**БУ ВО «Сургутский государственный университет»**

Политехнический институт

Кафедра автоматизированных систем обработки информации и управления

**ОТЧЕТ**

ПО ЛАБОРАТОРНОЙ РАБОТЕ №1

ПО ТЕМЕ «Расчёт вероятностных характеристик и энтропий»

ПО ДИСЦИПЛИНЕ «Теория Информации»

Выполнил: студент группы №606-12,

Демьянцев Виталий Владиславович

Принял: ст. преподаватель кафедры АСОИУ,

Гавриленко Анна Владимировна

Сургут 2025

# 1 Введение

Целью данной работы является разработка программы для расчёта вероятностных характеристик и энтропий на основе заданных вероятностных матриц. Программа реализована на языке программирования C# с использованием технологии WPF (Windows Presentation Foundation).

Задача заключается в вычислении совместных вероятностей p(Zi, Wj ), условных вероятностей p(Wj |Zi) и p(Zi|Wj ), маргинальных вероятностей p(Zi) и p(Wj ), а также энтропий H(Z), H(W ), H(Z, W ), H(Z|W ), H(W |Z) и взаимной информации I(Z; W ). Пользователь может вводить данные в виде одной из трёх матриц (совместной или условной) и соответствующих ансамблей, а программа автоматически вычисляет остальные характеристики.

# Основные функции программы

1. **Инициализация матриц:** Программа создаёт интерфейс для ввода матриц *p*(*Zi,Wj*), *p*(*Wj*|*Zi*), *p*(*Zi*|*Wj*) и ансамблей *p*(*Zi*), *p*(*Wj*). Размеры матриц задаются пользователем (количество строк и столбцов).
2. **Валидация ввода:** Проверяется корректность введённых данных: неотрицательность вероятностей и соответствие сумм вероятностей (например, сумма элементов *p*(*Zi,Wj*) должна быть равна 1).
3. **Вычисление характеристик:** Программа вычисляет недостающие матрицы и ансамбли на основе введённых данных, а также рассчитывает энтропии и взаимную информацию.
4. **Отображение результатов:** Результаты выводятся в текстовом формате, включая все рассчитанные матрицы, ансамбли и энтропии.

**3 Скриншот программы**

# 

# 4 Функции в коде

## 4.1 InitializeMatrixInputs(int rows, int cols)

Инициализирует матрицы и ансамбли заданного размера, создавая структуры данных для хранения значений и привязывая их к интерфейсу.

## 4.2 SetDefaultValues()

Устанавливает значения по умолчанию для матрицы *p*(*Zi,Wj*) на основе задачи №9, обеспечивая начальные данные для тестирования.

## 4.3 UpdateMatrixButton\_Click(object sender, RoutedEventArgs e)

Обрабатывает изменение размеров матриц, переинициализируя структуры данных в соответствии с выбранными пользователем параметрами.

## 4.4 CalculateButton\_Click(object sender, RoutedEventArgs e)

Основной обработчик, который запускает вычисления. Проверяет введённые данные, определяет тип задачи (на основе заполненной матрицы) и вызывает функции для валидации и решения.

**4.5 ValidateInput(double[,]matrix,double[]ensemble,inttaskType,**

## int rows, int cols)

Проверяет корректность введённых данных: неотрицательность вероятностей и соответствие сумм (например, сумма по строкам для *p*(*Wj*|*Zi*) равна 1).

## 4.6 SolveTask(double[,] inputMatrix, double[] inputEnsemble, int taskType, int rows, int cols)

Реализует основной алгоритм вычислений. Определяет совместные вероятности *p*(*Zi,Wj*), вычисляет маргинальные и условные вероятности, а также энтропии и взаимную информацию.

## 4.7 Log2(double n)

Вспомогательная функция для вычисления логарифма по основанию 2, используемого при расчёте энтропий.

# 5 Заключение

Программа успешно решает задачу расчёта вероятностных характеристик и энтропий для заданных матриц вероятностей. Интерфейс позволяет гибко задавать входные данные и получать подробные результаты, что делает программу полезной для изучения теории вероятностей и информационной теории. Реализация валидации и обработки ошибок обеспечивает надёжность работы программы.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Windows;

using System.Windows.Controls;

using System.ComponentModel;

using System.Collections.ObjectModel;

namespace lab1

{

public partial class MainWindow : Window, INotifyPropertyChanged

{

private ObservableCollection<ObservableCollection<MatrixCell>> matrixJointData;

private ObservableCollection<ObservableCollection<MatrixCell>> matrixCondWgivenZData;

private ObservableCollection<ObservableCollection<MatrixCell>> matrixCondZgivenWData;

private ObservableCollection<MatrixCell> ensembleZData;

private ObservableCollection<MatrixCell> ensembleWData;

public event PropertyChangedEventHandler PropertyChanged;

public MainWindow()

{

InitializeComponent();

DataContext = this;

InitializeMatrixInputs(2, 4); // Инициализация матриц

SetDefaultValues(); // Устанавливаем значения из задачи №9 для p(Zi, Wj)

}

private void InitializeMatrixInputs(int rows, int cols)

{

// Инициализация p(Zi, Wj)

matrixJointData = new ObservableCollection<ObservableCollection<MatrixCell>>();

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

{

var row = new ObservableCollection<MatrixCell>();

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

{

row.Add(new MatrixCell { Value = "0" });

}

matrixJointData.Add(row);

}

MatrixJointInput.ItemsSource = matrixJointData;

// Инициализация p(Wj | Zi)

matrixCondWgivenZData = new ObservableCollection<ObservableCollection<MatrixCell>>();

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

{

var row = new ObservableCollection<MatrixCell>();

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

{

row.Add(new MatrixCell { Value = "0" });

}

matrixCondWgivenZData.Add(row);

}

MatrixCondWgivenZInput.ItemsSource = matrixCondWgivenZData;

// Инициализация p(Zi | Wj)

matrixCondZgivenWData = new ObservableCollection<ObservableCollection<MatrixCell>>();

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

{

var row = new ObservableCollection<MatrixCell>();

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

{

row.Add(new MatrixCell { Value = "0" });

}

matrixCondZgivenWData.Add(row);

}

MatrixCondZgivenWInput.ItemsSource = matrixCondZgivenWData;

// Инициализация ансамбля p(Zi)

ensembleZData = new ObservableCollection<MatrixCell>();

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

{

ensembleZData.Add(new MatrixCell { Value = "0" });

}

EnsembleZInput.ItemsSource = ensembleZData;

// Инициализация ансамбля p(Wj)

ensembleWData = new ObservableCollection<MatrixCell>();

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

{

ensembleWData.Add(new MatrixCell { Value = "0" });

}

EnsembleWInput.ItemsSource = ensembleWData;

}

private void SetDefaultValues()

{

// Значения из задачи №9 для p(Zi, Wj)

var defaultMatrix = new double[,]

{

{ 0.32, 0.10, 0.16, 0.02 },

{ 0.08, 0.20, 0.04, 0.08 }

};

//var defaultEnsembleW = new double[] { 0.40, 0.30, 0.20, 0.10 };

for (int i = 0; i < matrixJointData.Count; i++)

{

for (int j = 0; j < matrixJointData[i].Count; j++)

{

matrixJointData[i][j].Value = defaultMatrix[i, j].ToString("0.00");

}

}

//for (int j = 0; j < ensembleWData.Count; j++)

//{

// ensembleWData[j].Value = defaultEnsembleW[j].ToString("0.00");

//}

}

private void UpdateMatrixButton\_Click(object sender, RoutedEventArgs e)

{

int rows = int.Parse((RowsComboBox.SelectedItem as ComboBoxItem).Content.ToString());

int cols = int.Parse((ColsComboBox.SelectedItem as ComboBoxItem).Content.ToString());

InitializeMatrixInputs(rows, cols);

}

private void CalculateButton\_Click(object sender, RoutedEventArgs e)

{

try

{

int rows = matrixJointData.Count;

int cols = matrixJointData[0].Count;

// Парсим все матрицы

double[,] matrixJoint = new double[rows, cols];

double[,] matrixCondWgivenZ = new double[rows, cols];

double[,] matrixCondZgivenW = new double[rows, cols];

double[] ensembleZ = new double[rows];

double[] ensembleW = new double[cols];

bool isJointFilled = false;

bool isCondWgivenZFilled = false;

bool isCondZgivenWFilled = false;

// p(Zi, Wj)

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

{

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

{

matrixJoint[i, j] = double.Parse(matrixJointData[i][j].Value);

if (matrixJoint[i, j] != 0) isJointFilled = true;

}

}

// p(Wj | Zi)

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

{

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

{

matrixCondWgivenZ[i, j] = double.Parse(matrixCondWgivenZData[i][j].Value);

if (matrixCondWgivenZ[i, j] != 0) isCondWgivenZFilled = true;

}

}

// p(Zi | Wj)

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

{

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

{

matrixCondZgivenW[i, j] = double.Parse(matrixCondZgivenWData[i][j].Value);

if (matrixCondZgivenW[i, j] != 0) isCondZgivenWFilled = true;

}

}

// Ансамбль p(Zi)

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

{

ensembleZ[i] = double.Parse(ensembleZData[i].Value);

}

// Ансамбль p(Wj)

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

{

ensembleW[j] = double.Parse(ensembleWData[j].Value);

}

// Проверка, что хотя бы одна матрица заполнена

if (!isJointFilled && !isCondWgivenZFilled && !isCondZgivenWFilled)

{

throw new Exception("Хотя бы одна матрица должна быть заполнена (содержать ненулевые значения).");

}

// Определяем тип задачи на основе заполненной матрицы

int taskType = -1;

double[,] inputMatrix = null;

double[] inputEnsemble = null;

if (isJointFilled)

{

taskType = 0;

inputMatrix = matrixJoint;

}

else if (isCondWgivenZFilled)

{

taskType = 1;

inputMatrix = matrixCondWgivenZ;

inputEnsemble = ensembleZ;

}

else if (isCondZgivenWFilled)

{

taskType = 2;

inputMatrix = matrixCondZgivenW;

inputEnsemble = ensembleW;

}

// Проверяем корректность введённых данных

ValidateInput(inputMatrix, inputEnsemble, taskType, rows, cols);

// Решаем задачу

var result = SolveTask(inputMatrix, inputEnsemble, taskType, rows, cols);

ResultsTextBlock.Text = result;

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка: {ex.Message}", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

private void ValidateInput(double[,] matrix, double[] ensemble, int taskType, int rows, int cols)

{

// Проверяем, что все элементы неотрицательны

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

{

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

{

if (matrix[i, j] < 0)

throw new Exception("Вероятности не могут быть отрицательными.");

}

}

if (taskType != 0) // Ансамбль проверяем только если он используется

{

for (int j = 0; j < ensemble.Length; j++)

{

if (ensemble[j] < 0)

throw new Exception("Вероятности ансамбля не могут быть отрицательными.");

}

}

// Проверяем суммы в зависимости от типа задачи

if (taskType == 0) // p(Zi, Wj)

{

double sum = 0;

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

{

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

{

sum += matrix[i, j];

}

}

if (Math.Abs(sum - 1.0) > 0.05)

throw new Exception($"Сумма элементов p(Zi, Wj) должна быть равна 1. Текущая сумма: {sum}");

}

else if (taskType == 1) // p(Wj | Zi)

{

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

{

double rowSum = 0;

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

{

rowSum += matrix[i, j];

}

if (Math.Abs(rowSum - 1.0) > 0.05)

throw new Exception("Сумма по строкам p(Wj | Zi) должна быть равна 1.");

}

double ensembleSum = ensemble.Sum();

if (Math.Abs(ensembleSum - 1.0) > 0.05)

throw new Exception("Сумма элементов p(Zi) должна быть равна 1.");

}

else if (taskType == 2) // p(Zi | Wj)

{

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

{

double colSum = 0;

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

{

colSum += matrix[i, j];

}

if (Math.Abs(colSum - 1.0) > 0.05)

throw new Exception("Сумма по столбцам p(Zi | Wj) должна быть равна 1.");

}

double ensembleSum = ensemble.Sum();

if (Math.Abs(ensembleSum - 1.0) > 0.05)

throw new Exception("Сумма элементов p(Wj) должна быть равна 1.");

}

}

private double Log2(double n)

{

return Math.Log(n) / Math.Log(2);

}

private string SolveTask(double[,] inputMatrix, double[] inputEnsemble, int taskType, int rows, int cols)

{

double[,] p\_Zi\_Wj = new double[rows, cols]; // Совместные вероятности

double[] p\_Zi = new double[rows];

double[] p\_Wj = new double[cols];

double[,] p\_Wj\_given\_Zi = new double[rows, cols];

double[,] p\_Zi\_given\_Wj = new double[rows, cols];

// Шаг 1: Определяем p(Zi, Wj) в зависимости от типа задачи

if (taskType == 0) // Дана p(Zi, Wj)

{

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

{

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

{

p\_Zi\_Wj[i, j] = inputMatrix[i, j];

}

}

}

else if (taskType == 1) // Дана p(Wj | Zi) и p(Zi)

{

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

{

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

{

p\_Zi\_Wj[i, j] = inputMatrix[i, j] \* inputEnsemble[i];

}

}

}

else if (taskType == 2) // Дана p(Zi | Wj) и p(Wj)

{

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

{

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

{

p\_Zi\_Wj[i, j] = inputMatrix[i, j] \* inputEnsemble[j];

}

}

}

// Шаг 2: Находим p(Zi) и p(Wj)

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

{

p\_Zi[i] = 0;

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

{

p\_Zi[i] += p\_Zi\_Wj[i, j];

}

}

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

{

p\_Wj[j] = 0;

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

{

p\_Wj[j] += p\_Zi\_Wj[i, j];

}

}

// Шаг 3: Находим p(Wj | Zi) и p(Zi | Wj)

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

{

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

{

if (p\_Zi[i] > 0)

p\_Wj\_given\_Zi[i, j] = p\_Zi\_Wj[i, j] / p\_Zi[i];

else

p\_Wj\_given\_Zi[i, j] = 0;

if (p\_Wj[j] > 0)

p\_Zi\_given\_Wj[i, j] = p\_Zi\_Wj[i, j] / p\_Wj[j];

else

p\_Zi\_given\_Wj[i, j] = 0;

}

}

// Шаг 4: Вычисляем энтропии

double H\_Z = 0;

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

{

if (p\_Zi[i] > 0)

H\_Z -= p\_Zi[i] \* Log2(p\_Zi[i]);

}

double H\_W = 0;

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

{

if (p\_Wj[j] > 0)

H\_W -= p\_Wj[j] \* Log2(p\_Wj[j]);

}

double H\_ZW = 0;

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

{

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

{

if (p\_Zi\_Wj[i, j] > 0)

H\_ZW -= p\_Zi\_Wj[i, j] \* Log2(p\_Zi\_Wj[i, j]);

}

}

double H\_W\_Z = 0;

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

{

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

{

if (p\_Zi\_Wj[i, j] > 0 && p\_Zi\_given\_Wj[i, j] > 0)

H\_W\_Z -= p\_Zi\_Wj[i, j] \* Log2(p\_Zi\_given\_Wj[i, j]);

}

}

double H\_Z\_W = 0;

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

{

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

{

if (p\_Zi\_Wj[i, j] > 0 && p\_Wj\_given\_Zi[i, j] > 0)

H\_Z\_W -= p\_Zi\_Wj[i, j] \* Log2(p\_Wj\_given\_Zi[i, j]);

}

}

// Шаг 5: Вычисляем I(Z;W)

double I\_ZW = H\_Z - H\_W\_Z;

// Формируем вывод

StringBuilder result = new StringBuilder();

result.AppendLine("Матрица p(Zi, Wj):");

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

{

var row = new List<string>();

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

{

row.Add(p\_Zi\_Wj[i, j].ToString("0.00"));

}

result.AppendLine(string.Join("\t", row));

}

result.AppendLine("\nМатрица p(Wj | Zi):");

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

{

var row = new List<string>();

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

{

row.Add(p\_Wj\_given\_Zi[i, j].ToString("0.00"));

}

result.AppendLine(string.Join("\t", row));

}

result.AppendLine("\nМатрица p(Zi | Wj):");

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

{

var row = new List<string>();

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

{

row.Add(p\_Zi\_given\_Wj[i, j].ToString("0.00"));

}

result.AppendLine(string.Join("\t", row));

}

result.AppendLine("\nАнсамбль p(Zi):");

result.AppendLine(string.Join("\t", p\_Zi.Select(p => p.ToString("0.00"))));

result.AppendLine("\nАнсамбль p(Wj):");

result.AppendLine(string.Join("\t", p\_Wj.Select(p => p.ToString("0.00"))));

result.AppendLine("\nЭнтропии:");

result.AppendLine($"H(Z) = {H\_Z.ToString("0.0000")}");

result.AppendLine($"H(W) = {H\_W.ToString("0.0000")}");

result.AppendLine($"H(Z, W) = {H\_ZW.ToString("0.0000")}");

result.AppendLine($"H(Z|W) = {H\_W\_Z.ToString("0.0000")}");//Hw(Z)

result.AppendLine($"H(W|Z) = {H\_Z\_W.ToString("0.0000")}");//Hz(W)

result.AppendLine($"I(Z;W) = {I\_ZW.ToString("0.0000")}");

return result.ToString();

}

}

public class MatrixCell : INotifyPropertyChanged

{

private string value;

public string Value

{

get => value;

set

{

this.value = value;

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(nameof(Value)));

}

}

public event PropertyChangedEventHandler PropertyChanged;

}

}