (fn)
#         self.mu = self.A * self.N  # Теоретическое мат. ожидание
#         self.sigma = math.sqrt(self.B * self.N)  # Теоретическое СКО
        self.fn = [norm.pdf(k, self.mu, self.sigma) for k in self.z]

        # (h)
        left_border = min(min(theta.gauss_distribution), min(self.intervals)) - 10  # Границы гистограммы
        right_border = max(max(theta.gauss_distribution), max(self.intervals)) + 10 # + 3 * self.sigma?
        fix_interval = [left_border] + self.intervals + [right_border]  # Полный интервал (border вместо inf)
        self.delta = [abs(fix_interval[i + 1] - fix_interval[i]) for i in range(len(fix_interval)- 1)]  # Длина
        # Высота столбцов гистограммы
        self.ni = []
        full_interval = [float('-inf')] + self.intervals + [float('inf')]  # Расширенный интервал на всю ось
        for k in range(len(full_interval)-1):
            self.ni.append(len(np.where([full_interval[k] <= elem < full_interval[k+1] for elem in self.gauss_d
        self.h = [self.ni[i] /  (self.size * self.delta[i]) for i in range(len(self.ni))]

    def create_table(self) -> pd.core.frame.DataFrame:
        """
        1) Таблица pd.DataFrame с четыремя строками:
        [interval[k], interval[k+1]) : typle
        z[k] - середина интервала interval[k], interval[k+1] : float
        fn(z[k]) - теоретическая плотность распределения в точке z[k] : float
        h[k] - высота k-го прямоугольника в гистограмме = n[i] / (delta[i] * n) : float
        2) max_loss - Максимальное расхождение  теоретической плотности распределения и гистограммы: float
        --------------------
        delta[k] - длина k-го интервала : float
        n[k] - абсолютная частота в интервале k : int
        n - размер выборки : int
        --------------------

        """
        if len(self.intervals) == 0:
            print('Интервалы не заданы!')
            return
        self.__get_params__()  # Получение характеристик для построения таблицы
        inter = [float('-inf')]+[self.intervals[k] for k in range(len(self.intervals)-1)]+[float('inf')]  # Инт
        # Результирующая таблица
        self.table = pd.DataFrame()
        self.table['info'] = ['delta', 'zk', 'fn', 'h']
        for i in range(len(self.h) - 1):
            if i == 0:
                self.table[str(i + 1)] = [str(f'({round(inter[i], 1)},{round(inter[i + 1],1)})'),self.z[i],self
            else:
                self.table[str(i + 1)] = [str(f'[{round(inter[i], 1)},{round(inter[i + 1], 1)})'),self.z[i],sel

        self.max_loss = max([abs(self.fn[i] - self.h[i]) for i in range(len(self.h))])
        return self.table

    def hist(self):
        """
        Построение гистограммы, перед этим вычисляется высота прямоугольников h в методе get_params
        """
        if len(self.intervals) == 0:
            print('Интервалы не заданы!')
            return

        self.__get_params__()
        # Гистограмма
        figure = plt.figure(figsize = (14, 8))
        x_left = [min(min(self.gauss_distribution), min(self.intervals)) - 1]
        x_right = [max(max(theta.gauss_distribution), max(self.intervals)) + 1]
        x = x_left + [val for val in self.intervals for _ in ('', '')] + x_right
        y = [val for val in self.h for _ in ('', '')]
        plt.plot(x[:-1], y[:-1], color = 'green')
        for i, x_ in enumerate(x):
            plt.vlines(x_, ymin = 0, ymax = y[i], color = 'green')

        # Параметры для теоретической плотности распределения
        # mu = self.A * self.N  # Теоретическое мат. ожидание
        # sigma = math.sqrt(self.B * self.N)  # Теоретическое СКО
        max_ =  np.max(self.gauss_distribution) + 3 * self.sigma  # Границы
        min_ = np.min(self.gauss_distribution) - 3 * self.sigma
        # Теоретическая плотность
        x = np.arange(min_ , max_, (max_ - min_ + 1 - 6 * self.sigma) / 50000)
        plt.plot(x, norm.pdf(x, self.mu, self.sigma), label = 'Теоретическая плотность')
        plt.title('Гистограмма')
```

```
        plt.legend()
        plt.grid()
        plt.show()

# Проверка гипотезы о виде распределения (то, что это нормальное распределение)
    def hypothesis_mapping(self):
        """
        Отображение гипотезы в виде теоретических вероятностей q1, q2,. . . , qk.
        q_i = P_{H_0}({η ∈Δ_j''}) = Fη(z_i)−Fη(z_{i−1})

        k - 1 - количество степеней свободы
        q_i - теоретические вероятности
        intervals_q - интервалы для проверки гипотезы(равновероятные)
        nq_i - абсолютная частота для равномерных интервалов
        """
        self.q_i = []
        self.intervals_q = []
        self.nq_i = []
        # k
        ###########################################################################
        ###########################################################################
        k = 4  # int(input('Количество интервалов для проверки гипотезы:'))
        ###########################################################################
        ###########################################################################
        step = 1 / k  # Вероятность каждого интервала
        self.q_i.append(step)
        for i in range(k - 1):
            # q_i
          #  print(f'k: {k}\nself.q_i: {self.q_i}\nself.intervals_q: {self.intervals_q}\nself.nq_i:{self.nq_i}'
            self.q_i.append(step)
            # intervals_q
            self.intervals_q.append(inverse_normal_cdf((i + 1) * step, self.mu, self.sigma))
        # nq_i
        full_interval = [float('-inf')] + self.intervals_q + [float('inf')]  # Расширенный интервал на всю ось
        for j in range(len(full_interval)-1):
            self.nq_i.append(len(np.where([full_interval[j]<=elem<full_interval[j+1] for elem in self.gauss_dis

      #  print(f'k: {k}\nself.q_i: {self.q_i}\nself.intervals_q: {self.intervals_q}\nself.nq_i:{self.nq_i}')

    def get_theoretical_probabilities(self):  # +
        self.hypothesis_mapping()
        res = pd.DataFrame()
        for i in range(len(self.q)):
            res[f'q{i+1}'] = [self.q[i]]
        return res

    def set_significance_level(self):
        ###########################################################################
        ###########################################################################
        # Ввод уровня значимости alpha
        self.alpha = 0.5#float(input('Введите уровнень значимости α: '))
        ###########################################################################
        ###########################################################################

    def get_R0(self) -> float:
        """
        Величина R0 характеризует меру расхождения между наблюдавшимися частотами
        и ожидаемым числом попаданий в интервал при нулевой гипотезы.
        """
        if len(self.intervals_q) == 0:
            self.hypothesis_mapping()
        self.R0 = 0
        for i in range(len(self.nq_i)):
         #   print(f'i = 0: R0 = {self.R0}')
            self.R0 += (self.nq_i[i] - self.size * self.q_i[i])**2 / (self.size * self.q_i[i])
        return self.R0

    def hypothesis_verdict(self):
        """
        Отображение вычисленного значения !F(R0) и решения о принятии или отвержении гипотезы H0.
        R0 характеризует меру расхождения между наблюдавшимися частотами
        и ожидаемым числом попаданий в интервал при нулевой гипотезы.
        При справедливости нулевой гипотезы величина R0 имеет распределение χ^2 с k − 1 степенями свободы.
        """
        self.get_R0()
        k = len(self.nq_i)
        self.nF_R0, _ = spi.quad(lambda x: chi2.pdf(x, k - 1), self.R0, float('inf'))  #
        self.set_significance_level()
#         if self.nF_R0 > self.alpha:
#             print(f'!F(R0) = {self.nF_R0} > {self.alpha} гипотеза принята')
#         else:
#             print(f'!F(R0) = {self.nF_R0} < {self.alpha} гипотеза отвергнута')


    def get_all_info(self):
        x = pd.DataFrame({"Реализации случайной величины": self.gauss_distribution})
        display(x)

        if len(self.intervals) == 0:
```

```
            self.set_intervals()
        table = self.create_table()
        D = str(self.statistical_characteristics()['D']).split('\nName')[0].split(" ")[-1]
        print(f'Размер выборки: {self.size}, Математическое ожидание: {self.mu}, Дисперсия: {self.sigma}')
        print(f'Мера расхождения теоретической и эмпирической функции распределения: {D}')
        print(f'Мера расхождения теоретической плотности распределения и гистограммы: {self.max_loss}')
        #self.check_distr()
        print('Статистические характеристики:')

        display(self.statistical_characteristics())

        print('-------------------------------------------------------------------------------------------------------')
        print('Характеристики для гистограммы')

        display(table)
        self.hist()
```

In [3]:

```
from scipy.stats import norm
def inverse_normal_cdf(p, mu=0, sigma=1, tolerance=0.00001):
    """Вычисляет обратную функцию распределения нормального распределения"""
    if mu != 0 or sigma != 1:
        return mu + sigma * inverse_normal_cdf(p, tolerance=tolerance)

    low_z = -10.0
    hi_z = 10.0
    while hi_z - low_z > tolerance:
        mid_z = (low_z + hi_z) / 2
        mid_p = norm.cdf(mid_z)
        if mid_p < p:
            low_z = mid_z
        else:
            hi_z = mid_z

    return mid_z
```