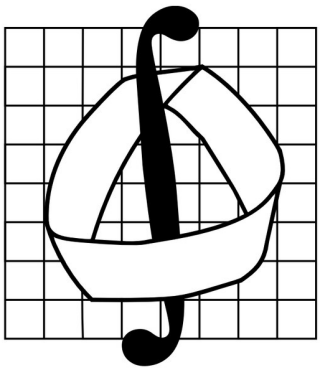
Московский государственный университет имени М. В. Ломоносова
механико-математический факультет
кафедра Теории Вероятностей



Отчёт по практикуму на ЭВМ
студента 409 группы
Сидоренко Артура Павловича

Численное интегрирование в одно- и двумерном случае

Москва, 2019

1 Постановка одномерной задачи

Пусть задан отрезок $[a, b]$ и функция $f : \mathbb{R} \to \mathbb{R}$, обладающая достаточной степенью гладкости. Требуется численно найти интеграл $(f) = \int_a^b f(x)p(x)dx$, где p - некоторая весовая функция.

Так как на практике мы можем вычислить значение функции лишь в конечном числе точек, то применяется метод квадратурных формул, т.е. составляется сумма вида $S(f) = \sum_1^n c_i f(x_i)$. Точки x_i принято называть узлами квадратуры.

Здесь мы разберём формулы центральных прямоугольников, Ньютона-Котеса и Гаусса дял четырёх узлов и для $p(x) = 1$.

2 Формула центральных прямоугольников

Это самая простая формула. Для отрезка $[a, b]$ $S_{rect}(f) = f(\frac{a+b}{2})*(b-a)$. Данная формула имеет алгебраический порядок точности 1, т.е. точно вычисляет интегралы от линейных функций. Верно следующее утверждение о её погрешности.

Утверждение 2.1. *Пусть функция f дважды непрерывно дифференцируема на отрезке $[a, b]$. Тогда для некоторой точки $\xi \in (a, b)$: $I(f) - S_{rect}(f) = \frac{1}{24} f''(\xi)(b - a)^3$.*

3 Формула Ньютона-Котеса

В общем случае это формула для $p(x) = 1$ вида $S_{nc}^{n+1}(f) = \sum_0^n c_i f(x_i)$ и узлов $x_i = a + \frac{i}{n}(b-a)$, $i = 0 \dots n$. Далее коэффициенты c_i рассчитываются так, чтобы квадратура была точна на многочленах 1, x, ..., x^n. Это можно достичь: получится линейная система с матрицей Вандермонда. Таким образом, для $n+1$ уздов формула имеет алгебраический порядок точности n.

Здесь мы рассматриваем формулу для $n = 4$. Числа c_i можно сосчитать на ЭВМ, так что обратим внимание на оценку погрешностей.

Утверждение 3.1. *Пусть функция $f \in C^{(4)}[a, b]$. Тогда*

$$|I(f) - S_{nc}^{n+1}(f)| \leq \frac{1}{480} ||f^{(IV)}|| (b - a)^5, \tag{1}$$

где $|| \dots ||$ - равномерная норма на $[a, b]$.

4 Формула Гаусса

Наиболее прогрессивной в этой области являются квадратуры, основанные на ортогональных многочленах. Оказывается, если выбирать узлы в соответствии с их корнями, то мы получим повышение в точности.

Постановка: пусть нам дан отрезок $[a,b]$, число n, вес p и функция f, и мы хотим найти x_i и c_i, чтобы квадратура $S(f) = \sum_1^n c_i f(x_i)$ имела максимальный алгебраический порядок точности.

Конспективно предъявим решение задачи в виде серии утверждений.

Определение 4.1. Рассмотрим $< f, g >= \int_a^b p(x)f(x)g(x)dx$, где $p(x) > 0$ п.в. на $[a,b]$. Многочлены $\{\psi_k\}_{k=0}^{\inf}$, где все ψ_k вещественные, различные и ненулевые, называются ортогональными, если для всех i,j: $< \psi_i, \psi_j >= 0$ при $i \neq j$.

Далее считаем, что все $\psi_k(x) = x^k + b_1 x^{k-1} + \cdots + b_k$. Ортогональные многочлены действительно существуют, так как их можно явно вычислить по методу Грамма-Шмидта.

Утверждение 4.1. *ψ_n имеет n корней на (a,b).*

Таким образом, мы получаем систему узлов x_k в виде корней ортогонального многочлена. Методом неопределённых коэффициентов можно вычислить c_i, чтобы получить алгебраический порядок $n-1$. Но при этом в подарок мы получаем повышение точности до $2n-1$, а нетрудно показать, что далее повысить порядок нельзя.

Теорема 4.1. *Для дюбого натурального n и для любого отрезка $[a,b]$ $\exists!$ квадратурная формула n узлами алгебраического порядка $2n-1$, причём её узлы суть корни n-го ортогонального многочлена.*

Утверждение 4.2. *Для квадратуры Гаусса с $n = 4$ узлами верна оценка:*

$$R \leq \frac{1}{16 * 9!} ||f^{(8)}|| (b-a)^5 \tag{2}$$

Теперь единственный вопрос - как построить этот ортогональный многочлен. На самом деле, для $n = 4$, отрезка $[-1,1]$ и веса $p(x)$ это сделать нетрудно. Достаточно воспользоваться соображениями симметрии и заметить, что коэффициенты при нечётных степенях будут равны нулю. Далее в явном виде надо взять скаярное произведение $x^4 + ax^2 + b$ с 1 и x^2, и получить систему из двух линейных уравнений. Ответ можно выписать в красивом виде: $\psi_4(x) = 35x^4 - 30x^2 + 3$. Поиск корней здесь нетруден, так как мы имеем биквадратное уравнение. Коэффициенты c_i можно высчитать методом неопределённых коэффициентов, причём при помощи компьютерной программы.

5 Составные квадратуры

Перечисленные выше квадратуры годятся для маленьких отрезков. КОгда дан большой отрезок, принято его разделять на малые части и применять квадратуру для каждого кусочка отдельно. Используя оценки погрешности для простых квадратур, легко получить оцнеки для квадратур составных.

Утверждение 5.1. *Если f достаточно гладкая, то верны следующие оценки погрешности для составных квадратур с N отрезками (центральных прямоугольников,*

Ньютона-Котеса и Гаусса с четыръмя узлами):

$$R_{rect} \leq \frac{1}{24N^2}||f''||(b-a)^3, \tag{3}$$

$$R_{NC} \leq \frac{1}{480N^4}||f^{(IV)}||(b-a)^5, \tag{4}$$

$$R_{Gauss} \leq \frac{1}{16 * 9!N^4}||f^{(8)}||(b-a)^5. \tag{5}$$

Таким образом, для достаточно хороших функций точность вычислений должна быть порядка $O(N^{-2})$ для формулы прямоугольников и $O(N^{-4})$ для двух других. Это мы позже и проверим численно.

6 Двумерная задача

Пусть дана функция $f : [0,1]^2 \to \mathbb{R}$, и мы хотим посчитать интеграл по всему квадрату. Для этого предлоагается вначале разрезать квадрат на треугольнички. Предлагается следующая схема:

- выбрать разбиение для вертикальной и горизонтальной сторон квадрата. Я выбрал равномерное разбиение на n_x отрезков по горизонтали и n_y - по вертикали;
- разрезать квадрат сообразно выбранному разбиению на сетку;
- провести диагонали в каждой клетке решётки;
- составить списки граней полученного графа (кроме внешней по отношению к квадрату), а так же список треугольников, инцидентных каждому ребру.

Таким образом, получаем:

- $(n_x + 1)(n_y + 1)$ dершин графа;
- $(n_y + 1)n_x$ горизонтальных рёбер;
- $(n_x + 1)n_y$ вертикальных рёбер;
- $n_y n_x$ диагональных рёбер;
- $2n_y n_x$ треугольных граней.

Далее мы считаем интеграл по каждому треугольнику отдельно и складываем полученные результаты. Предлагается формула, указанная ниже.

Теорема 6.1. *Квадратурная формула для треугольника ABC:*

$$S(f) = \frac{\mu(ABC)}{3}(f(M) + f(N) + f(P)), \tag{6}$$

где M, N, P - середины сторон треугольника, точна для многочленов степени не выше 1.

В ходе численных экспермиентов попробуем оценить погрешность вычислений, а так же зависимость её от разбиения квадрата.

7 Программа. Общая схема

Чтобы сделать код безопасным, эффективным и интегрируемым в комплексные решения, использовались методы ООП, имеющиеся в стандарте C++11. Программа для выполнения задач условно делится на две части: одномерную и двумерную. Для избежания ошибок, обе части инкапсулированы в разные пространства имён: quad и qu2d соотвтетственно.

Основой одномерной программы является класс qu_formula, который содержит в себе полную информацию об узлах и коэффициентах квадрвтурной формулы на отрезке $[-1, 1]$. Составлены отдельные конструкторы формул прямоугольников, Ньютона-Котеса и Гаусса, не считая тех, что рекомендуются стандартом C++11. Информация инкапсулирована и не может быть изменена в ходе выполнения программы, что гарантирует безопасность от трудно отлаживаемых ошибках. Функция расчёта простой квадратуры по заданному отрезку и функции передвигает узлы на требуемый отрезок. Для вычисления весовых коэффициентов была реализована функция для решения систем линейных уравнений.

В двумерной задаче, более сложной, основой является класс partition. Он содержит в себе информацию о

- вершинах (координаты),
- рёбрах (номера двух крайних вершин),
- гранях (номера вершин и рёбер),
- инцидентности рёбер и граней (номера граней для каждого ребра).

Помимо шести стандартных функций-членов (три конструктора, деструктор и два оператора присваивания) также имеются конструктор разбиения заданного прямоугольника, функция сохранения в файл, а ещё и конструктор, загружающий данные из файла.

Поддерживаются грани как с равной площадью, заданной заранее, так и с разной площадью (тогда используется каждый раз формула Герона).

Все данные опять инкапсулированы, и доступ к ним из вне не может быть получен.

Основная функция-член integrate производит интегрирование функции от двух вещественных переменных по квадарту: она перебирает все треугольники и для каждого треугольника считает квадратурную формулу.

Подробная информация содержится в заголовочном файле программы.

```
#pragma once
```

```
#include<iostream>
#include<fstream>
#include<cstdlib>
#include<cmath>
#include<exception>
#include<string>
#include<algorithm>
#include<utility>
#include<cstring>

namespace quad {

        // this thing stores all th coefficients
        //class structure prevents any changes during calculations
        //
        //NB this class stores data for the segment [-1, 1]


        class qu_formula
        {
    private:
                size_t n_;
                double *weights_, *points_;
    public:
                qu_formula();
                qu_formula(size_t n, double *w, double *p) : n_(n),
                   weights_(w), points_(p) {};
        qu_formula(const qu_formula &q); //copy constructor
        qu_formula(qu_formula &&q); //move constructor
        const qu_formula& operator=(const qu_formula &q);
        const qu_formula& operator=(qu_formula &&other);
        ~qu_formula();

        int get_n() const {return n_;};
        double get_weight(size_t k) const;
        double get_point(size_t k) const;
        };

        typedef  double(*REAL_FUNC)(double); //the real function

        double simple_quad(REAL_FUNC f, const qu_formula &q, double a,
           double b); //simple quadrature formula
        double comp_quad(REAL_FUNC f, const qu_formula &q, double a,
           double b, int N); //compound quadrature formula

        qu_formula Newton_Kotes(); //making quadratures
        qu_formula Gauss();
        qu_formula Naive();

        double col_max(double *A, int n, int k); //to control errors
```

```
        void swap_rows(double *A, int n, int p, int q);
        void add_rows(double *A, int n, int p, int q, double m);
        void divide_by(double *A, int n, int p, double d);
        void Lin_eq(int n, double *A, double *b);  //Linear system solving
            . The matrix A will be spoiled, the answer will be in the
            vector b

        void find_weights(int n, double *ans, const double *p);

}

namespace qu2d {
    //this namespace is for 2d integrating
    typedef std::pair<double, double> point;
    typedef std::pair<size_t, size_t> edge;


        typedef double(*REAL_FUNC2D)(double, double); //2d real fucnction

        point get_centre(const point &p1, const point &p2); //finds the
            centre of the segmet between the given points


        struct side {
                size_t v1, v2, v3; //vertices
                size_t e1, e2, e3; //edges
        };
        struct neib_side {
                int sides;
                size_t first, second;

                neib_side() { sides = 0; }
                void fit(size_t num);
        };




    class partition { //rectangle partition
    private:
        size_t npoints_; //total amount of points
        size_t nedges_; //amount of edges
        size_t nsides_; //amount of sides
        point* points_; //array of points
        edge* edges_; //array of edges
                neib_side* sides_neib_to_edge_; //which sides are
                    neighbouring to the edge
        side* sides_; //array of sides

                bool equal_square_; //whether the square of all the
                    segments is equal
```

```
                double sqr_; //square of a triangle

                double integrate_over_side(REAL_FUNC2D f, const side &k);
    public:
        partition(); //empty constructor
        partition(size_t nx, size_t ny, point down, point upper); //
            partition of a rectangle
        partition(const std::string &fname); //loader
        ~partition(); //destructor

        partition(const partition &other); //copy constructor
        partition(partition &&other); //move onstructor

        const partition & operator=(const partition &other);
        const partition & operator=(partition &&other);

                const edge & get_edge(size_t k); //get the edge by its
                    number
                const point & get_point(size_t k); //get the point by its
                    number

                double geron_formula(const side &k);
                double edge_length(const edge &k);

                //integrating

                double integrate(REAL_FUNC2D f);


                //saving
                void save(const std::string &fname);
        };

}
```

8 Численные эксперименты

Были проведены численные эксперименты для трёх функций: $f(x) = x^7$, $g(x) = e^x(x^2 - 2x + 0.5)$ и $h(x) = \sin(100.27\pi x)$ на отрезке $[0, 1]$. Брались разбиения на число отрезков, равное степени двойки, от 2 до 10^4. Сравнивались точные значения интегралов (их можно найти аналитически) с численными приближениями. Ниже приведены графики ошибок от величины разбиения в двойном логарифмическом масштабе.

Для монома седьмой степени квадратура Гаусса дала (как и обещалось) точный результат.

Теперь посмотрим на экспоненту.

В квадратуре Гаусса оказывается невыгодно брать слишком мелкое разбиение: всё равно

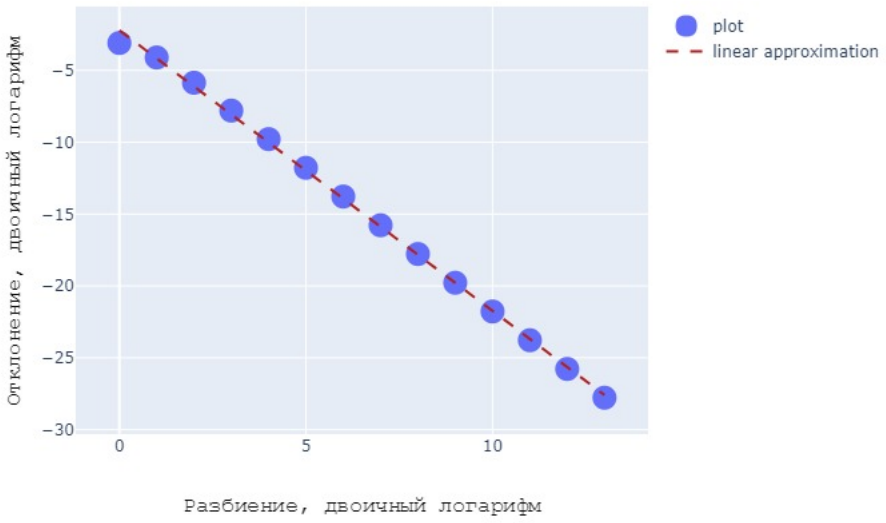


Рис. 1: $f(x) = x^7$. Метод прямоугольников. Угловой коэф-т = -1.95

более малой погрешности, чем определённый порог (около 10^{-15}), нельзя добиться.

Теперь рассмотрим быстро осциллирующую функцию.

Эту функцию тоже получилось сынтегрировать (формула Гаусса здесь себя показала на порядок лучше всех), но при этом скорость сходимости оказалась ниже обещанной.

9 Выводы

Были исследованы методы численного интегрирования в одно- и двумерном случае. Получены экспериментальные зависимости точности от мелкости разбиения. Численно полученные зависимости соответствуют в определённой степени соответствуют теоретическим, но неограниченного падения погрешности на ЭВМ достичь не удаётся.

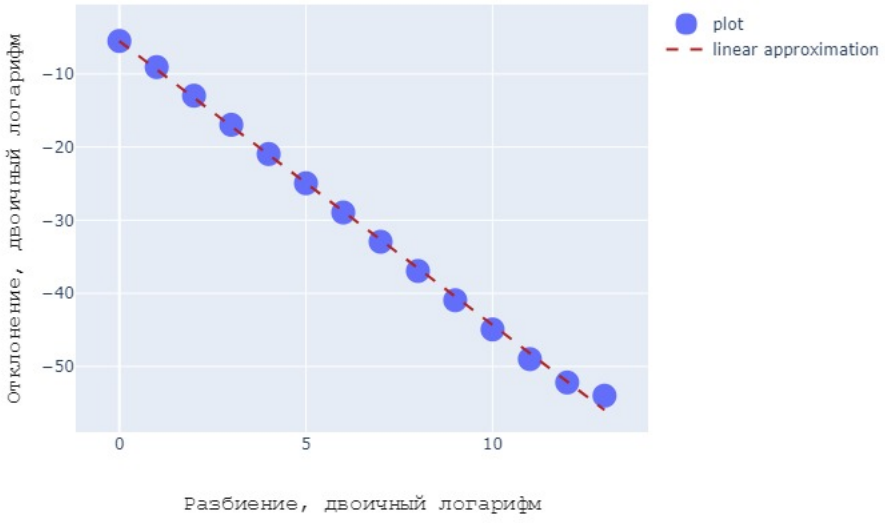


Рис. 2: $f(x) = x^7$. Метод Ньютона-Котеса. Угловой коэф-т = -3.87

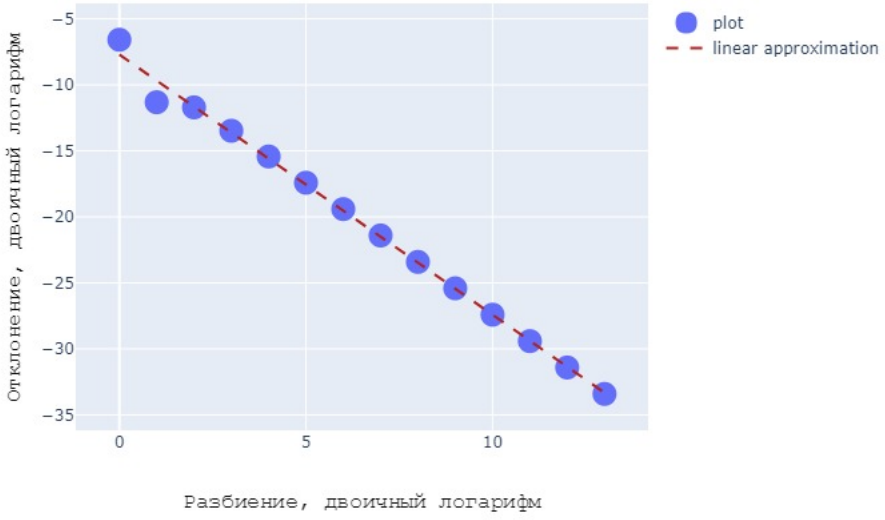


Рис. 3: $g(x) = e^x(x^2 - 2x + 0.5)$. Метод прямоугольников. Угловой коэф-т = -1.97

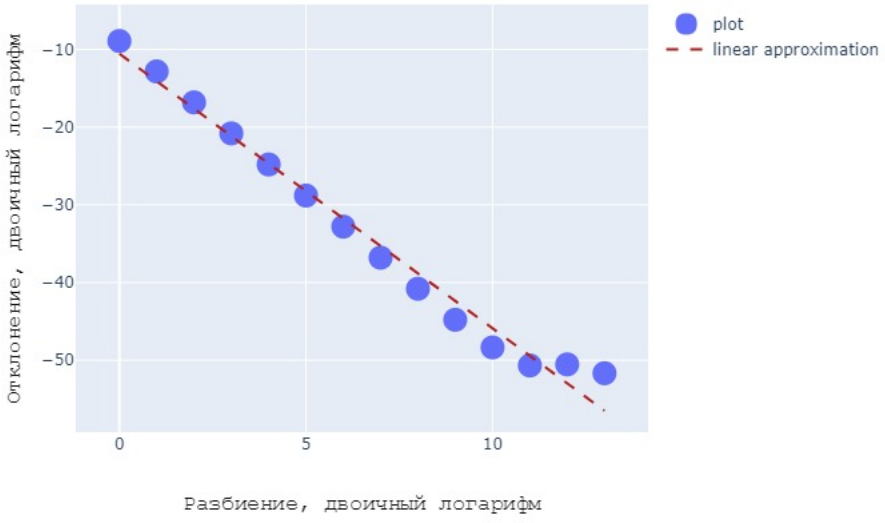


Рис. 4: $g(x) = e^x(x^2 - 2x + 0.5)$. Метод Ньютона-Котеса. Угловой коэф-т = -3.53

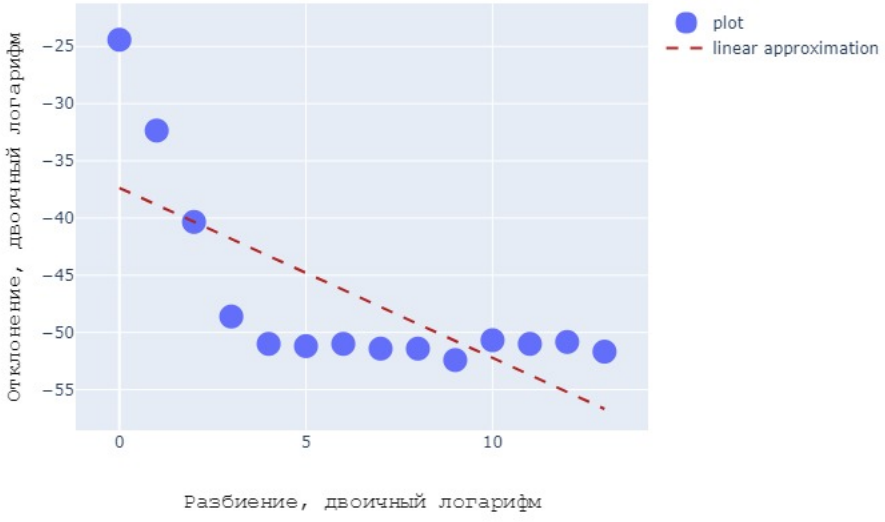


Рис. 5: $g(x) = e^x(x^2 - 2x + 0.5)$. Метод Гаусса. Угловой коэф-т = -3.53

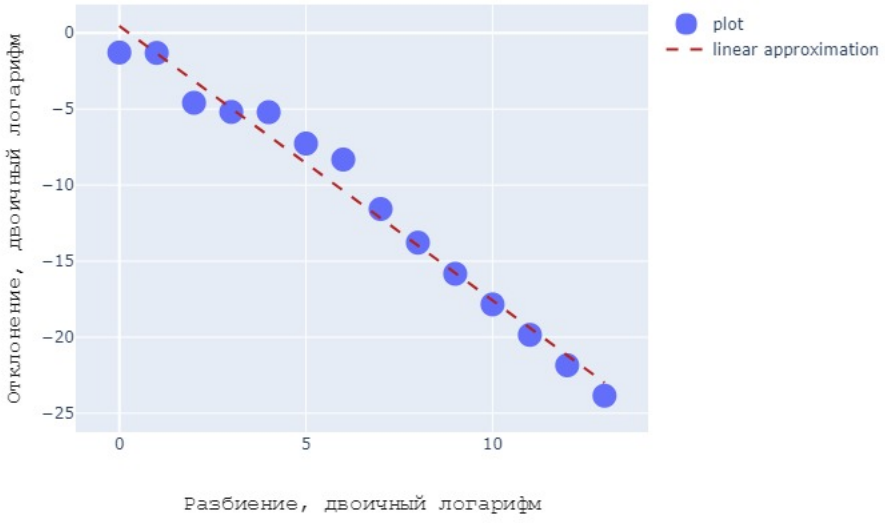


Рис. 6: $h(x) = \sin(100.27\pi x)$. Метод прямоугольников. Угловой коэф-т = -1.80

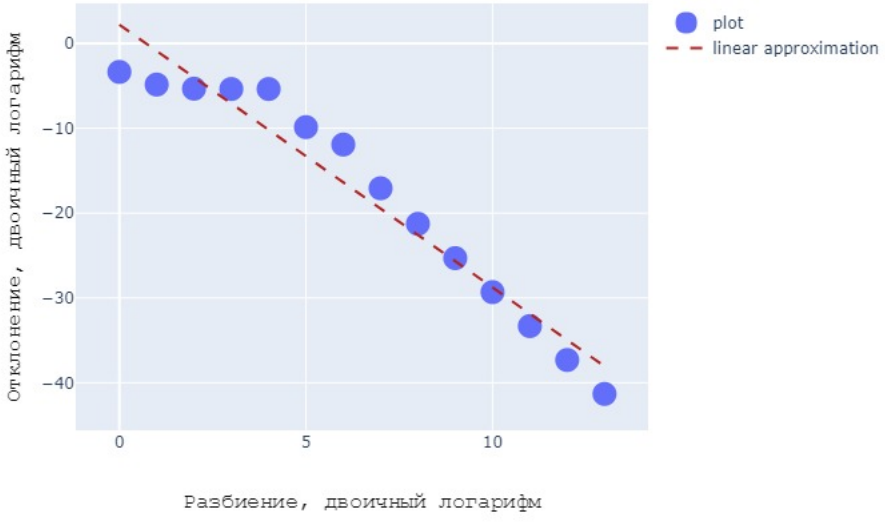


Рис. 7: $h(x) = \sin(100.27\pi x)$. Метод Ньютона-Котеса. Угловой коэф-т = -3.10

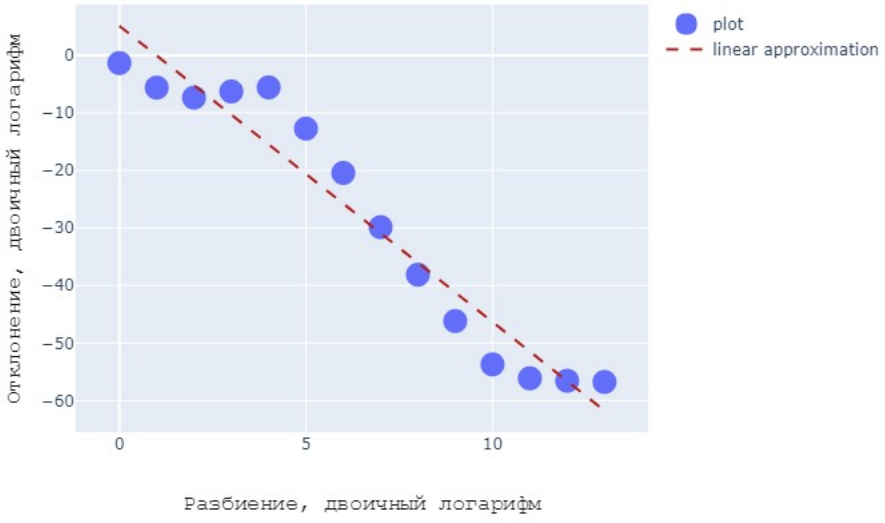


Рис. 8: $h(x) = \sin(100.27\pi x)$. Метод Гаусса. Угловой коэф-т = -3.10

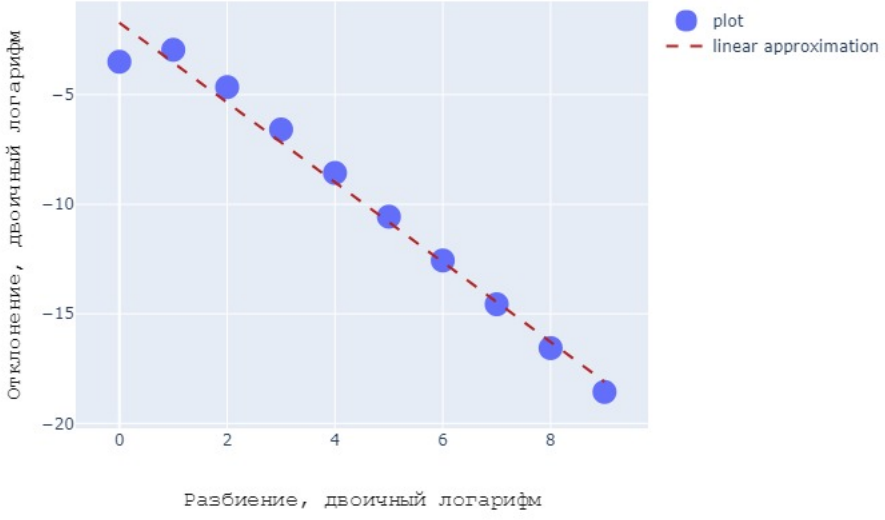


Рис. 9: $h(x, y) = \exp(x + 2y)(2x - y)$. Угловой коэф-т = -1.82

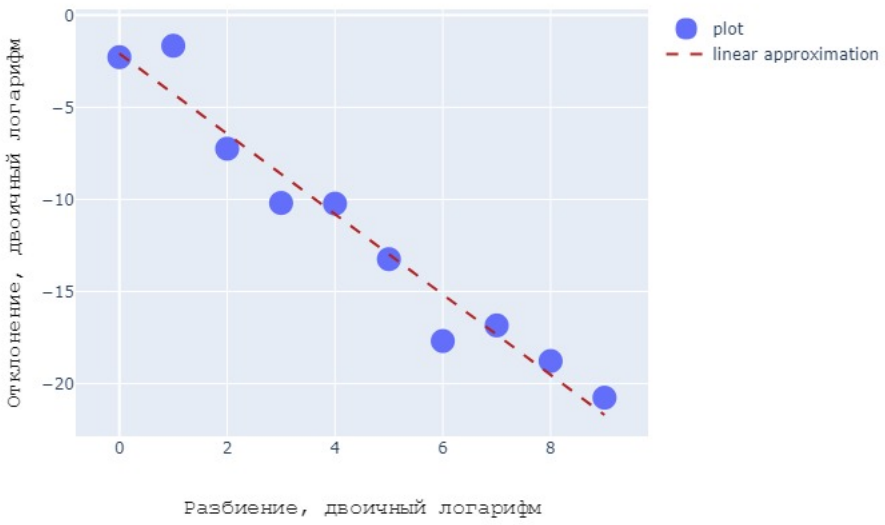


Рис. 10: $h(x) = \sin(100.27\pi(x - 2y))$. Угловой коэф-т = -2.18