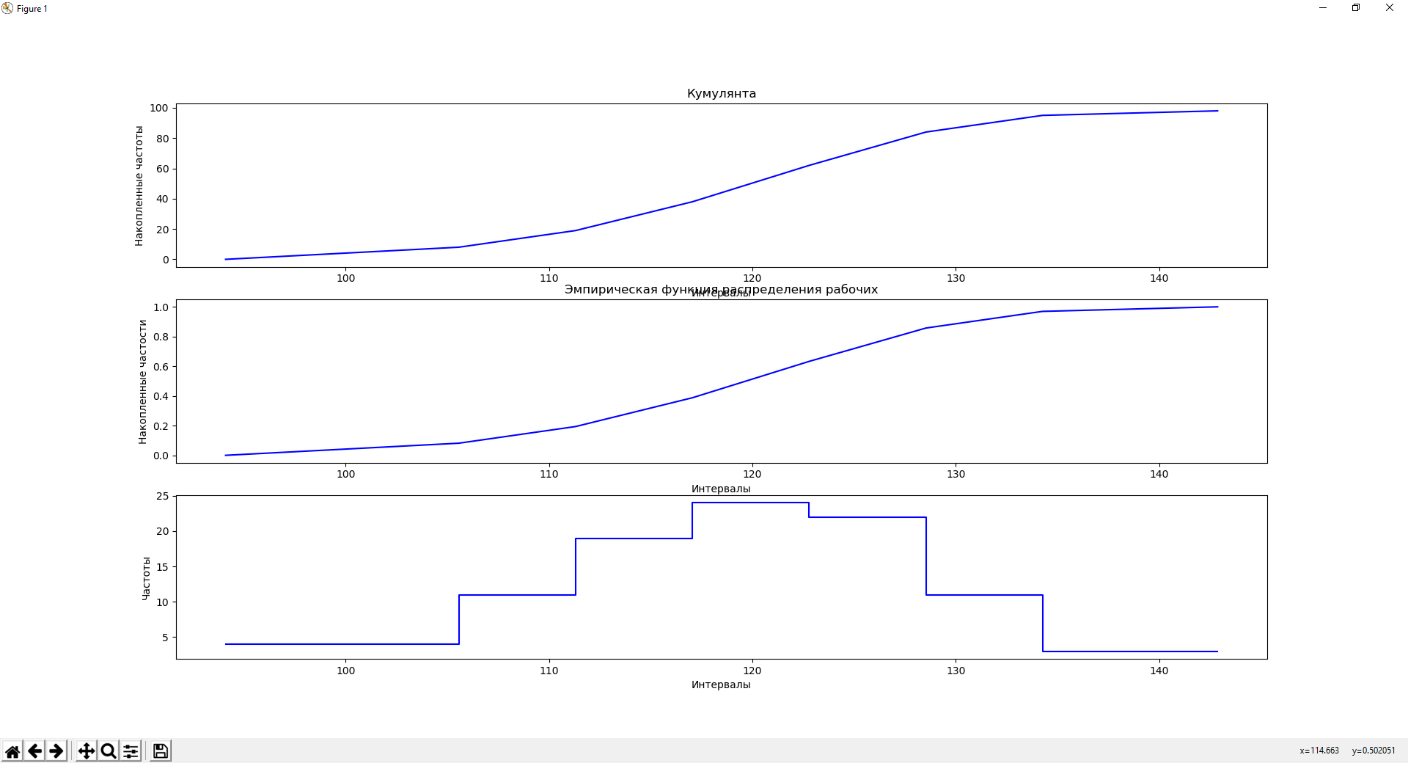
**Лабораторная работа №2**

# Задача №1

Постановка задачи: имеются данные о распределении 100 рабочих цеха по выработке в отчетном году (в процентах к предыдущему году). Всего n=100 значений.

Результаты:

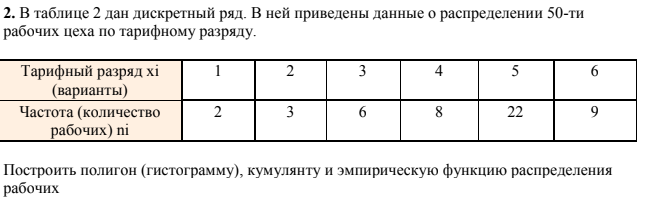


Код программы:

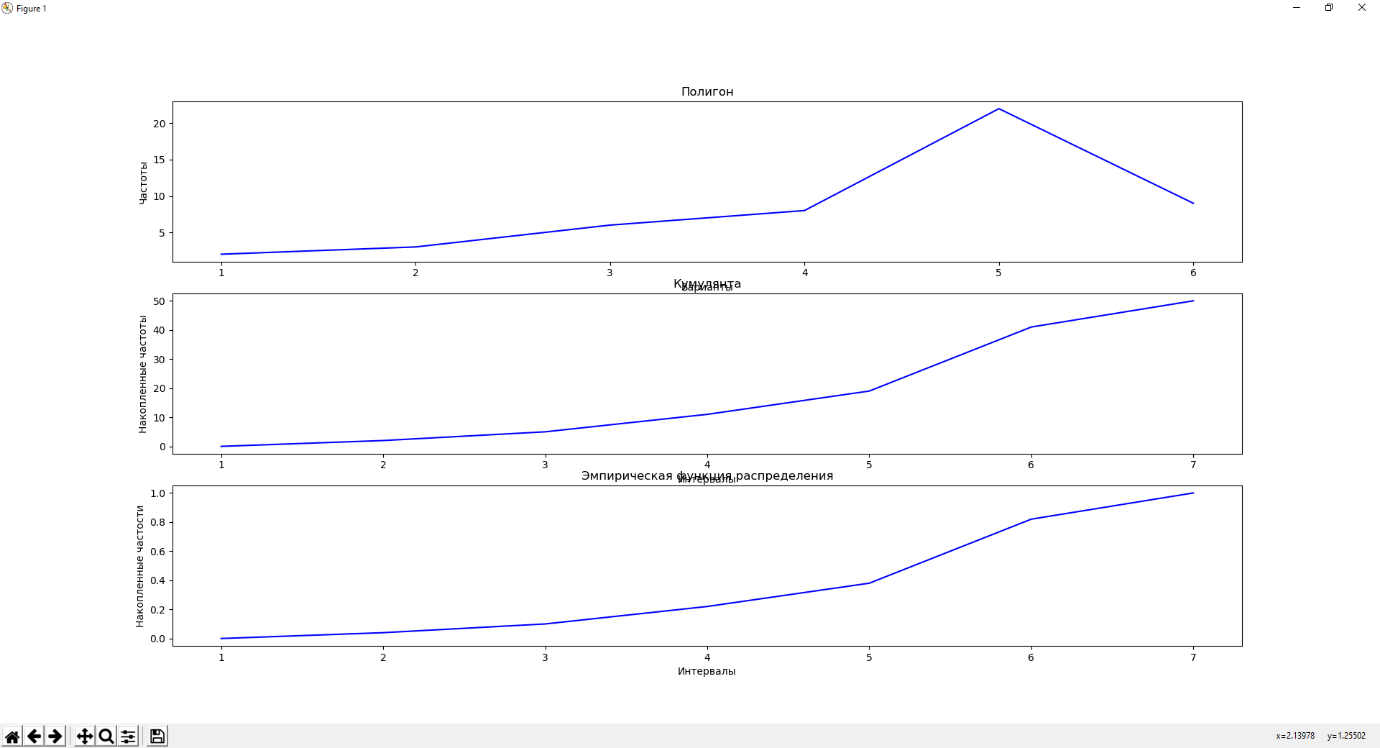
from varseries import ContinuousVS, DiscreteVS  
  
  
def task1():  
 values = [  
 103.4, 115.2, 127, 131, 114, 114.1, 119.6, 125.5, 116.9, 118.1, 123.5,  
 113.5, 112.3, 123, 125, 129.9, 99.2, 111, 122, 134, 107.1, 117, 117.5,  
 118.5, 124, 127.8, 108, 119.5, 123, 126.1, 100.1, 120.2, 122.2, 124.8,  
 109, 113, 122.5, 135.8, 97, 121.1, 123.8, 123.2, 105.9, 122.6, 123.9,  
 129.5, 107, 123.5, 128.5, 117.5, 121.5, 127.5, 113.2, 120.6, 126.5,  
 116, 122.9, 138, 115, 123.1, 140, 94.1, 110, 112.9, 132, 102, 109.5,  
 118.3, 135, 112.5, 115.5, 120, 126, 130, 105.5, 108.2, 119.2, 131.4,  
 106.5, 112, 120.8, 121.9, 134.2, 115.7, 118.9, 124.5, 111.5, 121, 133,  
 116.5, 119, 129, 106.1, 119.8, 133.6, 114.5, 118, 128  
 ]  
 v = ContinuousVS(values)  
 v.draw\_cumulate()\  
 .draw\_empiric\_dist\_func()\  
 .draw\_hist()

# Задача №2

Постановка задачи:



Результаты:

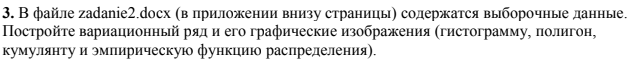


Код программы:

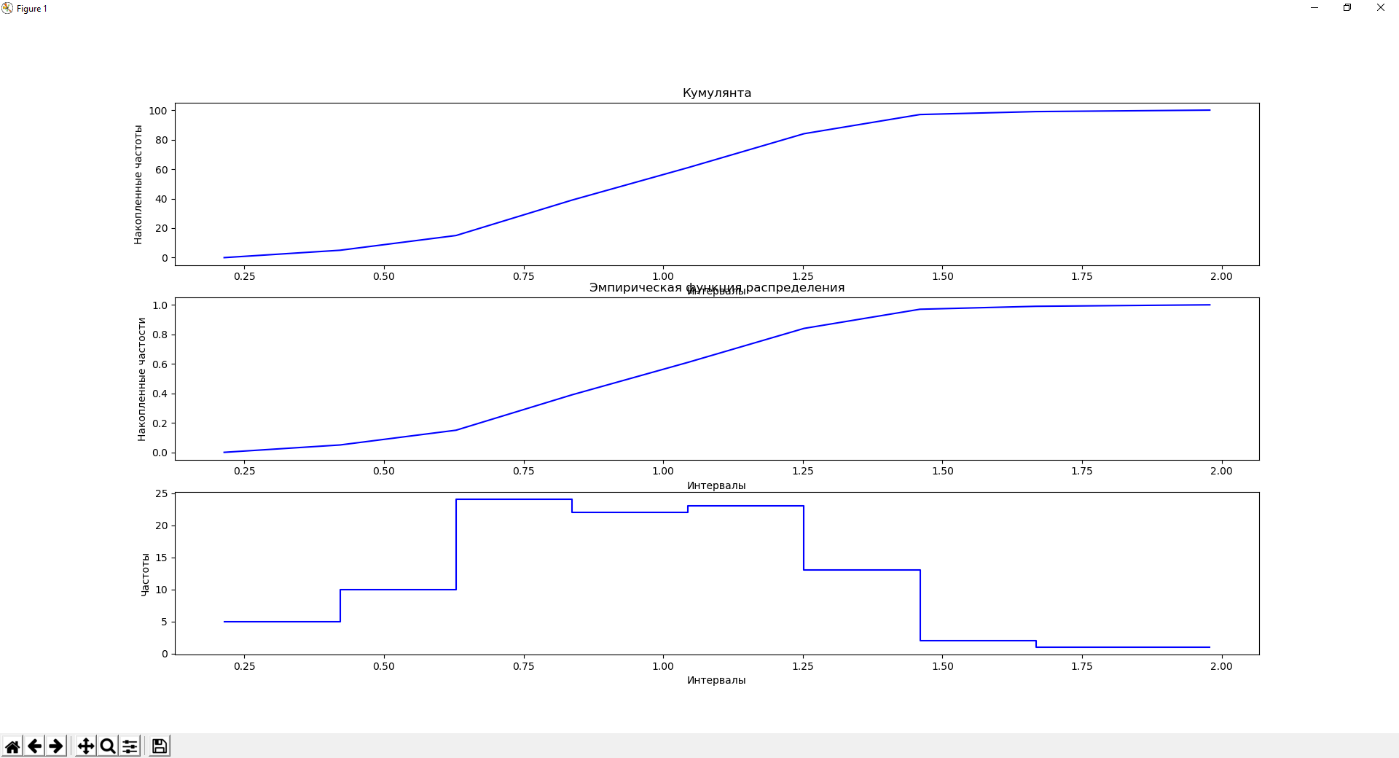
def task2():  
 values = {  
 1: 2, 2: 3, 3: 6, 4: 8, 5: 22, 6: 9  
 }  
 v = DiscreteVS(values)  
 v.draw\_polygon()\  
 .draw\_cumulate()\  
 .draw\_empiric\_dist\_func()

# Задача №3

Постановка задачи:



Результаты:

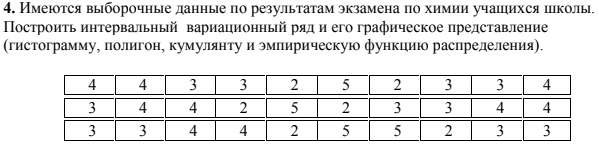


Код программы:

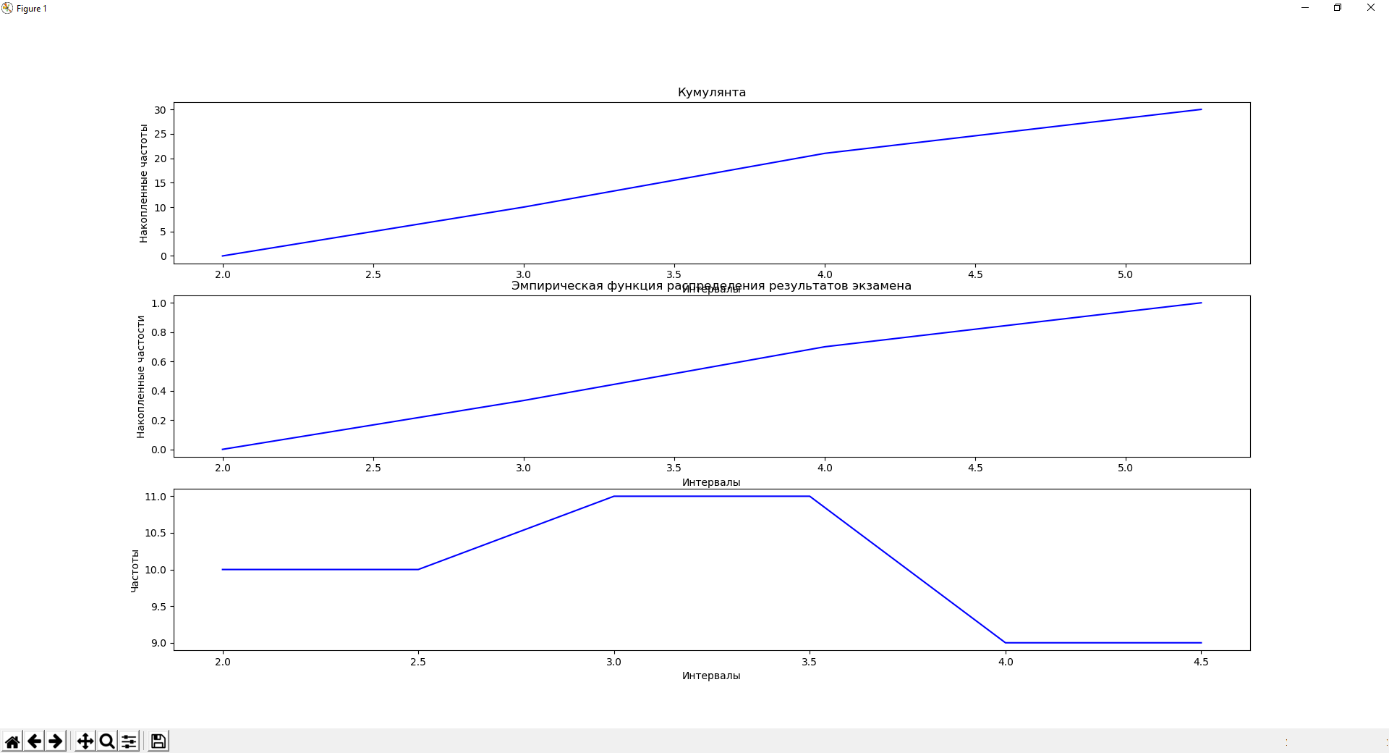
def task3():  
 values = [  
 1.14285583, 0.21398374, 1.25641624, 0.67329946, 1.21496283, 0.99101069,  
 1.39925669, 0.61109646, 0.85890088, 0.78632108, 0.9738463, 1.3846759,  
 0.49488379, 1.0979642, 1.02453946, 1.06382694, 0.78161594, 1.20567321,  
 1.38270281, 0.88719158, 0.75776634, 1.16915277, 1.23004829, 0.71265086,  
 1.02887585, 0.82302015, 1.24597822, 1.45686546, 0.91103144, 0.77406981,  
 1.09453619, 0.79865011, 0.88126134, 1.10711803, 1.00136848, 0.92217984,  
 1.24560914, 0.78720264, 0.954333, 0.99578226, 0.81526016, 0.77680747,  
 1.23527671, 1.73649997, 1.25015887, 0.71522997, 0.76771727, 1.0515177,  
 0.53930926, 1.32623785, 0.59025817, 0.84943463, 1.0391314, 0.87918459,  
 0.60738125, 1.18346139, 0.83580503, 0.95130778, 1.40929416, 0.60987357,  
 1.39038211, 1.06430415, 0.6048676, 1.36443751, 0.98420392, 1.31749231,  
 1.10304182, 0.25832193, 0.31529515, 0.43993342, 0.90625883, 1.49160615,  
 0.66502074, 0.3382135, 0.5468639, 0.66566206, 1.22896107, 1.32777678,  
 1.21582933, 1.00298477, 0.8827651, 1.07884146, 1.45221163, 0.63185447,  
 0.9416058, 1.07515286, 0.97412237, 1.51354811, 1.12753343, 0.72361969,  
 0.50409524, 0.68639066, 0.82355366, 0.69646316, 0.65239474, 0.72192621,  
 1.05932474, 1.25494818, 1.87487639, 0.74979352  
 ]  
 VariationSeries.PRECISION = 8  
  
 v = ContinuousVS(values)  
 v.draw\_cumulate(3, 1, 1)  
 v.draw\_empiric\_dist\_func(3, 1, 2)  
 v.draw\_hist(3, 1, 3)  
  
 ContinuousVS.show()def task3():  
 values = [  
 1.14285583, 0.21398374, 1.25641624, 0.67329946, 1.21496283, 0.99101069,  
 1.39925669, 0.61109646, 0.85890088, 0.78632108, 0.9738463, 1.3846759,  
 0.49488379, 1.0979642, 1.02453946, 1.06382694, 0.78161594, 1.20567321,  
 1.38270281, 0.88719158, 0.75776634, 1.16915277, 1.23004829, 0.71265086,  
 1.02887585, 0.82302015, 1.24597822, 1.45686546, 0.91103144, 0.77406981,  
 1.09453619, 0.79865011, 0.88126134, 1.10711803, 1.00136848, 0.92217984,  
 1.24560914, 0.78720264, 0.954333, 0.99578226, 0.81526016, 0.77680747,  
 1.23527671, 1.73649997, 1.25015887, 0.71522997, 0.76771727, 1.0515177,  
 0.53930926, 1.32623785, 0.59025817, 0.84943463, 1.0391314, 0.87918459,  
 0.60738125, 1.18346139, 0.83580503, 0.95130778, 1.40929416, 0.60987357,  
 1.39038211, 1.06430415, 0.6048676, 1.36443751, 0.98420392, 1.31749231,  
 1.10304182, 0.25832193, 0.31529515, 0.43993342, 0.90625883, 1.49160615,  
 0.66502074, 0.3382135, 0.5468639, 0.66566206, 1.22896107, 1.32777678,  
 1.21582933, 1.00298477, 0.8827651, 1.07884146, 1.45221163, 0.63185447,  
 0.9416058, 1.07515286, 0.97412237, 1.51354811, 1.12753343, 0.72361969,  
 0.50409524, 0.68639066, 0.82355366, 0.69646316, 0.65239474, 0.72192621,  
 1.05932474, 1.25494818, 1.87487639, 0.74979352  
 ]  
 v = ContinuousVS(values)  
 v.draw\_cumulate()\  
 .draw\_empiric\_dist\_func()\  
 .draw\_hist()

# Задача №4

Постановка задачи:



Результаты:

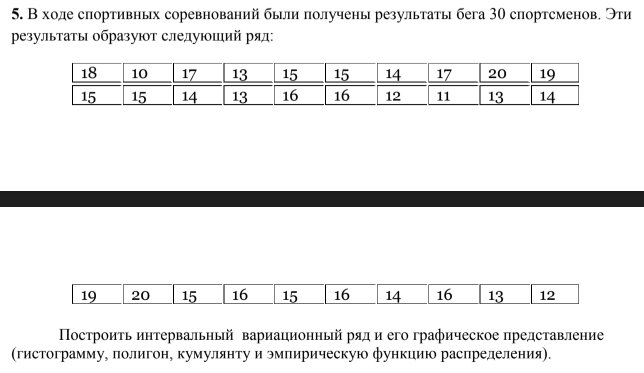


Код программы:

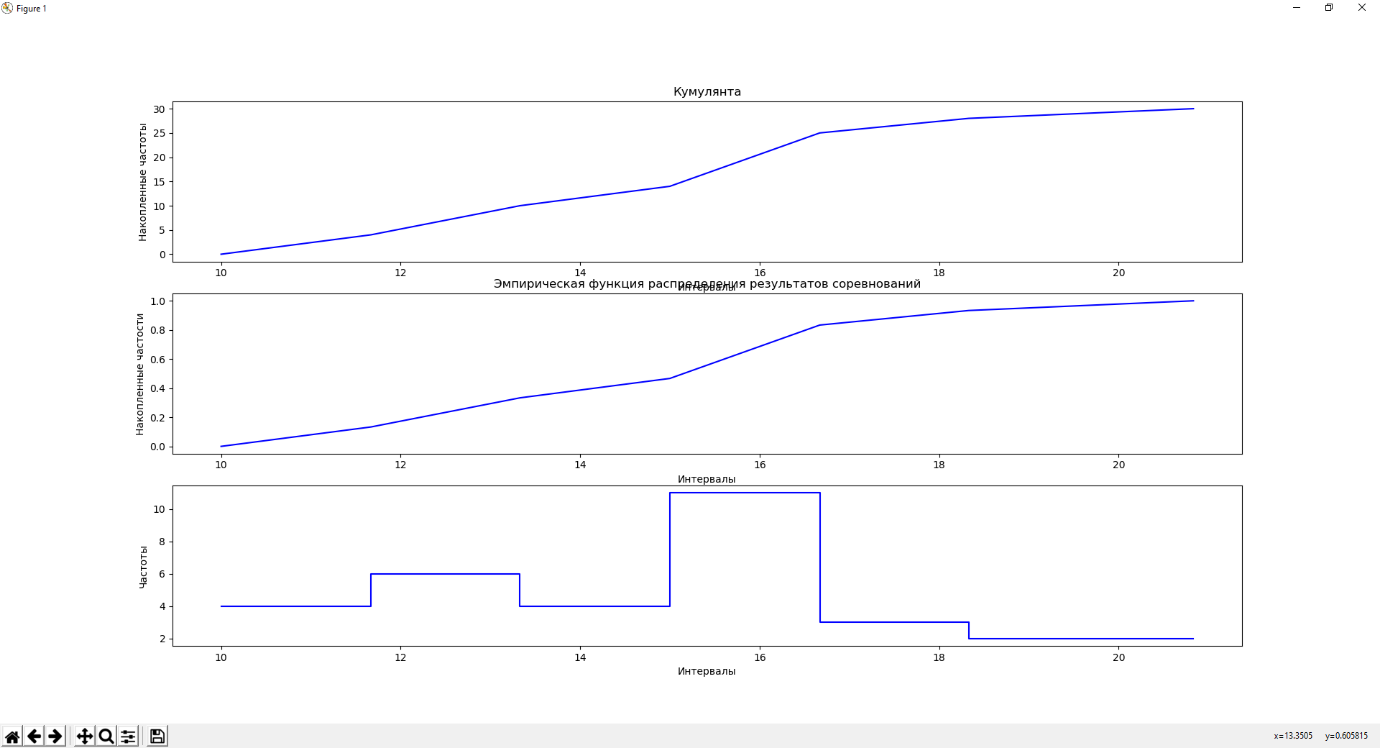
def task4():  
 values = [  
 4, 4, 3, 3, 2, 5, 2, 3, 3, 4,  
 3, 4, 4, 2, 5, 2, 3, 3, 4, 4,  
 3, 3, 4, 4, 2, 5, 5, 2, 3, 3  
 ]  
 v = ContinuousVS(values)  
 v.draw\_cumulate()\  
 .draw\_empiric\_dist\_func()\  
 .draw\_hist()

# Задача №5

Постановка задачи:



Результаты:



Код программы:

def task5():  
 values = [  
 18, 10, 17, 13, 15, 15, 14, 17, 20, 19,  
 15, 15, 14, 13, 16, 16, 12, 11, 13, 14,  
 19, 20, 15, 16, 15, 16, 14, 16, 13, 12  
 ]  
 v = ContinuousVS(values)  
 v.draw\_cumulate()\  
 .draw\_empiric\_dist\_func()\  
 .draw\_hist()

**Deep dark back end**

import collections

import math

import abc

import plotly.graph\_objects as go

# Common variation series calculations

class VariationSeries:

    """Abstacrt"""

    def \_\_init\_\_(self, values: [dict, list], precision=5):

        self.precision = precision

        # User passes list of raw values

        if isinstance(values, list):

            self.vals = sorted(values)

            self.vs = self.build\_vs\_from\_list(values)

        # User passes dict with a ready variation series

        elif isinstance(values, dict):

            self.vs = self.build\_vs\_from\_dict(values)

            self.vals = []

            for x, n in self.vs.items():

                self.vals.extend([x]\*n)

    def draw\_cumulate(

            self, title='Кумулянта (заголовок)', name='Кумулянта (легенда)',

            x\_label='Варианты', y\_label='Накопленные частоты'

        ):

        """Draw cumulate: variants..closest\_x => accumulated\_frequencies"""

        x = self.variants + [self.closest\_x]

        y = self.acc\_frequencies

        VariationSeries.\_prepare\_figure(x, y, title, name, x\_label, y\_label).show()

        return self

    def draw\_empiric\_dist\_func(

            self, title='Эмпирическая функция распределения (заголовок)',

            name='Эмпирическая функция распределения (легенда)',

            x\_label='Варианты', y\_label='Частости'

        ):

        """Draw empiric distribution function: variants..closest\_x => accumulated\_relative\_frequencies"""

        x = self.variants + [self.closest\_x]

        y = self.acc\_rel\_frequencies

        VariationSeries.\_prepare\_figure(x, y, title, name, x\_label, y\_label).show()

        return self

    @property

    def n(self) -> int:

        """Number of elements in the variation series"""

        return len(self.vals)

    @property

    def x\_max(self) -> [float, int]:

        if isinstance(self.vals[0], tuple):

            return self.vals[-1][1]

        return max(self.vals)

    @property

    def x\_min(self) -> [float, int]:

        if isinstance(self.vals[0], tuple):

            return self.vals[0][0]

        return min(self.vals)

    # Different for discrete and continuous vs

    @property

    @abc.abstractmethod

    def variants(self) -> list:

        """Variants x"""

        pass

    @property

    def acc\_frequencies(self) -> list:

        """Accumulated frequencies m(x(i))"""

        def gen\_acc\_frequencies(values):

            """Generator of accumulated sum"""

            accumulated\_sum = 0

            for m\_i in values:

                yield accumulated\_sum

                accumulated\_sum += m\_i

            yield accumulated\_sum

        return list(gen\_acc\_frequencies(self.var\_frequencies))

    @property

    def acc\_rel\_frequencies(self) -> list:

        """Accumulated relative frequencies w(x)"""

        n = self.n

        p = self.precision

        return [round(m\_x / n, p) for m\_x in self.acc\_frequencies]

    @property

    def var\_frequencies(self) -> list:

        """Variants frequencies m(i)"""

        return list(self.vs.values())

    @property

    def median(self) -> [int, float]:

        """Return median of the variation series"""

        n = self.n

        idx = n // 2

        if n % 2 == 0:

            idx -= 1

        return self.vals[idx]

    @property

    def mode(self) -> [int, float]:

        """Return mode of the variation series"""

        items = list(self.vs.items())

        mode\_val, mode\_reps = items[0]

        for val, reps in items:

            if mode\_reps < reps:

                mode\_val = val

                mode\_reps = reps

        return mode\_val

    @property

    @abc.abstractmethod

    def closest\_x(self) -> [int, float]:

        """Return closests x value for accumulated functions"""

        pass

    @abc.abstractmethod

    def build\_vs\_from\_list(self, values: list) -> 'collections.OrderedDict':

        """Build variation series from raw values"""

        pass

    @staticmethod

    def build\_vs\_from\_dict(values: dict) -> 'collections.OrderedDict':

        """Build variation series from dict"""

        # User shoud be consient about values he passes

        vs = collections.OrderedDict(sorted(values.items()))

        return vs

    @staticmethod

    def \_prepare\_figure(x, y, title, name, x\_label, y\_label) -> 'go.Figure':

        fig = go.Figure()

        scatter = go.Scatter(x=x, y=y, mode='lines', name=name)

        fig.add\_trace(scatter)

        fig.update\_layout(title=title, xaxis\_title=x\_label, yaxis\_title=y\_label)

        return fig

# Discrete variation series

class DiscreteVS(VariationSeries):

    def draw\_polygon(

        self, title='Полигон (заголовок)', name='Полигон (легенда)',

        x\_label='Варианты', y\_label='Частоты'

        ):

        """Draw polygon: variants => variants frequencies"""

        x = self.variants

        y = self.var\_frequencies

        VariationSeries.\_prepare\_figure(x, y, title, name, x\_label, y\_label).show()

        return self

    @property

    def variants(self) -> list:

        return list(self.vs.keys())

    @property

    def closest\_x(self) -> [float, int]:

        """The closests x for accumulated values"""

        variants = self.variants

        penult, last = variants[-2:]

        diff = last - penult

        return last + diff

    def build\_vs\_from\_list(self, values: list) -> 'collections.OrderedDict':

        vs = collections.OrderedDict()

        for val in sorted(values):

            vs[val] = vs.get(val, 0) + 1

        return vs

# Continuous variation series

class ContinuousVS(VariationSeries):

    def draw\_hist(self, title='Гистограмма'):

        x = self.vals

        if isinstance(x[0], tuple):

            x = [x[0][0]] + [right for \_, right in x]

        xbins = {

            'start': self.x\_min, 'end': self.x\_max,

            'size': self.delta,

        }

        trace = go.Histogram(x=x, xbins=xbins, autobinx=False)

        fig = go.Figure()

        fig.add\_trace(trace)

        fig.show()

        return self

    @property

    def k(self) -> int:

        """Number of intervals"""

        return math.ceil(1 + 1.4\*math.log(self.n))

    @property

    def delta(self) -> [float, int]:

        """Interval length"""

        return (self.x\_max - self.x\_min) / self.k

    @property

    def intervals(self) -> list:

        return list(self.vs.keys())

    @property

    def variants(self) -> list:

        return [left for left, right in self.intervals]

    @property

    def closest\_x(self) -> [int, float]:

        return self.intervals[-1][1]

    def build\_vs\_from\_list(self, values: list) -> 'collections.OrderedDict':

        """Build variation series from raw values"""

        # There are raw values

        p = self.precision

        delta = self.delta

        x\_min = self.x\_min

        x\_max = self.x\_max + delta/2

        intervals = []

        for i in range(self.k):

            bias = i \* delta

            left = round(x\_min + bias, p)

            right = round(x\_min + bias + delta, p)

            intervals += [(left, right)]

        # Move the last interval for capturing

        last\_interval = intervals[-1]

        new\_last\_interval = (last\_interval[0], x\_max)

        intervals[-1] = new\_last\_interval

        vs = collections.OrderedDict()  # Frequncies

        for val in sorted(self.vals):

            # Choose the correct interval

            for interval in intervals:

                left, right = interval

                if left <= val < right or val == x\_max:

                    vs[interval] = vs.get(interval, 0) + 1

                    break

        return vs