Домашнее задание по теории вероятностей и математической статистике 4. Горелкина РК6-32Б. Вариант 4.

Задание 1.

Импортируем все необходимые для расчетов модули.

In [1]:

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import math as math
```

1.1 Построим генератор случайных чисел на основании исходных данных из первого ДЗ.

In [2]:

```
a = 11
c = 6
x0 = 8
m = 100

gen_f = [x0]
i = 0
new = 0

while True:
    new = (a * gen_f[i] + c) % m
    if new == gen_f[0]:
        break
    gen_f.append(new)
    i += 1

for j in range(i+1):
    print('x', j, ' = ', gen_f[j], sep = '')

print('Период генератора равен n =', i+1)
```

```
x0 = 8
x1 = 94
x2 = 40
x3 = 46
x4 = 12
x5 = 38
x6 = 24
x7 = 70
x8 = 76
x9 = 42
x10 = 68
x11 = 54
x12 = 0
x13 = 6
x14 = 72
x15 = 98
x16 = 84
x17 = 30
x18 = 36
x19 = 2
x20 = 28
x21 = 14
x22 = 60
x23 = 66
x24 = 32
x25 = 58
x26 = 44
x27 = 90
x28 = 96
x29 = 62
x30 = 88
x31 = 74
x32 = 20
x33 = 26
x34 = 92
x35 = 18
x36 = 4
x37 = 50
x38 = 56
x39 = 22
x40 = 48
x41 = 34
x42 = 80
x43 = 86
x44 = 52
x45 = 78
x46 = 64
x47 = 10
x48 = 16
x49 = 82
Период генератора равен n = 50
```

1.2 Теперь построим генератор с подобранными параметрами для обеспечения максимального периода m = 100. И убедимся, что период действительно максимальный.

In [3]:

```
a = 41
c = 53
x0 = 8
m = 100

gen = [x0]
i = 0
new = 0

while True:
    new = (a * gen[i] + c) % m
    if new == gen[0]:
        break
    gen.append(new)
    i += 1

for j in range(i+1):
    print('x', j, ' = ', gen[j], sep = '')

print('Период генератора равен n =', i+1)
```

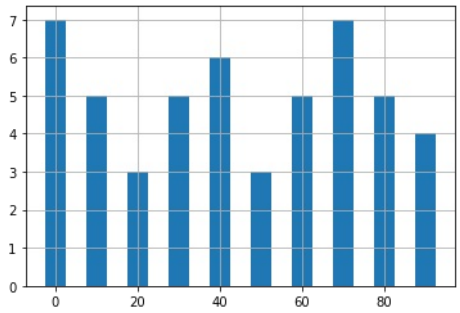
```
x0 = 8
x1 = 81
x2 = 74
x3 = 87
x4 = 20
x5 = 73
x6 = 46
x7 = 39
x8 = 52
x9 = 85
x10 = 38
x11 = 11
x12 = 4
x13 = 17
x14 = 50
x15 = 3
x16 = 76
x17 = 69
x18 = 82
x19 = 15
x20 = 68
x21 = 41
x22 = 34
x23 = 47
x24 = 80
x25 = 33
x26 = 6
x27 = 99
x28 = 12
x29 = 45
x30 = 98
x31 = 71
x32 = 64
x33 = 77
x34 = 10
x35 = 63
x36 = 36
x37 = 29
x38 = 42
x39 = 75
x40 = 28
x41 = 1
x42 = 94
x43 = 7
x44 = 40
x45 = 93
x46 = 66
x47 = 59
x48 = 72
x49 = 5
x50 = 58
x51 = 31
x52 = 24
x53 = 37
x54 = 70
x55 = 23
x56 = 96
x57 = 89
x58 = 2
x59 = 35
x60 = 88
x61 = 61
x62 = 54
x63 = 67
x64 = 0
x65 = 53
x66 = 26
x67 = 19
x68 = 32
x69 = 65
x70 = 18
x71 = 91
x72 = 84
```

```
x73 = 97
x74 = 30
x75 = 83
x76 = 56
x77 = 49
x78 = 62
x79 = 95
x80 = 48
x81 = 21
x82 = 14
x83 = 27
x84 = 60
x85 = 13
x86 = 86
x87 = 79
x88 = 92
x89 = 25
x90 = 78
x91 = 51
x92 = 44
x93 = 57
x94 = 90
x95 = 43
x96 = 16
x97 = 9
x98 = 22
x99 = 55
Период генератора равен n = 100
```

1.3 Построим гисторграмму распределения случайных чисел.

In [4]:

```
list1 = [0 for i in range(10)]
x = [10*i for i in range(10)]
for i in range(50):
    list1[gen[i] // 10] += 1
fig = plt.figure()
plt.bar(x, list1, 5)
plt.grid(True)
```



1.4 Определим коэффициент согласия согласно критерию Пирсона.

In [5]:

```
hi2 = 0
for i in range(10):
    hi2 += (list1[i]  - 5) ** 2
hi2 /= 5
print('Коэффициент согласия равен: ', hi2)
```

```
Коэффициент согласия равен:  3.6
```

1.5 Согласно табличным значениям уровень значимости равен 0.9.

1.6 Матожидание равномерного дискретного распределения рассчитаем как (a+b)/2, а дисперсию ((b-a+1)^2 - 1)/12

In [6]:

```
M = [49.5 for i in range(4)]
D = [833.25 for i in range(4)]
n = [5, 10, 25, 50]
sum1 = s_n = [0, 0, 0, 0]
for i in n:
    ind = int(2 * (i ** (0.5) // 5) +  abs(i % 2 - 1))
    for j in range(i):
        sum1[ind] += gen[j]
med = [sum1[i] / n[i] for i in range(4)]

for i in n:
    ind = int(2 * (i ** (0.5) // 5) +  abs(i % 2 - 1))
    for j in range(i):
        s_n[ind] += (gen[j] - med[ind])**2

S = [s_n[i] / n[i] for i in range(4)]
sigma = [s_n[i] / (n[i] - 1) for i in range(4)]
delta1 = [abs(M[i] - med[i]) for i in range(4)]
delta2 = [abs(D[i] - sigma[i]) for i in range(4)]
delta3 = [abs(D[i] - S[i]) for i in range(4)]
```

Сформируем таблицу.

In [7]:

```
df_ = pd.DataFrame({"n" : n,
             "Матожидание" : M,
             "Дисперсия" : D,
             "Выборочное среднее" : med,
             "Несмещенная дисперсия" : S,
             "Смещенная дисперсия" : sigma,
             "|M - x_cp.|": delta1,
             "|D - sigma^2|": delta3,
             "|D - S^2|": delta2})
df_.style.hide_index()
```

Out[7]:

n	Матожидание	Дисперсия	Выборочное среднее	Несмещенная дисперсия	Смещенная дисперсия	|M - x_cp.|	|D - sigma^2|	|D - S^2|
5	49.500000	833.250000	54.000000	1152.000000	1440.000000	4.500000	318.750000	606.750000
10	49.500000	833.250000	56.500000	760.750000	845.277778	7.000000	72.500000	12.027778
25	49.500000	833.250000	48.000000	820.000000	854.166667	1.500000	13.250000	20.916667
50	49.500000	833.250000	48.500000	920.750000	939.540816	1.000000	87.500000	106.290816

Задание 2.

2.1 В список dataset запишем значения, полученные в результате 100 экспериментов.

In [8]:

```
rnd = np.array([i+1 for i in range(100)])
dataset=[[0.780, 0.791, 0.776, 0.809, 0.797, 0.819, 0.777, 0.832, 0.795, 0.807,
          0.834, 0.830, 0.784, 0.796, 0.809, 0.810, 0.804, 0.817, 0.796, 0.802,
          0.775, 0.800, 0.806, 0.828, 0.797, 0.797, 0.789, 0.780, 0.739, 0.791,
          0.809, 0.825, 0.784, 0.845, 0.802, 0.816, 0.811, 0.781, 0.814, 0.781,
          0.801, 0.803, 0.808, 0.795, 0.820, 0.838, 0.791, 0.786, 0.819, 0.813,
          0.778, 0.807, 0.826, 0.781, 0.808, 0.813, 0.803, 0.779, 0.793, 0.790,
          0.801, 0.839, 0.786, 0.797, 0.786, 0.823, 0.790, 0.808, 0.831, 0.778,
          0.781, 0.808, 0.776, 0.818, 0.821, 0.809, 0.809, 0.793, 0.824, 0.825,
          0.806, 0.823, 0.785, 0.797, 0.783, 0.797, 0.777, 0.791, 0.775, 0.801,
          0.777, 0.817, 0.759, 0.813, 0.809, 0.828, 0.782, 0.823, 0.819, 0.809],
         [0.538, 0.549, 0.557, 0.536, 0.533, 0.569, 0.490, 0.559, 0.513, 0.518,
          0.640, 0.587, 0.483, 0.513, 0.530, 0.540, 0.567, 0.548, 0.575, 0.522,
          0.487, 0.580, 0.597, 0.580, 0.608, 0.542, 0.531, 0.523, 0.487, 0.524,
          0.586, 0.525, 0.471, 0.628, 0.566, 0.555, 0.575, 0.541, 0.518, 0.513,
          0.537, 0.534, 0.559, 0.493, 0.587, 0.595, 0.466, 0.563, 0.607, 0.566,
          0.504, 0.615, 0.552, 0.455, 0.548, 0.563, 0.530, 0.528, 0.522, 0.573,
          0.521, 0.560, 0.555, 0.564, 0.527, 0.541, 0.525, 0.586, 0.600, 0.481,
          0.544, 0.543, 0.534, 0.546, 0.551, 0.614, 0.558, 0.488, 0.544, 0.554,
          0.506, 0.541, 0.539, 0.560, 0.524, 0.561, 0.528, 0.501, 0.567, 0.521,
          0.493, 0.549, 0.489, 0.549, 0.589, 0.558, 0.553, 0.626, 0.557, 0.525],
         [0.171, 0.233, 0.167, 0.157, 0.170, 0.276, 0.151, 0.271, 0.182, 0.223,
          0.377, 0.300, 0.184, 0.160, 0.179, 0.244, 0.231, 0.243, 0.159, 0.178,
          0.138, 0.235, 0.206, 0.344, 0.251, 0.220, 0.160, 0.210, 0.104, 0.187,
          0.243, 0.242, 0.158, 0.396, 0.180, 0.272, 0.187, 0.159, 0.208, 0.138,
          0.206, 0.147, 0.219, 0.191, 0.401, 0.237, 0.153, 0.292, 0.263, 0.275,
          0.174, 0.236, 0.207, 0.130, 0.259, 0.240, 0.214, 0.167, 0.166, 0.241,
          0.167, 0.204, 0.201, 0.233, 0.161, 0.225, 0.188, 0.269, 0.367, 0.136,
          0.176, 0.260, 0.164, 0.167, 0.219, 0.252, 0.199, 0.169, 0.234, 0.230,
          0.152, 0.199, 0.132, 0.245, 0.123, 0.203, 0.145, 0.195, 0.194, 0.224,
          0.135, 0.239, 0.143, 0.167, 0.213, 0.272, 0.202, 0.364, 0.182, 0.181],
         [0.039, 0.073, 0.066, 0.045, 0.041, 0.074, 0.030, 0.105, 0.054, 0.086,
          0.184, 0.117, 0.051, 0.043, 0.043, 0.070, 0.094, 0.093, 0.047, 0.066,
          0.033, 0.083, 0.061, 0.151, 0.088, 0.082, 0.037, 0.061, 0.028, 0.049,
          0.068, 0.076, 0.035, 0.182, 0.057, 0.073, 0.054, 0.048, 0.058, 0.037,
          0.046, 0.027, 0.090, 0.049, 0.213, 0.078, 0.021, 0.106, 0.108, 0.087,
          0.031, 0.069, 0.030, 0.025, 0.107, 0.074, 0.064, 0.076, 0.048, 0.076,
          0.035, 0.061, 0.061, 0.059, 0.044, 0.077, 0.073, 0.113, 0.146, 0.045,
          0.051, 0.075, 0.053, 0.058, 0.065, 0.107, 0.046, 0.031, 0.070, 0.046,
          0.037, 0.061, 0.024, 0.097, 0.035, 0.041, 0.024, 0.038, 0.057, 0.078,
          0.025, 0.060, 0.027, 0.035, 0.050, 0.090, 0.066, 0.108, 0.044, 0.045]]
```

Сформируем таблицу с результатами.

In [9]:

```
pd.options.display.max_rows = 100
df = pd.DataFrame({"rnd" : rnd,
             "UTIL 1" : dataset[0],
             "UTIL 2" : dataset[1],
             "AVE.CONT 1" : dataset[2],
             "AVE.CONT 2" : dataset[3]})
df.style.hide_index()
```

Out[9]:

rnd	UTIL 1	UTIL 2	AVE.CONT 1	AVE.CONT 2
1	0.780000	0.538000	0.171000	0.039000
2	0.791000	0.549000	0.233000	0.073000
3	0.776000	0.557000	0.167000	0.066000
4	0.809000	0.536000	0.157000	0.045000

5	0.797000	0.533000	0.170000	0.041000
6	0.819000	0.569000	0.276000	0.074000
7	0.777000	0.490000	0.151000	0.030000
8	0.832000	0.559000	0.271000	0.105000
9	0.795000	0.513000	0.182000	0.054000
10	0.807000	0.518000	0.223000	0.086000
11	0.834000	0.640000	0.377000	0.184000
12	0.830000	0.587000	0.300000	0.117000
13	0.784000	0.483000	0.184000	0.051000
14	0.796000	0.513000	0.160000	0.043000
15	0.809000	0.530000	0.179000	0.043000
16	0.810000	0.540000	0.244000	0.070000
17	0.804000	0.567000	0.231000	0.094000
18	0.817000	0.548000	0.243000	0.093000
19	0.796000	0.575000	0.159000	0.047000
20	0.802000	0.522000	0.178000	0.066000
21	0.775000	0.487000	0.138000	0.033000
22	0.800000	0.580000	0.235000	0.083000
23	0.806000	0.597000	0.206000	0.061000
24	0.828000	0.580000	0.344000	0.151000
25	0.797000	0.608000	0.251000	0.088000
26	0.797000	0.542000	0.220000	0.082000
27	0.789000	0.531000	0.160000	0.037000
28	0.780000	0.523000	0.210000	0.061000
29	0.739000	0.487000	0.104000	0.028000
30	0.791000	0.524000	0.187000	0.049000
31	0.809000	0.586000	0.243000	0.068000
32	0.825000	0.525000	0.242000	0.076000
33	0.784000	0.471000	0.158000	0.035000
34	0.845000	0.628000	0.396000	0.182000
35	0.802000	0.566000	0.180000	0.057000
36	0.816000	0.555000	0.272000	0.073000
37	0.811000	0.575000	0.187000	0.054000
38	0.781000	0.541000	0.159000	0.048000
39	0.814000	0.518000	0.208000	0.058000
40	0.781000	0.513000	0.138000	0.037000
41	0.801000	0.537000	0.206000	0.046000
42	0.803000	0.534000	0.147000	0.027000
43	0.808000	0.559000	0.219000	0.090000
44	0.795000	0.493000	0.191000	0.049000
45	0.820000	0.587000	0.401000	0.213000
46	0.838000	0.595000	0.237000	0.078000
47	0.791000	0.466000	0.153000	0.021000
48	0.786000	0.563000	0.292000	0.106000
49	0.819000	0.607000	0.263000	0.108000
50	0.813000	0.566000	0.275000	0.087000
51	0.778000	0.504000	0.174000	0.031000
52	0.807000	0.615000	0.236000	0.069000
53	0.826000	0.552000	0.207000	0.030000
54	0.781000	0.455000	0.130000	0.025000
55	0.808000	0.548000	0.259000	0.107000
56	0.813000	0.563000	0.240000	0.074000
57	0.803000	0.530000	0.214000	0.064000
58	0.779000	0.528000	0.167000	0.076000
59	0.793000	0.522000	0.166000	0.048000
60	0.790000	0.573000	0.241000	0.076000
61	0.801000	0.521000	0.167000	0.035000
62	0.839000	0.560000	0.204000	0.061000
63	0.786000	0.555000	0.201000	0.061000
64	0.797000	0.564000	0.233000	0.059000
65	0.786000	0.527000	0.161000	0.044000
66	0.823000	0.541000	0.225000	0.077000
67	0.790000	0.525000	0.188000	0.073000

68	0.808000	0.586000	0.269000	0.113000
69	0.831000	0.600000	0.367000	0.146000
70	0.778000	0.481000	0.136000	0.045000
71	0.781000	0.544000	0.176000	0.051000
72	0.808000	0.543000	0.260000	0.075000
73	0.776000	0.534000	0.164000	0.053000
74	0.818000	0.546000	0.167000	0.058000
75	0.821000	0.551000	0.219000	0.065000
76	0.809000	0.614000	0.252000	0.107000
77	0.809000	0.558000	0.199000	0.046000
78	0.793000	0.488000	0.169000	0.031000
79	0.824000	0.544000	0.234000	0.070000
80	0.825000	0.554000	0.230000	0.046000
81	0.806000	0.506000	0.152000	0.037000
82	0.823000	0.541000	0.199000	0.061000
83	0.785000	0.539000	0.132000	0.024000
84	0.797000	0.560000	0.245000	0.097000
85	0.783000	0.524000	0.123000	0.035000
86	0.797000	0.561000	0.203000	0.041000
87	0.777000	0.528000	0.145000	0.024000
88	0.791000	0.501000	0.195000	0.038000
89	0.775000	0.567000	0.194000	0.057000
90	0.801000	0.521000	0.224000	0.078000
91	0.777000	0.493000	0.135000	0.025000
92	0.817000	0.549000	0.239000	0.060000
93	0.759000	0.489000	0.143000	0.027000
94	0.813000	0.549000	0.167000	0.035000
95	0.809000	0.589000	0.213000	0.050000
96	0.828000	0.558000	0.272000	0.090000
97	0.782000	0.553000	0.202000	0.066000
98	0.823000	0.626000	0.364000	0.108000
99	0.819000	0.557000	0.182000	0.044000
100	0.809000	0.525000	0.181000	0.045000

Рассчитаем выборочные средние и исправленные оценки дисперсии для каждой характеристики и для каждого n.

In [10]:

```
cov = [[0, 0] for i in range(4)]                    #список ковариаций
r = [[0, 0] for i in range(4)]                      #список коэффициентов корреляции
med = [[0 for i in range(4)] for j in range(4)]     #список с выборочными средними
s = [[0 for i in range(4)] for j in range(4)]       #список с исправленными дисперсиями
sigma = [0 for i in range(4)]                       #список с половинными ср.кв.отклонениями для n = 100
n = [10, 25, 50, 100]                               #необходимые значения n согласно условию
for i in range(4):                                  #проходимся по каждой индексам n
    for k in range(4):                              #проходимся по каждой величине
        for j in range(n[i]):                       #проходимся по n значениям из dataset
            med[i][k] += dataset[k][j]
        med[i][k] = med[i][k] / n[i]
        for j in range(n[i]):
            s[i][k] += (dataset[k][j] - med[i][k])**2
        s[i][k] = s[i][k] / (n[i] - 1)
for i in range(4):
    sigma[i] = math.sqrt(s[3][i]) / 2
for i in range(4):
    for j in range(n[i]):
        cov[i][0] += (dataset[0][j] - med[i][0]) * (dataset[1][j] - med[i][1])
        cov[i][1] += (dataset[2][j] - med[i][2]) * (dataset[3][j] - med[i][3])
    cov[i][0] /= (n[i] - 1)
    cov[i][1] /= (n[i] - 1)
for i in range(4):
    r[i][0] = cov[i][0] / math.sqrt(s[i][0] * s[i][1])
    r[i][1] = cov[i][1] / math.sqrt(s[i][2] * s[i][3])
med = np.array(med).transpose()                     #транспонируем матрицу для формирования таблицы,
                                                    #так как каждая строка матрицы соответствует
                                                    #определенному значению n, нам нужно сделать
                                                    #соответствие каждой строки определенной величине
s = np.array(s).transpose()
sigma = np.array(sigma).transpose()
cov = np.array(cov).transpose()
r = np.array(r).transpose()
```

Сформируем необходимые таблицы.

In [11]:

```
df_1 = pd.DataFrame({"n" : n,
             "med UTIL 1" : med[0],
             "med UTIL 2" : med[1],
             "med AVE.CONT 1" : med[2],
             "med AVE.CONT 2" : med[3]})
print('Выборочные средние:')
df_1.style.hide_index()
```

Выборочные средние:

Out[11]:

n	med UTIL 1	med UTIL 2	med AVE.CONT 1	med AVE.CONT 2
10	0.798300	0.536200	0.200100	0.061300
25	0.802840	0.548760	0.217200	0.073480
50	0.802180	0.546220	0.217560	0.072140
100	0.801610	0.544730	0.210430	0.065650

In [12]:

```
df_2 = pd.DataFrame({"n" : n,
             "sigma^2 UTIL 1" : s[0],
             "sigma^2 UTIL 2" : s[1],
             "sigma^2 AVE.CONT 1" : s[2],
             "sigma^2 AVE.CONT 2" : s[3]})
print('Исправленные выборочные оценки дисперсии: ')
df_2.style.hide_index()
```

Исправленные выборочные оценки дисперсии:

Out[12]:

n	sigma^2 UTIL 1	sigma^2 UTIL 2	sigma^2 AVE.CONT 1	sigma^2 AVE.CONT 2
10	0.000345	0.000579	0.002204	0.000561
25	0.000301	0.001498	0.003809	0.001362
50	0.000379	0.001563	0.004395	0.001662
100	0.000357	0.001350	0.003550	0.001217

In [13]:

```
n_100 = [100]
df_3 = pd.DataFrame({"n" : n_100,
             "sigma/2 UTIL 1" : sigma[0],
             "sigma/2 UTIL 2" : sigma[1],
             "sigma/2 AVE.CONT 1" : sigma[2],
             "sigma/2 AVE.CONT 2" : sigma[3]})
print('Половинные средние квадратические отклонения для n = 100: ')
df_3.style.hide_index()
```

Половинные средние квадратические отклонения для n = 100:

Out[13]:

n	sigma/2 UTIL 1	sigma/2 UTIL 2	sigma/2 AVE.CONT 1	sigma/2 AVE.CONT 2
100	0.009446	0.018374	0.029791	0.017444

Чтобы вычислить необходимые диапазоны, определим функцию delta, с помощью которой будем вычислять число диапазонов. Далее в список span запишем значения всех необходимых диапазонов для различных характеристик.

In [14]:

```
def delta(n):
    return math.ceil((max(dataset[n]) - min(dataset[n])) / sigma[n])

span = [[min(dataset[i]) + sigma[i] * j for j in range(delta(i) + 1)] for i in range(4)]  #диапазоны
```

Заполним значения гисторграммы, предварительно отсортировав списки dataset.

In [19]:

```
k = 0
flag = 0
gist = [[0 for i in range(len(span[j])- 1)] for j in range(4)]       #списокк значений гистограммы
for i in range(4):
    dataset[i].sort()
    for j in range(100):
        if(dataset[i][j] < span[i][k+1]):
            gist[i][k] += 1
        else:
            k += 1
            gist[i][k] += 1
    k = 0
```

Проверим соответствие условию задачи: если крайние значения < 5, о объединяем диапазоны.

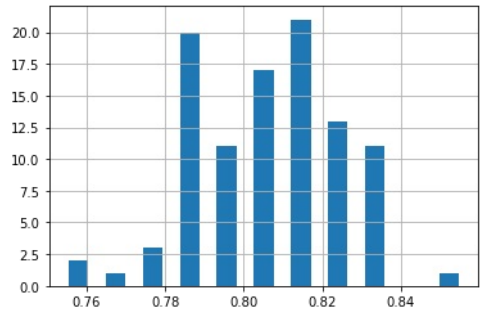
In [20]:

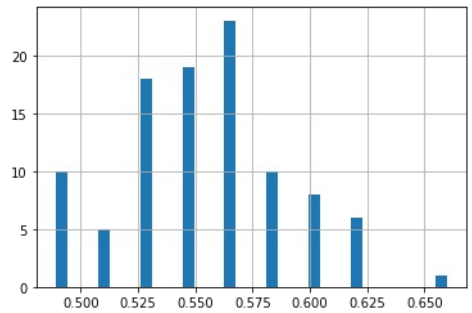
```
for i in range(4):                    #проверяем условие для крайних значений диапазона
    del span[i][0]                    #удаляем первый элемент для соответствия размерностей векторов
    if(gist[i][0] < 5):
        gist[i][1] += gist[i][0]
        del gist[i][0]
        del span[i][0]
        flag += 1
    if(gist[i][len(gist[i]) - flag - 1] < 5):
        gist[i][len(gist[i]) - flag - 2] += gist[i][len(gist[i]) - flag - 1]
        del gist[i][len(gist[i]) - flag - 1]
        del span[i][len(span[i]) - flag - 1]
    flag = 0
```

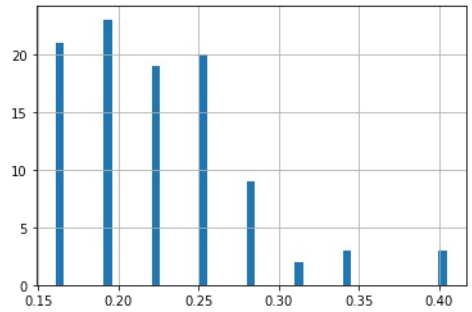
Построим гистограммы.

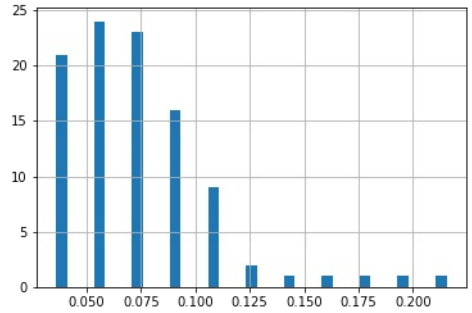
In [22]:

```
x = [[i for i in range(len(gist[j]))] for j in range(4)]
for i in range(4):
    fig = plt.figure()
    plt.bar(span[i], gist[i], 0.005,)
    plt.grid(True)
```









2.2 Сформируем таблицы ковариаций и коэффициентов корреляции.

In [15]:

```
df_3 = pd.DataFrame({"n" : n,
             "cov UTIL" : cov[0],
             "cov AVE.CONT" : cov[1],
             "r UTIL" : r[0],
             "r AVE.CONT" : r[1]})
df_3.style.hide_index()
```

Out[15]:

n	cov UTIL	cov AVE.CONT	r UTIL	r AVE.CONT
10	0.000196	0.000943	0.437873	0.848239
25	0.000400	0.002161	0.595537	0.948661
50	0.000503	0.002559	0.654403	0.946883
100	0.000413	0.001899	0.594679	0.913428

2.3 Рассчитаем доверительные интервалы.

In [16]:

```
n = [10, 10, 25, 25, 60, 60]
med_60 = [0 for i in range(4)]
s_60 = [0 for i in range(4)]
for k in range(4):
    for i in range(60):
        med_60[k] += dataset[k][i]
    med_60[k] /= 60
    for i in range(60):
        s_60[k] = (dataset[k][i] - med_60[k]) ** 2
    s_60[k] /= 59
```

Зададим значения параметров tn_(n-1)(a), исходя из таблицы для a = 0.1 и a = 0.01.

In [17]:

```
tn = [1.83, 3.25, 0.71, 2.8, 1.67, 2.66]
trust_l = [[0 for i in range(4)] for i in range(6)]
trust_r = [[0 for i in range(4)] for i in range(6)]
for i in range(2):
    for j in range(4):
        trust_l[i][j] = med[j][0] - tn[i] * s[j][0] / math.sqrt(10)
        trust_r[i][j] = med[j][0] + tn[i] * s[j][0] / math.sqrt(10)
for i in range(2):
    for j in range(4):
        trust_l[2 + i][j] = med[j][1] - tn[2 + i] * s[j][1] / math.sqrt(25)
        trust_r[2 + i][j] = med[j][1] + tn[2 + i] * s[j][1] / math.sqrt(25)
for i in range(2):
    for j in range(4):
        trust_l[4 + i][j] = med_60[j] - tn[4 + i] * s_60[j] / math.sqrt(60)
        trust_r[4 + i][j] = med_60[j] + tn[4 + i] * s_60[j] / math.sqrt(60)
trust_l = np.array(trust_l).transpose()
trust_r = np.array(trust_r).transpose()
```

Сформируем таблицу.

In [18]:

```
df_3 = pd.DataFrame({"n" : n,
                     "UTIL 1 left" : trust_l[0],
                     "UTIL 1 right" : trust_r[0],
                     "UTIL 2 left" : trust_l[1],
                     "UTIL 2 right" : trust_r[1],
                     "A.C 1 left" : trust_l[2],
                     "A.C 1 right" : trust_r[2],
                     "A.C 2 left" : trust_l[3],
                     "A.C 2 right" : trust_r[3],
                    })
df_3.style.hide_index()
```

Out[18]:

n	UTIL 1 left	UTIL 1 right	UTIL 2 left	UTIL 2 right	A.C 1 left	A.C 1 right	A.C 2 left	A.C 2 right
10	0.798100	0.798500	0.535865	0.536535	0.198824	0.201376	0.060975	0.061625
10	0.797945	0.798655	0.535605	0.536795	0.197835	0.202365	0.060724	0.061876
25	0.802797	0.802883	0.548547	0.548973	0.216659	0.217741	0.073287	0.073673
25	0.802672	0.803008	0.547921	0.549599	0.215067	0.219333	0.072717	0.074243
60	0.801450	0.801450	0.545014	0.545020	0.215198	0.215202	0.070117	0.070117
60	0.801449	0.801451	0.545012	0.545021	0.215196	0.215204	0.070116	0.070117