**Задание** на летнюю практику «Научно-исследовательская работа (получение первичных навыков научно-исследовательской работы)»

**Тема**: «Численное моделирование нелинейных процессов массопереноса».

Написать программу для решения методом конечных разностей нестационарной многомерной начально-краевой задачи для системы связных уравнений реакции-диффузии.

Доцент каф. ПМ

к.т.н., доцент Н.С. Бузало

Оглавление

[1. Постановка задачи 3](#_Toc204289520)

[1.1 Уравнения модели 3](#_Toc204289521)

[1.2 Начальные и граничные условия 3](#_Toc204289522)

[2. Методы решения 4](#_Toc204289523)

[2.1 Метод расщепления 4](#_Toc204289524)

[2.2 Неявная разностная схема 4](#_Toc204289525)

[2.3 Метод прогонки (алгоритм Томаса) 4](#_Toc204289526)

[3. Реализация 5](#_Toc204289527)

[3.1 Структура программы 5](#_Toc204289528)

[4. Листинг программы 6](#_Toc204289529)

[Листинг 1 - Класс Practice 6](#_Toc204289530)

[Листинг 2 - Класс Solve 9](#_Toc204289531)

[Листинг 3 - Класс Create 15](#_Toc204289532)

[Листинг 4 - Класс Apply 18](#_Toc204289533)

[Листинг 5 - Класс Result 23](#_Toc204289534)

[Листинг 5 - Класс Config 27](#_Toc204289535)

[5. Пример работы программы 27](#_Toc204289536)

[6. Заключение 30](#_Toc204289537)

[7. Список литературы 30](#_Toc204289538)

# 1. Постановка задачи

Целью практики является разработка программы для моделирования взаимодействия популяций водорослей и креветок в трёхмерной среде. Модель учитывает:

Диффузию водорослей и креветок в пространстве.

Взаимодействие между видами (поедание водорослей креветками).

Различные граничные условия (Неймана для изолированной системы).

## 1.1 Уравнения модели

Для водорослей (A):

где:

1. DA - коэффициент диффузии водорослей,
2. rA​ - скорость роста,
3. γ - скорость поедания водорослей креветками,
4. S - концентрация креветок.

Для креветок (S):

где:

1. DS - коэффициент диффузии креветок,
2. δ - смертность креветок,
3. ϵ - эффективность преобразования водорослей в креветок.

## 1.2 Начальные и граничные условия

Начальные условия: равномерное распределение концентраций

Граничные условия: Неймана (нулевой поток на границах) для изолированной системы.

# 2. Методы решения

## 2.1 Метод расщепления

Трёхмерная задача разбивается на последовательность одномерных задач по осям x, y, и z:

Решение для оси x:

Решение для оси y:

Решение для оси z:

Аналогично для креветок.

## 2.2 Неявная разностная схема

Используется неявная схема для устойчивости при любых шагах по времени. Например, для оси x:

Где

## 2.3 Метод прогонки (алгоритм Томаса)

Для решения трёхдиагональных систем на каждом этапе расщепления применяется метод прогонки.

# 3. Реализация

## 3.1 Структура программы

1. Программа реализована на языке C# с использованием библиотеки MathNet.Numerics для линейной алгебры. Основные компоненты:
2. Класс Practice: Точка входа, задание параметров модели.
3. Класс Solve\_2: Реализация метода расщепления для двух видов.
4. Классы Create и Apply: Создание матриц и применение граничных условий.
5. Класс Results: Визуализация результатов (срезы по осям).

3.2 Учёт граничных условий

Граничные условия задаются в словарях:

// Граничные условия для водорослей (Algae)

var algaeBoundaryConditions = new Dictionary<string, (int type, double value)>

{

{ "x0", (1, 0.0) },

{ "x1", (1, 0.0) },

{ "y0", (1, 0.0) },

{ "y1", (1, 0.0) },

{ "z0", (1, 0.0) },

{ "z1", (1, 0.0) }

};

// Граничные условия для креветок (Shrimps)

var shrimpsBoundaryConditions = new Dictionary<string, (int type, double value)>

{

{ "x0", (1, 0.0) },

{ "x1", (1, 0.0) },

{ "y0", (1, 0.0) },

{ "y1", (1, 0.0) },

{ "z0", (1, 0.0) },

{ "z1", (1, 0.0) }

};

3.3 Взаимодействие видов

Члены взаимодействия добавляются в правые части уравнений:

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

{

vector[i] += dt \* interactionTerm(solution[i, j, k], otherSolution[i, j, k]);

}

# 4. Листинг программы

## Листинг 1 - Класс Practice

using PracticeLibrary;

public class Practice

{

public static void Main(string[] args)

{

Console.WriteLine("Добро пожаловать");

// Параметры сетки

int nx = Config.nx;

int ny = Config.ny;

int nz = Config.nz;

double lx = Config.lx;

double ly = Config.ly;

double lz = Config.lz;

// Параметры времени

double tMax = 0.1;

double dt = 0.001;

// Коэффициенты диффузии

double algaeDiffusion = 0.1; // для водорослей

double shrimpDiffusion = 0.1; // для креветок

Func<double, double, double, double> initialAlgae = (x, y, z) => 1;

//Func<double, double, double, double> initialAlgae = (x, y, z) => 1;

Func<double, double, double, double> initialShrimps = (x, y, z) => 1;

// Граничные условия для водорослей (Algae)

var algaeBoundaryConditions = new Dictionary<string, (int type, double value)>

{

{ "x0", (1, 0.0) },

{ "x1", (1, 0.0) },

{ "y0", (1, 0.0) },

{ "y1", (1, 0.0) },

{ "z0", (1, 0.0) },

{ "z1", (1, 0.0) }

};

// Граничные условия для креветок (Shrimps)

var shrimpsBoundaryConditions = new Dictionary<string, (int type, double value)>

{

{ "x0", (1, 0.0) },

{ "x1", (1, 0.0) },

{ "y0", (1, 0.0) },

{ "y1", (1, 0.0) },

{ "z0", (1, 0.0) },

{ "z1", (1, 0.0) }

};

// Коэффициенты для граничного условия Робина

var robinCoefficients = new Dictionary<string, (double alpha, double beta)>

{

};

double[,,] algaeSolution;

double[,,] shrimpsSolution;

(algaeSolution, shrimpsSolution) = Solve\_2.SolveGeneral(

nx,

ny,

nz,

lx,

ly,

lz,

tMax,

dt,

algaeDiffusion,

shrimpDiffusion,

initialAlgae,

initialShrimps,

algaeBoundaryConditions,

shrimpsBoundaryConditions,

robinCoefficients);

Console.WriteLine("\nВыберите способ вывода результатов:\n1) Вывод в .txt файл\n2) Визуализировать в консоли водоросли\n2) Визуализировать в консоли креветки");

string writeCheck = Console.ReadLine();

switch (writeCheck)

{

default:

//Results.WriteToFile(solution);

break;

case "2":

while (true)

{

Console.Clear();

Console.WriteLine("\n1) Выбрать срез\n2) Выйти");

int wCheck = Convert.ToInt16(Console.ReadLine());

if (wCheck == 1)

{

Results.PrintSlice(algaeSolution);

}

else

{

break;

}

}

break;

case "3":

while (true)

{

Console.Clear();

Console.WriteLine("\n1) Выбрать срез\n2) Выйти");

int wCheck = Convert.ToInt16(Console.ReadLine());

if (wCheck == 1)

{

Results.PrintSlice(shrimpsSolution);

}

else

{

break;

}

}

break;

case "4":

while (true)

{

Console.Clear();

Console.WriteLine("\n1) Выбрать срез\n2) Выйти");

int wCheck = Convert.ToInt16(Console.ReadLine());

if (wCheck == 1)

{

//Results.PrintSlice(shrimpsSolution, algaeSolution);

}

else

{

break;

}

}

break;

}

}

}

## Листинг 2 - Класс Solve

using MathNet.Numerics.LinearAlgebra;

namespace PracticeLibrary

{

public class Solve

{

private static double CalculateTotalMass(double[,,] solution)

{

double dx = Config.dx;

double dy = Config.dy;

double dz = Config.dz;

double totalMass = 0;

for (int i = 0; i < solution.GetLength(0); i++)

for (int j = 0; j < solution.GetLength(1); j++)

for (int k = 0; k < solution.GetLength(2); k++)

totalMass += solution[i, j, k] \* dx \* dy \* dz;

return totalMass;

}

private static double CalculateAverageConcentration(double[,,] solution)

{

double lx = Config.lx;

double ly = Config.lx;

double lz = Config.lx;

double totalVolume = lx \* ly \* lz;

double totalAveragge = CalculateTotalMass(solution) / totalVolume;

return totalAveragge;

}

public static (double[,,] algaeSolution, double[,,] shrimpsSolution) SolveGeneral(

int nx,

int ny,

int nz,

double lx,

double ly,

double lz,

double tMax,

double dt,

double algaeDiffusion,

double shrimpDiffusion,

Func<double, double, double, double> initialAlgae,

Func<double, double, double, double> initialShrimps,

Dictionary<string, (int type, double value)> algaeBoundaryConditions,

Dictionary<string, (int type, double value)> shrimpsBoundaryConditions,

Dictionary<string, (double alpha, double beta)> robinCoefficients)

{

double algaeGrowthRate = 0; // Скорость роста водорослей

double shrimpDeathRate = 0; // Смертность креветок

double interactionRate = 1; // Скорость поедания водорослей креветками

double conversionRate = 1; // Эффективность преобразования водорослей в креветок

double dx = Config.dx;

double dy = Config.dy;

double dz = Config.dz;

double rx\_aglae = algaeDiffusion \* dt / (2.0 \* dx \* dx);

double ry\_aglae = algaeDiffusion \* dt / (2.0 \* dy \* dy);

double rz\_aglae = algaeDiffusion \* dt / (2.0 \* dz \* dz);

// Для креветок

double rx\_shrimp = shrimpDiffusion \* dt / (2.0 \* dx \* dx);

double ry\_shrimp = shrimpDiffusion \* dt / (2.0 \* dy \* dy);

double rz\_shrimp = shrimpDiffusion \* dt / (2.0 \* dz \* dz);

double[,,] algaeSolution = new double[nx, ny, nz];

double[,,] shrimpsSolution = new double[nx, ny, nz];

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

{

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

{

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

{

double x = i \* dx;

double y = j \* dy;

double z = k \* dz;

algaeSolution[i, j, k] = initialAlgae(x, y, z);

shrimpsSolution[i, j, k] = initialShrimps(x, y, z);

}

}

}

// Вывод информации о начальных условиях

Console.WriteLine("Начальные условия:");

Console.WriteLine($"Общая масса водорослей: {CalculateTotalMass(algaeSolution)}");

Console.WriteLine($"Общая масса креветок: {CalculateTotalMass(shrimpsSolution)}");

Console.WriteLine($"Общая масса: {CalculateTotalMass(shrimpsSolution) + CalculateTotalMass(algaeSolution)}\n");

Console.WriteLine($"Общая концентрация водорослей: {CalculateAverageConcentration(algaeSolution)}");

Console.WriteLine($"Общая концентрация креветок: {CalculateAverageConcentration(shrimpsSolution)}");

for (double t = 0; t < tMax; t += dt)

{

SolveX(algaeSolution, algaeDiffusion, dt, rx\_aglae, nx, ny, nz, dx, dy, dz,

algaeBoundaryConditions, robinCoefficients, t, shrimpsSolution,

(a, s) => a \* algaeGrowthRate \* (1 - a) - interactionRate \* a \* s);

SolveY(algaeSolution, algaeDiffusion, dt, ry\_aglae, nx, ny, nz, dx, dy, dz,

algaeBoundaryConditions, robinCoefficients, t, shrimpsSolution,

(a, s) => a \* algaeGrowthRate \* (1 - a) - interactionRate \* a \* s);

SolveZ(algaeSolution, algaeDiffusion, dt, rz\_aglae, nx, ny, nz, dx, dy, dz,

algaeBoundaryConditions, robinCoefficients, t, shrimpsSolution,

(a, s) => a \* algaeGrowthRate \* (1 - a) - interactionRate \* a \* s);

SolveX(shrimpsSolution, shrimpDiffusion, dt, rx\_shrimp, nx, ny, nz, dx, dy, dz,

shrimpsBoundaryConditions, robinCoefficients, t, algaeSolution,

(s, a) => -shrimpDeathRate \* s + conversionRate \* a \* s);

SolveY(shrimpsSolution, shrimpDiffusion, dt, ry\_shrimp, nx, ny, nz, dx, dy, dz,

shrimpsBoundaryConditions, robinCoefficients, t, algaeSolution,

(s, a) => -shrimpDeathRate \* s + conversionRate \* a \* s);

SolveZ(shrimpsSolution, shrimpDiffusion, dt, rz\_shrimp, nx, ny, nz, dx, dy, dz,

shrimpsBoundaryConditions, robinCoefficients, t, algaeSolution,

(s, a) => -shrimpDeathRate \* s + conversionRate \* a \* s);

Console.WriteLine($"После шага t={t}:");

Console.WriteLine($"Общая масса водорослей: {CalculateTotalMass(algaeSolution)}");

Console.WriteLine($"Общая масса креветок: {CalculateTotalMass(shrimpsSolution)}");

Console.WriteLine($"Общая масса: {CalculateTotalMass(shrimpsSolution) + CalculateTotalMass(algaeSolution)}\n");

Console.WriteLine($"Общая концентрация водорослей: {CalculateAverageConcentration(algaeSolution)}");

Console.WriteLine($"Общая концентрация креветок: {CalculateAverageConcentration(shrimpsSolution)}");

}

return (algaeSolution, shrimpsSolution);

}

private static void SolveX(

double[,,] solution,

double alpha, double dt, double rx,

int nx, int ny, int nz,

double dx, double dy, double dz,

Dictionary<string, (int type, double value)> boundaryConditions,

Dictionary<string, (double alpha, double beta)> robinCoefficients,

double t,

double[,,] otherSolution,

Func<double, double, double> interactionTerm)

{

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

{

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

{

Matrix<double> matrix = Create.TridiagonalMatrixX(nx, rx);

Vector<double> vector = Create.RightHandSideX(solution, alpha, dt, rx, nx, j, k, dx, dy, dz, t);

// Добавляем член взаимодействия

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

{

vector[i] += dt \* interactionTerm(solution[i, j, k], otherSolution[i, j, k]);

}

Apply.BoundaryConditionsX(matrix, vector, rx, nx, j, dx, boundaryConditions, robinCoefficients);

Vector<double> x = matrix.Solve(vector);

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

{

solution[i, j, k] = Math.Max(0, x[i]); // Гарантируем неотрицательность

}

}

}

}

private static void SolveY(

double[,,] solution,

double alpha, double dt, double ry,

int nx, int ny, int nz,

double dx, double dy, double dz,

Dictionary<string, (int type, double value)> boundaryConditions,

Dictionary<string, (double alpha, double beta)> robinCoefficients,

double t,

double[,,] otherSolution,

Func<double, double, double> interactionTerm)

{

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

{

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

{

Matrix<double> matrix = Create.TridiagonalMatrixY(ny, ry);

Vector<double> vector = Create.RightHandSideY(solution, alpha, dt, ry, ny, i, k, dx, dy, dz, t);

// Добавляем член взаимодействия

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

{

vector[j] += dt \* interactionTerm(solution[i, j, k], otherSolution[i, j, k]);

}

Apply.BoundaryConditionsY(matrix, vector, ry, ny, dy, boundaryConditions, robinCoefficients);

Vector<double> newValues = matrix.Solve(vector);

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

{

solution[i, j, k] = Math.Max(0, newValues[i]); // Гарантируем неотрицательность

}

}

}

}

private static void SolveZ(

double[,,] solution,

double alpha, double dt, double rz,

int nx, int ny, int nz,

double dx, double dy, double dz,

Dictionary<string, (int type, double value)> boundaryConditions,

Dictionary<string, (double alpha, double beta)> robinCoefficients,

double t,

double[,,] otherSolution,

Func<double, double, double> interactionTerm)

{

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

{

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

{

Matrix<double> matrix = Create.TridiagonalMatrixZ(nx, rz);

Vector<double> vector = Create.RightHandSideZ(solution, alpha, dt, rz, nx, i, j, dx, dy, dz, t);

// Добавляем член взаимодействия

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

{

vector[k] += dt \* interactionTerm(solution[i, j, k], otherSolution[i, j, k]);

}

Apply.BoundaryConditionsZ(matrix, vector, rz, nz, dz, boundaryConditions, robinCoefficients);

Vector<double> newValues = matrix.Solve(vector);

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

{

solution[i, j, k] = Math.Max(0, newValues[i]); // Гарантируем неотрицательность

}

}

}

}

}

}

## Листинг 3 - Класс Create

using MathNet.Numerics.LinearAlgebra;

namespace PracticeLibrary

{

public class Create

{

public static Matrix<double> TridiagonalMatrixX(int nx, double rx)

{

var matrix = Matrix<double>.Build.Dense(nx, nx, 0.0);

for (int i = 1; i < nx - 1; i++)

{

matrix[i, i - 1] = -rx;

matrix[i, i] = 1 + 2 \* rx;

matrix[i, i + 1] = -rx;

}

return matrix;

}

public static Matrix<double> TridiagonalMatrixY(int ny, double ry)

{

var matrix = Matrix<double>.Build.Dense(ny, ny, 0.0);

for (int i = 1; i < ny - 1; i++)

{

matrix[i, i - 1] = -ry;

matrix[i, i] = 1 + 2 \* ry;

matrix[i, i + 1] = -ry;

}

return matrix;

}

public static Matrix<double> TridiagonalMatrixZ(int nz, double rz)

{

var matrix = Matrix<double>.Build.Dense(nz, nz, 0.0);

for (int i = 1; i < nz - 1; i++)

{

matrix[i, i - 1] = -rz;

matrix[i, i] = 1 + 2 \* rz;

matrix[i, i + 1] = -rz;

}

return matrix;

}

public static Vector<double> RightHandSideX(

double[,,] solution,

double alpha,

double dt,

double rx,

int nx,

int j,

int k,

double dx,

double dy,

double dz,

double t)

{

Vector<double> vector = Vector<double>.Build.Dense(nx, 0.0);

for (int i = 1; i < nx - 1; i++)

{

vector[i] = solution[i, j, k] +

rx \* (solution[i - 1, j, k] - 2 \* solution[i, j, k] + solution[i + 1, j, k]);

}

return vector;

}

public static Vector<double> RightHandSideY(

double[,,] solution,

double alpha,

double dt,

double ry,

int ny,

int i,

int k,

double dx,

double dy,

double dz,

double t)

{

Vector<double> vector = Vector<double>.Build.Dense(ny, 0.0);

for (int j = 1; j < ny - 1; j++)

{

vector[j] = solution[i, j, k] +

ry \* (solution[i, j - 1, k] - 2 \* solution[i, j, k] + solution[i, j + 1, k]);

}

return vector;

}

public static Vector<double> RightHandSideZ(

double[,,] solution,

double alpha,

double dt,

double rz,

int nz,

int i,

int j,

double dx,

double dy,

double dz,

double t)

{

Vector<double> vector = Vector<double>.Build.Dense(nz, 0.0);

for (int k = 1; k < nz - 1; k++)

{

vector[k] = solution[i, j, k] +

rz \* (solution[i, j, k - 1] - 2 \* solution[i, j, k] + solution[i, j, k + 1]);

}

return vector;

}

}

}

## Листинг 4 - Класс Apply

using MathNet.Numerics.LinearAlgebra;

namespace PracticeLibrary

{

public class Apply

{

public static void BoundaryConditionsX(Matrix<double> matrix, Vector<double> vector, double rx, int nx, int j, double dx, Dictionary<string, (int type, double value)> boundaryConditions, Dictionary<string, (double alpha, double beta)> robinCoefficients)

{

// Граница X0

if (boundaryConditions.ContainsKey("x0"))

{

var bc = boundaryConditions["x0"];

if (bc.type == 1) // Нейман

{

matrix[0, 0] = 1;

matrix[0, 1] = -1;

vector[0] = vector[0]; // Сохраняем значение на границе

}

}

// Граница X1

if (boundaryConditions.ContainsKey("x1"))

{

var bc = boundaryConditions["x1"];

switch (bc.type)

{

case 0: // Дирихле

matrix.SetRow(nx - 1, Vector<double>.Build.Dense(nx));

matrix[nx - 1, nx - 1] = 1;

vector[nx - 1] = bc.value;

break;

case 1: // Нейман

matrix[nx - 1, nx - 1] = 1;

matrix[nx - 1, nx - 2] = -1;

vector[nx - 1] = vector[nx - 1];

break;

case 2: // Робин

if (robinCoefficients == null || !robinCoefficients.ContainsKey("x1"))

{

throw new ArgumentException("Robin coefficients are missing for x1 boundary.");

}

var robin = robinCoefficients["x1"];

double alpha = robin.alpha;

double beta = robin.beta;

matrix[nx - 1, nx - 1] = alpha \* (1 + 2 \* rx) + beta \* (2 \* rx / dx);

matrix[nx - 1, nx - 2] = -alpha \* (2 \* rx);

vector[nx - 1] = alpha \* vector[nx - 1] + 2 \* beta \* rx \* bc.value;

break;

}

}

else

{

Console.WriteLine("Boundary condition for x1 is missing. Setting to zero gradient");

matrix[nx - 1, nx - 1] = 1 + 2 \* rx;

matrix[nx - 1, nx - 2] = -2 \* rx;

vector[nx - 1] = vector[nx - 1];

}

}

public static void BoundaryConditionsY(Matrix<double> matrix, Vector<double> vector, double ry, int ny, double dy, Dictionary<string, (int type, double value)> boundaryConditions, Dictionary<string, (double alpha, double beta)> robinCoefficients)

{

// Граница Y0

if (boundaryConditions.ContainsKey("y0"))

{

var bc = boundaryConditions["y0"];

switch (bc.type)

{

case 0: // Дирихле

matrix.SetRow(0, Vector<double>.Build.Dense(ny));

matrix[0, 0] = 1;

vector[0] = bc.value;

break;

case 1: // Нейман

matrix[0, 0] = 1;

matrix[0, 1] = -1;

vector[0] = vector[0];

break;

case 2: // Робин

if (robinCoefficients == null || !robinCoefficients.ContainsKey("y0"))

{

throw new ArgumentException("Robin coefficients are missing for y0 boundary.");

}

var robin = robinCoefficients["y0"];

double alpha = robin.alpha;

double beta = robin.beta;

matrix[0, 0] = alpha \* (1 + 2 \* ry) + beta \* (2 \* ry / dy);

matrix[0, 1] = -alpha \* (2 \* ry);

vector[0] = alpha \* vector[0] + 2 \* beta \* ry \* bc.value;

break;

}

}

else

{

Console.WriteLine("Boundary condition for y0 is missing. Setting to zero gradient");

matrix[0, 0] = 1 + 2 \* ry;

matrix[0, 1] = -2 \* ry;

vector[0] = vector[0];

}

// Граница Y1

if (boundaryConditions.ContainsKey("y1"))

{

var bc = boundaryConditions["y1"];

switch (bc.type)

{

case 0: // Дирихле

matrix.SetRow(ny - 1, Vector<double>.Build.Dense(ny));

matrix[ny - 1, ny - 1] = 1;

vector[ny - 1] = bc.value;

break;

case 1: // Нейман

matrix[ny - 1, ny - 1] = 1;

matrix[ny - 1, ny - 2] = -1;

vector[ny - 1] = vector[ny - 1];

break;

case 2: // Робин

if (robinCoefficients == null || !robinCoefficients.ContainsKey("y1"))

{

throw new ArgumentException("Robin coefficients are missing for y1 boundary.");

}

var robin = robinCoefficients["y1"];

double alpha = robin.alpha;

double beta = robin.beta;

matrix[ny - 1, ny - 1] = alpha \* (1 + 2 \* ry) + beta \* (2 \* ry / dy);

matrix[ny - 1, ny - 2] = -alpha \* (2 \* ry);

vector[ny - 1] = alpha \* vector[ny - 1] + 2 \* beta \* ry \* bc.value;

break;

}

}

else

{

Console.WriteLine("Boundary condition for y1 is missing. Setting to zero gradient");

matrix[ny - 1, ny - 1] = 1 + 2 \* ry;

matrix[ny - 1, ny - 2] = -2 \* ry;

vector[ny - 1] = vector[ny - 1];

}

}

public static void BoundaryConditionsZ(Matrix<double> matrix, Vector<double> vector, double rz, int nz, double dz, Dictionary<string, (int type, double value)> boundaryConditions, Dictionary<string, (double alpha, double beta)> robinCoefficients)

{

// Граница Z0

if (boundaryConditions.ContainsKey("z0"))

{

var bc = boundaryConditions["z0"];

switch (bc.type)

{

case 0: // Дирихле

matrix.SetRow(0, Vector<double>.Build.Dense(nz));

matrix[0, 0] = 1;

vector[0] = bc.value;

break;

case 1: // Нейман

matrix[0, 0] = 1;

matrix[0, 1] = -1;

vector[0] = vector[0];

break;

case 2: // Робин

if (robinCoefficients == null || !robinCoefficients.ContainsKey("z0"))

{

throw new ArgumentException("Robin coefficients are missing for z0 boundary.");

}

var robin = robinCoefficients["z0"];

double alpha = robin.alpha;

double beta = robin.beta;

matrix[0, 0] = alpha \* (1 + 2 \* rz) + beta \* (2 \* rz / dz);

matrix[0, 1] = -alpha \* (2 \* rz);

vector[0] = alpha \* vector[0] + 2 \* beta \* rz \* bc.value;

break;

}

}

else

{

Console.WriteLine("Boundary condition for z0 is missing. Setting to zero gradient");

matrix[0, 0] = 1 + 2 \* rz;

matrix[0, 1] = -2 \* rz;

vector[0] = vector[0];

}

// Граница Z1

if (boundaryConditions.ContainsKey("z1"))

{

var bc = boundaryConditions["z1"];

switch (bc.type)

{

case 0: // Дирихле

matrix.SetRow(nz - 1, Vector<double>.Build.Dense(nz));

matrix[nz - 1, nz - 1] = 1;

vector[nz - 1] = bc.value;

break;

case 1: // Нейман

matrix[nz - 1, nz - 1] = 1;

matrix[nz - 1, nz - 2] = -1;

vector[nz - 1] = vector[nz - 1];

break;

case 2: // Робин

if (robinCoefficients == null || !robinCoefficients.ContainsKey("z1"))

{

throw new ArgumentException("Robin coefficients are missing for z1 boundary.");

}

var robin = robinCoefficients["z1"];

double alpha = robin.alpha;

double beta = robin.beta;

matrix[nz - 1, nz - 1] = alpha \* (1 + 2 \* rz) + beta \* (2 \* rz / dz);

matrix[nz - 1, nz - 2] = -alpha \* (2 \* rz);

vector[nz - 1] = alpha \* vector[nz - 1] + 2 \* beta \* rz \* bc.value;

break;

}

}

else

{

Console.WriteLine("Boundary condition for z1 is missing. Setting to zero gradient");

matrix[nz - 1, nz - 1] = 1 + 2 \* rz;

matrix[nz - 1, nz - 2] = -2 \* rz;

vector[nz - 1] = vector[nz - 1];

}

}

}

}

## Листинг 5 - Класс Result

namespace PracticeLibrary

{

public class Results

{

public static void WriteToFile(double[,,] solution)

{

string filename = "Result.txt";

try

{

using (StreamWriter writer = new StreamWriter(filename))

{

int nx = solution.GetLength(0);

int ny = solution.GetLength(1);

int nz = solution.GetLength(2);

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

{

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

{

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

{

writer.WriteLine($"{i} {j} {k} {solution[i, j, k]}"); // Сохраняем координаты и температуру

}

}

}

}

Console.Clear();

Console.WriteLine($"Решение записано в {filename}");

}

catch (Exception e)

{

Console.Clear();

Console.WriteLine($"Error writing to file: {e.Message}");

}

}

public static void PrintSlice(double[,,] solution)

{

while (true)

{

try

{

Console.Clear();

Console.WriteLine("\n1) Выберите ось:\nX\nY\nZ\n\nQ - Назад");

string axis = Console.ReadLine();

if (axis == "q" || axis == "Q" || axis == "й" || axis == "Й") break;

int size = axis switch

{

"x" or "X" => solution.GetLength(0),

"y" or "Y" => solution.GetLength(1),

"z" or "Z" => solution.GetLength(2),

\_ => throw new ArgumentException("Недопустимая ось. Используйте 'x', 'y' или 'z'")

};

Console.Clear();

Console.WriteLine($"\nПослойные срезы по оси {axis.ToUpper()}:");

Thread.Sleep(1000);

Console.WriteLine("(Для навигации используйте стрелки, для выхода - Esc)\n");

Thread.Sleep(2000);

int currentSlice = 0;

ConsoleKey key;

while (true)

{

Console.Clear();

PrintSingleSlice(solution, axis, currentSlice);

Console.WriteLine($"\nСрез {axis.ToUpper()} = {currentSlice} (Всего: {size - 1})");

Console.WriteLine("<- -> - листать слои | | Esc - выход");

key = Console.ReadKey(true).Key;

if (key == ConsoleKey.Escape)

break;

currentSlice = key switch

{

ConsoleKey.LeftArrow => currentSlice == 0 ? currentSlice : currentSlice - 1,

ConsoleKey.RightArrow => currentSlice == size - 1 ? currentSlice : currentSlice + 1

};

}

}

catch(ArgumentException ex)

{

Console.WriteLine(ex);

Console.ReadLine();

}

}

}

private static void PrintSingleSlice(double[,,] solution, string axis, int slice)

{

switch (axis.ToLower())

{

case "x":

PrintSliceYZ(solution, slice);

break;

case "y":

PrintSliceXZ(solution, slice);

break;

case "z":

PrintSliceXY(solution, slice);

break;

}

}

private static void PrintSliceXY(double[,,] solution, int z)

{

Console.WriteLine($"┌{new string('─', solution.GetLength(1) \* 10 + 1)}┐");

Console.Write("│ X→ ");

for (int x = 0; x < solution.GetLength(0); x++) Console.Write($"{x,8}");

Console.WriteLine(" │");

Console.WriteLine($"├{new string('─', solution.GetLength(1) \* 10 + 1)}┤");

for (int y = 0; y < solution.GetLength(1); y++)

{

Console.Write($"│ Y{y,2} │");

for (int x = 0; x < solution.GetLength(0); x++)

{

Console.Write($"{solution[x, y, z],8:F2}");

}

Console.WriteLine(" │");

}

Console.WriteLine($"└{new string('─', solution.GetLength(1) \* 10 + 1)}┘");

}

private static void PrintSliceXZ(double[,,] solution, int y)

{

Console.WriteLine($"┌{new string('─', solution.GetLength(0) \* 10 + 1)}┐");

Console.Write("│ X→ ");

for (int x = 0; x < solution.GetLength(0); x++) Console.Write($"{x,8}");

Console.WriteLine(" │");

Console.WriteLine($"├{new string('─', solution.GetLength(0) \* 10 + 1)}┤");

for (int z = 0; z < solution.GetLength(2); z++)

{

Console.Write($"│ Z{z,2} │");

for (int x = 0; x < solution.GetLength(0); x++)

{

Console.Write($"{solution[x, y, z],8:F2}");

}

Console.WriteLine(" │");

}

Console.WriteLine($"└{new string('─', solution.GetLength(0) \* 10 + 1)}┘");

}

private static void PrintSliceYZ(double[,,] solution, int x)

{

Console.WriteLine($"┌{new string('─', solution.GetLength(1) \* 10 + 1)}┐");

Console.Write("│ Y→ ");

for (int y = 0; y < solution.GetLength(1); y++) Console.Write($"{y,8}");

Console.WriteLine(" │");

Console.WriteLine($"├{new string('─', solution.GetLength(1) \* 10 + 1)}┤");

for (int z = 0; z < solution.GetLength(2); z++)

{

Console.Write($"│ Z{z,2} │");

for (int y = 0; y < solution.GetLength(1); y++)

{

Console.Write($"{solution[x, y, z],8:F2}");

}

Console.WriteLine(" │");

}

Console.WriteLine($"└{new string('─', solution.GetLength(1) \* 10 + 1)}┘");

}

}

}

## Листинг 5 - Класс Config

namespace PracticeLibrary

{

public static class Config

{

public static int nx { get; set; } = 10;

public static int ny { get; set; } = 10;

public static int nz { get; set; } = 10;

public static double lx { get; set; } = 1.0;

public static double ly { get; set; } = 1.0;

public static double lz { get; set; } = 1.0;

public static double tMax { get; set; } = 0.1;

public static double dt { get; set; } = 0.001;

public static double algaeDiffusion { get; set; } = 0.1;

public static double shrimpDiffusion { get; set; } = 0.1;

public static double dx = lx / (nx - 1);

public static double dy = ly / (ny - 1);

public static double dz = lz / (nz - 1);

}

}

# 5. Пример работы программы

Начальные условия:

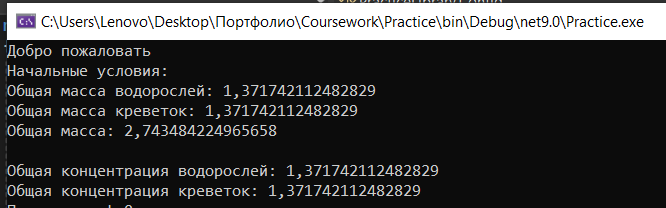


Рисунок 5.1 – Вывод массы и концентрации.

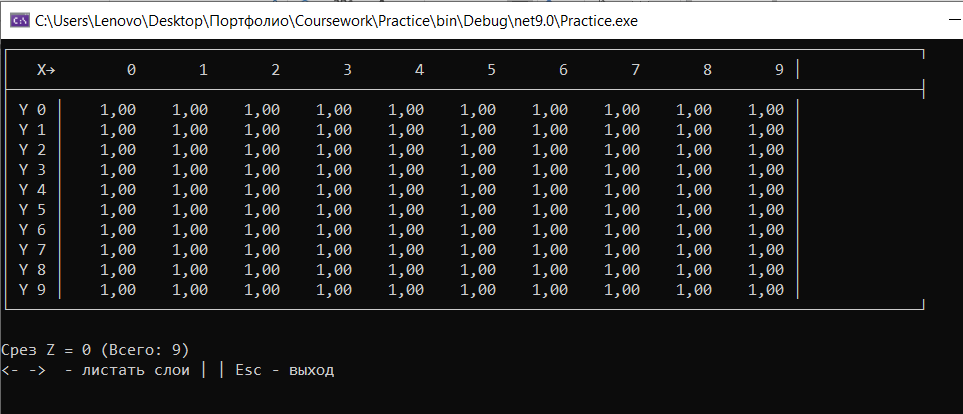


Рисунок 5.2 – Вывод начального распределения водорослей.

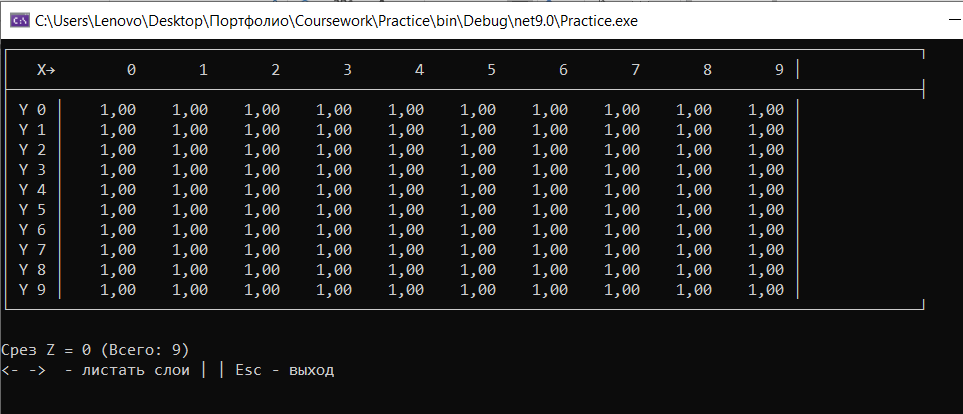


Рисунок 5.3 – Вывод начального распределения креветок.

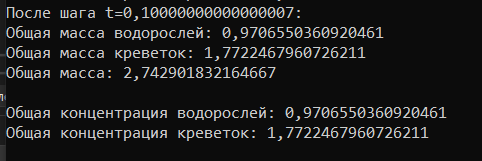


Рисунок 5.4 – Конечный вывод массы и концентрации.

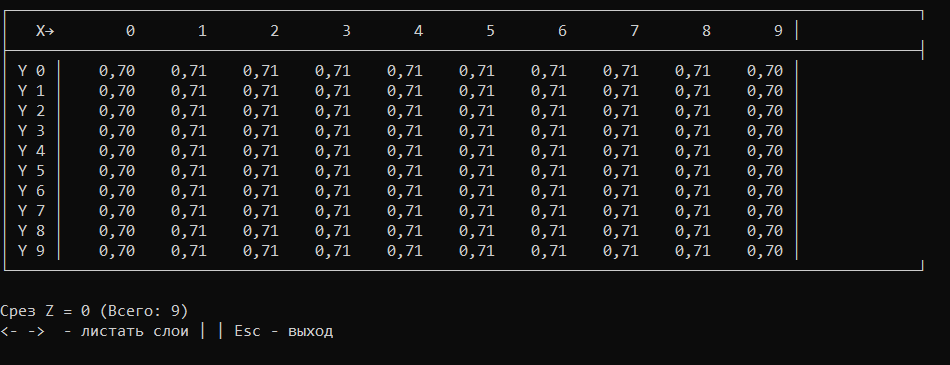


Рисунок 5.5 – Вывод конечного распределения водорослей.

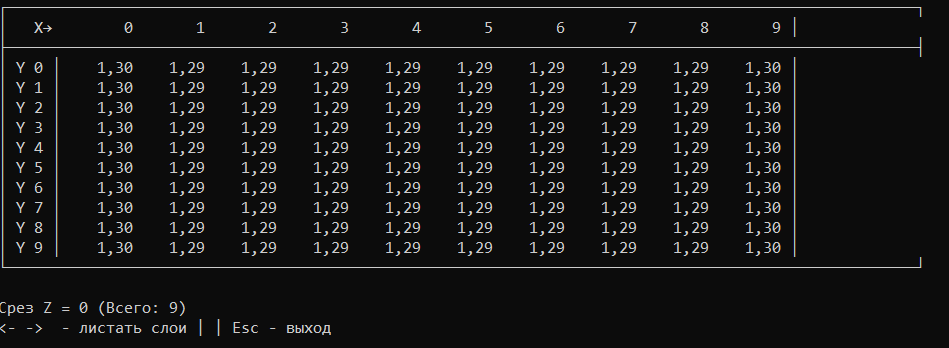


Рисунок 5.6 – Вывод конечного распределения водорослей.

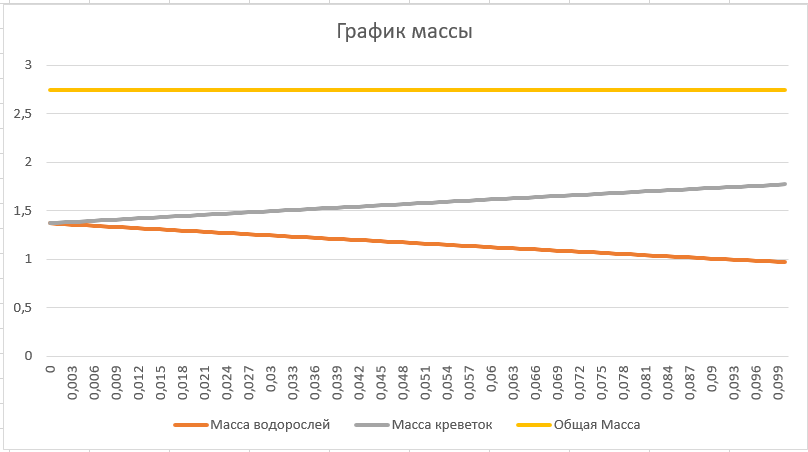


Рисунок 5.7 –График изменения массы.

# 6. Заключение

Программа успешно решает систему уравнений для модели взаимодействия водорослей и креветок, учитывая:

* Диффузию в трёхмерной среде.
* Взаимодействие видов.
* Граничные условия Неймана.

Код структурирован и может быть расширен для более сложных моделей.

# 7. Список литературы

1. Самарский А.А., Вабищевич П.Н. Численные методы решения задач теплопроводности и диффузии.

2. Тихонов А.Н., Самарский А.А. Уравнения математической физики.

3. Мюррей Дж. Математическая биология.

4. Самарский А.А. Численные методы.