Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления» (ИУ)
КАФЕДРА «Информационная безопасность» (ИУ8)

Лабораторная работа №1 на тему "Аналитический и численный (Брауна - Робинсон) методы решения антагонистической игры в смешанных стратегиях"

по дисциплине «Теория игр и исследование операций»

Вариант 12

Студент ИУ8-104 (Группа) Мильченко И. Д. (И. О. Фамилия) _____ (Подпись, дата)

Преподаватель Коннова Н. С. (И. О. Фамилия) _____ (Подпись, дата)

Москва, 2025 г.

ЦЕЛЬ РАБОТЫ

Изучить аналитический (обратной матрицы) и численный (Брауна–Робинсон) методы нахождения смешанных стратегий в антагонистической игре двух лиц в нормальной форме.

Задание

Найти цену игры и оптимальные стратегии обоих игроков методами обратной матрицы и Брауна–Робинсон. Сравнить полученные результаты.

$$\begin{pmatrix} 11 & 10 & 15 \\ 16 & 5 & 13 \\ 15 & 20 & 10 \end{pmatrix}$$

ХОД РАБОТЫ

Для решения данной лабораторной работы был использован язык программирования Go. В проекте был написан класс для решения матричных игр разными способами (аналитическим и Брауна–Робинсон соответственно). На вход программе подается файл формата JSON с матрицей.

Пример запуска программы:

```
go run cmd/lw1/main.go tasks/lw1/task.json
```

Для данной задачи была построена игровая матрица с расчитанными минимальным выигрышом игрока A и максимальным проигрышем игрока B.

strategies	b_1	b_2	b_3	min win of a player
a_1	11	10	15	10
a_2	16	5	13	5
a_3	15	20	10	10
max loss of B player	16	20	15	

```
Lowest Price: 10
Highest Price: 15
```

Рисунок 1 – Вывод игровой матрицы

Как видно на рисунке 1, нижняя цена игры не совпадает с верхней, что говорит об отсутствии седловой точки.

Перейдем к аналитическому методу решения данной игры.

1 Аналитический метод

Теорема 1.1. Вполне смешанная игра $(m \times n)$-игра Γ имеет единственную ситуацию равновесия (x^*, y^*) и квадратную матрицу $(m = n)$; если цена игры $v \neq 0$, то матрица C невырожденная и

$$x^* = \frac{uC^{-1}}{uC^{-1}u^T}, \quad y^* = \frac{C^{-1}u^T}{uC^{-1}u^T}, \quad v = \frac{1}{uC^{-1}u^T}, \tag{1}$$

где вектор $u = (1, 1, \ldots, 1) \in \mathbb{R}^m$.

Используя теорему 1.1, посчитаем цену игры и стратегии для обоих игроков согласно формулам 1.

Вывод программы с посчитанным результатом приведен на рисунке 2.

```
Analytical solution:
x* = [ 0.53  0.114  0.356]
y* = [0.341  0.129   0.53]
v=12.992
```

Рисунок 2 – Ответ для задачи, посчитанный аналитическим методом

Далее решим игру численным методом Брауна–Робинсон.

2 Численный метод Брауна–Робинсон

Метод Брауна—Робинсона является итерационным численным методом для приближенного нахождения решения матричных антагонистических игр. Этот метод позволяет находить оптимальные стратегии игроков в смешанных стратегиях, постепенно улучшая оценки верхней и нижней цены игры.

Метод заключается в последовательном обновлении частот выбора стратегий игроками. На каждом шаге k игроки выбирают стратегии, основываясь на накопленной статистике выигрышей:

Первый игрок выбирает стратегию, максимизирующую ожидаемый выигрыш, исходя из наблюденных действий второго игрока.

Второй игрок выбирает стратегию, минимизирующую ожидаемый выигрыш первого игрока.

Оценки верхней и нижней границы цены игры вычисляются по формулам:

$$\overline{v}[k] = \max_{i \in A} \sum_{j \in B} c_{ij} \tilde{y}_j[k], \tag{2}$$

$$\underline{v}[k] = \min \sum_{i \in A} c_{ij} \tilde{x}_i[k]. \tag{3}$$

Средние значения этих оценок используются для приближенного вычисления цены игры. Процесс продолжается до достижения заданной точности $\varepsilon \leq 0.1$.

На таблице 1 представлена таблица итеративного алгоритма Брауна–Робинсон для решения исходной матричной игры. Для решения всего потребовалось 108 итераций.

Таблица 1 – Результаты итерационного процесса с указанием стратегий и стоимостей игры

#	a	b	x_1	x_2	x_3	y_1	y_2	y_3	Верх.	Ниж.	ε
1	x_1	y_1	11	16	15	11	10	15	16.000	10.000	6.000
2	x_2	y_2	21	21	35	27	15	28	17.500	7.500	6.000
3	x_3	y_2	31	26	55	42	35	38	18.333	11.667	4.333
4	x_3	y_2	41	31	75	57	55	48	18.750	12.000	4.000
5	x_3	y_3	56	44	85	72	75	58	17.000	11.600	4.000
6	x_3	y_3	71	57	95	87	95	68	15.833	11.333	3.833

Продолжение таблицы 1

7	x_3	y_3	86	70	105	102	115	78	15.000	11.143	3.000
8	x_3	y_3	101	83	115	117	135	88	14.375	11.000	2.375
9	x_3	y_3	116	96	125	132	155	98	13.889	10.889	1.889
10	x_3	y_3	131	109	135	147	175	108	13.500	10.800	1.500
11	x_3	y_3	146	122	145	162	195	118	13.273	10.727	1.273
12	x_1	y_3	161	135	155	173	205	133	13.417	11.083	1.273
13	x_1	y_3	176	148	165	184	215	148	13.538	11.385	1.273
14	x_1	y_3	191	161	175	195	225	163	13.643	11.643	1.273
15	x_1	y_3	206	174	185	206	235	178	13.733	11.867	1.273
16	x_1	y_3	221	187	195	217	245	193	13.812	12.062	1.210
17	x_1	y_3	236	200	205	228	255	208	13.882	12.235	1.037
18	x_1	y_3	251	213	215	239	265	223	13.944	12.389	0.884
19	x_1	y_3	266	226	225	250	275	238	14.000	12.526	0.746
20	x_1	y_3	281	239	235	261	285	253	14.050	12.650	0.623
21	x_1	y_3	296	252	245	272	295	268	14.095	12.762	0.511
22	x_1	y_3	311	265	255	283	305	283	14.136	12.864	0.409
23	x_1	y_3	326	278	265	294	315	298	14.174	12.783	0.409
24	x_1	y_1	337	294	280	305	325	313	14.042	12.708	0.409
25	x_1	y_1	348	310	295	316	335	328	13.920	12.640	0.409
26	x_1	y_1	359	326	310	327	345	343	13.808	12.577	0.409
27	x_1	y_1	370	342	325	338	355	358	13.704	12.519	0.409
28	x_1	y_1	381	358	340	349	365	373	13.607	12.464	0.409
29	x_1	y_1	392	374	355	360	375	388	13.517	12.414	0.409
30	x_1	y_1	403	390	370	371	385	403	13.433	12.367	0.409
31	x_1	y_1	414	406	385	382	395	418	13.355	12.323	0.409
32	x_1	y_1	425	422	400	393	405	433	13.281	12.281	0.409
33	x_1	y_1	436	438	415	404	415	448	13.273	12.242	0.409
34	x_2	y_1	447	454	430	420	420	461	13.353	12.353	0.409
35	x_2	y_2	457	459	450	436	425	474	13.114	12.143	0.251
36	x_2	y_2	467	464	470	452	430	487	13.056	11.944	0.192
37	x_3	y_2	477	469	490	467	450	497	13.243	12.162	0.192
38	x_3	y_2	487	474	510	482	470	507	13.421	12.368	0.192

Продолжение таблицы 1

39	x_3	y_2	497	479	530	497	490	517	13.590	12.564	0.192
40	x_3	y_2	507	484	550	512	510	527	13.750	12.750	0.192
41	x_3	y_2	517	489	570	527	530	537	13.902	12.854	0.192
42	x_3	y_1	528	505	585	542	550	547	13.929	12.905	0.151
43	x_3	y_1	539	521	600	557	570	557	13.953	12.953	0.102
44	x_3	y_1	550	537	615	572	590	567	13.977	12.886	0.102
45	x_3	y_3	565	550	625	587	610	577	13.889	12.822	0.102
46	x_3	y_3	580	563	635	602	630	587	13.804	12.761	0.102
47	x_3	y_3	595	576	645	617	650	597	13.723	12.702	0.102
48	x_3	y_3	610	589	655	632	670	607	13.646	12.646	0.102
49	x_3	y_3	625	602	665	647	690	617	13.571	12.592	0.102
50	x_3	y_3	640	615	675	662	710	627	13.500	12.540	0.102
51	x_3	y_3	655	628	685	677	730	637	13.431	12.490	0.102
52	x_3	y_3	670	641	695	692	750	647	13.365	12.442	0.102
53	x_3	y_3	685	654	705	707	770	657	13.302	12.396	0.102
54	x_3	y_3	700	667	715	722	790	667	13.241	12.352	0.102
55	x_3	y_3	715	680	725	737	810	677	13.182	12.309	0.102
56	x_3	y_3	730	693	735	752	830	687	13.125	12.268	0.102
57	x_3	y_3	745	706	745	767	850	697	13.070	12.228	0.102
58	x_3	y_3	760	719	755	782	870	707	13.103	12.190	0.102
59	x_1	y_3	775	732	765	793	880	722	13.136	12.237	0.102
60	x_1	y_3	790	745	775	804	890	737	13.167	12.283	0.102
61	x_1	y_3	805	758	785	815	900	752	13.197	12.328	0.102
62	x_1	y_3	820	771	795	826	910	767	13.226	12.371	0.102
63	x_1	y_3	835	784	805	837	920	782	13.254	12.413	0.102
64	x_1	y_3	850	797	815	848	930	797	13.281	12.453	0.102
65	x_1	y_3	865	810	825	859	940	812	13.308	12.492	0.102
66	x_1	y_3	880	823	835	870	950	827	13.333	12.530	0.102
67	x_1	y_3	895	836	845	881	960	842	13.358	12.567	0.102
68	x_1	y_3	910	849	855	892	970	857	13.382	12.603	0.102
69	x_1	y_3	925	862	865	903	980	872	13.406	12.638	0.102
70	x_1	y_3	940	875	875	914	990	887	13.429	12.671	0.102

Продолжение таблицы 1

71	x_1	y_3	955	888	885	925	1000	902	13.451	12.704	0.102
72	x_1	y_3	970	901	895	936	1010	917	13.472	12.736	0.102
73	x_1	y_3	985	914	905	947	1020	932	13.493	12.767	0.102
74	x_1	y_3	1000	927	915	958	1030	947	13.514	12.797	0.102
75	x_1	y_3	1015	940	925	969	1040	962	13.533	12.827	0.102
76	x_1	y_3	1030	953	935	980	1050	977	13.553	12.855	0.102
77	x_1	y_3	1045	966	945	991	1060	992	13.571	12.870	0.102
78	x_1	y_1	1056	982	960	1002	1070	1007	13.538	12.846	0.102
79	x_1	y_1	1067	998	975	1013	1080	1022	13.506	12.823	0.102
80	x_1	y_1	1078	1014	990	1024	1090	1037	13.475	12.800	0.102
81	x_1	y_1	1089	1030	1005	1035	1100	1052	13.444	12.778	0.102
82	x_1	y_1	1100	1046	1020	1046	1110	1067	13.415	12.756	0.102
83	x_1	y_1	1111	1062	1035	1057	1120	1082	13.386	12.735	0.102
84	x_1	y_1	1122	1078	1050	1068	1130	1097	13.357	12.714	0.102
85	x_1	y_1	1133	1094	1065	1079	1140	1112	13.329	12.694	0.102
86	x_1	y_1	1144	1110	1080	1090	1150	1127	13.302	12.674	0.102
87	x_1	y_1	1155	1126	1095	1101	1160	1142	13.276	12.655	0.102
88	x_1	y_1	1166	1142	1110	1112	1170	1157	13.250	12.636	0.102
89	x_1	y_1	1177	1158	1125	1123	1180	1172	13.225	12.618	0.102
90	x_1	y_1	1188	1174	1140	1134	1190	1187	13.200	12.600	0.102
91	x_1	y_1	1199	1190	1155	1145	1200	1202	13.176	12.582	0.102
92	x_1	y_1	1210	1206	1170	1156	1210	1217	13.152	12.565	0.102
93	x_1	y_1	1221	1222	1185	1167	1220	1232	13.140	12.548	0.102
94	x_2	y_1	1232	1238	1200	1183	1225	1245	13.170	12.585	0.102
95	x_2	y_1	1243	1254	1215	1199	1230	1258	13.200	12.621	0.102
96	x_2	y_1	1254	1270	1230	1215	1235	1271	13.229	12.656	0.102
97	x_2	y_1	1265	1286	1245	1231	1240	1284	13.258	12.691	0.102
98	x_2	y_1	1276	1302	1260	1247	1245	1297	13.286	12.704	0.102
99	x_2	y_2	1286	1307	1280	1263	1250	1310	13.202	12.626	0.102
100	x_2	y_2	1296	1312	1300	1279	1255	1323	13.120	12.550	0.102
101	x_2	y_2	1306	1317	1320	1295	1260	1336	13.069	12.475	0.102
102	x_3	y_2	1316	1322	1340	1310	1280	1346	13.137	12.549	0.102

Продолжение таблицы 1

103	x_3	y_2	1326	1327	1360	1325	1300	1356	13.204	12.621	0.102
104	x_3	y_2	1336	1332	1380	1340	1320	1366	13.269	12.692	0.102
105	x_3	y_2	1346	1337	1400	1355	1340	1376	13.333	12.762	0.102
106	x_3	y_2	1356	1342	1420	1370	1360	1386	13.396	12.830	0.102
107	x_3	y_2	1366	1347	1440	1385	1380	1396	13.458	12.897	0.102
108	x_3	y_2	1376	1352	1460	1400	1400	1406	13.519	12.963	0.093

Алгоритм остановился, так как значение ε стало меньше 0.1. Значит ответом будут являться значения, представленные на рисунке 3.

```
x* = (19.000, 4.000, 12.667)
y* = (11.667, 6.667, 17.333)
v = 13.009
```

Рисунок 3 – Ответ для задачи, посчитанный алгоритмом Брауна–Робинсон

3 Сравщнительная оценка погрешностей

Сравним результаты, полученные аналитическим методом и алогоритмом Баруна–Робинсон. На таблице 2 представлены смешанные стратегии обоих игроков, цены игр, абсолютная и относительная погрешности.

Таблица 2 – Сводная таблица сравнительной оценки погрешностей

	Цена игры	a_1	a_2	a_3	b_1	b_2	b_3
Аналитический (матричный) метод	12.992	0.530	0.114	0.356	0.341	0.129	0.530
Численный метод Брауна-Робинсон	13.009	19.000	4.000	12.667	11.667	6.667	17.333
Абсолютная погрешность, Δ	0.017	18.470	3.886	12.311	11.326	6.538	16.803
Относительная погрешность, %	0.13	3484.9	3410.5	3458.1	3321.1	5069.8	3171.3

Абсолютная погрешность считается по формуле 4:

$$\Delta = |X_{\text{численный}} - X_{\text{аналитический}}| \tag{4}$$

Относительная погрешность считается по формуле 5:

$$\varepsilon = \frac{\Delta}{X_{\text{аналитический}}} \times 100\% \tag{5}$$

Графики сходимости приближенных значений цен игры и оценки погрешности приведены на рисуках 4 и 5 соответственно.

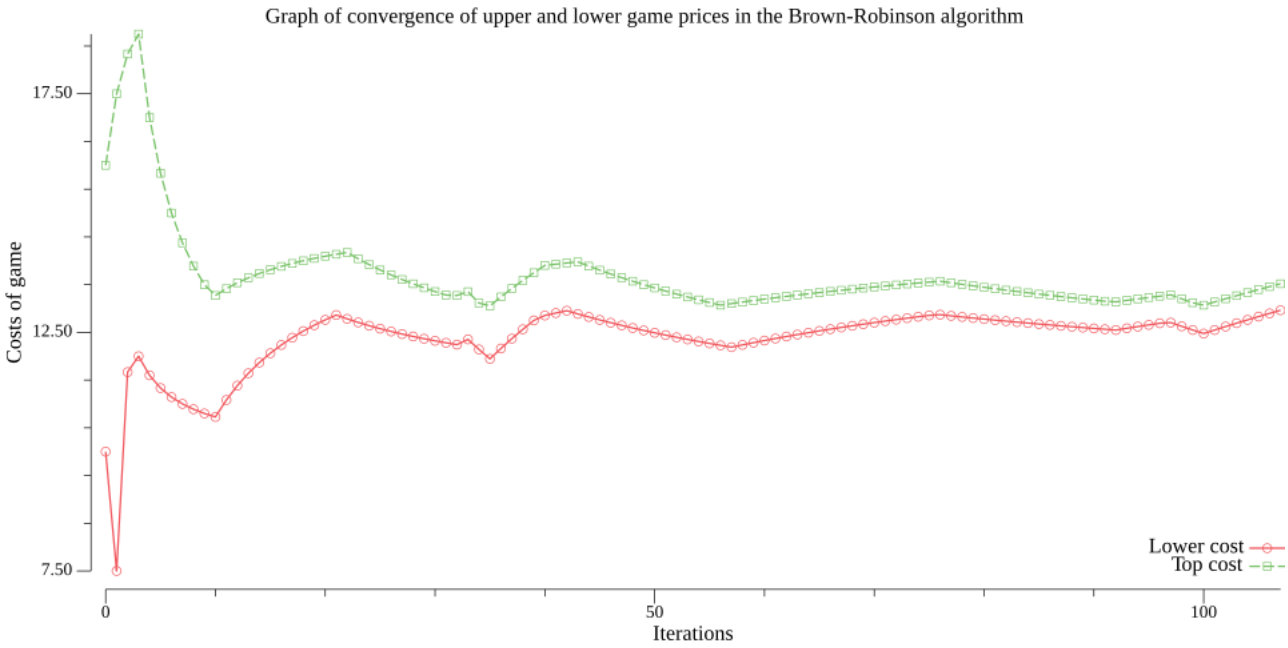

Рисунок 4 – График сходимости верхней и нижней цен игры в алгоритме Брауна–Робинсон

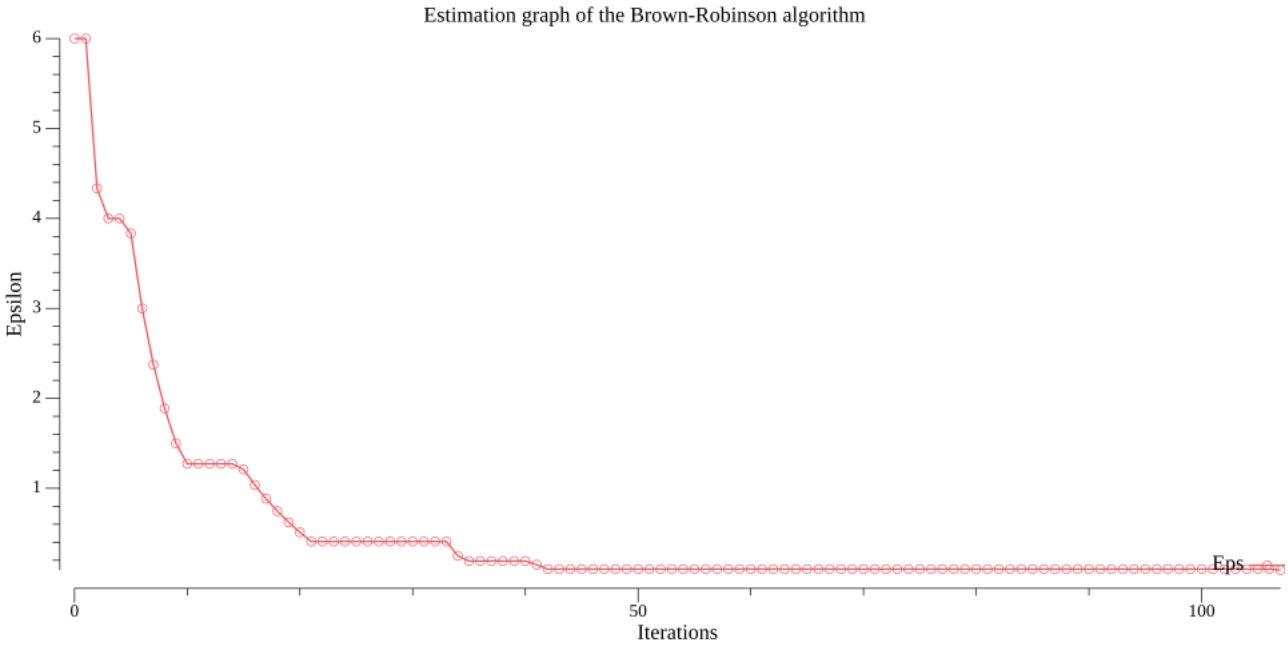

Рисунок 5 – График оценки погрешности алгоритма Брауна–Робинсон

ЗАКЛЮЧЕНИЕ

В данной работе было исследовано решение антагонистической игры двух лиц с нулевой суммой с использованием аналитического (метода обратной матрицы) и численного (метода Брауна–Робинсон) подходов.

$$
\begin{aligned}
v &= 12{,}992 \\
x^* &= (0{,}530,\ 0{,}114,\ 0{,}356) \\
y^* &= (0{,}341,\ 0{,}129,\ 0{,}530)
\end{aligned}
$$

Недостатки метода включают требования к квадратности и невырожденности матрицы игры, а также кубическую вычислительную сложность $O(n^3)$, где n – число стратегий.

Численный метод Брауна–Робинсон при $\varepsilon \leq 0{,}1$ дал приближённое решение:

$$
\begin{aligned}
v &= 13{,}009 \\
x^* &= (0{,}558,\ 0{,}118,\ 0{,}324) \\
y^* &= (0{,}324,\ 0{,}148,\ 0{,}528)
\end{aligned}
$$

Метод имеет линейную сложность $O(m+n)$ и свободен от ограничений аналитического метода, но обладает немонотонностью сходимости.

Сравнительный анализ показал, что метод Брауна–Робинсон обеспечивает решение в пределах заданной точности ε, что подтверждает его пригодность для приближённого решения произвольных матричных игр.

Таким образом, метод Брауна–Робинсон представляет собой эффективный инструмент для решения игр большой размерности.

ПРИЛОЖЕНИЕ А

Класс реализация аналитического метода

Листинг А.1 – solver.go

```
package analytical

import (
  "gonum.org/v1/gonum/mat"
)

type Solver struct {
  m           *mat.Dense
  mInv        *mat.Dense
  u           *mat.VecDense
  denominator *mat.Dense
}

func New(m *mat.Dense) (*Solver, error) {
  a := &Solver{
    m: m,
  }

  a.mInv = &mat.Dense{}
  if err := a.mInv.Inverse(a.m); err != nil {
    return nil, err
  }

  r := a.m.RawMatrix().Rows

  oneVec := make([]float64, 0, r)
  for range r {
    oneVec = append(oneVec, 1)
  }

  a.u = mat.NewVecDense(r, oneVec)

  var uTCInv mat.Dense
  uTCInv.Mul(a.u.T(), a.mInv)

  a.denominator = &mat.Dense{}
  a.denominator.Mul(&uTCInv, a.u)
```

```
  return a, nil
}

func (a *Solver) Solve() (*Solution, error) {
  x, err := a.solveX()
  if err != nil {
    return nil, err
  }

  y, err := a.solveY()
  if err != nil {
    return nil, err
  }

  v, err := a.solveV()
  if err != nil {
    return nil, err
  }

  return &Solution{
    x: x,
    y: y,
    v: v,
  }, nil
}

func (a *Solver) solveX() (*mat.Dense, error) {
  var numerator mat.Dense
  numerator.Mul(a.u.T(), a.mInv)

  var res mat.Dense
  res.Scale(1/a.denominator.At(0, 0), &numerator)

  return &res, nil
}

func (s *Solver) solveY() (*mat.Dense, error) {
  var numerator mat.Dense
  numerator.Mul(s.mInv, s.u)

  var res mat.Dense
```

```
  res.Scale(1/s.denominator.At(0, 0), &numerator)

  return &res, nil
}

func (s *Solver) solveV() (float64, error) {
  return 1 / s.denominator.At(0, 0), nil
}
```

Листинг A.2 – solution.go

```
package analytical

import (
  "fmt"

  "gonum.org/v1/gonum/mat"
)

type Solution struct {
  x *mat.Dense
  y *mat.Dense
  v float64
}

func (a *Solution) String() string {
  return fmt.Sprintf("x* = %.3v\ny* = %.3v\nv=%.3f",
    mat.Formatted(a.x),
    mat.Formatted(a.y.T()),
    a.v)
}
```

ПРИЛОЖЕНИЕ Б

Класс реализация алгоритма Брауна–Робинсон

Листинг Б.3 – solver.go

```
package brownrobinson

import (
  "math/rand"
  "slices"
)

const (
  epsilon float64 = 0.1
  N       int     = 5
)

type Config struct {
  graphics bool
  eps      *float64
}

type Opt func(*Config)

func Graphics() Opt {
  return func(c *Config) {
    c.graphics = true
  }
}

func Epsilon(eps float64) Opt {
  return func(c *Config) {
    c.eps = &eps
  }
}

type BrownRobinson struct {
  m     [][]float64
  iters []iter

  minTop   float64
  maxLower float64
```

```
  xCount []float64
  yCount []float64

  sol *Solution

  graphics bool
  eps      float64
}

type iter struct {
  num int

  aWin  []float64
  bLoss []float64

  x int
  y int

  top   float64
  lower float64
  eps   float64
}

func New(m [][]float64, opts ...Opt) *BrownRobinson {
  config := &Config{}
  for _, opt := range opts {
    opt(config)
  }

  br := &BrownRobinson{
    m:        m,
    graphics: config.graphics,
    eps:      epsilon,
  }

  if config.eps != nil {
    br.eps = *config.eps
  }

  init := iter{
    num:   1,
```

```
    aWin:  br.column(0),
    bLoss: br.row(0),
    x:     0,
    y:     0,
  }

  init.top = slices.Max(init.aWin)
  init.lower = slices.Min(init.bLoss)
  init.eps = init.top - init.lower

  br.iters = []iter{init}

  br.sol = newSolution(len(init.aWin), len(init.bLoss))

  br.sol.append(init)

  br.minTop = init.top
  br.maxLower = init.lower

  br.xCount = make([]float64, len(init.aWin))
  br.yCount = make([]float64, len(init.bLoss))

  return br
}

func (b *BrownRobinson) Solve() *Solution {
  // Start with 2 because first one in New() constructor.
  iterNum := 2

  for !b.isFinish() {
    b.step(iterNum)

    b.sol.append(b.iters[len(b.iters)-1])

    iterNum++
  }

  for _, v := range b.xCount {
    b.sol.X = append(b.sol.X, v/float64(len(b.iters[len(b.iters)
       -1].aWin)))
  }
  for _, v := range b.yCount {
```

```
    b.sol.Y = append(b.sol.Y, v/float64(len(b.iters[len(b.iters)
        -1].bLoss)))
  }
  b.sol.V = (b.minTop + b.maxLower) / 2

  b.sol.t.Render()

  if b.graphics {
    b.sol.drawGraphics(b.iters)
  }

  return b.sol
}

func (b *BrownRobinson) isFinish() bool {
  return b.iters[len(b.iters)-1].eps <= b.eps
}

func (b *BrownRobinson) step(iterNum int) {
  last := b.iters[len(b.iters)-1]

  it := iter{
    num: iterNum,
  }

  _, it.x = b.max(last.aWin)
  _, it.y = b.min(last.bLoss)

  b.xCount[it.x]++
  b.yCount[it.y]++

  it.aWin = make([]float64, 0, len(last.aWin))
  for i, v := range b.column(it.y) {
    it.aWin = append(it.aWin, last.aWin[i]+v)
  }

  it.bLoss = make([]float64, 0, len(last.bLoss))
  for i, v := range b.row(it.x) {
    it.bLoss = append(it.bLoss, last.bLoss[i]+v)
  }

  v, _ := b.max(it.aWin)
```

```
  it.top = v / float64(iterNum)

  if b.minTop > it.top {
    b.minTop = it.top
  }

  v, _ = b.min(it.bLoss)
  it.lower = v / float64(iterNum)

  if b.maxLower < it.lower {
    b.maxLower = it.lower
  }

  it.eps = b.minTop - b.maxLower

  b.iters = append(b.iters, it)
}

func (b *BrownRobinson) column(j int) []float64 {
  col := make([]float64, 0, len(b.m))

  for i := range b.m {
    col = append(col, b.m[i][j])
  }

  return col
}

func (b *BrownRobinson) row(i int) []float64 {
  row := make([]float64, 0, len(b.m[i]))

  return append(row, b.m[i]...)
}

func (b *BrownRobinson) max(s []float64) (float64, int) {
  mIdxs := make([]int, 0)

  m := slices.Max(s)

  for i, v := range s {
    if v == m {
      mIdxs = append(mIdxs, i)
```

```
    }
  }

  random := rand.Intn(len(mIdxs))

  return m, mIdxs[random]
}

func (b *BrownRobinson) min(s []float64) (float64, int) {
  mIdxs := make([]int, 0)

  m := slices.Min(s)

  for i, v := range s {
    if v == m {
      mIdxs = append(mIdxs, i)
    }
  }

  random := rand.Intn(len(mIdxs))

  return m, mIdxs[random]
}
```

Листинг Б.4 – solution.go

```
package brownrobinson

import (
  "fmt"
  "path/filepath"
  "strings"

  "github.com/jedib0t/go-pretty/v6/table"
  "github.com/jedib0t/go-pretty/v6/text"
  "gonum.org/v1/plot"
  "gonum.org/v1/plot/plotter"
  "gonum.org/v1/plot/plotutil"
  "gonum.org/v1/plot/vg"
)

type Solution struct {
```

```
  t table.Writer
  b *strings.Builder

  X []float64
  Y []float64
  V float64
}

func newSolution(xLen, yLen int) *Solution {
  s := &Solution{
    t: table.NewWriter(),
    b: &strings.Builder{},
    X: make([]float64, 0),
    Y: make([]float64, 0),
  }
  s.t.SetOutputMirror(s.b)

  header := table.Row{"#", "A", "B"}

  for i := range xLen {
    header = append(header, fmt.Sprintf("x_%d", i+1))
  }
  for i := range yLen {
    header = append(header, fmt.Sprintf("y_%d", i+1))
  }

  header = append(header, "top_game_cost", "lower_game_cost", "eps"
     )

  s.t.AppendHeader(header)

  st := table.StyleLight
  st.Format.Header = text.FormatLower
  s.t.SetStyle(st)

  return s
}

func (s *Solution) String() string {
  xStr := &strings.Builder{}
  xStr.WriteString("x* = (")
  for i, v := range s.X {
```

```
    fmt.Fprintf(xStr, "%.3f", v)

    if i != len(s.X)-1 {
      xStr.WriteString(", ")
    }
  }

  xStr.WriteString(")\n")

  s.b.WriteString(xStr.String())

  yStr := &strings.Builder{}
  yStr.WriteString("y* = (")
  for i, v := range s.Y {
    fmt.Fprintf(yStr, "%.3f", v)

    if i != len(s.Y)-1 {
      yStr.WriteString(", ")
    }
  }

  yStr.WriteString(")\n")

  s.b.WriteString(yStr.String())

  fmt.Fprintf(s.b, "v = %.3f", s.V)

  return s.b.String()
}

func (s *Solution) append(it iter) {
  r := table.Row{
    it.num, fmt.Sprintf("x_%d", it.x+1),
    fmt.Sprintf("y_%d", it.y+1),
  }

  for _, v := range it.aWin {
    r = append(r, fmt.Sprintf("%d", int(v)))
  }

  for _, v := range it.bLoss {
    r = append(r, fmt.Sprintf("%d", int(v)))
```

```
  }

  r = append(r, fmt.Sprintf("%.3f", it.top),
    fmt.Sprintf("%.3f", it.lower),
    fmt.Sprintf("%.3f", it.eps))

  s.t.AppendRow(r)
}

func (s *Solution) drawGraphics(iters []iter) {
  p := plot.New()

  p.Title.Text = "Graph of convergence of upper and lower " +
    "game prices in the Brown-Robinson algorithm"
  p.X.Label.Text = "Iterations"
  p.Y.Label.Text = "Costs of game"

  lower := make([]float64, 0, len(iters))
  for _, v := range iters {
    lower = append(lower, v.lower)
  }
  lowerXYs := s.getCostPoints(lower)

  top := make([]float64, 0, len(iters))
  for _, v := range iters {
    top = append(top, v.top)
  }
  topXYs := s.getCostPoints(top)

  plotutil.AddLinePoints(p, "Lower cost", lowerXYs, "Top cost",
     topXYs)

  p.Save(10*vg.Inch, 5*vg.Inch, filepath.Join("artifacts", "lw1", "
     costs.png"))

  p = plot.New()
  p.Title.Text = "Estimation graph of the Brown-Robinson algorithm"
  p.X.Label.Text = "Iterations"
  p.Y.Label.Text = "Epsilon"

  eps := make([]float64, 0, len(iters))
  for _, v := range iters {
```

```
    eps = append(eps, v.eps)
  }
  epsXYs := s.getCostPoints(eps)

  plotutil.AddLinePoints(p, "Eps", epsXYs)

  p.Save(10*vg.Inch, 5*vg.Inch, filepath.Join("artifacts", "lw1", "
     estimation.png"))
}

func (s *Solution) getCostPoints(costs []float64) plotter.XYs {
  pts := make(plotter.XYs, len(costs))

  for i, v := range costs {
    pts[i].X = float64(i)
    pts[i].Y = v
  }

  return pts
}
```

ПРИЛОЖЕНИЕ В

Класс матричных игр

Листинг В.5 – game_matrix.go

```
package gamematrix

import (
  "fmt"
  "slices"
  "strings"

  "github.com/jedib0t/go-pretty/v6/table"
  "github.com/jedib0t/go-pretty/v6/text"
  "github.com/themilchenko/game_theory/internal/game_matrix/
     analytical"
  brownrobinson "github.com/themilchenko/game_theory/internal/
     game_matrix/brown_robinson"
  "gonum.org/v1/gonum/mat"
)

type GameMatrix struct {
  plainM [][]float64
  m      *mat.Dense

  lowestPrice  float64
  lowestIdx    int
  highestPrice float64
  highestIdx   int
}

func New(m [][]float64) (*GameMatrix, error) {
  if len(m) == 0 {
    return nil, fmt.Errorf("matrix should be not empty")
  }

  prev := len(m[0])

  for i := 1; i < len(m); i++ {
    if prev != len(m[i]) {
      return nil, fmt.Errorf("strings of matrix should be with the
         same size")
```

```
    }

    prev = len(m[i])
  }

  rows := len(m)
  cols := len(m[0])
  flatData := make([]float64, rows*cols)

  for i := range rows {
    for j := range cols {
      flatData[i*cols+j] = float64(m[i][j])
    }
  }

  g := &GameMatrix{
    plainM: m,
    m:      mat.NewDense(rows, cols, flatData),
  }
  g.lowestPrice, g.lowestIdx = g.calulateLowestPrice()
  g.highestPrice, g.highestIdx = g.calculateHighestPrice()

  return g, nil
}

func (g *GameMatrix) String() string {
  t := table.NewWriter()

  b := &strings.Builder{}
  t.SetOutputMirror(b)

  r, c := g.m.Dims()

  header := table.Row{"Strategies"}
  for i := range c {
    header = append(header, fmt.Sprintf("b_%d", i+1))
  }
  header = append(header, "min win of A player")
  t.AppendHeader(header)

  for i := range r {
    r := table.Row{fmt.Sprintf("a_%d", i+1)}
```

```
    for j := range c {
      r = append(r, fmt.Sprintf("%d", int(g.m.At(i, j))))
    }

    t.AppendRow(append(r, findMinInVec(g.m.RowView(i))))
  }

  t.AppendSeparator()

  raw := table.Row{"max loss of B player"}
  for j := range c {
    raw = append(raw, findMaxInVec(g.m.ColView(j)))
  }
  t.AppendRow(raw)

  s := table.StyleLight
  s.Format.Header = text.FormatLower
  t.SetStyle(s)

  t.Render()

  return b.String()
}

func (g *GameMatrix) MatrixString() string {
  return fmt.Sprintf("%.3v\n", mat.Formatted(g.m))
}

func (g *GameMatrix) LowestPrice() (float64, int) {
  return g.lowestPrice, g.lowestIdx
}

func (g *GameMatrix) HighestPrice() (float64, int) {
  return g.highestPrice, g.highestIdx
}

func (g *GameMatrix) SolveAnalytical() (*analytical.Solution, error
  ) {
  solver, err := analytical.New(g.m)
  if err != nil {
    return nil, err
```

```
  }

  return solver.Solve()
}

func (g *GameMatrix) SolveBrownRobinson(opts ...brownrobinson.Opt)
   *brownrobinson.Solution {
  solver := brownrobinson.New(g.plainM, opts...)

  return solver.Solve()
}

func (g *GameMatrix) calulateLowestPrice() (float64, int) {
  minStrings := make([]float64, 0, g.m.RawMatrix().Rows)

  for i := range g.m.RawMatrix().Rows {
    minStrings = append(minStrings, findMinInVec(g.m.RowView(i)))
  }

  return slices.Max(minStrings), slices.Index(minStrings, slices.
     Max(minStrings))
}

func (g *GameMatrix) calculateHighestPrice() (float64, int) {
  maxColumns := make([]float64, 0, g.m.RawMatrix().Cols)

  for j := range g.m.RawMatrix().Cols {
    maxColumns = append(maxColumns, findMaxInVec(g.m.ColView(j)))
  }

  return slices.Min(maxColumns), slices.Index(maxColumns, slices.
     Min(maxColumns))
}
```

Листинг B.6 – utils.go

```
package gamematrix

import (
  "slices"

  "gonum.org/v1/gonum/mat"
```

```
)

func covertMatVec(v mat.Vector) []float64 {
  converted := make([]float64, 0, v.Len())

  for i := range v.Len() {
    converted = append(converted, v.AtVec(i))
  }

  return converted
}

func findMinInVec(v mat.Vector) float64 {
  return slices.Min(covertMatVec(v))
}

func findMaxInVec(v mat.Vector) float64 {
  return slices.Max(covertMatVec(v))
}
```

ПРИЛОЖЕНИЕ Г

Точка входа в программу

Листинг Г.7 – main.go

```
package main

import (
  "encoding/json"
  "errors"
  "fmt"
  "log"
  "os"

  gamematrix "github.com/themilchenko/game_theory/internal/
     game_matrix"
  brownrobinson "github.com/themilchenko/game_theory/internal/
     game_matrix/brown_robinson"
)

func main() {
  if len(os.Args) != 2 {
    log.Fatal("usage: ./path/to/exec task.json")
  }

  if _, err := os.Stat(os.Args[1]); err != nil {
    if errors.Is(err, os.ErrNotExist) {
      log.Fatal(fmt.Errorf("file %q not exists: %w", os.Args[1],
         err))
    }
    log.Fatal(err)
  }

  fContent, err := os.ReadFile(os.Args[1])
  if err != nil {
    log.Fatal(fmt.Errorf("failed to read %q file: %w", os.Args[1],
       err))
  }

  var matrix [][]float64

  if err := json.Unmarshal(fContent, &matrix); err != nil {
```

```
    log.Fatal(fmt.Errorf("failed to parse matrix from json: %w",
       err))
  }

  game, err := gamematrix.New(matrix)
  if err != nil {
    log.Fatal(fmt.Errorf("can't creage game matrix: %w", err))
  }

  fmt.Println(game.String())
  l, _ := game.LowestPrice()
  h, _ := game.HighestPrice()
  fmt.Printf("Lowest Price: %d\n", int(l))
  fmt.Printf("Highest Price: %d\n", int(h))

  fmt.Println()

  sol, err := game.SolveAnalytical()
  if err != nil {
    log.Fatal(fmt.Errorf("failed to solve analytical: %w", err))
  }

  fmt.Println("Analytical solution:")
  fmt.Println(sol.String())

  fmt.Println("Brown Robinson solution:")

  fmt.Println(game.SolveBrownRobinson(brownrobinson.Graphics()).
     String())
}
```