МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

УТВЕРЖДАЮ
Зав.кафедрой,
доцент, к. ф.-м. н.
_____ М. В. Огнева

ОТЧЕТ О ПРАКТИКЕ

студента 3 курса 341 группы факультета КНиИТ
Шарова Кирилла Владимировича

вид практики: производственная
кафедра: информатики и программирования
курс: 3
семестр: 6
продолжительность: 4 нед., с 22.06.2024 г. по 18.07.2024 г.

Руководитель практики от университета,

доцент, к. ф.-м. н. _____ ? ?.?.

Руководитель практики от организации (учреждения, предприятия),

_____ ?. ?. ?

Тема практики:«???»

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью настоящей работы является создание примера оформления студенческой работы средствами системы LaTeX.

Поставлена задача оформить документ в соответствии:

— со стандартом СТО 1.04.01-2012 Порядком выполнения, структурой и правилами оформления курсовых работ (проектов) и выпускных квалификационных работ, принятых в Саратовском государственном университете в 2012 году;

— с правилами оформления титульного листа отчета о прохождении практики в соответствии со стандартом СТО 1.01-2005.

Изложенный ниже текст не имеет особого смысла и приведен только для демонстрации оформления своих элементов.

1 Сумма функционального ряда

Вычислить сумму ряда следующего функционального ряда:

$$f(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} - ... = \sum_{n=0}^{\infty}(-1)^k \frac{x^{2k+1}}{(2k+1)!}$$

Условие остановки вычислений: $|f_{n+1}(x)| < \epsilon = 10^{-3}$

1.0.1 Код

```
void f(double x, double e)
{
    double S = x;
    double Fn =S;
    int i = 1;
    while (!abs(Fn) < e)
    {
        Fn *= ((-1) * x * x) / (2 * i * (2 * i + 1));
        S += Fn;
        i++;
    }

    std::cout << x << '\t' << S << '\t' << i <<'\n';
}
```

1.0.2 Вывод

Формат вывода:

x_0	x_1	...	x_m
$S_{n_0}(x_0)$	$S_{n_1}(x_1)$	...	$S_{n_m}(x_m)$
n_0	n_1	...	n_m

Полученный вывод:

-2	-1	0	1	2
-0.909297	-0.841471	0	0.841471	0.909297
103	90	1	90	103

2 Интерполяция функций одного аргумента

2.1 Интерполяционный многочлен в общем виде

Пусть известны значения функции f(x) в некотором наборе узловых точек:

x	-2	-1	0	1
f(x)	-8	-1	0	1

Тогда интерполяционный многочлен функции f(x) будет иметь вид:

$$P_3(x) = a_3x^3 + a_2x^2 + a_1x + a_0$$

Построить расширенную матрицу коэффициентов (РМК) относительно коэффициентов $a_i, i = 0...3$.

2.1.1 Код

```
struct MatrixOfCoefficients
{
    MatrixOfCoefficients(std::vector<double> Xi,
    std::vector<double> Fi)
    {
        for (size_t i = 0; i < Xi.size(); i++)
        {
            matrix.push_back(std::vector<double>());
            for (size_t j = 0; j < Fi.size(); ++j)
            {
                matrix[i].push_back(std::pow(Xi[i],
                Xi.size() - j - 1));
            }
            matrix[i].push_back(Fi[i]);
        }
    }

    void print()
    {
        std::cout << "Matrix of Coefficient:\n";
        const size_t width = 10;
        std::cout << std::fixed << std::setprecision(3);
        for (size_t i = 0; i < matrix.size(); i++,
        std::cout<<'\n')
            for (size_t j = 0; j < matrix[i].size(); ++j)
```

```
                std::cout << std::setw(width)
                << matrix[i][j];
    }

private:
    std::vector<std::vector<double>> matrix;
};
```

2.1.2 Вывод

Полученная таблица:

-8.000	4.000	-2.000	1.000	8.000
-1.000	1.000	-1.000	1.000	-1.000
0.000	0.000	0.000	1.000	0.000
1.000	1.000	1.000	1.000	1.000

2.2 Интерполяционный многочлен в форме Лагранжа

Построить интерполяционный многочлен в форме Лагранжа для даных из предыдущего задания.

2.2.1 Код

```
struct LagrangeTable
{
    LagrangeTable(std::vector<double> Xi,
    std::vector<double> Fi)
    {
        this->Fi = Fi;
        this->Xi = Xi;
    }

    void Print()
    {
        std::cout << "Lagrange Polynom function's table:\n";
        const size_t width = 10;
        std::cout << std::fixed << std::setprecision(3)
        << "Xi: ";
        for (size_t i = 0; i < Xi.size(); i++)
        {
            std::cout << std::setw(width) << Xi[i] << ' ';
```

```
            if (i != Xi.size() - 1)
            {
                std::cout << std::setw(width)
                << (Xi[i] + Xi[i+1])/2. << ' ';
            }
        }
        std::cout << "\nFi: ";
        for (size_t i = 0; i < Fi.size(); i++)
        {
            std::cout << std::setw(width) << Fi[i] << ' ';
            if (i != Fi.size() - 1)
            {
                std::cout << std::setw(width) <<
                LagrangePolynom((Xi[i] + Xi[i + 1]) / 2.,
                Xi, Fi) << ' ';
            }
        }
        std::cout << "\n";
    }

private:
    std::vector<double> Xi;
    std::vector<double> Fi;
};

double LagrangePolynom(double x, std::vector<double> Xi,
std::vector<double> Fi)
{
    double L = 0;
    double currentSum;
    for (int i = 0; i < Xi.size(); i++)
    {
        currentSum = Fi[i];
        for (int j = 0; j < Xi.size(); j++)
        {
            if (j != i)
            {
                currentSum *= (x - Xi[j]) / (Xi[i]-Xi[j]);
            }
        }
        L += currentSum;
```

```
    }
    return L;
}
```

2.2.2 Вывод

Полученный вывод:

x	-2	-1.5	-1	-0.5	0	0.5	1
f(x)	-8	-3.375	-1	-0.125	0	0.125	1

2.3 Интерполяционный многочлен в форме Ньютона

Построить интерполяционный многочлен в форме Ньютона по данным из предыдущего задания.

2.3.1 Код

```
struct NewtonTable
{
    NewtonTable(std::vector<double> Fi,
    std::vector<double> knots)
    {
        Knots = knots;
        for (size_t i = 0; i < Fi.size(); ++i)
        {
            SplitDifs.push_back(std::vector<double>());
        }
        for (size_t i = 0; i < Fi.size(); ++i)
        {
            SplitDifs[i].push_back(Fi[i]);
        }
        for (size_t i = 1; i < Fi.size(); i++)
        {
            for (size_t j = 0; j < Fi.size() - i; j++)
            {
                SplitDifs[j].push_back(
                (SplitDifs[j+1][i-1] - SplitDifs[j][i-1])/
                (knots[j+i] - knots[j]));
            }
        }
    }
```

```
    void print()
    {
        const size_t width = 10;
        std::cout << std::fixed << std::setprecision(3);
        for (size_t i=0; i<Knots.size(); i++,
        std::cout<<'\n')
            for (size_t j = 0; j < Knots.size() - i; j++)
            {
                std::cout << std::setw(width) <<
                SplitDifs[i][j]<<' ';
            }
    }

    std::vector<std::vector<double>> GetSplitDifs()
    { return SplitDifs; }

private:
    std::vector<std::vector<double>> SplitDifs;
    std::vector<double> Knots;
};


double NewtonPolynom(double x, std::vector<double> Xi,
std::vector<std::vector<double>> SplitDifs)
{
    double ans = 0;
    double temp;
    for (size_t i = 0; i < SplitDifs[0].size(); i++)
    {
        temp = SplitDifs[0][i];
        for (size_t j = 0; j < i; j++)
        {
            temp *= (x - Xi[j]);
        }
        ans += temp;
    }
    return ans;
}
```

2.3.2 Вывод

Таблица разделённых разностей:

-8.000	7.000	-3.000	1.000
-1.000	1.000	0.000	
0.000	1.000		
1.000			

Таблица значений интерполяционного многочлена в форме Ньютона:

x	-2	-1.5	-1	-0.5	0	0.5	1
f(x)	-8	-3.375	-1	-0.125	0	0.125	1

2.4 Интерполяция с помощью кубических сплайнов

Построить кусочно-непрерывную склейку кубических сплайнов для функции f(x), заданной в предыдущих заданиях.

Вспомогательная система:

$$\begin{cases} a_1 = f_0 \\ a_2 = f_1 \\ a_3 = f_2 \\ a_1 + b_1 h + c_1 h^2 + d_1 h^3 = f_1 \\ a_2 + b_2 h + c_2 h^2 + d_2 h^3 = f_2 \\ a_3 + b_3 h + c_3 h^2 + d_3 h^3 = f_3 \\ b_1 + 2c_1 h + 3d_1 h^2 = b_2 \\ b_2 + 2c_2 h + 3d_2 h^2 = b_3 \\ 2c_1 + 6d_1 h = 2c_2 \\ 2c_2 + 6d_2 h = 2c_3 \\ 2c_3 + 6d_3 h = 0 \\ 2c_1 = 0 \end{cases}$$

2.4.1 Код

```
struct GaussTable
{
```

```
public:
    GaussTable(double h, std::vector<double> Fi)
    {
    table = {{1,0,0,0,0,0,0,0,0,0,0,0, Fi[0]},
             {0,1,0,0,0,0,0,0,0,0,0,0, Fi[1]},
             {0,0,1,0,0,0,0,0,0,0,0,0, Fi[2]},
             {1,0,0,h,0,0,h * h,0,0,h * h * h,0,0,Fi[1]},
             {0,1,0,0,h,0,0,h * h,0,0,h * h * h,0,0,Fi[2]},
             {0,0,1,0,0,h,0,0,h * h,0,0,h * h * h,Fi[3]},
             {0,0,0,1,-1,0,2*h,0,0,3*h*h,0,0,0},
             {0,0,0,0,1,-1,0,2*h,0,0,3*h*h,0,0},
             {0,0,0,0,0,0,2,-2,0,6*h,0,0,0},
             {0,0,0,0,0,0,0,2,-2,0,6*h,0,0},
             {0,0,0,0,0,0,2,0,0,0,0,0,0},
             {0,0,0,0,0,0,0,0,2,0,0,6*h,0}
        };
    }
    void Print()
    {
        const size_t width = 8;
        std::cout << std::fixed << std::setprecision(3);
        for (size_t i = 0; i < table.size(); i++,
        std::cout << '\n')
            for (size_t j = 0; j < table[0].size(); j++)
            {
                if (j == table[0].size()-1)
                    std::cout << std::setw(width) << '|';
                std::cout << std::setw(width) << table[i][j]
                << ' ';
            }
    }

private:
    std::vector<std::vector<double>> table;
};
```

2.4.2 Вывод

При h = 1

1	0	0	0	0	0	0	0	0	0	0	0	-8
0	1	0	0	0	0	0	0	0	0	0	0	-1
0	0	1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0	0	1	0	0	-1
0	1	0	0	1	0	0	1	0	0	1	0	0
0	0	1	0	0	1	0	0	1	0	0	1	1
0	0	0	1	-1	0	2	0	0	3	0	0	0
0	0	0	0	1	-1	0	2	0	0	3	0	0
0	0	0	0	0	0	2	-2	0	6	0	0	0
0	0	0	0	0	0	0	2	-2	0	6	0	0
0	0	0	0	0	0 0	2	0	0	0	0	0	0
0	0	0	0	0	0	0	0	2	0	0	6	0

3 Численные методы решения СЛАУ

3.1 Метод Гаусса решения СЛАУ

Здесь и далее в работе v = 5.

Имеем:

$$A = \begin{pmatrix} 5 & 0.05 & 0.05 & 0.05 & 0.05 \\ 0.06 & 6 & 0.06 & 0.06 & 0.06 \\ 0.07 & 0.07 & 7 & 0.07 & 0.07 \\ 0.08 & 0.08 & 0.08 & 8 & 0.08 \\ 0.09 & 0.09 & 0.09 & 0.09 & 9 \end{pmatrix}$$

$$B = A * \begin{pmatrix} 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{pmatrix}$$

Необходимо вывести:

1. РМК (A|B)
2. Результирующую матрицу прямого хода метода Гаусса
3. Результирующий вектор $(x_1, x_2, ..., x_n)$ обратного хода метода Гаусса

3.1.1 Код

```
void Matrix::printGaussAB(const Matrix& A, const Matrix& B)
{
    const size_t width = 10;
    std::cout << std::fixed << std::setprecision(3);
    for (size_t i = 0; i < A.getN(); i++)
    {
        for (size_t j = 0; j < A.getM(); j++)
        {
            std::cout << std::setw(width)
            << A.mat[i][j] << ' ';
        }
        std::cout << std::setw(width) << "| "
        << B.mat[i][0] << '\n';
    }
}
```

```
Matrix Matrix::merge(const Matrix& A, const Matrix& B)
{
    Matrix ans;
    ans.n = A.n;
    ans.m = A.m + B.m;
    for (int i = 0; i < A.n; i++)
    {
        ans.mat.push_back(std::vector<double>(ans.m));
    }
    for (int i = 0; i < A.n; i++)
    {
        for (int j = 0; j < A.m; j++)
        {
            ans.mat[i][j] = A.mat[i][j];
        }
        for (int j = 0; j < B.m; j++)
        {
            ans.mat[i][A.m + j] = B.mat[i][j];
        }
    }

    return ans;
}

Matrix Matrix::forward_gauss(const Matrix& A,
const Matrix& B)
{
    Matrix ans = merge(A, B);
    for (int i = 0; i < ans.n; i++)
    {
        double koeff = 1, eps = 1e-6;
        if (std::abs(ans.mat[i][i]) > eps)
            koeff = ans.mat[i][i];
        else
        {
            bool isChanged = false;
            double maxEl = 0;
            int index = -1;
            for (int j = i + 1; j < ans.n; j++)
            {
```

```
            if (std::abs(std::abs(ans.mat[j][i]) - maxEl)
            > eps)
            {
                maxEl = ans.mat[j][i];
                index = j;
            }
        }
        if (index != -1)
        {
            std::swap(ans.mat[i], ans.mat[index]);
            isChanged = true;
            koeff = ans.mat[i][i];
        }
        if (!isChanged)
        {
            maxEl = 0;
            index = -1;
            for (int j = 0; j < ans.m; j++)
            {
                if (std::abs(std::abs(ans.mat[i][j])
                - maxEl) > eps)
                {
                    maxEl = ans.mat[i][j];
                    index = j;
                }
            }
            if (index != -1)
            {
                for (int j = 0; j < ans.n; j++)
                    std::swap(ans.mat[j][i],
                    ans.mat[j][index]);
                isChanged = true;
                koeff = ans.mat[i][i];
            }
        }
        if (!isChanged)
            continue;
    }
    for (int j = i; j < ans.m; j++)
    {
        ans.mat[i][j] = ans.mat[i][j] / koeff;
```

```
        }
        for (int j = i + 1; j < ans.n; j++)
        {

            for (int k = ans.m-1; k >= i; k--)
            {
                double diff = ans.mat[i][k] * ans.mat[j][i];
                ans.mat[j][k] -= diff;
            }
        }
    }


    return ans;
}

std::vector<double> Matrix::back_gauss(const Matrix& A)
{
    std::vector<double> Xn(A.getN());
    for (int i = A.getN() - 1; i>=0; i--)
    {
        Xn[i] = A.mat[i][A.getM()-1];
        for (int j = A.getM()-2; j > i; j--)
        {
            Xn[i] -= Xn[j] * A.mat[i][j];
        }
    }
    return Xn;
}
```

3.1.2 Вывод

1)

5.000	0.050	0.050	0.050	0.050	26.500
0.060	6.000	0.060	0.060	0.060	37.740
0.070	0.070	7.000	0.070	0.070	50.960
0.080	0.080	0.080	8.000	0.080	66.160
0.090	0.090	0.090	0.090	9.000	83.340

2)

3)

1.000	0.010	0.010	0.010	0.010	5.300
0.000	1.000	0.010	0.010	0.010	6.238
0.000	0.000	1.000	0.010	0.010	7.167
0.000	0.000	0.000	1.000	0.010	8.087
0.000	0.000	0.000	0.000	1.000	9.000

5.000	6.000	7.000	8.000	9.000

3.2 Приложения метода Гаусса

Используя метод Гаусса вычислить определитель и обратную матрицу для матрицы

$$A = \begin{pmatrix} -9 & 1 & 3 & 4 \\ 3 & 0 & -1 & 4 \\ -5 & 2 & 3 & 0 \\ 4 & -1 & 2 & 6 \end{pmatrix}$$

3.2.1 Код

```
double Matrix::determinant(const Matrix& A)
{
    Matrix copy = A;
    int changes_count = 0;
    double determinant_koeff = 1;
    for (int i = 0; i < copy.n; i++)
    {
        double koeff = 1, eps = 1e-6;
        if (std::abs(copy.mat[i][i]) > eps)
            koeff = copy.mat[i][i];
        else
        {
            bool isChanged = false;
            double maxEl = 0;
            int index = -1;
            for (int j = i + 1; j < copy.n; j++)
            {
                if (std::abs(std::abs(copy.mat[j][i]) -
                maxEl) > eps)
                {
                    maxEl = copy.mat[j][i];
                    index = j;
                }
```

```
        }
        if (index != -1)
        {
            std::swap(copy.mat[i], copy.mat[index]);
            isChanged = true;
            koeff = copy.mat[i][i];
        }
        if (!isChanged)
        {
            maxEl = 0;
            index = -1;
            for (int j = 0; j < copy.m; j++)
            {
                if (std::abs(std::abs(copy.mat[i][j]) -
                maxEl) > eps)
                {
                    maxEl = copy.mat[i][j];
                    index = j;
                }
            }
            if (index != -1)
            {
                for (int j = 0; j < copy.n; j++)
                    std::swap(copy.mat[j][i],
                    copy.mat[j][index]);
                isChanged = true;
                koeff = copy.mat[i][i];
            }
        }
        if (isChanged)
            changes_count += 1;
        if (!isChanged)
            continue;
    }
    for (int j = i; j < copy.m; j++)
    {
        copy.mat[i][j] = copy.mat[i][j] / koeff;
    }
    determinant_koeff *= koeff;
    for (int j = i + 1; j < copy.n; j++)
    {
```

```
            for (int k = copy.m - 1; k >= i; k--)
            {
                double diff = copy.mat[i][k] * copy.mat[j][i];
                copy.mat[j][k] -= diff;
            }
        }
    }
    return std::pow(1, changes_count) * determinant_koeff;
}

Matrix Matrix::reverse_matrix(const Matrix& A)
{
    Matrix B = A;
    Matrix ans(A.n, A.m);
    std::vector<std::vector<double>> collumn;
    for (size_t i = 0; i < A.n; i++)
        collumn.push_back(std::vector<double>(1));
    for (size_t i = 0; i < A.m; i++)
    {
        collumn[i][0] = 1;
        Matrix temp = forward_gauss(B, collumn);
        std::vector<double> current_collumn =
        back_gauss(temp);
        for (size_t j = 0; j < ans.n; j++)
        {
            ans.mat[j][i] = current_collumn[j];
        }
        collumn[i][0] = 0;
    }

    return ans;
}
```

3.2.2 Вывод

$\Delta A = -272$

$$A^{-1} = \begin{pmatrix} -0.059 & 0.169 & 0.066 & 0.074 \\ -0.412 & 0.059 & 0.588 & -0.235 \\ 0.176 & 0.243 & 0.051 & 0.279 \\ 0.088 & 0.184 & -0.037 & 0.015 \end{pmatrix}$$

3.3 Метод прогонки решения СЛАУ с трёхдиагональной матрицей коэффициентов

При помощи метода прогонки решить СЛАУ вида $A * x = B$, где

$$A = \begin{pmatrix} -5 & 0.05 & 0 & 0 & 0 \\ 0.06 & -6 & 0.06 & 0 & 0 \\ 0 & 0.07 & -7 & 0.07 & 0 \\ 0 & 0 & 0.08 & -8 & 0.08 \\ 0 & 0 & 0 & 0.09 & -9 \end{pmatrix}$$

$$B = A * \begin{pmatrix} 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{pmatrix}$$

Необходимо вывести:

1. РМК (A|B)
2. Результирующую матрицу прямого хода метода прогонки (коэффициенты P и Q)
3. Результирующий вектор $(x_1, x_2, ..., x_n)$ обратного хода метода прогонки

3.3.1 Код

```
std::vector<std::vector<double>>
Matrix::three_diagonal_matrix(const Matrix& A,
const std::vector<double>& d)
{
    std::vector<double> P(A.getN()- 1);
    std::vector<double> Q(A.getN());
    P[0] = -A[0][1] / A[0][0]; Q[0] = d[0]/A[0][0];
    for (int i = 1; i < A.getN() - 1; i++)
    {
        double det = -A[i][i] - A[i][i - 1] * P[i - 1];
        P[i] = A[i][i + 1] / det;
        Q[i] = (A[i][i - 1] * Q[i - 1] - d[i]) / det;

    }
    size_t size = A.getN();
```

```
    Q[A.getN() - 1] = (A[size - 1][size - 2] *
    Q[size - 2] - d[size - 1]) / (-A[size - 1][size - 1] -
    A[size - 1][size - 2] * P[size - 2]);

    size = Q.size();
    std::vector<double> Xn(size);
    Xn[size - 1] = Q[size - 1];
    for (int i = size-1; i > 0; --i)
    {
        Xn[i - 1] = P[i - 1] * Xn[i] + Q[i - 1];
    }

    return std::vector<std::vector<double>>{P, Q, Xn};
}
```

3.3.2 Вывод

1)

-5.000	0.050	0.000	0.000	0.000	-24.700
0.060	-6.000	0.060	0.000	0.000	-35.280
0.000	0.070	-7.000	0.070	0.000	-48.020
0.000	0.000	0.080	-8.000	0.080	-62.720
0.000	0.000	0.000	0.090	-9.000	-80.280

2)

P	0.01	0.010001	0.010001	0.010001	
Q	4.94	5.92999	6.91999	7.90999	9

3)

5	6	7	8	9

4 Численные методы решения дифференциальных уравнений

4.1 Метод Эйлера решения задачи Коши для обыкновенного дифференциального уравнения (ОДУ) 1-го порядка

Решить следующую задачу Коши методом Эйлера, усовершенствованным методом Эйлера, методом предварительного и корректирующего счёта:

$$\begin{cases} y'(x) = 2 * 5 * x + v * x^2 - y(x) \\ y(1) = 5 \end{cases}$$

,

где $y_{correct}(x) = 5 * x^2$

Необходимо вывести:

1. Таблицу результатов для метода Эйлера
2. Таблицу результатов для усовершенствованного метода Эйлера
3. Таблицу результатов для метода предварительного и корректирующего счёта

Общий вид таблицы результатов (здесь и далее в работе):

где $y_{met}(x_i)$ - значение метода в точке x_i, $y_{correct}(x_i)$ - истинное значение функции y(x) в точке x_i, e_i - погрешность метода в точке x_i.

4.1.1 Код

```
std::vector<double> eiler_method(std::vector<double> Xk,
double Y0, double h, std::function<double(double, double)> f)
{
    std::vector<double> Yk;
    Yk.push_back(Y0);
    for (size_t i = 1; i < Xk.size(); i++)
    {
        Yk.push_back(Yk[i - 1] + h * f(Xk[i - 1],
        Yk[i - 1]));
    }

    return Yk;
}

std::vector<double> modernize_eiler_method
(std::vector<double> Xk, double Y0, double h,
std::function<double(double, double)> f)
```

```
{
    std::vector<double> Yk;
    Yk.push_back(Y0);

    for (size_t i = 1; i < Xk.size(); i++)
    {
        Yk.push_back(Yk[i - 1] + h * f(Xk[i-1] + h/2,
        Yk[i-1] + (h/2)* f(Xk[i-1], Yk[i-1])));
    }

    return Yk;
}


std::vector<double> correct_count_method
(std::vector<double> Xk, double Y0, double h,
std::function<double(double, double)> f)
{
    std::vector<double> Yk;
    Yk.push_back(Y0);

    for (size_t i = 1; i < Xk.size(); i++)
    {
        Yk.push_back(Yk[i - 1] + (h/2) * (f(Xk[i - 1],
        Yk[i - 1]) + f(Xk[i],
        Yk[i-1] + h * f(Xk[i-1], Yk[i-1]))));
    }

    return Yk;
}


void print_eiler_method(std::vector<double> Xk,
std::vector<double> Yk, std::vector<double> Y)
{
    const size_t width = 10;
    std::cout << std::fixed << std::setprecision(3);
    for (size_t i = 0; i < Xk.size(); i++)
    {
        std::cout << std::setw(width) << Xk[i];
    }
```

```
    std::cout << "\n";
    for (size_t i = 0; i < Xk.size(); i++)
    {
        std::cout << std::setw(width) << Yk[i];
    }
    std::cout << "\n";
    for (size_t i = 0; i < Xk.size(); i++)
    {
        std::cout << std::setw(width) << Y[i];
    }
    std::cout << "\n";
    for (size_t i = 0; i < Yk.size(); i++)
    {
        std::cout << std::setw(width) << Yk[i] - Y[i];
    }
}
```

4.1.2 Вывод

1)

1.000	1.100	1.200	1.300	1.400	1.500	1.600	1.700
5.000	6.000	7.105	8.315	9.628	11.045	12.566	14.189
5.000	6.050	7.200	8.450	9.800	11.250	12.800	14.450
0.000	-0.050	-0.095	-0.136	-0.172	-0.205	-0.234	-0.261

2)

1.000	1.100	1.200	1.300	1.400	1.500	1.600	1.700
5.000	6.051	7.202	8.453	9.804	11.255	12.806	14.457
5.000	6.050	7.200	8.450	9.800	11.250	12.800	14.450
0.000	0.001	0.002	0.003	0.004	0.005	0.006	0.007

3)

1.000	1.100	1.200	1.300	1.400	1.500	1.600	1.700
5.000	6.053	7.205	8.457	9.809	11.260	12.812	14.463
5.000	6.050	7.200	8.450	9.800	11.250	12.800	14.450
0.000	0.002	0.005	0.007	0.009	0.010	0.012	0.013

4.2 Разностный метод решения краевой задачи для ОДУ 2-го порядка

4.2.1 Код

```
std::vector<double> differences_method(const int n,
const double h, const std::function<double(double)>& p,
const std::function<double(double)>& q,
    const std::function<double(double)>& f)
{
    std::vector<std::vector<double>> mat;
    std::vector<double> xk, fk;
    xk.push_back(0);
    for (size_t i = 1; i <= n; ++i)
    {
        xk.push_back(xk[i - 1] + h);
    }
    for (size_t i = 0; i < xk.size(); ++i)
    {
        fk.push_back(f(xk[i]));
    }
    int y0 = 0, yn = 0;
    for (size_t i = 0; i < n - 2; ++i)
        mat.push_back(std::vector<double>(n - 2));
    for (size_t i = 0; i < n - 2; ++i)
    {
        mat[i][i] = -2. / (h * h) + q(xk[i]);
        if (i != 0)
        {
            mat[i][i - 1] = 1. / (h * h) - p(xk[i]) /(2 * h);
        }
        if (i != n - 3)
        {
            mat[i][i + 1] = 1. / (h * h) + p(xk[i]) /(2 * h);
        }
    }

    Matrix a = Matrix(mat);
    auto yk = Matrix::three_diagonal_matrix(a, fk)[2];
    yk.push_back(0.);
    yk.insert(yk.begin(), 0);
    return yk;
}
```

```
void print_differences_method(std::vector<double> Xk,
std::vector<double> Yk, std::vector<double> Y)
{
    const size_t width = 20;
    std::cout << std::fixed << std::setprecision(6);
    for (size_t i = 0; i < Xk.size(); i++)
    {
        std::cout << std::setw(width) << Xk[i];
    }
    std::cout << "\n";
    for (size_t i = 0; i < Yk.size(); i++)
    {
        std::cout << std::setw(width) << Yk[i];
    }
    std::cout << "\n";
    for (size_t i = 0; i < Y.size(); i++)
    {
        std::cout << std::setw(width) << Y[i];
    }
    std::cout << "\n";
    for (size_t i = 0; i < Yk.size(); i++)
    {
        std::cout << std::setw(width) << Yk[i] - Y[i];
    }
}
```

4.2.2 Вывод

0	0.625	1.250	1.875	2.5	3.125	3.750000	4.375	5
0	541.605	1063.679	1223.736	838.756	484.769	312.843	135.178	0
0	-8.544	-29.296	-54.931	-78.125	-91.552	-87.890	-59.814	0
0	550.150	1092.976	1278.668	916.881	576.322	400.734	194.993	0

4.3 Метод неопределённых коэффициентов решения краевой задачи для ОДУ 2-го порядка

Решить методом неопределённых коэффициентов краевую задачу из прошлого задания.

4.3.1 Код

```
std::vector<double> unknown_koeff_method(const int n,
std::vector<double> xk, std::function<double(double)> f,
std::function<double(double)> p, std::function<double(double)> q,
std::function<double(double, int)> phi_k,
std::function<double(double, int)> phi_k_derived,
std::function<double(double, int)> phi_k_derived2)
{
    std::vector<std::vector<double>> mat_ak;
    for (size_t i = 0; i < n; i++)
        mat_ak.push_back(std::vector<double>(n));


    for (size_t j = 0; j < n; j++)
    {
        for (size_t k = 0; k < n; k++)
        {
            mat_ak[j][k] = phi_k_derived2(xk[j], k) +
            p(xk[j]) * phi_k_derived(xk[j], k) +
            q(xk[j]) * phi_k(xk[j], k);
        }
    }
    Matrix A(mat_ak);


    std::vector<std::vector<double>> fk;
    for (size_t i = 0; i < n; i++)
    {
        fk.push_back(std::vector<double>(1));
    }
    for (size_t i = 0; i < n; i++)
        fk[i][0] = f(xk[i]);
    Matrix B(fk);

    auto temp = Matrix::forward_gauss(A, B);
    auto ak = Matrix::back_gauss(temp);

    return ak;
}

//in main function
auto y_new = [&ak, &phi_k](double x)
```

```
{
        double ans = 0;
        for (size_t i = 0; i < ak.size(); ++i)
            ans += ak[i] * phi_k(x, i);
        return ans;
};
```

4.3.2 Вывод

0.625	1.250	1.875	2.500	3.125	3.750	4.375	5.000
-704.808	-1134.035	-1009.307	-747.954	-571.162	-413.318	-250.851	0.000
-8.545	-29.297	-54.932	-78.125	-91.553	-87.891	-59.814	0.000
-696.263	-1104.738	-954.375	-669.829	-479.609	-325.427	-191.037	0.000

Примечание. При увеличении количества внутренних узловых точек погрешность уменьшается (при n=100 погрешность составляет порядка 2.2801)

5 Численные методы решения интегральных уравнений

5.1 Решение интегрального уравнения Фредгольма в случае вырожденного ядра

Решить следующее интегральное уравнение с вырожденным ядром:

$$y(x) + 1 * \int_0^1 (x * 5 + x^2 t^2 + x^3 t^3) * y(t) dt = 5 * (\frac{4}{3}x + \frac{1}{4}x^2 + \frac{1}{5}x^3)$$

Где $y_{correct}(x) = 5x$

5.1.1 Код

```
std::vector<double> integral_equation()
{

    std::vector<std::vector<double>> A_mat = {
        {1 + 1. / 3, 1. / 4, 1. / 5},
        {1. / 4, 1 + 1. / 5, 1. / 6},
        {1. / 5, 1. / 6, 1 + 1. / 7}
    };
    Matrix A(A_mat);

    std::vector<std::vector<double>> phi_mat = {
        {20. / 9 + 5. / 16 + 1. / 5},
        {20. / 12, 5. / 20, 1. / 6},
        {20. / 15, 5. / 24, 1. / 7}
    };
    Matrix phi(phi_mat);

    auto tmp = Matrix::forward_gauss(A, phi);
    auto q = Matrix::back_gauss(tmp);


    return q;
}


\\in main function
auto q = integral_equation();
auto y_method = [&f, lambda, &q, &ak, n](double x)
```

```
{
    double sum = 0;
    for (size_t i = 0; i < n; ++i)
        sum += q[i] * ak(x, i + 1);
    sum *= lambda;
    return f(x) - sum;
};
```

5.1.2 Вывод

0.000	0.100	0.200	0.300	0.400	0.500	0.600	0.700	0.800	0.900
0.000	0.493	0.995	1.506	2.029	2.565	3.117	3.684	4.271	4.876
0.000	0.500	1.000	1.500	2.000	2.500	3.000	3.500	4.000	4.500
0.000	-0.007	-0.005	0.006	0.029	0.065	0.117	0.184	0.271	0.376

5.2 Решение интегрального уравнения Фредгольма квадратурным методом

Решить следующее интегральное уравнение квадратурным методом:

$$y(x) + 1 * \int_0^1 (x * 5 + x^2 t^2 + x^3 t^3) * y(t) dt = 5 * (\frac{4}{3}x + \frac{1}{4}x^2 + \frac{1}{5}x^3)$$

Где $y_{correct}(x) = 5x$

5.2.1 Код

```
std::vector<double> integral_equation_quadro(std::vector<double> xk,
std::function<double(double, double)> A,
std::function<double(double)> f,
double h, double lambda)
{
    std::vector<std::vector<double>> A_vec;
    const size_t n = xk.size();
    for (size_t i = 0; i < n - 1; ++i)
    {
        A_vec.push_back(std::vector<double>(n - 1));
    }
```

```
    for (size_t i = 0; i < n-1; ++i)
        for (size_t j = 0; j < n - 1; ++j)
        {
            if (i == j)
            {
                A_vec[i][j] = 1 + lambda * h * A(xk[i], xk[j]);
            }
            else
            {
                A_vec[i][j] = lambda * h * A(xk[i], xk[j]);
            }
        }

    std::vector<std::vector<double>> B_vec(n - 1);
    for (size_t i = 0; i < n - 1; ++i)
    {
        B_vec[i].push_back(f(xk[i]));
    }


    Matrix A_mat(A_vec);
    Matrix B(B_vec);

    auto tmp = Matrix::forward_gauss(A_mat, B);
    return Matrix::back_gauss(tmp);
}
```

5.2.2 Вывод

0	0.07	0.15	0.23	0.30	0.38	0.46	0.53	0.61	0.69	0.76	0.84	0.92
0	0.41	0.83	1.25	1.68	2.13	2.58	3.04	3.51	3.99	4.49	5.00	
0	0.38	0.76	1.15	1.53	1.92	2.30	2.69	3.07	3.46	3.84	4.23	4.61
0	0.02	0.06	0.10	0.15	0.20	0.27	0.34	0.43	0.53	0.64	0.77	

ЗАКЛЮЧЕНИЕ

В настоящей работы приведен пример оформления студенческой работы средствами системы LaTeX.

Показано, как можно оформить документ в соответствии:

— с правилами оформления курсовых и выпускных квалификационных работ, принятых в Саратовском государственном университете в 2012 году;

— с правилами оформления титульного листа отчета о прохождении практики в соответствии со стандартом.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 *Lienhart, R.* An extended set of haar-like features for rapid object detection // Proceedings of the International Conference on Image Processing. — Vol. 1. — Rochester, USA: 2002. — Pp. 900–903.

2 Создание приложения Windows Forms с помощью .NET Framework (C++) [Электронный ресурс]. — URL: http://msdn.microsoft.com/ru-ru/library/vstudio/ms235634(v=vs.100).aspx (Дата обращения 12.07.2013). Загл. с экр. Яз. рус.

ПРИЛОЖЕНИЕ А
Нумеруемые объекты в приложении

Таблица 1 – Results of pass-fail dictionary reduction with the help of masks

Circuit	Number of modelled faults	Number of test vectors in the test set	The volume of pass-fail dictionary, bit	The volume of found mask	The volume of masked dictionary, bit	% of pass-fail dictionary	CPU running time, min
S298	177	322	56994	30	5310	9,32%	0,07
S344	240	127	30480	29	6960	22,83%	0,04
S349	243	134	32562	35	8505	26,12%	0,05
S382	190	2074	394060	28	5320	1,35%	0,43
S386	274	286	78364	65	17810	22,73%	0,26
S400	194	2214	429516	32	6208	1,45%	0,99
S444	191	2240	427840	30	5730	1,34%	0,98
S526	138	2258	311604	28	3864	1,24%	0,61
S641	345	209	72105	58	20010	27,75%	0,24
S713	343	173	59339	58	19894	33,53%	0,19
S820	712	1115	793880	147	104664	13,18%	9,09
S832	719	1137	817503	151	108569	13,28%	9,20
S953	326	14	4564	13	4238	92,86%	0,01
S1423	293	150	43950	58	16994	38,67%	0,15
S1488	1359	1170	1590030	158	214722	13,50%	26,69

$$F(x) = \int_a^b f(x)\,dx. \tag{1}$$

Рисунок 1 – Подпись к рисунку

Таблица 2

0	1
1	0

ПРИЛОЖЕНИЕ Б

Листинг программы

Код приложения task.pl.