

**НЕГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ ЧАСТНОЕ** **УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ** **«МОСКОВСКИЙ ФИНАНСОВО-ПРОМЫШЛЕННЫЙ УНИВЕРСИТЕТ** **«СИНЕРГИЯ»**

|  |  |  |
| --- | --- | --- |
| **Факультет/Институт** |  | информационных технологий |
|  |  | (наименование факультета/ Института) |
| **Направление/специальность** |  | 09.02.07 Информационные системы и программирование |
| **подготовки:** |  | (код и наименование направления /специальности подготовки) |
| **Форма обучения:** |  | очная |
|  |  | (очная, очно-заочная, заочная) |
|  |  |  |

.

**Отчет по лабораторной работе № 5**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **на тему** | |  | Проектирование классов | | | | |
|  | | | | | |  | (наименование темы) |
|  |  | |  | | | | |
| **по дисциплине** | | | |  | Разработка программных модулей | | |
|  | | | | | |  | (наименование дисциплины) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Обучающийся** |  | Чилингиров Нури Эскендерович |  |  |
|  |  | (ФИО) |  | (подпись) |
| **Группа** |  | ДКИП-112 Прог |  |
|  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Преподаватель** |  | Сибирев Иван Валерьевич |  |  |
|  |  | (ФИО) |  | (подпись) |

**Москва, 2025**

**7** Вариант

Написать программу по созданию, добавлению (в начало, в конец), просмотру (с начала, с конца) и решению приведенной в подразделе 3.3 задачи для двунаправленных линейных списков

Листинг кода:

using System;

// Класс узла двунаправленного списка

class ListNode<T>

{

public T Data { get; set; }

public ListNode<T> Next { get; set; }

public ListNode<T> Previous { get; set; }

public ListNode(T data)

{

Data = data;

Next = null;

Previous = null;

}

}

// Класс двунаправленного списка

class DoublyLinkedList<T>

{

private ListNode<T> head;

private ListNode<T> tail;

public int Count { get; private set; }

public DoublyLinkedList()

{

head = null;

tail = null;

Count = 0;

}

// Добавление в начало списка

public void AddFirst(T data)

{

ListNode<T> newNode = new ListNode<T>(data);

if (head == null)

{

head = newNode;

tail = newNode;

}

else

{

newNode.Next = head;

head.Previous = newNode;

head = newNode;

}

Count++;

}

// Добавление в конец списка

public void AddLast(T data)

{

ListNode<T> newNode = new ListNode<T>(data);

if (tail == null)

{

head = newNode;

tail = newNode;

}

else

{

newNode.Previous = tail;

tail.Next = newNode;

tail = newNode;

}

Count++;

}

// Просмотр с начала списка

public void DisplayForward()

{

Console.WriteLine("Список (с начала):");

ListNode<T> current = head;

while (current != null)

{

Console.Write(current.Data + " ");

current = current.Next;

}

Console.WriteLine();

}

// Просмотр с конца списка

public void DisplayBackward()

{

Console.WriteLine("Список (с конца):");

ListNode<T> current = tail;

while (current != null)

{

Console.Write(current.Data + " ");

current = current.Previous;

}

Console.WriteLine();

}

// Решение задачи из подраздела 3.3 (Вариант 7)

public void SolveTask()

{

if (Count < 2)

{

Console.WriteLine("Для решения задачи нужно минимум 2 элемента в списке");

return;

}

// Задача: Найти сумму элементов между первым и последним отрицательными элементами

ListNode<T> firstNegative = FindFirstNegative();

ListNode<T> lastNegative = FindLastNegative();

if (firstNegative == null || lastNegative == null || firstNegative == lastNegative)

{

Console.WriteLine("В списке нет двух разных отрицательных элементов");

return;

}

double sum = 0;

ListNode<T> current = firstNegative.Next;

while (current != lastNegative)

{

if (current.Data is double)

{

sum += (double)(object)current.Data;

}

current = current.Next;

}

Console.WriteLine($"Сумма элементов между первым и последним отрицательными: {sum}");

}

private ListNode<T> FindFirstNegative()

{

ListNode<T> current = head;

while (current != null)

{

if (IsNegative(current.Data))

return current;

current = current.Next;

}

return null;

}

private ListNode<T> FindLastNegative()

{

ListNode<T> current = tail;

while (current != null)

{

if (IsNegative(current.Data))

return current;

current = current.Previous;

}

return null;

}

private bool IsNegative(object value)

{

if (value is double d) return d < 0;

if (value is int i) return i < 0;

if (value is float f) return f < 0;

return false;

}

}

class Program

{

static void Main()

{

DoublyLinkedList<double> list = new DoublyLinkedList<double>();

Console.WriteLine("Демонстрация работы с двунаправленным списком (Вариант 7)");

Console.WriteLine("--------------------------------------------------------");

// Добавление элементов

list.AddFirst(3.5);

list.AddFirst(-2.1);

list.AddLast(4.7);

list.AddLast(-1.8);

list.AddLast(5.2);

list.AddFirst(1.0);

// Просмотр списка

list.DisplayForward();

list.DisplayBackward();

// Решение задачи

list.SolveTask();

// Дополнительные операции

Console.WriteLine("\nДобавляем -3.0 в начало и 2.4 в конец:");

list.AddFirst(-3.0);

list.AddLast(2.4);

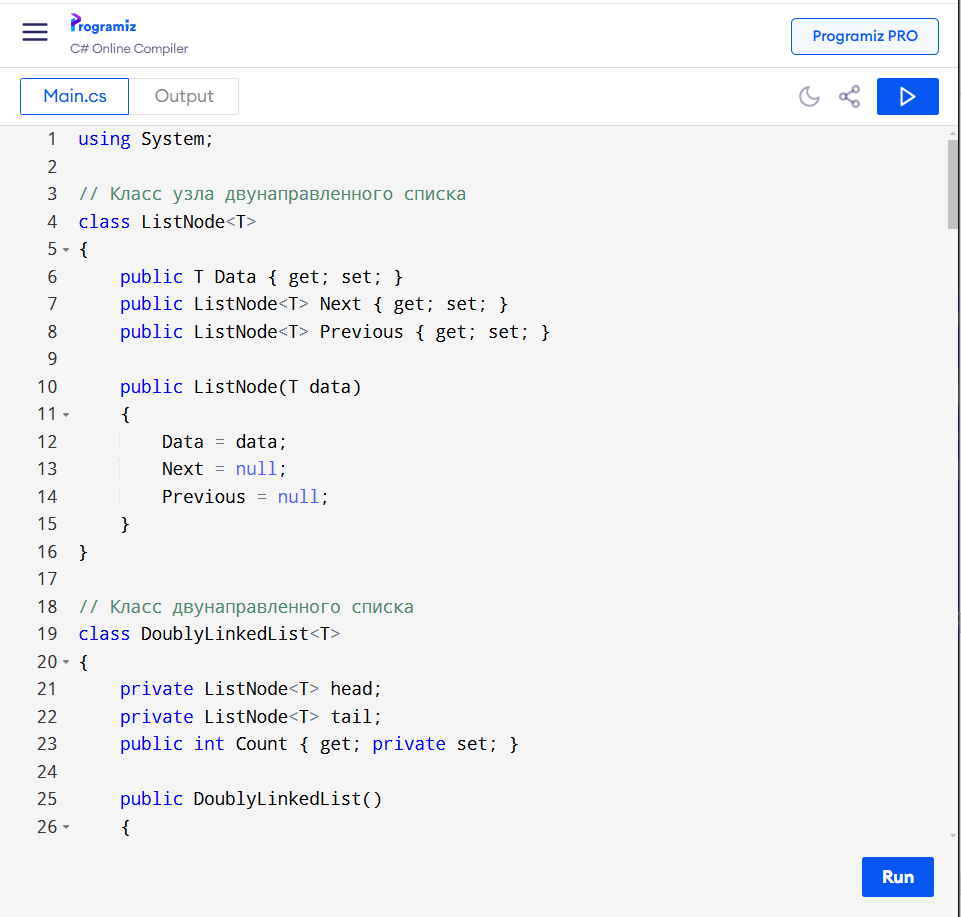
list.DisplayForward();

list.SolveTask();

}

}

Скрин кода:



2. требуется аппроксимировать заданную исходную функцию f(x) многочленом на интервале [a, b]. Задано количество неизвестных параметров n, вид аппроксимации и m – количество точек, в которых задана функция. Таблица исходной функции yi=f(xi) вычисляется в точках ( 1)( )/( 1), 1, . i x a i b a m i m = + − − − = Используя полученную таблицу ( , ), i i x y требуется вычислить значения функций ( ), ( , ) j j f x x c ϕ и погрешность ( ) ( ) ( , ) j j j d x f x x c = − ϕ в точках ( 1)( )/ 20; 1, 21 j x a j b a j = + − − = , построить графики и проанализировать качество полученной аппроксимации

Листинг кода: using System;

using System.Linq;

class FunctionApproximation

{

static void Main()

{

Console.WriteLine("Аппроксимация функции многочленом");

Console.WriteLine("Исходная функция: f(x) = x - 5\*sin^2(x)");

Console.WriteLine("----------------------------------------");

// Параметры задачи

double a = 1.0; // Начало интервала

double b = 4.0; // Конец интервала

int n = 4; // Степень многочлена (количество параметров - 1)

int m = 11; // Количество точек исходной функции

string method = "MHK"; // Метод аппроксимации (МНК)

// 1. Создаем таблицу исходной функции

(double[] x, double[] y) = GenerateFunctionTable(a, b, m);

Console.WriteLine("\nТаблица исходной функции:");

Console.WriteLine("i\tx\_i\t\ty\_i");

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

{

Console.WriteLine($"{i+1}\t{x[i]:F4}\t{y[i]:F4}");

}

// 2. Аппроксимируем функцию многочленом

double[] coefficients = LeastSquaresApproximation(x, y, n);

Console.WriteLine("\nКоэффициенты аппроксимирующего многочлена:");

for (int i = 0; i < coefficients.Length; i++)

{

Console.WriteLine($"c\_{i} = {coefficients[i]:F6}");

}

// 3. Вычисляем значения в контрольных точках

int pointsCount = 21;

(double[] xTest, double[] yExact, double[] yApprox, double[] errors) =

CalculateTestPoints(a, b, pointsCount, coefficients);

Console.WriteLine("\nРезультаты аппроксимации в контрольных точках:");

Console.WriteLine("j\tx\_j\t\tf(x\_j)\t\tφ(x\_j)\t\tПогрешность");

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

{

Console.WriteLine($"{j+1}\t{xTest[j]:F4}\t{yExact[j]:F4}\t{yApprox[j]:F4}\t{errors[j]:E2}");

}

// 4. Анализ качества аппроксимации

double maxError = errors.Max();

double avgError = errors.Average();

double rmsError = Math.Sqrt(errors.Select(e => e \* e).Average());

Console.WriteLine("\nАнализ качества аппроксимации:");

Console.WriteLine($"Максимальная погрешность: {maxError:E2}");

Console.WriteLine($"Средняя погрешность: {avgError:E2}");

Console.WriteLine($"Среднеквадратичная погрешность: {rmsError:E2}");

// 5. Вывод данных для построения графиков

Console.WriteLine("\nДанные для построения графиков:");

Console.WriteLine("x\tf(x)\tφ(x)");

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

{

Console.WriteLine($"{xTest[j]:F4}\t{yExact[j]:F4}\t{yApprox[j]:F4}");

}

}

// Генерация таблицы исходной функции

static (double[] x, double[] y) GenerateFunctionTable(double a, double b, int m)

{

double[] x = new double[m];

double[] y = new double[m];

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

{

x[i] = a + i \* (b - a) / (m - 1);

y[i] = OriginalFunction(x[i]);

}

return (x, y);

}

// Исходная функция

static double OriginalFunction(double x)

{

return x - 5 \* Math.Pow(Math.Sin(x), 2);

}

// Метод наименьших квадратов

static double[] LeastSquaresApproximation(double[] x, double[] y, int n)

{

// Создаем матрицу для системы уравнений

double[,] A = new double[n+1, n+1];

double[] B = new double[n+1];

// Заполняем матрицу A и вектор B

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

{

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

{

A[k, l] = x.Sum(xi => Math.Pow(xi, k + l));

}

B[k] = x.Select((xi, i) => Math.Pow(xi, k) \* y[i]).Sum();

}

// Решаем систему линейных уравнений (используем упрощенный метод Гаусса)

return SolveLinearSystem(A, B);

}

// Решение системы линейных уравнений (упрощенная реализация)

static double[] SolveLinearSystem(double[,] A, double[] B)

{

int n = B.Length;

double[] result = new double[n];

// Прямой ход метода Гаусса

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

{

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

{

double factor = A[i, k] / A[k, k];

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

{

A[i, j] -= factor \* A[k, j];

}

B[i] -= factor \* B[k];

}

}

// Обратный ход метода Гаусса

for (int i = n - 1; i >= 0; i--)

{

result[i] = B[i];

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

{

result[i] -= A[i, j] \* result[j];

}

result[i] /= A[i, i];

}

return result;

}

// Вычисление значений в контрольных точках

static (double[] xTest, double[] yExact, double[] yApprox, double[] errors)

CalculateTestPoints(double a, double b, int pointsCount, double[] coefficients)

{

double[] xTest = new double[pointsCount];

double[] yExact = new double[pointsCount];

double[] yApprox = new double[pointsCount];

double[] errors = new double[pointsCount];

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

{

xTest[j] = a + j \* (b - a) / (pointsCount - 1);

yExact[j] = OriginalFunction(xTest[j]);

yApprox[j] = ApproximateFunction(xTest[j], coefficients);

errors[j] = Math.Abs(yExact[j] - yApprox[j]);

}

return (xTest, yExact, yApprox, errors);

}

// Аппроксимирующая функция (многочлен)

static double ApproximateFunction(double x, double[] coefficients)

{

double result = 0;

for (int i = 0; i < coefficients.Length; i++)

{

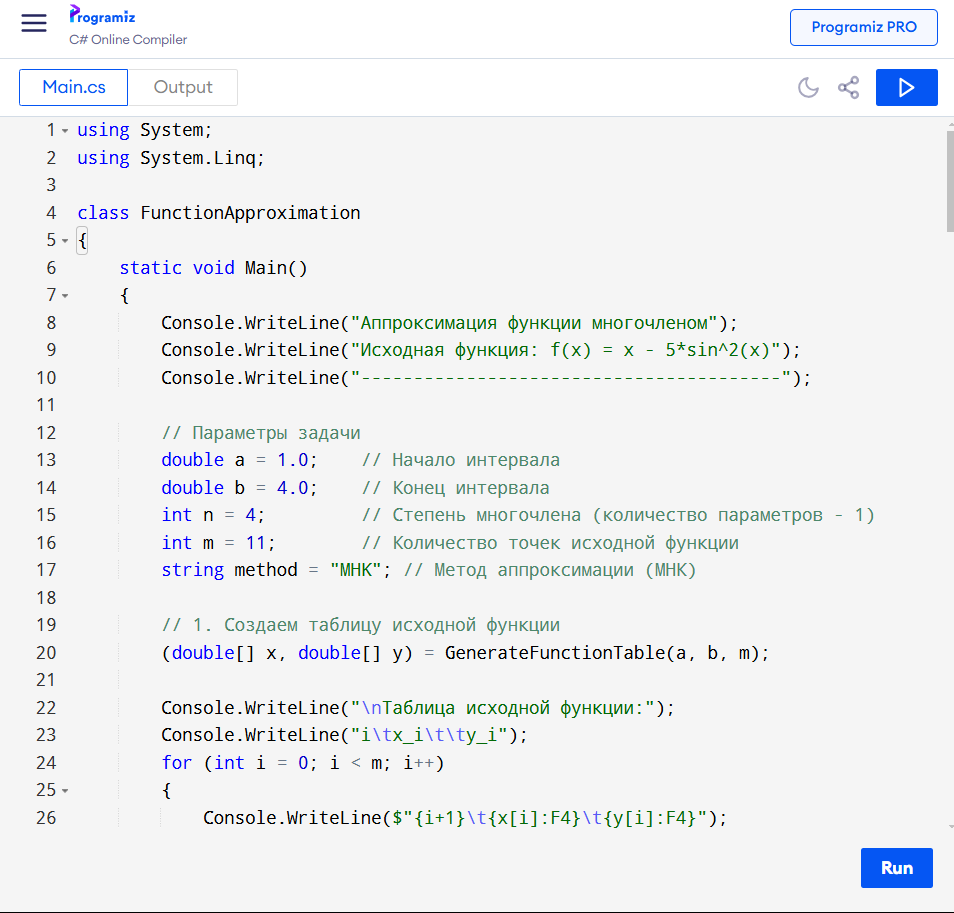
result += coefficients[i] \* Math.Pow(x, i);

}

return result;

}

}

Скрин кода: 

3. По схеме, приведенной на рис. 9.9, отладить программу определения всех корней функции f(x) в указанном интервале [a, b], использовать метод в соответствии с полученным вариантом из табл. 9.1

Листинг кода:

using System;

class RootFinder

{

static void Main()

{

Console.WriteLine("Нахождение корней функции методом итераций");

Console.WriteLine("Функция: f(x) = x - 5\*sin²(x) - 5");

Console.WriteLine("Интервал: [3, 9]");

Console.WriteLine("----------------------------------------");

// Параметры метода

double a = 3.0;

double b = 9.0;

double epsilon = 1e-6;

int maxIterations = 100;

// Находим корни

FindRoots(a, b, epsilon, maxIterations);

}

// Исходная функция

static double Function(double x)

{

return x - 5 \* Math.Pow(Math.Sin(x), 2) - 5;

}

// Функция для метода итераций (преобразованная форма)

static double IterationFunction(double x)

{

return 5 \* Math.Pow(Math.Sin(x), 2) + 5;

}

// Метод итераций для нахождения корня

static double IterationMethod(double x0, double epsilon, int maxIterations)

{

double x = x0;

double xPrev;

int iterations = 0;

Console.WriteLine($"\nНачальное приближение: x0 = {x0:F6}");

Console.WriteLine("n\tx\_n\t\tf(x\_n)\t\t|x\_n - x\_{n-1}|");

do

{

xPrev = x;

x = IterationFunction(xPrev);

iterations++;

Console.WriteLine($"{iterations}\t{x:F6}\t{Function(x):F6}\t{Math.Abs(x - xPrev):F6}");

if (iterations >= maxIterations)

{

Console.WriteLine("Достигнуто максимальное количество итераций");

return double.NaN;

}

} while (Math.Abs(x - xPrev) > epsilon);

Console.WriteLine($"Корень найден за {iterations} итераций");

return x;

}

// Поиск всех корней на интервале

static void FindRoots(double a, double b, double epsilon, int maxIterations)

{

// Шаг для поиска смены знака функции

double step = (b - a) / 100;

double x1 = a;

double x2 = x1 + step;

int rootCount = 0;

Console.WriteLine("\nПоиск корней на интервале:");

while (x2 <= b)

{

// Если функция меняет знак, значит есть корень

if (Function(x1) \* Function(x2) <= 0)

{

rootCount++;

Console.WriteLine($"\nНайден интервал с корнем: [{x1:F6}, {x2:F6}]");

// Выбираем начальное приближение

double x0 = (x1 + x2) / 2;

double root = IterationMethod(x0, epsilon, maxIterations);

if (!double.IsNaN(root))

{

Console.WriteLine($"Корень {rootCount}: {root:F6}");

Console.WriteLine($"f({root:F6}) = {Function(root):F6}");

}

// Переходим к следующему интервалу

x1 = x2;

}

else

{

x1 += step;

}

x2 = x1 + step;

if (x2 > b) x2 = b;

}

if (rootCount == 0)

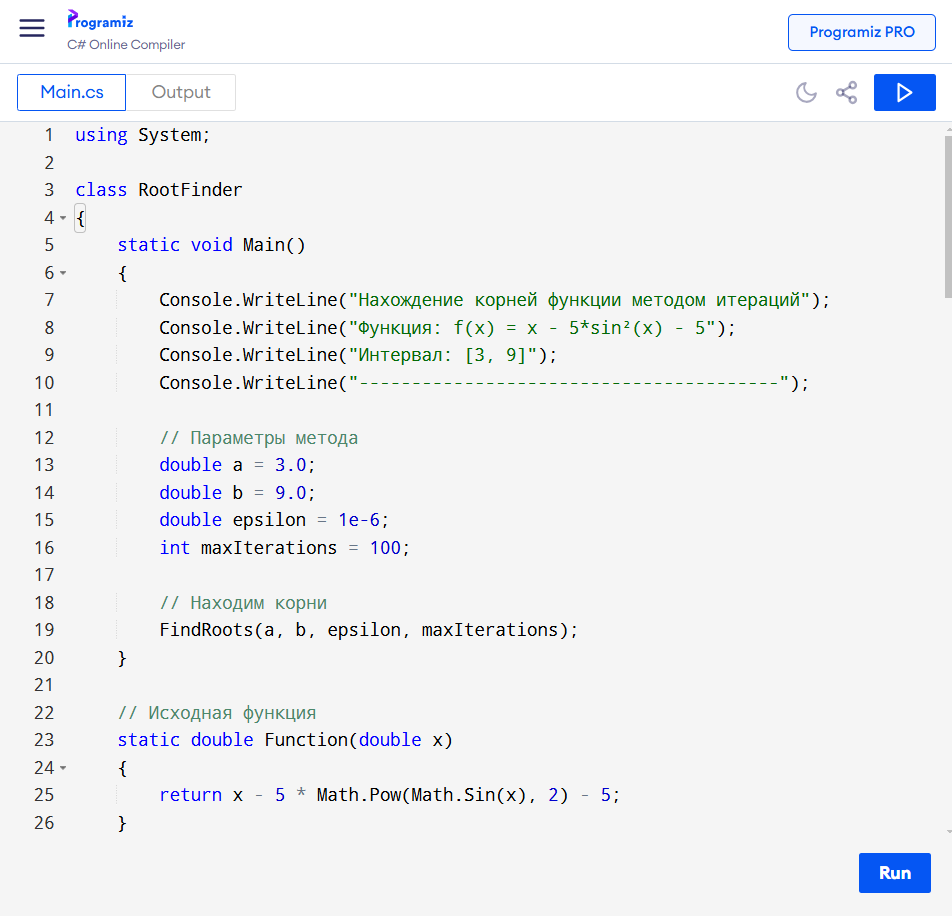
{

Console.WriteLine("Корни не найдены на заданном интервале");

}

}

}

Скрин кода: 

4. Написать и отладить программу вычисления интеграла указанным методом двумя способами – по заданному количеству разбиений n и заданной точности ε (метод 1). Реализацию указанного метода оформить отдельной функцией, алгоритм которой описать в виде блок-схемы

Листинг кода:

using System;

class IntegralCalculation

{

static void Main()

{

Console.WriteLine("Вычисление интеграла методом средних прямоугольников");

Console.WriteLine("Функция: f(x) = x - 5\*sin²(x)");

Console.WriteLine("Интервал: [1, 4]");

Console.WriteLine("----------------------------------------");

// Параметры интегрирования

double a = 1.0;

double b = 4.0;

double epsilon = 0.100;

int initialN = 10;

// 1. Вычисление по заданному количеству разбиений

Console.WriteLine("\n1. Вычисление по заданному количеству разбиений (n = {0})", initialN);

double resultFixedN = MidpointRectangleMethod(a, b, initialN);

Console.WriteLine("Результат: {0:F6}", resultFixedN);

// 2. Вычисление с заданной точностью

Console.WriteLine("\n2. Вычисление с заданной точностью (ε = {0})", epsilon);

double resultAdaptive = MidpointRectangleMethodAdaptive(a, b, epsilon);

Console.WriteLine("Итоговый результат: {0:F6}", resultAdaptive);

}

// Исходная функция

static double Function(double x)

{

return x - 5 \* Math.Pow(Math.Sin(x), 2);

}

// Метод средних прямоугольников (фиксированное количество разбиений)

static double MidpointRectangleMethod(double a, double b, int n)

{

double h = (b - a) / n;

double sum = 0.0;

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

{

double x = a + (i + 0.5) \* h;

sum += Function(x);

}

return sum \* h;

}

// Метод средних прямоугольников с автоматическим выбором количества разбиений

static double MidpointRectangleMethodAdaptive(double a, double b, double epsilon)

{

int n = 4; // Начальное количество разбиений

double prevResult = 0;

double currentResult = MidpointRectangleMethod(a, b, n);

int iteration = 1;

Console.WriteLine("Итерация\tРазбиений\tРезультат\tПогрешность");

Console.WriteLine("{0}\t\t{1}\t\t{2:F6}\t-", iteration, n, currentResult);

do

{

n \*= 2; // Удваиваем количество разбиений

prevResult = currentResult;

currentResult = MidpointRectangleMethod(a, b, n);

iteration++;

Console.WriteLine("{0}\t\t{1}\t\t{2:F6}\t{3:F6}",

iteration, n, currentResult, Math.Abs(currentResult - prevResult));

}

while (Math.Abs(currentResult - prevResult) > epsilon && n < 1000000);

return currentResult;

}

}

