Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

ОТЧЕТ ПО ПРАКТИКЕ – МЕТОДЫ ВЫЧИСЛЕНИЙ

РЕФЕРАТ

студента 3 курса 351 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Григорьева Алексея Александровича

Проверил
доцент, к. ф.-м. н. ______________ Д. В. Поплавский

Саратов 2018

СОДЕРЖАНИЕ

1 Вычисление суммы функционального ряда

Необходимо вычислить сумму ряда, на который раскладывается функция:

$$sin(2x) = 2x - \frac{(2x)^3}{3!} + \frac{(2x)^5}{5!} - ...$$

Решение:

```
vector <funValue> CalcSinPartSums(int start, int end, int step) {
	vector <funValue> ans;

	for (int x = start; x <= end; x += step) {
		int sign = 1;
		int factNext = 1;
		double factDenominator = 1;
		double chisl_start = 2 * x;
		double chisl = chisl_start;
		double partSum = 2 * x;

		double prevSum = 0;
		int nslags = 0;
		double diff;

		cout << "x = " << x << endl;

		while (nslags == 0 || abs(diff) > 10e-9) {
			prevSum = partSum;

			nslags++;
			// Вычисляем факториал в знаменателе
			factNext++;
			factDenominator *= factNext;
			factNext++;
			factDenominator *= factNext;
			// Вычисляем числитель
			chisl *= chisl_start;
			chisl *= chisl_start;
			// Вычисляем частичную сумму
			sign *= -1;
			partSum += sign * (chisl / factDenominator);

			diff = partSum - prevSum;
```

```
                }

                funValue thisStep; thisStep.x = x; thisStep.f = partSum;
                ans.push_back(thisStep);
        }
        return ans;
}
//
// Вычислить сумму ряда функции
// для x от -30 до 30 с шагом 5
//
// Ответ представляется в виде массива
// пар: x и функция в точке x
//
vector <funValue> part_sums = (*CalcSinPartSums(-30, 30, 5))
```

-30	-25	-20	-15	-10	-5	0
-4.965e+07	-5.706e+04	-0.6174	0.9878	-0.9129	0.544	0
0.3048	0.2624	-0.7451	0.988	-0.9129	0.544	0

5	10	15	20	25	30
-0.544	0.9129	-0.9878	0.6174	5.706e+04	4.965e+07
-0.544	0.9129	-0.988	0.7451	-0.2624	-0.3048

Рисунок 1 – Верхняя строка — значение x, средняя — значение функции через сумму ряда, нижнее — через вычисление синуса

Программа выдает высокую погрешность при вычислении суммы ряда при $|x| >= 15$. Это объясняется особенностью языка C++, в данном случае происходит переполнение типа данных при вычислении степеней в числителе и факториала в знаменателе.

2 Задача интерполяции

Пусть задана некоторая функция f(x) с дискретным набором значений. Требуется указать некоторую непрерывную определенную на области функцию g(x), такую что g(x^k) = f_k

Для следующих двух методов в качестве тестового примера взят многочлен, полученный как первые четыре точки $f(x) = x^3$

2.1 Интерполяционный многочлен в форме Лагранжа

Решение:

```
vector <funValue> CalcLangranj(vector <funValue> * inputVector) {
        vector <funValue> result_middleInter;
        vector <funValue> input = *inputVector;

        // инициализация двумерного вектора - знаментали
        vector <vector <double> > denom_val;
        for (int i = 0; i < input.size(); i++)
        {
                vector <double> temp(input.size());
                denom_val.push_back(temp);
        }

        // подсчитываем все возможные скобки знаменателя
        for (int i = 0; i < input.size(); i++) {
                for (int j = 0; j < input.size(); j++) {
                        if (i == j) continue;
                        denom_val[i][j] = input[i].x - input[j].x;
                }
        }

        // цикл по всем значениям желаемым 'x'
        for (int i = 0; i < input.size() - 1; i++)
        {
                // точка, которую собираемся вычислить
                funValue newPoint;
                newPoint.x = (input[i + 1].x + input[i].x) / 2;
                newPoint.f = 0;

                // подсчитываем всевозможные скобки числителя
                vector <double> num_val;
                for (int j = 0; j < input.size(); j++)
```

```
                {
                        double temp = newPoint.x - input[j].x;
                        num_val.push_back(temp);
                }

                // Суммируем слагаемые, из которых составляются интерполируемые точки
                for (int j = 0; j < input.size(); j++)
                {
                        double num = 1;
                        double denom = 1;
                        for (int k = 0; k < input.size(); k++)
                        {
                                if (j == k) continue;
                                num *= num_val[k];
                                denom *= denom_val[j][k];
                        }
                        newPoint.f += input[j].f * num / denom;
                }
                result_middleInter.push_back(input[i]);
                result_middleInter.push_back(newPoint);
        }
        result_middleInter.push_back(input[input.size() - 1]);

        return result_middleInter;
}

void startInterpolation() {
    // все 'x' и 'f' из дано ТУТ
    vector <funValue> power3;
    funValue temp = { 1, 1 };
    power3.push_back(temp);
    temp = { 2, 8 };
    power3.push_back(temp);
    temp = { 3, 27 };
    power3.push_back(temp);
    temp = { 4, 64 };
    power3.push_back(temp);

    vector <funValue> result_middleInter = CalcLangranj(&power3);

    for (int i = 0; i < result_middleInter.size(); i++) {
```

```
        cout << "| " << setw(6) << setprecision(4) << result_middleInter[i].x << " ";
    }
    cout << endl;
    for (int i = 0; i < result_middleInter.size() * 9; i++) {
        cout << "-";
    }
    cout << endl;
    for (int i = 0; i < result_middleInter.size(); i++) {
        cout << "| " << setw(6) << setprecision(4) << result_middleInter[i].f << " ";
    }
    cout << endl;
}
c
```

```
|      1 |    1.5 |      2 |    2.5 |      3 |    3.5 |      4
---------------------------------------------------------------
|      1 |  3.375 |      8 |  15.63 |     27 |  42.88 |     64
```

Рисунок 2 – Верхняя строка — значение x, нижняя — значение функции, чередуются интерполируемые и исходные

Погрешности нет благодаря подбору многочлена степени равной количеству искомых точек.

2.2 Интерполяционный многочлен в форме Ньютона

Решение:

```
vector <funValue> CalcNewton(vector <funValue> * inputVector) {
        vector <funValue> result_middleInter;
        vector <funValue> input = *inputVector;

        // инициализациЯ массива разделенных разностей
        vector <vector <double>> divDiff;

        for (int i = 0; i < input.size(); i++) {
                vector <double> temp;
                divDiff.push_back(temp);
        }

        // разделенные разности нулевого порЯдка
        for (int i = 0; i < input.size(); i++) {
                divDiff[0].push_back(input[i].f);
        }
```

```
        // разделенные разности других порЯдков
        for (int i = 1; i < input.size(); i++) {
                for (int j = 0; j < input.size() - i; j++) {
                        double calcThisDiv = (divDiff[i - 1][j + 1] - divDiff[i - 1][
                        divDiff[i].push_back(calcThisDiv);
                }
        }

        // Ќахождение значение функции в срединных точках
        for (int i = 0; i < input.size() - 1; i++) {
                funValue newPoint; newPoint.x = (input[i + 1].x + input[i].x) / 2;

                // нахождение множителей вида (x - x0), (x - x1), ...
                vector <double> xdifx;
                xdifx.push_back(newPoint.x - input[0].x);

                for (int j = 1; j < input.size() - 1; j++) {
                        xdifx.push_back(xdifx[j - 1] * (newPoint.x - input[j].x));
                }

                // подсчет значениЯ функции в данной неузловой точке
                newPoint.f = 0;
                newPoint.f += divDiff[0][0];
                for (int j = 1; j < input.size(); j++) {
                        newPoint.f += divDiff[j][0] * xdifx[j - 1];
                }
                result_middleInter.push_back(newPoint);
        }

        return result_middleInter;
}

void startInterpolation() {
    // все 'x' и 'f' из дано ТУТ
    vector <funValue> power3;
    funValue temp = { 1, 1 };
    power3.push_back(temp);
    temp = { 2, 8 };
    power3.push_back(temp);
    temp = { 3, 27 };
```

```
    power3.push_back(temp);
    temp = { 4, 64 };
    power3.push_back(temp);

    vector <funValue> result_middleInter = CalcLangranj(&power3);

    for (int i = 0; i < result_middleInter.size(); i++) {
        cout << "| " << setw(6) << setprecision(4) << result_middleInter[i].x << " ";
    }
    cout << endl;
    for (int i = 0; i < result_middleInter.size() * 9; i++) {
        cout << "-";
    }
    cout << endl;
    for (int i = 0; i < result_middleInter.size(); i++) {
        cout << "| " << setw(6) << setprecision(4) << result_middleInter[i].f << " ";
    }
    cout << endl;
}
C
```

```
|    1.5 |    2.5 |    3.5
---------------------------
|  3.375 |  15.63 |  42.88
```

Рисунок 3 – Верхняя строка — значение x, нижняя — значение функции, найденное с помощью интерполяции

3 Численные методы решения СЛАУ

3.1 Метод Гаусса

Решение с примитивным пользовательским интерфейсом для демонстрации работы:

```
double * gauss(double **a, double *y, int n)
{
        double *x, max;
        int k, index;
        const double eps = 0.00001;  // точность
        x = new double[n];
        k = 0;
        while (k < n)
        {
                // Поиск строки с максимальным a[i][k]
                max = abs(a[k][k]);
                index = k;
                for (int i = k + 1; i < n; i++)
                {
                        if (abs(a[i][k]) > max)
                        {
                                max = abs(a[i][k]);
                                index = i;
                        }
                }
                // Перестановка строк
                if (max < eps)
                {
                        // нет ненулевых диагональных элементов
                        cout << "Решение получить невозможно из-за нулевого столбца "
                        cout << index << " матрицы A" << endl;
                        return 0;
                }
                for (int j = 0; j < n; j++)
                {
                        double temp = a[k][j];
                        a[k][j] = a[index][j];
                        a[index][j] = temp;
                }
                double temp = y[k];
                y[k] = y[index];
```

```
			y[index] = temp;
			// Нормализация уравнений
			for (int i = k; i < n; i++)
			{
				double temp = a[i][k];
				if (abs(temp) < eps) continue;
				for (int j = 0; j < n; j++)
					a[i][j] = a[i][j] / temp;
				y[i] = y[i] / temp;
				if (i == k)  continue;
				for (int j = 0; j < n; j++)
					a[i][j] = a[i][j] - a[k][j];
				y[i] = y[i] - y[k];
			}
			k++;
		}
		// обратный ход метода Гаусса
		for (k = n - 1; k >= 0; k--)
		{
			x[k] = y[k];
			for (int i = 0; i < k; i++)
				y[i] = y[i] - a[i][k] * x[k];
		}
		return x;
}

// Функция вывода введенной СЛАУ
void outInput(double **a, double *y, int n)
{
		for (int i = 0; i < n; i++)
		{
			for (int j = 0; j < n; j++)
			{
				cout << a[i][j] << "*x" << j;
				if (j < n - 1)
					cout << " + ";
			}
			cout << " = " << y[i] << endl;
		}
}

```

```
// Ввод данных и запуск метода Гаусса
void startGauss() {
    setlocale(LC_ALL, "RUSSIAN");
    double **a, *y, *x;
    int n;
    cout << "Введите количество уравнений: ";
    cin >> n;
    a = new double*[n];
    y = new double[n];
    for (int i = 0; i < n; i++)
    {
        a[i] = new double[n];
        for (int j = 0; j < n; j++)
        {
            cout << "a[" << i << "][" << j << "]= ";
            cin >> a[i][j];
        }
    }
    for (int i = 0; i < n; i++)
    {
        cout << "y[" << i << "]= ";
        cin >> y[i];
    }
    outInput(a, y, n);
    x = gauss(a, y, n);
    for (int i = 0; i < n; i++)
        cout << "x[" << i << "]=" << x[i] << endl;
}
c
```

```
Введите количество уравнений: 3
a[0][0]= 3
a[0][1]= 2
a[0][2]= -5
a[1][0]= 2
a[1][1]= -1
a[1][2]= 3
a[2][0]= 1
a[2][1]= 2
a[2][2]= -1
y[0]= -1
y[1]= 13
y[2]= 9
3*x0 + 2*x1 + -5*x2 = -1
2*x0 + -1*x1 + 3*x2 = 13
1*x0 + 2*x1 + -1*x2 = 9
x[0]=3
x[1]=5
x[2]=4
```

Рисунок 4 – Ввод данных, вывод СЛАУ на основе данных, вывод ответа

4 Численные методы решения обыкновенных дифференциальных уравнений

4.1 Решение задачи Коши методом Эйлера

Рассмотрим следующую задачу — однородное дифференциальное уравнение первого порядка

$$\begin{cases} y' = f(x, y) \quad \text{y = y(x)} \\ y(x_0) = y_0 \end{cases}$$

Она может быть решена методом Эйлера.

Пусть y(x) = $x^3 + x + 1$

Решение:

```
// Функция задана вручную
void methodEuler(double from, double to, double step, int V) {
    int n = (int)((to - from) / step);
    // Значения x
    double * x = new double[n];
    double * yt = new double[n];
    double * ym = new double[n];
    double * e = new double[n];
    for (int i = 0; from < to + step / 2; from += step) {
        x[i] = from;
        if (i == 0) {
            yt[0] = ym[0] = x[0] * x[0] * x[0] + x[0] + 1;
            e[0] = 0;
            i++;
            continue;
        }
        yt[i] = V * x[i] * x[i] * x[i] + V * x[i] + V;
        ym[i] = V + step * (3 * V * x[i] * x[i] + V - V * x[i] * x[i] * x[i] - V - V
        e[i] = abs(yt[i] - ym[i]);
        i++;
    }

    cout << "x = ";
    for (int i = 0; i < n; i++) {
        cout << setw(9) << setprecision(2) << i;
    }
    cout << endl << "y_t ";
```

```
    for (int i = 0; i < n; i++) {
        cout << setw(9) << setprecision(2) << yt[i];
    }
    cout << endl << "y_m ";
    for (int i = 0; i < n; i++) {
        cout << setw(9) << setprecision(2) << ym[i];
    }

    cout << endl << "e:  ";
    for (int i = 0; i < n; i++) {
        cout << setw(9) << setprecision(2) << e[i];
    }
}

methodEuler(1, 11, 1, 1);
```

x =	1	1.2	1.4	1.6	1.8	2	2.2	2.4	2.6	2.8	3
y_t	3	3.9	5.1	6.7	8.6	11	14	17	21	26	31
y_m	3	4.3	5.6	7	8.4	9.8	11	12	13	14	15
e:	0	0.35	0.48	0.33	0.19	1.2	2.7	4.8	7.8	12	16

Рисунок 5 – Метод Эйлера для x от 1 до 3 с шагом 0.2

x =	0	1	2	3	4	5	6	7	8	9
y_t	3	11	31	69	1.3e+02	2.2e+02	3.5e+02	5.2e+02	7.4e+02	1e+03
y_m	3	6	4	-15	-69	-1.8e+02	-3.8e+02	-7.1e+02	-1.2e+03	-1.9e+03
e:	0	5	27	84	2e+02	4.1e+02	7.4e+02	1.2e+03	1.9e+03	2.9e+03

Рисунок 6 – Метод Эйлера для x от 1 до 11 с шагом 1

Как можно заметить, погрешность возрастает при увеличении количества шагов и расстояния между точками. Так как интегральное слагаемое считалось по квадратурной формуле левого прямоугольника, то погрешность только возрастает. Её можно было бы уменьшить, воспользовавшись усовершенствованными методами Эйлера.