**МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**им. Н.Э. БАУМАНА**

Факультет: Информатика и системы управления Кафедра: Информационная безопасность (ИУ8)

**ТЕОРИЯ ИГР И ИССЛЕДОВАНИЕ ОПЕРАЦИЙ**

**Лабораторная работа №2 на тему:**

«Решение матричных игр с нулевой суммой аналитическим

(матричным) и численным (Брауна–Робинсона) методами»

Вариант 4

**Преподаватель:**

Коннова Н.С.

**Студент**:

Куликова А.В.

**Группа:**

ИУ8-21М

**Цель работы**

Изучить аналитический (метод обратной матрицы) и численный (метод

Брауна–Робинсона) подходы к нахождению смешанных стратегий в

антагонистической игре двух лиц в нормальной форме.

**Постановка задачи**

Решите приведенную в варианте задания игру (найдите цену игры и оптимальные стратегии обоих игроков) методами обратной матрицы и Брауна-Робинсон. Сравните полученные результаты.

**Ход работы**

Матрица стратегий представлена в таблице 1, где строки соответствуют

стратегиям игрока A, столбцы – стратегиям игрока B. Будет выполнено N итераций численного метода до достижения заданной точности ɛ.

Таблица 1 – Матрица стратегий

|  |  |  |  |
| --- | --- | --- | --- |
| Стратегии | b1 | b2 | b3 |
| a1 | 17 | 4 | 9 |
| a2 | 0 | 16 | 9 |
| a3 | 12 | 2 | 19 |

Результат аналитического метода представлен в рисунке 1.

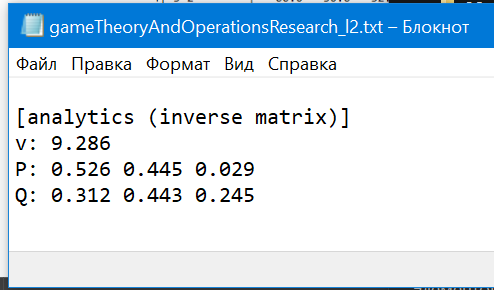


Рисунок 1 – Результат аналитического метода

Результат итерационного метода за первые 9 шагов представлен в Таблице 2.

Таблица 2 – Первые шаги итерационного метода алгоритма Брауна–Робинсон

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| №пп | Выбор  A | Выбор  B | Выигрыш A | | | Проигрыш B | | |  |  |  |
| x1 | x2 | x3 | y1 | y2 | y3 |
| 1 | x1 | y1 | 17 | 0 | 12 | 17 | 4 | 9 | 17.000 | 4.000 | 13.000 |
| 2 | x1 | y2 | 21 | 16 | 14 | 34 | 8 | 18 | 10.500 | 4.000 | 6.500 |
| 3 | x1 | y2 | 25 | 32 | 16 | 51 | 12 | 27 | 10.667 | 4.000 | 6.500 |
| 4 | x2 | y2 | 29 | 48 | 18 | 51 | 28 | 36 | 12.000 | 7.000 | 3.500 |
| 5 | x2 | y2 | 33 | 64 | 20 | 51 | 44 | 45 | 12.800 | 8.800 | 1.700 |
| 6 | x2 | y2 | 37 | 80 | 22 | 51 | 60 | 54 | 13.333 | 8.500 | 1.700 |
| 7 | x2 | y1 | 54 | 80 | 34 | 51 | 76 | 63 | 11.429 | 7.286 | 1.700 |
| 8 | x2 | y1 | 71 | 80 | 46 | 51 | 92 | 72 | 10.000 | 6.375 | 1.200 |
| 9 | x2 | y1 | 88 | 80 | 58 | 51 | 108 | 81 | 9.778 | 5.667 | 0.978 |

Результат итерационного метода представлен в рисунках 2 – 3.

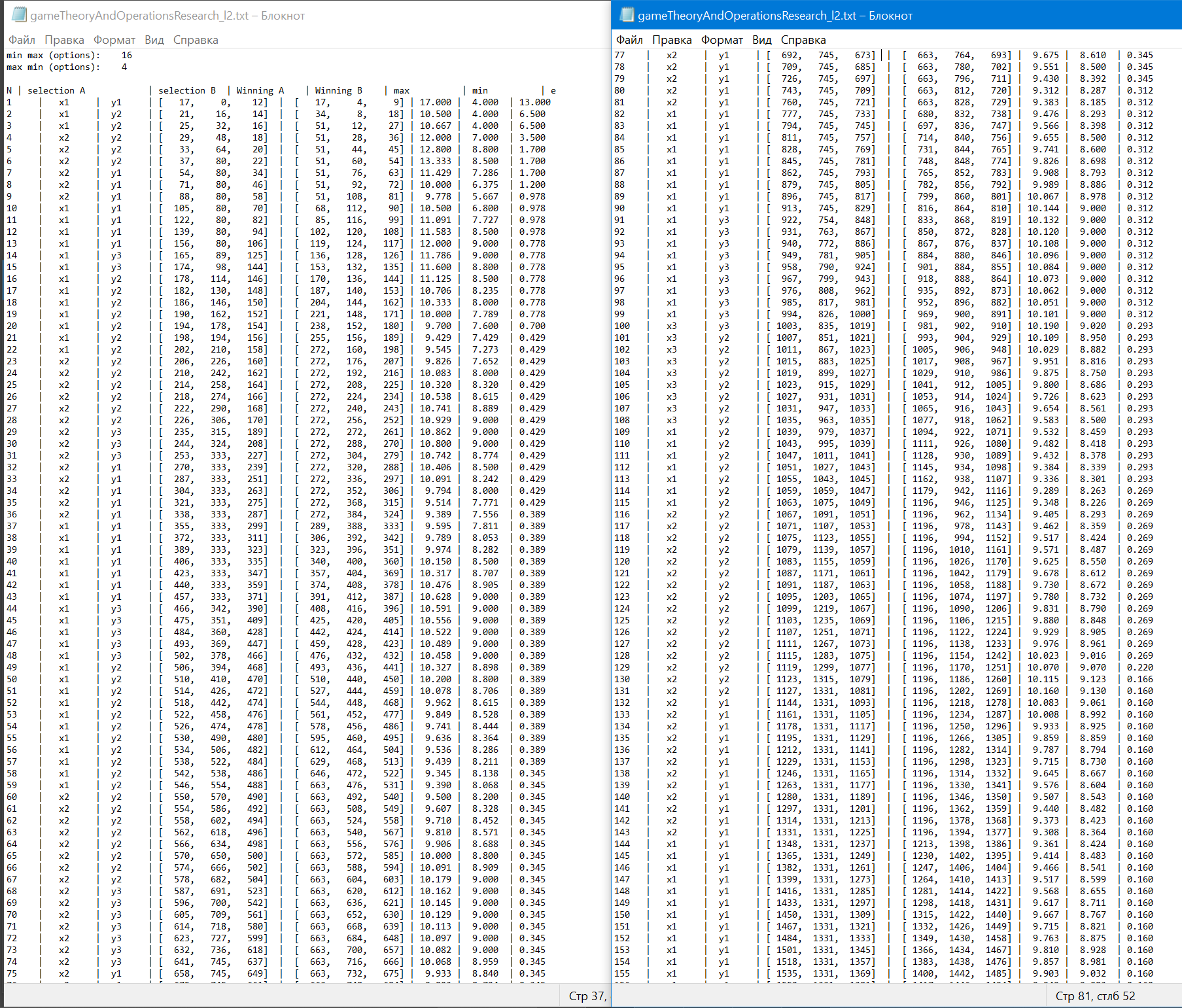


Рисунок 2 – Результат итерационного метода

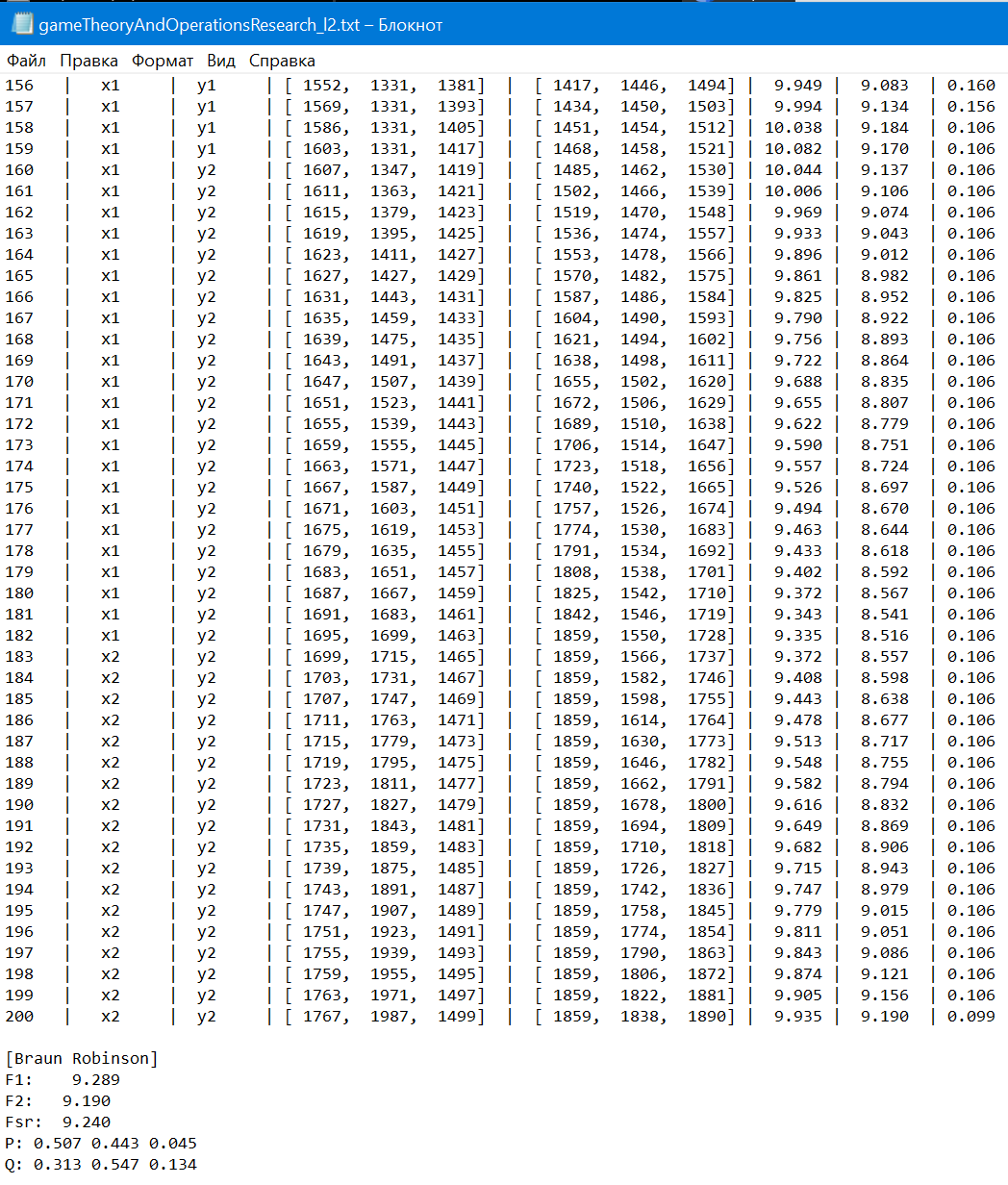


Рисунок 3 – Результат итерационного метода

Начиная с 158 итерации погрешность согласно меньше или равна 0.1. Выполнено всего 200 итераций численного метода до достижения заданной точности погрешности.

Результат всей программы представлен в приложении А.

**Выводы**

В ходе проделанной работы была изучена работа матричных игр с нулевой суммой аналитическим (матричным) и численным (Брауна–Робинсон) методами. Главным плюсом метода Браун-Робинсон является возможность получения максимально близкого значения к ошибке.

**Контрольные вопросы**

**1. Дайте определение смешанной стратегии.**

Смешанные стратегии — совокупность (комбинация) чистых стратегий A1, A2, … Am и B1, B2, …Bn в сочетании с векторами вероятностей выбора каждой из них.

**2. Что такое существенная матричная игра?**

Существенная матричная игра — чистая стратегия i ∈ A( j∈ B) игрока A(B), если существует оптимальная стратегия x\*=(x1\*, x2\*, ….. xm\*) ∈ Sm y\*=(y1\*, y2\*, ….. yn\*) ∈ Sn этого игрока, для которой xi\*> 0 (yj\*> 0).

**3. Каковы условия применимости аналитического метода нахождения смешанных стратегий?**

Если в игре, заданной платежной матрицей, отсутствует седловая точка и требуется решение игры в смешанных стратегиях, используется аналитический и различные численные методы решения.

**4. Какая основная идея итерационного метода нахождения смешанных стратегий?**

В игре заданной матрицей A размерности m x n каждое разыгрывание игры в чистых стратегиях будем далее называть партией.

Метод Брауна-Робинсон — это итеративная процедура построения последовательности пар смешанных стратегий игроков, сходящейся к решению матричной игры.

В 1 партии оба игрока выбирают произвольную чистую стратегию. Пусть сыграно k партий, причем выбор стратегии в каждой партии запоминается. В (k + 1) партии каждый игрок выбирает ту чистую стратегию, которая максимизирует его ожидаемый выигрыш, если противник играет в соответствии с эмпирическим вероятностным распределением, сформировавшимся за k партий.

Оценивается интервал для цены игры и, если он достаточно мал, процесс останавливается. Полученные при этом вероятностные распределения определяют смешанные стратегии игроков.

Достоинства метода Брауна-Робинсон:

* Этот метод ориентирован на произвольную игру G(m×n);
* Не требует условия aij>0;
* Легко реализуем программными методами.

Недостатки метода Брауна-Робинсон:

* С ростом размерности матрицы игры скорость сходимости метода быстро уменьшается.

**Приложение А**

Результат кода:

min max (options): 16

max min (options): 4

N | selection A | selection B | Winning A | Winning B | max | min | e

1 | x1 | y1 | [ 17, 0, 12] | [ 17, 4, 9] | 17.000 | 4.000 | 13.000

2 | x1 | y2 | [ 21, 16, 14] | [ 34, 8, 18] | 10.500 | 4.000 | 6.500

3 | x1 | y2 | [ 25, 32, 16] | [ 51, 12, 27] | 10.667 | 4.000 | 6.500

4 | x2 | y2 | [ 29, 48, 18] | [ 51, 28, 36] | 12.000 | 7.000 | 3.500

5 | x2 | y2 | [ 33, 64, 20] | [ 51, 44, 45] | 12.800 | 8.800 | 1.700

6 | x2 | y2 | [ 37, 80, 22] | [ 51, 60, 54] | 13.333 | 8.500 | 1.700

7 | x2 | y1 | [ 54, 80, 34] | [ 51, 76, 63] | 11.429 | 7.286 | 1.700

8 | x2 | y1 | [ 71, 80, 46] | [ 51, 92, 72] | 10.000 | 6.375 | 1.200

9 | x2 | y1 | [ 88, 80, 58] | [ 51, 108, 81] | 9.778 | 5.667 | 0.978

10 | x1 | y1 | [ 105, 80, 70] | [ 68, 112, 90] | 10.500 | 6.800 | 0.978

11 | x1 | y1 | [ 122, 80, 82] | [ 85, 116, 99] | 11.091 | 7.727 | 0.978

12 | x1 | y1 | [ 139, 80, 94] | [ 102, 120, 108] | 11.583 | 8.500 | 0.978

13 | x1 | y1 | [ 156, 80, 106] | [ 119, 124, 117] | 12.000 | 9.000 | 0.778

14 | x1 | y3 | [ 165, 89, 125] | [ 136, 128, 126] | 11.786 | 9.000 | 0.778

15 | x1 | y3 | [ 174, 98, 144] | [ 153, 132, 135] | 11.600 | 8.800 | 0.778

16 | x1 | y2 | [ 178, 114, 146] | [ 170, 136, 144] | 11.125 | 8.500 | 0.778

17 | x1 | y2 | [ 182, 130, 148] | [ 187, 140, 153] | 10.706 | 8.235 | 0.778

18 | x1 | y2 | [ 186, 146, 150] | [ 204, 144, 162] | 10.333 | 8.000 | 0.778

19 | x1 | y2 | [ 190, 162, 152] | [ 221, 148, 171] | 10.000 | 7.789 | 0.778

20 | x1 | y2 | [ 194, 178, 154] | [ 238, 152, 180] | 9.700 | 7.600 | 0.700

21 | x1 | y2 | [ 198, 194, 156] | [ 255, 156, 189] | 9.429 | 7.429 | 0.429

22 | x1 | y2 | [ 202, 210, 158] | [ 272, 160, 198] | 9.545 | 7.273 | 0.429

23 | x2 | y2 | [ 206, 226, 160] | [ 272, 176, 207] | 9.826 | 7.652 | 0.429

24 | x2 | y2 | [ 210, 242, 162] | [ 272, 192, 216] | 10.083 | 8.000 | 0.429

25 | x2 | y2 | [ 214, 258, 164] | [ 272, 208, 225] | 10.320 | 8.320 | 0.429

26 | x2 | y2 | [ 218, 274, 166] | [ 272, 224, 234] | 10.538 | 8.615 | 0.429

27 | x2 | y2 | [ 222, 290, 168] | [ 272, 240, 243] | 10.741 | 8.889 | 0.429

28 | x2 | y2 | [ 226, 306, 170] | [ 272, 256, 252] | 10.929 | 9.000 | 0.429

29 | x2 | y3 | [ 235, 315, 189] | [ 272, 272, 261] | 10.862 | 9.000 | 0.429

30 | x2 | y3 | [ 244, 324, 208] | [ 272, 288, 270] | 10.800 | 9.000 | 0.429

31 | x2 | y3 | [ 253, 333, 227] | [ 272, 304, 279] | 10.742 | 8.774 | 0.429

32 | x2 | y1 | [ 270, 333, 239] | [ 272, 320, 288] | 10.406 | 8.500 | 0.429

33 | x2 | y1 | [ 287, 333, 251] | [ 272, 336, 297] | 10.091 | 8.242 | 0.429

34 | x2 | y1 | [ 304, 333, 263] | [ 272, 352, 306] | 9.794 | 8.000 | 0.429

35 | x2 | y1 | [ 321, 333, 275] | [ 272, 368, 315] | 9.514 | 7.771 | 0.429

36 | x2 | y1 | [ 338, 333, 287] | [ 272, 384, 324] | 9.389 | 7.556 | 0.389

37 | x1 | y1 | [ 355, 333, 299] | [ 289, 388, 333] | 9.595 | 7.811 | 0.389

38 | x1 | y1 | [ 372, 333, 311] | [ 306, 392, 342] | 9.789 | 8.053 | 0.389

39 | x1 | y1 | [ 389, 333, 323] | [ 323, 396, 351] | 9.974 | 8.282 | 0.389

40 | x1 | y1 | [ 406, 333, 335] | [ 340, 400, 360] | 10.150 | 8.500 | 0.389

41 | x1 | y1 | [ 423, 333, 347] | [ 357, 404, 369] | 10.317 | 8.707 | 0.389

42 | x1 | y1 | [ 440, 333, 359] | [ 374, 408, 378] | 10.476 | 8.905 | 0.389

43 | x1 | y1 | [ 457, 333, 371] | [ 391, 412, 387] | 10.628 | 9.000 | 0.389

44 | x1 | y3 | [ 466, 342, 390] | [ 408, 416, 396] | 10.591 | 9.000 | 0.389

45 | x1 | y3 | [ 475, 351, 409] | [ 425, 420, 405] | 10.556 | 9.000 | 0.389

46 | x1 | y3 | [ 484, 360, 428] | [ 442, 424, 414] | 10.522 | 9.000 | 0.389

47 | x1 | y3 | [ 493, 369, 447] | [ 459, 428, 423] | 10.489 | 9.000 | 0.389

48 | x1 | y3 | [ 502, 378, 466] | [ 476, 432, 432] | 10.458 | 9.000 | 0.389

49 | x1 | y2 | [ 506, 394, 468] | [ 493, 436, 441] | 10.327 | 8.898 | 0.389

50 | x1 | y2 | [ 510, 410, 470] | [ 510, 440, 450] | 10.200 | 8.800 | 0.389

51 | x1 | y2 | [ 514, 426, 472] | [ 527, 444, 459] | 10.078 | 8.706 | 0.389

52 | x1 | y2 | [ 518, 442, 474] | [ 544, 448, 468] | 9.962 | 8.615 | 0.389

53 | x1 | y2 | [ 522, 458, 476] | [ 561, 452, 477] | 9.849 | 8.528 | 0.389

54 | x1 | y2 | [ 526, 474, 478] | [ 578, 456, 486] | 9.741 | 8.444 | 0.389

55 | x1 | y2 | [ 530, 490, 480] | [ 595, 460, 495] | 9.636 | 8.364 | 0.389

56 | x1 | y2 | [ 534, 506, 482] | [ 612, 464, 504] | 9.536 | 8.286 | 0.389

57 | x1 | y2 | [ 538, 522, 484] | [ 629, 468, 513] | 9.439 | 8.211 | 0.389

58 | x1 | y2 | [ 542, 538, 486] | [ 646, 472, 522] | 9.345 | 8.138 | 0.345

59 | x1 | y2 | [ 546, 554, 488] | [ 663, 476, 531] | 9.390 | 8.068 | 0.345

60 | x2 | y2 | [ 550, 570, 490] | [ 663, 492, 540] | 9.500 | 8.200 | 0.345

61 | x2 | y2 | [ 554, 586, 492] | [ 663, 508, 549] | 9.607 | 8.328 | 0.345

62 | x2 | y2 | [ 558, 602, 494] | [ 663, 524, 558] | 9.710 | 8.452 | 0.345

63 | x2 | y2 | [ 562, 618, 496] | [ 663, 540, 567] | 9.810 | 8.571 | 0.345

64 | x2 | y2 | [ 566, 634, 498] | [ 663, 556, 576] | 9.906 | 8.688 | 0.345

65 | x2 | y2 | [ 570, 650, 500] | [ 663, 572, 585] | 10.000 | 8.800 | 0.345

66 | x2 | y2 | [ 574, 666, 502] | [ 663, 588, 594] | 10.091 | 8.909 | 0.345

67 | x2 | y2 | [ 578, 682, 504] | [ 663, 604, 603] | 10.179 | 9.000 | 0.345

68 | x2 | y3 | [ 587, 691, 523] | [ 663, 620, 612] | 10.162 | 9.000 | 0.345

69 | x2 | y3 | [ 596, 700, 542] | [ 663, 636, 621] | 10.145 | 9.000 | 0.345

70 | x2 | y3 | [ 605, 709, 561] | [ 663, 652, 630] | 10.129 | 9.000 | 0.345

71 | x2 | y3 | [ 614, 718, 580] | [ 663, 668, 639] | 10.113 | 9.000 | 0.345

72 | x2 | y3 | [ 623, 727, 599] | [ 663, 684, 648] | 10.097 | 9.000 | 0.345

73 | x2 | y3 | [ 632, 736, 618] | [ 663, 700, 657] | 10.082 | 9.000 | 0.345

74 | x2 | y3 | [ 641, 745, 637] | [ 663, 716, 666] | 10.068 | 8.959 | 0.345

75 | x2 | y1 | [ 658, 745, 649] | [ 663, 732, 675] | 9.933 | 8.840 | 0.345

76 | x2 | y1 | [ 675, 745, 661] | [ 663, 748, 684] | 9.803 | 8.724 | 0.345

77 | x2 | y1 | [ 692, 745, 673] | [ 663, 764, 693] | 9.675 | 8.610 | 0.345

78 | x2 | y1 | [ 709, 745, 685] | [ 663, 780, 702] | 9.551 | 8.500 | 0.345

79 | x2 | y1 | [ 726, 745, 697] | [ 663, 796, 711] | 9.430 | 8.392 | 0.345

80 | x2 | y1 | [ 743, 745, 709] | [ 663, 812, 720] | 9.312 | 8.287 | 0.312

81 | x2 | y1 | [ 760, 745, 721] | [ 663, 828, 729] | 9.383 | 8.185 | 0.312

82 | x1 | y1 | [ 777, 745, 733] | [ 680, 832, 738] | 9.476 | 8.293 | 0.312

83 | x1 | y1 | [ 794, 745, 745] | [ 697, 836, 747] | 9.566 | 8.398 | 0.312

84 | x1 | y1 | [ 811, 745, 757] | [ 714, 840, 756] | 9.655 | 8.500 | 0.312

85 | x1 | y1 | [ 828, 745, 769] | [ 731, 844, 765] | 9.741 | 8.600 | 0.312

86 | x1 | y1 | [ 845, 745, 781] | [ 748, 848, 774] | 9.826 | 8.698 | 0.312

87 | x1 | y1 | [ 862, 745, 793] | [ 765, 852, 783] | 9.908 | 8.793 | 0.312

88 | x1 | y1 | [ 879, 745, 805] | [ 782, 856, 792] | 9.989 | 8.886 | 0.312

89 | x1 | y1 | [ 896, 745, 817] | [ 799, 860, 801] | 10.067 | 8.978 | 0.312

90 | x1 | y1 | [ 913, 745, 829] | [ 816, 864, 810] | 10.144 | 9.000 | 0.312

91 | x1 | y3 | [ 922, 754, 848] | [ 833, 868, 819] | 10.132 | 9.000 | 0.312

92 | x1 | y3 | [ 931, 763, 867] | [ 850, 872, 828] | 10.120 | 9.000 | 0.312

93 | x1 | y3 | [ 940, 772, 886] | [ 867, 876, 837] | 10.108 | 9.000 | 0.312

94 | x1 | y3 | [ 949, 781, 905] | [ 884, 880, 846] | 10.096 | 9.000 | 0.312

95 | x1 | y3 | [ 958, 790, 924] | [ 901, 884, 855] | 10.084 | 9.000 | 0.312

96 | x1 | y3 | [ 967, 799, 943] | [ 918, 888, 864] | 10.073 | 9.000 | 0.312

97 | x1 | y3 | [ 976, 808, 962] | [ 935, 892, 873] | 10.062 | 9.000 | 0.312

98 | x1 | y3 | [ 985, 817, 981] | [ 952, 896, 882] | 10.051 | 9.000 | 0.312

99 | x1 | y3 | [ 994, 826, 1000] | [ 969, 900, 891] | 10.101 | 9.000 | 0.312

100 | x3 | y3 | [ 1003, 835, 1019] | [ 981, 902, 910] | 10.190 | 9.020 | 0.293

101 | x3 | y2 | [ 1007, 851, 1021] | [ 993, 904, 929] | 10.109 | 8.950 | 0.293

102 | x3 | y2 | [ 1011, 867, 1023] | [ 1005, 906, 948] | 10.029 | 8.882 | 0.293

103 | x3 | y2 | [ 1015, 883, 1025] | [ 1017, 908, 967] | 9.951 | 8.816 | 0.293

104 | x3 | y2 | [ 1019, 899, 1027] | [ 1029, 910, 986] | 9.875 | 8.750 | 0.293

105 | x3 | y2 | [ 1023, 915, 1029] | [ 1041, 912, 1005] | 9.800 | 8.686 | 0.293

106 | x3 | y2 | [ 1027, 931, 1031] | [ 1053, 914, 1024] | 9.726 | 8.623 | 0.293

107 | x3 | y2 | [ 1031, 947, 1033] | [ 1065, 916, 1043] | 9.654 | 8.561 | 0.293

108 | x3 | y2 | [ 1035, 963, 1035] | [ 1077, 918, 1062] | 9.583 | 8.500 | 0.293

109 | x1 | y2 | [ 1039, 979, 1037] | [ 1094, 922, 1071] | 9.532 | 8.459 | 0.293

110 | x1 | y2 | [ 1043, 995, 1039] | [ 1111, 926, 1080] | 9.482 | 8.418 | 0.293

111 | x1 | y2 | [ 1047, 1011, 1041] | [ 1128, 930, 1089] | 9.432 | 8.378 | 0.293

112 | x1 | y2 | [ 1051, 1027, 1043] | [ 1145, 934, 1098] | 9.384 | 8.339 | 0.293

113 | x1 | y2 | [ 1055, 1043, 1045] | [ 1162, 938, 1107] | 9.336 | 8.301 | 0.293

114 | x1 | y2 | [ 1059, 1059, 1047] | [ 1179, 942, 1116] | 9.289 | 8.263 | 0.269

115 | x1 | y2 | [ 1063, 1075, 1049] | [ 1196, 946, 1125] | 9.348 | 8.226 | 0.269

116 | x2 | y2 | [ 1067, 1091, 1051] | [ 1196, 962, 1134] | 9.405 | 8.293 | 0.269

117 | x2 | y2 | [ 1071, 1107, 1053] | [ 1196, 978, 1143] | 9.462 | 8.359 | 0.269

118 | x2 | y2 | [ 1075, 1123, 1055] | [ 1196, 994, 1152] | 9.517 | 8.424 | 0.269

119 | x2 | y2 | [ 1079, 1139, 1057] | [ 1196, 1010, 1161] | 9.571 | 8.487 | 0.269

120 | x2 | y2 | [ 1083, 1155, 1059] | [ 1196, 1026, 1170] | 9.625 | 8.550 | 0.269

121 | x2 | y2 | [ 1087, 1171, 1061] | [ 1196, 1042, 1179] | 9.678 | 8.612 | 0.269

122 | x2 | y2 | [ 1091, 1187, 1063] | [ 1196, 1058, 1188] | 9.730 | 8.672 | 0.269

123 | x2 | y2 | [ 1095, 1203, 1065] | [ 1196, 1074, 1197] | 9.780 | 8.732 | 0.269

124 | x2 | y2 | [ 1099, 1219, 1067] | [ 1196, 1090, 1206] | 9.831 | 8.790 | 0.269

125 | x2 | y2 | [ 1103, 1235, 1069] | [ 1196, 1106, 1215] | 9.880 | 8.848 | 0.269

126 | x2 | y2 | [ 1107, 1251, 1071] | [ 1196, 1122, 1224] | 9.929 | 8.905 | 0.269

127 | x2 | y2 | [ 1111, 1267, 1073] | [ 1196, 1138, 1233] | 9.976 | 8.961 | 0.269

128 | x2 | y2 | [ 1115, 1283, 1075] | [ 1196, 1154, 1242] | 10.023 | 9.016 | 0.269

129 | x2 | y2 | [ 1119, 1299, 1077] | [ 1196, 1170, 1251] | 10.070 | 9.070 | 0.220

130 | x2 | y2 | [ 1123, 1315, 1079] | [ 1196, 1186, 1260] | 10.115 | 9.123 | 0.166

131 | x2 | y2 | [ 1127, 1331, 1081] | [ 1196, 1202, 1269] | 10.160 | 9.130 | 0.160

132 | x2 | y1 | [ 1144, 1331, 1093] | [ 1196, 1218, 1278] | 10.083 | 9.061 | 0.160

133 | x2 | y1 | [ 1161, 1331, 1105] | [ 1196, 1234, 1287] | 10.008 | 8.992 | 0.160

134 | x2 | y1 | [ 1178, 1331, 1117] | [ 1196, 1250, 1296] | 9.933 | 8.925 | 0.160

135 | x2 | y1 | [ 1195, 1331, 1129] | [ 1196, 1266, 1305] | 9.859 | 8.859 | 0.160

136 | x2 | y1 | [ 1212, 1331, 1141] | [ 1196, 1282, 1314] | 9.787 | 8.794 | 0.160

137 | x2 | y1 | [ 1229, 1331, 1153] | [ 1196, 1298, 1323] | 9.715 | 8.730 | 0.160

138 | x2 | y1 | [ 1246, 1331, 1165] | [ 1196, 1314, 1332] | 9.645 | 8.667 | 0.160

139 | x2 | y1 | [ 1263, 1331, 1177] | [ 1196, 1330, 1341] | 9.576 | 8.604 | 0.160

140 | x2 | y1 | [ 1280, 1331, 1189] | [ 1196, 1346, 1350] | 9.507 | 8.543 | 0.160

141 | x2 | y1 | [ 1297, 1331, 1201] | [ 1196, 1362, 1359] | 9.440 | 8.482 | 0.160

142 | x2 | y1 | [ 1314, 1331, 1213] | [ 1196, 1378, 1368] | 9.373 | 8.423 | 0.160

143 | x2 | y1 | [ 1331, 1331, 1225] | [ 1196, 1394, 1377] | 9.308 | 8.364 | 0.160

144 | x1 | y1 | [ 1348, 1331, 1237] | [ 1213, 1398, 1386] | 9.361 | 8.424 | 0.160

145 | x1 | y1 | [ 1365, 1331, 1249] | [ 1230, 1402, 1395] | 9.414 | 8.483 | 0.160

146 | x1 | y1 | [ 1382, 1331, 1261] | [ 1247, 1406, 1404] | 9.466 | 8.541 | 0.160

147 | x1 | y1 | [ 1399, 1331, 1273] | [ 1264, 1410, 1413] | 9.517 | 8.599 | 0.160

148 | x1 | y1 | [ 1416, 1331, 1285] | [ 1281, 1414, 1422] | 9.568 | 8.655 | 0.160

149 | x1 | y1 | [ 1433, 1331, 1297] | [ 1298, 1418, 1431] | 9.617 | 8.711 | 0.160

150 | x1 | y1 | [ 1450, 1331, 1309] | [ 1315, 1422, 1440] | 9.667 | 8.767 | 0.160

151 | x1 | y1 | [ 1467, 1331, 1321] | [ 1332, 1426, 1449] | 9.715 | 8.821 | 0.160

152 | x1 | y1 | [ 1484, 1331, 1333] | [ 1349, 1430, 1458] | 9.763 | 8.875 | 0.160

153 | x1 | y1 | [ 1501, 1331, 1345] | [ 1366, 1434, 1467] | 9.810 | 8.928 | 0.160

154 | x1 | y1 | [ 1518, 1331, 1357] | [ 1383, 1438, 1476] | 9.857 | 8.981 | 0.160

155 | x1 | y1 | [ 1535, 1331, 1369] | [ 1400, 1442, 1485] | 9.903 | 9.032 | 0.160

156 | x1 | y1 | [ 1552, 1331, 1381] | [ 1417, 1446, 1494] | 9.949 | 9.083 | 0.160

157 | x1 | y1 | [ 1569, 1331, 1393] | [ 1434, 1450, 1503] | 9.994 | 9.134 | 0.156

158 | x1 | y1 | [ 1586, 1331, 1405] | [ 1451, 1454, 1512] | 10.038 | 9.184 | 0.106

159 | x1 | y1 | [ 1603, 1331, 1417] | [ 1468, 1458, 1521] | 10.082 | 9.170 | 0.106

160 | x1 | y2 | [ 1607, 1347, 1419] | [ 1485, 1462, 1530] | 10.044 | 9.137 | 0.106

161 | x1 | y2 | [ 1611, 1363, 1421] | [ 1502, 1466, 1539] | 10.006 | 9.106 | 0.106

162 | x1 | y2 | [ 1615, 1379, 1423] | [ 1519, 1470, 1548] | 9.969 | 9.074 | 0.106

163 | x1 | y2 | [ 1619, 1395, 1425] | [ 1536, 1474, 1557] | 9.933 | 9.043 | 0.106

164 | x1 | y2 | [ 1623, 1411, 1427] | [ 1553, 1478, 1566] | 9.896 | 9.012 | 0.106

165 | x1 | y2 | [ 1627, 1427, 1429] | [ 1570, 1482, 1575] | 9.861 | 8.982 | 0.106

166 | x1 | y2 | [ 1631, 1443, 1431] | [ 1587, 1486, 1584] | 9.825 | 8.952 | 0.106

167 | x1 | y2 | [ 1635, 1459, 1433] | [ 1604, 1490, 1593] | 9.790 | 8.922 | 0.106

168 | x1 | y2 | [ 1639, 1475, 1435] | [ 1621, 1494, 1602] | 9.756 | 8.893 | 0.106

169 | x1 | y2 | [ 1643, 1491, 1437] | [ 1638, 1498, 1611] | 9.722 | 8.864 | 0.106

170 | x1 | y2 | [ 1647, 1507, 1439] | [ 1655, 1502, 1620] | 9.688 | 8.835 | 0.106

171 | x1 | y2 | [ 1651, 1523, 1441] | [ 1672, 1506, 1629] | 9.655 | 8.807 | 0.106

172 | x1 | y2 | [ 1655, 1539, 1443] | [ 1689, 1510, 1638] | 9.622 | 8.779 | 0.106

173 | x1 | y2 | [ 1659, 1555, 1445] | [ 1706, 1514, 1647] | 9.590 | 8.751 | 0.106

174 | x1 | y2 | [ 1663, 1571, 1447] | [ 1723, 1518, 1656] | 9.557 | 8.724 | 0.106

175 | x1 | y2 | [ 1667, 1587, 1449] | [ 1740, 1522, 1665] | 9.526 | 8.697 | 0.106

176 | x1 | y2 | [ 1671, 1603, 1451] | [ 1757, 1526, 1674] | 9.494 | 8.670 | 0.106

177 | x1 | y2 | [ 1675, 1619, 1453] | [ 1774, 1530, 1683] | 9.463 | 8.644 | 0.106

178 | x1 | y2 | [ 1679, 1635, 1455] | [ 1791, 1534, 1692] | 9.433 | 8.618 | 0.106

179 | x1 | y2 | [ 1683, 1651, 1457] | [ 1808, 1538, 1701] | 9.402 | 8.592 | 0.106

180 | x1 | y2 | [ 1687, 1667, 1459] | [ 1825, 1542, 1710] | 9.372 | 8.567 | 0.106

181 | x1 | y2 | [ 1691, 1683, 1461] | [ 1842, 1546, 1719] | 9.343 | 8.541 | 0.106

182 | x1 | y2 | [ 1695, 1699, 1463] | [ 1859, 1550, 1728] | 9.335 | 8.516 | 0.106

183 | x2 | y2 | [ 1699, 1715, 1465] | [ 1859, 1566, 1737] | 9.372 | 8.557 | 0.106

184 | x2 | y2 | [ 1703, 1731, 1467] | [ 1859, 1582, 1746] | 9.408 | 8.598 | 0.106

185 | x2 | y2 | [ 1707, 1747, 1469] | [ 1859, 1598, 1755] | 9.443 | 8.638 | 0.106

186 | x2 | y2 | [ 1711, 1763, 1471] | [ 1859, 1614, 1764] | 9.478 | 8.677 | 0.106

187 | x2 | y2 | [ 1715, 1779, 1473] | [ 1859, 1630, 1773] | 9.513 | 8.717 | 0.106

188 | x2 | y2 | [ 1719, 1795, 1475] | [ 1859, 1646, 1782] | 9.548 | 8.755 | 0.106

189 | x2 | y2 | [ 1723, 1811, 1477] | [ 1859, 1662, 1791] | 9.582 | 8.794 | 0.106

190 | x2 | y2 | [ 1727, 1827, 1479] | [ 1859, 1678, 1800] | 9.616 | 8.832 | 0.106

191 | x2 | y2 | [ 1731, 1843, 1481] | [ 1859, 1694, 1809] | 9.649 | 8.869 | 0.106

192 | x2 | y2 | [ 1735, 1859, 1483] | [ 1859, 1710, 1818] | 9.682 | 8.906 | 0.106

193 | x2 | y2 | [ 1739, 1875, 1485] | [ 1859, 1726, 1827] | 9.715 | 8.943 | 0.106

194 | x2 | y2 | [ 1743, 1891, 1487] | [ 1859, 1742, 1836] | 9.747 | 8.979 | 0.106

195 | x2 | y2 | [ 1747, 1907, 1489] | [ 1859, 1758, 1845] | 9.779 | 9.015 | 0.106

196 | x2 | y2 | [ 1751, 1923, 1491] | [ 1859, 1774, 1854] | 9.811 | 9.051 | 0.106

197 | x2 | y2 | [ 1755, 1939, 1493] | [ 1859, 1790, 1863] | 9.843 | 9.086 | 0.106

198 | x2 | y2 | [ 1759, 1955, 1495] | [ 1859, 1806, 1872] | 9.874 | 9.121 | 0.106

199 | x2 | y2 | [ 1763, 1971, 1497] | [ 1859, 1822, 1881] | 9.905 | 9.156 | 0.106

200 | x2 | y2 | [ 1767, 1987, 1499] | [ 1859, 1838, 1890] | 9.935 | 9.190 | 0.099

[Braun Robinson]

F1: 9.289

F2: 9.190

Fsr: 9.240

P: 0.507 0.443 0.045

Q: 0.313 0.547 0.134

[analytics (inverse matrix)]

v: 9.286

P: 0.526 0.445 0.029

Q: 0.312 0.443 0.245

**Приложение Б**

Листинг Б.1 — maxmin.h

#ifndef MAXMIN\_H

#define MAXMIN\_H

#include <iostream>

#include <iomanip>

#include <fstream>

namespace gameTheoryAndOperationsResearch {

    class gameTheoryAndOperationsResearch\_maxmin

    {

        public:

            double max\_min(double \*\*, int, int, int&);

            double min\_max(double \*\*, int, int, int&);

    };

}

#endif // MAXMIN\_H

Продолжение приложения Б

Листинг Б.2 — maxmin.cpp

#include "maxmin.h"

namespace gameTheoryAndOperationsResearch {

    // Метод max\_min находит максимум из минимальных значений по строкам

    double gameTheoryAndOperationsResearch\_maxmin::max\_min(double \*\*Matr, int n, int m, int& iMax)

    {

        double min;

        double max = 0;

        for (int i = 0; i < n; i++)

        {

            min = Matr[i][0];

            // Находим минимальное значение в текущей строке

            for (int j = 1; j < m; j++)

                if (min > Matr[i][j])

                    min = Matr[i][j];

            // Сравниваем минимальное значение с максимальным

            if (max < min) {

                max = min;

                iMax = i; // Сохраняем индекс строки с максимальным минимальным значением

            }

        }

        return max; // Возвращаем максимальное из минимальных значений

    }

    // Метод min\_max находит минимум из максимальных значений по столбцам

    double gameTheoryAndOperationsResearch\_maxmin::min\_max(double \*\*Matr, int n, int m, int& iMin)

    {

        double min = 9e99; // Инициализация переменной min большим значением

        double max;

        for (int j = 0; j < m; j++)

        {

            max = Matr[0][j];

            // Находим максимальное значение в текущем столбце

            for (int i = 1; i < n; i++)

                if (max < Matr[i][j])

                    max = Matr[i][j];

            // Сравниваем максимальное значение с минимальным

            if (max < min) {

                min = max;

                iMin = j; // Сохраняем индекс столбца с минимальным максимальным значением

            }

        }

        return min; // Возвращаем минимум из максимальных значений

    }

}

Продолжение приложения Б

Листинг Б.3 — vector.h

#ifndef VECTOR\_H

#define VECTOR\_H

#include <iostream>

#include <iomanip>

#include <fstream>

namespace gameTheoryAndOperationsResearch {

    class gameTheoryAndOperationsResearch\_vector

    {

        public:

            void product\_vector\_matrix(double \*, double\*\*, double\*, int, int);

            void product\_matrix\_vector(double \*\*, double\*,  double\*, int, int);

            double product\_vector\_vector(double \*, double\*, int);

    };

}

#endif // VECTOR\_H

Продолжение приложения Б

Листинг Б.4 — vector.cpp

#include "vector.h"

namespace gameTheoryAndOperationsResearch {

    // Произведение вектора на матрицу

    // Функция умножает вектор на матрицу и записывает результат в выходной вектор

    void gameTheoryAndOperationsResearch\_vector::product\_vector\_matrix(double \*v, double \*\*matr, double\* v\_out, int n, int m)

    {

        for (int i = 0; i < m; i++)

        {

            v\_out[i] = 0; // Инициализация i-го элемента выходного вектора

            for (int j = 0; j < n; j++)

                v\_out[i] += v[j] \* matr[j][i]; // Умножение элементов вектора на элементы матрицы и суммирование

        }

    }

    // Произведение матрицы на вектор

    // Функция умножает матрицу на вектор и записывает результат в выходной вектор

    void gameTheoryAndOperationsResearch\_vector::product\_matrix\_vector(double \*\*matr, double \*v,  double\* v\_out, int n, int m)

    {

        for (int i = 0; i < n; i++)

        {

            v\_out[i] = 0; // Инициализация i-го элемента выходного вектора

            for (int j = 0; j < m; j++)

                v\_out[i] += v[j] \* matr[i][j]; // Умножение элементов строки матрицы на элементы вектора и суммирование

        }

    }

    // Произведение вектор на вектор - на выходе число (скалярное произведение)

    // Функция вычисляет скалярное произведение двух векторов

    double gameTheoryAndOperationsResearch\_vector::product\_vector\_vector(double \*v1, double \*v2, int n)

    {

        double rez = 0; // Инициализация результата

        for (int i = 0; i < n; i++)

            rez += v1[i] \* v2[i]; // Умножение соответствующих элементов двух векторов и суммирование

        return rez; // Возвращаем полученное скалярное произведение

    }

}

Продолжение приложения Б

Листинг Б.5 — print.h

#ifndef PRINT\_H

#define PRINT\_H

#include <iostream>

#include <iomanip>

#include <fstream>

namespace gameTheoryAndOperationsResearch {

    class gameTheoryAndOperationsResearch\_print

    {

        public:

            void print\_vector(std::ostream&, char\*, double\*, int);

    };

}

#endif // PRINT\_H

Продолжение приложения Б

Листинг Б.5 — print.cpp

#include "print.h"

namespace gameTheoryAndOperationsResearch {

    // Метод print\_vector выводит вектор на экран

    void gameTheoryAndOperationsResearch\_print::print\_vector(std::ostream& out, char \* str, double \*p, int n)

    {

        out.precision(3); // Устанавливаем точность вывода чисел

        out << str << ": "; // Выводим строку-метку для вектора

        for (int i = 0; i < n; i++)

            out << p[i] << " "; // Выводим элементы вектора через пробел

        out << std::endl; // Переходим на новую строку после вывода всех элементов

    }

}

Продолжение приложения Б

Листинг Б.6 — braun\_robinson.h

#ifndef BRAUN\_ROBINSON\_H

#define BRAUN\_ROBINSON\_H

#include <iostream>

#include <iomanip>

#include <fstream>

namespace gameTheoryAndOperationsResearch {

    class gameTheoryAndOperationsResearch\_braun\_robinson

    {

        public:

        void braun\_robinson(double\*\*, double \*, double \*, int, int, double&, double&, std::ostream&);

    };

}

#endif // BRAUN\_ROBINSON\_H

Продолжение приложения Б

Листинг Б.7 — braun\_robinson.cpp

#include "braun\_robinson.h"

#include "maxmin.h"

#include <memory.h>

#include <cmath>

namespace gameTheoryAndOperationsResearch {

    gameTheoryAndOperationsResearch::gameTheoryAndOperationsResearch\_maxmin \_braun\_robinson\_maxmin;

    // Классичекий метод Брауна-Робинсона для матрицы игры

    void gameTheoryAndOperationsResearch\_braun\_robinson::braun\_robinson(

            double \*\*pM, double \*p, double \*q, int n, int m, double& vmin, double& vmax, std::ostream & fout)

    {

        int \*pX = new int[n];           // Частоты по строкам

        int \*pY = new int[m];           // Частоты по столбцам

        int k = 1;                      // Общее число выбора строк и столбцов

        double \*pV1 = new double[n];    // Суммарный выигрыш 1-го игрока

        double \*pV2 = new double[m];    // Суммарный выигрыш 2-го игрока

        for (int i = 0; i < n; i++)

           pV1[i] = pX[i] = 0;

        for (int i = 0; i < m; i++)

           pV2[i] = pY[i] = 0;

        fout << std::fixed << std::setprecision(2);

        int iMax, iMin;

        fout << "N" << std::setprecision(0) << std::left << std::setw(25)

            << " | selection A" << std::left << std::setw(15)

            << " | selection B" << std::left << std::setw(15)

            << " | Winning A" << std::left << std::setw(15)

            << " | Winning B" << std::left << std::setw(15)

            << " | max" << std::left << std::setw(15)

            << " | min" << std::left << std::setw(15)

            << " | e" << std::endl;

        // Первые ходы по минмаксу и максимину

        \_braun\_robinson\_maxmin.max\_min(pM, n, m, iMax);

        \_braun\_robinson\_maxmin.min\_max(pM, n, m, iMin);

        iMin = iMax = 0;

        double vminmax = 9e99,

               vmaxmin = -9e99;

        do

        {

            fout << std::setprecision(0) << std::left << std::setw(5) << k << " | "

                << "  x" << std::left << std::setw(5) << (iMax + 1) << " | " << " y" << std::setw(5) << (iMin + 1) << " | [";

            // Выигрыш 1-го игрока

            for (int i = 0; i < n; i++) {

                pV1[i] += pM[i][iMin];

                fout << std::right << std::setw(5) << pV1[i];

                if (i != n - 1)

                    fout << ", ";

                else

                    fout << "]  |  [";

           }

           // Проигрыш 2-го игрока

           for (int i = 0; i < m; i++)

           {

                pV2[i] += pM[iMax][i];

                fout << std::setw(5) << pV2[i];

                if (i != m - 1)

                    fout << ", ";

                else

                    fout << "] | ";

           }

           double min = 9e99; // Выбор второго игрока

           for (int i = 0; i < m; i++) {

               if (pV2[i] < min) {

                   min = pV2[i];

                   iMin = i;

               }

           }

           pY[iMin]++;

           vmin = min / k; // Нижняя цена игры

           if (vmin > vmaxmin)

               vmaxmin = vmin;

           // Выбор первого игрока

           double max = -9e99;

           for (int i = 0; i < n; i++) {

               if (pV1[i] > max) {

                   max = pV1[i];

                   iMax = i;

               }

           }

           pX[iMax]++;

           vmax = max / k; // Верхняя цена игры

           if (vmax < vminmax)

               vminmax = vmax;

           k++;

           fout << std::setprecision(3) << std::setw(6) << vmax << " | "

                << std::setw(6) << vmin << "  | " << fabs(vminmax - vmaxmin) << std::endl;

        } while (fabs(vminmax - vmaxmin) > 0.1); // Условие остановки

        vmin = vminmax; vmax = vmaxmin;

        // Расчет оценок вероятностей

        for (int i = 0; i < n; i++)

           p[i] = (double)pX[i] / (k);

        for (int i = 0; i < m; i++)

           q[i] = (double)pY[i] / (k);

        delete[]pX;

        delete[]pY;

        delete[]pV1;

        delete[]pV2;

    }

}

Продолжение приложения Б

Листинг Б.8 — analytical\_inverse\_matrix\_method.h

#ifndef ANALYTICAL\_INVERSE\_MATRIX\_METHOD\_H

#define ANALYTICAL\_INVERSE\_MATRIX\_METHOD\_H

#include <iostream>

#include <iomanip>

#include <fstream>

namespace gameTheoryAndOperationsResearch {

    class gameTheoryAndOperationsResearch\_analytical\_inverse\_matrix\_method

    {

        public:

            double determinant\_matrix(double\*\*, int);

            void matrix\_inverse\_matrix(double\*\*, double\*\*, int);

    };

}

#endif // ANALYTICAL\_INVERSE\_MATRIX\_METHOD\_H

Продолжение приложения Б

Листинг Б.9 — analytical\_inverse\_matrix\_method.cpp

#include "analytical\_inverse\_matrix\_method.h"

#include <memory.h>

#include <cmath>

namespace gameTheoryAndOperationsResearch {

    // функции для аналитического метода обратной матрицы

    // Функция для вычисления определителя матрицы

    double gameTheoryAndOperationsResearch\_analytical\_inverse\_matrix\_method::determinant\_matrix(double \*\*p, int n)

    {

        if (n == 1)

            return p[0][0]; // Если размер матрицы равен 1, возвращаем элемент

        if (n == 2)

            return p[0][0] \* p[1][1] - p[0][1] \* p[1][0]; // Если размер матрицы равен 2, используем формулу для определителя 2x2 матрицы

        else

        {

            double \*\*p2 = new double \*[n - 1]; // Создаем временную матрицу меньшего размера

            for (int i = 0; i < n - 1; i++)

                p2[i] = new double[n - 1];

            int zn = 1; // Знак определителя

            double Det = 0; // Инициализируем определитель

            for (int k = 0; k < n; k++, zn \*= -1)

            {

                if (p[0][k] == 0)

                    continue; // Пропускаем нулевые элементы

                for (int i = 1; i < n; i++)

                    for (int j = 0; j < n; j++)

                    {

                        if (j < k)

                            p2[i - 1][j] = p[i][j]; // Заполняем временную матрицу

                        if (j > k)

                            p2[i - 1][j - 1] = p[i][j]; // Заполняем временную матрицу

                    }

                Det += zn \* p[0][k] \* determinant\_matrix(p2, n - 1); // Рекурсивно вычисляем определитель

            }

            for (int i = 0; i < n - 1; i++)

                delete[] p2[i]; // Освобождаем память временной матрицы

            delete[] p2; // Освобождаем память временной матрицы

            return Det; // Возвращаем определитель

        }

    }

    // Функция для вычисления обратной матрицы для матрицы

    void gameTheoryAndOperationsResearch\_analytical\_inverse\_matrix\_method::matrix\_inverse\_matrix(double \*\*p, double \*\*pObr, int n)

    {

        double Det = 0;

        // Если размер матрицы равен 2, используем простую формулу для нахождения обратной матрицы

        if (n == 2)

        {

            Det = p[0][0] \* p[1][1] - p[0][1] \* p[1][0];

            pObr[0][0] = p[1][1];

            pObr[0][1] = p[1][0] \* -1;

            pObr[1][0] = p[0][1] \* -1;

            pObr[1][1] = p[0][0];

        }

        else

        {

            double \*\*p2 = new double \*[n - 1]; // Создаем временную матрицу меньшего размера

            for (int i = 0; i < n - 1; i++)

                p2[i] = new double[n - 1];

            int zn = 1; // Знак определителя

            Det = 0;

            // Вычисляем обратную матрицу методом алгебраических дополнений

            for (int k0 = 0; k0 < n; k0++)

            {

                for (int k = 0; k < n; k++)

                {

                    if ((k + k0) % 2)

                        zn = -1;

                    else

                        zn = 1;

                    for (int i = 0; i < n; i++)

                        for (int j = 0; j < n; j++)

                        {

                            if (i > k0)

                            {

                                if (j < k)

                                    p2[i - 1][j] = p[i][j];

                                if (j > k)

                                    p2[i - 1][j - 1] = p[i][j];

                            }

                            if (i < k0)

                            {

                                if (j < k)

                                    p2[i][j] = p[i][j];

                                if (j > k)

                                    p2[i][j - 1] = p[i][j];

                            }

                        }

                    double opr = determinant\_matrix(p2, n - 1);

                    pObr[k0][k] = zn \* opr;

                    if (k0 == 0)

                        Det += p[k0][k] \* pObr[k0][k];

                }

            }

            // Освобождаем память временной матрицы

            for (int i = 0; i < n - 1; i++)

                delete[] p2[i];

            delete[] p2;

        }

        // Транспонируем матрицу обратной матрицы

        for (int i = 0; i < n - 1; i++)

            for (int j = i + 1; j < n; j++) {

                double buf = pObr[i][j];

                pObr[i][j] = pObr[j][i];

                pObr[j][i] = buf;

            }

        // Делим все элементы обратной матрицы на определитель и округляем значения

        for (int i = 0; i < n; i++)

            for (int j = 0; j < n; j++) {

                pObr[i][j] /= Det;

                if (fabs(pObr[i][j]) < 1e-9)

                    pObr[i][j] = 0;

            }

    }

}

Продолжение приложения Б

Листинг Б.10 — main.define.h

#ifndef MAIN\_DEFINE\_H

#define MAIN\_DEFINE\_H

#define n (int)(3)

#define m (int)(3)

#define gameTheoryAndOperationsResearch\_filename "gameTheoryAndOperationsResearch\_l2.txt"

#include <iostream>

#include <fstream>

#include <string>

//#define \_gameTheoryAndOperationsResearch\_readfile () (std::string line;std::ifstream in(gameTheoryAndOperationsResearch\_filename);while (std::getline(in, line))std::cout << line << std::endl)

//#define \_gameTheoryAndOperationsResearch\_deletefile () (std::remove(gameTheoryAndOperationsResearch\_filename))

#endif // MAIN\_DEFINE\_H

Продолжение приложения Б

Листинг Б.11 — main.cpp

#include <iostream>

#include <iomanip>

#include <fstream>

#include "maxmin.h"

#include "vector.h"

#include "print.h"

#include "braun\_robinson.h"

#include "analytical\_inverse\_matrix\_method.h"

#include "main.define.h"

namespace gameTheoryAndOperationsResearch {

    void \_gameTheoryAndOperationsResearch\_readfile() {

        std::string line;std::ifstream in(gameTheoryAndOperationsResearch\_filename);

        while (std::getline(in, line))

            std::cout << line << std::endl;

    }

    void \_gameTheoryAndOperationsResearch\_deletefile() {

        std::remove(gameTheoryAndOperationsResearch\_filename);

    }

    gameTheoryAndOperationsResearch::gameTheoryAndOperationsResearch\_analytical\_inverse\_matrix\_method \_gameTheoryAndOperationsResearch\_analytical\_inverse\_matrix\_method;

    gameTheoryAndOperationsResearch::gameTheoryAndOperationsResearch\_braun\_robinson \_gameTheoryAndOperationsResearch\_braun\_robinson;

    gameTheoryAndOperationsResearch::gameTheoryAndOperationsResearch\_maxmin \_gameTheoryAndOperationsResearch\_maxmin;

    gameTheoryAndOperationsResearch::gameTheoryAndOperationsResearch\_print \_gameTheoryAndOperationsResearch\_print;

    gameTheoryAndOperationsResearch::gameTheoryAndOperationsResearch\_vector \_gameTheoryAndOperationsResearch\_vector;

    int \_main(int argc, char\* argv[])

    {

        // Удаляем файл

        \_gameTheoryAndOperationsResearch\_deletefile();

        // Выделяем память под матрицу a размером n x m

        double \*\*a = new double \*[n];

        for (int i = 0; i < n; i++)

            a[i] = new double[m];

        int i;

        std::ofstream fout(gameTheoryAndOperationsResearch\_filename);

        // Инициализируем матрицу a:

        //      17   4   9

        //      0    16  9

        //      12   2   19

        a[0][0] = 17;       a[0][1] = 4;        a[0][2] = 9;

        a[1][0] = 0;        a[1][1] = 16;       a[1][2] = 9;

        a[2][0] = 12;       a[2][1] = 2;        a[2][2] = 19;

        // Вычисляем min max и max min

        fout << "min max (options):    " << \_gameTheoryAndOperationsResearch\_maxmin.min\_max(a, n, m, i) << std::endl;

        fout << "max min (options):    " << \_gameTheoryAndOperationsResearch\_maxmin.max\_min(a, n, m, i) << std::endl;

        fout << std::endl;

        // Выделяем память под векторы p и q

        double \*p = new double[n];

        double \*q = new double[m];

        double F1, F2;

        // Применяем метод Брауна-Робинсона

        \_gameTheoryAndOperationsResearch\_braun\_robinson.braun\_robinson(a, p, q, n, m, F1, F2, fout);

        fout << std::endl << "[Braun Robinson]" << std::endl;

        fout << "F1:    " << F1 << std::endl

             << "F2:   " << F2 << std::endl

             << "Fsr:  " << (F1 + F2) / 2 << std::endl;

        // Печатаем векторы p и q

        \_gameTheoryAndOperationsResearch\_print.print\_vector(fout, (char \*)"P", p, n);

        \_gameTheoryAndOperationsResearch\_print.print\_vector(fout, (char \*)"Q", q, m);

        fout << std::endl << "[analytics (inverse matrix)]" << std::endl;

        // Выделяем память под матрицу c размером n x m

        double \*\*c = new double \*[n];

        for (int i = 0; i < n; i++)

            c[i] = new double[m];

        // Получаем обратную матрицу

        \_gameTheoryAndOperationsResearch\_analytical\_inverse\_matrix\_method.matrix\_inverse\_matrix(a, c, n);

        // Создаем вектор u из всех единиц

        double \* u = new double[n];

        for (int i = 0;

             i < n; i++) u[i] = 1;

        // Вычисляем v и обновляем векторы p и q

        double \*r1 = new double[n]; // Вектор для промежуточных вычислений

        \_gameTheoryAndOperationsResearch\_vector.product\_vector\_matrix(u, c, r1, n, n);

        double v = \_gameTheoryAndOperationsResearch\_vector.product\_vector\_vector(r1, u, n);

        \_gameTheoryAndOperationsResearch\_vector.product\_vector\_matrix(u, c, p, n, n);

        for (int i = 0; i < n; i++)

            p[i] /= v;

        \_gameTheoryAndOperationsResearch\_vector.product\_matrix\_vector(c, u, q, n, m);

        for (int i = 0; i < n; i++)

            q[i] /= v;

        fout << "v: " << 1 / v << std::endl;

        // Печатаем обновленные векторы p и q

        \_gameTheoryAndOperationsResearch\_print.print\_vector(fout, (char \*)"P", p, n);

        \_gameTheoryAndOperationsResearch\_print.print\_vector(fout, (char \*)"Q", q, m);

        // Считываем файл

        \_gameTheoryAndOperationsResearch\_readfile();

        return 0;

    }

}

int main(int argc, char\* argv[])

{

    gameTheoryAndOperationsResearch::\_main(argc, argv);

    return 0;

}

Продолжение приложения Б

Листинг Б.12 — CMakeLists.txt

cmake\_minimum\_required(VERSION 2.8)

add\_executable(main

    maxmin.h    maxmin.cpp

    vector.h    vector.cpp

    print.h     print.cpp

    braun\_robinson.h    braun\_robinson.cpp

    analytical\_inverse\_matrix\_method.h  analytical\_inverse\_matrix\_method.cpp

    main.define.h

    main.cpp

    )

**Приложение В**

Ссылка на исходный код: https://github.com/Kulikova-A18/gameTheoryAndOperationsResearch\_lab2