УФИМСКИЙ НАУЧНЫЙ ЦЕНТР РАН
ИНСТИТУТ МАТЕМАТИКИ С ВЫЧИСЛИТЕЛЬНЫМ ЦЕНТРОМ

На правах рукописи

Рахматуллин Джангир Ялкинович

ИНТЕГРИРОВАНИЕ ФУНКЦИЙ ПО ВЫПУКЛЫМ ОБЛАСТЯМ РЕШЕТЧАТЫМИ КУБАТУРНЫМИ ФОРМУЛАМИ НА МНОГОПРОЦЕССОРНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ

01.01.07 — вычислительная математика

Диссертация на соискание ученой степени
кандидата физико-математических наук

Научный руководитель
д.ф.-м.н., проф.
Рамазанов М.Д.

Уфа — 2006

Содержание

Введение

§1. Основные положения диссертации

Любая теория аппроксимации заинтересована в максимально эффективном практическом использовании математических алгоритмов, этой теорией построенных и обоснованных. Эта проблема особенно актуальна в наши дни, когда вычислительная техника стремительно морально устаревает, уступая место новой, отличающейся порой не только мощностью компонентов, но и принципиально иной архитектурой.

Дело усложняется тем, что стоящая перед потребителем задача, как правило, имеет не один способ решения (часто в рамках нескольких сильно разнящихся теорий), что порождает неизбежную конкуренцию между имеющимися на данный момент алгоритмами. Они могут значительно различаться по своим характеристикам — количеству требуемых операций, устойчивости, наличием теоретических оценок погрешности вычислений, гибкости к изменению входных параметров и т.п.

Поэтому пользователь, для которого, в конечном счете, и создается теория, применяя ее результаты для решения своей задачи, несет важную функцию естественного индикатора практической пользы того или иного алгоритма.

Однако следует учесть, что любой математический алгоритм является в известной мере абстракцией и сравнение нескольких алгоритмов решения одной задачи, по большому счету, бессмысленно до тех пор, пока каждый из них не найдет свое воплощение в конкретной *реализации* — программе, написанной на каком-либо языке программирования под некий класс вычислительных систем. Только тогда можно будет оценить характеристики решения задачи, имеющие непосредственную важность для пользователя, — максимальную точность вычислений, время, необходимое для достижения заданной точности, системные требования (необходимая оперативная память, частота процессора компьютера и т.п.), скорость и эффективность вычислений (для многопроцессорных систем). Не нужно забывать о таких важных качествах программы, как настраиваемость, до-

кументированность, а также трудоемкость написания (отражающаяся, в конечном счете, на ее цене).

Из сказанного следует, что любой алгоритм, как бы ни был он хорош теоретически, не будет востребован на практике при отсутствии на данный момент его эффективной реализации, максимально использующей возможности современной вычислительной техники. Опыт показывает, что в связи с прогрессом вычислительной техники, алгоритмы, когда-то пользовавшиеся популярности, уступали первенство другим, лучше реализованным (например, на многопроцессорной технике). Следовательно, понятия «хороший» и «плохой» алгоритм весьма относительны, главным критерием истинности знания по прежнему остается практика.

Вот почему так важно квалифицированное отображение математических достижений в «механическую» плоскость.

Объектом исследования данной диссертации является приближенное интегрирование функций по многомерным областям (УДК 519.644 + 517.518.87).

Объект исследования представляет значительный интерес для широкого круга прикладных задач атомной физики, квантовой химии, статистической механики, может использоваться для решения интегральных уравнений, задач математической физики и т.д.

На сегодняшний день задача приближенного интегрирования по многомерным областям решается в рамках нескольких теорий *кубатурных формул.*

Кубатурной формулой (КФ) $K_N^\Omega(f)$ назовем линейный функционал, задающий приближенное значение интеграла $I^\Omega(f) \stackrel{def}{=} \int\limits_\Omega dx f(x)$ в виде линейной комбинации значений функции $f \in \widetilde{W}_p^m(\Omega)$ в N произвольных точках — узлах кубатурной формулы:

$$K_N^\Omega(f) \stackrel{def}{=} \sum_{k=1}^{N} c_{k,N} f(x^{(k)}), \quad \left\{x^{(k)}\right\}_{k=\overline{1,N}} \in \mathbb{R}^n.$$

Перечислим основные методы построения кубатурных формул.

- Вероятностные методы типа Монте-Карло.
- Методы построения формул высокого алгебраического порядка (точных на многочленах большой степени) и инвариантных относительно некоторой группы изометрических преобразований.

- Функциональные методы построения решетчатых кубатурных формул.

- Теоретико-числовые методы построения формул, точных для конечных тригонометрических рядов.

Отметим, что несмотря на большие теоретические достижения во всех четырех направлениях, на практике для интегралов большой кратности используются, в основном, методы интегрирования типа Монте-Карло. Они обладают несомненным преимуществом — порядок сходимости построенных такими методами кубатурных формул не зависит от размерности пространства, в котором задана область интегрирования и от формы этой области.

Слабыми сторонами вероятностного подхода, однако, являются

- малый порядок сходимости кубатурных формул

- вероятностные оценки точности формул

- «подстроение» класса интегрантов под свойства решений интегральных уравнений, а также итерационный метод их решения

Существуют также программы, реализующие методы решетчатых кубатурных формул, созданные в разные годы Н. И. Блиновым [3, 2], Л. В. Войтишек [10], И. Умархановым [45], а также на основе методов теоретико-числового анализа, созданные Н. М. Коробовым [14]. Их основные недостатки:

- малая доступная размерность пространства интегрирования (не больше трех)

- малая точность вычислений

- реализация на устаревших языках программирования и, следовательно, плохая переносимость программ

- реализация на устаревших вычислительных платформах, а значит, отсутствие возможности реального применения на практике

Обозначив, таким образом, имеющиеся на данный момент *недостатки* в решении задачи приближенного интегрирования, обозначим *проблему*:

необходима программная реализация какого-либо из развитых на данный момент теоретических методов, преодолевающая указанные недостатки.

Актуальность проблемы обусловлена следующими факторами:

- большой прикладной значимостью приближенного вычисления интегралов по областям больших размерностей
- потребностью в реализации алгоритмов, имеющих гарантированные оценки точности результатов вычислений и высокую скорость сходимости
- необходимостью эффективного использования современной вычислительной техники
- потребностью в стандартных прикладных программах, вычисляющих интегралы с достаточной точностью за разумное время

Для решения указанной выше проблемы определим *предмет исследований* — решетчатые кубатурные формулы с ограниченным пограничным слоем.

Основными *целями* данной диссертации являются:

- решение задачи численного интегрирования функций по многомерным областям. В качестве теоретической базы используется аппарат решетчатых кубатурных формул с ограниченным пограничным слоем. Посредником, обеспечивающим преобразование математического алгоритма в машинный, является язык программирования C++ [25] с библиотекой параллельных функций MPI [36]. Конечным результатом является стандартная параллельная программа, предназначенная для использования в суперкомпьютерах с MPP (Massively Parallel Processing - массовый параллелизм) архитектурой — многопроцессорных системах с распределенной памятью.
- теоретическое исследование решеток узлов, дающих наилучший порядок приближения для оптимальных кубатурных формул в неизотропном пространстве $W_2^{\overline{m}}(\mathbb{R}^2)$

В диссертационной работе мы используем следующие *методы исследования*:

- теоретические методы функционального анализа

- теоретические методы вычислительной математики
- экспериментальные методы тестирования программ на многопроцессорных вычислительных системах

Перечислим результаты, *выносимые на защиту* (попутно отмечая их *новизну*).

- *Реализован алгоритм* (в виде параллельной программы) нахождения коэффициентов кубатурной формулы с ограниченным пограничным слоем, имеющей универсальные асимптотические свойства на классе пространств. Реализация алгоритма превосходит имеющиеся аналоги по быстродействию и эффективности.
- С помощью программы *решена* поставленная *проблема*. Впервые была решена задача интегрирования по многомерным областям с гладкими границами с подобными показателями точности, быстродействия и реально доступной размерности пространства.
- Создана *параллельная* программа, эффективно использующая возможности передовых технологий (суперкомпьютеров) и языков программирования (C++ и MPI).
- Произведено *тестирование* программы и *анализ* ее основных показателей (скорости, эффективности, точности) при изменении нескольких параметров (размерности, гладкости, количества точек, входных функций и формы области) и даны рекомендации по ее использованию.
- Впервые была произведена модификация существующего алгоритма, оптимизирующая его программную реализацию.
- *Найден новый способ* построения решеток узлов в пространстве $W_2^{\overline{m}}(\mathbb{R}^2)$, на которых оптимальные формулы имеют наилучший порядок приближения.
- Проведено теоретическое сравнение нового способа с уже существующими и сделаны предположения об их вычислительных свойствах. Обнаружено новое явление — свойство функций дискретных аргументов на изотропных решетках хорошо приближать функции из неизотропных классов.

- Экспериментально исследованы вычислительные свойства предложенного способа, сделаны выводы.

Основные результаты диссертанта отражены в работах [29, 34, 35, 37, 38].

§2. Краткая история вопроса

Одномерный вариант задачи приближенного интегрирования относится к разряду классических и имеет историю в несколько веков. Формулы, приближающие определенные интегралы функций одной переменной называются *квадратурными*. Начиная с работ академика С. М. Никольского [18] в теории квадратурных формул начали применяться функциональные методы. Проблема приближения интеграла функции одной переменной рассматривалась в работах В. И. Крылова [15], Н. П. Корнейчука [12], А. Сарда [49], А. Страуда [50], Н. С. Бахвалова [1], В. И. Половинкина [27], В. Л. Васкевича [4, 5] и других авторов.

Формулы, приближающие интегралы по многомерным областям, называются *кубатурными*. Существенным их отличием от квадратурных формул является то, что многомерные области могут произвольную, часто весьма сложной формы, границу.

Теория кубатурных формул появилась в 60-х годах двадцатого века в связи с исследованиями академика С. Л. Соболева (см. [40, 43]) и имела продолжение в трудах его учеников и последователей.

На данный момент можно выделить несколько направлений в теории кубатурных формул.

Во-первых, это методы построения формул *высокого алгебраического порядка* (точных на многочленах большой степени) и инвариантных относительно некоторой группы изометрических преобразований. Отражение исследований в данном направлении можно найти в трудах С. Л. Соболева [41, 42], В. И. Лебедева [16], И. П. Мысовских [17], М. В. Носкова [19, 20], Н. Н. Осипова [19, 21], Г. Н. Салихова [39] и других авторов.

Во-вторых, это *теоретико-числовые* методы построения формул, точных для конечных тригонометрических рядов, заложенные в трудах И. М. Виноградова [8] и продолженные в работах Н. М. Коробова [13, 14] и Н. Н. Осипова [22, 23, 24].

Широко известны и наиболее часто применяются *вероятностные* методы типа Монте-Карло построения кубатурных формул. Основным достоинством этих методов является независимость порядка сходимости кубатурной формулы от размерности пространства, недостатками — маленькая скорость сходимости (порядка $N^{-\frac{1}{2}}$, где N — количество узлов КФ), а также ее независимость от гладкости подынтегральной функции. Исследования по этому направлению можно найти, например, в работах И. М. Соболя [44] и А. В. Войтишека [9].

Наконец, четвертым направлением является развитие функциональных методов построения решетчатых кубатурных формул. Они были предложены С. Л. Соболевым [40, 43] и развиты в работах В. Л. Васкевича [6, 7], В. И. Половинкина [26, 28], М. Д. Рамазанова [30, 31, 32, 33, 48], Ц. Б. Шойнжурова [46, 47] и других авторов.

Узлы решетчатых кубатурных формул задаются на какой-либо последовательности параллелепипедальных решеток, сгущающихся при стремлении к нулю малого параметра h. О специфике функционального подхода в теории кубатурных формул В. Л. Васкевич пишет следующее [7]:

«Функциональный подход к построению и исследованию формул для приближения многомерных интегралов предполагает, во-первых, использование выбранной или построенной формулы не столько для какой-то одной конкретной функции, сколько сразу для целого их семейства, представляющего собой шар в некотором наперед заданном банаховом пространстве X.

Во-вторых, разность между интегралом и приближающей его конечной комбинацией значений подынтегрального элемента рассматривается как результат действия на подынтегральную функцию некоторой обобщенной функции, полностью определяемой исходной кубатурной формулой и называемой по этой причине ее функционалом погрешности.

В-третьих, исходное банахово пространство X предполагается вложенным непрерывно в пространство $C(\Omega)$ функций, непрерывных в замыкании области интегрирования Ω. В этом случае функционал погрешности кубатурной формулы не только линеен, но и ограничен на X, причем знание численной мажоранты для его нормы в сопряженном пространстве X^* позволяет получать для произвольной функции из единичной сферы пространства X гарантированные оценки близости истинного значения интеграла от этой функции к рассматриваемой на ней кубатурной сумме.»

В данной диссертации реализуется, в основном, функциональный подход к построению кубатурных формул.

В первом разделе приведен алгоритм и описание программы интегрирования функций по многомерным выпуклым областям с гладкими границами. Теоретическое обоснование алгоритма и программы дается функциональными методами.

Во втором разделе диссертации функциональные и теоретико-числовые методы используются для доказательства теоремы о достижении наилучшего порядка в неизотропном пространстве $W_2^{m_1,m_2}\left(\mathbb{R}^2\right)$ на последовательностях повернутых кубических решеток.

§3. Список основных обозначений

КФ — кубатурная формула.
ПКФ — последовательность кубатурных формул.
ФП — функционал погрешности.
ПФП — последовательность функционалов погрешности.
ОПС — ограниченный пограничный слой.
$\mathbb{Z}$ — множество целых чисел.
$\mathbb{R}$ — множество вещественных чисел.
$x \stackrel{def}{=} y$ — x по определению равен y
$[x]$ — целая часть числа x.
$\{x\}$ — дробная часть числа x.
$\asymp$ — двусторонняя оценка.
$\int dx\, f(x)$ — интеграл от функции $f(x)$.
(l, f) — действие функционала l на функцию f.
$\langle x, y\rangle$ — скалярное произведение x и y.

Глава I. Численное интегрирование по многомерным областям

§1. Общая постановка проблемы

Пусть задана ограниченная область $\Omega \subset \mathbb{R}^n$ с гладкой границей $\partial\Omega$.

Рассмотрим пространство Соболева $\widetilde{W}_p^m(Q),\ p > 1,\ m > \frac{n}{p}$ периодических функций, интегрируемых с p-ой степенью вместе с производными до m-го порядка включительно, в одной из эквивалентных норм:

$$\|g\|_{\widetilde{W}_p^m(Q)} \stackrel{def}{=} \left(|g_0|^p + \int\limits_Q dx \left| \sum_{k \in \mathbb{Z}^n \setminus \{0\}} g_k \ (1 + |2\pi k|)^{m/2} \, e^{2\pi i x k} \right|^p \right)^{\frac{1}{p}}, \qquad \text{(I.1)}$$

$$g_k \stackrel{def}{=} \int\limits_Q dx \ g(x) e^{-2\pi i x k}, \qquad \text{(I.2)}$$

где Q — единичный гиперкуб: $Q \stackrel{def}{=} \{x :\ x_i \in [0;\, 1),\ i = \overline{1, n}\}$.

Взяв это пространство за основу, построим новое пространство $\widetilde{W}_p^m(\Omega)$ с нормой

$$\|f\|_{\widetilde{W}_p^m(\Omega)} \stackrel{def}{=} \inf_{\substack{g|_\Omega = f|_\Omega, \\ g \in \widetilde{W}_p^m(Q)}} \|g\|_{\widetilde{W}_p^m(Q)}. \qquad \text{(I.3)}$$

Требуется приближенно вычислить интеграл произвольной функции $f \in \widetilde{W}_p^m(\Omega)$ по области Ω.

Мы решаем эту задачу путем приближения интеграла решетчатыми кубатурными формулами с ограниченным пограничным слоем (ОПС-формулами). Поясним эти термины.

Решеткой узлов назовем множество $\{hHk\}_{k \in \mathbb{Z}^n}$, где $h \in \mathbb{R},\ h > 0,\ H$ — матрица $n \times n,\ \det H = 1$. Устремив h к нулю, получим последовательность решеток со сгущающимися узлами. Мы будем рассматривать частный случай — $H \equiv I$, т.е. последовательность решеток $\{hk\}_{k \in \mathbb{Z}^n}$, называемую *прямоугольной (кубической)*. Заметим, что выбор решетки вполне согласуется с изотропностью нашего пространства — функции из него по всем направлениям имеют одну и ту же гладкость m.

Кубатурной формулой (КФ) $K_N^\Omega(f)$ назовем линейный функционал, задающий приближенное значение интеграла $I^\Omega(f) \stackrel{def}{=} \int\limits_\Omega dx f(x)$ в виде линейной комбинации значений функции $f \in \widetilde{W}_p^m(\Omega)$ в N произвольных точках — узлах кубатурной формулы:

$$K_N^\Omega(f) \stackrel{def}{=} \sum_{k=1}^{N} c_{k,N} f(x^{(k)}), \quad \left\{x^{(k)}\right\}_{k=\overline{1,N}} \in \mathbb{R}^n. \tag{I.4}$$

(мы можем вычислять значения $f(x^{(k)})$ благодаря условию $m > \frac{n}{p}$)

КФ называется *решетчатой*, если ее узлы лежат на решетке узлов.

Мы будем рассматривать последовательности решетчатых кубатурных формул $\left\{K_h^\Omega(f)\right\}$, соответствующие последовательностям решеток со сгущающимися узлами $\{hk\}_{k\in\mathbb{Z}^n},\ h \to 0$:

$$K_h^\Omega(f) = h^n \sum_{hk\in\Omega} c_k(h) f(hk), \quad h \to 0. \tag{I.5}$$

Здесь за знак суммы вынесен масштабный множитель h^n, равный объему элементарной (наименьшей) ячейки, образуемой узлами решетки.

Заметим, что члены последовательности кубатурных формул (ПКФ) обычно задаются единым правилом, поэтому ПКФ можно считать последовательностью значений единой кубатурной формулы — функции аргумента h. Сделав данное замечание, впредь будем опускать слово «последовательность», указывая вместо этого, что параметр h кубатурной формулы стремится к нулю (по одной из последовательностей) и считая $c_k(h)$ заданными как функции от k и h.

Будем говорить, что КФ обладает *ограниченным пограничным слоем* (ОПС), если выполнены следующие два условия [33] :

1) коэффициенты КФ равномерно ограничены по k и h :

$$\exists\ L_1 : \quad \sup_{k,\ h} |c_k(h)| < L_1\ ; \tag{I.6}$$

2)

$$\exists\ L_2 : \quad \forall h,\ k : \rho(hk, \mathbb{R}^n \backslash \Omega) \geqslant L_2 h \Rightarrow c_k(h) = 1, \tag{I.7}$$

т.е. при любом фиксированном h все коэффициенты, соответствующие узлам решетки, расположенным внутри области Ω достаточно глубоко — на расстоянии, не меньшем $L_2 h$ от границы, равны единице.

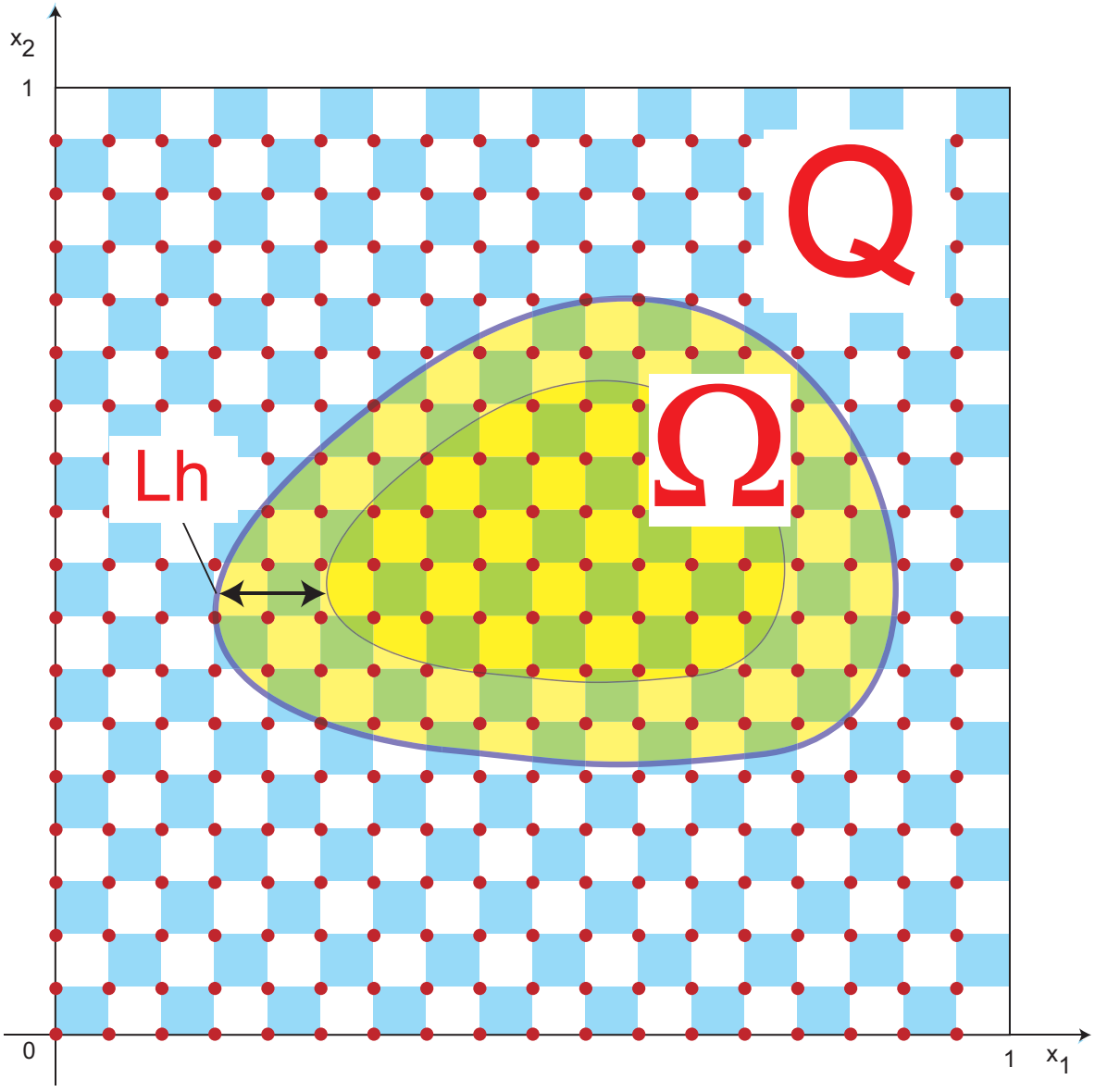

Рис. I.1. Пример решетки узлов и области Ω с ОПС

Отметим, что выбор решетчатых КФ позволяет достичь необходимой универсальности и единообразия в интегрировании по областям произвольных форм вследствие того, что последовательности решеток не зависят от форм областей интегрирования. Использование же КФ с ОПС позволяет, не сильно сужая выбор КФ, использовать алгоритмы вычисления коэффициентов формул с хорошими и теоретически обоснованными аппроксимационными свойствами.

Для оценки качества КФ введем понятие функционала погрешности (ФП).

Функционалом погрешности $l_h^\Omega(f)$, соответствующим КФ $K_h^\Omega(f)$, называется линейный функционал, представляющий собой разность точного значения интеграла $I^\Omega(f)$ функции по области Ω и кубатурной формулы $K_h^\Omega(f)$:

$$l_h^\Omega(f) \stackrel{def}{=} I^\Omega(f) - K_h^\Omega(f) \tag{I.8}$$

или в виде обобщенной функции:

$$l_h^\Omega(x) = \chi_\Omega(x) - \sum_{\substack{hk\in\Omega,\\ k\in\mathbb{Z}^n}} c_k(h)\delta(x-hk), \tag{I.9}$$

где $\chi_\Omega : \ f \mapsto \int dx \chi_\Omega(x) f(x) \equiv \int\limits_\Omega dx f(x)$.

Мы также будем опускать слово «последовательность» для последовательностей функционалов погрешности, соответствующих ПКФ, отмечая лишь стремление h к нулю.

В качестве численного показателя точности приближения интеграла кубатурной формулой берется норма соответствующего ФП в пространстве, сопряженном пространству подынтегральных функций:

$$\|l_h^\Omega\|_{(\widetilde{W}_p^m(\Omega))^*}. \tag{I.10}$$

Желание использовать лучшие для данной последовательности решеток кубатурные формулы приводит нас к понятию оптимальной КФ, а также асимптотически оптимальной и оптимальной по порядку КФ.

КФ, минимизирующая при любом фиксированном h нормы элементов соответствующего ФП, называется *оптимальной*:

$$K_h^{\Omega,opt} \stackrel{def}{=} \arg \min_{K_h^\Omega} \left\| I^\Omega - K_h^\Omega \right\|_{(\widetilde{W}_p^m(\Omega))^*} \equiv \arg \min_{K_h^\Omega} \left\| l_h^\Omega \right\|_{(\widetilde{W}_p^m(\Omega))^*}. \tag{I.11}$$

На практике оптимальные КФ трудновычислимы, поэтому часто ограничиваются рассмотрением КФ, имеющим оптимальные свойства лишь в пределе, при $h \to 0$.

КФ $K_h^{\Omega,as}$ называется *асимптотически оптимальной*, если выполняется равенство:

$$\lim_{h\to 0} \frac{\|I^\Omega - K_h^{\Omega,as}\|_{(\widetilde{W}_p^m(\Omega))^*}}{\|I^\Omega - K_h^{\Omega,opt}\|_{(\widetilde{W}_p^m(\Omega))^*}} = 1. \tag{I.12}$$

КФ $K_h^{\Omega,ord}$ называется *оптимальной по порядку*, если

$$\exists\, C: \ \overline{\lim_{h\to 0}} \frac{\|I^\Omega - K_h^{\Omega,ord}\|_{(\widetilde{W}_p^m(\Omega))^*}}{\|I^\Omega - K_h^{\Omega,opt}\|_{(\widetilde{W}_p^m(\Omega))^*}} \leqslant C. \tag{I.13}$$

§2. Алгоритм

Зададимся произвольным целым числом M, таким, что $M > \frac{n}{p}$. Ниже мы приведем алгоритм вычисления КФ, оптимальной по порядку на каждом из пространств $\widetilde{W}_p^m(\Omega)$ с $m = \left(\frac{n}{p}, M\right)$, т.е. *универсально оптимальной по порядку* на этом классе пространств (приведенный алгоритм является улучшенной версией алгоритма, изложенного в [40, стр.254–263]).

Учитывая оценки [40, стр.138, 152])

$$\exists\, C_1: \quad \left\|l_h^\Omega\right\|_{(\widetilde{W}_p^m(\Omega))^*} \geqslant C_1 \|l_\infty\|_{(\widetilde{W}_p^m(\Omega))^*} \geqslant C_1 h^m \tag{I.14}$$

для оптимальности по порядку достаточно оценки сверху:

$$\exists\, C_2: \quad \|l_h^\Omega\|_{(\widetilde{W}_p^m(\Omega))^*} \leqslant C_2 h^m, \quad h \to 0. \tag{I.15}$$

По следующей теореме универсально оптимальные по порядку КФ с ОПС являются *универсально асимптотически оптимальными.*

Теорема (М. Д. Рамазанов). *КФ с ОПС асимптотически оптимальна на каждом пространстве из множества*

$$\left\{\widetilde{W}_p^m(\Omega)\right\}_{\substack{m\in(m_1,m_2),\\ p\in(p_1,p_2)}} \tag{I.16}$$

тогда и только тогда, когда она оптимальна на каждом из них по порядку, при $\frac{n}{p} < m_1 < m_2 < M,\ 1 < p_1 < p_2 < \infty$. *[31]*

Отметим, что теорема верна в нашей нормировке (I.3), но может быть неверна при использовании других норм. Например, формулы Соболева при $p \neq 2$ не являются асимптотически оптимальными в W_p^m с классической нормой

$$\|f\|_{W_p^m} \stackrel{def}{=} \left(\int dx \sum_{|\alpha| \leqslant m} |D^\alpha f(x)|^p \right)^{\frac{1}{p}} \tag{I.17}$$

(см. [28]).

Итак, построим КФ с нужными свойствами.

В технических целях наложим дополнительные ограничения на условия задачи.

Пусть Ω — выпуклая область, лежащая внутри единичного гиперкуба вместе с замыканием: $\overline{\Omega} \subset \text{int } Q$, а параметр h стремится к нулю по одной из последовательностей, каждый член которой можно представить в виде N^{-1}, где $N \in \mathbb{N}$.

Алгоритм основан на сведении задачи интегрирования по области с гладкими границами к нескольким однотипным задачам интегрирования по единичному гиперкубу.

Рассмотрим вначале способ интегрирования функции $f \in \widetilde{W}_p^m(Q)$ по единичному гиперкубу Q. Он основан на приближении интеграла кубатурной формулой, получаемой суммированием КФ более простого вида. Каждая из них приближает интеграл, взятый по одной из ячеек решетки и является точной на многочленах до степени M включительно.

Нам понадобится понятие *элементарного функционала*. Его одномерный вариант в виде обобщенной функции записывается следующим образом:

$$\lambda(x) \stackrel{def}{=} \chi_{[0,1]}(x) - \sum_{s \in S} a_s \delta(x - s), \tag{I.18}$$

где S — конечное множество целочисленных индексов, например, $\{i\}_{i=1}^{M+1}$. Говорят, что элементарный функционал имеет порядок M, если он точен на всех одночленах до степени M включительно:

$$(\lambda(x), x^\alpha) \equiv 0, \ \alpha = \overline{0, M} \tag{I.19}$$

или

$$\sum_{s \in S} a_s s^\alpha = \frac{1}{\alpha + 1}, \ \alpha = \overline{0, M}. \tag{I.20}$$

Для этого, очевидно, достаточно, чтобы множество S имело $M+1$ элементов.

На рисунке I.2 изображена область определения элементарного функционала для $M = 5$. Толстый отрезок обозначает носитель характеристической функции, жирные точки — носители дельта-функций.

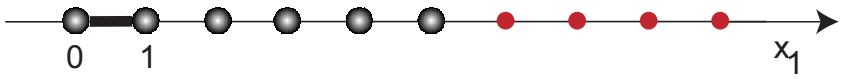

Рис. I.2. Область определения функции $\lambda(x)$ $(M = 5)$

То же обозначение будем использовать и для многомерного варианта элементарного функционала:

$$\lambda(x) \stackrel{def}{=} \chi_Q(x) - \sum_{s \in S} a_s \delta(x - s). \tag{I.21}$$

Здесь $x \in \mathbb{R}^n$, $S \subset \mathbb{Z}^n$, S — конечное множество. Будем говорить, что элементарный функционал имеет порядок M, если

$$(\lambda(x), x^\alpha) \equiv 0, \quad |\alpha| \leqslant M, \tag{I.22}$$

где α — мультииндекс.

Обычно множество S берут с избытком коэффициентов, представляя его как декартово произведение n одномерных множеств S_i, каждое из которых является множеством индексов одномерного элементарного функционала $\lambda_i(x)$ порядка M с набором коэффициентов $\{a_j^{(i)}\}_{j \in S_i}$. В таком случае, если каждому вектору индексов $s \stackrel{def}{=} (s_1, \ldots, s_n)$, $s_i \in S_i$, $i = \overline{1,n}$ сопоставить коэффициент, получаемый как $a_s \stackrel{def}{=} a_{s_1}^{(1)} \cdot \ldots \cdot a_{s_n}^{(n)}$, то полученный многомерный элементарный функционал будет иметь порядок M, имея при этом $(M+1)^n$ ненулевых коэффициентов (см. [31, стр.122]).

Мы будем рассматривать следующий частный случай:

$$S \stackrel{def}{=} S_1 \times \ldots \times S_n, \quad S_1 \equiv \ldots \equiv S_n \stackrel{def}{=} \{1, \ldots, M+1\}. \tag{I.23}$$

На рисунке I.3 серым цветом выделена область определения такого элементарного функционала ($n = 2$, $M = 5$). Серый квадрат соответствует характеристической функции, а крупные серые точки — дельта-функциям.

В нашем случае достаточно найти коэффициенты $a_1, \ldots, a_n$ одномерного элементарного функционала, чтобы затем любой векторный коэффициент a_s записать через их произведение.

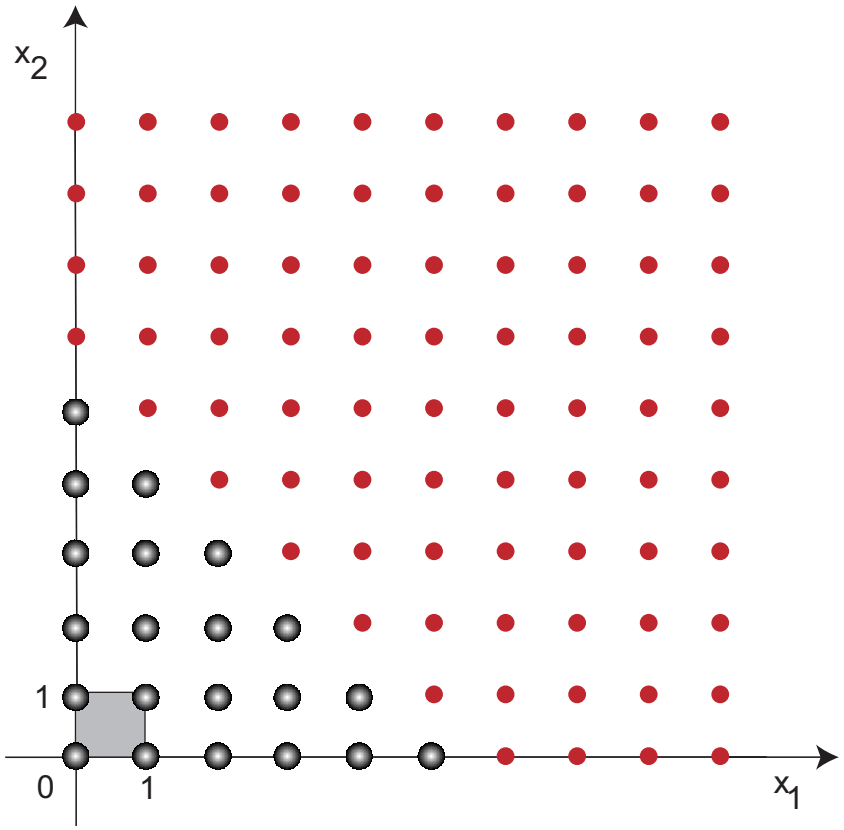

Рис. I.3. Область определения функции $\lambda(x)$ $(n = 2,\ M = 5)$

Для этого необходимо решить систему (I.20), матрица которой имеет определитель Вандермонда:

$$\begin{pmatrix} 1^0 & 2^0 & \dots & (M+1)^0 \\ 1^1 & 2^1 & \dots & (M+1)^1 \\ \dots & \dots & \dots & \dots \\ 1^M & 2^M & \dots & (M+1)^M \end{pmatrix}. \tag{I.24}$$

Обозначив элементы *обратной* к ней матрицы за $\{v_{ij}\}_{ij}^{M+1}$, получим:

$$a_r = \sum_{p=1}^{M+1} v_{rp} \frac{1}{p}, \tag{I.25}$$

откуда

$$a_s = a_{s_1} \cdot \ldots \cdot a_{s_n} = \left(\sum_{p=1}^{M+1} v_{s_1 p} \frac{1}{p} \right) \cdot \ldots \cdot \left(\sum_{p=1}^{M+1} v_{s_n p} \frac{1}{p} \right). \tag{I.26}$$

Заменой переменных произведем сжатие носителя элементарного функционала:

$$\lambda\left(\frac{x}{h}\right) = \chi_{hQ}(x) - h^n \sum_{s \in S} a_s \delta(x - hs) \tag{I.27}$$

Этот функционал приближает интеграл по ячейке

$$hQ \stackrel{def}{=} \{x : 0 \leqslant x_i < h,\ i = \overline{1,n}\} \tag{I.28}$$

линейной комбинацией значений функции в $(M+1)^n$ узлах решетки. Так как множество всех многочленов до степени M переходит само в себя при преобразованиях сжатия и сдвига аргумента, функционал $\lambda\left(\frac{x}{h}\right)$ точен на том же классе одночленов, что и $\lambda(x)$.

Следовательно, если учесть, что функционал $\lambda\left(\frac{x}{h} - k\right)$ дает погрешность приближения интеграла по ячейке с вершиной в точке hk, функционал

$$q_h^{Q_h}(x) \stackrel{def}{=} \sum_{hk \in Q_h} \lambda\left(\frac{x}{h} - k\right), \tag{I.29}$$

где Q_h — некая область, составленная из ячеек решетки при фиксированном h, очевидно, представляет собой разность характеристической функции с носителем Q_h и линейной комбинации $\delta-$функций в «облаке» узлов, «размытом» в ту или иную сторону, в зависимости от конструкции элементарных функционалов. Этот ФП называется *однородным* функционалом погрешности [40, стр. 143].

ФП $q_h^{Q_h}(x)$ является оптимальным по порядку на периодических функциях из пространства $\widetilde{W}_p^m(Q)$ и имеет порядок $O(h^m)$ (см. [31, стр.131–133]):

$$\|q_h^{Q_h}\|_{(\widetilde{W}_p^m(Q))^*} \asymp O(h^m), \tag{I.30}$$

где символ $\asymp$ обозначает двустороннюю оценку. Заметим, что если за область определения ФП $q_h^{Q_h}(x)$ принять гиперкуб Q, то полученный ФП $q_h^{Q}(x)$ за счет периодичности совпадает с *однородным* ФП $l_\infty(x)$ (см. [40, стр.132]), все коэффициенты которого равны единице.

Случай, когда Ω — произвольная область с гладкой границей $\partial\Omega$, вложенная в единичный гиперкуб Q, сводится к разобранному случаю следующим образом.

Для нашей области всегда можно подобрать специальное разбиение единицы — конечный набор функций (будем называть их *срезывающими*), удовлетворяющий условиям:

1) они являются периодическими с основным периодом Q и в сумме составляют функцию, тождественно равную единице в периодически продолженной с основным периодом Q фиксированной окрестности $\hat{\Omega}$ области Ω, целиком лежащей в гиперкубе Q;

2) они принадлежат пространству $\widetilde{C}^M(Q)$;

3) носитель[1] одной из срезывающих функций лежит полностью внутри области Ω, не пересекаясь с ее границей; пересечение носителя каждой из остальных срезывающих функций с границей $\partial\Omega$ непусто и гладко (класса C^M) и взаимно однозначно проектируется на одну из координатных гиперплоскостей;

Примем количество срезывающих функций за $\mathrm{T}+1$. Обозначим их как

$$\{\varphi_\tau(x)\}_{\tau=0}^{\mathrm{T}}, \qquad \sum_{\tau=0}^{\mathrm{T}} \varphi_\tau(x)|_{x\in\hat{\Omega}} \equiv 1. \tag{I.31}$$

Функцию $\varphi_0(x)$ подберем так, чтобы множество точек E_{φ_0}, где она равна единице, находилось не ближе, чем на расстоянии $\varepsilon_0 < \frac{1}{2}$ от границы области:

$$\rho(E_{\varphi_0}, \mathbb{R}^n\backslash\Omega) \geqslant \varepsilon_0, \quad E_{\varphi_0} \stackrel{def}{=} \{x:\ \varphi_0(x) \equiv 1\}. \tag{I.32}$$

Конкретный способ построения функций разбиения единицы приведем ниже. Схематичное изображение носителей срезывающих функций для двумерного случая представлен на рисунке I.4.

Введем обозначения $\Upsilon_\tau \stackrel{def}{=} \operatorname{supp}\varphi_\tau \cap \overline{\Omega}$ и $\Gamma_\tau \stackrel{def}{=} \operatorname{supp}\varphi_\tau \cap \partial\Omega$.

Интеграл $\int\limits_\Omega dx f(x)$ можно представить в виде суммы интегралов:

$$\int\limits_\Omega dx f(x) \equiv \sum_{\tau=0}^{\mathrm{T}} \int\limits_{\Upsilon_\tau} dx\ \varphi_\tau(x) f(x). \tag{I.33}$$

Для удобства и универсальности программной реализации, область Ω будем задавать неявно — как множество точек, где неотрицательна гладкая функция $\Phi(x)$:

$$\Omega \stackrel{def}{=} \{x:\ \Phi(x) \geqslant 0\}, \quad \Phi(x) \in C^M(Q). \tag{I.34}$$

При этом граница задается как

$$\partial\Omega \stackrel{def}{=} \{x:\ \Phi(x) = 0\}. \tag{I.35}$$

Пусть градиент функции $\Phi(x)$ на границе не обращается в ноль:

$$\nabla\Phi(x)|_{\partial\Omega} \neq 0. \tag{I.36}$$

[1] Для удобства под носителем срезывающей функции будем иметь ввиду лишь одну из ее связных компонент, лежащую в гиперкубе Q.

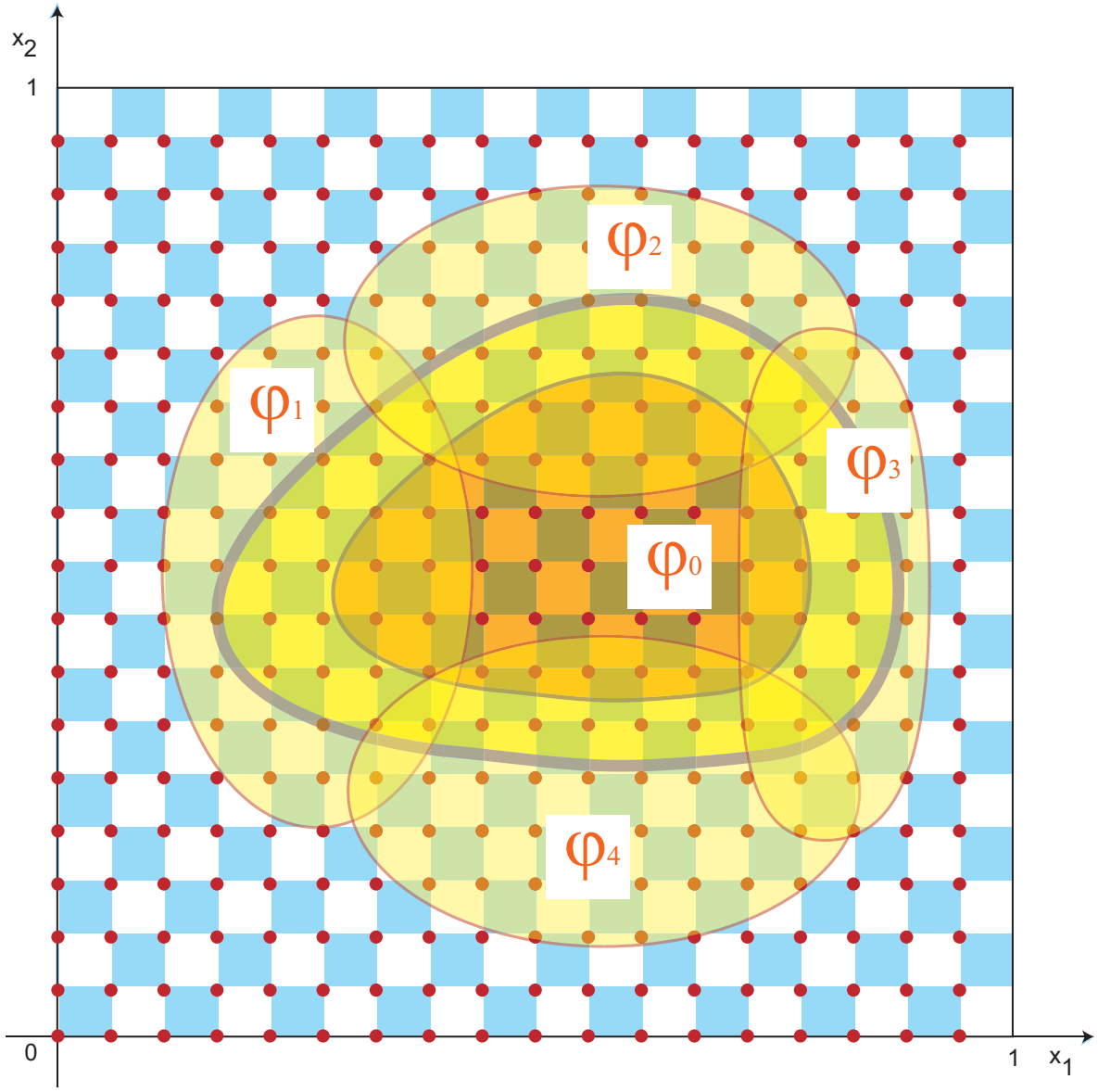

Рис. I.4. Срезывающие функции

Наложенных на $\Phi(x)$ условий достаточно для того, чтобы уравнение для каждого из участков границы в достаточно малой окрестности каждой точки границы можно было разрешить относительно координаты, задающей направление, ортогональное гиперплоскости, на которую мы гладко проектируем Γ_τ :

$$\forall\tau\ \exists j_\tau,\ \exists\gamma_\tau(\hat{x}_{j_\tau}):\quad \Gamma_\tau\equiv\{x\in\Upsilon_\tau:\ x_{j_\tau}=\gamma_\tau(\hat{x}_{j_\tau})\},\tag{I.37}$$

где $\hat{x}_{j_\tau}\stackrel{def}{=}(x_1,\ldots,x_{j_\tau-1},x_{j_\tau+1},\ldots,x_n)$.

Если каждой области Υ_τ сопоставить ФП

$$l_h^{\Upsilon_\tau}(x)\stackrel{def}{=}\chi_{\Upsilon_\tau}(x)-\sum_{\substack{hk\in\Upsilon_\tau,\\ k\in\mathbb{Z}^n}}c_k^\tau(h)\delta(x-hk),\tag{I.38}$$

то общий функционал погрешности можно собрать из таких функционалов следующим образом:

$$l_h^\Omega(x)\stackrel{def}{=}\sum_{\tau=0}^{\mathrm{T}}\varphi_\tau(x)l_h^{\Upsilon_\tau}(x).\tag{I.39}$$

При этом коэффициенты $c_k(h)$ функционала погрешности $l_h^\Omega(x)$ получаются следующим образом:

$$c_k(h)=\sum_{\tau=0}^{\mathrm{T}}\varphi_\tau(x)c_k^\tau(h).\tag{I.40}$$

Так как в определении ФП $l_h^\Omega(x)$ каждый ФП $l_h^{\Upsilon_\tau}(x)$ умножен на срезывающую функцию $\varphi_\tau(x)$, мы можем в дальнейшем доопределять $l_h^{\Upsilon_\tau}(x)$ вне $\operatorname{supp}\varphi_\tau(x)$, не изменяя итоговой суммы.

Для того, чтобы ФП $l_h^\Omega(x)$ был оптимальным по порядку в $\widetilde{W}_p^m(\Omega)$, достаточно оптимальности по порядку каждого из функционалов $l_h^{\Upsilon_\tau}(x)$. Ниже для произвольного τ подберем коэффициенты $c_k^\tau(h)$ так, чтобы выполнялось $\|l_h^{\Upsilon_\tau}\|_{(\widetilde{W}_p^m(\Omega))^*}\asymp O(h^m)$.

Для определенности, во-первых, будем предполагать, что уравнение для границы Γ_τ задается функцией, выражающей последнюю координату вектора x через остальные:

$$\Gamma_\tau\stackrel{def}{=}\{x\in\Upsilon_\tau:\ x_n=\gamma(x')\},\quad x'\stackrel{def}{=}(x_1,\ldots,x_{n-1}).\tag{I.41}$$

Во-вторых, пусть

$$x_n\geqslant\gamma(x'),\quad\forall x\in\Upsilon_\tau.\tag{I.42}$$

Для удобства впредь будем опускать символ τ в обозначениях.

В дальнейшем мы будем использовать одно- и $(n-1)$-мерные δ-функции, действующие следующим образом:

$$(\delta_n(x_n), f(x)) = f(x', 0), \quad (\delta'(x'), f(x)) = f(0', x_n). \tag{I.43}$$

Преобразуем $l_h^\Upsilon(x)$ заменой индексов $\begin{cases} k' = t', \\ k_n = t_n + \sigma \end{cases}$:

$$l_h^\Upsilon(x) = \chi_\Upsilon(x) - h^n \sum_{hk \in \Upsilon} c_{k',k_n} \delta'(x' - hk')\delta_n(x_n - hk_n) = \tag{I.44}$$

$$= \chi_\Upsilon(x) - h^n \sum_{h\binom{t'}{t_n+\sigma} \in \Upsilon} c_{t',t_n+\sigma}\, \delta'(x' - ht')\delta_n(x_n - h(t_n + \sigma)), \tag{I.45}$$

где

$$\sigma \equiv \sigma(ht') \stackrel{def}{=} \left[\frac{\gamma(ht')}{h}\right] \tag{I.46}$$

— целая часть числа $\frac{\gamma(ht')}{h}$.

Если фиксировать $t' = t'_0$, то значение $\sigma(ht'_0)$ равно количеству узлов решетки с неотрицательной последней координатой, лежащих на прямой $t' = t'_0$ не выше, чем точка $(t'_0, \gamma(ht'_0))$ пересечения этой прямой с границей Γ. Число $h\sigma(ht'_0)$ равняется расстоянию от координатной плоскости $t_n = 0$ до последнего узла на прямой $t' = t'_0$, не попадающего внутрь области Υ.

Продолжим преобразования заменой переменных

$$y = \Psi(x) : \begin{cases} y' = x', \\ y_n = x_n - \gamma(x') \end{cases}. \tag{I.47}$$

Оператор Ψ осуществляет преобразование области Υ с кривой нижней (относительно направления оси x_n) границей в область ω с выпрямленной нижней границей, лежащей в плоскости $x_n = 0$. Имеем:

$$l_h^\Upsilon(x) = \chi_\Upsilon\left(\Psi^{-1}(y)\right) - h^n \sum_{h\binom{t',}{t_n+\sigma} \in \Upsilon} c_{t',t_n+\sigma}\, \delta'(y' - ht')\delta_n(y_n + \gamma(y') - ht_n - h\sigma) =$$

$$= \chi_\omega(y) - h^n \sum_{h\binom{t',}{t_n-\eta} \in \omega} c_{t',t_n+\sigma}\, \delta'(y' - ht')\delta_n(y_n - h(t_n - \eta)) \quad \stackrel{def}{=} l_h^\omega(y), \tag{I.48}$$

где $\omega \stackrel{def}{=} \Psi(\Upsilon)$ и $\eta \equiv \eta(ht') \stackrel{def}{=} \left\{\frac{\gamma(ht')}{h}\right\}$ — дробная часть числа $\frac{\gamma(ht')}{h}$, $\eta \in [0, 1)$.

Исследование этого функционала затруднено тем, что решетка узлов $\{h(t', t_n - \eta)\}$ на котором он задан, криволинейна. Этот факт наглядно продемонстрирован на рис. I.5, где изображен двумерный случай. Белым цветом отмечены узлы решетки, попавшие в область Υ, а также их образ после преобразования функцией Ψ. Как видно, эти узлы не совпадают с теми узлами, значения в которых на данном шаге нам известны (черные точки), образуя криволинейную решетку.

Эта проблема ниже будет решаться приближением значений ФП в узлах криволинейной решетки линейной комбинацией значений в узлах кубической решетки с помощью оператора конечной разности высокого порядка.

Доопределим функцию $\gamma(x')$. Пусть для $x' : (x', 0) \notin \Psi(\Gamma)$ она гладко и независимо от h сходит на нуль. Тогда величина $\eta(ht')$ будет определена для любого $t' \in \mathbb{Z}^{n-1}$, а значит, если учесть, что носитель срезывающей функции лежит в Υ, можно перейти от рассмотрения функционала $l_h^{\omega}(y)$ к функционалу с более широким носителем, например, $l_h^{\tilde{Q}}(y)$, где

$$\tilde{Q} \stackrel{def}{=} Q' \times \left[0, \frac{1}{2}\right], \; Q' \stackrel{def}{=} \{x' \in \mathbb{R}^{n-1} : \; x_i \in [0, 1), \; i = \overline{1, n-1}\}.$$

Для того, чтобы свести задачу к интегрированию по гиперкубу, рассмотрим функционал

$$q_h^{\tilde{Q}}(x) = \sum_{hk \in \tilde{Q}} \lambda\left(\frac{x}{h} - k\right), \tag{I.49}$$

Или, после приведения подобных,

$$q_h^{\tilde{Q}}(x) = \chi_{\tilde{Q}}(x) - \sum_{hk' \in Q'} \sum_{k_n=1}^{\tilde{N}} b_k \delta(x - hk), \tag{I.50}$$

где $\tilde{N} \stackrel{def}{=} \left[\frac{N}{2}\right]$. Теперь заменим каждую функцию $\delta_n(x_n - hk_n)$ на линейную комбинацию δ-функций, сосредоточенных в узлах криволинейной решетки: $\sum_{z=1}^{M+1} d_z \delta_n(x_n - h(k_n + z - \eta))$.

Получаем другой функционал:

$$\tilde{q}_h^{\tilde{Q}}(x) \stackrel{def}{=} \chi_{\tilde{Q}}(x) - \sum_{hk' \in Q'} \sum_{k_n=1}^{\tilde{N}} b_k \delta'(x' - hk') \sum_{z=1}^{M+1} d_z \delta_n(x_n - h(k_n + z - \eta)). \tag{I.51}$$

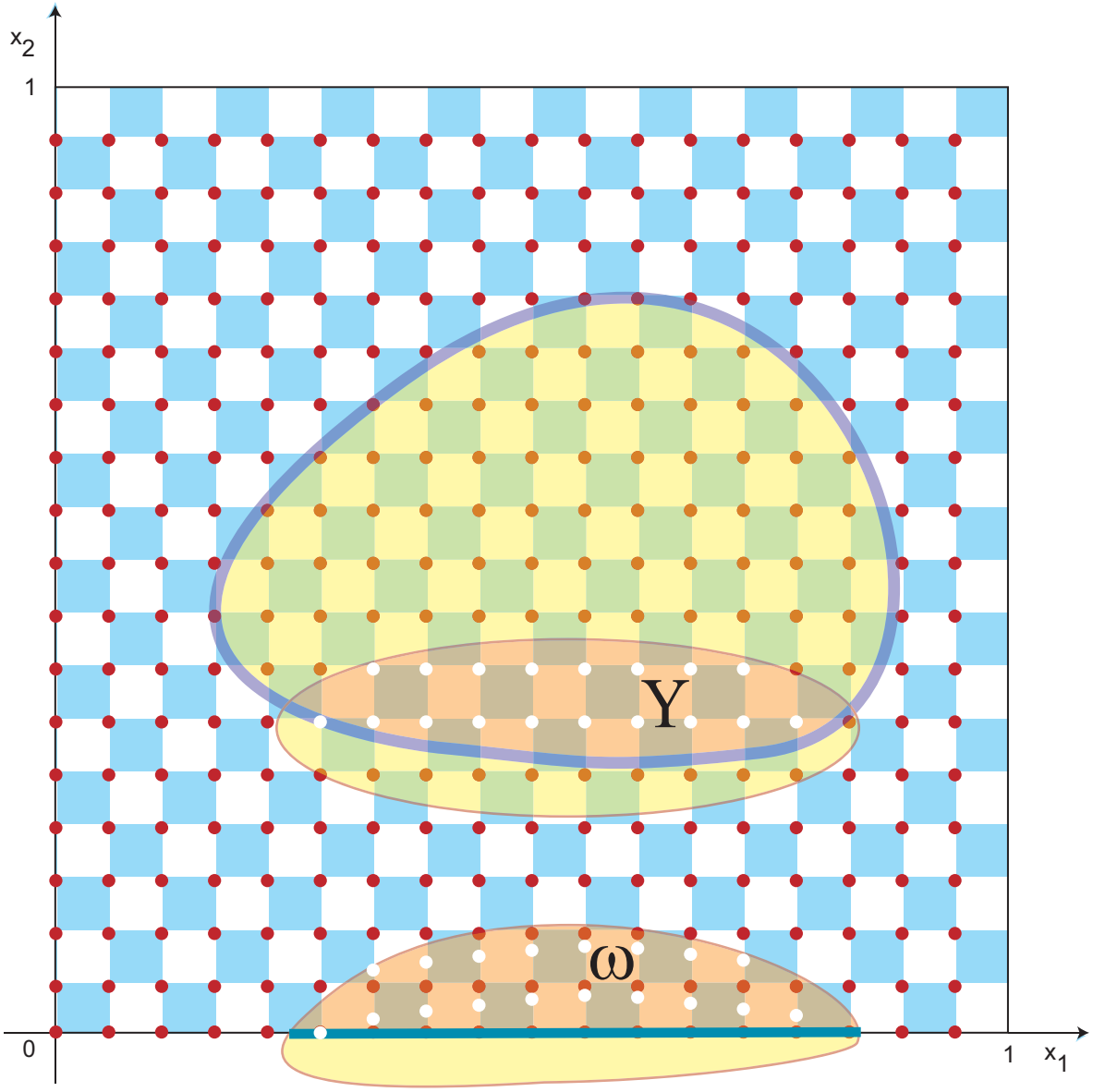

Рис. I.5. Ψ-преобразование области Υ в область ω; преобразование решетки

Мы произвели параллельный координатной оси k_n сдвиг узлов решетки $\{hk\}_{k\in\mathbb{Z}^n}$ на долю величины h. Этот сдвиг, вообще говоря, неодинаков на разных прямых $\{x : x = (k'h, tk_n),\ t \in \mathbb{R}\}$ и зависит лишь от значения функции γ в точке hk'_0.

Легко видеть, что полученная криволинейная решетка совпадает с решеткой узлов, на которой должен быть задан ФП $l_h^\omega(y)$, следовательно, можно получить таким образом искомый функционал. Следует подобрать коэффициенты d_p так, чтобы функционалы $q_h^{\tilde{Q}}(y)$ и $\tilde{q}_h^{\tilde{Q}}(y)$ достаточно мало отличались. Тогда мы сведем задачу к уже решенной задаче интегрирования по гиперкубу.

Рассмотрим функционал

$$\Delta(y) \stackrel{def}{=} \tilde{q}_h^{\tilde{Q}}(y) - q_h^{\tilde{Q}}(y) = \sum_{hk'\in Q'} \sum_{k_n=1}^{\tilde{N}} b_k \delta'(y' - hk')\Delta_n(y_n - hk_n), \qquad \text{(I.52)}$$

где

$$\Delta_n(y_n) \stackrel{def}{=} \delta_n(y_n) - \sum_{z=1}^{M+1} d_z \delta_n(y_n - h(z - \eta)). \qquad \text{(I.53)}$$

Эту одномерную обобщенную функцию следует рассматривать как $n-$мерную, учитывая возможную зависимость коэффициентов dz от y'. Наложим на функционал $\Delta_n(y_n)$ условие ортогональности всем одночленам $\{y_n^j\}$ до степени M включительно. Тогда будет верна оценка (см. [40, стр. 259–261])

$$\|\Delta\|_{(\widetilde{W}_p^m(Q))^*} \asymp O(h^m). \qquad \text{(I.54)}$$

Таким образом, нужный нам функционал погрешности можно получить как сумму двух функционалов, точных на одночленах до степени M. Именно это свойство придает первому из них оптимальные свойства при интегрировании по гиперкубу, а второму дает возможность с достаточным порядком точности связать два типа решеток — прямоугольную и криволинейную.

Для определения коэффициентов ФП $\tilde{q}_h^{\tilde{Q}}(y)$ достаточно найти коэффициенты b_k и d_z.

Коэффициент b_k в формуле (I.50) получается приведением подобных слагаемых в правой части (I.49). Это можно представить, как пересечение в точке k некоторого количества «облаков» — носителей δ-функций, каждое из которых представляет собой часть решетки узлов $\{ht\}_{t\in\mathbb{Z}^n}$, которые

составляют сингулярный носитель функционала $\lambda\left(\frac{x}{h}-p\right)$. Если в точке k пересеклись все возможные «облака», т.е. $b_k=\sum\limits_{s\in S}a_s$, то благодаря тому, что элементарные функционалы λ, в частности, точны на одночленах степени 0, имеем:

$$b_k=\sum_{s\in S}a_s=\sum_{s_1=1}^{M+1}a_{s_1}\cdot\ldots\cdot\sum_{s_n=1}^{M+1}a_{s_n}=1\cdot\ldots\cdot 1=1. \tag{I.55}$$

Таких коэффициентов большинство, неполные «наложения облаков» возникают, очевидно, лишь в слое толщины Mh около некоторых координатных граней $\tilde{Q}$. Однако, если учесть, что Ψ-образ носителя срезывающей функции сосредоточен лишь у одной грани $k_n=0$ параллелепипеда $\tilde{Q}$, можно считать, что слой узлов, которым соответствуют коэффициенты, не равные единице, сосредоточен только у этой грани. Тогда по направлениям k' «облака» перекрываются в полной мере, т.е. после группировки коэффициентов при соответствующих δ-функциях, итоговые коэффициенты слагаются из $M+1$ членов и, следовательно, в сумме дают единицу:

$$b_k=\sum_{s_1=1}^{M+1}a_{s_1}\cdot\ldots\cdot\sum_{s_{n-1}=1}^{M+1}a_{s_{n-1}}\sum_{s_n=1}^{\min\{k_n,M+1\}}a_{s_n}=1\cdot\ldots\cdot 1\cdot\sum_{s_n=1}^{\min\{k_n,M+1\}}a_{s_n}=\sum_{s_n=1}^{\min\{k_n,M+1\}}a_{s_n}. \tag{I.56}$$

По направлению k_n узлы с большими значениями величины hk_n выходят из носителя срезывающей функции.

Коэффициент b_k, таким образом, зависит только от последней компоненты своего индекса. Окончательно, используя (I.25), имеем:

$$b_k=\sum_{r=1}^{\min\{k_n,M+1\}}\sum_{p=1}^{M+1}v_{rp}\frac{1}{p}. \tag{I.57}$$

Вычислим теперь коэффициенты d_z. Воспользуемся тем, что одномерная обобщенная функция Δ_n, заданная в (I.53) ортогональна одночленам до степени M включительно:

$$(\Delta_n(y_n),y_n^\alpha)=0,\quad \alpha=\overline{0,M} \tag{I.58}$$

или

$$(\Delta_n(y_n-h\eta),(y_n-h\eta)^\alpha)=0,\quad \alpha=\overline{0,M}. \tag{I.59}$$

Увеличивая α и возводя в степень, легко показать, что последнее равенство эквивалентно следующему:

$$(\Delta_n(y_n-h\eta),y_n^\alpha)=0,\quad \alpha=\overline{0,M}. \tag{I.60}$$

Подставляя сюда выражение для Δ_n из (I.53), получим систему алгебраических уравнений:

$$\sum_{z=1}^{M+1} d_z z^\alpha = \eta^\alpha. \tag{I.61}$$

Матрицей это системы является матрица с определителем Вандермонда (I.24). Пользуясь уже введенными обозначениями для элементов обратной матрицы, запишем решение системы:

$$d_z = \sum_{i=1}^{M+1} v_{zi}\, \eta^{i-1}. \tag{I.62}$$

Подставляя приведенные в (I.57) и (I.62) выражения для коэффициентов b_k и d_z в (I.51), получим:

$$\tilde{q}_h^{\tilde{Q}}(y) = \chi_{\tilde{Q}}(y) - \sum_{hk' \in Q'} \delta'(y' - hk') D(y_n), \tag{I.63}$$

где

$$D(y_n) \stackrel{def}{=} \sum_{k_n=1}^{\tilde{N}} \sum_{r=1}^{\min\{k_n, M+1\}} \sum_{p=1}^{M+1} v_{rp} \frac{1}{p} \sum_{z=1}^{M+1} \sum_{i=1}^{M+1} v_{zi} \eta^{i-1} \delta_n(y_n - h(k_n + z - \eta)).$$

После замены переменных $k_n = \tilde{k_n} - z$ получим:

$$D(y_n) = \sum_{p=1}^{M+1} \frac{1}{p} \sum_{i=1}^{M+1} \eta^{i-1} \sum_{z=1}^{M+1} v_{zi} \sum_{\tilde{k_n}=z+1}^{z+\tilde{N}} \sum_{r=1}^{\min\{\tilde{k_n}-z, M+1\}} v_{rp} \delta_n(y_n - h\tilde{k_n} + h\eta). \tag{I.64}$$

Переставим порядки суммирования по переменным z и $\tilde{k_n}$. Имеем:

$$\begin{cases} 1 \leqslant z \leqslant M+1, \\ z+1 \leqslant \tilde{k_n} \leqslant z + \tilde{N} \end{cases} \Leftrightarrow \begin{cases} 2 \leqslant \tilde{k_n} \leqslant M + \tilde{N} + 1, \\ \max\{1, \tilde{k_n} - \tilde{N}\} \leqslant z \leqslant \min\{M+1, \tilde{k_n} - 1\}. \end{cases} \tag{I.65}$$

Благодаря влиянию срезывающей функции, верхнюю грань изменения переменной $\tilde{k_n}$ можно заменить на $\tilde{N}$. Тогда $\max\{1, \tilde{k_n} - \tilde{N}\} = 1$ и

$$D(y_n) = \sum_{\tilde{k_n}=2}^{\tilde{N}} \sum_{p=1}^{M+1} \frac{1}{p} \sum_{i=1}^{M+1} \eta^{i-1} \sum_{z=1}^{\min\{\tilde{k_n}-1, M+1\}} v_{zi} \sum_{r=1}^{\min\{\tilde{k_n}-z, M+1\}} v_{rp}\, \delta_n(y_n - h(\tilde{k_n} - \eta)). \tag{I.66}$$

Подставляя полученное выражение в (I.63), сравнивая результат с (I.48) и вспоминая, что мы для удобства опускали символ τ, можем выписать искомые коэффициенты:

$$c_k^\tau \stackrel{def}{=} \begin{cases} \sum\limits_{p=1}^{M+1} \frac{1}{p} \sum\limits_{i=1}^{M+1} \eta^{i-1} \sum\limits_{z=1}^{\min\{k_n-\sigma-1,M+1\}} v_{zi} \sum\limits_{r=1}^{\min\{k_n-\sigma-z,M+1\}} v_{rp}, & k_n \geqslant 2+\sigma \\ 0, & k_n \leqslant 1+\sigma. \end{cases} \tag{I.67}$$

Условие $k_n \geqslant 2+\sigma$ мы получили из неравенства $\tilde{k_n} \geqslant 2$.

Итоговые коэффициенты находятся по формуле (I.40).

§3. Модификация алгоритма

Формула (I.67) изменена диссертантом следующим образом:

$$c_t^\tau(h) = \begin{cases} \sum\limits_{i=0}^{\min\{t_n,M\}} A_{t_n-i} \sum\limits_{l=0}^{M} \eta^l \tilde{v}_{il}, & t_n \geqslant 0 \\ 0, & t_n < 0. \end{cases} \tag{I.68}$$

Здесь $t' \stackrel{def}{=} k',\ t_n \stackrel{def}{=} k_n - \sigma - 2; \quad \tilde{v}_{ij} \stackrel{def}{=} v_{i+1,\ j+1},\ i,j = \overline{0,M};$

$$A_i \stackrel{def}{=} \sum_{p=0}^{M} \frac{1}{p+1} \sum_{j=0}^{i} \tilde{v}_{jp}, \quad i = \overline{0..M} \tag{I.69}$$

Заметим, что, в отличие от формулы (I.67), в формуле (I.68) явно выделены множители A_i, которые для каждого M можно вычислить заранее, избавившись от необходимости пересчитывать четыре вложенные суммы в правой части (I.67) в каждой точке.

Объем вычислений сокращается также из-за того, что при $t_n \geqslant 2M$ вне зависимости от t' коэффициент $c_k^\tau(h)$ равен единице. Поэтому вычисление коэффициентов данной локальной кубатурной формулы $l_h^{\Upsilon_\tau}(x)$ необходимо провести лишь для точек пограничного слоя, при $t_n = \overline{0\ ..\ 2M-1}$.

Для практической реализации алгоритма необходимо построить T+1 срезывающих функций

$$\{\varphi_\tau(x)\}_{\tau=0}^{\mathrm{T}}, \qquad \sum_{\tau=0}^{\mathrm{T}} \varphi_\tau(x)|_{x\in\hat{\Omega}} \equiv 1, \tag{I.70}$$

удовлетворяющих наложенным выше условиям.

Так как в алгоритме фактически используются лишь внутренние узлы области Ω, можно не следить за разбиением единицы вне области и не соблюдать условие периодичности срезывающих функций. То же касается и подынтегральной функции $f(x)$ — она не обязана быть периодична и определена вне области Ω.

Разбиение единицы произведем в два этапа. Вначале построим центральную срезывающую функцию $\varphi_0(x)$, удовлетворяющую условию (I.32). Затем построим разбиение единицы для гиперкуба Q

$$\{\psi_\tau(x)\}_{\tau=1}^{\mathrm{T}}, \qquad \sum_{\tau=1}^{\mathrm{T}} \psi_\tau(x)|_{x\in Q} \equiv 1. \tag{I.71}$$

И, наконец, получим окончательный вид срезывающих функций:

$$\varphi_\tau(x) \stackrel{def}{=} \psi_\tau(x)(1-\varphi_0(x)), \quad \tau = \overline{1..\mathrm{T}} \tag{I.72}$$

При этом

$$\sum_{\tau=0}^{\mathrm{T}} \varphi_t(x) \equiv \varphi_0(x) + \sum_{\tau=1}^{\mathrm{T}} \varphi_\tau(x) \equiv \varphi_0(x) + (1-\varphi_0(x)) \sum_{\tau=1}^{\mathrm{T}} \psi_\tau(x) \equiv 1, \ x \in Q. \tag{I.73}$$

Нам понадобится вспомогательная функция $\hat{\xi}(t)$:

$$\hat{\xi}(t) \stackrel{def}{=} \begin{cases} 0, \ t < 0 \\ \int\limits_0^t z(t)dt \Big/ \int\limits_0^1 z(t)dt, \quad z(t) = (t(1-t))^M, \qquad 0 \leqslant t < 1, \\ 1, \ 1 \leqslant t. \end{cases} \tag{I.74}$$

Ее график при $M = 4$ приведен на рис. II.2.

Центральную срезывающую функцию $\varphi_0(x)$ зададим так:

$$\varphi_0(x) \stackrel{def}{=} \hat{\xi}\left(\frac{\Phi(x)}{\varepsilon_2-\varepsilon_1} - \frac{\varepsilon_1}{\varepsilon_2-\varepsilon_1}\right), \tag{I.75}$$

где $\Phi(x)$ — функция, задающая область Ω (см. (I.34 – I.36)).

Тогда

$$\varphi_0(x) = \begin{cases} 0, \ \Phi(x) \leqslant \varepsilon_1 \\ p \in (0,1), \ \varepsilon_1 < \Phi(x) < \varepsilon_2, \\ 1, \ \varepsilon_2 \leqslant \Phi(x). \end{cases} \tag{I.76}$$

Таким образом, конкретный вид функции $\varphi_0(x)$ зависит как от способа задания области Ω, так и от параметров ε_1 и ε_2. При этом для выполнения условия (I.32) необходимо, чтобы неравенство $\Phi(x) \geqslant \varepsilon_2$ выполнялось лишь для точек $x:\ \rho(x, \mathbb{R}^n \backslash \Omega) \geqslant \varepsilon_0$, то есть для точек, достаточно удаленных от границы.

Изображение функций $\Phi(x)$ и $\varphi_0(x)$ при $\varepsilon_1 = 0.2,\ \varepsilon_2 = 0.5,\ M = 2,\ n = 2$ и $\Phi(x) = 1 - (2x_1 - 1)^2 - (2x_2 - 1)^2$ см. на рис. (I.6) и (I.7).

Построим функции $\{\psi_\tau(x)\}_{\tau=1}^{\mathrm{T}}$. Потребуем, чтобы центр гиперкуба Q принадлежал области Ω. Тогда, учитывая звездность выпуклой области, можно построить ровно $2n$ срезывающих функций (по числу граней гиперкуба), удовлетворяющих условиям, наложенных на них выше. Далее будем считать $\mathrm{T} \equiv 2n$.

Для удобства будем нумеровать координатные оси от 0 до $n-1$, а систему функций $\{\psi_\tau(x)\}_{\tau=1}^{2n}$ заменим на систему

$$\{\tilde{\psi}_{dim,dir}(x)\}_{dim=0..n-1,\ dir=\pm 1}. \tag{I.77}$$

Первый индекс dim (от dimension) означает размерность, второй индекс dir (от direction) — направление (отрицательное или положительное) . Будем строить систему срезывающих функций так, чтобы функция $\tilde{\psi}_{dim,dir}(x)$ «опиралась» на грань гиперкуба Q, ортогональную оси x_{dim} и такой, что $x_{dim} \equiv \frac{dir+1}{2}$ для любой точки x, принадлежащей этой грани. Под выражением «$\tilde{\psi}_{dim,dir}(x)$ опирается на грань» имеется ввиду то, что соответствующая срезывающая функция $\tilde{\varphi}_{dim,dir}(x)$, определяемая из

$$\tilde{\varphi}_{dim,dir}(x) \stackrel{def}{=} \tilde{\psi}_{dim,dir}(x)(1 - \varphi_0(x)), \quad dim = 0..n-1,\ dir = \pm 1 \tag{I.78}$$

(получено из (I.72) в новых обозначениях) должна гладко и взаимноодно-значно проектировать вырезаемый ей участок границы области Ω на координатную плоскость, содержащую эту грань.

Зададим вначале разбиение единицы для двумерного случая.

Введем вспомогательные функции

$$\hat{\psi}(t, A) \stackrel{def}{=} \hat{\xi}(At)\hat{\xi}(A(1-t)) \tag{I.79}$$

(при $A = 6$ см. рис. (I.8)),

$$A_{-1}(t, b, c) \stackrel{def}{=} -\frac{b}{c}t + b, \qquad A_1(t, b, c) \stackrel{def}{=} \frac{b}{c}t - \frac{(1-c)b}{c}. \tag{I.80}$$

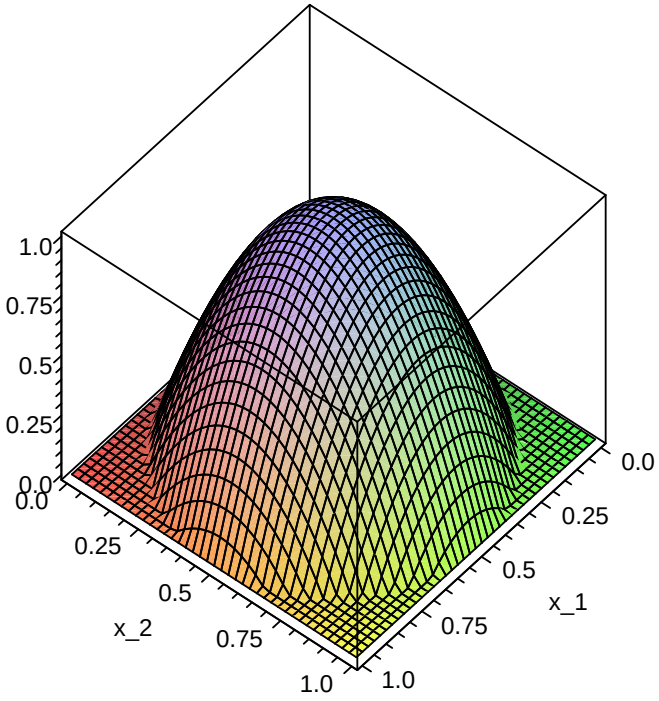

Рис. I.6. Функция $\Phi(x_1, x_2)$

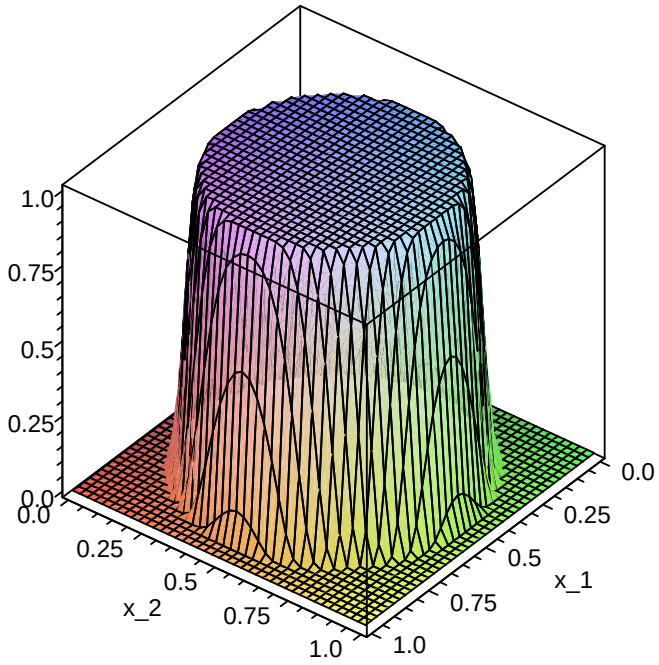

Рис. I.7. Функция $\varphi_0(x_1, x_2)$

Рассмотрим плоскость $\{x_0, x_1\}$. Зададим разбиение единицы:

$$\tilde{\psi}_{0,-1}(x_0, x_1) \stackrel{def}{=} \hat{\psi}(x_1, A_{-1}(x_0, b, c)), \tag{I.81}$$

$$\tilde{\psi}_{0,1}(x_0, x_1) \stackrel{def}{=} \hat{\psi}(x_1, A_1(x_0, b, c)) \tag{I.82}$$

$$\tilde{\psi}_{1,-1}(x_0, x_1) + \tilde{\psi}_{1,1}(x_0, x_1) \stackrel{def}{=} 1 - (\tilde{\psi}_{0,-1}(x_0, x_1) + \tilde{\psi}_{0,1}(x_0, x_1)). \tag{I.83}$$

На рис. I.9 и I.10 изображены графики функций соответственно $\tilde{\psi}_{0,-1}(x_0, x_1) + \tilde{\psi}_{0,1}(x_0, x_1)$ и $\tilde{\psi}_{1,-1}(x_0, x_1) + \tilde{\psi}_{1,1}(x_0, x_1)$ со следующими значениями параметров: $M = 2,\ b = 6,\ c = 0.5$.

Отметим, что вспомогательные функции, используемые в приведенных выше формулах построения разбиения единицы, не являются изобретением диссертанта и описаны, например, в [11]. Однако, в отличие от [11], где для построения многомерных срезок используется градиентное разбиение единицы (что дает универсальность разбиения для произвольных ограниченных областей) и функции свертки (для сглаживания), описанный ниже алгоритм не использует эти крайне долгие для вычислений операции.

Основой для построения разбиения единицы в многомерном случае будут двумерные разбиения во всевозможных координатных плоскостях. В n-мерном пространстве таких плоскостей ровно $\frac{n(n-1)}{2}$ — число сочетаний из n по 2.

Каждую из плоскостей $\{x_i,\ x_j\}$ будем разбивать двумя (для удобства) функциями:

$$\breve{\psi}_{i,j} \stackrel{def}{=} \tilde{\psi}_{i,-1}(x_i, x_j) + \tilde{\psi}_{i,1}(x_i, x_j) \tag{I.84}$$

и

$$\breve{\psi}_{j,i} \stackrel{def}{=} \tilde{\psi}_{j,-1}(x_i, x_j) + \tilde{\psi}_{j,1}(x_i, x_j) \tag{I.85}$$

(см. те же рис. I.9 и I.10).

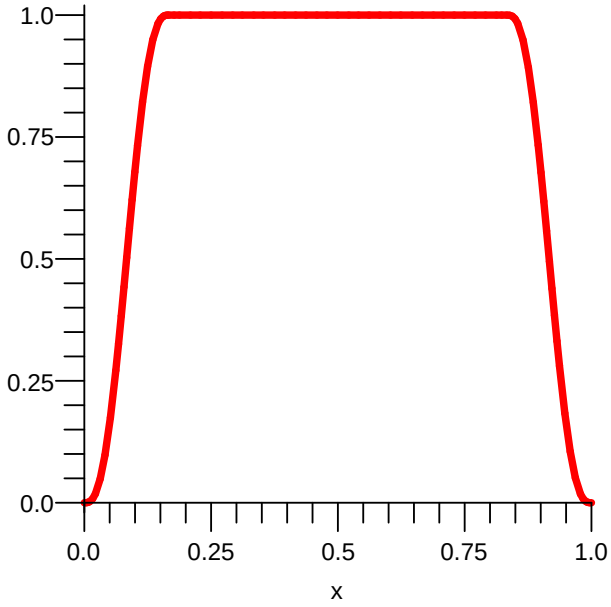

Рис. I.8. Функция $\hat{\psi}(x, 6)$

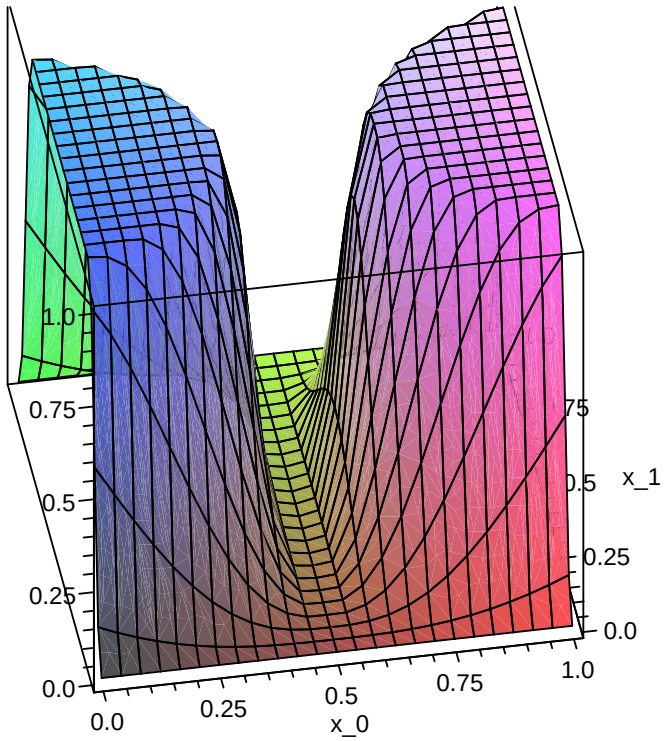

Рис. I.9. Функция $\tilde{\psi}_{0,-1}(x_0, x_1) + \tilde{\psi}_{0,1}(x_0, x_1)$

Имеем:

$$\begin{cases} \breve{\psi}_{0,1} & + & \breve{\psi}_{1,0} & = 1 \\ \breve{\psi}_{0,2} & + & \breve{\psi}_{2,0} & = 1 \\ & \dots & & \\ \breve{\psi}_{0,n-1} & + & \breve{\psi}_{n-1,0} & = 1 \\ \breve{\psi}_{1,2} & + & \breve{\psi}_{2,1} & = 1 \\ \breve{\psi}_{1,3} & + & \breve{\psi}_{3,1} & = 1 \\ & \dots & & \\ \breve{\psi}_{1,n-1} & + & \breve{\psi}_{n-1,1} & = 1 \\ & \dots & & \\ & \dots & & \\ \breve{\psi}_{n-2,n-1} & + & \breve{\psi}_{n-1,n-2} & = 1 \end{cases} \tag{I.86}$$

Перемножив левые части этих уравнений, получим выражение с $2^{\frac{n(n-1)}{2}}$ слагаемыми, тождественно равное единице. Каждое из этих слагаемых представляет собой функцию n переменных и может быть использовано в качестве одной из многомерных функций разбиения единицы. Однако количество слагаемых может быть очень велико — в десятимерном случае их $2^{45} \approx 35 \cdot 10^{12}$ штук.

Сгруппируем их в n слагаемых следующим образом (опуская аргументы функций):

$$\begin{cases} \tilde{\breve{\psi}}_0 \stackrel{def}{=} \breve{\Psi}^0_{n-1}, \\ \tilde{\breve{\psi}}_1 \stackrel{def}{=} \breve{\Psi}^1_{n-1}, \\ \tilde{\breve{\psi}}_2 \stackrel{def}{=} 1 - \breve{\Psi}^0_2 - \breve{\Psi}^1_2, \\ \tilde{\breve{\psi}}_3 \stackrel{def}{=} \breve{\Psi}^0_2 + \breve{\Psi}^1_2 - \breve{\Psi}^0_3 - \breve{\Psi}^1_3, \\ \dots \\ \tilde{\breve{\psi}}_{n-1} \stackrel{def}{=} \breve{\Psi}^0_{n-2} + \breve{\Psi}^1_{n-2} - \breve{\Psi}^0_{n-1} - \breve{\Psi}^1_{n-1}, \end{cases} \tag{I.87}$$

где

$$\begin{aligned} &\breve{\Psi}^0_i \stackrel{def}{=} \breve{\psi}_{0,1}\breve{\psi}_{0,2}\dots\breve{\psi}_{0,i}, \\ &\breve{\Psi}^1_i \stackrel{def}{=} \breve{\psi}_{1,0}\breve{\psi}_{1,2}\dots\breve{\psi}_{1,i}, \\ &i = \overline{0..n-1}. \end{aligned} \tag{I.88}$$

Каждая из полученных функций $\tilde{\breve{\psi}}_i$ легко разбивается на два слагаемых, в зависимости от знака выражения $(x_i - 0.5)$. Таким образом, получаем необходимые $2n$ функций ψ_τ разбиения единицы для гиперкуба Q, дающие искомую систему срезывающих функций (I.72). Для на-

глядности приведем систему функций из (I.72) на одном графике для $n = 2,\ M = 2,\ b = 6,\ c = 0.5$.

§4. Программа

По изложенному алгоритму была создана программа на языке C++ с использованием библиотеки параллельных функций MPI [36]. Программа подробно закомментирована и приведена в прил. 1.

Программа состоит из пяти файлов, каждый из которых имеет специальное предназначение.

- Файл `ri.cpp` является главным файлом программы. Из него вызываются функции остальных файлов. Содержит основные вычислительные циклы.
- Файл `functions.cpp` содержит вспомогательные функции программы.
- Файл `proto.h` содержит описания глобальных переменных.
- Файл `vand.cpp` содержит описания трех массивов, связанных с элементами матрицы, обратной к матрице с определителем Вандермонда.
- Файл `print.cpp` содержит функции вывода результатов счета на экран и в текстовый файл.

Рассмотрим каждый из файлов подробнее.

В заголовочном файле `proto.h` описаны глобальные (внешние) переменные, доступные из всех остальных файлов программы. Назначение каждой из них указано в комментариях к программе. Наиболее важные из них мы упомянем ниже, при их использовании.

Файл `vand.cpp` содержит единственную функцию `Init_Vand`, в которой инициализируются три массива. Массив `vand` задается явно. Он имеет три индекса, первый из которых варьируется от 2 до 6, обозначая гладкость M и задает размер матрицы $(M+1) \times (M+1)$, получающейся при трактовке второго индекса как номера строки, а третьего — как номера столбца. Элементы этих матриц вычислены заранее с высокой точностью — это матрицы разных размеров (от 3×3 до 7×7), обратные к матрицам с

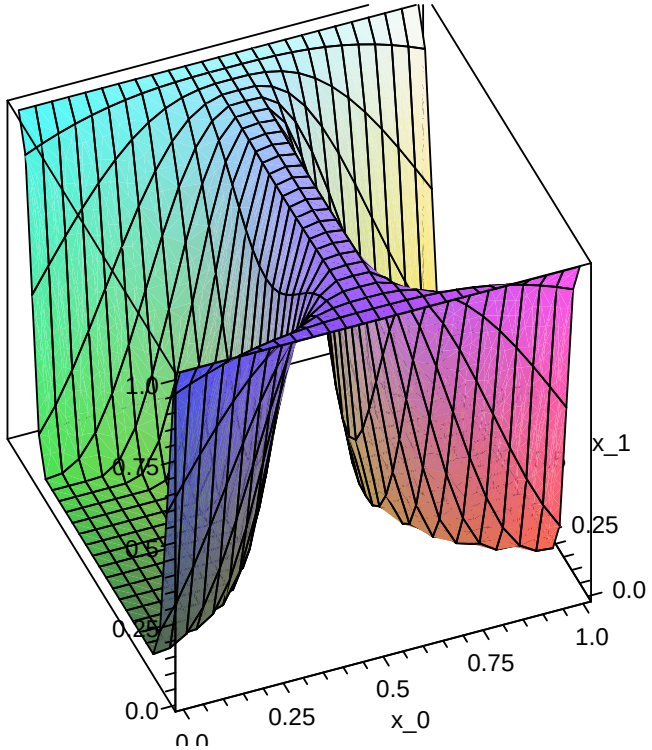

Рис. I.10. Функция $\tilde{\psi}_{1,-1}(x_0, x_1) + \tilde{\psi}_{1,1}(x_0, x_1)$

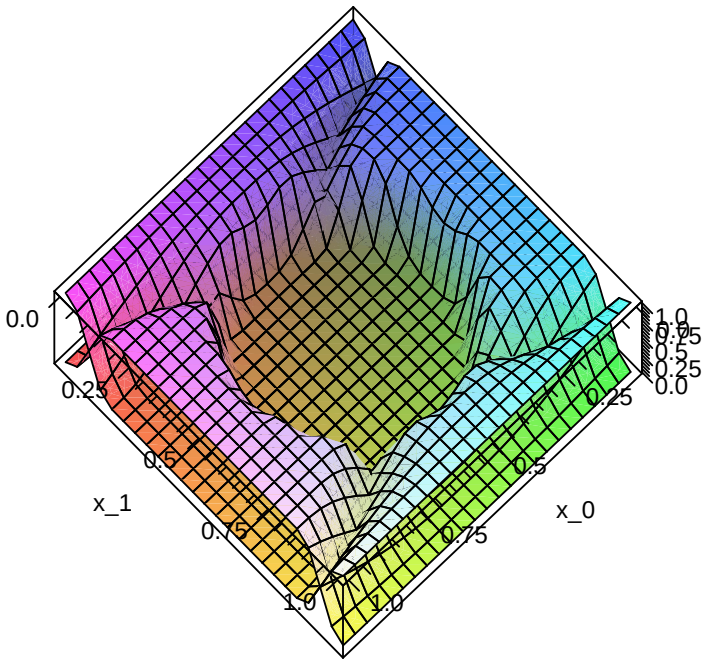

Рис. I.11. Пример срезывающих функций, заданных в (I.72)

определителем Вандермонда соответствующих размеров. Массив `a` является вспомогательным. В результате вычисляется массив `A`, непосредственно использующийся для вычисления коэффициентов КФ (см. I.69).

Файл `print.cpp` содержит несколько функций печати.

Функция `OpenFile` создает текстовый файл для вывода результатов. Для удобства его название меняется в зависимости от числа процессоров P, выделенных на задачу, размерности пространства n, числа точек на ребре гиперкуба N, гладкости M, даты и времени создания файла.

Функция `InitPrint` выводит значения начальных данных задачи и подготавливает шапку для вывода результатов.

Функция `PrintResults` вызывается каждый раз, когда получен очередной ответ. Результаты выводятся построчно, образуя таблицу.

Функция `SquarePrint` при необходимости выводит двумерную картину коэффициентов.

Дадим обзор функций файла `functions.cpp`.

В первой из них — `f` — задается интегрируемая функция. Ее аргументом является целочисленный вектор k, который преобразуется в вектор x, вычисляемый как $x = hk$.

Функция `Phi` задает область интегрирования Ω.

Функция `SetVar` инициализирует значения основных переменных. Это n — размерность пространства, N_{start}, N_{end} — начальное и конечное значение величины N, M_{start}, M_{end} — начальное и конечное значение параметра гладкости M, ta — значение точного ответа (если известен заранее).

Почти все параметры можно вводить из командной строки, заменяя их значения по умолчанию новыми значениями. Примеры использования командной строки приведем ниже.

Функция `xi`, в зависимости от значения аргумента t и гладкости M, возвращает значение функции $\hat{\xi}$ (см. I.74).

Функция `psi_prep` является вспомогательной для вычисления срезывающих функций в точке hk. Ее основное назначение — вычисление массивов `PP0` и `PP1`, соответствующих функциям $\breve{\Psi}_i^0$ и $\breve{\Psi}_i^1$ (см. I.87 и I.88).

Функция `psi` возвращает значение одной из n функций разбиения единицы (см. I.87), используя значения массивов `PP0` и `PP1`.

Процедура `Sigma` вычисляет значение переменной `sigma` (I.46) соответствующей целому числу узлов от границы до ближайшей координатной плоскости.

`gammaN` — функция участка границы области интегрирования. По заданной точке kh и направлению проектирования границы (`dim` и `dir`) находит точку пересечения направления проектирования и границы области интегрирования. Возвращает точки, деленную на шаг решетки h (в зависимости он направления проектирования). Уточнение `dim`-ной координаты происходит методом секущих до тех пор, пока погрешность не станет меньше `eps_secant`. Последняя величина по умолчанию равна 10^{-15}, но может быть изменена из командной строки.

Функция `phi0` соответствует функции $\varphi_0(x)$ (см. I.75).

Опишем файл `ri.cpp`. В начале идет список подключаемых библиотек и файлов программы. Затем идет список определений глобальных переменных и начинается тело единственной функции `main`. После списка локальных переменных происходит инициализация параллельной части программы (`MPI_Init`). Далее каждый процесс[1] узнает общее количество процессов `size` в коммуникаторе (`MPI_Comm_size`), а также свой номер `rank` (`MPI_Comm_rank`). После вызова функции инициализации переменных `SetVar`, главный процесс (нулевой) подготавливает печать результатов на экран и в файл.

После инициализации вспомогательных переменных начинается самый большой цикл — по количеству точек (на ребре гиперкуба) N. При помощи командной строки можно задать границы цикла N_{start} и N_{end}, а также коэффициенты b_N и a_N, управляющие линейным законом увеличения параметра N : $N = b_N\, N + a_N$. Затем идет подсчет количества узлов, содержащихся в каждой из гипергграней гиперкуба Q. Это необходимо для того, чтобы подсчитать общее количество точек и разделить его как можно равномернее между процессами, а также для того, чтобы произвести процесс превращения одномерной координаты в n-мерную и наоборот.

Второй вложенный цикл происходит по параметру гладкости M. Начиная с этого момента задача окончательно определена, поэтому засекается время (`begin_time= MPI_Wtime()`).

Далее происходит распределение узлов по процессам. Наилучшая равномерность распределения вычислений достигается тем, что разрезается одномерный массив `V[n-1]`. Процессу с номером `rank` доверяется вычисление `(e-s+1)` узлов, одномерные координаты которых заполняют отрезок

[1]Как правило, на одном процессоре запускается один процесс, но в общем случае это не верно, поэтому мы будем различать эти понятия

массива `V[n-1]` с началом в `s` и концом в `e`.

Затем происходит обнуление отладочных счетчиков и инициализация переменных, используемых в режиме мониторинга (в этом режиме на экран в реальном времени выводится степень завершенности вычислений в процентах).

Цикл по `s` (одномерной координате точки, рассматриваемой в данный момент) является главным циклом алгоритма. В нем происходит накопление ответа в переменной `sum_local` — части интегральной суммы, приходящейся на один процесс.

После восстановления многомерных координат точки из одномерной, идет условие `Phi(k)>0`, отсеивающее узлы, не попавшие в область Ω.

Второе отсеивающее условие `Phi_t< eps2` или, что то же самое, `phi0(Phi_t)<1` позволяет не вычислять коэффициенты для узлов, попавших во множество (I.32), то есть лежащих достаточно далеко от границы, так они равны единице.

Далее идет перебор срезывающих функций вида (I.87). Для тех из них, которые не равны нулю в данной точке hk, происходит вычисление локального (различного для разных срезок) коэффициента `koeff`. В цикле накапливается глобальный коэффициент `KOEFF`. Наконец, вычисляется очередное слагаемое кубатурной формулы, накапливаемое в переменной `sum_local`.

После того, как каждый процесс переберет все свои узлы, выполняется коллективная функция сбора данных `MPI_Reduce`. Общая сумма записывается в переменную `sum` главного процесса и домножается им на объем элементарной ячейки h^n. Происходит вторая отметка времени и результаты выводятся на экран и в файл.

Функция `MPI_Finalize` завершает параллельную часть программы.

Компиляция программы осуществляется командной строкой

```
mpiCC ri.cpp -o ri -lm
```

Запуск программы:

```
mpirun -np 10 -maxtime 5 ri n 5 N 100 1000 1 100 M 2 6
monit 1 eps1 0.15 eps2 0.2 mid_psi 0.4 tol
```

Программа `ri` запустится на 10 процессорах с ограничением времени в 5 минут. Размерность подынтегральной функции равна 5. Величина

N будет меняться от 100 до 1000 с шагом 100, для каждого N будет вычисляться ответ с несколькими параметрами гладкости (от 2 до 6). Режим мониторинга включен. Параметры функции $\psi_0(x)$ — $eps1$ и $eps2$ заданы значениями 0.15 и 0.2 соответственно. Один из параметров срезывающих функций `mid_psi` равен 0.4. Включен режим (`tol`) вычисления погрешности по устойчивости знаков (каждый ответ, кроме самого первого, сравнивается с предыдущим (т.е. с ответом при $N-1$, но с тем же M), вычисляется модуль их разности).

Приведем теперь полный список параметров программы.

- P — количество процессоров.
- f(x) — подынтегральная функция. Задается в тексте программы (функция `f` файла `functions.cpp`). Допустимо также задание в командной строке массивов параметров `b1` и `p1` в качестве коэффициентов функции $f(x) = \sin\left(\sum\limits_{i=0}^{n-1} b1_i x_i^{p1_i}\right)$.
- n — размерность пространства (от 2 до 10).
- M — декларируемая гладкость функции (от 2 до 6). Для того, чтобы вычислить интеграл при нескольких значениях M следует задать параметры M_{start} и M_{end}.
- $\Phi(x)$ — функция, задающая выпуклую область интегрирования с гладкими границами (функция `Phi` файла `functions.cpp`)
- Параметр N — число точек на одном ребре гиперкуба, величина обратная шагу решетки h. Общее количество узлов $\widehat{N} = N^n$. Для того, чтобы вычислить интеграл при нескольких значениях N следует задать параметры N_{start} и N_{end}, а также коэффициенты b_N и a_N.
- Параметры функции $\psi_0(x)$ — $eps1$ и $eps2$, заданные по умолчанию значениями 0.2 и 0.5 соответственно.
- Параметры срезывающих функций: `top_psi` и `mid_psi`, равные по умолчанию 6 и 0.3 соответственно.
- `eps_secant` — параметр метода секущих. По умолчанию равен 10^{-15}, но может быть изменен из командной строки.

- `tol` — параметр вычисления погрешности по устойчивости знаков. Включен, если указан в командной строке.
- `ta` — эталон. При наличии верного ответа, он указывается в командной строке, через пробел после символов `ta`. В этом случае каждый вычисленный ответ сравнивается с эталоном, и их разность выводится в строке результата. Если параметры `tol` и `ta` отсутствуют в командной строке, за верный ответ берется объем n-мерного гипершара с радиусом 0.5.
- `monit` — параметр режима мониторинга. Если равен единице, на экран в реальном времени выводится степень завершенности вычислений в процентах.

§5. Вычислительный эксперимент

Приведем данные вычислительного эксперимента со следующими входными данными:

- P — от 1 до 1000.
- $f(x) \equiv 1$.
- n — от 2 до 10.
- M — от 2 до 6.
- $\Phi(x) = 1 - \sum\limits_{i=0}^{n-1} (2x_i - 1)^2$
- N — от 10 до 10^5.
- $eps1 = 0.2,\ eps2 = 0.5$.
- `top_psi`$= 6$, `mid_psi`$= 0.3$.
- `eps_secant`$= 10^{-15}$.
- Параметры `tol` и `ta` отсутствуют, за верный ответ берется объем n-мерного гипершара с радиусом 0.5.

Такой набор параметров позволяет, во-первых, оценить реальную алгоритмическую погрешность вычислений — без влияния погрешности вычисления функции $f(x)$, а во-вторых, использовать в качестве эталонных

ответов известные значения объемов n-мерных шаров с радиусами 0.5, вычисленные с высокой точностью:

$$\begin{aligned}
&n = 2: \ ta = 0.78539816339744830962 \\
&n = 3: \ ta = 0.52359877559829887309 \\
&n = 4: \ ta = 0.30842513753404245685 \\
&n = 5: \ ta = 0.16449340668482264365 \\
&n = 6: \ ta = 0.080745512188280781712 \\
&n = 7: \ ta = 0.036912234143214071640 \\
&n = 8: \ ta = 0.015854344243815500853 \\
&n = 9: \ ta = 0.006442402006615368546 \\
&n = 10: \ ta = 0.002490394570192720 1601
\end{aligned} \tag{I.89}$$

Имеем четыре изменяющихся параметра: $P,\ n,\ M$ и N. Будем исследовать точность и скорость вычислений при различных комбинациях этих параметров.

Как показали эксперименты, точность вычислений не зависит от количества процессоров P, поэтому будем варьировать значения n, M и N.

Результаты вычислений и погрешностей при $n = 2,\ 3,\ 4,\ 10$ приведены в таблицах I.1 – I.8.

Анализ результатов счета показывает, что точность вычислений во многих случаях не уступает теоретической, а порой даже на несколько порядков превосходит ее. Исключение составляют случаи, когда либо M велико при малом N (т.е. слишком широк пограничный слой толщины $2Mh$), либо, когда опять же при больших M точность снижается из-за больших ошибок округления из-за конечной точности типа данных `double` — 15 верных цифр. С ростом размерности увеличивается количество множителей и слагаемых, из которых составляются срезывающие функции, что также ухудшает точность. Однако «пиковый» эксперимент с десятимерным интегралом показывает, что при малом M можно добиться удовлетворительного результата даже при $N = 10$.

Перейдем теперь к анализу быстроты счета и качества распараллеливания программы. Рассмотрим, для примера, результаты счета для $n = 3, M = 2, N = 1000$.

Для анализа качества распараллеливания введем понятия ускорения и эффективности программы:

Таблица I.1. n=2. Результаты вычислений

$N \setminus M$	2	3	4
100	0.785401080578356	0.785444570997023	0.785166984064381
1000	0.785398167437040	0.785398163381368	0.785398163398298
2000	0.785398163897492	0.785398163395937	0.785398163397469
3000	0.785398163545688	0.785398163397109	0.785398163397450
4000	0.785398163460001	0.785398163397316	0.785398163397449
5000	0.785398163429596	0.785398163397400	0.785398163397450
10000	0.785398163401474	0.785398163397448	0.785398163397448

$N \setminus M$	5	6
100	0.784763292469021	0.791580805056431
1000	0.785398163397452	0.785398163397454
2000	0.785398163397450	0.785398163397450
3000	0.785398163397448	0.785398163397448
4000	0.785398163397447	0.785398163397447
5000	0.785398163397448	0.785398163397449
10000	0.785398163397448	0.785398163397448

Таблица I.2. n=2. Погрешности вычислений

$N \setminus M$	2	3	4
100	2.92E-06	4.64E-05	2.31E-04
1000	4.04E-09	1.61E-11	8.50E-13
2000	5.00E-10	1.51E-12	2.04E-14
3000	1.48E-10	3.40E-13	1.55E-15
4000	6.26E-11	1.33E-13	3.33E-16
5000	3.21E-11	4.87E-14	1.33E-15
10000	4.03E-12	0.00E+00	1.11E-16

$N \setminus M$	5	6
100	6.35E-04	6.18E-03
1000	4.00E-15	5.44E-15
2000	1.78E-15	2.22E-15
3000	4.44E-16	0.00E+00
4000	1.78E-15	1.44E-15
5000	2.22E-16	5.55E-16
10000	1.11E-16	1.11E-16

Таблица I.3. n=3. Результаты вычислений

$N \setminus M$	2	3	4
100	0.523603551778959	0.523610585402113	0.523501930562029
200	0.523599159101737	0.523598770175491	0.523598774322774
300	0.523598888234927	0.523598774947450	0.523598775643928
400	0.523598823360721	0.523598775585911	0.523598775546532
500	0.523598797748788	0.523598775584992	0.523598775674013
600	0.523598790442605	0.523598775615717	0.523598775631787
700	0.523598784071800	0.523598775499833	0.523598775579111
1000	0.523598778405505	0.523598775587932	0.523598775598429

$N \setminus M$	5	6
100	0.523730286599139	0.525265551878725
200	0.523598776585500	0.523598826784667
300	0.523598775311062	0.523598775249301
400	0.523598775651952	0.523598775678095
500	0.523598775552826	0.523598775517708
600	0.523598775613498	0.523598775611494
700	0.523598775591087	0.523598775589617
1000	0.523598775599847	0.523598775599947

Таблица I.4. n=3. Погрешности вычислений

$N \setminus M$	2	3	4
100	4.78E-06	1.18E-05	9.68E-05
200	3.84E-07	5.42E-09	1.28E-09
300	1.13E-07	6.51E-10	4.56E-11
400	4.78E-08	1.24E-11	5.18E-11
500	2.22E-08	1.33E-11	7.57E-11
600	1.48E-08	1.74E-11	3.35E-11
700	8.47E-09	9.85E-11	1.92E-11
1000	2.81E-09	1.04E-11	1.31E-13

$N \setminus M$	5	6
100	1.32E-04	1.67E-03
200	9.87E-10	5.12E-08
300	2.87E-10	3.49E-10
400	5.37E-11	7.98E-11
500	4.55E-11	8.06E-11
600	1.52E-11	1.32E-11
700	7.21E-12	8.68E-12
1000	1.55E-12	1.65E-12

Таблица I.5. n=4. Результаты вычислений

$N \setminus M$	2	3	4
60	0.308492451782825	0.308073371939551	0.307338274735536
70	0.308429570509967	0.308288625420327	0.308927068272527
80	0.308417084282851	0.308415000205045	0.308530884676604
90	0.308419331845900	0.308436394045568	0.308389314273512
100	0.308418465553032	0.308422955211545	0.308402657792097
150	0.308424149252623	0.308424371728451	0.308426636128291
200	0.308424627908327	0.308424429022102	0.308423981551464
250	0.308424841730835	0.308424880923024	0.308425288950057

$N \setminus M$	5	6
60	0.313892145335748	0.336673729565596
70	0.309464821245380	0.300898734230410
80	0.307610238098610	0.307164583849154
90	0.308267885396172	0.309619223158653
100	0.308531966345698	0.308677128943248
150	0.308441119239107	0.308484383543192
200	0.308424319099250	0.308423714550707
250	0.308426836272804	0.308431355761822

Таблица I.6. n=4. Погрешности вычислений

$N \setminus M$	2	3	4
60	4.78E-06	1.18E-05	9.68E-05
70	3.84E-07	5.42E-09	1.28E-09
80	1.13E-07	6.51E-10	4.56E-11
90	4.78E-08	1.24E-11	5.18E-11
100	2.22E-08	1.33E-11	7.57E-11
150	1.48E-08	1.74E-11	3.35E-11
200	8.47E-09	9.85E-11	1.92E-11
250	2.81E-09	1.04E-11	1.31E-13

$N \setminus M$	5	6
60	1.32E-04	1.67E-03
70	9.87E-10	5.12E-08
80	2.87E-10	3.49E-10
90	5.37E-11	7.98E-11
100	4.55E-11	8.06E-11
150	1.52E-11	1.32E-11
200	7.21E-12	8.68E-12
250	1.55E-12	1.65E-12

Таблица I.7. n=10. Результаты вычислений

$N \setminus M$	2
10	0.002448461315145
11	0.002375399819127
12	0.002575455835213

$$S_P = \frac{T_1}{T_P} \qquad \text{(ускорение)},$$

$$E_P = \frac{S_P}{P} \qquad \text{(эффективность)},$$

где T_P — время, за которое задача выполняется на P процессорах. Будем варьировать P от 100 до 1000 с шагом 100. При этом положим $T_1 = 100\,T_{100}$. Дело в том, что реальное T_1 слишком велико, поэтому фактически идеальным случаем для нас будет случай, когда $P = 100$. Данные экспериментов отражены в табл. I.9, «идеальные» данные — в табл. I.10. На графике (рис. I.12) наглядно показано отклонение экспериментального ускорения S_P (темная ломаная) от идеального ускорения (светлая прямая).

Как видно, эффективность распараллеливания близка к идеальной, следовательно, в использовании большого числа процессоров есть практическая выгода.

Таблица I.8. n=10. Погрешности вычислений

$N \setminus M$	2
10	4.19E-05
11	1.15E-04
12	8.51E-05

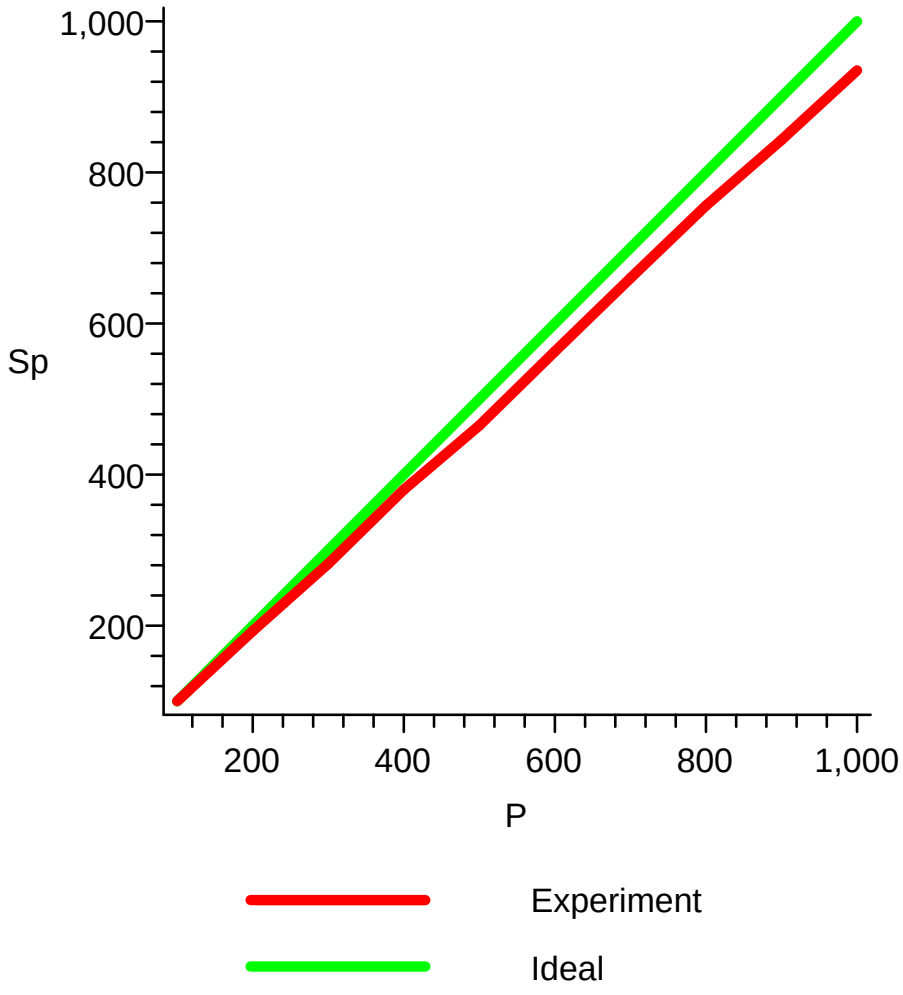

Рис. I.12. Зависимость ускорения от числа процессоров (n=3)

Таблица I.9. $n = 3,\ M = 2,\ N = 1000$ (эксперимент)

P	100	200	300	400	500	600	700	800	900	1000
T_P	135.0	69.9	47.9	35.5	29.0	24.0	20.5	17.9	16.0	14.4
S_P	100.0	193.0	282.0	380.0	465.0	563.0	660.0	756.0	843.0	935.0
E_P	1.00	0.97	0.94	0.95	0.93	0.94	0.94	0.95	0.94	0.94

Таблица I.10. $n = 3,\ M = 2,\ N = 1000$ (идеальный случай)

P	100	200	300	400	500	600	700	800	900	1000
T_P	135.0	67.5	45.0	33.8	27.0	22.5	19.3	16.9	15.0	13.5
S_P	100.0	200.0	300.0	400.0	500.0	600.0	700.0	800.0	900.0	1000.0
E_P	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Глава II. Наилучший порядок приближения интегралов функций из $W_2^{\overline{m}}(\mathbb{R}^2)$

§1. Постановка задачи

Во втором разделе диссертации приводится теоретический результат, полученный совместно М. Д. Рамазановым и Д. Я. Рахматуллиным [29, 35]. В отличие от пространства $\widetilde{W}_p^m(\Omega)$ функций, имеющих одинаковую по всем направлениям гладкость m, здесь рассматривается неизотропное пространство $W_2^{\overline{m}}(\mathbb{R}^2)$, $\overline{m} = (m_1, m_2)$ размерности 2. Исследуются последовательности кубатурных формул, имеющие на этом пространстве не только оптимальный порядок сходимости, но и *наилучший*.

Последовательность кубатурных формул

$$K_N^{\Omega,best}(f) \equiv \sum_{k=1}^{N} c_{k,N} f(x^{(k)}),\ N \to \infty \tag{II.1}$$

где функции f принадлежат банаховому пространству B^*, называется *наилучшей*, если при любом фиксированном N она минимизирует норму ее разности с точным значением интеграла I^Ω как по набору коэффициентов $\{c_{k,N}\}$, так и по расположению узлов $\{x^{(k)}\}$:

$$K_N^{\Omega,best} \stackrel{def}{=} \arg \min_{K_N^\Omega} \left\|K_N^\Omega - I^\Omega\right\|_{B^*} \equiv \arg \min_{\{x^{(k)}\},\{c_{k,N}\}} \left\|K_N^\Omega - I^\Omega\right\|_{B^*}. \tag{II.2}$$

ПКФ $K_N^{\Omega,bord}$ называется *наилучшей по порядку*, если выполняется неравенство:

$$\exists\, C:\ \overline{\lim_{N\to\infty}} \frac{\|I^\Omega - K_N^{\Omega,bord}\|_{B^*}}{\|I^\Omega - K_N^{\Omega,best}\|_{B^*}} \leqslant C. \tag{II.3}$$

Известен наилучший возможный порядок приближения в n-мерном пространстве $W_2^{\overline{m}}(\mathbb{R}^n)$, $\overline{m} = (m_1, \ldots, m_n)$ интегралов по ограниченной области с помощью кубатурных формул вида (II.1):

$$\left\|K_N^{\Omega,best}\right\|_{(W_2^{\overline{m}}(\mathbb{R}^n))^*} \equiv \min_{\{x^{(k)}\},\{c_{k,N}\}} \left\|K_N^\Omega\right\|_{(W_2^{\overline{m}}(\mathbb{R}^n))^*} = C\, N^{-\frac{1}{\sum_{j=1}^{n} \frac{1}{m_j}}}. \tag{II.4}$$

С исследованиями, проведенными в первом разделе диссертации, тему второго раздела роднит тот факт, что наилучший порядок может достигаться на **решетчатых** формулах. В частности, установлено [40, стр. 239], что такой порядок достигается, если взять векторный шаг решетки, подчиненный заданной гладкости подынтегральной функции. В направлениях большей гладкости следует брать бóльшие шаги решетки:

$$h \stackrel{def}{=} (h_1, \ldots, h_n), \quad h_1^{m_1} = \ldots = h_n^{m_n}. \tag{II.5}$$

Еще большее родство с темой первого раздела диссертации дает установленный результат. Оказывается, в двумерном случае наилучший порядок аппроксимации достигается и на **кубических** решетках за счет поворота на угол θ, тангенс которого есть «золотое сечение», $\alpha \stackrel{def}{=} \operatorname{tg}\theta = \frac{1+\sqrt{5}}{2}$. То есть следует взять последовательность решеток узлов (при $h \to 0$)

$$\{hHk\},\ k \in \mathbb{Z}^2,\ \text{с матрицей } H \stackrel{def}{=} \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \equiv \cos\theta \begin{pmatrix} 1 & \alpha \\ -\alpha & 1 \end{pmatrix}. \tag{II.6}$$

Отметим, что эта решетка узлов не зависит от параметров гладкости m_1 и m_2.

В следующем параграфе мы приведем строгую формулировку этого утверждения в виде теоремы и докажем его.

Таким образом, использование последовательностей кубических решеток оказывается удобным не только в смысле простоты их использования в компьютерных программах, но и в теоретически обоснованных хороших свойствах — на их основе можно строить кубатурные формулы наилучшие как на изотропных классах функций (Раздел 1), так и на неизотропных (Раздел 2).

§2. Теорема о достижении наилучшего порядка на решетчатых формулах

Докажем следующую теорему.

Теорема. *Решетчатые кубатурные формулы вида*

$$K_h^\Omega(f) \equiv \sum_{hHk \in \Omega} c_k(h) f(hHk), \quad h \to 0 \tag{II.7}$$

с узлами на кубической решетке узлов

$$\{hHk\},\ k\in\mathbb{Z}^2,\ H\stackrel{def}{=}\begin{pmatrix}\cos\theta & \sin\theta\\ -\sin\theta & \cos\theta\end{pmatrix}\equiv\cos\theta\begin{pmatrix}1 & \alpha\\ -\alpha & 1\end{pmatrix} \tag{II.8}$$

и оптимальные по порядку в неизотропном пространстве

$$W_2^{\overline{m}}(\mathbb{R}^2),\quad \overline{m}=(m_1,m_2),\quad \min\{m_1,m_2\}\geqslant\frac{3}{2}$$

являются в нем ***наилучшими по порядку***.

2.1. Преобразование постановки задачи

Введем в рассмотрение пространства $W_2^{\mu}\left(\mathbb{R}^2\right)$ и $\widetilde{W}_2^{\mu}(Q)$ с

$$\mu(\xi)\stackrel{def}{=}\nu(H\xi),\ \text{где}\ \nu(\eta)\stackrel{def}{=}\sqrt{1+|\eta_1|^{2m_1}+|\eta_2|^{2m_2}},\ \text{и нормами}$$

$$\|f\|_{W_2^{\mu}(\mathbb{R}^2)}\stackrel{def}{=}\left(\int\limits_{\mathbb{R}^2}d\xi\ \left|\tilde{f}(\xi)\,\mu(\xi)\right|^2\right)^{\frac{1}{2}},$$

$$\|f\|_{\widetilde{W}_2^{\mu}(Q)}\stackrel{def}{=}\left(\int\limits_{Q}d\xi\ \left|\sum_{k\in\mathbb{Z}^2}f_k\,\mu(k)\,e^{2\pi i\langle\xi,k\rangle}\right|^2\right)^{\frac{1}{2}},$$

где

$$Q\stackrel{def}{=}\{x=(x_1,\,x_2):\ x_i\in[-\frac{1}{2},\frac{1}{2}),\ i=1,2\,\},$$

$\tilde{f}(\xi)\stackrel{def}{=}\int\limits_{\mathbb{R}^2}dx\,f(x)\,e^{-2\pi i\langle x,\xi\rangle}$ — преобразование Фурье функции $f(x)$, а $f_k\stackrel{def}{=}\int\limits_{\mathbb{R}^2}dx\,f(x)\,e^{-2\pi i\langle x,k\rangle}$ — её коэффициенты Фурье. За $W_2^{\overline{m}}(\mathbb{R}^2)$ мы принимаем $W_2^{\nu}\left(\mathbb{R}^2\right)$. Далее удобно считать $m_2\geqslant m_1>1$.

Пусть $\overline{\Omega}$ обладает кусочно-гладкой границей и лежит в круге

$$\{x:|x|<\phi<1\}$$

Функционал погрешности кубатурной формулы с решеткой узлов $\{hHk\},k\in\mathbb{Z}^2$ имеет вид

$$l_h^{\Omega,\,H}(f)\stackrel{def}{=}\int\limits_{\Omega}dx\ f(x)-h^2\sum_{\substack{k\in\mathbb{Z}^2,\\ hHk\in\Omega}}c_k(h)\,f(hHk). \tag{II.9}$$

Заметим, что

$$\left\|l_h^{\Omega,H}\right\|_{(W_2^{\overline{m}}(\mathbb{R}^2))^*} = \left(\int\limits_{\mathbb{R}^2} d\eta\ \frac{\left|\tilde{l}_h^{\Omega,H}(\eta)\right|^2}{\nu^2(\eta)}\right)^{\frac{1}{2}} = |\eta = H\xi| =$$

$$= \left(\int\limits_{\mathbb{R}^2} d\xi\ \frac{\left|\tilde{l}_h^{H^{-1}\Omega,I}(\xi)\right|^2}{\mu^2(\xi)}\right)^{\frac{1}{2}} = \left\|l_h^{H^{-1}\Omega,I}\right\|_{(W_2^{\mu}(\mathbb{R}^2))^*}.$$

На функциях из $C_0^{m_2}(|x| \leqslant \sigma < 1)$ нормы пространств $W_2^{\mu}\left(\mathbb{R}^2\right)$ и $\widetilde{W}_2^{\mu}(Q)$ эквивалентны равномерно по всевозможным матрицам поворотов $HH^T = I$, то есть

$$\exists C\ \ \forall f \in C_0^{m_2}(|x| \leqslant \sigma < 1) \quad \frac{1}{C}\|f\|_{\widetilde{W}_2^{\mu}(Q)} \leqslant \|f\|_{W_2^{\mu}(\mathbb{R}^2)} \leqslant C\|f\|_{\widetilde{W}_2^{\mu}(Q)}.$$

Тогда

$$C\|l\|_{\widetilde{W}_2^{\mu}(Q)} \geqslant \|l\|_{W_2^{\mu}(\mathbb{R}^2)} \geqslant \frac{1}{C}\|l\|_{\widetilde{W}_2^{\mu}(Q)},$$

а значит, для любых коэффициентов $\{c_k\}$ и любой матрицы поворота H норма $\left\|l_h^{H^{-1}\Omega,I}\right\|_{(W_2^{\mu}(\mathbb{R}^2))^*}$ имеет двусторонюю оценку через $\left\|l_h^{H^{-1}\Omega,I}\right\|_{(\widetilde{W}_2^{\mu}(Q))^*}$.

Для оптимального по порядку функционала $l_h^{\omega,I}$ на пространстве $\widetilde{W}_2^{\mu}(Q)$ выполняется оценка [40, стр. 138, 229]

$$\exists C: \qquad \frac{1}{C} \leqslant \frac{\left\|l_h^{\omega,I}\right\|_{(\widetilde{W}_2^{\mu}(Q))^*}}{\|l_h^{\infty}\|_{(\widetilde{W}_2^{\mu}(Q))^*}} \leqslant C. \tag{II.10}$$

Здесь $l_h^{\infty}(x) \stackrel{def}{=} \chi_Q(x) - h^2 \sum\limits_{\substack{k\in\mathbb{Z}^2,\\ hk\in Q}} \delta(x - hk)$.

Поэтому вычисление порядка достаточно провести для однородного функционала погрешности $l_h^{\infty}(x)$. Мы покажем, что при $\alpha \stackrel{def}{=} \operatorname{tg}\theta = \frac{1+\sqrt{5}}{2}$

$$\exists c: \qquad \|l_h^{\infty}\|_{(\widetilde{W}_2^{\mu}(Q))^*} \leqslant c\, h^{\frac{2}{\frac{1}{m_1}+\frac{1}{m_2}}} \tag{II.11}$$

Вместе с формулой $(II.4)$ это означает, что оптимальный порядок кубатурной формулы с функционалом погрешности $(II.9)$ при $H = \cos\theta \begin{pmatrix} 1 & \alpha \\ -\alpha & 1 \end{pmatrix}$ является наилучшим.

2.2. Оценка нормы однородного функционала

Рассмотрим $\|l_h^\infty\|^2_{(\widetilde{W}_2^\mu(Q))^*} = \sum\limits_{k\in\mathbb{Z}^2\setminus\{0\}} \dfrac{1}{\mu^2(\frac{k}{h})} = \sum\limits_{k\in\mathbb{Z}^2\setminus\{0\}} \dfrac{1}{\nu^2(H\frac{k}{h})}$, где

$$\nu(\eta) = \sqrt{1 + |\eta_1|^{2m_1} + |\eta_2|^{2m_2}}, \qquad 1 < m_1 \leqslant m_2.$$

Взяв за новое h прежнее, поделённое на $\cos\theta$, будем оценивать порядок

$$\zeta(h) \stackrel{def}{=} \sum_{k\in\mathbb{Z}^2\setminus\{0\}} \frac{1}{1 + \left|\frac{k_1+\alpha k_2}{h}\right|^{2m_1} + \left|\frac{-\alpha k_1+k_2}{h}\right|^{2m_2}} \qquad \text{при } h \to 0.$$

Вначале «отрежем» окрестность бесконечности плоскости (k_1, k_2). Подберем такое $K(h)$, чтобы выполнялось

$$\sum_{\substack{k\in\mathbb{Z}^2\setminus\{0\},\\ |k|\geqslant K(h)}} \frac{1}{\nu^2(H\frac{k}{h})} = O\left(h^{\frac{4m_1m_2}{m_1+m_2}}\right) = O\left(N^{-\frac{2m_1m_2}{m_1+m_2}}\right).$$

Для этого достаточно, чтобы

$$\int\limits_{|k|\geqslant K(h)} dk\, \frac{1}{\nu^2(H\frac{k}{h})} = O\left(h^{\frac{4m_1m_2}{m_1+m_2}}\right).$$

Заметим, что $\dfrac{1}{\nu^2\left(H\,\frac{\xi}{h}\right)} \leqslant \dfrac{C}{1 + \left|\frac{\xi}{h}\right|^{2m_1}}$. Используя это неравенство и переходя в полярные координаты, получаем:

$$\sum_{\substack{k\in\mathbb{Z}^2\setminus\{0\},\\ |k|\geqslant K(h)}} \frac{1}{\nu^2(H\frac{k}{h})} \asymp \int\limits_{|\rho|\geqslant K(h)} d\rho\, \frac{\rho}{1 + \left(\frac{\rho}{h}\right)^{2m_1}} \asymp$$

$$\asymp h^{2m_1} \int\limits_{|\rho|\geqslant K(h)} d\rho\, \rho^{-(2m_1-1)} = h^{2m_1}(K(h))^{-2m_1+2}.$$

Здесь и далее знак $\asymp$ означает существование двусторонней оценки.

Последнее выражение оценивается сверху через $h^{\frac{4m_1m_2}{m_1+m_2}}$ при

$$K(h) \geqslant C\, h^{-\frac{1-\frac{m_1}{m_2}}{\left(1+\frac{m_1}{m_2}\right)\left(1-\frac{1}{m_1}\right)}}.$$

Отбросив, для удобства, константу, положим

$$K(h) = h^{-\frac{1-\frac{m_1}{m_2}}{\left(1+\frac{m_1}{m_2}\right)\left(1-\frac{1}{m_1}\right)}}.$$

Далее будем рассматривать только те точки плоскости, для которых

$$k \in \mathbb{K} \stackrel{def}{=} \{k \in \mathbb{Z}^2 : 1 \leqslant |k| \leqslant K(h)\}.$$

Выделим из суммы

$$\psi(h) \equiv \sum_{k \in \mathbb{K}} \frac{1}{1 + \left|\frac{k_1+\alpha k_2}{h}\right|^{2m_1} + \left|\frac{-\alpha k_1+k_2}{h}\right|^{2m_2}}$$

группу слагаемых, для которой требуемая оценка получается сравнительно просто.

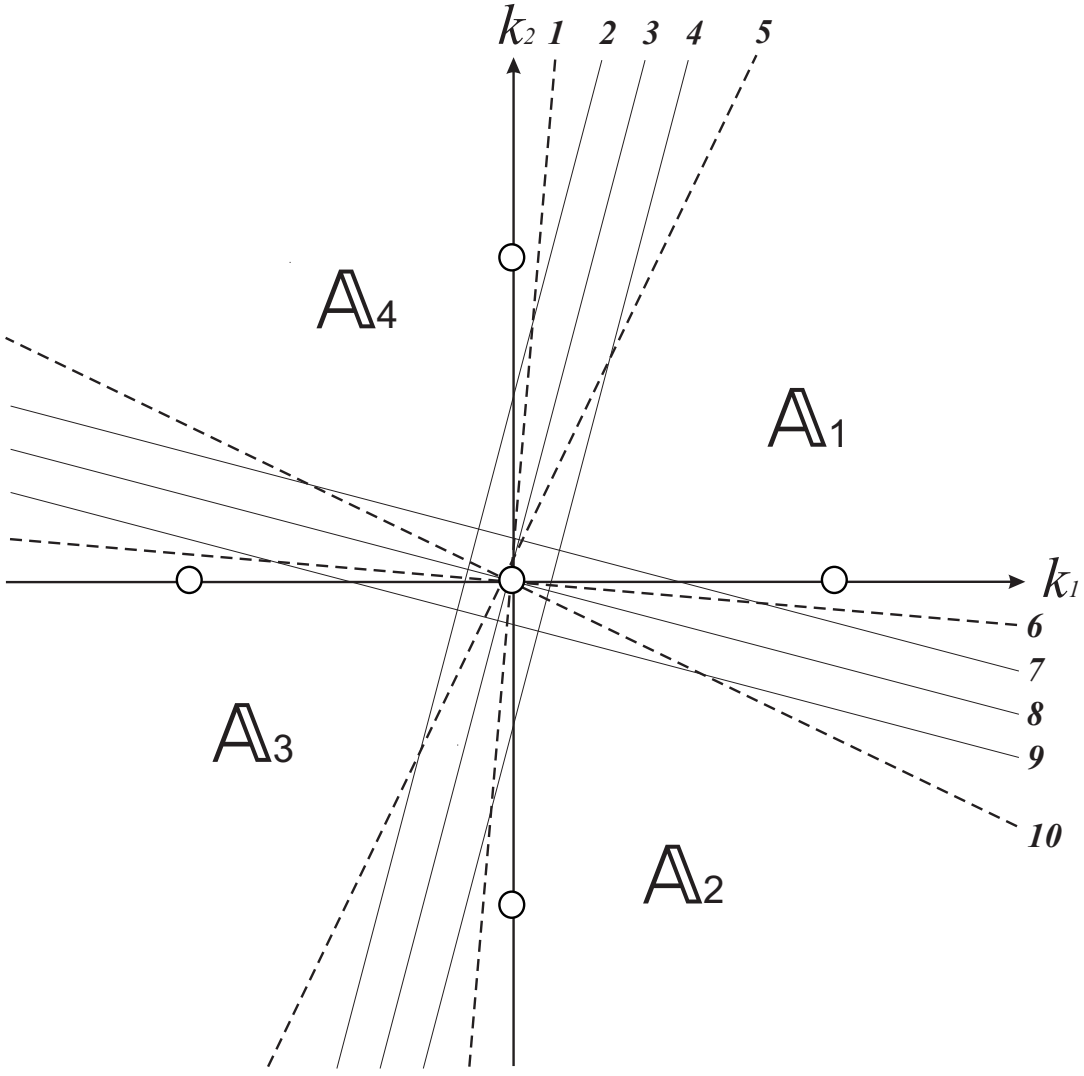

Рис. II.1. Разбиение плоскости

Мы имеем ввиду часть суммы, индексы слагаемых которых попадают в обозначенные на рисунке большие области $\mathbb{A}_1, \mathbb{A}_2, \mathbb{A}_3, \mathbb{A}_4$, ограниченные пунктирными линиями.

Запишем, для ясности, уравнения прямых, обозначенных на рисунке цифрами:

1 – $k_2 = (\alpha+\delta)k_1$,
2 – $k_2 = \alpha k_1 + \frac{1}{2}$,
3 – $k_2 = \alpha k_1$,
4 – $k_2 = \alpha k_1 - \frac{1}{2}$,
5 – $k_2 = (\alpha-\delta)k_1$,
6 – $k_1 = -(\alpha+\delta)k_2$,
7 – $k_1 = -\alpha k_2 + \frac{1}{2}$,
8 – $k_1 = -\alpha k_2$,
9 – $k_1 = -\alpha k_2 - \frac{1}{2}$,
10 – $k_1 = -(\alpha-\delta)k_2$,

Все четыре области $\mathbb{A}_1, \mathbb{A}_2, \mathbb{A}_3, \mathbb{A}_4$ оцениваются одинаково. Для примера займемся оценкой суммы со множеством индексов

$$\mathbb{A}_1 = \{k : k_1 \in \mathbb{K}_1 \stackrel{def}{=} \{k_1 : 1 \leqslant k_1 \leqslant K(h)\} \cap \mathbb{Z},\ k_2 \in [-\frac{1}{\alpha+\delta}k_1, (\alpha-\delta)k_1] \cap \mathbb{Z}\},$$

где δ — достаточно маленькое положительное число.

$$\sum_{k\in\mathbb{A}_1} \frac{1}{1+\left(\frac{k_1+\alpha k_2}{h}\right)^{2m_1} + \left|\frac{-\alpha k_1+k_2}{h}\right|^{2m_2}} \asymp$$

$$\asymp \sum_{k_1\in\mathbb{K}_1} \int\limits_{-\frac{1}{\alpha+\delta}k_1}^{(\alpha-\delta)k_1} dk_2 \ \frac{1}{1+\left(\frac{k_1+\alpha k_2}{h}\right)^{2m_1} + \left|\frac{-\alpha k_1+k_2}{h}\right|^{2m_2}} =$$

$$= |-\tau = k_2 - \alpha k_1| = \sum_{k_1\in\mathbb{K}_1} \int\limits_{\delta k_1}^{(\alpha+\frac{1}{\alpha+\delta})k_1} d\tau \ \ \frac{1}{1+\left(\frac{(1+\alpha^2)k_1-\alpha\tau}{h}\right)^{2m_1} + \left(\frac{\tau}{h}\right)^{2m_2}} \leqslant$$

$$\leqslant \left| \begin{array}{c} \alpha\tau \leqslant (\alpha^2+1-\alpha\delta)k_1, \\ \alpha\tau - (1+\alpha^2)k_1 \leqslant -\alpha\delta k_1, \\ |(1+\alpha^2)k_1 - \alpha\tau| \geqslant \delta\alpha k_1 \end{array} \right| \leqslant \sum_{k_1\in\mathbb{K}_1} \int\limits_{\delta k_1}^{(\alpha+\frac{1}{\alpha+\delta})k_1} d\tau \ \ \frac{1}{1+\left(\frac{\alpha\delta k_1}{h}\right)^{2m_1} + \left(\frac{\tau}{h}\right)^{2m_2}} \leqslant$$

$$\leqslant \left| \begin{array}{c} a^{2m_2} \stackrel{def}{=} 1+\left(\frac{\alpha\delta k_1}{h}\right)^{2m_1}, \\ \tau = hta \end{array} \right| \leqslant \sum_{k_1\in\mathbb{K}_1} \int\limits_{\frac{\delta k_1}{ha}}^{\frac{1}{ha}(\alpha+\frac{1}{\alpha+\delta})k_1} dt \ \ \frac{ha}{a^{2m_2}\left(1+t^{2m_2}\right)} =$$

$$= \left| \text{Пусть } \frac{\delta k_1}{ha} \geq 1 \right| \asymp \sum_{k_1 \in \mathbb{K}_1} ha^{1-2m_2} \left(\frac{\delta k_1}{ha} \right)^{-2m_2+1} = \sum_{k_1 \in \mathbb{K}_1} h^{2m_2} \left(\delta k_1 \right)^{-2m_2+1} =$$

$$= \left| A \stackrel{def}{=} \sum_{k_1 \in \mathbb{K}_1} \left(\delta k_1 \right)^{-2m_2+1}, \; 2m_2 \geqslant \frac{4m_1 m_2}{m_1 + m_2} \right| \leqslant A \, h^{\frac{4m_1 m_2}{m_1+m_2}}.$$

Остаётся оценить сумму по слагаемым с индексами в узких областях-углах. Рассмотрим множество индексов

$$\mathbb{B}_1 \stackrel{def}{=} \{k : k_1 \in \mathbb{K}_1, k_2 \in [(\alpha - \delta)k_1, \alpha k_1 - \frac{1}{2}] \cap \mathbb{Z}\}.$$

При этом $k_1 \geqslant \frac{1}{2\delta}$. Пусть $\mathbb{K}_4 \stackrel{def}{=} \mathbb{K}_1 \cap \{k_1 : k_1 \geqslant \frac{1}{2\delta}\}$. Произведя замены и преобразования, аналогичные проделанным выше, получим:

$$\sum_{k \in \mathbb{B}_1} \frac{1}{1 + \left(\frac{k_1 + \alpha k_2}{h}\right)^{2m_1} + \left(\frac{-\alpha k_1 + k_2}{h}\right)^{2m_2}} \asymp \sum_{k_1 \in \mathbb{K}_4} h \, a^{1-2m_2} \left(\frac{1}{2ha} \right)^{-2m_2+1} =$$

$$= c \, h^{2m_2} \sum_{k_1 \in \mathbb{K}_4} 1 = c \, h^{2m_2} \left(K(h) - \frac{1}{2\delta} \right) \leqslant c \, h^{2m_2 - \frac{1 - \frac{m_1}{m_2}}{\left(1 + \frac{m_1}{m_2}\right)\left(1 - \frac{1}{m_1}\right)}} \equiv L(h).$$

Достаточным условием выполнения оценки $L(h) = O\left(h^{\frac{4m_1 m_2}{m_1+m_2}}\right)$ является, например,

$$m_1 \geqslant \frac{3}{2}.$$

Остается полоска области индексов около прямых $k_2 = \alpha k_1$ и $k_1 = -\alpha k_2$. Рассмотрим первую из них:

$$\mathbb{C}_1 \stackrel{def}{=} \{k : k_1 \in \mathbb{K}_1, \, k_2 \in [\alpha k_1 - \frac{1}{2}, \, \alpha k_1] \cap \mathbb{Z}\}.$$

Сразу заметим, что двойная сумма по этой области превратится в одинарную, т.к. каждому k_1 соответствует не более одного значения k_2. А именно, это такое k_2, для которого $0 \leqslant \alpha k_1 - k_2 \leqslant \frac{1}{2}$ или $\alpha_1 k_1 - k_2 = (\alpha k_1)$, где $(\,\cdot\,)$ — расстояние до ближайшей целой точки.

Величины $k_1 + \alpha k_2$ при $k_1 \in [1, K(h)]$, не стремятся к нулю при уменьшении h. При этом, $(\alpha + 1)k_1 \leqslant k_1 + \alpha k_2 \leqslant (\alpha + 2)k_1 \Rightarrow k_1 + \alpha k_2 \asymp k_1$. Поэтому достаточно оценить сверху сумму

$$S(h) \stackrel{def}{=} \sum_{k_1 \in \mathbb{K}_1} \frac{1}{1 + \left(\frac{k_1}{h}\right)^{2m_1} + \left(\frac{(\alpha k_1)}{h}\right)^{2m_2}}.$$

С другой стороны, величина (αk_1) с изменением k_1 может сколь угодно близко подходить к нулю, поэтому оценку $S(h)$ придется провести максимально точно.

Нам понадобятся некоторые формулы для последовательностей чисел Фибоначчи

$$\{u_j\}_{j=1}^{\infty},\ u_1 = u_2 = 1,\ u_{j+1} = u_j + u_{j-1},\ j \geqslant 2.$$

Рассмотрим квадратное уравнение $x^2 - x - 1 = 0$. Любое число Фибоначчи можно выразить через корни этого уравнения $\alpha = \frac{1+\sqrt{5}}{2}$ и $\beta = \frac{1-\sqrt{5}}{2}$:

$$u_j = \frac{\alpha^j - \beta^j}{\sqrt{5}}, \quad j \geqslant 1.$$

Вторым важным свойством последовательности $\{u_j\}_{j=1}^{\infty}$ является то, что ее можно использовать для взаимно однозначной записи натуральных чисел по следующему правилу:

$$n = \sum_{j=2}^{J(n)} \varepsilon_j u_j, \text{ где } u_{J(n)} \in \left(\frac{n}{2}, n\right];\ \varepsilon_{J(n)} = 1;$$

$$\varepsilon_j \in \{0, 1\}\ \forall j \geqslant 2;\ \varepsilon_j\, \varepsilon_{j-1} = 0,\ \forall j \geqslant 2.$$

Оценим $(\alpha k_1),\ \ k \in \mathbb{C}_1$. Имеем:

$$(\alpha k_1) = \alpha k_1 - k_2 = \alpha \sum_{j=j_1}^{J} \varepsilon_j u_j - k_2 = \sum_{j=j_1}^{J} \varepsilon_j\, \alpha u_j - k_2 =$$

$$= \sum_{j=j_1}^{J} \varepsilon_j \frac{\alpha^{j+1} - \alpha\beta^j}{\sqrt{5}} - k_2 = \sum_{j=j_1}^{J} \varepsilon_j \frac{\beta^{j-1}(\beta^2+1)}{\sqrt{5}} + \alpha \sum_{j=2}^{J} \varepsilon_j u_{j+1} - k_2 =$$

$$= \sum_{j=j_1}^{J} \varepsilon_j \frac{\beta^{j-1}(\beta^2+1)}{\sqrt{5}} + Z = \left(\sum_{j=j_1}^{J} \varepsilon_j \frac{\beta^{j-1}(\beta^2+1)}{\sqrt{5}}\right) =$$

$$= \left(\sum_{j=j_1}^{J} \varepsilon_j\, \beta^{j-1} \frac{(\beta+2)}{1-2\beta}\right) = \left(-\sum_{j=j_1}^{J} \varepsilon_j\, \beta^j\right) = \left(\sum_{j=j_1}^{J} \varepsilon_j\, \beta^j\right).$$

где j_1 — наименьший индекс, такой что $\varepsilon_{j_1} = 1$. Порядок последнего выражения цепочки равенств при $k_1 \to \infty$ равен порядку выражения внутри

скобок $(\cdot)$, которое в свою очередь имеет порядок $|\beta|^{j_1}$ — наибольшего слагаемого. Следовательно, $(\alpha k_1) \asymp |\beta|^{j_1}$. Получим:

$$S(h) \asymp \sum_{k_1 \in \mathbb{K}_1} \frac{1}{1 + \left(\frac{k_1}{h}\right)^{2m_1} + \left(\frac{|\beta|^{j_1}}{h}\right)^{2m_2}} = \left| k_1 = \sum_{j=j_1(k_1)}^{J(k_1)} \varepsilon_j \, u_j \right| =$$

$$= \sum_{J=2}^{J(K(h))} \sum_{j_1=2}^{J} \sum_{\substack{\{\varepsilon_j\} \\ j \in [j_1, J]}} \frac{1}{1 + \left(\frac{\sum\limits_{j=j_1}^{J} \varepsilon_j \, u_j}{h}\right)^{2m_1} + \left(\frac{|\beta|^{j_1}}{h}\right)^{2m_2}} \asymp$$

$$\asymp \sum_{J \geqslant 2} \sum_{j_1 \in [2, J]} \frac{\alpha^{J - j_1}}{\left(\frac{\alpha^J}{h}\right)^{2m_1} + \left(\frac{|\beta|^{j_1}}{h}\right)^{2m_2}} =$$

$$= h^{2m_2} \sum_{J \geqslant 2} \sum_{j_1 \in [2, J]} \frac{\alpha^{J + j_1(2m_2 - 1)}}{\alpha^{2(Jm_1 + j_1 m_2)} h^{2(m_2 - m_1)} + 1} \asymp$$

$$\asymp C \, h^{2m_1 + 1 - \frac{m_1}{m_2} + (2m_1 - 1 + \frac{m_1}{m_2}) \frac{m_2 - m_1}{m_2 + m_1}}.$$

Несложно проверить, что последнее выражение оценивается сверху через $C \, h^{\frac{4m_1 m_2}{m_1 + m_2}}$.

Таким образом, при $m_2 \geqslant m_1 \geqslant \frac{3}{2}$ для решетки узлов $\{hHk\}$, $k \in \mathbb{Z}^2$, $hHk \in \Omega$, мы получаем искомую оценку сверху

$$\zeta(h) \equiv \sum_{k \in \mathbb{Z}^2 \setminus \{0\}} \frac{1}{1 + \left|\frac{k_1 + \alpha k_2}{h}\right|^{2m_1} + \left|\frac{-\alpha k_1 + k_2}{h}\right|^{2m_2}} \leqslant C \, h^{\frac{4m_1 m_2}{m_1 + m_2}} = C \, N^{-\frac{2m_1 m_2}{m_1 + m_2}}.$$

Сравнивая полученную оценку с формулой $(II.4)$ заключаем, что оптимальный порядок кубатурной формулы с функционалом погрешности $(II.9)$ при

$H = \cos\theta \begin{pmatrix} 1 & \alpha \\ -\alpha & 1 \end{pmatrix}$ является наилучшим. Теорема доказана.

§3. Вычислительный эксперимент

Для проверки хороших свойств квадратурных формул с повернутой решеткой узлов была создана программа (прил. 2), вычисляющая методом прямоугольников приближенный интеграл по квадрату

$$Q \stackrel{def}{=} \{x = (x_1, \, x_2) : \, x_i \in [-\frac{1}{2}, \frac{1}{2}), \, i = 1, 2 \},$$

на трех видах решеток: квадратной, прямоугольной и повернутой квадратной.

Подынтегральная функция $f(x_1, x_2)$ бралась в виде

$$f(x_1, x_2) \stackrel{def}{=} \sin(s_1 x_1)\sin(s_2 x_2)\hat{\varphi}(x_1)\hat{\varphi}(x_2), \tag{II.12}$$

где

$$\hat{\varphi}(x) \stackrel{def}{=} \hat{\xi}(2x)\hat{\xi}(2(1-x)), \tag{II.13}$$

$$\hat{\xi}(t) \stackrel{def}{=} \begin{cases} 0, \ t < 0 \\ \int\limits_0^t z(t)dt \Big/ \int\limits_0^1 z(t)dt, \quad z(t) = (t(1-t))^M, \qquad 0 \leqslant t < 1, \\ 1, \ 1 \leqslant t. \end{cases} \tag{II.14}$$

Графики функций $\hat{\xi}(t)$ и $\hat{\varphi}(x)$ при $M = 4$ приведены на рис. II.2 и II.3. Функция $\hat{\Phi}(x, y) \stackrel{def}{=} \hat{\varphi}(x)\hat{\varphi}(y)$ изображена на рис. II.4.

Выберем теперь параметры гладкости m_1 и m_2 неизотропного пространства $W_2^{\overline{m}}(\mathbb{R}^2)$. Из условия согласования шагов решетки (II.5) следует, что

$$N_1^{m_1} = N_2^{m_2}, \quad N_1 = 1/h_1, \ N_2 = 1/h_2. \tag{II.15}$$

Удобно взять $m_1 = 1$. Тогда из $N_2 \in \mathbb{Z} \Rightarrow N_1 \in \mathbb{Z}$.

Положим $m_2 = M$. Это согласуется с тем, что функция $\hat{\xi}(t)$, а следовательно и функция $f(x_1, x_2)$, непрерывно-дифференцируема ровно M раз.

Для того, чтобы смоделировать гладкость $m_1 = 1$ по переменной x_1, положим $s_1 = s_2^M$, обеспечивая тождество

$$\frac{\partial^{m_1}}{\partial x_1^{m_1}}(\sin(s_1 x_1)\sin(s_2 x_2)) \equiv \frac{\partial^{m_2}}{\partial x_2^{m_2}}(\sin(s_1 x_1)\sin(s_2 x_2)).$$

При $M = 4, \ s_2 = 5$ получаем функцию

$$f(x_1, x_2) = \sin(625 x_1)\sin(5 x_2)\hat{\varphi}(x_1)\hat{\varphi}(x_2), \tag{II.16}$$

Результаты вычислений показаны в табл. II.1, II.2 и II.3.

В первой колонке каждой таблицы указано количество узлов N. В зависимости от типа решетки оно «округляется» так, чтобы числа N_1 и N_2 были целыми. Для простой квадратной решетки $N = k^2, \ k \in \mathbb{Z}$. Для прямоугольной решетки $N = k^5, \ k \in \mathbb{Z}$, так как $N = N_1 N_2 = N_2^4 N_2 = N_2^5$. Для повернутой квадратной решетки приводится фактическое количество узлов, попавших в единичный квадрат после поворота решетки.

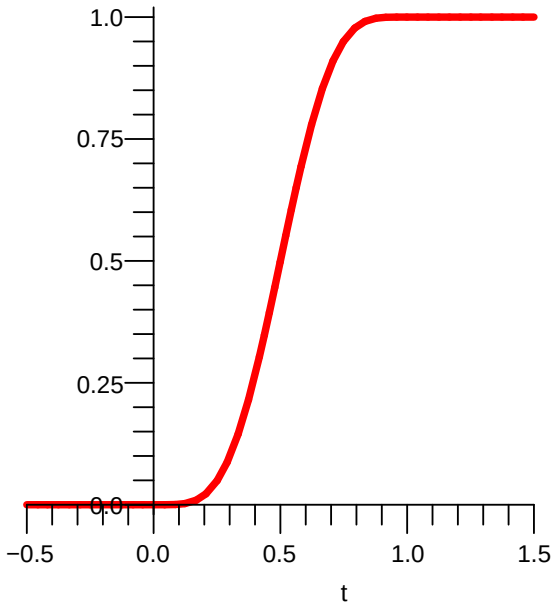

Рис. II.2. Функция $\hat{\xi}(t)$

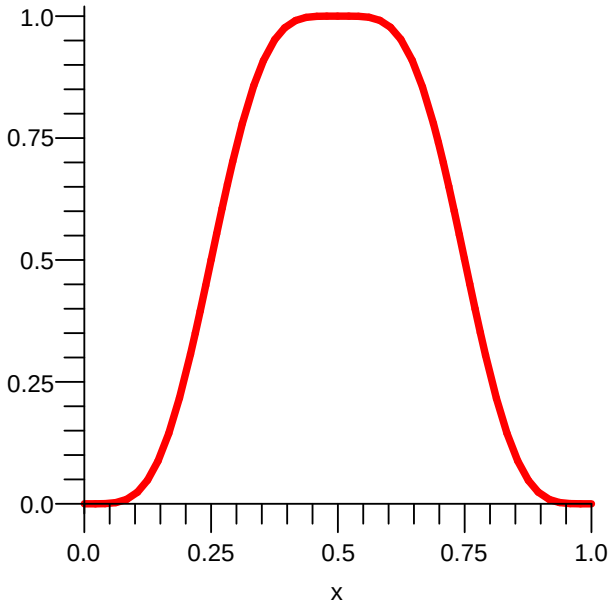

Рис. II.3. Функция $\hat{\varphi}(x)$

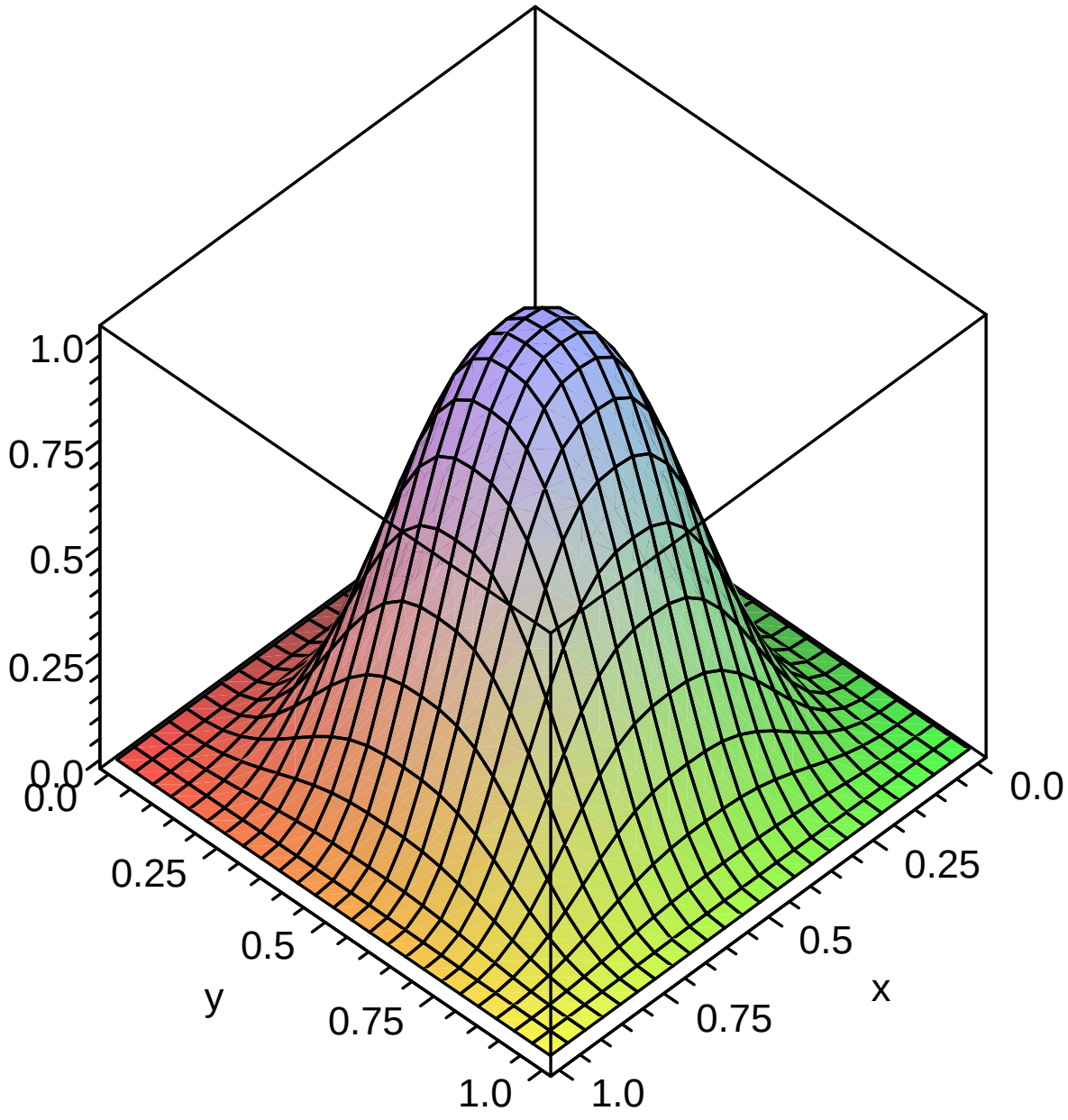

Рис. II.4. Функция $\hat{\Phi}(x, y)$

Таблица II.1. Расчеты для квадратной решетки

N	Абс. ошибка	Отн. ошибка
1024	3.0E-03	-8.7E+08
2025	-1.1E-06	3.0E+05
4096	-9.4E-09	2.7E+03
8281	1.2E-05	-3.6E+06
16384	-7.9E-09	2.3E+03
32761	9.4E-12	-2.7E+00
65536	-2.6E-13	7.3E-02
131044	-1.2E-14	3.4E-03
262144	-8.2E-16	2.3E-04
524176	-8.3E-17	2.4E-05
1048576	-1.7E-18	4.9E-07
2096704	3.6E-17	-1.0E-05
4194304	-1.1E-17	3.1E-06
8386816	7.9E-19	-2.3E-07
16777216	1.9E-17	-5.3E-06
33558849	-5.6E-18	1.6E-06
67108864	4.7E-17	-1.4E-05
134212225	8.5E-17	-2.4E-05
268435456	-9.9E-17	2.8E-05
536848900	1.5E-16	-4.4E-05

Таблица II.2. Расчеты для прямоугольной решетки

N	Абс. ошибка	Отн. ошибка
1024	4.6E-15	-1.3E-03
3125	1.7E-14	-4.9E-03
7776	-2.8E-15	8.0E-04
32768	-1.4E-15	4.0E-04
59049	2.5E-16	-7.2E-05
100000	-4.5E-16	1.3E-04
248832	-1.7E-16	4.9E-05
371293	1.8E-17	-5.2E-06
1048576	-3.3E-17	9.5E-06
1889568	-3.0E-18	8.5E-07
4084101	-8.6E-18	2.5E-06
7962624	1.0E-17	-2.8E-06
14348907	3.2E-17	-9.2E-06
33554432	-5.6E-17	1.6E-05
60466176	-9.8E-17	2.8E-05
130691232	-1.3E-16	3.6E-05
254803968	5.3E-17	-1.5E-05
503284375	4.3E-16	-1.2E-04

Таблица II.3. Расчеты для повернутой квадратной решетки

N	Абс. ошибка	Отн. ошибка
1024	8.9E-10	-2.5E+02
2024	6.6E-06	-1.9E+06
4096	9.8E-14	-2.8E-02
8280	-9.2E-16	2.6E-04
16384	-8.7E-16	2.5E-04
32761	2.2E-15	-6.3E-04
65536	-2.4E-18	7.0E-07
131044	1.4E-18	-4.1E-07
262144	-5.5E-19	1.6E-07
524176	-5.0E-19	1.4E-07
1048576	4.2E-20	-1.2E-08
2096704	-6.2E-19	1.8E-07
4194303	-8.3E-19	2.4E-07
8386816	-2.4E-19	7.0E-08
16777216	2.3E-20	-6.7E-09
33558849	3.1E-19	-8.8E-08
67108864	2.2E-20	-6.3E-09
134212224	-2.0E-19	5.8E-08
268435456	-5.8E-21	1.7E-09
536848899	-2.3E-20	6.5E-09

В двух последних колонках таблицы приводятся абсолютная и относительная погрешность вычислений. В качестве эталона берется самое точное из вычисленных значений: $-3.49795450960822E-12$. Символ E следует читать как "умножить на десять в степени".

Результаты счета графически изображены на рис. II.5. По оси абсцисс откладывается значение N в масштабе десятичного логарифма, по оси ординат - модуль относительной погрешности в том же масштабе.

График показывает, что начиная со значений $N \approx 10^4$, погрешность вычислений с повернутой решеткой (зеленая ломаная) на один-два порядка меньше, чем в случае с квадратной (красная линия) и прямоугольной (синяя линия) решетками, что подтверждает ее хорошие вычислительные свойства.

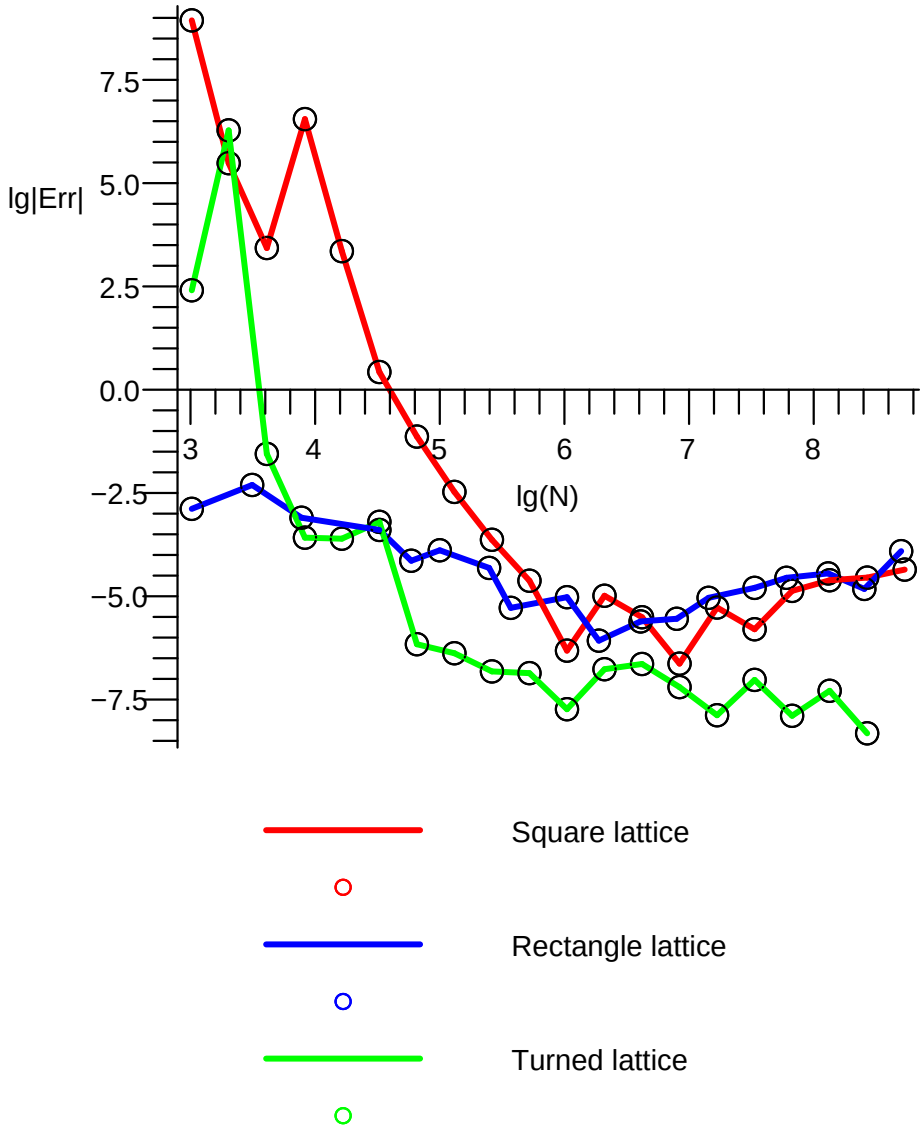

Рис. II.5. Сравнение погрешности вычислений при разных решетках

Заключение

Одной из важных проблем, возникающих при решении некоторых прикладных задач математической физики, квантовой химии, механики является приближенное интегрирование функций по многомерным областям с кривыми границами.

В диссертации предложен, реализован и проанализирован один из самых быстрых и точных способов приближенного интегрирования по многомерным выпуклым областям с гладкими границами по функциям различной гладкости (из пространства $\widetilde{W}_p^m(\Omega)$ с $m = 2..6$). За основу берется алгоритм М.Д. Рамазанова, дополненный, оптимизированный и реализованный в виде стандартной параллельной программы автором диссертации.

Полученный результат обладает как вычислительными достоинствами (точностью, быстродействием, эффективностью, хорошей масштабируемостью на произвольное количество процессоров, переносимостью на любые платформы, совместимые с библиотекой MPI), так и теоретическими — строгим обоснованием алгоритма и гарантированной оценкой погрешности вычислений. Теоретическое обоснование стало возможным благодаря достижениям теории кубатурных формул, развитой С.Л. Соболевым и продолженной его последователями, в частности М.Д. Рамазановым.

Результат является новым как по форме (используется язык программирования C++ вкупе со стандартной библиотекой параллельных функций MPI; программа предназначена для современных многопроцессорных вычислительных систем с распределенной памятью), так и по содержанию (алгоритм дополнен и оптимизирован).

Практическая польза результата заключается в создании реальной альтернативы имеющимся на сегодняшний день программам интегрирования для многомерных областей, в частности методам типа Монте-Карло.

Второй результат, изложенный в диссертации, является продолжением исследования хороших свойств кубатурных формул, построенных на основе последовательностей кубических решеток. М.Д. Рамазановым и автором диссертации совместно был установлен неожиданный факт достижения наилучшего порядка приближения интегралов в неизотропных про-

странствах (размерности 2) кубатурными формулами с кубической решеткой узлов, не зависящей от параметров гладкости, а лишь только повернутой на фиксированный угол.

Для практической проверки теоретического результата диссертантом была написана программа для сравнения точности вычислений с тремя видами решеток: обычной кубической решетки, прямоугольной решетки с векторным шагом, подчиненным параметрам гладкости пространства и кубической решетки, повернутой на специальный угол. Вычислительный эксперимент продемонстрировал хорошие вычислительные свойства предложенной решетки.

Авторы результата предполагают, что он может оказаться полезным как в теоретическом, так и в практическом плане, инициировать дальнейшие исследования.

Литература

1. *Бахвалов Н. С.* Об оптимальных оценках сходимости квадратурных процессов и методов интегрирования типа Монте-Карло на классах функций // *Численные методы решения дифференциальных и интегральных уравнений и квадратурные формулы: (Дополнение к «Журн. вычисл. математики и мат. физики»).* — 1964. — Т. 4, № 4. — С. 5–63.

2. *Блинов Н. И.* Приближенное вычисление двойных интегралов // *Аннот. сб.: Алгоритмы и программы. ВНТИ центр.* — 1974. — № 2, 3.

3. *Блинов Н. И.* Алгоритм для вычисления кратных интегралов от функций с особенностями // *Аннот. сб.: Алгоритмы и программы. ВНТИ центр.* — 1977. — № 2.

4. *Васкевич В. Л.* О сходимости квадратурных формул Грегори // *Докл. АН СССР.* — 1981. — Т. 261, № 5. — С. 1041–1043.

5. *Васкевич В. Л.* Об одной задаче теории квадратурных формул. — Препринт АН СССР. Сиб. отд-ние. Институт математики, Новосибирск, № 3. — 1982.

6. *Васкевич В. Л.* Кубатурные формулы в гармонических пространствах типа Бергмана-Половинкина // Кубатурные формулы и их приложения. Доклады III семинара-совещания. — Уфа: ИМВЦ УНЦ РАН, 1996. — С. 4–13.

7. *Васкевич В. Л.* Гарантированная точность вычисления многомерных интегралов: Дис. ... д-ра физ.-мат. наук. — Новосибирск, 2003. — 243 с.

8. *Виноградов И. М.* К вопросу об оценке тригонометрических сумм // *Изв. АН СССР.* — 1965. — Т. 29, № 3.

9. *Войтишек А. В.* Использование аппроксимационных функциональных базисов в методах Монте-Карло // Кубатурные формулы и их приложения. Материалы VII семинара-совещания. — Красноярск, 2003. — С. 45–53.

10. *Войтишек Л. В.* Об одном частном случае построения кубатурных формул с пограничным слоем // *ЖВМ и МФ.* — 1969. — Т. 9, № 2. — С. 774–781.

11. *Игнатьев А. Н.* Универсальный алгоритм вычисления интегралов по ограниченным областям с гладкими границами // Кубатурные формулы и их приложения: Доклады, представленные на III семинар-совещание. — Уфа, ИМВЦ УНЦ РАН, 1996. — С. 21–31.

12. *Корнейчук Н. П.* Экстремальные задачи теории приближения. — М.: Наука, 1981. — 431 с.

13. *Коробов Н. М.* Вычисление кратных интегралов методом оптимальных коэффициентов // *Вестник МГУ.* — 1959. — № 4.

14. *Коробов Н. М.* Теоретико-числовые методы в приближенном анализе. — М.: Наука, 1963. — 224 с.

15. *Крылов В. И.* Приближенное вычисление интегралов. — М.: Наука, 1967. — 500 с.

16. *Лебедев В. И.* О квадратурах для сферы // *Журн. вычисл. математики и мат. физики.* — 1976. — Т. 16, № 2. — С. 293–306.

17. *Мысовских И. П.* Интерполяционные кубатурные формулы. — М.: Наука, 1981. — 336 с.

18. *Никольский С. М.* Квадратурные формулы. — М.: Наука, 1979. — 256 с.

19. *Носков М. В., Осипов Н. Н.* О минимальных кубатурных формулах для интегрирования по 2-мерной сфере // Комплексный анализ, дифференциальные уравнения, численные методы и приложения. V. Численные методы. — Уфа: ИМВЦ УНЦ РАН, 1996. — С. 107–112.

20. *Носков М. В.* О декартовых произведениях кубатурных формул. — Новосибирск: Наука, 1980. — 114-116 с.

21. *Осипов Н. Н.* Минимальные кубатурные формулы для тора и сферы: Автореферат дис. . . . д-ра физ.-мат. наук. — Красноярск, 1997. — 14 с.

22. *Осипов Н. Н.* Наилучшие по числу узлов серии решетчатых кубатурных формул, точных на тригонометрических многочленах трех переменных // *Журн. вычисл. матем. и матем. физ.* — 2004. — Т. 44, № 12. — С. 2150–2161.

23. *Осипов Н. Н.* Кубатурные формулы для периодических функций: Дис. ... д-ра физ.-мат. наук. — Красноярск, 2005. — 192 с.

24. *Осипов Н. Н.* О минимальных кубатурных формулах с тригонометрическим d-свойством в двумерном случае // *Журн. вычисл. матем. и матем. физ.* — 2005. — Т. 45, № 1. — С. 7–14.

25. *Подбельский В. В.* Язык Си++: Учеб. пособие. — М.: Финансы и статистика, 2004. — 560 с.

26. *Половинкин В. И.* О кубатурных формулах с регулярным пограничным слоем // *Сиб. мат. журн.* — 1972. — Т. 13, № 4. — С. 951–954.

27. *Половинкин В. И.* Последовательность кубатурных формул и функционалов с пограничным слоем: Дис. ... д-ра физ.-мат. наук. — Ленинград, 1979. — 240 с.

28. *Половинкин В. И.* Асимптотически наилучшие последовательности кубатурных и квадратурных формул // Теория кубатурных формул и вычислительная математика: Тр. конф. по дифференц. уравнениям и вычисл. математике / Отв. ред. С. Л. Соболев. — Новосибирск, 1980. — С. 116–118.

29. *Рамазанов М. Д., Рахматуллин Д. Я.* Достижение наилучшего порядка приближения интегралов функций из $W_2^{\overline{m}}(\mathbb{R}^2)$ на решетчатых кубатурных формулах за счет поворота решетки узлов // Материалы VIII международного семинара-совещания 15–22 августа 2005 г., ВСГТУ, г. Улан-Удэ. — 2005. — С. 109–116.

30. *Рамазанов М. Д., Шадиметов Х. М.* Весовые оптимальные кубатурные формулы в периодическом пространстве Соболева // *Докл. РАН.* — 1999. — Т. 368, № 4. — С. 453–455.

31. *Рамазанов М. Д.* Лекции по теории кубатурных формул. — Уфа: Изд-во БашГУ, 1973. — 177 с.

32. *Рамазанов М. Д.* О порядке сходимости решетчатых кубатурных формул в пространствах с доминирующей производной // *Докл. АН СССР.* — 1984. — Т. 277, № 3. — С. 551–553.

33. *Рамазанов М. Д.* Теория решетчатых кубатурных формул с ограниченным пограничным слоем // Кубатурные формулы и их приложения: Доклады, представленные на III семинар-совещание. — Уфа, ИМВЦ УНЦ РАН, 1996. — С. 77–89.

34. *Рахматуллин Д. Я.* Вычисление интегралов по многомерным областям на многопроцессорных вычислительных системах // Международная уфимская зимняя школа-конференция по математике и физике с участием студентов, аспирантов и молодых ученых, БашГУ, Уфа, 30 ноября – 06 декабря 2005 года. — Уфа: РИО БашГУ, 2005. — С. 151–157.

35. *Рахматуллин Д. Я.* Достижение наилучшего порядка приближения интегралов функций из $W_2^{\overline{m}}(\mathbb{R}^2)$ в частных случаях // V Региональная школа-конференция для студентов, аспирантов и молодых ученых по математике и физике: Тезисы докладов. — Уфа: РИО БашГУ, 2005. — С. 20.

36. *Рахматуллин Д. Я.* Введение в MPI: Учеб. пособие // Современные информационные и компьютерные технологии в инженерно-научных исследованиях. Научно-исследовательская стажировка молодых ученых. Сборник материалов. Том I. Математика. — Уфа: РИЦ БашГУ, 2006. — С. 3–68.

37. *Рахматуллин Д. Я.* Вычисление интегралов по многомерным областям на многопроцессорных вычислительных системах // *Вычислительные технологии.* — 2006. — Т. 11, № 3. — С. 118–125.

38. *Рахматуллин Д. Я.* Интегрирование на суперкомпьютере // Математика. Механика. Информатика: Тез. докл. Всерос. науч. конф., Челябинск. — Челябинск: ЧГУ, 2006. — С. 114.

39. *Салихов Г. Н.* К теории кубатурных формул на сферах // *Теория кубатурных формул и приложения функционального анализа к некоторым задачам математической физики.* — 1973. — С. 22–27.

40. *Соболев С. Л., Васкевич В. Л.* Кубатурные формулы. — Новосибирск: Изд-во ИМ СО РАН, 1996. — 484 с.

41. *Соболев С. Л.* О кубатурных формулах на сфере, инвариантных при преобразованиях конечных групп вращений // *Докл. АН СССР.* — 1962. — Т. 146, № 2. — С. 310–313.

42. *Соболев С. Л.* О числе узлов кубатурной формулы на сфере // *Докл. АН СССР.* — 1962. — Т. 146, № 4. — С. 770–773.

43. *Соболев С. Л.* Введение в теорию кубатурных формул. — М.: Наука, 1974. — 808 с.

44. *Соболь И. М.* Численные методы Монте-Карло. — М.: Наука, 1973. — 311 с.

45. *Умарханов И.* Построение и обоснование решетчатых кубатурных формул для областей с кусочно-гладкими границами: Дис. ... канд. физ.-мат. наук. — Ташкент: ТашГУ, 1986. — 173 с.

46. *Шойнжуров Ц. Б.* Некоторые вопросы теории кубатурных формул в пространстве w_p^m // *Сиб. мат. журн.* — 1967. — Т. 7, № 2. — С. 41–45.

47. *Шойнжуров Ц. Б.* Некоторые вопросы теории кубатурных формул в неизотропных пространствах С.Л. Соболева // *Докл. ДАН СССР.* — 1973. — Т. 209, № 5. — С. 1036–1038.

48. *Ramazanov M. D.* To the L_p-theory of Sobolev formulas // *Siberian advances in mathematics.* — 1999. — Vol. 9, no. 1. — Pp. 99–125.

49. *Sard A.* Best approximate integration formulas, best approximate formulas // *Amer. J. Math.* — 1949. — Vol. 71. — Pp. 80–91.

50. *Stroud A. H.* Approximate calculation of multiple integrals. — Englewood Chiffs, New Jersey: Prentice-Hall, 1971.

Приложение 1

Главный файл программы расчета приближенного значения интегралов по многомерным областям.

```
/*
  Name:        ri.cpp
  Copyright:   Institute of Mathematics, Ufa
  Author:      Rakhmatullin D.Y.
  Description: Main file in the program. The program calculates
               integral of function defined on multidimensional
               domain by means of cubature formulas algorithm.
*/

/*СПИСОК ПОДКЛЮЧАЕМЫХ БИБЛИОТЕК==============================*/
//Стандартные библиотеки:
#include <fstream>//Библ. файловых ф-й потокового ввода-вывода
#include <iomanip>//Библ. манипуляторов потокового ввода-вывода
#include <iostream>//Библиотека потокового ввода-вывода
#include <math.h>//Библиотека математических функций
#include <mpi.h>//Библиотека параллельных функций
#include <float.h>//Библ. констант вычислений с плав. точкой
#include <time.h>//Библ. ф-й для работы с временем
//Пользовательские файлы:
#include "proto.h"//Заголовочный файл с объяв-ми осн. перем-х
#include "functions.cpp"//Файл со вспомогательными функциями
#include "print.cpp"//Вывод результатов  счета
#include "vand.cpp"//Инициализация массивов vand и a
/*=========================КОНЕЦ СПИСКА ПОДКЛЮЧАЕМЫХ БИБЛИОТЕК*/

/*СПИСОК ОПРЕДЕЛЕНИЙ ГЛОБАЛЬНЫХ ПЕРЕМЕННЫХ====================*/
double eps_corr;//Корректирующее эпсилон
double eps1;//Используется в ф-ии phi0
double eps2;//Используется в ф-ии phi0
double eps_xi;//Параметр крутости ф-ии xi
double top_psi;//Используется в ф-ии psi
double mid_psi;//Используется в ф-ии psi
double eps_secant;//Исп-ся в мет. секущих (gammaN)

//Для возможности включать потоковые заголовочные файлы без .h
using namespace std;

typedef unsigned long ul;//Сокращенное название типа
ofstream out;//Файловая переменная для вывода результатов
```

```
double infinity= 6.0;//Используется в ф-ии psi

int n;//Размерность пространства

// True Answer - может содержать изначально известный
// верный ответ (используется для сравнения с полученным ответом)
double ta;
//Общее количество процессов и номер данного процесса
int size, rank;
//Гладкость функции и интервал ее изменения
int M, M_start, M_end;
//Кол-во точек на ребре единичного куба и интервал его изменения
long N, N_start, N_end;
//Коэффициенты в выражениях N= N*b_N+a_N
long a_N, b_N;
//Счетчики циклов по N
ul c_N= 0;
double c_Phi, c_psi, c_koeff, c_sec;
//Коэф-ты, использующиеся при задании интегрируемой ф-ии
double b1[10], p1[10];
//Массивы, связанные с матрицей Вандермонда. Первый индекс
//массива vand - гладкость, второй - номер строки, третий -
//номер столбца. Первый индекс массивов a и A - гладкость,
//второй - компонента вектора.
double vand[7][10][10], a[7][10], A[7][10];

//Начальная и конечная координаты узлов, доверенных процессу
//(используются одномерные координаты в промежутке [0; V[n-1]]
double s, e;

//Целая и дробная части расстояния от точки, лежащей на границе
//области до соответствующей координатной плоскости
ul sigma; double eta;

double h, h_M;//Шаг решетки и M-я его степень
double begin_time, end_time;//Нач. и конеч. отметки времени
int monit;//Если monit==1, включен режим мониторинга
//Если tol==1, включен режим вычисления погрешности
//по устойчивости знаков
int tol;

//Вспомогательные переменные
double *times[5], *answers[5], koe[100][100],
x[10], eps_1_2_inv, a1_psi, b1_psi, a2_psi, b2_psi, P[10][10],
PP0[10],PP1[10],ans;
long k_dim;
/*==============КОНЕЦ СПИСКА ОПРЕДЕЛЕНИЙ ГЛОБАЛЬНЫХ ПЕРЕМЕННЫХ*/

int main(int argc, char **argv)
```

```
{
 /*СПИСОК ОПРЕДЕЛЕНИЙ ЛОКАЛЬНЫХ ПЕРЕМЕННЫХ====================*/
 double koeff;//Локальный коэффициент кубатурной формулы
 double KOEFF;//Глобальный коэффициент кубатурной формулы
 double sum_local;//Инт-ая сумма, вычисленная одним процессом
 double sum;//Общая инт-ая сумма, вычисленная нулевым процессом

 //V[i] - количество узлов, содержащихся в (i+1)-мерной грани
 //гиперкуба. V[n-1] - общее число узлов, попавших в
 //гиперкуб. Так как число узлов может превосходить
 // 4 миллиарда, используется вещественный тип данных.
 double V[20];
 //Динамический массив целочисленных координат узлов решетки
 long k[20];
 //N, деленное нацело на 2
 long Nd2;

 int dim;//Размерность пространства (значения от 0 до n-1)
 int dir;//Направление (для выбора срезок) (-1 или 1)

 //Вспомогательные переменные
 long t, min_t_M;
 double t1, w1, Phi_t, p0, psi_t;
 double vv[100][100];
 /*==============КОНЕЦ СПИСКА ОПРЕДЕЛЕНИЙ ЛОКАЛЬНЫХ ПЕРЕМЕННЫХ*/

 //Инициализация MPI
 MPI_Init(&argc,&argv);
 //Каждый процесс узнает общее количество процессов
 MPI_Comm_size(MPI_COMM_WORLD, &size);
 //Каждый процесс узнает свой уникальный номер
 MPI_Comm_rank(MPI_COMM_WORLD, &rank);

 //Функция, задающая значения переменных по умолчанию,
 //а также заполняющая их значениями параметров командной строки
 SetVar(argc,argv);

 if (rank==0)//Главный процесс подготавливает печать результатов
 {
  //Открыть файл
  OpenFile();
  //и вывести туда значения осн. переменных и шапку
  InitPrint(size,&out);
  //Дублировать вывод в стандартный поток вывода (на экран)
  InitPrint(size,&cout);
 }

 Init_Vand();//Вызов функции инициализации массивов vand, a и A
```

```
 //Вспомогательная переменная, исп-ся в ф-ии phi0
 eps_1_2_inv= 1/(eps2-eps1);

 a1_psi= -top_psi/mid_psi;
 b1_psi= top_psi;
 a2_psi= top_psi/(mid_psi);
 b2_psi= -(1-mid_psi)*top_psi/(mid_psi);

/*============================================================*/
/*НАЧАЛО САМОГО БОЛЬШОГО ЦИКЛА - ПО КОЛИЧЕСТВУ ТОЧЕК N =======*/
/*============================================================*/
 for(N=N_start;N<=N_end;N= N*b_N+a_N)
 {
  c_N++;//Счетчик кол-ва циклов по N
  h= 1./N;//Текущий шаг решетки
  Nd2= N/2;
  //Подсчет количества узлов, содержащихся в каждой из гипер-
  //граней гиперкуба. V[n-1] - общее число узлов гиперкуба.
  V[0]= (double) N;
  for(int i=1;i<n;i++)
   V[i]= V[i-1]*N;
  /*ЦИКЛ №2 - ПО M ============================================*/
  for(M=M_start;M<=M_end;M++)
  {
   begin_time= MPI_Wtime();//Первая отметка времени
   //Распределение узлов по процессам. Наилучшая
   //равномерность распределения вычислений достигается
   //тем, что разрезается одномерный массив V[n-1].
   //Процессу с номером rank доверяется вычисление (e-s+1)
   //узлов, одномерные координаты которых заполняют отрезок
   //массива V[n-1] с началом в s и концом в e.
   s= floor((rank*V[n-1])/size);
   e= floor(((rank+1)*V[n-1])/size)-1;
   c_psi= c_koeff= c_sec= c_Phi= 0;

   //Переменные, используемые в режиме мониторинга
   long per= 1; double cent= (e+1-s)/100, percent= per*cent;

   //============================================================
   /*ЦИКЛ №3 - ПО s ============================================*/
   //Главная часть алгоритма, содержащая цикл, в котором=======
   //происходит вычисление коэффициентов кубатурной формулы и,
   //с их помощью, локальной инт-ой суммы (перем. sum_local)===
   //============================================================
   for(sum_local=0.0; s<=e; s++)
   {
    //Если включен режим мониторинга, выводится информация о
    //проценте перебранных коэффициентов и пройденном времени
    if(monit==1)
```

```
 if (fabs(s-percent)<0.4)
 {
  cout<<char(13);
  cout<<per++<<"%\tTime elapsed: "
      <<ul(MPI_Wtime()-begin_time)<<" sec";
  if(size>1)
   out<<rank<<":\t"<<per<<"%\tTime elapsed: "
      <<ul(MPI_Wtime()-begin_time)<<" sec\ts=\t"<<s<<endl;
  percent= per*cent;
 }
//Восстановление многомерных координат из одномерной.
//Для того, чтобы числа типа 0.(9) преобразовывались в 1,
//произв-ся умн-ие на корректирующий множитель (1+eps_corr).
w1= s;//Запись s во временную переменную w1
for(int i=n-1;i>=1;i--)
{
 k[i]= long( (w1/V[i-1])*(1+eps_corr) );
 w1-=k[i]*V[i-1];
}
k[0]= long(w1*(1+eps_corr));
//=====================================================
/*УСЛОВНЫЙ БЛОК Phi>=0 ================================*/
//Рассматриваются лишь точки, лежащие в области интегрир-ия
//=====================================================
if ((Phi_t= Phi(k))>0.)//Далее исп. врем. перем Phi_t
{
 c_Phi++;//Счетчик кол-ва точек в области интегрирования
 KOEFF= 0.;
 //=====================================================
 /*УСЛОВНЫЙ БЛОК Phi_t< eps2 (<=> phi0(Phi_t)<1)=========*/
 //Если точка не "срезается" центральной срезкой, для нее
 //необходимо вычислить коэффициент, а также значения всех
 //срезок в точке. Другими словами, если т-ка
 //не попала в ту область, где phi0(Phi_t)==1, она принад-
 //лежит приграничной области - множеству точек x, таких
 //что Phi(x)< eps2. В противном случае, не нужно вычислять
 //ничего, кроме значения ф-ии f - коэф-т равен единице.
 //=====================================================
 if (Phi_t< eps2)
 {
  //Перебор 2n срезывающих функций. Для каждой из них
  //вычисляется коэффициент и умножается на значение срезки.
  //Полученный глобальный к-т прибавл-ся к переменной KOEFF.
  psi_prep(k);
  for(dim=0; dim<n;dim++)
   //Если срезка в точке отличается от нуля есть смысл
   //вычислять коэффициент. Иначе - переходим к следующей.
   if((psi_t= psi(k,dim))> 0)
   {
```

```
   dir=(k[dim]-Nd2<0? -1:1);
   koeff=0.;
   Sigma(k,dim,dir);
   //t - показатель глубины точки в области
   t= (dir==-1)? k_dim-sigma-2:(N-k_dim)-sigma-2;
   if(t>=0)//Иначе, если t<0, к-т равен нулю
    if(t>=2*M)//Если точка лежит достаточно глубоко
     koeff= 1.;
    else//иначе, при t из [0;2M-1], к-т необх. вычислить
    {
     c_koeff++;//Счетчик количества вычислений коэф-в
     //eta= {gamma(hk)/h} - дробная часть
     eta= gammaN(k,dim,dir,Phi_t,psi_t)-sigma;
     min_t_M= (t<M) ? t:M;//Заранее вычисл. верх. предел

     ////////////////////////////////////////////////
     for(int i=0; i<=min_t_M; i++)  //                //
     {                              //    ВЫЧИСЛЕНИЕ  //
      t1= vand[M][i][M];            //    ЛОКАЛЬНОГО  //
      for(int l=M-1;l>=0;l--)       //   КОЭФФИЦИЕНТА //
       t1= t1*eta+vand[M][i][l];    //    КУБАТУРНОЙ  //
      koeff+=t1*A[M][t-i<M ? t-i:M];//     ФОРМУЛЫ    //
     }                              //                //
     ////////////////////////////////////////////////
    }
    //Восстанавливаем dim-ную коор-ту из врем. перем. k_dim
    k[dim]= k_dim;
    //В KOEFF собирается ГЛОБАЛЬНЫЙ к-т КФ в точке kh
    KOEFF+= psi_t*koeff;
  }//Конец условия psi_t>0 и циклов перебора срезок

 //Для быстроты счета для кажд. из вариантов p0 записана
 //отдельная формула очередного слагаемого суммы sum_local
 if (Phi_t<= eps1)//Если p0==0, т.е. точка близка к границе
  sum_local+= KOEFF*f(k);
 else
 {
  p0= phi0(Phi_t);
  //Глобальным коэффициентом будет уже
  //не KOEFF, а (KOEFF*(1-p0)+p0)
  sum_local+= (KOEFF*(1-p0)+p0)*f(k);
 }
}
else//Если точка kh "срезается" центральной срезкой, то
 sum_local+= f(k);//глобальный к-т KOEFF равен единице
//=========================================================
/*=КОНЕЦ УСЛОВНОГО БЛОКА Phi_t< eps2 (<=> phi0(Phi_t)<1)*/
//===Рассматриваются лишь точки, лежащие у границы области
//=========================================================
```

```
    }
    //=============================================================
    /*=========================КОНЕЦ УСЛОВНОГО БЛОКА Phi>=0*/
    //Рассматриваются лишь точки, лежащие в области интегрир-ия
    //=============================================================
   }
   //=============================================================
   /*==================================КОНЕЦ ЦИКЛА №3 - ПО s*/
   //=============================================================

   //Сбор результатов вычислений всех процессов
   //в переменную sum процесса 0
   MPI_Reduce(&sum_local,&sum,1,MPI_DOUBLE,
              MPI_SUM,0,MPI_COMM_WORLD);

   if (rank==0)
   {
    sum*= pow(h,n);
    end_time= MPI_Wtime();//Вторая отметка времени
    answers [M-M_start][c_N-1]= sum;
    times   [M-M_start][c_N-1]= end_time-begin_time;
    //Вывод результатов в файл и на экран
    PrintResults(&out);  cout << endl;
    PrintResults(&cout); out << endl;
   }
  }
  /*==================================КОНЕЦ ЦИКЛА №2 - ПО M*/
 }
/*=============================================================*/
/*=========КОНЕЦ САМОГО БОЛЬШОГО ЦИКЛА - ПО КОЛИЧЕСТВУ ТОЧЕК N*/
/*=============================================================*/

 //Завершение параллельной части программы
 MPI_Finalize();
 //Конец программы
 return 0;
}
```

Файл со вспомогательными вычислительными функциями.

```
/*
  Name:        functions.cpp
  Copyright:   Institute of Mathematics, Ufa
  Author:      Rakhmatullin D.Y.
  Description: auxiliary file with functions definitions
*/

/*СПИСОК ПОДКЛЮЧАЕМЫХ БИБЛИОТЕК==============================*/
//Стандартные библиотеки:
```

```
#include <fstream>//Библ. файловых ф-й потокового ввода-вывода
#include <iomanip>//Библ. манипуляторов потокового ввода-вывода
#include <iostream>//Библиотека потокового ввода-вывода
#include <math.h>//Библиотека математических функций
#include <mpi.h>//Библиотека параллельных функций
#include <float.h>//Библ. констант вычислений с плав. точкой
#include <limits.h>//Библ. констант целочисленных вычислений
//Пользовательские файлы:
#include "proto.h"//Заголовочный файл с объяв-ми осн. перем-х
/*=========================КОНЕЦ СПИСКА ПОДКЛЮЧАЕМЫХ БИБЛИОТЕК*/

//////////////////////////////////////////////////////////////
//////////////////////независимые функции//////////////////////
//////////////////////////////////////////////////////////////

inline double f(long *k)
//Интегрируемая функция. Аргументом является целочисленный
//вектор k, который преобраз-ся в в-р x, вычисляемый как x= hk.
{
 if (k!=NULL)
  for(int i=0;i<n;i++)
   x[i]= k[i]*h;
 double answer=0.0;
 for(int i=0;i<n;i++)
  answer+= b1[i]*pow(k[i]*h,p1[i]);
 return (sin(answer));
}

inline double Phi(long *k)
//Функция, задающая область интегрирования Омега.
//Точка x лежит внутри области, если Phi(x)>0 и на границе, если
//Phi(x)==0. Если в качестве параметра указан адрес NULL,
//аргументом является глобально определенный массив x. Если же
//указан адрес целоч. массива k, вектор x вычисляется как x= hk.
{
 //Если аргумент ф-ии - целочисленный в-р k, нужно вычислить x
 if (k!=NULL)
  for(int i=0;i<n;i++)
   x[i]= k[i]*h;

 double p= 1., t;
 for(int i=0;i<n;i++)
 {
  t= 2*x[i]-1.;
  p-= t*t;
 }
 return p;
}
```

```
void SetVar(int argc, char **argv)
//Функция, задающая значения переменных по умолчанию,
//а также заполняющая их значениями параметров командной строки
{
 /*Определение значений по умолчанию*/

 n= 2;//Размерность пространства

 //Интервал изменения переменной N - количества точек
 //на одном ребре гиперкуба
 N_start= N_end= 10;
 //Границы интервала изменения переменной M - "учитываемой"
 //гладкости интегрируемой функции
 M_start= M_end= 2;
 //Коэффициенты в выражениях N= N*b_N+a_N
 a_N= 1;  b_N= 1;//По умолчанию N= N+1
 //Заполнение начальными значениями массивов
 //с циклом по размерностям
 for(int i=0;i<n;i++)
 {
  b1[i]= i+1;//к-ты, использующиеся при задании интегрируемой ф-ии
  p1[i]= i+1;//к-ты, использующиеся при задании интегрируемой ф-ии
 }

 monit= 1;//Включен режим мониторинга
 eps_corr= 1e-15;//Корректирующее эпсилон
 eps_xi= 1.0;//Параметр крутости ф-ии xi
 top_psi= 6.0;//Используется в ф-ии psi
 mid_psi= 0.3;//Используется в ф-ии psi
 eps_secant= 1e-15;//Исп-ся в мет. секущих (gammaN)
 //Задание eps1 и eps2, исп-ся в построении ф-ии phi0
 eps1= 0.2;//Если (Phi(x)<=eps1) => phi0(x)==0
 eps2= 0.5;//Если (Phi(x)>=eps2) => phi0(x)==1

  //Режим вычисления погрешности по устройчивости знаков
 //выключен по умолчанию.
 tol= 0;

 /*Изменение знач-й по умолчанию параметрами командной строки*/

 //Цикл по элементам командной строки (количество: argc-1)
 for(int i=1;i<argc;i++)
 {
  //Если очередной элемент - строка "M", и имеется следующий
  //элемент, он рассматривается как значение начала и конца
  //отрезка изменения параметра M - M_start и M_end.
  if (!strcmp(argv[i],"M")&&(i+1<argc))
  {
   M_start= M_end= atoi(argv[i+1]);
```

```
 //Если через пробел следует число, то M_end исправляется
 if ((i+2<argc)&&(atoi(argv[i+2])!=0))
  M_end= atoi(argv[i+2]);
}
//Ниже - аналогично с N
if (!strcmp(argv[i],"N")&&(i+1<argc))
{
 N_start= N_end= atol(argv[i+1]);
 if ((i+2<argc)&&(atol(argv[i+2])!=0))
 {
  N_end= atol(argv[i+2]);
  if ((i+3<argc)&&(atol(argv[i+3])!=0))
  {
   b_N= atol(argv[i+3]);
   a_N= 0;
    if ((i+4<argc)&&(atol(argv[i+4])!=0))
     a_N= atol(argv[i+4]);
  }
 }
}
if (!strcmp(argv[i],"n")&&(i+1<argc))
 n= atoi(argv[i+1]);

if(!( 2<=M_start&&M_start<=M_end&&M_end<=6&&
      1<=N_start&&N_start<=N_end&&N_end<=LONG_MAX&&
      ((a_N==0&&b_N>=2)||(a_N>=1&&b_N>=1))&&(n>=2)))
{
 out.open("result.txt",ios::trunc);
 out<<"Wrong initial data!"<<endl;
 cout<<"Wrong initial data!"<<endl;
 MPI_Abort(MPI_COMM_WORLD,1);
}

if (!strcmp(argv[i],"b1"))
for(int j=1;(j<=n)&&(i+j<argc)&&atoi(argv[i+j]);j++)
 b1[j-1]=atof(argv[i+j]);
if (!strcmp(argv[i],"p1"))
for(int j=1;(j<=n)&&(i+j<argc)&&atoi(argv[i+j]);j++)
 p1[j-1]=atof(argv[i+j]);
if (!strcmp(argv[i],"ta")&&(i+1<argc)&&atof(argv[i+1]))
 ta= atof(argv[i+1]);
if (!strcmp(argv[i],"eps_corr")&&(i+1<argc)&&atof(argv[i+1]))
 eps_corr= atof(argv[i+1]);
if (!strcmp(argv[i],"eps1")&&(i+1<argc)&&atof(argv[i+1]))
 eps1= atof(argv[i+1]);
if (!strcmp(argv[i],"eps2")&&(i+1<argc)&&atof(argv[i+1]))
 eps2= atof(argv[i+1]);
if (!strcmp(argv[i],"eps_xi")&&(i+1<argc)&&atof(argv[i+1]))
 eps_xi= atof(argv[i+1]);
```

```
  if (!strcmp(argv[i],"top_psi")&&(i+1<argc)&&atof(argv[i+1]))
   top_psi= atof(argv[i+1]);
  if (!strcmp(argv[i],"mid_psi")&&(i+1<argc)&&atof(argv[i+1]))
   mid_psi= atof(argv[i+1]);
  if (!strcmp(argv[i],"eps_secant")&&(i+1<argc)&&atof(argv[i+1]))
   eps_secant= atof(argv[i+1]);
  if (!strcmp(argv[i],"monit")&&(i+1<argc))
   monit= atoi(argv[i+1]);
  if (!strcmp(argv[i],"tol"))
   tol= 1;
 }

 //Рассчитывается и выделяется примерное кол-во памяти, необх.
 //для хранение всех ответов и отметок времени
 long k= (b_N==1)?
          long(((N_end-N_start+1)/a_N+1)):
          long((log(double(N_end/N_start))/log(double(b_N))+1));
 for(int i=0;i<5;i++)
 {
  answers[i]= new double[k];
  times[i]= new double[k];
 }

 //Если режим устойчивости знаков выключен и не указан верный
 //ответ (ta), за ответ берется объем n-мерного
 //гипершара с радиусом 0.5
 if(tol==0&&ta==0.&&N_end>N_start)
  switch(n)
  {
   case 2:  ta= 0.78539816339744830962; break;
   case 3:  ta= 0.52359877559829887309; break;
   case 4:  ta= 0.30842513753404245685; break;
   case 5:  ta= 0.16449340668482264365; break;
   case 6:  ta= 0.08074551218828078171 2; break;
   case 7:  ta= 0.03691223414321407164 0; break;
   case 8:  ta= 0.01585434424381550085 3; break;
   case 9:  ta= 0.00644240020066153685 46; break;
   case 10: ta= 0.00249039457019272016 01; break;
  }
}

double xi (double t)
//Функция, равная нулю при t<=0, гладко возрастающая до
//единицы при 0<t<eps_xi и равная единице при t>=eps_xi
{
 if (t<=0)
  return 0.0;
 else
  if (t>=eps_xi)
```

```
   return 1.0;
  else
  {
   if (eps_xi==1.)
    switch (M)
    {
     case 2: return (((6.*t-15.)*t+10.)*t*t*t);     break;
     case 3: return ((((-20.*t+70.)*t-84.)*t+35.)*t*t*t*t);
      break;
     case 4: return (((((70.*t-315.)*t+540.)*t-420.)*t+126.)
      *t*t*t*t*t);     break;
     case 5: return ((((((-252.*t+1386.)*t-3080.)*t+3465.)*t-
      1980.)*t+462.)*t*t*t*t*t*t);     break;
     case 6: return ((((((((924.*t-6006.)*t+16380.)*t-24024.)*t+
      20020.)*t-9009.)*t+1716.)*t*t*t*t*t*t*t);     break;
    }
   if (eps_xi==0.25)
    switch (M)
    {
     case 2: return (((6144.*t-3840.)*t+640.)*t*t*t);     break;
     case 3: return ((((-327680.*t+286720.)*t-86016.)*t+8960.)*
      t*t*t*t);     break;
     case 4: return (((((18350080.*t-20643840.)*t+8847360.)*t-
      1720320.)*t+129024.)*t*t*t*t*t);     break;
     case 5: return ((((((-1056964608.*t+1453326336.)*t-
      807403520.)*t+227082240.)*t-32440320.)*t+1892352.)*
      t*t*t*t*t*t);     break;
     case 6: return ((((((((62008590336.*t-100763959296.)*t+
     68702699520.)*t-25190989824.)*t+5248122880.)*t-590413824.)*
     t+28114944.)*t*t*t*t*t*t*t);     break;
    }
  }
}

////////////////////////////////////////////////////////////////
//////////////////////зависимые функции/////////////////////////
////////////////////////////////////////////////////////////////

void psi_prep (long *k)
{
 ans= 0.;
 for(int dim=0;dim<n-1;dim++)
 {
  double x_dim= h*k[dim];
  double A1= a1_psi*x_dim+b1_psi;
  double A2= a2_psi*x_dim+b2_psi;

  for(int i=dim+1;i<n;i++)
  {
```

```
   double x_i= h*k[i];
   double A1x= A1*x_i;
   double A2x= A2*x_i;
   double PHI2D_1= xi(A1-A1x)*xi(A1x);
   double PHI2D_2= xi(A2-A2x)*xi(A2x);
   P[dim][i]= PHI2D_1+PHI2D_2;
   P[i][dim]= 1-P[dim][i];//Сумма двух других срезок в т.
  }
 }
 if(n>3)
 {
  PP0[2]= P[0][1]*P[0][2];
  PP1[2]= P[1][0]*P[1][2];
  for(int i=3;i<n;i++)
  {
   PP0[i]= PP0[i-1]*P[0][i];
   PP1[i]= PP1[i-1]*P[1][i];
  }
 }
}

double psi (long *k, int dim)
{
 c_psi++;
 double answer= 1.;

 switch (n)
 {
  case 2:
    if(dim==0)
     answer= P[0][1];
    if(dim==1)
     answer= 1.- ans;
    break;
  case 3:
    if(dim==0)
     answer= P[0][1]*P[0][2];
    if(dim==1)
     answer= P[1][0]*P[1][2];
    if(dim==2)
     answer= 1.- ans;
    break;
  case 4:
    if(dim==0)
     answer= PP0[n-1];
    if(dim==1)
     answer= PP1[n-1];
    if(dim==2)
     answer= 1- PP0[2]-PP1[2];
```

```
  if(dim==3)
   answer= 1.- ans;
  break;
case 5:
  if(dim==0)
   answer= PP0[n-1];
  if(dim==1)
   answer= PP1[n-1];
  if(dim==2)
   answer= 1- PP0[2]-PP1[2];
  if(dim==3)
   answer= PP0[2]+PP1[2]-PP0[3]-PP1[3];
  if(dim==4)
   answer= 1.- ans;
  break;
case 6:
  if(dim==0)
   answer= PP0[n-1];
  if(dim==1)
   answer= PP1[n-1];
  if(dim==2)
   answer= 1- PP0[2]-PP1[2];
  if(dim==3)
   answer= PP0[2]+PP1[2]-PP0[3]-PP1[3];
  if(dim==4)
   answer= PP0[3]+PP1[3]-PP0[4]-PP1[4];
  if(dim==5)
   answer= 1.- ans;
  break;
case 7:
  if(dim==0)
   answer= PP0[n-1];
  if(dim==1)
   answer= PP1[n-1];
  if(dim==2)
   answer= 1- PP0[2]-PP1[2];
  if(dim==3)
   answer= PP0[2]+PP1[2]-PP0[3]-PP1[3];
  if(dim==4)
   answer= PP0[3]+PP1[3]-PP0[4]-PP1[4];
  if(dim==5)
   answer= PP0[4]+PP1[4]-PP0[5]-PP1[5];
  if(dim==6)
   answer= 1.- ans;
  break;
case 8:
  if(dim==0)
   answer= PP0[n-1];
  if(dim==1)
```

```
    answer= PP1[n-1];
   if(dim==2)
    answer= 1- PP0[2]-PP1[2];
   if(dim==3)
    answer= PP0[2]+PP1[2]-PP0[3]-PP1[3];
   if(dim==4)
    answer= PP0[3]+PP1[3]-PP0[4]-PP1[4];
   if(dim==5)
    answer= PP0[4]+PP1[4]-PP0[5]-PP1[5];
   if(dim==6)
    answer= PP0[5]+PP1[5]-PP0[6]-PP1[6];
   if(dim==7)
    answer= 1.- ans;
   break;
 case 9:
   if(dim==0)
    answer= PP0[n-1];
   if(dim==1)
    answer= PP1[n-1];
   if(dim==2)
    answer= 1- PP0[2]-PP1[2];
   if(dim==3)
    answer= PP0[2]+PP1[2]-PP0[3]-PP1[3];
   if(dim==4)
    answer= PP0[3]+PP1[3]-PP0[4]-PP1[4];
   if(dim==5)
    answer= PP0[4]+PP1[4]-PP0[5]-PP1[5];
   if(dim==6)
    answer= PP0[5]+PP1[5]-PP0[6]-PP1[6];
   if(dim==7)
    answer= PP0[6]+PP1[6]-PP0[7]-PP1[7];
   if(dim==8)
    answer= 1.- ans;
   break;
 case 10:
   if(dim==0)
    answer= PP0[n-1];
   if(dim==1)
    answer= PP1[n-1];
   if(dim==2)
    answer= 1- PP0[2]-PP1[2];
   if(dim==3)
    answer= PP0[2]+PP1[2]-PP0[3]-PP1[3];
   if(dim==4)
    answer= PP0[3]+PP1[3]-PP0[4]-PP1[4];
   if(dim==5)
    answer= PP0[4]+PP1[4]-PP0[5]-PP1[5];
   if(dim==6)
    answer= PP0[5]+PP1[5]-PP0[6]-PP1[6];
```

```
    if(dim==7)
     answer= PP0[6]+PP1[6]-PP0[7]-PP1[7];
    if(dim==8)
     answer= PP0[7]+PP1[7]-PP0[8]-PP1[8];
    if(dim==9)
     answer= 1.- ans;
    break;
 }
 ans+= answer;
 return answer;
}

void Sigma (long *k, int dim, int dir)
{
 k_dim= k[dim];
 while (Phi(k)> 1e-15)//Пока точка kh лежит внутри области
 {
  k[dim]+= dir;//смещаемся по направлению dir разм-ти dim
  }
 //Вычисляем [gamma(hk)/h] - целое кол-во кубиков от
 //границы до рассматриваемой координатной плоскости
 sigma= (dir==-1)? k[dim]:N-k[dim];
}

double gammaN (long *k, int dim, int dir, double Phi_t,
                double psi_t)
//Функция участка границы области интегрирования.
//По заданной точке kh и направлению проектирования границы
//(dim и dir) находит точку пересечения направления проектирова-
//ния и границы области интегрирования. Возвращает dim-ную коор-
//динату точки, деленную на шаг решетки h (см. поправку ниже).
{
 double x_n;
 if(Phi(k)==0.)//Если точка попала на гран., x_n м. не уточнять
  x_n= k[dim]*h;
 else
 {
  for (int i=0;i<n;i++)//Вычисление вещественных коор-т точки
   x[i]= k[i]*h;

  //Три последовательных шага приближения метода секущих
  double x_n_min_1= x[dim];
  x_n= x[dim]+h/2;
  double x_n_plus_1;
  //Значения ф-ии Phi в точках x_n и x_n_plus_1
  double Phi_x_n, Phi_x_n_min_1;
  //Расстояние между шагами x_n и x_n_plus_1
  double x_n_eps;
```

```
  //Уточнение dim-ной коор-ты методом секущих
  do
  {
   c_sec++;
   x[dim]= x_n;
   Phi_x_n= Phi(NULL);
   x[dim]= x_n_min_1;
   Phi_x_n_min_1= Phi(NULL);
   if(Phi_x_n-Phi_x_n_min_1==0.) break;
   x_n_plus_1= x_n- Phi_x_n*(x_n-x_n_min_1)/
               (Phi_x_n-Phi_x_n_min_1);
   x_n_eps= fabs(x_n_plus_1- x_n);
   x_n_min_1= x_n;
   x_n= x_n_plus_1;
  }
  while (x_n_eps> eps_secant);
 }

 //Результат выводится в виде расстояния (в масштабе h) до
 //той координатной плоскости, на которую осущ-ся проектирование
 if (dir==-1)
  return (x_n*N);
 else
  return (1-x_n)*N;
}

inline double phi0 (double x)
//Центральная срезывающая функция. Равна нулю, если неположителен
//аргумент ф-ии xi, т.е. при x<= eps1. Равна единице, если
//аргумент ф-ии xi не меньше единицы, т.е. x>= eps2.
//Многомерность и др. требуемые св-ва обеспечиваются тем, что
//в качестве агрумента выступает значение ф-ии Phi, постепенно
//увеличивающееся с нуля от границы до внутренности области.
{
 return (xi((x-eps1)*eps_1_2_inv));
}
```

Файл с функциями печати.

```
/*
  Name:        print.cpp
  Copyright:   Institute of Mathematics, Ufa
  Author:      Rakhmatullin D.Y.
  Description: auxiliary file devoted to the printing functions
*/

/*СПИСОК ПОДКЛЮЧАЕМЫХ БИБЛИОТЕК==============================*/
//Стандартные библиотеки:
#include <fstream>//Библ. файловых ф-й потокового ввода-вывода
```

```
#include <iomanip>//Библ. манипуляторов потокового ввода-вывода
#include <iostream>//Библиотека потокового ввода-вывода
#include <math.h>//Библиотека математических функций
#include <mpi.h>//Библиотека параллельных функций
#include <float.h>//Библ. констант вычислений с плав. точкой
//Пользовательские файлы:
#include "proto.h"//Заголовочный файл с объяв-ми осн. перем-х
/*=========================КОНЕЦ СПИСКА ПОДКЛЮЧАЕМЫХ БИБЛИОТЕК*/

void OpenFile()
{
  char result[100], s[100];
  sprintf(result,"P= %.4d, n= %.2d, N=%.5ld..%.5ld, M= %d..%d",
                  size, n, N_start, N_end, M_start, M_end);
  const time_t mytime_t= time(NULL);
  const tm *t= localtime(&mytime_t);;
  strftime(s, 20, "(%b %d, %H.%M.%S)", t);
  strcat(result,"  ");
  strcat(result,s);
  strcat(result,".txt");
  //Создать ф-л результатов или открыть, уничтожив сод-е (trunc)
  out.open(result,ios::trunc);
}

void InitPrint(int size, ostream* out1)
// Функция InitPrint выполняет следующие действия:
//-выводит в поток *out1 значения основных переменных программы
//-выводит в поток *out1 шапку таблицы для посл-щего вывода рез-в
{
 //Вывод в формате чисел с фиксированной точкой
 *out1<<fixed;
 *out1<<"Number of processes:\t"<<size<<endl;
 *out1<<"Dimension n:\t"<<n<<endl;
 *out1<<"N_start:\t"<<N_start<<"\t";
 *out1<<"N_end:\t"<<N_end<<endl;
 *out1<<"M_start:\t"<<M_start<<"\t";
 *out1<<"M_end:\t"<<M_end<<endl;
 *out1<<endl;
 //Установка формата вывода чисел (5 цифр)
 (*out1).precision(5);
 *out1<<"a_N:\t"<<a_N<<"\t";
 *out1<<"b_N:\t"<<b_N<<endl;
 for(int i=0;i<n;i++)
  *out1<<"b1["<<i<<"]:\t"<<b1[i]<<"\t";
 *out1<<endl;
 for(int i=0;i<n;i++)
  *out1<<"p1["<<i<<"]:\t"<<p1[i]<<"\t";
 *out1<<endl;
 *out1<<"eps1= "<<eps1<<"\teps2= "<<eps2<<endl;
```

```
 //Установка формата вывода чисел как для
 //вещественных чисел двойной точности (15 цифр)
 (*out1).precision(DBL_DIG);
 if(ta)//Если известен верный ответ, он выводится
  *out1<<"True answer:\t"<<ta<<endl;
 *out1<<endl;
 //Вывод шапки таблицы результатов
 *out1<<"N\t"<<"M\t"<<"Answer\t\t\t"<<"Time\t"<<"Error\t\t";
 *out1<<"Points\t"<<"Domain\t"<<"Shear\t";
 *out1<<"Koeffs\t"<<"Secant"<<endl;
 *out1<<"\t\t\t\t\t\t\t\t\t\t"<<"points\t"<<"func.\t"
      <<"\tmethod\t";
 *out1<<endl;
}

void PrintResults(ostream* out1)
//Функция вывода результатов счета в поток *out1
{
 //Печать основных параметров и ответа
 (*out1).precision(DBL_DIG);//Выв. 15-и зн. после точки
 *out1<<N <<"\t"<<M<<"\t"<<answers[M-M_start][c_N-1]<<"\t";

 //Печать затраченного времени
 (*out1).precision(0);//Время будет ОКРУГЛЯТЬСЯ с точ. до целого
 *out1<<times[M-M_start][c_N-1]<<"\t";//Печать времени
 (*out1).precision(DBL_DIG);//Возв. к выв. 15-и зн. после точки

 //Вывод в формате чисел с плавающей точкой
 *out1<<scientific;
 (*out1).precision(2);//Округление до 2-го знака после точки
 //Печать погрешности вычислений
 if(ta)//Если известен точный ответ, выводится разность с ним
  *out1<<fabs(ta-answers[M-M_start][c_N-1])<<"\t";
 else//Иначе, если включен режим погр. по устойчивости зн-в и
  if(tol&&c_N>1)//выч-нный ответ-не первый (есть с чем сравнить)
   *out1<<fabs(answers[M-M_start][c_N-1]-
               answers[M-M_start][c_N-2])<<"\t";
  else
   *out1<<"_________\t";

 (*out1).precision(DBL_DIG);//Возв. к выв. 15-и зн. после точки
 //Вывод в формате чисел с фиксированной точкой
 *out1<<fixed;

 (*out1).precision(0);
 *out1<<e+1<<"\t";
 *out1<<c_Phi<<"\t";
 *out1<<c_psi<<"\t";
 *out1<<c_koeff<<"\t";
```

```
 *out1<<c_sec;
 (*out1).precision(DBL_DIG);//Возв. к выв. 15-и зн. после точки
}

double SquarePrint(double vv[100][100], int N1, int N2)
//Функция, для вывода двумерных картин коэффициентов
{
 out<<endl;
 out<<"\t";
 for(int i=0;i<N1;i++)
  out<<i<<"\t";
 out<<endl;
 out<<endl;
 out.precision(2);
 for(int j=N2-1;j>=0;j--)
 {
  out<<j<<"\t";
  for(int i=0;i<N1;i++)
   if(vv[i][j]==0.)
    out<<"   "<<((i==N1-1)?("\n"):("\t"));
   else
    if(vv[i][j]==1.)
     out<<"####"<<((i==N1-1)?("\n"):("\t"));
    else
     out<<vv[i][j]<<((i==N1-1)?("\n"):("\t"));
 }
 out<<endl;
 out<<endl;
 out<< resetiosflags(ios::showpos);
}
```

Файл, в котором инициализируются массивы, связанные с матрицей, обратной матрице Вандермонда.

```
/*
  Name:        vand.cpp
  Copyright:   Institute of Mathematics, Ufa
  Author:      Rakhmatullin D.Y.
  Description: auxiliary file devoted to the one function
               "Init_Vand()" which initializes three arrays
               concerned with Vandermonde Matrix
*/

/*СПИСОК ПОДКЛЮЧАЕМЫХ БИБЛИОТЕК==============================*/
#include "proto.h"//Заголовочный файл с объяв-ми осн. перем-х
/*=========================КОНЕЦ СПИСКА ПОДКЛЮЧАЕМЫХ БИБЛИОТЕК*/

void Init_Vand ()
{
```

```
vand[2][0][0] = 3.0000000000000000e+00;
vand[2][0][1] = -2.5000000000000000e+00;
vand[2][0][2] = 5.0000000000000000e-01;
vand[2][1][0] = -3.0000000000000000e+00;
vand[2][1][1] = 4.0000000000000000e+00;
vand[2][1][2] = -1.0000000000000000e+00;
vand[2][2][0] = 1.0000000000000000e+00;
vand[2][2][1] = -1.5000000000000000e+00;
vand[2][2][2] = 5.0000000000000000e-01;

vand[3][0][0] = 4.0000000000000000e+00;
vand[3][0][1] = -4.3333333333333333e+00;
vand[3][0][2] = 1.5000000000000000e+00;
vand[3][0][3] = -1.6666666666666667e-01;
vand[3][1][0] = -6.0000000000000000e+00;
vand[3][1][1] = 9.5000000000000000e+00;
vand[3][1][2] = -4.0000000000000000e+00;
vand[3][1][3] = 5.0000000000000000e-01;
vand[3][2][0] = 4.0000000000000000e+00;
vand[3][2][1] = -7.0000000000000000e+00;
vand[3][2][2] = 3.5000000000000000e+00;
vand[3][2][3] = -5.0000000000000000e-01;
vand[3][3][0] = -1.0000000000000000e+00;
vand[3][3][1] = 1.8333333333333333e+00;
vand[3][3][2] = -1.0000000000000000e+00;
vand[3][3][3] = 1.6666666666666667e-01;

vand[4][0][0] = 5.0000000000000000e+00;
vand[4][0][1] = -6.4166666666666667e+00;
vand[4][0][2] = 2.9583333333333333e+00;
vand[4][0][3] = -5.8333333333333333e-01;
vand[4][0][4] = 4.1666666666666667e-02;
vand[4][1][0] = -1.0000000000000000e+01;
vand[4][1][1] = 1.7833333333333333e+01;
vand[4][1][2] = -9.8333333333333333e+00;
vand[4][1][3] = 2.1666666666666667e+00;
vand[4][1][4] = -1.6666666666666667e-01;
vand[4][2][0] = 1.0000000000000000e+01;
vand[4][2][1] = -1.9500000000000000e+01;
vand[4][2][2] = 1.2250000000000000e+01;
vand[4][2][3] = -3.0000000000000000e+00;
vand[4][2][4] = 2.5000000000000000e-01;
vand[4][3][0] = -5.0000000000000000e+00;
vand[4][3][1] = 1.0166666666666667e+01;
vand[4][3][2] = -6.8333333333333333e+00;
vand[4][3][3] = 1.8333333333333333e+00;
vand[4][3][4] = -1.6666666666666667e-01;
vand[4][4][0] = 1.0000000000000000e+00;
vand[4][4][1] = -2.0833333333333333e+00;
```

```
vand[4][4][2] = 1.4583333333333333e+00;
vand[4][4][3] = -4.1666666666666667e-01;
vand[4][4][4] = 4.1666666666666667e-02;

vand[5][0][0] = 6.0000000000000000e+00;
vand[5][0][1] = -8.7000000000000000e+00;
vand[5][0][2] = 4.8333333333333333e+00;
vand[5][0][3] = -1.2916666666666667e+00;
vand[5][0][4] = 1.6666666666666667e-01;
vand[5][0][5] = -8.3333333333333333e-03;
vand[5][1][0] = -1.5000000000000000e+01;
vand[5][1][1] = 2.9250000000000000e+01;
vand[5][1][2] = -1.9208333333333333e+01;
vand[5][1][3] = 5.7083333333333333e+00;
vand[5][1][4] = -7.9166666666666667e-01;
vand[5][1][5] = 4.1666666666666667e-02;
vand[5][2][0] = 2.0000000000000000e+01;
vand[5][2][1] = -4.2333333333333333e+01;
vand[5][2][2] = 3.1000000000000000e+01;
vand[5][2][3] = -1.0083333333333333e+01;
vand[5][2][4] = 1.5000000000000000e+00;
vand[5][2][5] = -8.3333333333333333e-02;
vand[5][3][0] = -1.5000000000000000e+01;
vand[5][3][1] = 3.3000000000000000e+01;
vand[5][3][2] = -2.5583333333333333e+01;
vand[5][3][3] = 8.9166666666666667e+00;
vand[5][3][4] = -1.4166666666666667e+00;
vand[5][3][5] = 8.3333333333333333e-02;
vand[5][4][0] = 6.0000000000000000e+00;
vand[5][4][1] = -1.3500000000000000e+01;
vand[5][4][2] = 1.0833333333333333e+01;
vand[5][4][3] = -3.9583333333333333e+00;
vand[5][4][4] = 6.6666666666666667e-01;
vand[5][4][5] = -4.1666666666666667e-02;
vand[5][5][0] = -1.0000000000000000e+00;
vand[5][5][1] = 2.2833333333333333e+00;
vand[5][5][2] = -1.8750000000000000e+00;
vand[5][5][3] = 7.0833333333333333e-01;
vand[5][5][4] = -1.2500000000000000e-01;
vand[5][5][5] = 8.3333333333333333e-03;

vand[6][0][0] = 7.0000000000000000e+00;
vand[6][0][1] = -1.1150000000000000e+01;
vand[6][0][2] = 7.0888888888888889e+00;
vand[6][0][3] = -2.3125000000000000e+00;
vand[6][0][4] = 4.0972222222222222e-01;
vand[6][0][5] = -3.7500000000000000e-02;
vand[6][0][6] = 1.3888888888888889e-03;
vand[6][1][0] = -2.1000000000000000e+01;
```

```
vand[6][1][1] = 4.3950000000000000e+01;
vand[6][1][2] = -3.2741666666666667e+01;
vand[6][1][3] = 1.1833333333333333e+01;
vand[6][1][4] = -2.2500000000000000e+00;
vand[6][1][5] = 2.1666666666666667e-01;
vand[6][1][6] = -8.3333333333333333e-03;
vand[6][2][0] = 3.5000000000000000e+01;
vand[6][2][1] = -7.9083333333333333e+01;
vand[6][2][2] = 6.4833333333333333e+01;
vand[6][2][3] = -2.5395833333333333e+01;
vand[6][2][4] = 5.1458333333333333e+00;
vand[6][2][5] = -5.2083333333333333e-01;
vand[6][2][6] = 2.0833333333333333e-02;
vand[6][3][0] = -3.5000000000000000e+01;
vand[6][3][1] = 8.2000000000000000e+01;
vand[6][3][2] = -7.0694444444444444e+01;
vand[6][3][3] = 2.9333333333333333e+01;
vand[6][3][4] = -6.2777777777777778e+00;
vand[6][3][5] = 6.6666666666666667e-01;
vand[6][3][6] = -2.7777777777777778e-02;
vand[6][4][0] = 2.1000000000000000e+01;
vand[6][4][1] = -5.0250000000000000e+01;
vand[6][4][2] = 4.4666666666666667e+01;
vand[6][4][3] = -1.9270833333333333e+01;
vand[6][4][4] = 4.3125000000000000e+00;
vand[6][4][5] = -4.7916666666666667e-01;
vand[6][4][6] = 2.0833333333333333e-02;
vand[6][5][0] = -7.0000000000000000e+00;
vand[6][5][1] = 1.6983333333333333e+01;
vand[6][5][2] = -1.5408333333333333e+01;
vand[6][5][3] = 6.8333333333333333e+00;
vand[6][5][4] = -1.5833333333333333e+00;
vand[6][5][5] = 1.8333333333333333e-01;
vand[6][5][6] = -8.3333333333333333e-03;
vand[6][6][0] = 1.0000000000000000e+00;
vand[6][6][1] = -2.4500000000000000e+00;
vand[6][6][2] = 2.2555555555555556e+00;
vand[6][6][3] = -1.0208333333333333e+00;
vand[6][6][4] = 2.4305555555555556e-01;
vand[6][6][5] = -2.9166666666666667e-02;
vand[6][6][6] = 1.3888888888888889e-03;

for(int m=2;m<=6;m++)
 for(int k=0;k<=m;k++)
 {
  a[m][k]= 0.;
  double A_start= 0.;
  for(int i=0;i<=m;i++)
   a[m][k]+= vand[m][k][i]/(i+1);
```

```
   if(k>0)
    A[m][k]= A[m][k-1]+ a[m][k];
   else
    A[m][0]= a[m][0];
  }
}
```

Файл, содержащий прототипы глобальных переменных.

```
/*
  Name:        proto.h
  Copyright:   Institute of Mathematics, Ufa
  Author:      Rakhmatullin D.Y.
  Description: auxiliary file with variables declarations
*/

/*СПИСОК ОПРЕДЕЛЕНИЙ ГЛОБАЛЬНЫХ ПЕРЕМЕННЫХ====================*/
extern double eps_corr;//Корректирующее эпсилон
extern double eps1;//Используется в ф-ии phi0
extern double eps2;//Используется в ф-ии phi0
extern double eps_xi;//Параметр крутости ф-ии xi
extern double top_psi;//Используется в ф-ии psi
extern double mid_psi;//Используется в ф-ии psi
extern double eps_secant;//Исп-ся в мет. секущих (gammaN)

//Для возможности включать потоковые заголовочные файлы без .h
using namespace std;

typedef unsigned long ul;//Сокращенное название типа
extern ofstream out;//Файловая переменная для вывода результатов

extern int n;//Размерность пространства

//True Answer - может содержать изначально известный
//верный ответ (используется для сравнения с полученным ответом)
extern double ta;
//Общее количество процессов и номер данного процесса
extern int size, rank;
//Гладкость функции и интервал ее изменения
extern int M, M_start, M_end;
//Кол-во точек на ребре единичного куба и интервал его изменения
extern long N, N_start, N_end;
//Коэффициенты в выражениях N= N*b_N+a_N
extern long a_N, b_N;
//Счетчики циклов по N
extern ul c_N;
extern double c_Phi, c_psi, c_koeff, c_sec;
//Коэф-ты, использующиеся при задании интегрируемой ф-ии
extern double b1[10], p1[10];
```

```
//Массивы, связанные с матрицей Вандермонда. Первый индекс
//массива vand - гладкость, второй - номер строки, третий -
//номер столбца. Первый индекс масс. a - гладкость,
//второй - компонента вектора.
extern double vand[7][10][10], a[7][10], A[7][10];

//Начальная и конечная координаты узлов, доверенных процессу
//(используются одномерные координаты в промежутке [0; V[n-1]]
extern double s, e;

//Целая и дробная часть расстояния о точки на границе до
//соответствующей координатной плоскости
extern ul sigma; extern double eta;

extern double h, h_M;//Шаг решетки и M-я его степень
extern double begin_time, end_time;//Нач. и кон. отметки времени
extern int monit;//Если monit==1, включен режим мониторинга
//Если tol==1, включен режим вычисления погрешности
//по устойчивости знаков
extern int tol;

//Вспомогательные переменные
extern double *times[5], *answers[5],
              koe[100][100], x[10], eps_1_2_inv,
              a1_psi, b1_psi, a2_psi, b2_psi, P[10][10],
              PP0[10], PP1[10], ans;
extern long k_dim;
```

Приложение 2

Главный файл программы вычисления интегралов по квадрату с различными решетками.

```
/*
  Name:        turn.cpp
  Copyright:   Institute of Mathematics, Ufa
  Author:      Rakhmatullin D.Y.
  Description: Main file of the program. The program calculates
               integral of function defined on two-dimentional
               square by using of three types of lattices:
               common square lattice,
               common rectangle lattice
               and turned square lattice.
*/

#include <float.h>
#include <conio.h>
#include <iostream>
#include <fstream>
#include <math.h>
#include <time.h>
#include "proto.h"

char type[10];
int m1, m2, true_answer_exists, pr, full_print, next;
ul N, N_start, N_end, N1, N2, N_new, N_old, N_fact;
li step, factor;
ld ANSWER, true_answer, old_answer, h1, h2, sin1, sin2, eq_eps;
long double alpha,alpha_1,theta,Cos,Sin,C_S,a1,a2,a3,a4,a5,b1,
b2,b3,b4,b5,c1,c2,c3,c4,c5;
clock_t begin_time, end_time;

int main(int argc, char **argv)
{
 start(argc,argv);//Задание нач. значений переменных
 begin_time= clock();//Первая отметка времени

 init_print(type);
 if (strcmp(type,"turn")==0)
  init_turn();

 short sgn= (step<0)? -1 : 1;
```

```
 ul square_bound= ul(sqrt(ULONG_MAX)*(1+eq_eps));
 ul rect_bound=   ul(pow(ULONG_MAX,1./(m1+m2))*(1+eq_eps));
 N_old= 0;

 /*Основной цикл (sqn нужен, чтобы работать при N_start>N_end)*/
 for (N=N_start; sgn*N<=sgn*N_end; N=N*factor+step)
 {
  if (strcmp(type,"square")==0)
  {
   ANSWER= square(N,square_bound);
   if (ANSWER==0x7fc00000)
    continue;
   print("square");
   old_answer= ANSWER;
  }
  if (strcmp(type,"rect")==0)
  {
   ANSWER= rect(N,rect_bound);
   if (ANSWER==0x7fc00000)
    continue;
   print("rect");
   old_answer= ANSWER;
  }
  if (strcmp(type,"turn")==0)
  {
   ANSWER= turn(N,square_bound);
   if (ANSWER==0x7fc00000)
    continue;
   print("turn");
   old_answer= ANSWER;
  }
 }
 end_time= clock();//Вторая отметка времени
 print_time(begin_time,end_time);
}
```

Файл со вспомогательными вычислительными функциями.

```
#include <float.h>
#include <conio.h>
#include <iostream>
#include <fstream>
#include <math.h>
#include <time.h>

#include "proto.h"

inline ul int_part(ld x)
{
```

```
 return (ul)(x*(1+eq_eps));
}

inline int less_(ld x, ld y)
{
 if (x*(1+eq_eps)<y)
  return 1;
 else
  return 0;
}

ld square(ul N, ul square_bound)
{
 ul ip= int_part(sqrt(N));
 if (ip!=square_bound)
  N1= (less_(sqrt(N)-ip,ip+1-sqrt(N)))? ip : ip+1;
 else
  N1= ip;
 N_new= N1*N1;
 if (N_new==N_old)
  return 0x7fc00000;//NaN
 N_old= N_new;
 ld h1= 1./N1;
 ld ANSWER= 0.;
 for (ul i=0;i<N1;i++)
 for (ul j=0;j<N1;j++)
  ANSWER+= fun(i*h1,j*h1);
 ANSWER/= N_new;
 return ANSWER;
}

ld rect(ul N, ul rect_bound)
{
 ul ip= int_part(pow(N,1./(m1+m2)));
 if (ip!=rect_bound)
  ul root= (less_(pow(N,1./(m1+m2))-ip,ip+1-pow(N,1./(m1+m2))))?
  ip : ip+1;
 else
  ul root= ip;
 /*Количество точек по горизонтали*/
 N1= ul(pow(ip,m2)*(1+eq_eps));
 /*Количество точек по вертикали*/
 N2= ul(pow(ip,m1)*(1+eq_eps));
 N_new= N1*N2;
 if (N_new==N_old)
  return 0x7fc00000;
 N_old= N_new;
 h1= 1./N1;//Шаг по горизонтали
 h2= 1./N2;//Шаг по вертикали
```

```
 ANSWER= 0.;
 for(ul i=0;i<N1;i++)
 for(ul j=0;j<N2;j++)
  ANSWER+= fun(i*h1,j*h2);
 ANSWER/= N_new;
 return ANSWER;
}

ld turn(ld N, ul square_bound)
{
 ul ip= int_part(sqrt(N));
 if (ip!=square_bound)
  N1= (less_(sqrt(N)-ip,ip+1-sqrt(N)))? ip : ip+1;
 else
  N1= ip;
 N_new= N1*N1;
 if (N_new==N_old)
  return 0x7fc00000;//NaN
 N_old= N_new;
 ld h1= 1./N1;
 ld ANSWER= 0.;
 ld x,y;
 N_fact= 0;

 for(li i=li(-N1);i<li(N1);i++)
 for(ul j=0;j<2*N1;j++)
 {
  x= i*h1;
  y= j*h1;
  if
   (
    ((a1<=x)&&(x<a2) && (a3*x<=y)&&(y<=a4*x+a5))
    ||
    ((b1<=x)&&(x<b2) && (b3*x<=y)&&(y<b4*x+b5))
    ||
    ((c1<=x)&&(x<c2) && (c3*x<y)&&(y<c4*x+c5))
   )
   {
    ANSWER+= fun(c2*x-a1*y,a1*x+c2*y);
    N_fact++;
   }
 }
 ANSWER/= N_fact;
 return ANSWER;
}
```

Второй файл с вспомогательными вычислительными функциями.

```
#include <float.h>
#include <conio.h>
#include <iostream>
#include <fstream>
#include <math.h>
#include <time.h>

#include "proto.h"

inline int eq(ld x, ld y)
{
 if (fabs(x-y)<=eq_eps*x)
  return 1;
 else
  return 0;
}

inline int less_eq(ld x,  ld y)
{
 if ((fabs(x-y)<eq_eps*x)||(x<y))
  return 1;
 else
  return 0;
}

ld xi (ld t)
{
 ld answer;
 int M= 4;

 if (t<0)
  answer= 0.0;
 else
  if (t<epsilon)
   switch (M)
   {
    case 2: answer= ((6*t-15)*t+10)*t*t*t;
     break;
    case 3: answer= (((-20*t+70)*t-84)*t+35)*t*t*t*t;
     break;
    case 4: answer= ((((70*t-315)*t+540)*t-420)*t+126)
                         *t*t*t*t*t*t;
     break;
    case 5: answer= (((((-252*t+1386)*t-3080)*t+3465)*t-1980)
     *t+462)*t*t*t*t*t*t*t;
     break;
    case 6: answer= ((((((924*t-6006)*t+16380)*t-24024)*t+20020)
                          *t-9009)*t+1716)*t*t*t*t*t*t*t*t;
    break;
```

```
   }
  else
   answer= 1.0;
 return answer;
}

inline ld phi (ld x)
{
 ld answer= xi(2*epsilon*x)*xi(2*epsilon*(1-x));
 return answer;
}

ld fun(ld x1, ld x2)
{
  return sin(sin1*x1)*sin(sin2*x2)*phi(x1)*phi(x2);
}

int start(int argc, char **argv)
{
 //Начальные значения переменных:
 N_start= N_end= 100;
 factor= 2;
 step= 0;
 m1= m2= 1;
 alpha= (1+sqrt(5))/2;
 eq_eps= DBL_EPSILON;
 true_answer_exists= 0;
 pr= DBL_DIG;
 full_print= 0;
 next= 1;
 sin1= sin2= 0.;

 char *ch,*c;
 if (argc>1)
  for (int i=1; i<argc; i++)
  {
   ch= strdup(argv[i]);
   c= strdup("m1");
   if ((strcmp(argv[i],"m1")==0)&&(i+1<argc))
    m1= atoi(argv[i+1]);
   c= strdup("m2");
   if ((strncmp(ch,c,strlen(ch))==0)&&
       (strlen(ch)==strlen(c))&&(i+1<argc))
    m2= atoi(argv[i+1]);
   c= strdup("N");
   if ((strncmp(ch,c,strlen(ch))==0)&&
       (strlen(ch)==strlen(c))&&(i+1<argc))
   {
    N_start= N_end= atol(argv[i+1]);
```

```
 if ((i+2<argc)&&(atol(argv[i+2])!=0))
 {
  N_end= atol(argv[i+2]);
  if ((i+3<argc)&&(atol(argv[i+3])!=0))
  {
   factor= atol(argv[i+3]);
   if ((i+4<argc)&&(atol(argv[i+4])!=0))
    step= atol(argv[i+4]);
  }
 }
}
c= strdup("tan");
if ((strncmp(ch,c,strlen(ch))==0)&&(strlen(ch)==strlen(c)))
 alpha= atof(argv[i+1]);
c= strdup("square");
if ((strncmp(ch,c,strlen(ch))==0)&&(strlen(ch)==strlen(c)))
 strcpy(type,"square");
c= strdup("rect");
if ((strncmp(ch,c,strlen(ch))==0)&&(strlen(ch)==strlen(c)))
 strcpy(type,"rect");
c= strdup("turn");
if ((strncmp(ch,c,strlen(ch))==0)&&(strlen(ch)==strlen(c)))
 strcpy(type,"turn");
c= strdup("sin1");
if ((strncmp(ch,c,strlen(ch))==0)&&(strlen(ch)==strlen(c))&&
    (i+1<argc))
 sin1= atof(argv[i+1]);
c= strdup("sin2");
if ((strncmp(ch,c,strlen(ch))==0)&&(strlen(ch)==strlen(c))&&
    (i+1<argc))
 sin2= atof(argv[i+1]);
c= strdup("eq_eps");
if ((strncmp(ch,c,strlen(ch))==0)&&(strlen(ch)==strlen(c))&&
    (i+1<argc))
 eq_eps= atof(argv[i+1]);
c= strdup("pr");
if ((strncmp(ch,c,strlen(ch))==0)&&(strlen(ch)==strlen(c))&&
    (i+1<argc))
 pr= atoi(argv[i+1])-1;
c= strdup("next");
if ((strncmp(ch,c,strlen(ch))==0)&&(strlen(ch)==strlen(c)))
 next= atoi(argv[i+1]);
c= strdup("fp");
if ((strncmp(ch,c,strlen(ch))==0)&&(strlen(ch)==strlen(c)))
 full_print= 1;
c= strdup("ta");
if ((strncmp(ch,c,strlen(ch))==0)&&(strlen(ch)==strlen(c))&&
    (i+1<argc))
{
```

```
    true_answer= atof(argv[i+1]);
    true_answer_exists= 1;
   }
  }

 if (eq(sin1*sin2,0.))
  if(eq(sin1+sin2,0.))
  {
   sin2= 2.;
   sin1= pow(sin2,double(m2)/double(m1));
  }
  else
   if (eq(sin1,0.))
     sin1= pow(sin2,double(m2)/double(m1));
   else
     sin2= pow(sin1,double(m1)/double(m2));
 if ((step==1)&&(N_start>N_end))
  step= -1;
}

void init_turn()
//Вычисление коэффициентов прямых, ограничивающих
//повернутую решетку
{
 alpha_1= pow(alpha,-1);//1/alpha;
 Cos= pow(1+alpha*alpha,-0.5);//1/sqrt(1+alpha*alpha);
 Sin= alpha*pow(1+alpha*alpha,-0.5);///sqrt(1+alpha*alpha);
 C_S= Cos-Sin;
 a1= -Sin;
 a2= C_S;
 a3= -alpha_1;
 a4= alpha;
 a5= pow(Cos,-1);
 b1= a2;
 b2= 0;
 b3= -alpha_1;
 b4= -alpha_1;
 b5= pow(Sin,-1);
 c1= b2;
 c2= Cos;
 c3= alpha;
 c4= -alpha_1;
 c5= pow(Sin,-1);
}
```

Файл с функциями печати.

```
#include <iostream>
#include <iomanip>
```

```
#include "proto.h"
ofstream output;//Описание потока для вывода результатов

void init_print(char *ch)
{
 //Создание файла output.txt
 output.open("output.txt",ios::trunc);
 output << scientific;
 if (strcmp(ch,"square")==0)
  output<< "<<<Square lattice>>>" << endl;
 if (strcmp(ch,"rect")==0)
  output<< "<<<Rectangle lattice>>>" << endl;
 if (strcmp(ch,"turn")==0)
  output<< "<<<Turned lattice>>>" << endl;
 if (strcmp(ch,"rect")==0)
 {
  output << "m1:\t" << m1 << endl;
  output << "m2:\t" << m2 << endl;
 }
 if (!full_print)
  output << "N\tAnswer\t\t\t\tAbsolute error\tRelative error"
         << endl;
  output.precision(pr);
}

void print(char *ch)
{
 if (full_print)
 {
  output<< "Original N:\t" << N << endl;
  output<< "Rounded N:\t" << N_new << endl;
  if (strcmp(ch,"rect")==0)
  {
   output << "N1:\t" << N1 << "\th1:\t" << h1 << endl;
   output << "N2:\t" << N2 << "\th2:\t" << h2 << endl;
  }
  if (strcmp(ch,"turn")==0)
  {
   output << "N_fact:\t" << N_fact << endl;
   output << "alpha:\t\t" << alpha << endl;
  }
  output<< "Answer:\t\t" << ANSWER << endl;
  if (true_answer_exists)
  {
   output<< "True answer:\t" << true_answer << endl;
   output<< "" << ANSWER-true_answer << endl;
   if (true_answer)
    output << "Relative error:\t" <<
```

```
    (ANSWER-true_answer)/true_answer << endl;
  }
  output<< "\n";
 }
 else
 {
  if (N>N_start && !true_answer_exists)
   output << "\t" << ANSWER-old_answer << "\t" <<
   (ANSWER-old_answer)/ANSWER << endl;
  output << resetiosflags(ios::showpos);
  if (strcmp(ch,"turn")==0)
   output << N_fact << "\t";
  else
   output << N_new << "\t";
  output << showpos;
  output.precision(pr);
  output << ANSWER << "\t";
  output.precision(2);
  if (true_answer_exists)
  {
   output << ANSWER-true_answer << "\t";
   output << (ANSWER-true_answer)/true_answer << endl;
  }
 }
}

void print_time(clock_t b, clock_t e)
{
 output << fixed << resetiosflags(ios::showpos);
 output.precision(2);
 output << "\nTime elapsed: " <<'\t'<<((double)(e-b))/CLK_TCK;
}
```

Файл, содержащий прототипы глобальных переменных.

```
#include <float.h>
#include <conio.h>
#include <iostream>
#include <fstream>
#include <math.h>
#include <time.h>

#define epsilon 1.0

using namespace std;
typedef unsigned long ul;
typedef double ld;
typedef long int li;
typedef unsigned long ul;
```

```
extern char type[10];
extern int m1, m2, true_answer_exists, pr, full_print, next;
extern ul N, N_start, N_end, N1, N2, N_new, N_old, N_fact;
extern li step, factor;
extern ld ANSWER, true_answer, old_answer, h1, h2, sin1, sin2,
          eq_eps;
extern long double alpha,alpha_1,theta,Cos,Sin,C_S,a1,a2,a3,
                   a4,a5,b1,b2,b3,b4,b5,c1,c2,c3,c4,c5;
extern clock_t begin_time, end_time;

inline ul int_part(ld x);
inline int eq(ld x, ld y);
inline int less_(ld x, ld y);
inline int less_eq(ld x, ld y);
inline ld phi (ld x);
ld square(ul N, ul square_bound);
ld rect(ul N,  ul rect_bound);
ld turn(ld N, ul);
ld xi (ld t);
ld fun(ld x1, ld x2);
int start(int argc, char **argv);
void init_print(char*);
void init_turn();
void print(char*);
void print_time(clock_t b, clock_t e);
```