Лабораторная работа № 3	ФИО	Титов А.К.
	Группа	ИВТ - 360
	Предмет	Методы принятия решений
	Вариант задания	4
	Дата отчета	
	Оценка	
	Подпись преподавателя	

Цель работы

Исследовать методы принятия решений в условиях неопределенности (риска) на основе теории игр с природой.

Постановка задачи

Необходимо разработать программное обеспечение для поддержки принятия решения в условиях неопределенности (риска) на основе теории игр с природой. Программное обеспечение должно реализовывать следующие критерии: Минимаксный (Вальда), Сэвиджа, Байеса-Лапласа, Гурвица, Гермейера, Ходжа-Лемана.

Индивидуальный вариант

	0.8	0.1	0.1
П1	4	3	5
П2	6	2	3
П3	2	5	-2

Ход выполнения лабораторной работы

Для решения задачи написан скрипт на языке Python 3.5.

На вход скрипта подается файл с данными в формате:	На выходе получаем файл с выходными данными в формате:
$p_1;p_2;p_3;\dots;p_n$ $a_{11};a_{12};a_{13};\dots;a_{1n}$ … … … $a_{n1};a_{n2};a_{n3};\dots;a_{nn}$	$p_1;p_2;p_3;\dots;p_n;\quad C_1;C_2;\dots;C_n$ $a_{11};a_{12};a_{13};\dots;a_{1n};\ C_{11};C_{12};\dots;C_{1n}$ … … … … … … … … … … $a_{n1};a_{n2};a_{n3};\dots;a_{nn};\ C_{n1};C_{n2};\dots;C_{nn}$

Где
P_1 – вероятность исхода $И_1$
A_{IJ} – выигрыш в случае выбора $I^{-ого}$ решения и возникновения $J^{-ого}$ события
C_i – наименование $I^{-ого}$ критерия
C_{JI} – значение $I^{-ого}$ критерия для $J^{-ого}$ решения

Результаты для тестового примера

Input

0.8	0.1	0.1
4	3	5
6	2	3
2	5	-2

Output

[0.8, 0.1, 0.1]	Vald	Savidge	Laplas-Baies	Ghurvitz	Ghermeyer	Hojj-Leman
[4.0, 3.0, 5.0]	| 3.0 |	| 2.0 |	4.0	| 4.0 |	3.2	| 6.5 |
[6.0, 2.0, 3.0]	2.0	4.0	| 5.3 |	4.0	4.8	5.65
[2.0, 5.0, -2.0]	-2.0	7.0	1.9	1.5	| 1.6 |	-2.05

Пояснение: | 3.0 | - таким образом выделяются лучшие значения критериев

Код программы

Функция main

```
def main():
  gurvitz_alpha    = 0.5 #input("Введите коэффициент для критерия Гурвица [0,1]: ")
  hojj_leman_alpha = 0.5 #input("Введите коэффициент для критерия Ходжа Лемана [0;1]:
")

  input_data = matrix_to_float(read_data())
  print(input_data)
  probability_vector = input_data[0]
  payoff_matrix      = input_data[1:]


  # Calculations
  vald         = Vald_criterium(payoff_matrix)
  savidge      = Savidge_criterium(payoff_matrix)
  laplas_baies = Laplas_Baies_criterium(probability_vector, payoff_matrix)
  ghurvitz     = Ghurvitz_criterium(gurvitz_alpha, payoff_matrix)
  ghermeyer    = Ghermeyer_criterium(probability_vector, payoff_matrix)
  hojj_leman   = Hojj_Leman_criterium(hojj_leman_alpha, probability_vector,
payoff_matrix)
```

Функции для расчета критериев

```
# Variuos criteries of "game with nature" teory
# Max(Mins)
def Vald_criterium(payoff_matrix):
  criterium_row = []
  for row in payoff_matrix:
    criterium_row.append(min(row))

  value_of_best = max(criterium_row)
  index_of_best = criterium_row.index(value_of_best)
  return highlight_best_value_row(criterium_row, index_of_best, value_of_best)

# Min(Max - Min)
def Savidge_criterium(payoff_matrix):
  criterium_row = []
  for row in payoff_matrix:
    criterium_row.append(max(row) - min(row))

  value_of_best = min(criterium_row)
  index_of_best = criterium_row.index(value_of_best)
  return highlight_best_value_row(criterium_row, index_of_best, value_of_best)

# Max(summ(a_i * q_i))
def Laplas_Baies_criterium(probability_vector, payoff_matrix):
  criterium_row = []
  for row in payoff_matrix:
    criterium_for_row = 0.0
    for i in range(0, len(row)):
      criterium_for_row += row[i] * probability_vector[i]

    criterium_row.append(criterium_for_row)

  value_of_best = max(criterium_row)
  index_of_best = criterium_row.index(value_of_best)
  return highlight_best_value_row(criterium_row, index_of_best, value_of_best)
```

```
# Max(alpha * min(row) + (1-alpha) * max(row))
def Ghurvitz_criterium(alpha, payoff_matrix):
  criterium_row = []
  for row in payoff_matrix:
    criterium_row.append(alpha * min(row) + (1-alpha) * max(row))

  value_of_best = max(criterium_row)
  index_of_best = criterium_row.index(value_of_best)
  return highlight_best_value_row(criterium_row, index_of_best, value_of_best)

# Min(Max(a_i*q_i of each in row))
def Ghermeyer_criterium(probability_vector, payoff_matrix):
  criterium_row = []
  for row in payoff_matrix:
    inefficiency_row = []
    for i in range(0, len(row)):
      inefficiency_row.append(row[i] * probability_vector[i])

    criterium_for_row = max(inefficiency_row)
    criterium_row.append(criterium_for_row)

  value_of_best = min(criterium_row)
  index_of_best = criterium_row.index(value_of_best)
  return highlight_best_value_row(criterium_row, index_of_best, value_of_best)

# Max(alpha * (a_i * q_i) + (1-alpha) * min(row))
def Hojj_Leman_criterium(alpha, probability_vector, payoff_matrix):
  criterium_row = []
  for row in payoff_matrix:
    criterium_for_row = 0.0
    for i in range(0, len(row)):
      criterium_for_row += (alpha * (row[i] * probability_vector[i]) +
                            (1-alpha) * min(row))

    criterium row.append(criterium for row)

  value_of_best = max(criterium_row)
  index_of_best = criterium_row.index(value_of_best)
  return highlight_best_value_row(criterium_row, index_of_best, value_of_best)
```