ИНСТИТУТ ТРАНСПОРТА И СВЯЗИ

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ НАУК И ТЕЛЕКОММУНИКАЦИЙ

Лабораторная работа №2
По дисциплине
«Численные методы и прикладное программирование»

Тема:
«Методы приближения функции.
Интерполяция и аппроксимация»

Работу выполнили:

Дзенис Ричард
Кобелев Денис
Якушин Владислав

Рига
2017 г.

Содержание

1 Формулировка задания

- Реализовать программным путём аппроксимацию функции по заданным точкам методом наименьших квадратов (МНК);
- Реализовать программным путём интерполяцию функции по заданным точкам с помощью кубических сплайнов;
- Используя примеры функций приведённых ниже, графически сравнить получившиеся результаты;
- Сделать выводы по реализованным методам.

$$y = \sin(5x) \cdot e^{x} \tag{1}$$

где: $x \in [-2; +2]$; $\Delta x = 0.5$

x	-3.2	-2.1	0.4	0.7	2	2.5	2.777
y	10	-2	0	-7	7	0	0

(2)

2 Аппроксимация методом наименьших квадратов

Аппроксимацию методом МНК можно разбить на 2 этапа:

1. Построение матрицы коэффициентов СЛАУ согласно универсальной СЛАУ МНК;
2. Решение СЛАУ для нахождения аппроксимирующего полинома.

В силу того, что универсальная СЛАУ для МНК является симметричной, то для экономии памяти можно сохранять только верную или нижнюю треугольную матрицу.

2.1 Листинги

2.1.1 Листинг построения нижней треугольной матрицы коэффициентов для универсальной СЛАУ МНК

```
// Input: points {x, y}, order of approximation
// Output: lower triangular matrix of coefficients
// with right-hand side constants
std::vector<std::vector<double>>
find_coef(std::vector<point> &points, size_t order)
{
    double coef;
    std::vector<std::vector<double>> result;
    result.resize(order + 1);
    for (size_t row = 0; row < order + 1; row++) {
        // build only lower triangular matrix (because upper == lower)
        result[row].resize(row + 2);
        // left-hand side
        for (size_t col = 0; col < row + 1; col++) {
            coef = 0.0;
            for (auto &&point : points) {
                coef += pow(point.x, row) * pow(point.x, col);
            }
            result[row][col] = coef;
        }
        // right-hand side
        coef = 0.0;
        for (size_t i = 0; i < points.size(); i++) {
            coef += points[i].y * pow(points[i].x, row);
        }
        result[row].back() = coef;
    }
    return result;
}
```

Для того чтобы воспользоваться уже разработанной программой решения СЛАУ методом исключения Гаусса (в первой лабораторной работе), необходимо вывести всю матрицу целиком, с вектором правой части.

2.1.2 Листинг вывода СЛАУ

```
auto coefficients = find_coef(points, order);
assert(coefficients.size() == order + 1);
std::cout << coefficients.size() << std::endl;
for (size_t row = 0; row < order + 1; row++) {
    for (size_t col = 0; col < row + 1; col++)
        std::cout << coefficients[row][col] << '\t';
    for (size_t col = row + 1; col < order + 1; col++)
        std::cout << coefficients[col][row] << '\t';
    std::cout << coefficients[row].back() << '\n';
}
std::cout.flush();
```

Далее СЛАУ решается с помощью программы разработанной в ходе первой лабораторной работы.

2.2 Результаты работы метода

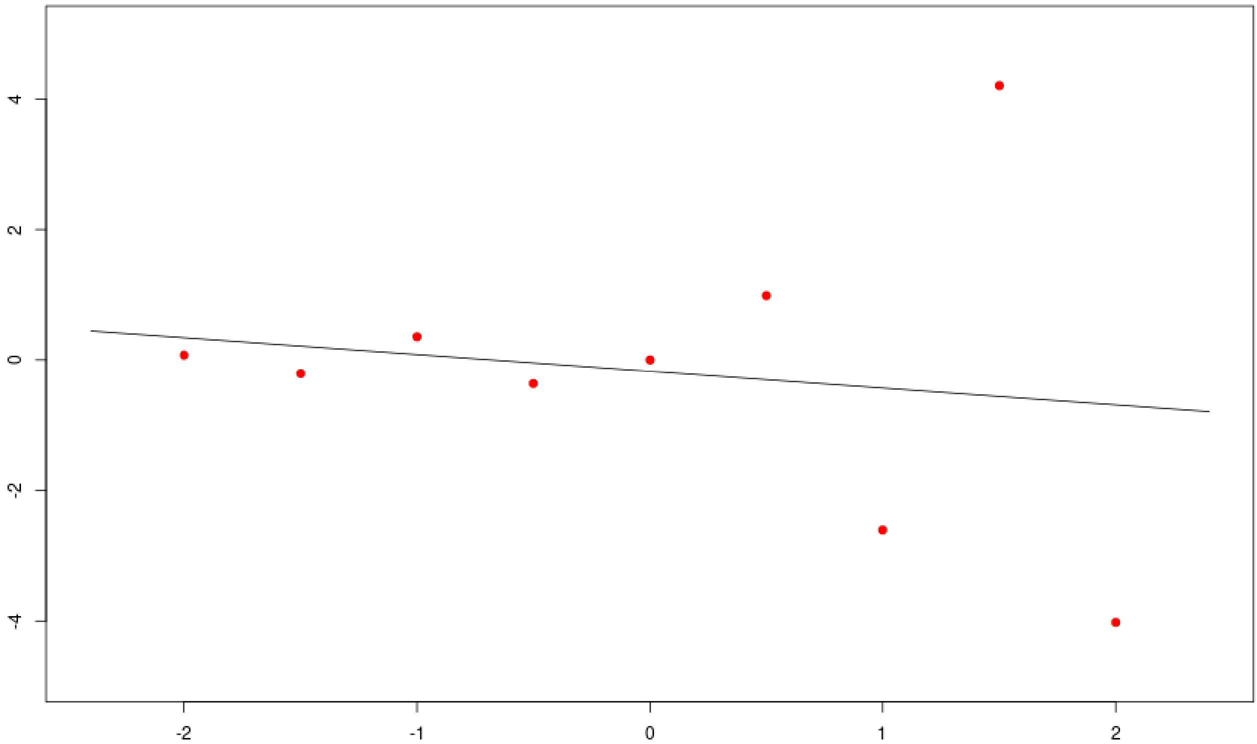

Рис. 1: График аппроксимирующего полинома 1ого порядка для задания №1

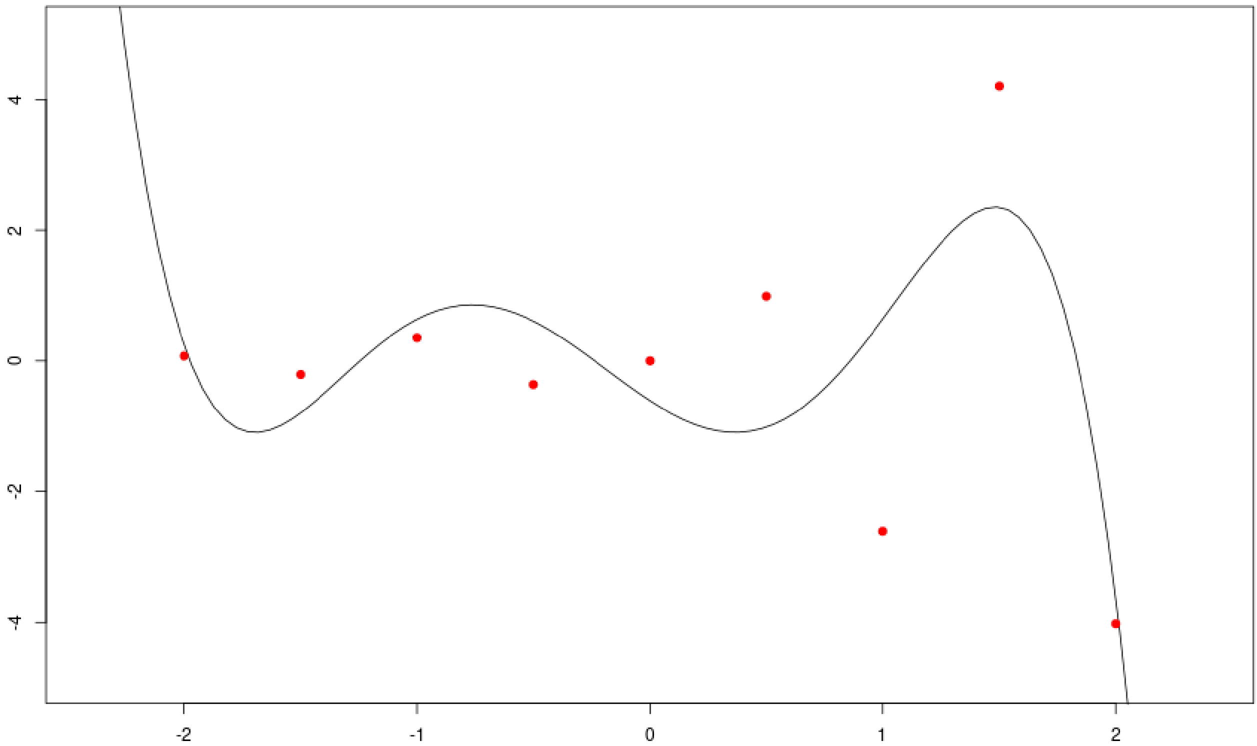

Рис. 2: График аппроксимирующего полинома 5ого порядка для задания №1

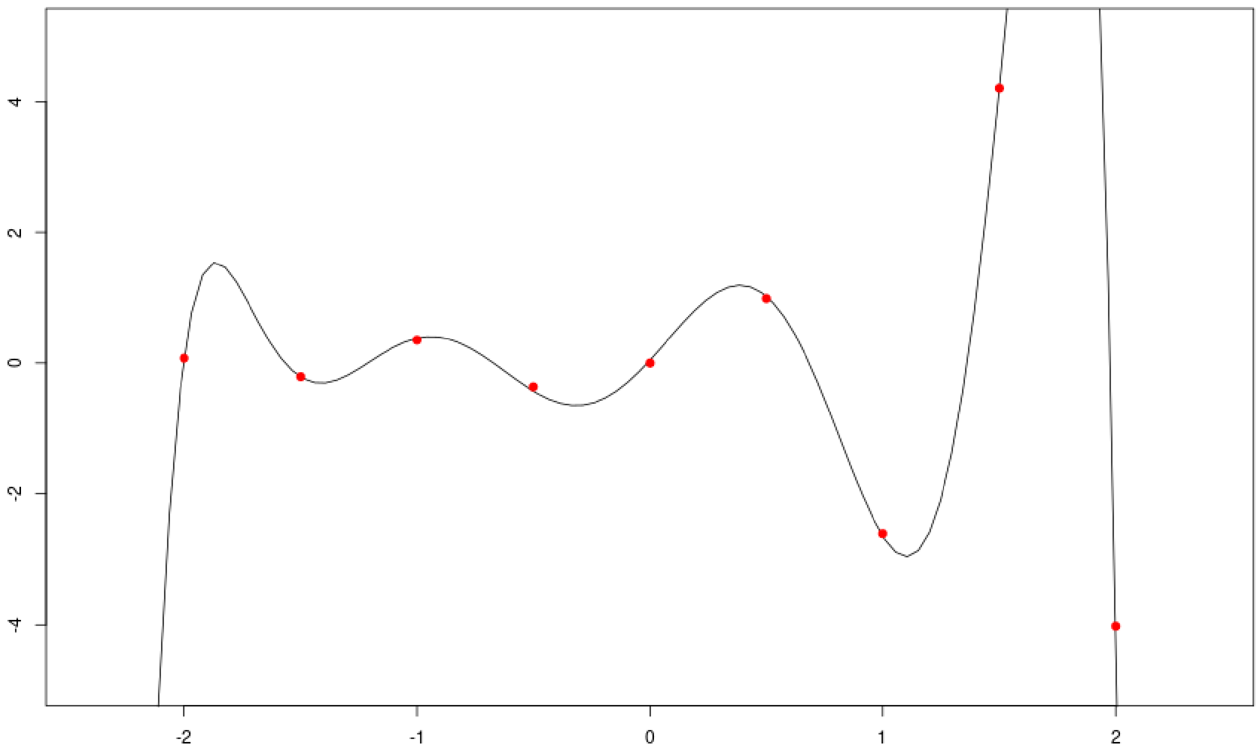

Рис. 3: График аппроксимирующего полинома 8ого порядка для задания №1

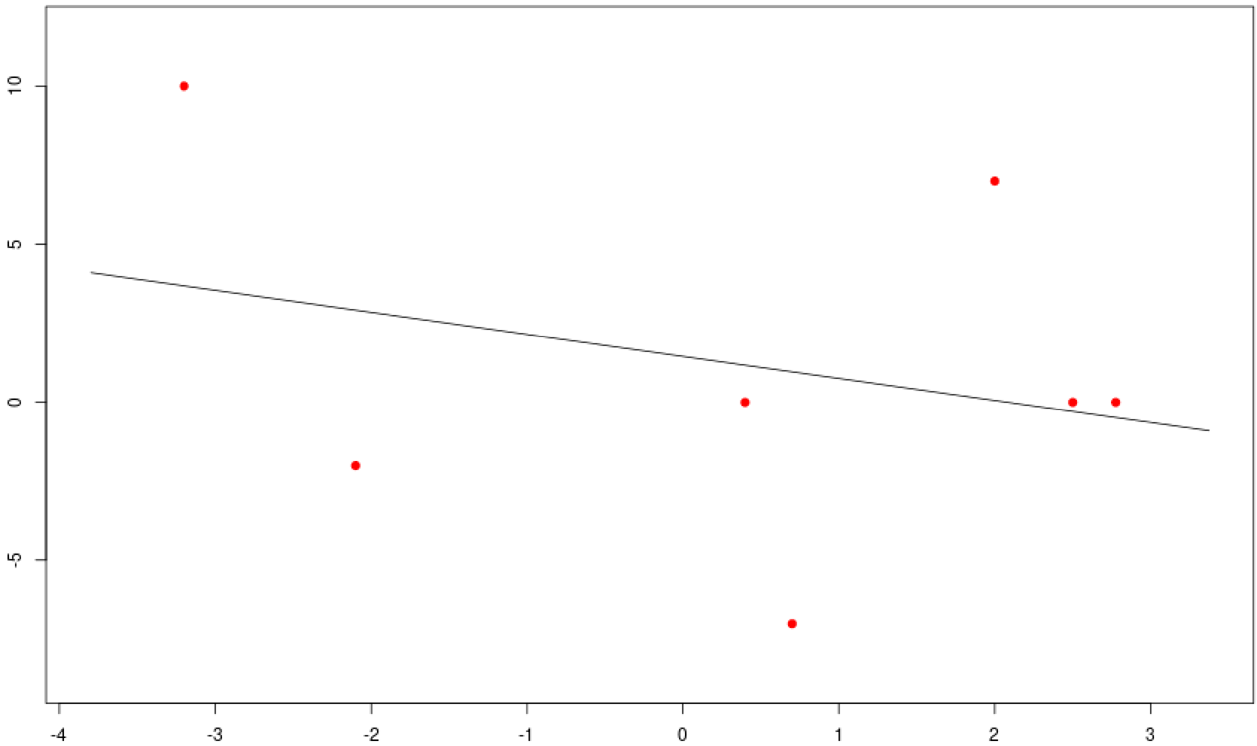

Рис. 4: График аппроксимирующего полинома 1ого порядка для задания №2

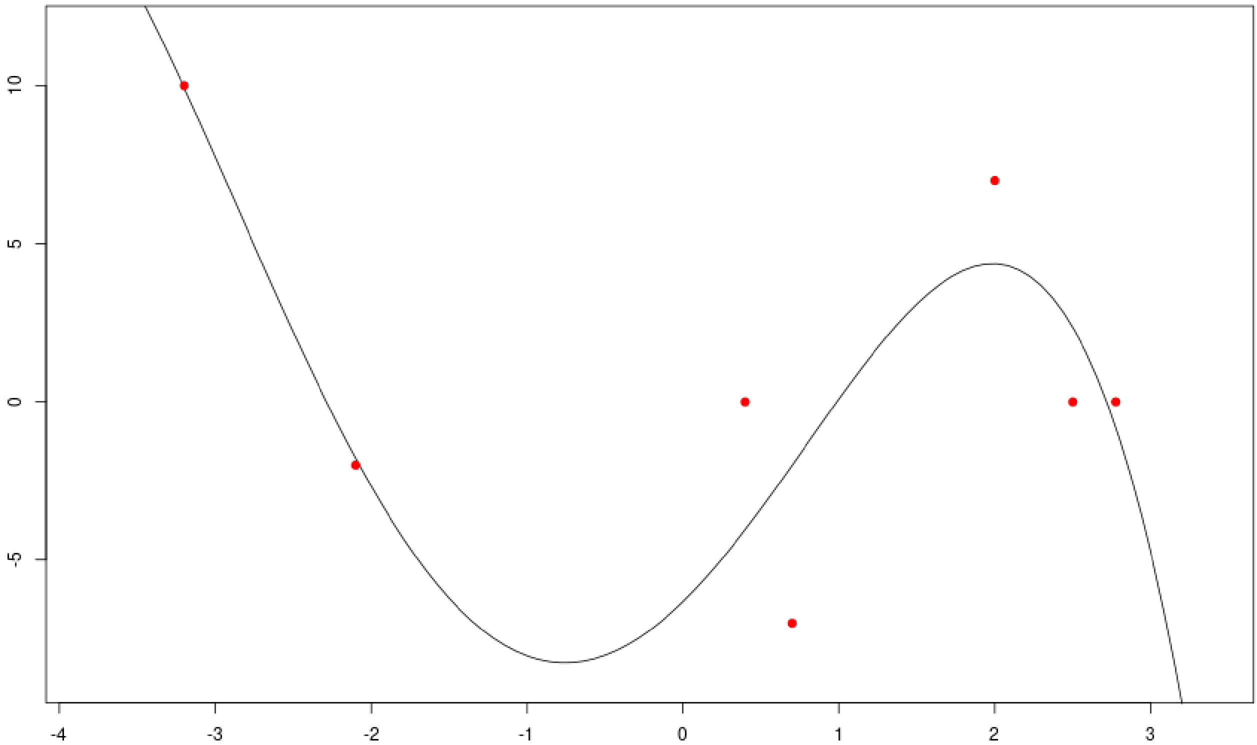

Рис. 5: График аппроксимирующего полинома 4ого порядка для задания №2

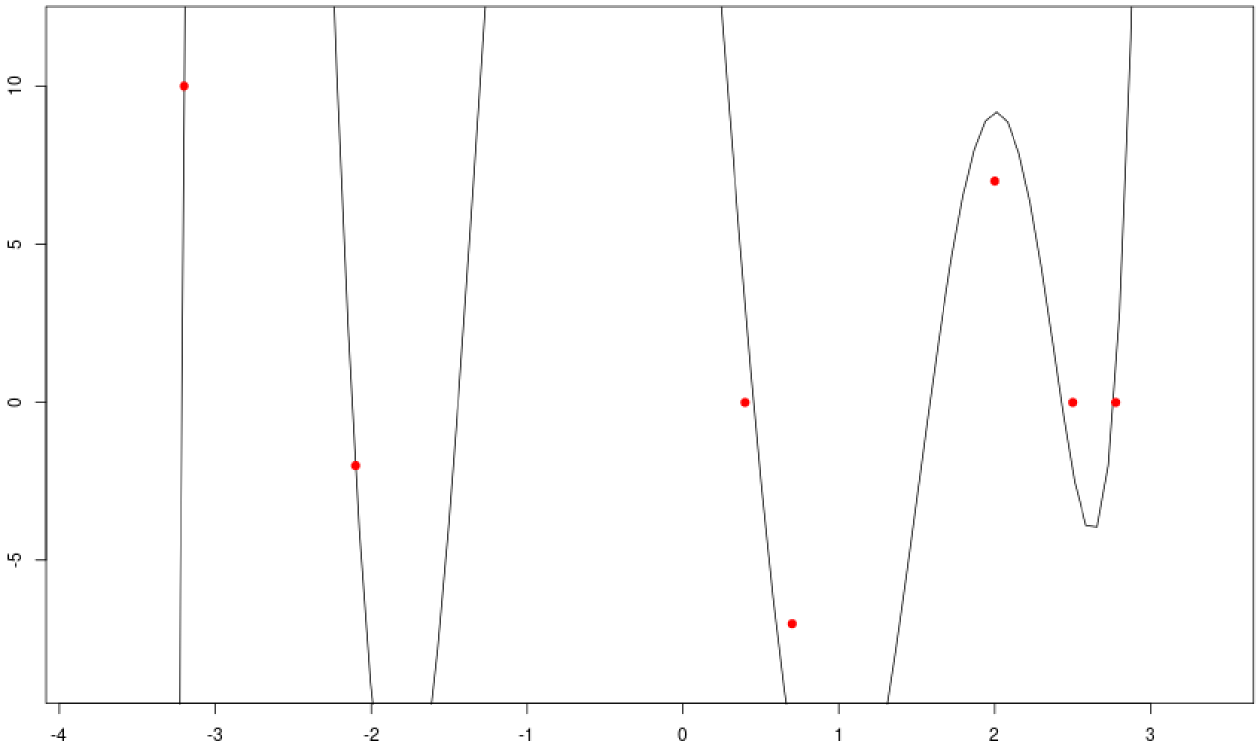

Рис. 6: График аппроксимирующего полинома 7ого порядка для задания №2

3 Кубическая сплайн интерполяция

3.1 Листинг

3.1.1 Листинг функции решения СЛАУ с 3ёх диагональными матрицами коэффициентов

```
inline void
solveTridiagonal(std::vector<double> &x, size_t offset,
        std::function<double(size_t j, size_t i)> a,
        std::function<double(size_t j)> b)
{
    ssize_t k;
    ssize_t size = x.size() - offset;
    std::vector<double> alpha(size), beta(size);
    alpha[0] = a(0,1) / a(0,0);
    beta[0] = b(0) / a(0,0);
    for (k = 1; k < size; k++) {
        alpha[k] = a(k,k+1) / (a(k,k) - a(k,k-1) * alpha[k-1]);
        beta[k] = (b(k) - a(k,k-1) * beta[k-1]) /
            (a(k,k) - a(k,k-1) * alpha[k-1]);
    }
    x[k + offset] = beta[k];
    for (; k >= 0; k--) {
        x[k + offset] = beta[k] - alpha[k] * x[k + offset + 1];
    }
}
```

3.1.2 Листинг функции нахождения коэффициентов сплайнов

```
coefficients
interpolate(std::vector<point> &points)
{
    ssize_t size = points.size() - 1;
    coefficients coefs(size);
    auto h = [&](size_t i) {
```

```
        return points[i + 1].x - points[i].x;
    };
    auto a = [&](size_t j, size_t i) -> double {
        if (j > i) return h(j);
        else if (j < i) return h(j + 1);
        else return 2 * (h(j) + h(j + 1));
    };
    auto b = [&](size_t j) {
        return 3 * (
                (points[j + 2].y - points[j + 1].y) / h(j + 1) -
                (points[j + 1].y - points[j + 0].y) / h(j + 0)
                );
    };
    if (size > 0) {
        coefs.c[0] = 0;
    }
    if (size > 1) {
        solveTridiagonal(coefs.c, 1, a, b);
    }
    for (ssize_t i = 0; i < size; i++) {
        coefs.a[i] = points[i].y;
        coefs.d[i] = (coefs.c[i + 1] - coefs.c[i]) / (3 * h(i));
        coefs.b[i] = (points[i + 1].y - points[i].y) / h(i)
                   - (coefs.c[i + 1] + 2 * coefs.c[i + 0]) * h(i) / 3.0;
    }
    return coefs;
}
```

3.2 Результаты работы метода

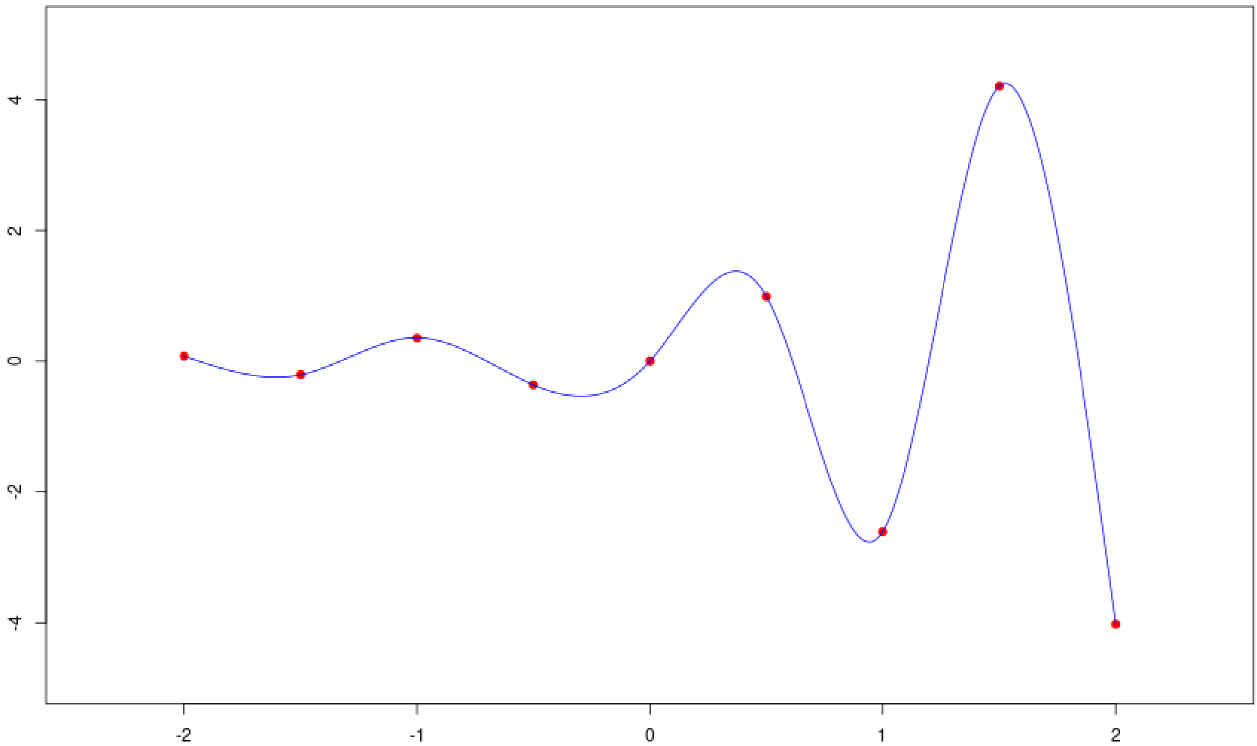

Рис. 7: График сплайн интерполяции для задания №1

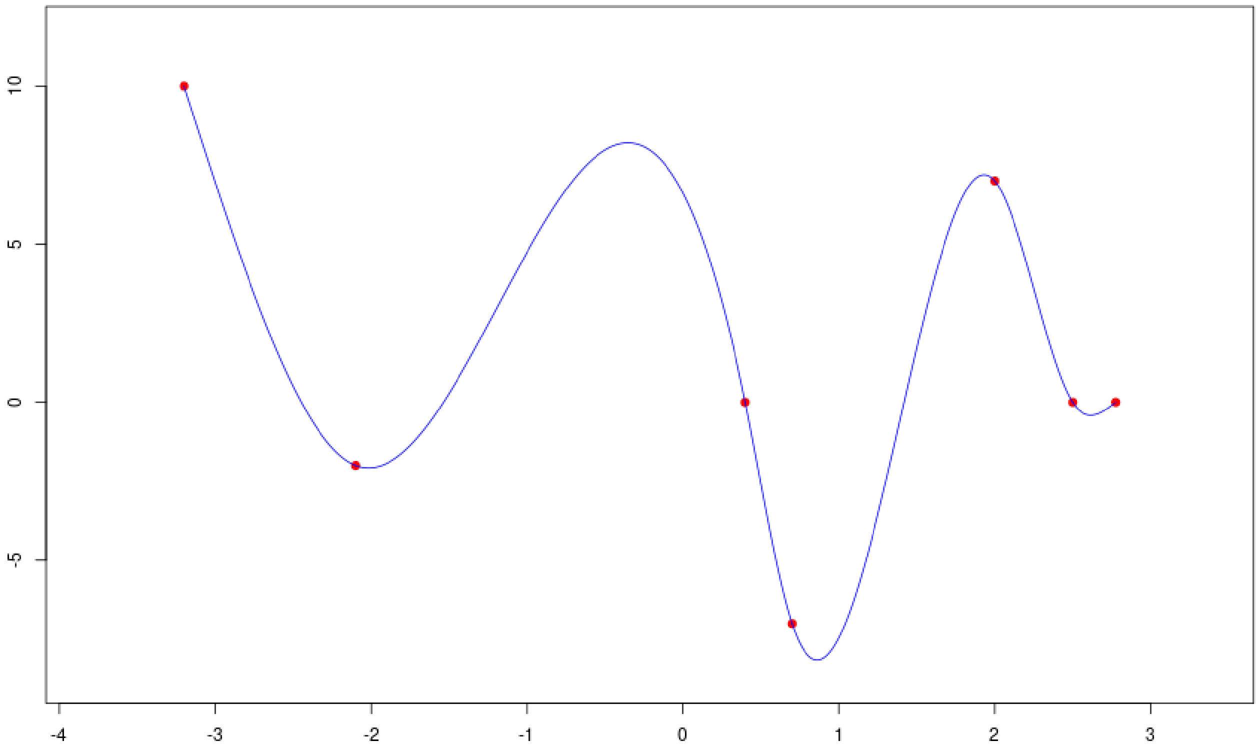

Рис. 8: График сплайн интерполяции для задания №2

4 Графическое сравнение двух методов

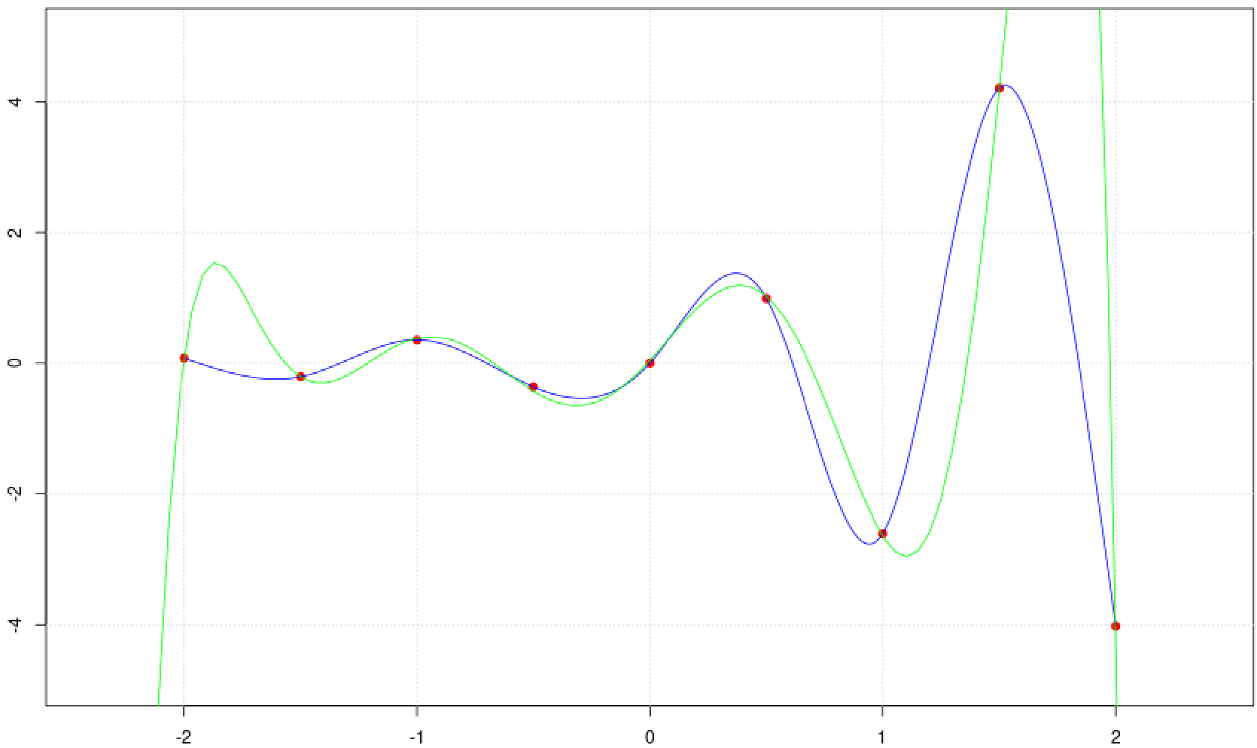

Рис. 9: График аппроксимирующего полинома 8ого порядка (зелёный), и график сплайна (синий) для задания №1

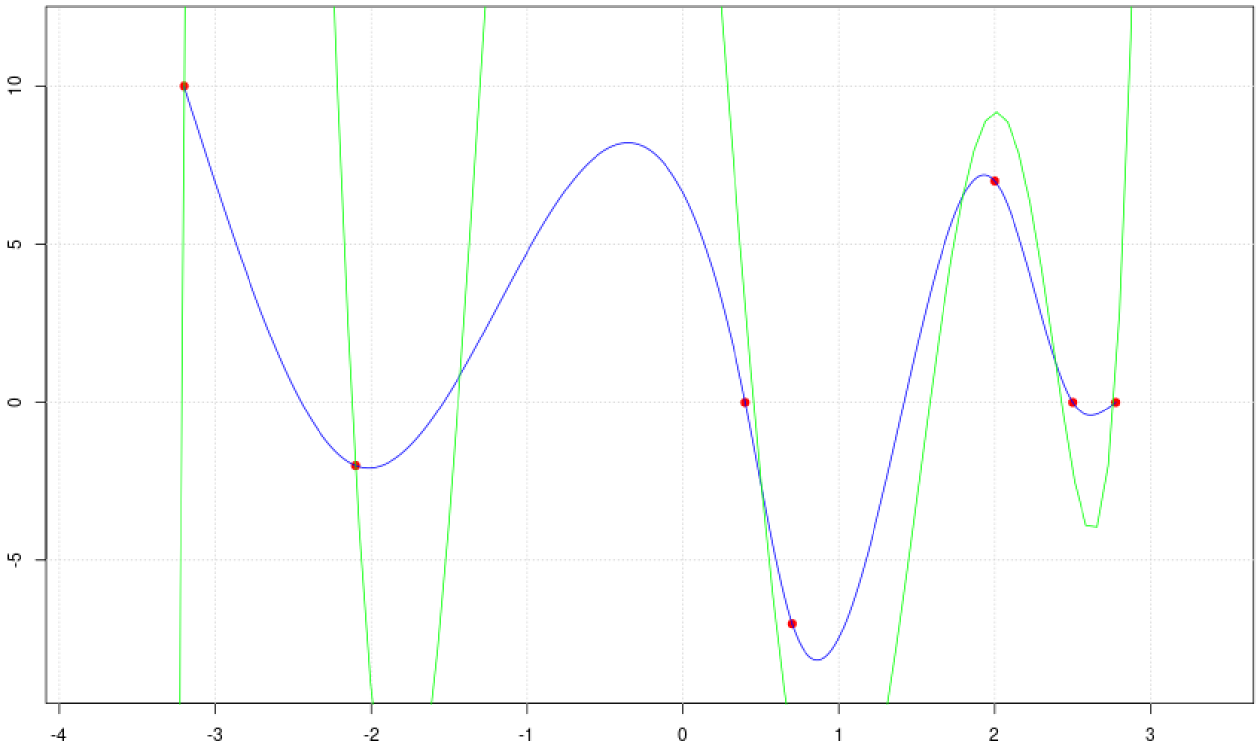

Рис. 10: График аппроксимирующего полинома 7ого порядка (зелёный), и график сплайна (синий) для задания №2

5 Выводы

В ходе выполнения лабораторной работы были реализованы программным способом методы:

- сплайн интерполяции; и
- метода наименьших квадратов.

В результате было разработано две программы на языке «C++» реализующие данные методы. Для МНК было сделано только составление (и вывод) универсальной МНК СЛАУ. А также были разработаны вспомогательные скрипты: для отображения графиков (на языке «R»); для решения СЛАУ библиотечной функцией методом LU-разложения (на языке «Python»); для связки всех программ и скриптов (на языке «Bash»).

Перед сравнением обоих методов стоит отметить, что они выполняют разные задачи: один является методом аппроксимации, а второй – методом интерполяции, по этой причине для сравнения используются данные с графиков (9 и 10), когда аппроксимация переходит в её частный случай – интерполяцию.

Визуальное сравнение “точности”

Как можно заметить метод аппроксимации МНК имеет куда больший разброс при переходе между точками в обоих случаях, в отличии от сплайн интерполяции. Данное обстоятельство объясняется особенностью этих методов. При **аппроксимации** МНК вычисляется один полином поясняющий все точки. А при сплайн **интерполяции** строятся локальные полиномы (3его порядка) между каждой парой точке, поясняющие поведение кривой между меньшим набором точек, что и свидетельствует его гладкости.

Сравнение сложности вычислений

Сложность решения задачи аппроксимации методом МНК теоретически выше чем сплайн интерполяция, т.к. в то время, как основные этапы МНК требуют $O(n^3)$ времени, в методе интерполяции сплайнами все этапы занимают $O(n)$ времени.

Что можно видеть во времени выполнения программы обоих методов.

Составление универсальной СЛАУ для МНК	$O(n^3)$
Решение СЛАУ	$O(n^3)$
Общая сложность	$\boldsymbol{O(n^3)}$

Таблица 1: Временная сложность вычисления для аппроксимации МНК

Подготовка трёх-диагональной матрицы коэффициентов для c_i	$O(n)$
Решение СЛАУ с трёх-диагональной матрицей	$O(n)$
Вычисление остальных коэффициентов b_i, d_i	$O(n)$
Общая сложность	$\boldsymbol{O(n)}$

Таблица 2: Временная сложность вычисления для аппроксимации МНК

Сравнение по затратам памяти

В интерполяции сплайнов необходимо хранить всю таблицу сплайнов размер которой зависит от количества заданных точек.

$$M_{\text{spline}} = 5 \times n \tag{3}$$

А в МНК необходимо хранить матрицу размер которой зависит от порядка аппроксимации – только во время вычислений, а на выходе получается только полином.

$$M_{\text{МНК}} = P \tag{4}$$

Таким образом затраты по памяти данных методов плохо сравнимы, однако, можно утверждать, что в МНК требуется меньше памяти по завершению всех вычислений.

Сравнение сложности реализации

В ходе данной работы было установлено, что время потраченное на реализацию МНК оказалось несколько меньше, чем на реализацию метода интерполяции с помощью сплайнов. Это объясняется тем, что для МНК было достаточно реализовать составление матрицы. В то время как при реализации сплайнов основное время ушло на отладку, и поиска ошибки в индексах. Данную ошибку было допустить очень легко, т.к. сплайны в теории нумеровались начиная с 1, а в программе с 0.

Сравнение применимости

Сплайн-интерполяцию следует применять в тех случаях, когда нужно показать поведение некоторого феномена основываясь на наборе данных, так как график функции легко вычисляется и получается более гладким чем график построенный методом аппроксимации порядка n. Так же её следует использовать вместо методов аппроксимации порядка n, так как сплайн будет более “гладким”.

С помощью МНК-аппроксимации возможно отобразить поведение исследуемых данных в общем (тренд) и избежать зависимости от выбросов в показаниях. Тогда как сплайн интерполяцию можно применять только для интерполяции меж-точечных значений, и не применим для большого набора зашумлённых данных.