

VILNIAUS UNIVERSITETAS  
MATEMATIKOS IR INFORMATIKOS FAKULTETAS

**STATISTINIS MODELIAVIMAS**  
Praktinės užduoties Nr. **1—11** ataskaita

Užduotį atliko: **Matas Gaulia**  
Duomenų mokslas 2 kursas, 2 grupė

2021.05.12

## Užduotis 1—11

1. Sugeneruokite pseudoatsitiktinių skaičių sekas tiesiniu kongruentiniu metodu su maksimaliu periodu, kai modulis  $m = 1264$  ir  $m = 729$ .  
Daugiklius a parinkite taip, kad galingumai b utų didžiausi. Prieauglio c parinkimui naudokitės gretimų narių koreliacija (teoriniai testai).
2. Gautas sekas patikrinkite su dviem testais. Pirma su intervalų testu.  
Imkite intervalą  $[3/4, 1)$ . Kitą testą pasirinkite patys.
3. Naudodami sugeneruotą geresniają pseudoatsitiktinių skaičių seką sumodeliuokite du atsitiktinius dydžius, vieną pasiskirsčiusj pagal geometrinį skirstinį su parametru  $p = 0.1$ , o kitą parinkite patys.
4. Naudodami sugeneruotą geresniają pseudoatsitiktinių skaičiu seką ir parinkdami tankius (tolygiai pasiskirsčiusio intervale  $[0, 2]$ ) atsitiktinio dydžio ir kitų savo nuožiura) suskaičiuokite integralą:

$$\int_0^2 \frac{x + x^5}{1 + x} dx$$

5. Sugeneruokite Markovo grandinę, kurią pavaizdavus grafu gautume tokias viršūnių ir biraunų aibes:

$$S = \{v_1, v_2, v_3, v_4, v_5\}$$

$$E = \{(v_1, v_3), (v_1, v_5), (v_2, v_4), (v_2, v_5), (v_3, v_4),\}$$

Iš vienos viršūnės į kitą kaimyninę viršūnę pereinama su lygiomis tikimybėmis, priklausomai nuo kaimyninių viršūnių skaičiaus

## Turinys

1 užduotis.....	4
Pirma seka:.....	4
Pirmos sekos kriterijai:.....	4
Skaičiaus a parinkimo kriterijai: .....	4
Skaičiaus c parinkimo kriterijai: .....	4
Pirmos sekos parametrai: .....	4
Pirmos sekos generavimas:.....	5
Antra seka: .....	6
Antra sekos kriterijai: .....	6
Skaičiaus a parinkimo kriterijai: .....	6
Skaičiaus c parinkimo kriterijai: .....	6
Antros sekos parametrai:.....	6
Antros sekos generavimas: .....	7
2 užduotis.....	8
Intervalų testas: .....	8
Skaitmenų testas:.....	9
3 užduotis.....	11
Geometrinis skirstinys:.....	11
Puasono skirstinys:.....	12
4 užduotis.....	13
Pirmas tankis:.....	13
Antras tankis: .....	13
5 užduotis.....	14

# 1 užduotis

Sugeneruokite pseudoatsitiktinių skaičių sekas tiesiniu kongruentiniu metodu su maksimaliu periodu, kai modulis  $m = 1264$  ir  $m = 729$ . Daugiklius a parinkite taip, kad galingumai būtų didžiausi. Prieauglio c parinkimui naudokitės gretimų narių koreliacija (teoriniai testai).

Pirma seka:

Pirma seka turi būti sugeneruota su  $m = 1264$ .

Pirmos sekos kriterijai:

Skaičiaus a parinkimo kriterijai:

- Turi būti mažesnis už 1264
- A-1 turi dalintis iš visų pirminių m daugiklių ( $1264 = 2^4 \cdot 79$ )
- A-1 turi dalintis iš 4 , jei m dalinasi iš 4. (Šiuo atveju tiesa, nes  $1264/4 = 316$ )

Skaičiaus c parinkimo kriterijai:

- $\text{DBD}(c,m) = \text{DBD}(c, 1264)$  turi būti lygu 1
- $c < m$
- $\frac{c}{m} \approx \frac{1}{2} \pm \frac{1}{6}\sqrt{3}$

Pirmos sekos parametrai:

Galimi a = [317, 633, 949]

Didžiausias jmanomas galingumas yra 2 ir taip sutapo kad visi galimi a yra galingumo 2

Ir  $316^2 \bmod m = 0$ , ir  $632^2 \bmod m = 0$ , Ir  $948^2 \bmod m = 0$

Tad iš šių trjų skaičių renkuosi didžiausia (949) nes a didėjant seka tampa labiau atsitiktinė

Renkuosi a = 949:

- $949 < 1264$
- tada  $a - 1 = 948$ :
  - $948 / 2 = 474$
  - $948 / 79 = 12$
- $948 / 4 = 237$

Renkuosi  $c = 631$ :

- apskaičiuoju DBD(631, 1264) su jau parašyta Python funkcija gcd() iš bibliotekos "math":

```
In [2]: from math import gcd  
gcd(631,1264)  
Out[2]: 1
```

- $631 < 1264$
- $\frac{c}{m} = \frac{631}{1264} = 0.499$ , labai arti 0.5, tad c parinktas tinkamai

Pirmą sekos narj ir pirmoje, ir antroje sekose imsiu 0.

Pirmajai sekai rasti tokie parametrai:

- $m = 1264$
- $a = 949$
- $c = 631$
- $X_0 = 0$

Pirmos sekos generavimas:

Generuoju seką su Python funkcija LCG (Linear Congruential Generator), funkcijos kodas parodytas priede:

```
In [4]: pirma = LCG(seed = 0, a = 949, c = 631, m = 1264)  
print(pirma)
```

[631, 314, 313, 628, 1259, 942, 941, 1256, 623, 306, 305, 620, 1251, 934, 933, 1248, 615, 298, 297, 612, 1243, 926, 925, 1240, 607, 290, 289, 604, 1239, 918, 917, 1232, 599, 282, 281, 596, 1227, 910, 909, 1224, 591, 274, 273, 588, 1219, 902, 901, 1216, 5 83, 266, 265, 580, 1211, 894, 893, 1208, 575, 258, 257, 572, 1203, 886, 885, 1208, 567, 250, 249, 564, 1195, 878, 877, 1192, 55 9, 242, 241, 556, 1187, 870, 869, 1184, 551, 234, 233, 548, 1179, 862, 861, 1176, 543, 226, 225, 540, 1171, 854, 853, 1168, 53 5, 218, 217, 532, 1163, 846, 845, 1160, 527, 210, 209, 524, 1155, 837, 837, 1152, 519, 202, 201, 516, 1147, 830, 829, 1144, 51 1, 194, 193, 508, 1139, 822, 821, 1136, 503, 186, 185, 500, 1131, 814, 813, 1128, 495, 178, 177, 492, 1123, 806, 805, 1120, 48 7, 170, 169, 484, 1115, 798, 797, 1112, 479, 162, 161, 476, 1107, 790, 789, 1104, 471, 154, 153, 468, 1099, 782, 781, 1096, 46 3, 146, 145, 460, 1091, 774, 773, 1088, 455, 138, 137, 452, 1083, 766, 765, 1080, 447, 130, 129, 444, 1075, 758, 757, 1072, 43 9, 122, 121, 436, 1067, 750, 749, 1064, 431, 114, 113, 428, 1059, 742, 741, 1056, 423, 106, 105, 420, 1051, 734, 733, 1048, 41 5, 98, 97, 412, 1043, 726, 725, 1040, 407, 90, 89, 404, 1035, 718, 717, 1032, 399, 82, 81, 396, 1027, 710, 709, 1024, 391, 74, 73, 388, 1019, 702, 701, 1016, 383, 66, 65, 380, 1011, 694, 693, 1008, 375, 58, 57, 372, 1003, 686, 685, 1000, 367, 58, 49, 36 4, 995, 678, 677, 992, 359, 42, 41, 356, 987, 670, 669, 984, 351, 34, 33, 348, 979, 682, 661, 976, 343, 26, 25, 340, 971, 654, 653, 968, 335, 18, 17, 332, 963, 646, 645, 960, 327, 10, 9, 324, 955, 638, 637, 952, 319, 2, 1, 316, 947, 630, 629, 944, 311, 1 258, 1257, 308, 939, 622, 621, 936, 303, 1250, 1249, 300, 931, 614, 613, 928, 295, 1242, 1241, 292, 923, 606, 605, 920, 287, 12 34, 1223, 284, 915, 598, 597, 912, 279, 1226, 1225, 276, 907, 590, 589, 904, 271, 1218, 1217, 268, 899, 582, 581, 896, 263, 12 0, 1209, 260, 891, 574, 573, 888, 255, 1202, 1201, 252, 883, 566, 565, 880, 247, 1194, 1193, 244, 875, 558, 557, 872, 239, 118 6, 1185, 236, 867, 550, 549, 864, 231, 1178, 1177, 228, 859, 542, 541, 856, 223, 1170, 1169, 228, 851, 534, 533, 848, 215, 116 2, 1161, 212, 843, 526, 525, 840, 207, 1154, 1153, 204, 835, 518, 517, 832, 199, 1146, 1145, 196, 827, 510, 509, 824, 191, 113 8, 1137, 188, 819, 502, 501, 816, 183, 1130, 1129, 180, 811, 494, 493, 808, 175, 1122, 1121, 172, 803, 486, 485, 800, 167, 111 4, 1113, 164, 795, 478, 477, 792, 159, 1106, 1105, 156, 787, 470, 469, 784, 151, 1098, 1097, 148, 779, 462, 461, 776, 143, 109 0, 1089, 140, 771, 454, 453, 768, 135, 1802, 1081, 132, 763, 446, 445, 750, 127, 1074, 1073, 124, 755, 438, 437, 752, 119, 106 6, 1065, 116, 747, 430, 429, 744, 111, 1058, 1057, 108, 739, 422, 421, 736, 1026, 1025, 76, 707, 390, 389, 704, 71, 1018, 1017, 6 8, 699, 382, 381, 696, 63, 1810, 1009, 60, 691, 374, 373, 688, 55, 1002, 1001, 52, 683, 366, 365, 680, 47, 994, 993, 44, 675, 3 58, 357, 672, 39, 986, 985, 36, 667, 350, 349, 664, 31, 978, 977, 28, 659, 342, 341, 656, 23, 970, 969, 20, 651, 334, 333, 648, 15, 962, 961, 12, 643, 326, 325, 640, 7, 954, 953, 4, 635, 318, 317, 632, 1263, 946, 945, 1260, 627, 310, 309, 624, 1255, 938, 937, 1252, 619, 302, 301, 616, 1247, 930, 929, 1244, 611, 294, 293, 608, 1239, 922, 921, 1236, 603, 286, 285, 600, 1231, 914, 9 13, 1228, 595, 278, 277, 592, 1223, 906, 905, 1220, 587, 270, 269, 584, 1215, 898, 897, 1212, 579, 262, 261, 576, 1207, 898, 88 9, 1204, 571, 254, 253, 568, 1199, 882, 881, 1196, 563, 246, 245, 560, 1191, 874, 873, 1188, 555, 238, 237, 552, 1183, 866, 86 5, 1180, 547, 230, 229, 544, 1175, 858, 857, 1172, 539, 222, 221, 536, 1167, 850, 849, 1164, 531, 214, 213, 528, 1159, 842, 84 1, 1156, 523, 206, 205, 520, 1151, 834, 833, 1148, 515, 198, 197, 512, 1143, 826, 825, 1140, 507, 190, 189, 504, 1135, 818, 81 7, 1132, 499, 182, 181, 496, 1127, 810, 809, 1124, 491, 174, 173, 488, 1119, 802, 801, 1116, 483, 166, 165, 480, 1111, 794, 79 3, 1108, 475, 158, 157, 472, 1103, 786, 785, 1100, 467, 158, 149, 464, 1095, 778, 777, 1092, 459, 142, 141, 456, 1087, 770, 76 9, 1084, 451, 134, 133, 448, 1079, 762, 761, 1076, 443, 126, 125, 440, 1071, 754, 753, 1068, 435, 118, 117, 432, 1063, 746, 74 5, 1060, 427, 110, 109, 424, 1055, 738, 737, 1052, 419, 102, 101, 416, 1047, 730, 729, 1044, 411, 94, 93, 408, 1039, 722, 721, 1036, 403, 86, 85, 400, 1031, 714, 713, 1028, 395, 78, 77, 392, 1023, 706, 705, 1020, 387, 70, 69, 384, 1015, 698, 697, 1012, 3 79, 62, 61, 376, 1007, 690, 689, 1004, 371, 54, 53, 368, 999, 682, 681, 996, 363, 46, 45, 360, 991, 674, 673, 988, 355, 38, 37, 352, 983, 666, 665, 980, 347, 30, 29, 344, 975, 658, 657, 972, 339, 22, 21, 336, 967, 650, 649, 964, 331, 14, 13, 328, 959, 64 2, 641, 956, 323, 6, 5, 320, 951, 634, 633, 948, 315, 1262, 1261, 312, 943, 626, 940, 387, 1254, 1253, 304, 935, 618, 617, 932, 299, 1246, 1245, 296, 927, 610, 609, 924, 291, 1238, 1237, 288, 919, 682, 681, 916, 283, 1230, 1229, 280, 911, 594, 593, 9 08, 275, 1222, 1221, 272, 903, 586, 585, 900, 267, 1214, 264, 895, 578, 577, 892, 259, 1206, 1205, 256, 887, 570, 569, 88 4, 251, 1198, 1197, 248, 879, 562, 561, 876, 243, 1190, 1189, 240, 871, 554, 553, 868, 235, 1182, 1181, 232, 863, 546, 545, 86 0, 227, 1174, 1173, 224, 855, 538, 537, 852, 219, 1166, 1165, 216, 847, 530, 529, 844, 211, 1158, 1157, 208, 839, 522, 521, 83 6, 203, 1150, 1149, 200, 831, 514, 513, 828, 195, 1142, 1141, 192, 823, 506, 505, 820, 187, 1134, 1133, 184, 815, 498, 497, 81 2, 179, 1126, 1125, 176, 807, 490, 489, 804, 171, 1118, 1117, 168, 799, 482, 481, 796, 163, 1110, 1109, 160, 791, 474, 473, 78 8, 155, 1102, 1101, 152, 783, 466, 465, 780, 147, 1094, 1093, 144, 775, 458, 457, 772, 139, 1086, 1085, 136, 767, 450, 449, 44 4, 131, 1078, 1077, 128, 759, 442, 441, 756, 123, 1078, 1069, 120, 751, 434, 433, 748, 115, 1062, 1061, 112, 743, 426, 425, 74 0, 107, 1054, 1053, 104, 735, 418, 417, 732, 99, 1046, 1045, 96, 727, 410, 409, 724, 91, 1038, 1037, 88, 719, 402, 401, 718, 8 3, 1030, 1029, 80, 711, 394, 393, 708, 75, 1022, 1021, 72, 703, 386, 385, 700, 67, 1014, 1013, 64, 695, 378, 377, 692, 59, 100 6, 1005, 56, 687, 370, 369, 684, 51, 998, 997, 48, 679, 362, 361, 676, 43, 990, 989, 40, 671, 354, 353, 668, 35, 982, 981, 32, 663, 346, 345, 660, 27, 974, 973, 24, 655, 338, 337, 652, 19, 966, 965, 16, 647, 330, 329, 644, 11, 958, 957, 8, 639, 322, 321, 636, 3, 950, 949, 0]

Antra seka:

Antra sekos kriterijai:

Skaičiaus a parinkimo kriterijai:

- Turi būti mažesnis už 729
- $a - 1$  turi dalintis iš visų pirminių 729 daugiklių ( $729 = 3^6$ )
- $a - 1$  turi dalintis iš 4 , jei 729 dalinasi iš 4. (Šiuo atveju netiesa, nes  $729/4 = 182.25$ )

Skaičiaus c parinkimo kriterijai:

- $\text{DBD}(c,m) = \text{DBD}(c, 729)$  turi būti lygu 1
- $c < m$
- $\frac{c}{m} \approx \frac{1}{2} \pm \frac{1}{6}\sqrt{3}$

Antros sekos parametrai:

Galimi a yra visi skaičiai iki 729, kurie išreiškiami  $3i + 1$ , tokų skaičių yra 242.

Maksimalus galingumas s = 6.

Tokie galimi a:

4, 7, 13, 16, 22, 25, 31, 34, 40, 43, 49, 52, 58, 61, 67, 70, 76, 79, 85, 88, 94, 97, 103, 106, 112, 115, 121, 124, 130, 133, 139, 142, 148, 151, 157, 160, 166, 169, 175, 178, 184, 187, 193, 196, 202, 205, 211, 214, 220, 223, 229, 232, 238, 241, 247, 250, 256, 259, 265, 268, 274, 277, 283, 286, 292, 295, 301, 304, 310, 313, 319, 322, 328, 331, 337, 340, 346, 349, 355, 358, 364, 367, 373, 376, 382, 385, 391, 394, 400, 403, 409, 412, 418, 421, 427, 430, 436, 439, 445, 448, 454, 457, 463, 466, 472, 475, 481, 484, 490, 493, 499, 502, 508, 511, 517, 520, 526, 529, 535, 538, 544, 547, 553, 556, 562, 565, 571, 574, 580, 583, 589, 592, 598, 601, 607, 610, 616, 619, 625, 628, 634, 637, 643, 646, 652, 655, 661, 664, 670, 673, 679, 682, 688, 691, 697, 700, 706, 709, 715, 718, 724, 727

Iš galimų a imu kuo didesni.

Renkuosi a = 727:

- $727 < 729$
- $a - 1 = 726, 726/3 = 242$

Renkuosi c = 365:

- apskaičiuoju  $\text{DBD}(365, 729)$  taip pat kaip ir pirmoje sekoje:

```
In [3]: from math import gcd  
gcd(365,729)  
Out[3]: 1
```

- $365 < 729$
- $\frac{c}{m} = \frac{365}{729} = 0.5007$ , vel labai arti 0.5, tad c parinktas gerai.

Antrajai sekai rasti tokie parametrai:

- $m = 729$
- $a = 727$
- $c = 365$
- $X_0 = 0$

Antros sekos generavimas:

```
antra = LCG(seed = 0, a = 727, c = 365, m = 729)
print(antra)

[365, 364, 366, 362, 370, 354, 386, 322, 450, 194, 706, 411, 272, 550, 723, 377, 340, 414, 266, 562, 699, 425, 244, 606, 611, 6
01, 621, 581, 661, 501, 92, 181, 3, 359, 376, 342, 410, 274, 546, 2, 361, 372, 350, 394, 306, 482, 130, 105, 155, 55, 255, 584,
655, 513, 68, 229, 636, 551, 721, 381, 332, 430, 234, 626, 571, 681, 461, 172, 21, 323, 448, 198, 698, 427, 240, 614, 595, 633,
557, 709, 405, 284, 526, 42, 281, 532, 30, 305, 484, 126, 113, 139, 87, 191, 712, 399, 296, 502, 90, 185, 724, 375, 344, 406, 2
82, 530, 34, 297, 500, 94, 177, 11, 343, 408, 278, 538, 18, 329, 436, 222, 650, 523, 48, 269, 556, 711, 401, 292, 510, 74, 217,
660, 503, 88, 189, 716, 391, 312, 470, 154, 57, 251, 592, 639, 545, 4, 357, 380, 334, 426, 242, 610, 603, 617, 589, 645, 533, 2
8, 309, 476, 142, 81, 203, 688, 447, 200, 694, 435, 224, 646, 531, 32, 301, 492, 110, 145, 75, 215, 664, 495, 104, 157, 51, 26
3, 568, 687, 449, 196, 702, 419, 256, 582, 659, 505, 84, 197, 700, 423, 248, 598, 627, 569, 685, 453, 188, 718, 387, 320, 454,
186, 722, 379, 336, 422, 250, 594, 635, 553, 717, 389, 316, 462, 170, 25, 315, 464, 166, 33, 299, 496, 102, 161, 43, 279, 536,
22, 321, 452, 190, 714, 395, 304, 486, 122, 121, 123, 119, 127, 111, 143, 79, 207, 680, 463, 168, 29, 307, 480, 134, 97, 171, 2
3, 319, 456, 182, 1, 363, 368, 358, 378, 338, 418, 258, 578, 667, 489, 116, 133, 99, 167, 31, 303, 488, 118, 129, 107, 151, 63,
239, 616, 591, 641, 541, 12, 341, 412, 270, 554, 715, 393, 308, 478, 138, 89, 187, 720, 383, 328, 438, 218, 658, 507, 80, 205,
684, 455, 184, 726, 371, 352, 390, 314, 466, 162, 41, 283, 528, 38, 289, 516, 62, 241, 612, 599, 625, 573, 677, 469, 156, 53, 2
59, 576, 671, 481, 132, 101, 163, 39, 287, 520, 54, 257, 580, 663, 497, 100, 165, 35, 295, 504, 86, 193, 708, 407, 280, 534, 2
6, 313, 468, 158, 49, 267, 560, 703, 417, 260, 574, 675, 473, 148, 69, 227, 640, 543, 8, 349, 396, 302, 490, 114, 137, 91, 183,
728, 367, 360, 374, 346, 402, 290, 514, 66, 233, 628, 567, 689, 445, 204, 686, 451, 192, 710, 403, 288, 518, 58, 249, 596, 631,
561, 701, 421, 252, 590, 643, 537, 20, 325, 444, 206, 682, 459, 176, 13, 339, 416, 262, 570, 683, 457, 180, 5, 355, 384, 326, 4
42, 210, 674, 475, 144, 77, 211, 672, 479, 136, 93, 179, 7, 351, 392, 310, 474, 146, 73, 219, 656, 511, 72, 221, 652, 519, 56,
253, 588, 647, 529, 36, 293, 508, 78, 209, 676, 471, 152, 61, 243, 608, 607, 609, 605, 613, 597, 629, 565, 693, 437, 220, 654,
515, 64, 237, 620, 583, 657, 509, 76, 213, 668, 487, 120, 125, 115, 135, 95, 175, 15, 335, 424, 246, 602, 619, 585, 653, 517, 6
0, 245, 604, 615, 593, 637, 549, 725, 373, 348, 398, 298, 498, 98, 169, 27, 311, 472, 150, 65, 235, 624, 575, 673, 477, 140, 8
5, 195, 704, 415, 264, 566, 691, 441, 212, 670, 483, 128, 109, 147, 71, 223, 648, 527, 40, 285, 524, 46, 273, 548, 727, 369, 35
6, 382, 330, 434, 226, 642, 539, 16, 333, 428, 238, 618, 587, 649, 525, 44, 277, 540, 14, 337, 420, 254, 586, 651, 521, 52, 26
1, 572, 679, 465, 164, 37, 291, 512, 70, 225, 644, 535, 24, 317, 460, 174, 17, 331, 432, 230, 634, 555, 713, 397, 300, 494, 10
6, 153, 59, 247, 600, 623, 577, 669, 485, 124, 117, 131, 103, 159, 47, 271, 552, 719, 385, 324, 446, 202, 690, 443, 208, 678, 4
67, 160, 45, 275, 544, 6, 353, 388, 318, 458, 178, 9, 347, 400, 294, 506, 82, 201, 692, 439, 216, 662, 499, 96, 173, 19, 327, 4
40, 214, 666, 491, 112, 141, 83, 199, 696, 431, 232, 630, 563, 697, 429, 236, 622, 579, 665, 493, 108, 149, 67, 231, 632, 559,
705, 413, 268, 558, 707, 409, 276, 542, 10, 345, 404, 286, 522, 50, 265, 564, 695, 433, 228, 638, 547, 0]
```

## 2 užduotis

Gautas sekas patikrinkite su dviem testais. Pirma su intervalų testu.  
Imkite intervalą [3/4, 1). Kitą testą pasirinkite patys.

### Intervalų testas:

Kadangi alpha = 0.75, o beta = 1, p = beta – alpha = 1 - 0.75 = 0.25

Tada turime tikimybes:

$$p_i = P(\text{intervalo ilgis} = i) = p(1-p)^i, 0 \leq i \leq t,$$
$$p_{>i} = P(\text{intervalo ilgis} > t) = (1-p)^{t+1}$$

$$p_0 = 0.25 * 0.75^0 = 0.25$$
$$p_1 = 0.25 * 0.75^1 = 0.1875$$
$$p_2 = 0.25 * 0.75^2 = 0.140625$$
$$p_3 = 0.25 * 0.75^3 = 0.10546875$$
$$p_4 = 0.25 * 0.75^4 = 0.0791015625$$
$$p_{>4} = 0.75^5 = 0.2373046875$$

Chi-Kvadrato formulė :

$$\chi^2 = \sum \frac{(o - e)^2}{e}$$

Kur o – stebėta reikšmė, o e – tikėtasi reikšmė (expected)

Šiuo atveju tai atitinkamai bus intervalų dažniai

Pasirenku t = 4. r = t+2 = 4+2 = 6

Reikšmingumo lygis: 0.05

Laisvės laipsniai: 5

Kritinė reikšmė: 11.070

Pirma seka:

```
n = 316 , p = 0.25
{'0': 79, '1': 1, '2': 79, '3': 0, '4': 78, '>4': 79}
Pi: [0.25, 0.1875, 0.140625, 0.10546875, 0.0791015625, 0.2373046875]
Tikimasi tokiai dažnių: [79.0, 59.25, 44.4375, 33.328125, 24.99609375, 74.98828125]
Laisvės laipsniai: 5
Chi-Kvadrato kriterijus: Power_divergenceResult(statistic=230.08569047246965, pvalue=1.0252518096810503e-47)
```

**Išvada:** Matome, kad  $230.085 > 11.070$ , hipotezė kad pirmoji seka pasiskirsčiu tolygiai, galime atesti, tą antrina ir labai maža p reikšmė.

Dabar antroji seka, t<sub>1</sub>, reikšmingumo lygmuo tokie patys:

```
Antra seka:  
n = 182 , p = 0.25  
{'0': 91, '1': 0, '2': 23, '3': 11, '4': 11, '>4': 46}  
Pi: [0.25, 0.1875, 0.140625, 0.10546875, 0.0791015625, 0.2373046875]  
Tikimasi tokiu dažniu: [45.5, 34.125, 25.59375, 19.1953125, 14.396484375, 43.189453125]  
Laisvės laipsniai: 5  
Chi-Kvadrato kriterijus: Power_divergenceResult(statistic=84.37100348211459, pvalue=1.0198276906110155e-16)
```

**Išvada:** 84.37 > 11.070, hipotezę kad antroji seka pasiskirsčiui tolygiai, galime atesti, tą parodo ir labai maža p<sub>0</sub> reikšmė

Skaitmenų testas:

Naudosiu antrajį skaitmenų testą.

$$P(\text{skaitmens } s \text{ pasirodymas}) = \frac{1}{10}, s = 0, 1, \dots, 9$$

Pradžioje seką paverčiu į seką tarp 0 ir 1 (padalinu iš m).

Tuomet imu 5 skaičius po kablelio, jei tiek nėra, pridedu reikiama kiekj 0.

Tuomet gaunu seką tarp 0 ir 9, kurioje teoriškai kiekvienas skaitmuo turėtu būti pasiskirstęs vienodai (hipotezė tokia)

Tuomet skaičiuoju chi-kvadrato kriterijų su anksčiau naudota formule, kur tikėtasis dažnis kiekvieno skaitmens yra n\*0.1

Reikšmingumo lygis: 0.05

Laisvės laipsniai: 9

Kritinė reikšmė: 16.919

Pirma seka:

```
Pirma seka:  
SortedDict({0: 639, 1: 636, 2: 633, 3: 636, 4: 624, 5: 639, 6: 636, 7: 633, 8: 636, 9: 608})  
n = m*5 = 1264 *5 = 6320  
Pi: [0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]  
Tikimasi tokiu dažniu: [632, 632, 632, 632, 632, 632, 632, 632, 632]  
Chi-Kvadrato kriterijus: Power_divergenceResult(statistic=1.2721518987341773, pvalue=0.9985106386952636)
```

**Išvada:** 1.27 < 16.919, tad hipotezės, kad pirma seka pasiskirsčiusi tolygiai, negalime atesti

Antra seka:

```
Antra seka:  
SortedDict({0: 427, 1: 424, 2: 420, 3: 423, 4: 417, 5: 396, 6: 296, 7: 291, 8: 283, 9: 268})  
n = m*5 = 729 *5 = 3645  
Pi: [0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]  
Tikimasi tokiu dažniu: [364, 364, 364, 364, 364, 364, 364, 364, 364]  
Chi-Kvadrato kriterijus: Power_divergenceResult(statistic=120.18956043956042, pvalue=1.2219567176506306e-21)
```

**Išvada:**  $120.19 > 16.919$ , tad hipotezę, kad antra seka pasiskirsčiusi tolygiai, galime atmesti

**Rezultatai:**

Intervalų testas parodė, kad antroji seka yra geresnė (kriterijus mažesnis),  
O skaitmenų testas parodė atvirkščiai: pirmoji seka yra geresnė.  
Kadangi intervalų testą laikau kaip tikslesniu, toliau užduotyse naudosiu **antrą seką**

# 3 užduotis

Naudodami sugeneruotą geresnįją pseudoatsitiktinių skaičių seką sumodeliuokite du atsitiktinius dydžius, vieną pasiskirsčiusį pagal geometrinį skirstinį su parametru  $p = 0.1$ , o kitą parinkite patys.

Geometrinis skirstinys:

Pradžioje paversiu savo seką kad būtu tarp 0 ir 1 padalindamas seką iš  $m = 729$   
Kad gaučiau geometrinį skirstinį naudosiu tokią formulę:

$$N = \left\lceil \frac{\ln U}{\ln(1 - p)} \right\rceil$$

Kadangi  $\ln(0)$  nėra apibrėžtas, sugeneruotų skaičių kiekis bus  $m - 1 = 728$

Turiu  $p = 0.1$ , tai  $1-p = 0.9$ , o  $\ln(1-p) = -0.10536051565$

Su  $p = 0.1$ , teoriškai vidurkis bus  $1/p = 1/0.1 = 10$

Generuoju:

```
In [275]: geom = generuoti_geometrini(p = 0.1, seką = antra, m = 729)
print(geom)
print("\nTeorinis vidurkis: ", 1/0.1)
print("Gautas vidurkis: ", mean(geom))

[7, 7, 7, 7, 7, 7, 8, 5, 13, 1, 6, 10, 3, 1, 7, 8, 6, 10, 3, 1, 6, 11, 2, 2, 2, 3, 1, 4, 20, 14, 53, 7, 7, 8, 6, 10, 3, 5
6, 7, 7, 7, 6, 9, 4, 17, 19, 15, 25, 10, 3, 2, 4, 23, 11, 2, 3, 1, 7, 8, 6, 11, 2, 3, 1, 5, 14, 34, 8, 5, 13, 1, 6, 11, 2, 2,
2, 3, 1, 6, 9, 4, 28, 10, 3, 31, 9, 4, 17, 18, 16, 21, 13, 1, 6, 9, 4, 20, 14, 1, 7, 8, 6, 10, 4, 30, 9, 4, 20, 14, 40, 8, 6, 1
0, 3, 36, 8, 5, 12, 2, 4, 26, 10, 3, 1, 6, 9, 4, 22, 12, 1, 4, 21, 13, 1, 6, 9, 5, 15, 25, 11, 2, 2, 3, 50, 7, 7, 8, 6, 11, 2,
2, 2, 3, 2, 3, 31, 9, 5, 16, 21, 13, 1, 5, 13, 1, 5, 12, 2, 4, 30, 9, 4, 18, 16, 22, 12, 1, 4, 19, 15, 26, 10, 3, 1, 5, 13, 1,
6, 10, 3, 1, 4, 21, 13, 1, 6, 11, 2, 2, 3, 1, 5, 13, 1, 7, 8, 5, 13, 1, 7, 8, 6, 11, 2, 2, 3, 1, 6, 8, 5, 14, 33, 8, 5, 15, 20
9, 4, 19, 15, 27, 10, 3, 34, 8, 5, 13, 1, 6, 9, 4, 17, 18, 17, 18, 16, 22, 12, 1, 5, 14, 31, 9, 4, 17, 20, 14, 33, 8,
5, 14, 63, 7, 7, 7, 8, 6, 10, 3, 1, 4, 18, 17, 19, 14, 30, 9, 4, 18, 17, 19, 15, 24, 11, 2, 2, 3, 39, 8, 6, 10, 3, 1, 6,
9, 5, 16, 20, 13, 1, 7, 8, 5, 12, 1, 4, 21, 13, 1, 5, 14, 1, 7, 7, 6, 8, 5, 15, 28, 9, 4, 29, 9, 4, 24, 11, 2, 2, 3, 1, 5, 1
5, 25, 10, 3, 1, 4, 17, 19, 15, 28, 9, 4, 25, 10, 3, 1, 4, 19, 15, 29, 9, 4, 21, 13, 1, 6, 10, 3, 32, 9, 5, 15, 26, 10, 3, 1,
6, 10, 3, 1, 5, 16, 23, 12, 2, 3, 43, 7, 6, 9, 4, 18, 16, 20, 14, 1, 7, 7, 8, 6, 9, 4, 23, 11, 2, 3, 1, 5, 13, 1, 5, 13, 1,
6, 9, 4, 25, 11, 2, 3, 1, 6, 11, 3, 2, 3, 35, 8, 5, 12, 1, 5, 14, 39, 8, 6, 10, 3, 1, 5, 14, 48, 7, 7, 8, 5, 12, 1, 5, 16, 2
2, 12, 1, 4, 16, 20, 14, 45, 7, 6, 9, 5, 16, 22, 12, 2, 4, 22, 12, 2, 4, 25, 11, 3, 2, 4, 29, 9, 4, 22, 12, 1, 5, 15, 24, 11,
2, 2, 2, 2, 2, 2, 3, 1, 5, 12, 2, 4, 24, 11, 2, 3, 1, 4, 22, 12, 1, 4, 18, 17, 18, 17, 20, 14, 37, 8, 6, 11, 2, 2, 3, 2, 4,
24, 11, 2, 2, 2, 3, 1, 7, 8, 6, 9, 4, 20, 14, 32, 9, 5, 16, 23, 11, 2, 3, 1, 5, 16, 21, 13, 1, 6, 10, 3, 1, 5, 12, 1, 4, 17,
19, 16, 23, 12, 2, 4, 28, 9, 4, 27, 10, 3, 1, 7, 7, 8, 5, 12, 2, 3, 37, 8, 6, 11, 2, 3, 2, 4, 27, 10, 3, 38, 8, 6, 11, 3, 2,
4, 26, 10, 3, 1, 5, 15, 29, 9, 4, 23, 12, 2, 3, 33, 8, 5, 14, 36, 8, 5, 11, 2, 3, 1, 6, 9, 4, 19, 15, 24, 11, 2, 2, 3, 1, 4, 1
7, 18, 17, 19, 15, 27, 10, 3, 1, 7, 8, 5, 13, 1, 5, 12, 1, 5, 15, 27, 10, 3, 46, 7, 6, 8, 5, 14, 42, 8, 6, 9, 4, 21, 13, 1, 5,
12, 1, 4, 20, 14, 35, 8, 5, 12, 1, 4, 18, 16, 21, 13, 1, 5, 11, 2, 3, 1, 4, 19, 16, 23, 11, 2, 3, 1, 6, 10, 3, 1, 6, 10, 3,
1, 6, 10, 3, 41, 8, 6, 9, 4, 26, 10, 3, 1, 5, 12, 2, 3]
```

Teorinis vidurkis: 10.0  
Gautas vidurkis: 9.958791208791208

Sugeneravau 728 skaičius.

Gautas vidurkis labia arti teorinio vidurkio.

## Puasono skirstinys:

Pasirinkau puasono su vidurkiu 0.5

$$P(N = n) = \frac{e^{-\mu} \mu^n}{n!}$$

vadinama Puasono vardu ir sakoma, kad atsitiktinis dydis  $N$  yra psiskirstęs pagal Puasono dėsnį su vidurkiu  $\mu$ .

Mano U skirstinys bus mano antroji seka padalinta iš m, tada gausiu seką kuri bus tarp 0 ir 1.

## Salygos:

- Jei  $U_1 < p$ , tai  $N = 0$
  - Jei  $U_1 \geq p$ , imu  $U_1$ 
    - Jei  $U_1 * U_2 < p$ , tai  $N = 1$
  - Jei  $U_1 \geq p$  ir  $U_1 * U_2 \geq p$ ,  
imu dar vieną skaičių  $U_{n+1}$  (ir dar jei reikia)
    - Jei  $U_1 * U_2 * \dots * U_{n+1} < p$ , tai  $N = n$

## Generuoju:

Sugeneravau 489 skaičius

Kaip matome gautas vidurkis labai arti teorinio vidurkio (0.5)

# 4 užduotis

Naudodami sugeneruotį geresnįjį pseudoatsitiktiniu skaičiu seką ir parinkdami tankius (tolygiai pasiskirsčiusio intervale  $[0, 2]$  atsitiktinio dydžio ir kitą savo nuožiura) suskaičiuokite integralą:

$$I = \int_0^2 \frac{x + x^5}{1 + x} dx$$

Su internetine skaičiuokle šis integralas lygus **4.87**

Kad galėčiau naudoti savo antrają seką kaip tolygiai pasiskirsčiusij dydį, turiu kiekvieną skaičių padalinti iš  $m = 729$

```
In [281]: useka = [i/729 for i in antra]
N = 729
```

Pirmas tankis:

Pirmas tankis bus  $p_{\varepsilon_1}(x) = \frac{1}{2}$ ;  $\int_0^2 \frac{1}{2} dx = 1$ , tada  $\int_0^\varepsilon \frac{1}{2} dx = \frac{\varepsilon}{2}$

Jei  $U$  yra tolygiai pasikirstęs intervalė  $(0,1)$ , tada  $\frac{\varepsilon}{2} = U$ , tai  $\varepsilon = 2U$

Tada galime skaičiuoti integralą  $I \approx \frac{1}{N} \sum_{j=1}^N \left( \frac{\varepsilon_j + \varepsilon_j^5}{1 + \varepsilon_j} \right) \cdot \frac{2}{1} = \frac{2}{N} \sum_{j=1}^N \frac{\varepsilon_j + \varepsilon_j^5}{1 + \varepsilon_j}$

```
In [287]: suma_1 = 0
epsilon_1 = [2*u for u in useka]
for i in epsilon_1:
    if i!= 0:
        suma_1+=(2/N)*(i+i**5)/(1+i)
print(suma_1)
```

4.853909617063339

Su pirmiu tankiu gauname, kad integralas lygus apytiksliai **4.85**

Antras tankis:

Antras tankis bus  $p_{\varepsilon_2}(x) = \frac{x^3}{4}$ ;  $\int_0^2 \frac{x^3}{4} dx = 1$ , tada  $\int_0^\varepsilon \frac{x^3}{4} dx = \frac{\varepsilon^4}{16}$

Jei  $U$  yra tolygiai pasikirstęs intervalė  $(0,1)$ , tada  $\frac{\varepsilon^4}{16} = U$ , tai  $\varepsilon = 2\sqrt[4]{U}$

Tada galime skaičiuoti integralą  $I \approx \frac{1}{N} \sum_{j=1}^N \left( \frac{\varepsilon_j + \varepsilon_j^5}{1 + \varepsilon_j} \right) \cdot \frac{4}{\varepsilon_j^3} = \frac{4}{N} \sum_{j=1}^N \frac{\varepsilon_j + \varepsilon_j^5}{\varepsilon_j^3(1+\varepsilon_j)}$

```
In [288]: suma_2 = 0
epsilon_2 = [2*(u**(1/4)) for u in useka]
for i in epsilon_2:
    if i!= 0:
        suma_2+=(4/N)*(i+i**5)/((1+i)*(i**3))
print(suma_2)
```

4.820724891201739

Su antru tankiu gauname, kad integralas lygus apytiksliai **4.82**

Geresnis tankis yra pirmasis, nes artimesnis tikrajai integralo reikšmei

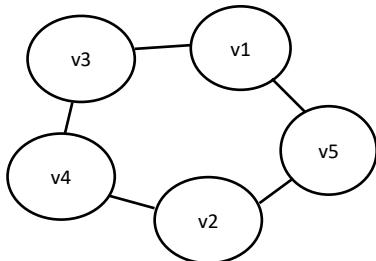
# 5 užduotis

Sugeneruokite Markovo grandinę, kurią pavaizdavus grafu gautume tokias viršūnių ir biraunų aibes:

$$S = \{v_1, v_2, v_3, v_4, v_5\}$$
$$E = \{(v_1, v_3), (v_1, v_5), (v_2, v_4), (v_2, v_5), (v_3, v_4)\}$$

Iš vienos viršūnės į kitą kaimyninę viršūnę pereinama su lygiomis tikimybėmis, priklausomai nuo kaimyninių viršūnių skaičiaus

Grafa pavaizdavus jis atrodo taip:



Pradžiai jsivedu kintamuosius

```
In [1]: import numpy as np
nodes = ["v1","v2","v3","v4","v5"]
edges = [("v1","v3"),("v1","v5"),("v2","v4"),("v2","v5"),("v3","v4")]
n = len(nodes)
m = len(edges)
```

Susiskaičiuoju kiek kiekviena viršūnė turi kaimyninių viršūnių:

```
In [2]: nodes_neighbours = {}
for node in nodes:
    nodes_neighbours[node] = 0
    for edge in edges:
        if edge[0] == node or edge[1] == node:
            nodes_neighbours[node] += 1

print("Kaimynių viršūnių kiekiai",nodes_neighbours)

Kaimynių viršūnių kiekiai {'v1': 2, 'v2': 2, 'v3': 2, 'v4': 2, 'v5': 2}
```

Turėdamas kaimyninių viršūnių skaičių, galu skaiciuoti tikimybę pereiti į kiekvieną viršūnę:

```
In [3]: p_nodes_neighbours = {node:(1/nodes_neighbours[node]) for node in nodes}
print("Tikimybės pereiti į viršunes",p_nodes_neighbours)

Tikimybės pereiti į viršunes {'v1': 0.5, 'v2': 0.5, 'v3': 0.5, 'v4': 0.5, 'v5': 0.5}
```

Pavaizduosiu perėjimų matricą:

```
In [298]: matrix = np.zeros((n, n))

for i in range(n):
    for j in range(n):
        if (nodes[i],nodes[j]) in edges:
            matrix[i][j] = p_nodes_neighbours[nodes[i]]
            matrix[j][i] = p_nodes_neighbours[nodes[j]]

print("Perėjimų matrica: \n",matrix)

Perėjimų matrica:
[[0.  0.  0.5  0.  0.5]
 [0.  0.  0.  0.5  0.5]
 [0.5 0.  0.  0.5  0. ]
 [0.  0.5 0.5  0.  0. ]
 [0.5 0.5 0.  0.  0. ]]
```

Simuliavimas vyks taip:

Paleisiu ciklą kur p bus mano sekos skaičius padalintas iš m = 729

- Esu 1 būsenoj
    - $p < 0.5$ 
      - Pereinu j 3 būsenaq
    - $p > 0.5$ 
      - Pereinu j 5 būsenaq
  - Esu 2 būsenoj
    - $p < 0.5$ 
      - Pereinu j 4 būsenaq
    - $p > 0.5$ 
      - Pereinu j 5 būsenaq
  - Esu 3 būsenoj
    - $p < 0.5$ 
      - Pereinu j 1 būsenaq
    - $p > 0.5$ 
      - Pereinu j 4 būsenaq
  - Esu 4 būsenoj
    - $p < 0.5$ 
      - Pereinu j 2 būsenaq
    - $p > 0.5$ 
      - Pereinu j 3 būsenaq
  - Esu 5 būsenoj
    - $p < 0.5$ 
      - Pereinu j 1 būsenaq
    - $p > 0.5$ 
      - Pereinu j 2 būsenaq

Generuosių penkis kartus vis pasirinkdamas kita startinę būseną.

Pradedu nuo v1

```
one = random_walk(seka = antra, start = 1, m = 729 )
print(one)
SortedDict(dict(Counter(one)))
```

```
SortedDict({1: 141, 2: 152, 3: 137, 4: 151, 5: 149})
```

Pradedu nuo v2

Pradedu nuo v3

Pradedu nuo v4

`SortListBy([1..140, 2..152, 3..158, 4..152, 5..148])`

Pradedu nuo v5

Matome kad pakeitus startinę būseną labai nežymiai kinta būsenų aplankymo dažniai.

# Kodas:

# In[1]:

```
from collections import Counter
from sortedcontainers import SortedDict
import numpy as np
import matplotlib.pyplot as plt
import math
from statistics import mean
from scipy.stats import chisquare

def uzpildyti_nulius(counts):
    app = {}
    for idx in range(min(counts), max(counts) + 1):
        val = counts.get(idx)
        if val == None:
            val = 0
        app[idx] = val
    return SortedDict(dict(app))

def LCG(seed, a, c, m):
    numbers = []
    for i in range(m):
        seed = (a * seed + c) % m
        numbers.append(seed)

    return numbers

def intervalu_testas(seq, m, a = 0.75, b = 1, t = 4):
    seq = [i/m for i in seq]
    intervals = []
    temp = []
    for num in seq:
        if num >= a and num <= b:
            temp.append(num)
        intervals.append(temp)
        temp = []
    else:
        temp.append(num)

    interval_lengths = [len(i)-1 for i in intervals]
    counts = SortedDict(dict(Counter(interval_lengths)))
    counts = uzpildyti_nulius(counts)
    n = sum(counts.values())
```

```

p = b-a

counts = {str(key): value for key, value in counts.items()}

print("n =", n, ", p =", p)
counts[">>" + str(t)] = 0
topop = []
for key, value in counts.items():
    if key != ">" + str(t) and int(key) > t:
        counts[">>" + str(t)] += value
        topop.append(key)

for key in topop:
    counts.pop(key, None)
print(counts)
pi = []
p = 0.25
n = sum(counts.values())
for key in counts:
    if key == ">" + str(t):
        pi.append((1-p)**int(t+1))
    else:
        pi.append(p*((1-p)**int(key)))

print("Pi: ", pi)
pi = [i*n for i in pi]
print("Tikimasi tokiu dažniu: ", pi)
print("Laisvės laipsniai: ", t+1)
print("Chi-Kvadrato kriterijus: ", chisquare(list(counts.values()), f_exp = pi))

def skaitmenu_testas(seka, m):
    counts = {0:0,1:0,2:0,3:0,4:0,5:0,6:0,7:0,8:0,9:0}
    seka = [i/1264 for i in seka]
    seka = [str(i).split(".")[1] for i in seka]
    firsth = [i+"0"*(5-len(i)) for i in seka if len(i)<5]
    secondh = [i for i in seka if len(i)>=5]
    seka = firsth + secondh
    seka = [i[:5] for i in seka]

    counts = []
    for i in seka:
        for letter in i:
            counts.append(int(letter))
    #print(counts)
    counts = Counter(counts)
    counts = SortedDict(dict(Counter(counts)))

```

```
print(counts)

n = sum(counts.values())
pi = [1/10] * 10
print("n = m*5 = ", m,"*5 = ", n)
print("Pi: ", pi)
pi = [int(round(i*n,0)) for i in pi]
print("Tikimasi tokiu dažnių: ", pi)
print("Chi-Kvadrato kriterijus: ", chisquare(list(counts.values()),f_exp = pi))
```

```
def generuoti_puasono(avg, seka, m):
    useka = [i/m for i in seka]
    p = math.exp(-avg)
    temp = []
    rez = []
    for i in useka:
        temp.append(i)
        if np.prod(temp) < p:
            rez.append(len(temp) - 1)
            temp = []
    return rez
```

```
def generuoti_geometrini(p, seka, m):
    useka = [i/729 for i in seka]
    geom = []
    for i in useka:
        if i != 0:
            geom.append(math.ceil(np.log(i)/np.log(0.9)))
    return geom
```

```
# In[2]:
```

```
from math import gcd
gcd(631,1264)
```

```
# In[3]:
```

```
from math import gcd
gcd(365,729)
```

```
# ## 1 užduotis
```

```
# In[4]:
```

```
pirma = LCG(seed = 0, a = 949, c = 631, m = 1264)
print(pirma)
```

```
# In[5]:
```

```
antra = LCG(seed = 0, a = 727, c = 365, m = 729)
print(antra)
```

```
# ## 2 užduotis
```

```
# In[6]:
```

```
print("Pirma seka:")
intervalu_testas(pirma, m = 1264, t = 4)
```

```
print("\n \n Antra seka:")
intervalu_testas(antra, m = 729, t = 4)
```

```
# In[7]:
```

```
print("Pirma seka:")
skaitmenu_testas(pirma, m = 1264)
```

```
print("\n\nAntra seka:")
skaitmenu_testas(antra, m = 729)
```

```
# ## 3 užduotis
```

```
# In[8]:
```

```
geom = generuoti_geometrini(p = 0.1, seka = antra, m = 729)
print(geom)
print("\nTeorinis vidurkis: ", 1/0.1)
print("Gautas vidurkis: ", mean(geom))
```

```
# In[9]:
```

```
poisson = generuoti_puasono(avg = 0.5, seka = antra, m = 729)
print(poisson)
print("\nVidurkis = ", mean(poisson))
```

```
# ## 4 užduotis
```

```
# In[10]:
```

```
useka = [i/729 for i in antra]
N = 729
```

```
# In[11]:
```

```
suma_1 = 0
epsilon_1 = [2*u for u in useka]
for i in epsilon_1:
    if i!= 0:
        suma_1+=(2/N)*(i+i**5)/(1+i)
print(suma_1)
```

```
# In[12]:
```

```
suma_2 = 0
epsilon_2 = [2*(u**(1/4)) for u in useka]
for i in epsilon_2:
    if i!= 0:
        suma_2+=(4/N)*(i+i**5)/((1+i)*(i**3))
print(suma_2)
```

```
# ## 5 užduotis
```

```
# In[13]:
```

```
nodes = ["v1","v2","v3","v4","v5"]
```

```
edges = [("v1","v3"),("v1","v5"),("v2","v4"),("v2","v5"),("v3","v4")]
n = len(nodes)
m = len(edges)
```

# In[14]:

```
nodes_neighbours = {}
for node in nodes:
    nodes_neighbours[node] = 0
    for edge in edges:
        if edge[0] == node or edge[1] == node:
            nodes_neighbours[node] += 1

print("Kaimynių viršunių kiekiai",nodes_neighbours)
```

# In[15]:

```
p_nodes_neighbours = {node:(1/nodes_neighbours[node]) for node in nodes}
print("Tikimybės pereiti į viršunes",p_nodes_neighbours)
```

# In[16]:

```
matrix = np.zeros((n, n))

for i in range(n):
    for j in range(n):
        if (nodes[i],nodes[j]) in edges:
            matrix[i][j] = p_nodes_neighbours[nodes[i]]
            matrix[j][i] = p_nodes_neighbours[nodes[j]]

print("Perėjimų matrica: \n",matrix)
```

# In[17]:

```
def random_walk(seka, start, m):
    uantra = [i/m for i in antra]
    state = start
    states = []
    states.append(state)
```

```
for p in uantra:
    if state == 1:
        if p < 0.5:
            state = 3
            states.append(state)
            continue
        if p > 0.5:
            state = 5
            states.append(state)
            continue
    if state == 2:
        if p < 0.5:
            state = 4
            states.append(state)
            continue
        if p > 0.5:
            state = 5
            states.append(state)
            continue
    if state == 3:
        if p < 0.5:
            state = 1
            states.append(state)
            continue
        if p > 0.5:
            state = 4
            states.append(state)
            continue
    if state == 4:
        if p < 0.5:
            state = 2
            states.append(state)
            continue
        if p > 0.5:
            state = 3
            states.append(state)
            continue
    if state == 5:
        if p < 0.5:
            state = 1
            states.append(state)
            continue
        if p > 0.5:
            state = 2
            states.append(state)
            continue
return states
```

```
# In[18]:
```

```
one = random_walk(seka = antra, start = 1, m = 729 )
print(one)
SortedDict(dict(Counter(one)))
```

```
# In[19]:
```

```
two = random_walk(seka = antra, start = 2, m = 729 )
print(two)
SortedDict(dict(Counter(two)))
```

```
# In[20]:
```

```
three = random_walk(seka = antra, start = 3, m = 729 )
print(three)
SortedDict(dict(Counter(three)))
```

```
# In[21]:
```

```
four = random_walk(seka = antra, start = 4, m = 729 )
print(four)
SortedDict(dict(Counter(four)))
```

```
# In[22]:
```

```
five = random_walk(seka = antra, start = 5, m = 729 )
print(five)
SortedDict(dict(Counter(five)))
```