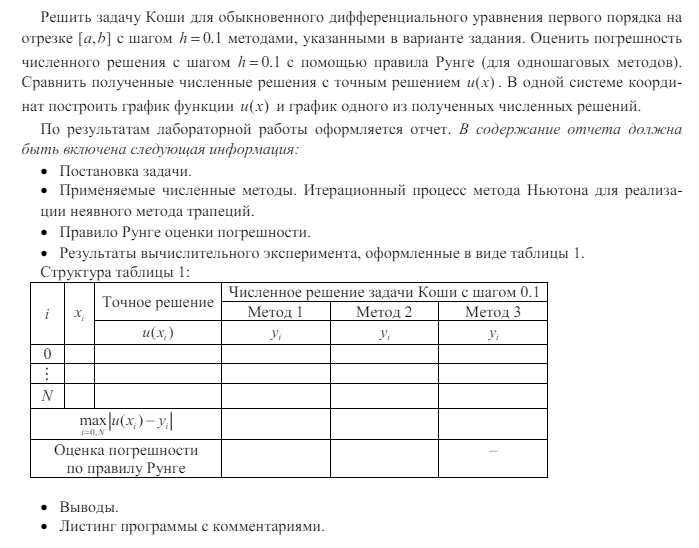
Отчет по дисциплине: «Численные методы»

Лабораторная работа №4

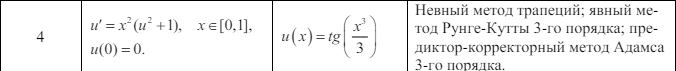
## «Численные методы решения задачи Коши»

Подготовил студент 3 курса 4 группы

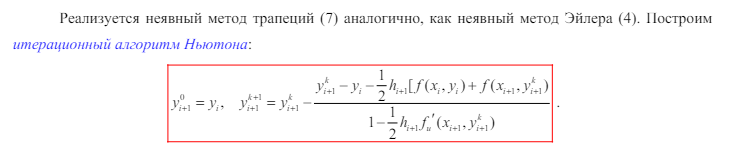
Кондратович Артём

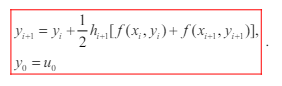


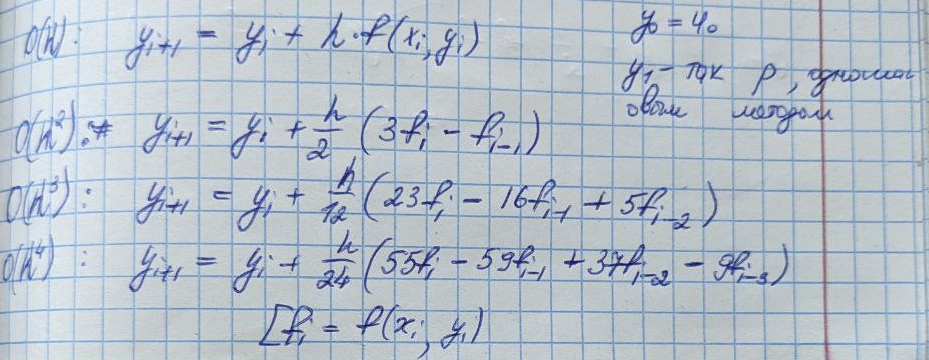
Вариант 4

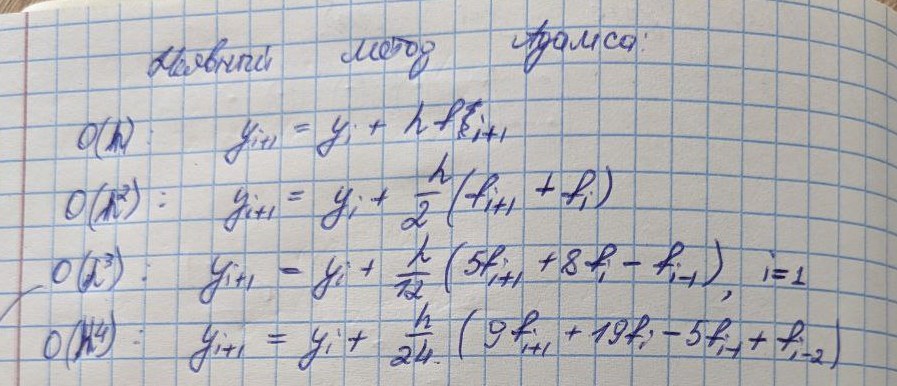


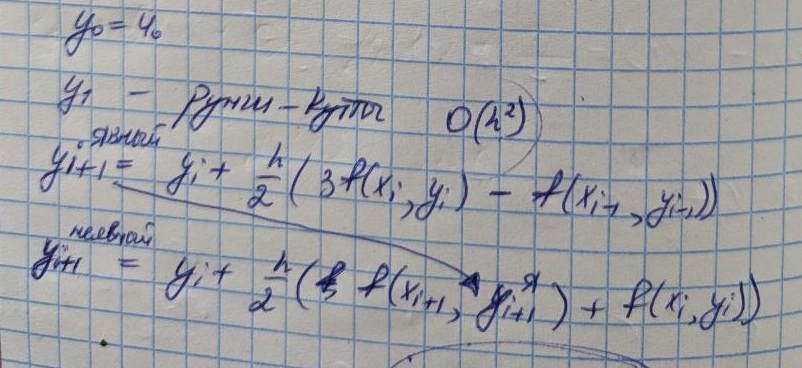
Теория:



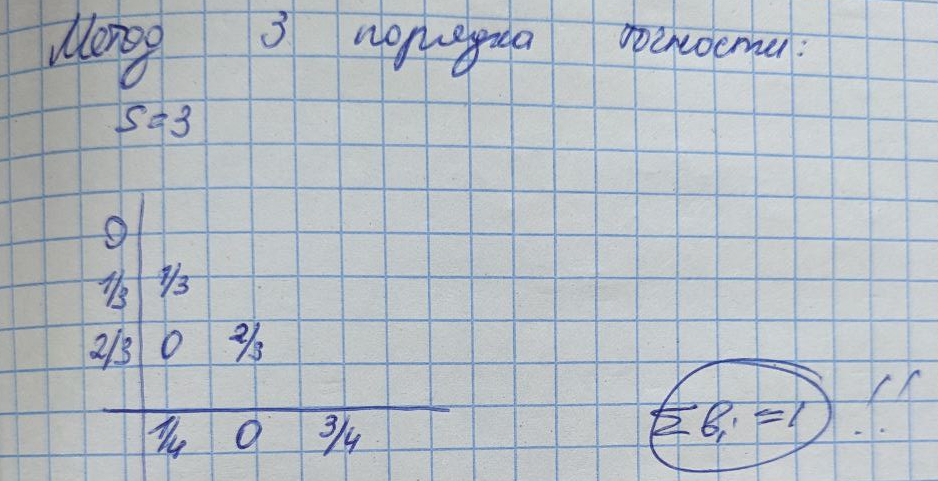


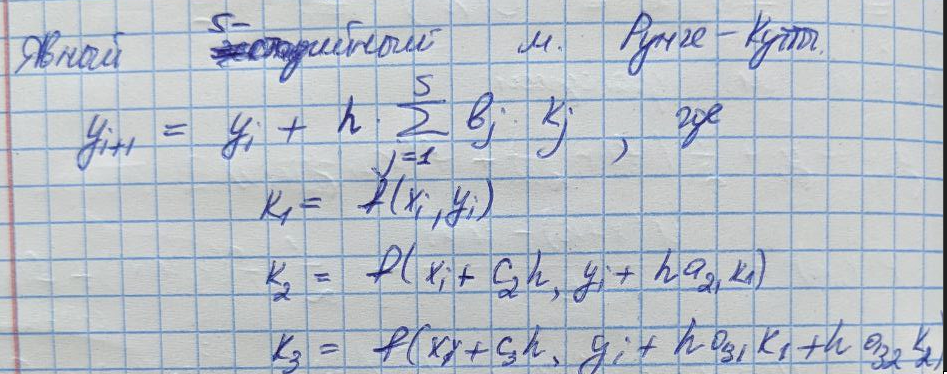






С поправкой, что в моём варианте 3 порядок точности





Листинг программы:

Methods.cs

namespace lab\_4.HelpClasses

{

public static class Methods

{

public static IEnumerable<double> SolveTrapezoidal(Func<double, double, double> f, Func<double, double, double> dfdu, double a, double b, double h, double u0, double epsilon)

{

var x = GetNodes(a, b, h).ToArray();

double[] u = new double[x.Length];

u[0] = u0;

for (int i = 1; i < x.Length; i++)

{

double uiPrev = u[i - 1];

double ui = uiPrev;

while (Math.Abs(ui - uiPrev).CompareTo(epsilon) <= 0)

{

uiPrev = ui;

ui = uiPrev - (uiPrev - u[i - 1] - h / 2 \* (f(x[i - 1], u[i - 1]) + f(x[i], uiPrev)))

/ (1 - h / 2 \* dfdu(x[i], uiPrev));

}

u[i] = u[i - 1] + h / 2 \* (f(x[i-1], u[i-1]) + f(x[i], ui));

}

return u;

}

public static IEnumerable<double> SolveRungeKutta(Func<double, double, double> f, double a, double b, double h, double u0)

{

var x = GetNodes(a, b, h).ToArray();

double[] u = new double[x.Length];

u[0] = u0;

for (int i = 1; i < x.Length; i++)

{

double xi = x[i - 1];

double uiPrev = u[i - 1];

double k1 = h \* f(xi, uiPrev);

double k2 = h \* f(xi + h / 3, uiPrev + k1 / 3);

double k3 = h \* f(xi + 2 \* h / 3, uiPrev + 2 \* k2 / 3);

double deltaUi = (k1 + 3 \* k3) / 4;

u[i] = uiPrev + deltaUi;

}

return u;

}

public static IEnumerable<double> SolveAdamsBashforthMoulton(Func<double, double, double> f, double a, double b, double h, double u0)

{

var x = GetNodes(a, b, h).ToArray();

double[] u = new double[x.Length];

double[] initialU = SolveRungeKutta(f, a, a + 2 \* h, h, u0).ToArray();

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

{

u[i] = initialU[i];

}

for (int i = 3; i < x.Length; i++)

{

double xi = x[i - 1];

double uiPrev = u[i - 1];

// Predictor step using Adams-Bashforth 3rd order

double predictor = uiPrev + (h / 12) \* (23 \* f(xi, uiPrev) - 16 \* f(xi - h, u[i - 1]) + 5 \* f(xi - 2 \* h, u[i - 2]));

// Corrector step using Adams-Moulton 3rd order

double corrector = uiPrev + (h / 12) \* (5 \* f(xi + h, predictor) + 8 \* f(xi, uiPrev) - f(xi - h, u[i - 1]));

u[i] = corrector;

}

return u;

}

public static IEnumerable<double> GetNodes(double a, double b, double h)

{

var nodes = new List<double>();

for (var node = a; node <= b; node += h)

{

nodes.Add(node);

}

return nodes;

}

public static double ErrorEstimateByRunge(int order, double[] y1, double[] y2)

{

var max = 0.0;

var j = 0;

for (var i = 0; i < y2.Length; i++, j += 2)

{

max = Math.Max(Math.Abs(y1[j] - y2[i]), max);

}

return max / (Math.Pow(2, order) - 1);

}

public static double ErrorEstimateByAbsoluteDifference(double[] exactValues, double[] methodValues)

{

var max = 0.0;

for (var i = 0; i < exactValues.Length; i++)

{

max = Math.Max(Math.Abs(exactValues[i] - methodValues[i]), max);

}

return max;

}

}

}

Main.cs

private void Main()

{

ReportPrinting.PrintTableTitle(10);

var x = Methods.GetNodes(A, B, 0.1).ToArray();

var u = x.Select(x => \_exactSolution(x)).ToArray();

var y1\_1 = Methods.SolveTrapezoidal(\_f, \_dfdu, A, B, 0.1, 0, EPSILON).ToArray();

var y2\_1 = Methods.SolveRungeKutta(\_f, A, B, 0.1, 0).ToArray();

var y3\_1 = Methods.SolveAdamsBashforthMoulton(\_f, A, B, 0.1, 0).ToArray();

var y1\_2 = Methods.SolveTrapezoidal(\_f, \_dfdu, A, B, 0.2, 0, EPSILON).ToArray();

var y2\_2 = Methods.SolveRungeKutta(\_f, A, B, 0.2, 0).ToArray();

var y3\_2 = Methods.SolveAdamsBashforthMoulton(\_f, A, B, 0.2, 0).ToArray();

for (var i = 0; i < x.Length; i++)

{

ReportPrinting.PrintTableRow(i, x[i], u[i], y1\_1[i], y2\_1[i], y3\_1[i], 10, 6);

}

var maxError\_1 = Methods.ErrorEstimateByAbsoluteDifference(u, y1\_1);

var maxError\_2 = Methods.ErrorEstimateByAbsoluteDifference(u, y2\_1);

var maxError\_3 = Methods.ErrorEstimateByAbsoluteDifference(u, y3\_1);

var rungeError\_1 = Methods.ErrorEstimateByRunge(2, y1\_1, y1\_2);

var rungeError\_2 = Methods.ErrorEstimateByRunge(2, y2\_1, y2\_2);

var rungeError\_3 = Methods.ErrorEstimateByRunge(2, y3\_1, y3\_2);

ReportPrinting.PrintErrorRow("max|u[i] - y[i]|", maxError\_1, maxError\_2, maxError\_3, 10, 6);

ReportPrinting.PrintErrorRow("runge", rungeError\_1, rungeError\_2, rungeError\_3, 10, 6);

DrawLine(x, u, "tg(x^3 / 3)", ScottPlot.Color.FromColor(Color.Black));

DrawLine(x, y1\_1, "Трапеция", ScottPlot.Color.FromColor(Color.Red));

DrawLine(x, y2\_1, "Рунги-Кутты", ScottPlot.Color.FromColor(Color.Green));

DrawLine(x, y3\_1, "Адамс", ScottPlot.Color.FromColor(Color.Blue));

}

Результаты:

------------------------------------------------------------------------------

| i | x\_i | точное | метод 1 | метод 2 | метод 3 |

------------------------------------------------------------------------------

| 0 | 0.00 | 0.000000e+000 | 0.000000e+000 | 0.000000e+000 | 0.000000e+000 |

------------------------------------------------------------------------------

| i | x\_i | точное | метод 1 | метод 2 | метод 3 |

------------------------------------------------------------------------------

| 0 | 0.00 | 0.000000e+000 | 0.000000e+000 | 0.000000e+000 | 0.000000e+000 |

------------------------------------------------------------------------------

| 1 | 0.10 | 3.333333e-004 | 5.000001e-004 | 3.333333e-004 | 3.333333e-004 |

------------------------------------------------------------------------------

| 2 | 0.20 | 2.666673e-003 | 3.000018e-003 | 2.666672e-003 | 2.666672e-003 |

------------------------------------------------------------------------------

| 3 | 0.30 | 9.000243e-003 | 9.500442e-003 | 9.000223e-003 | 9.000327e-003 |

------------------------------------------------------------------------------

| 4 | 0.40 | 2.133657e-002 | 2.200472e-002 | 2.133645e-002 | 2.133715e-002 |

------------------------------------------------------------------------------

| 5 | 0.50 | 4.169080e-002 | 4.253120e-002 | 4.169034e-002 | 4.169310e-002 |

------------------------------------------------------------------------------

| 6 | 0.60 | 7.212467e-002 | 7.315008e-002 | 7.212327e-002 | 7.213101e-002 |

------------------------------------------------------------------------------

| 7 | 0.70 | 1.148341e-001 | 1.160763e-001 | 1.148305e-001 | 1.148468e-001 |

------------------------------------------------------------------------------

| 8 | 0.80 | 1.723432e-001 | 1.738726e-001 | 1.723345e-001 | 1.723589e-001 |

------------------------------------------------------------------------------

| 9 | 0.90 | 2.478987e-001 | 2.498638e-001 | 2.478789e-001 | 2.478907e-001 |

------------------------------------------------------------------------------

| 10 | 1.00 | 3.462535e-001 | 3.489641e-001 | 3.462087e-001 | 3.461203e-001 |

------------------------------------------------------------------------------

| max|u[i] - y[i]| | 2.710574e-003 | 4.489061e-005 | 1.332993e-004 |

------------------------------------------------------------------------------

| runge | 2.687174e-003 | 9.326121e-005 | 3.899830e-004 |

------------------------------------------------------------------------------

