Федеральное государственное автономное образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Институт математики и фундаментальной информатики
Базовая кафедра вычислительных и информационных технологий

УТВЕРЖДАЮ
Заведующий кафедрой
В.В. Шайдуров
«21» июня 2016 г.

БАКАЛАВРСКАЯ РАБОТА

Направление 02.03.01 Математика и компьютерные науки

ВЫЧИСЛИТЕЛЬНЫЙ АЛГОРИТМ ПОСТРОЕНИЯ ПОВЕРХНОСТИ РЕЛЬЕФА МЕСТНОСТИ НА ОСНОВЕ ВЕКТОРНЫХ КАРТ В ПОЛЬСКОМ ФОРМАТЕ

Научный руководитель
кандидат физико-математических наук 21.06.2016 Е.В. Кучунова

Выпускник 21.06.2016 А.В. Рулёв

Красноярск 2016

РЕФЕРАТ

Выпускная бакалаврская работа по теме «Вычислительный алгоритм построения поверхности рельефа местности на основе векторных карт в польском формате» содержит 38 страниц текстового документа, 21 рисунок, 0 таблиц, 2 приложения, 5 использованных источников литературы.

ВЕКТОРНЫЕ КАРТЫ, ПОЛЬСКИЙ ФОРМАТ, ИНТЕРПОЛЯЦИОННЫЕ СПЛАЙНЫ, ЦИФРОВЫЕ МОДЕЛИ МЕСТНОСТИ.

Целью бакалаврской работы является построение цифровой модели местности по векторным картам в польском формате; разработка вычислительного алгоритма построения цифровой модели рельефа на основе изолиний; реализация вычислительного алгоритма в виде программного комплекса на языке C++ и тестирование вычислительного алгоритма на серии тестовых задач.

В работе представлен алгоритм построения цифровой модели рельефа, который восстанавливает матрицу высот по заданному набору изолиний, представлены результаты вычислительных экспериментов.

Данная программа может быть использована при проведении исследований.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

С появлением первых компьютеров возможности хранения информации многократно возросли и практически сразу же были сделаны попытки реализовать на компьютере карту и связать с каждым объектом карты его описание, внесенное в базу данных.

Цифровая модель очень удобна при использовании, поскольку ее можно вращать, посмотреть на нее с любой точки и получить более полное представление о рельефе.С помощью цифровых моделей можно быстро получить информацию о высоте, угле наклона, экспозиции склона в любой точке модели.При использовании цифровых моделей значительно увеличивается возможность обычного пользователя проложить более удобный маршрут для подъема в гору или найти более подходящий рельеф.

В начале 2000 годов появился формат представления геоданных имеющий простую и хорошо документированную структуру. Этот формат получил название «*польский формат*» или формат MP. Польский формат (формат для векторных карт) — это текстовый формат хранения данных, используемый как в качестве промежуточного формата при конвертировании между различными промышленными форматами электронных карт, так и для их редактирования. Название свое получил потому, что изначально был разработан польскими программистами.

Польский формат можно легко расширять и дополнять своими объектами или полями объектов, что делает его удобным при разработке картографического ПО со специфическими требованиями. Расширенные данные польского формата сохраняют совместимость с программами, работающими со стандартной структурой польского формата.

Изолиния (*линия уровня*) - условное обозначение на карте, чертеже, схеме или графике, представляющее собой линию, в каждой точке которой измеряемая величина сохраняет одинаковое значение.

Полилиния - это замкнутая или незамкнутая последовательность соединенных между собой отрезков или дуговых сегментов, которые рассматриваются как цельный объект.

Цель работы: построить цифровую модель местности по векторным картам в польском формате. В качестве исходных данных выступают наборы изолиний. Каждая из изолиний представляет собой набор точек местности, имеющих одинаковую высоту.

Строится цифровая модель с помощью прямоугольной сетки, в узлах которой заданы значения *z* (высоты). Прямоугольная сетка обладает преимуществом простой и удобной структуры, требующей хранения только значений высот и позволяющей упростить алгоритмы анализа и обработки поверхности.

Для достижения поставленной цели в бакалаврской работе требовалось решить следующие задачи.

1. Разработать вычислительный алгоритм построения поверхности рельефа.
2. Реализовать разработанный алгоритм на языке C++ в видеприкладной программы.
3. Проверить разработанный алгоритм на тестовых задачах.

1 Обзор цифровых моделей рельефа

1.1 Определение модели TIN

Модель TIN (*TriangulationIrregularNetwork*– триангуляционная нерегулярная сеть) географических объектов – модель поверхности в виде сети смежных не пересекающихся треугольных граней, определенная по узлам и ребрам, которые покрывают поверхность.

Геометрия модели TIN образуется гранями, узлами и ребрами в трехмерном пространстве. Каждая грань TIN является частью поверхности в 3D-пространстве.

Модель TIN обладает следующими свойствами.

1 Модель TIN позволяет получить точное представление о локальной части поверхности, используя переменную плотность узлов со значением Z и линии перегиба поверхности;

2 Модель TIN является основой 3D-визуализации поверхности;

3 Модель TIN позволяет выполнить сложный анализ поверхности (вычисление высот, уклонов, экспозиций склонов, получение изолиний поверхности, расчеты объемов, вертикальные профили по трассе линии, анализ видимости).

На рис. 1 продемонстрирован пример модели TIN.

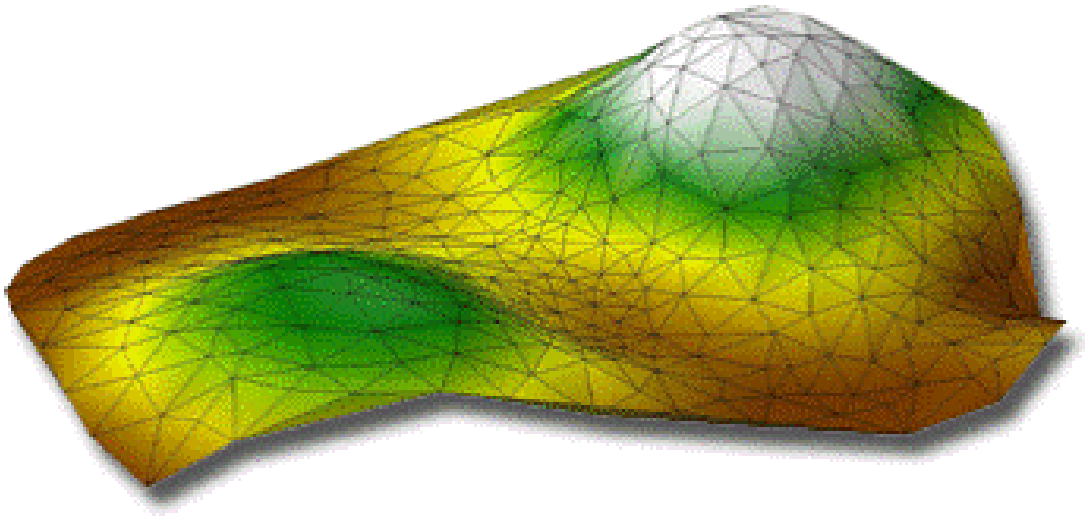

Рисунок 1- Модель TIN

1.2 Растровое представление поверхности

Это формат представления поверхности в виде матрицы равномерно распределенных точек, каждая из которых характеризуется своимпараметром *Z*. Параметром *Z*может являться высота местности, количество особей на данной территории и т.д.

Существует две основные модели – «решеточная» и «ячеистая» (рис. 2).Первая модель представлена в виде значений интерполируемых по значениям Z в нескольких соседних точках.Во второймоделииспользуется коллекция центральных точек ячеек со значениями *Z*, расположенных регулярно через горизонтальные интервалы.

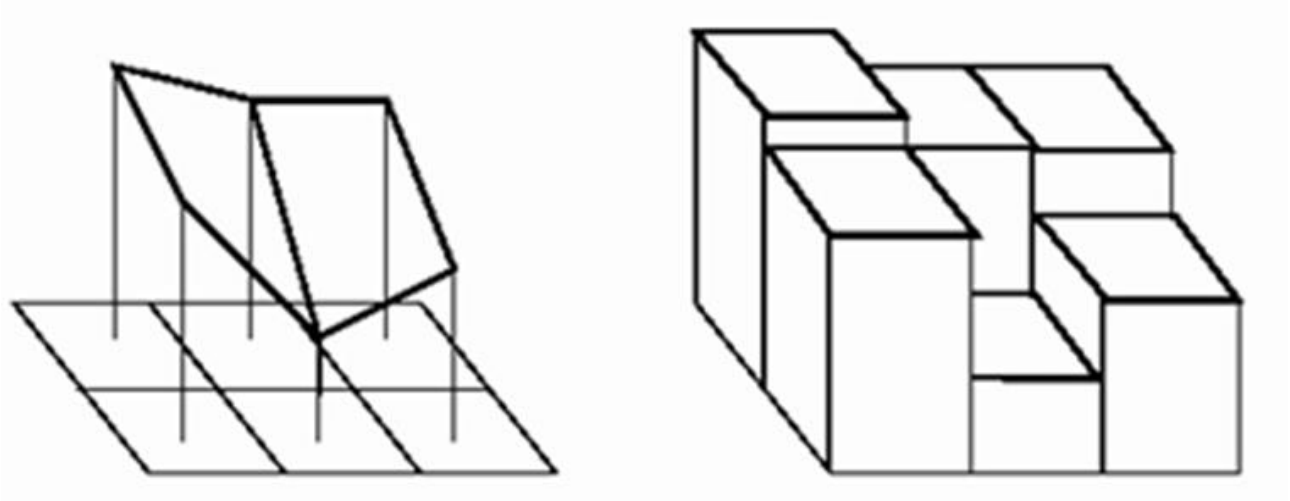

Рисунок 2- Растровые модели

Преимущества растрового представления поверхности:

1) простота модель;
2) применение в приложениях, где точность местоположений не имеет преимущественного значения и не требуется точного представления пространственных объектов поверхности.

Недостатки растрового представления поверхности:

1) регулярная структура не приспособлена к изменениям сложного рельефа;
2) разрывы непрерывности передаются недостаточно хорошо;
3) точные местоположения точек вершин, дна теряются.

По сравнению с векторными моделями растровые модели обладают следующими недостатками:

- географические объекты характеризуются менее точной информацией о местоположении и размерах;
- растры требуют больших объемов памяти.

Представление географических объектов растровыми моделями имеет следующие преимущества:

- растр отображает непрерывно охватываемую территорию;
- растровые данные проще для обработки и обеспечивают более высокое быстродействие;
- ввод растровых данных менее трудоемкий.

В растровых моделях выборочные точки расположены в узлах регулярной растровой решетки, а в триангуляционных сетях располагаются нерегулярно так, чтобы наилучшим образом обогнуть поверхность.

2 Построение поверхности рельефа местности

2.1 Постановка задачи

Пусть имеется некоторая исходная прямоугольная область $\Omega = [x_{min}, x_{max}] \times [y_{min}, y_{max}]$, в которой задан набор изолиний $L = \{l_0, l_1, \dots, l_{m-1}\}$. Каждая изолиния представляет собой пару$l_k =< h_k, V_k >$, где h_k- высота набора вершин $V_k = \{(x_i^k, y_i^k), i = 0, \dots, n_k\}$. Предположим набор вершин V_k является упорядоченным вдоль изолинии l_k(в противном случае, вершины легко переупорядочить). Каждая изолиния может быть как замкнутой, так и не замкнутой.

Построим в $\overline{\Omega}$равномерную прямоугольную сеткуΩ_h:

$\Omega_h = \{(x_i, y_j);\ x_i = x_{min} + i * h_x, y_j = y_{min} + j * h_y; i = 0, \dots, N; j = 0, \dots, M\}$,

где N, M - количество узлов сетки вдоль осей OX и OY соответственно, $h_x = \frac{x_{max} - x_{min}}{N}$,$h_y = \frac{y_{max} - y_{min}}{M}$.

Требуется вычислить значение высоты z_{ij} в каждой точке Ω_hтаким образом, чтобы полученная поверхность интерполировала бы весь заданный набор изолиний L.

Пример исходного набора изолиний изображен на рис. 3.

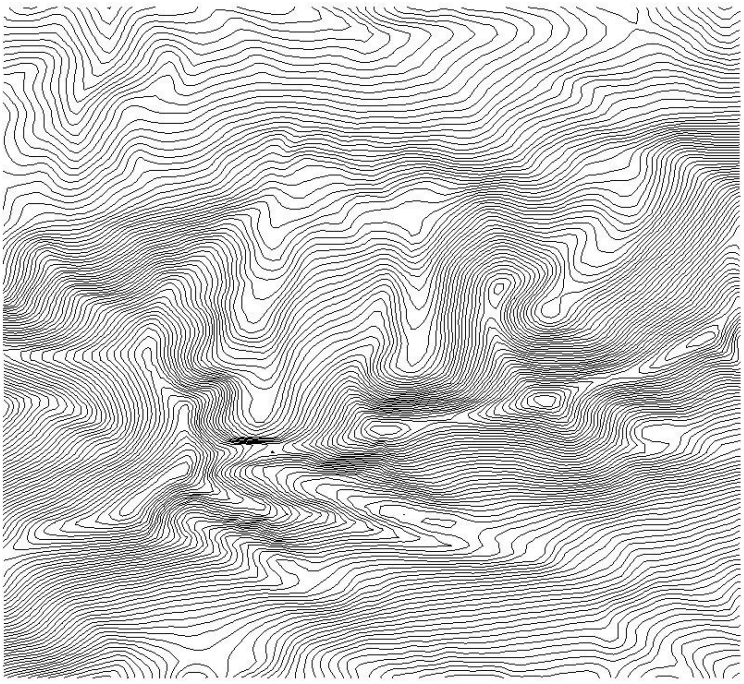

Рисунок 3- Пример исходного набора изолиний

2.2 Вычислительный алгоритм

Представляемый вычислительный алгоритм разбивается на три этапа.

1. Получение данных об исходном наборе изолиний.
2. Перенос данных о высотах изолиний в узлы сетки.
3. Получение поверхности с помощью сплайн-интерполяции.

Этап №1. Получение данных об исходном наборе изолиний

Данные об исходном наборе изолиний хранится в текстовом файле с расширением .mp. В этом файле хранится информацию о всех объектах местности (озера, реки и ручьи, тропы, изолинии, участки леса, здания и сооружения). Все объекты задаются при помощи полилиний и полигонов (замкнутых полилиний), содержащих физические координаты (широта и долгота) набора точек. Из всего набора данных нам необходимо выделить только изолинии и считать координаты их вершин.

Этап №2. Перенос данных о высотах изолиний в узлы сетки

Рассмотрим отдельно некую изолинию $l =< h, V >$множестваL. Для простоты изложения опустим индекс, означающий ее номер. Предположим, что множество Vэтой изолинии содержит nвершин $V = \{V_0, V_1, \dots, V_{n-1}\}$. Необходимо перенести информацию о этой изолинии во все узлы сетки Ω_hрасполагающиеся вдоль изолинииl. Для этого дополняем изолинию точками при помощи линейной интерполяции.

Рассмотрим две соседних точки V_jи V_{j+1}изолинии l. Необходимо найти пересечение прямой линии V_jV_{j+1}со всеми линиями сетки Ω_h. Для этого разбиваем отрезок V_jV_{j+1} на систему подотрезков $\{v_0, v_1, \dots, v_k\}$так, чтобы $v_0 = V_j$, $v_k = V_{j+1}$и расстояние между соседними точками не превосходило шаг сетки: $|v_j - v_{j-1}| < \frac{h_x+h_y}{2}$.

Далее для каждой точки v_jотрезка V_jV_{j+1}вычисляется ближайший узел (x_i^k, y_i^k) сеткиΩ_h, которому приписывается высота hизолинииl.

Результат переноса отрезка на сетку изображен на рис. 4.

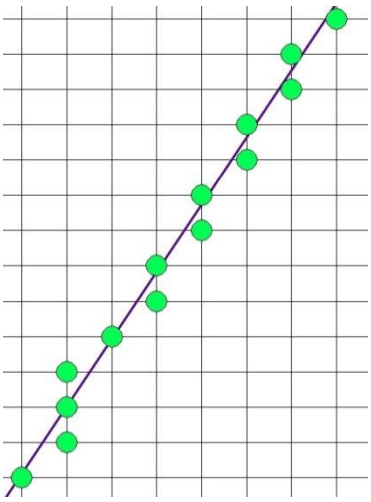

Рисунок 4- Перенос отрезка изолинии на сетку

Таким образом, изолиния lаппроксимируется набором узлов сеткиΩ_hдля которых высота является постоянной и равнаh. Применяя данный алгоритм ко всем изолиниям набора L получаем множество узлов сетки Ω_hс известными высотами. Пример такого множества вершин изображен на рис.5.

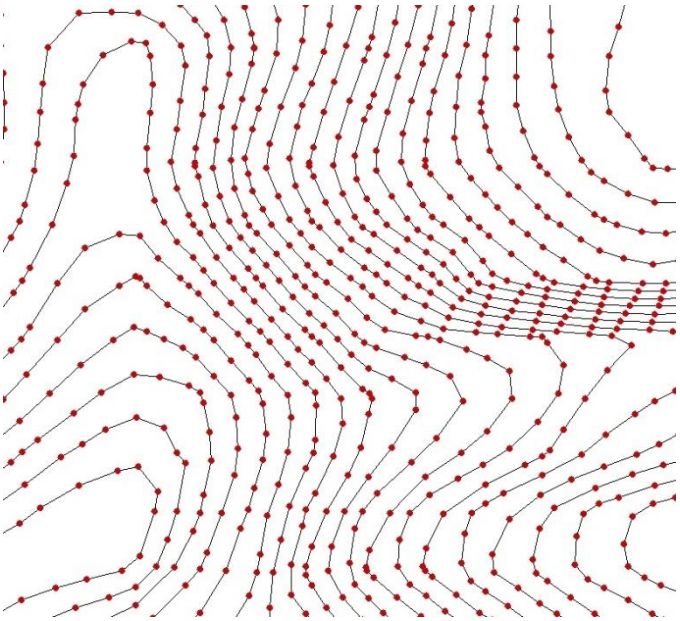

Рисунок 5- Перенос изолиний набора на сетку

Этап №3. Получение поверхности с помощью сплайн-интерполяции

Следующей задачей является вычисление высот во всех узлах сетки Ω_h.

Рассмотрим некоторый разрез области Ωвдоль оси OX (зафиксируем значение $y = const$). Вдоль этого разреза в некотором наборе узлов известны высоты, полученные после переноса изолиний. Во всех остальных узлах построим интерполяционный сплайн первого порядка. Переберем по очереди

все разрезы сетки Ω_h при каждом значении $y = y_j, j = 0,1, \dots, M$ и вдоль каждого разреза построим интерполяционные сплайны. Аналогично поступаем и вдоль оси OY фиксируя по очереди значения $x = x_i,$ для каждого значения $i = 0,1, \dots, N$ строим интерполяционный сплайн. Таким образом, в каждом узле (x_i, y_j) сетки Ω_h получаем вычисленное значение высоты $z_{ij},\ i = 0,1, \dots, N, j = 0,1, \dots, M.$

3 Результаты численных экспериментов

3.1 Описание численных экспериментов

Целью проведения вычислительных экспериментов является проверка разработанного вычислительного алгоритма. В тестовых задачах берутся различные поверхности с известным аналитическим представлением, для которых вычисляются наборы изолиний. Эти наборы изолиний в дальнейшем используются в качестве исходных данных для построения рельефа местности. Далее вычисляются нормы погрешности в дискретном аналоге пространства L_1:

$$\varepsilon = \int_{\Omega} |z^h(x,y) - z(x,y)| d\Omega,$$

где$z(x,y)$ – точное значение высоты в точке, $z^h(x,y)$ – вычисленное значение высоты. Интеграл поΩвычисляется численно с использованием кубатурных формул трапеций.

Анализируется сходимость вычисленного решения к точному при помощи серии расчетов на сгущающихся сетках и при увеличении количества изолиний.

3.2 Тестовая задача №1

В качестве точного решения возьмем поверхность эллиптического параболоида $z = 100 - (\frac{x^2}{4} + \frac{y^2}{4})$ на области $\Omega = (0,10) \times (0,10)$.

Для начала необходимовычислить набор I изолиний $\{l_0, l_1 \ldots l_{I-1}\}$. Для этого нужно определить шаг по высоте dz. На области Ω вычисляем z_{min} и z_{max},следовательно шаг равен$dz = (z_{max} - z_{min})/I$. Полученыфункции, описывающие изолинии:$dz * i = 100 - (\frac{x^2}{4} + \frac{y^2}{4})$, где i=0,1,…,I-1.

В данном примере изолинии представляют собойокружности с радиусами$r = 2 * \sqrt{(100 - dz * i)}$:

$x^2 + y^2 = 4 * (100 - dz * i)$.

Перейдем в полярные координаты:$x = r * \cos\varphi$, $y = r * \sin\varphi$, где $\varphi \in [0,2\pi]$. Пусть расстояние между точками равноr, тогда $d\varphi = L/r$ и координаты точек изолинии l_iвычисляются по формулам:

$x_j = r * \cos(d\varphi * j)$, $y_j = r * \sin(d\varphi * j)$ где $j = 0,1, \dots, M - 1$.

$M = 2\pi/d\varphi$– количество точек в данной изолинии.

На рис.6показаныпостроенные поверхности по вычисленному набору изолиний на серии сгущающихся сеток (N= 50, 100, 200, 400).

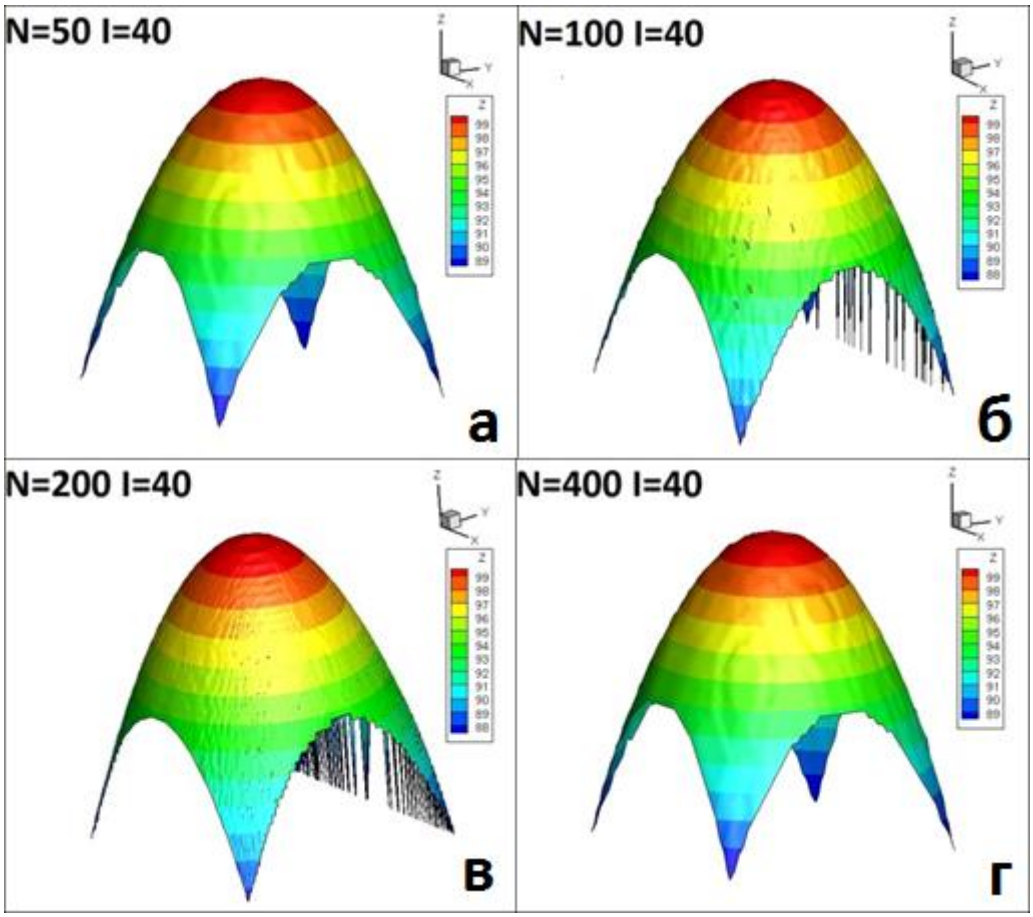

Рисунок 6-Построенные поверхности на серии сгущающихся сеток (а-50 узлов, б-100 узлов, в- 200 узлов г-400 узлов)

На рис.7 представлен график зависимости величины погрешностиε от количества узловв сетке. Из графикавидно, что погрешность стремится к константе.

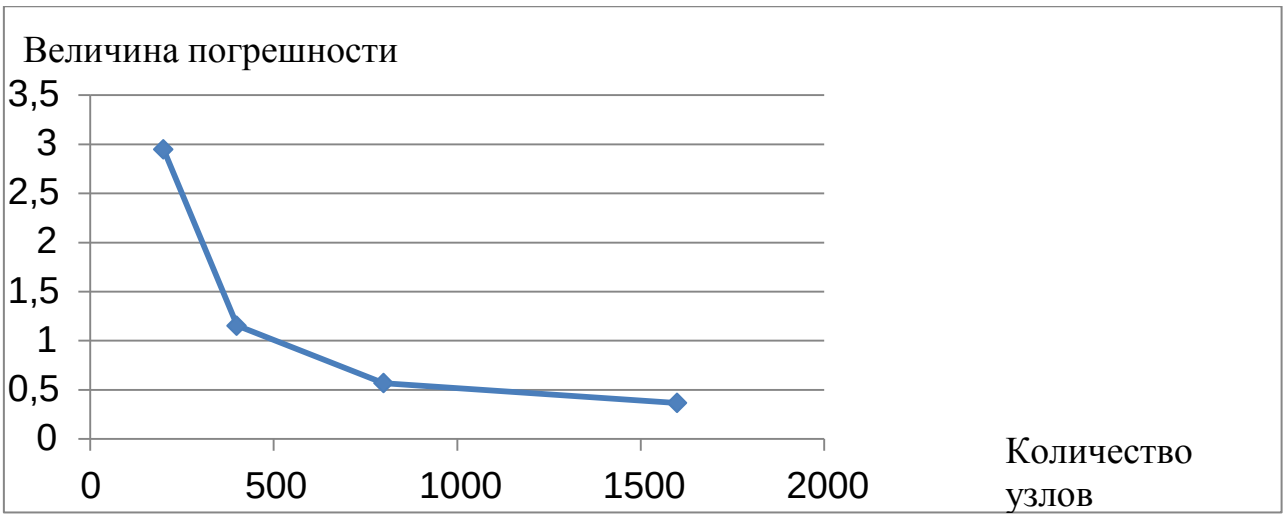

Рисунок 7- График зависимости величины погрешности от количества узлов

На рис.8 показанопостроенные поверхности при разном количестве исходных изолиний (I=40, 80, 160, 320, 640 и 1280) и при фиксированном количестве узлов сетки (N = 200).

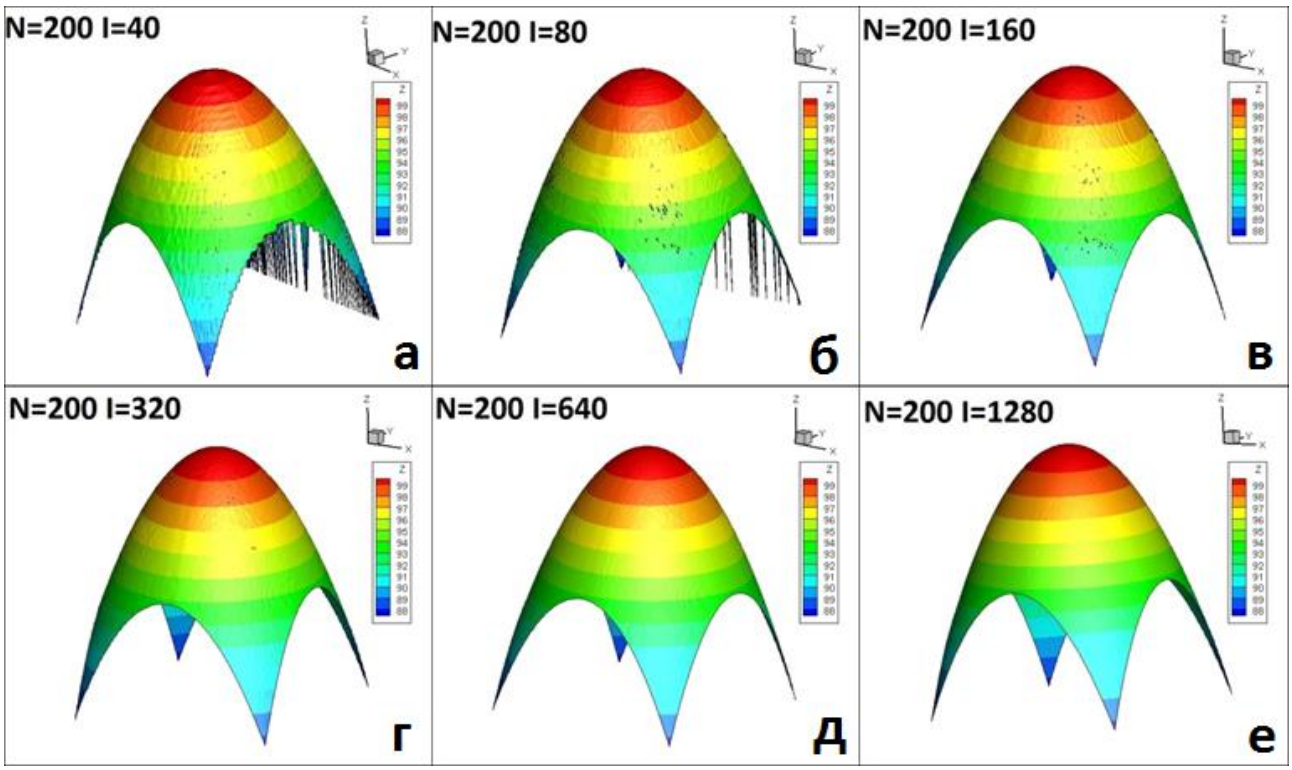

Рисунок 8-Построенные поверхности при различном числе исходных изолиний (а-40 изолиний, б-80 изолиний, в-160 изолиний, г-320 изолиний, д-640 изолиний, е-1280 изолиний)

По графикузависимости величины погрешностиε от количества изолиний (рис. 9) видно, что эта погрешность стремится к нулю.

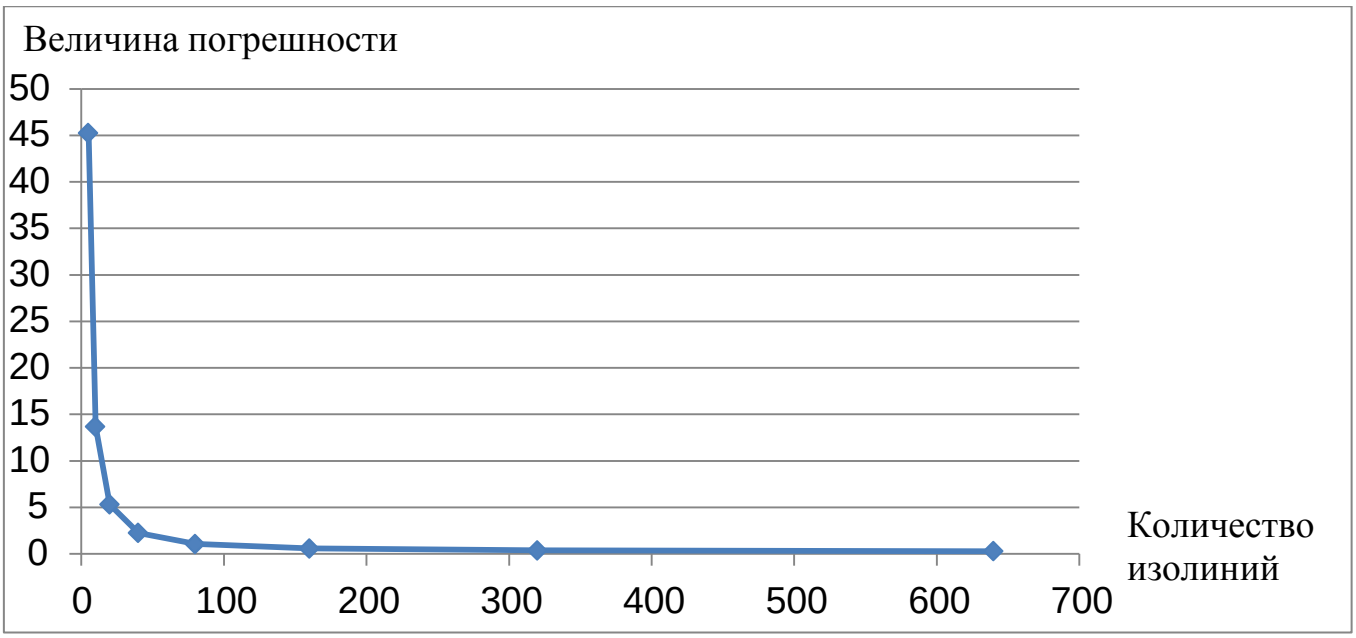

Рисунок 9- График зависимости величины погрешности от количества изолиний

Таким образом, чем больше наборов изолиний и чем больше узлов в сетке, тем больше величина погрешности стремиться к нулю.

3.3 Тестовая задача №2

В качестве точного решения возьмем поверхность гиперболического параболоида $z = \frac{x^2}{4} - \frac{y^2}{4}$ на области $\Omega = (0,10) \times (0,10)$.

Как и в предыдущей тестовой задаче, необходимо по аналитической поверхности вычислить координаты набора изолиний.

Шаг изолиний равен

$$dz = (z_{max} - z_{min})/I,$$

где$z_{min} = \min_{\Omega} z(x,y)$,$z_{max} = \max_{\Omega} z(x,y)$.

Аналитическое представление i-ой изолинии:

$$dz * i = \frac{x^2}{4} - \frac{y^2}{4}$$ где i=0,1,…,I-1.

В данном примере если $dz * i \leq 0$, то изолинии вычисляются по формулам $y = \sqrt{x^2 - 4 * dz * i}$ и $y = -\sqrt{x^2 - 4 * dz * i}$. В этом случае на каждой высоте будет две изолинии. Расстояния между точками равна dx и первая изолиния равна $l_i = \{x_{0,}y_0, x_{1,}y_1, \dots, x_{M-1}y_{M-1}\}$, а вторая изолиния равна $l_{i+1} = \{x'_{0,}y'_0, x'_{1,}y'_1, \dots, x'_{M-1}y'_{M-1}\}$ и$M = (10 - 0)/dx$ –это количество точек в данной изолинии.

И второй случай, когда $dz * i > 0$, то изолинии вычисляются по формулам $x = \sqrt{y^2 + 4 * dz * i}$ и $x = -\sqrt{y^2 + 4 * dz * i}$. В этом случае тоже на каждой высоте будет две изолинии. Расстояния между точками равна dy и первая изолиния равна $l_i = \{x_{0,}y_0, x_{1,}y_1, \dots, x_{M-1}y_{M-1}\}$,а вторая изолиния равна $l_{i+1} = \{x'_{0,}y'_0, x'_{1,}y'_1, \dots, x'_{M-1}y'_{M-1}\}$ и$M = (10 - 0)/dy$ –это количество точек в данной изолинии.

На рис. 10 показаны построенные поверхности по фиксированному количеству изолиний на серии сгущающихся сеток.

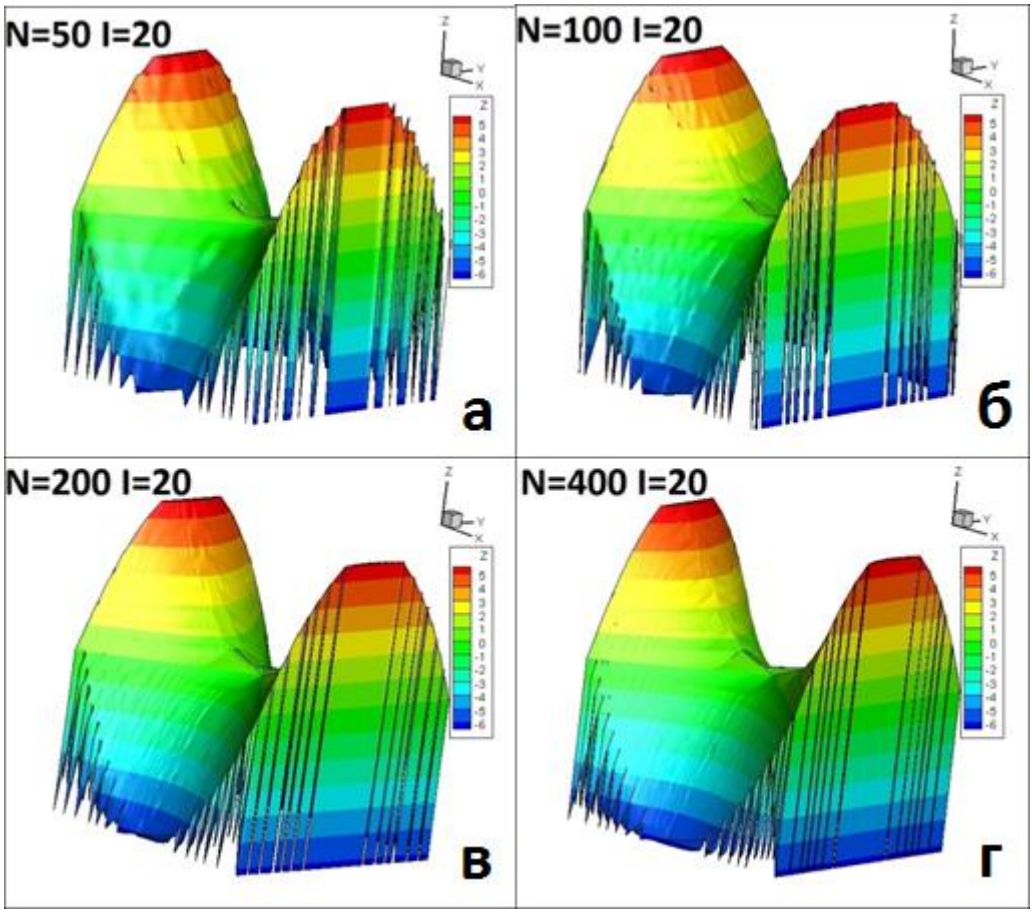

Рисунок 10-Вычисленные поверхности на серии сгущающихся сеток (а-50 узлов, б-100 узлов, в- 200 узлов г-400 узлов)

По графику зависимости погрешности εот количества узлов сетки (рис. 11) видно, что эта погрешность стремится к константе.

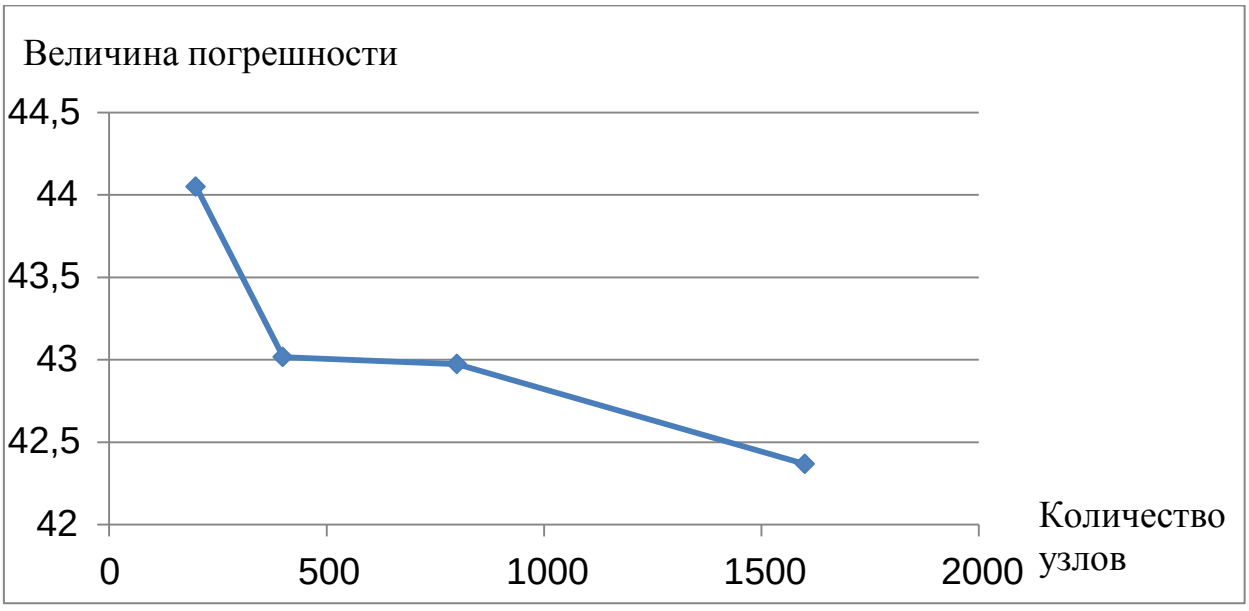

Рисунок 11- График зависимости величины погрешности от количества узлов

На рис. 12 продемонстрированы вычисленные поверхности при фиксированном количестве узлов сетки (N = 1600) и различном количестве изолиний (5, 10, 20, 640, 1280, 2560).

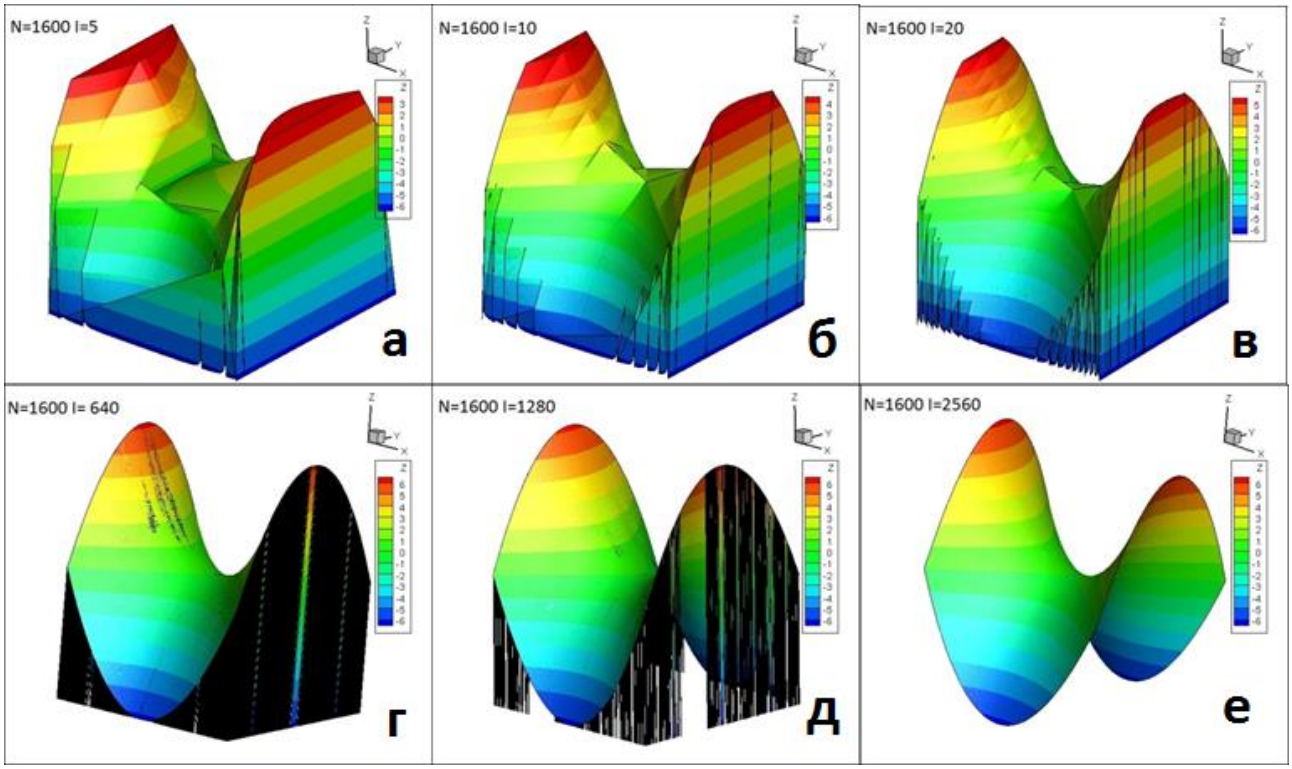

Рисунок 12-Вычисленные погрешности при фиксированном количестве узлов сетки и различным количестве изолиний (а-40 изолиний, б-80 изолиний, в-160 изолиний, г-320 изолиний, д-640 изолиний, е-1280 изолиний)

По графику зависимости погрешности εот количества изолиний (рис. 13) видно, что эта погрешность стремится к нулю.

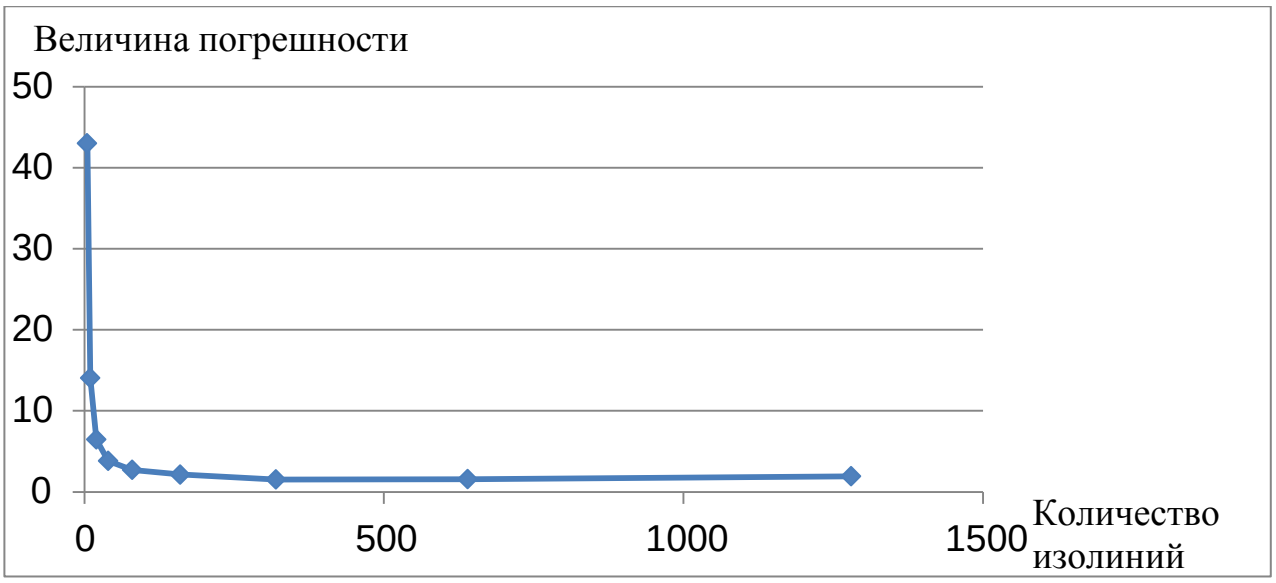

Рисунок 13- График зависимости величины погрешности от количества изолиний

Таким образом, чем больше наборов изолиний и чем больше узлов в сетке, тем больше величина погрешности стремиться к нулю.

3.4 Тестовая задача №3

В качестве точного решения возьмем поверхность, заданную аналитически$z = (\sin x)^2 + (\sin y)^2$ на области $\Omega = (0,10) \times (0,10)$.Она изображена на рис. 14.

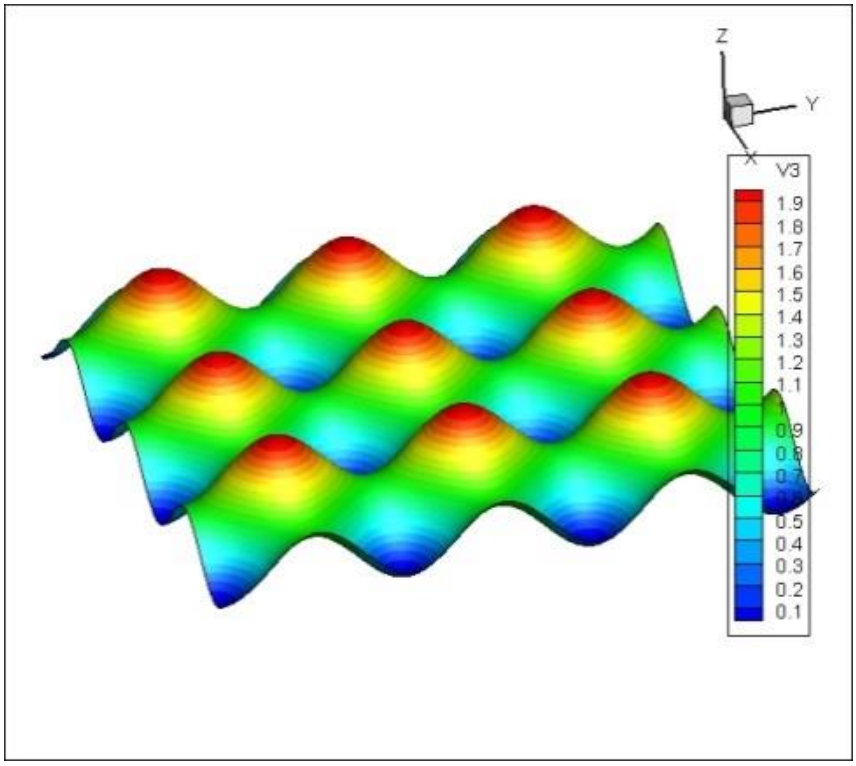

Рисунок 14- Поверхность из синусов

Как и в предыдущей тестовой задаче, необходимо по аналитической поверхности вычислить координаты набора изолиний.

Шаг изолиний равен

$$dz = (z_{max} - z_{min})/I,$$

где$z_{min} = \min_{\Omega} z(x,y)$, $z_{min} = \min_{\Omega} z(x,y)$.

Аналитическое представление i-ой изолинии:

$$dz * i = (\sin x)^2 + (\sin y)^2,$$

где i=0,1,…,I-1.

Перейдем в полярные координаты:$\sin x' = r * \cos\varphi$, $\sin y' = r * \sin\varphi$, где $\varphi \in [0,2\pi]$, то есть $x = x' + a * 2\pi$, $y = y' + b * 2\pi$. Тогда $\sin(x - a * 2\pi) = r * \cos\varphi$, $\sin(y - b * 2\pi) = r * \sin\varphi$, следовательно,$x = \arcsin(r * \cos\varphi) + a * 2\pi$, $y = arcsin(r * \sin\varphi) + b * 2\pi$, где$A = (10 - 0)/2\pi$,$a \in [0, A]$,$B = (10 - 0)/2\pi$,$b \in [0, B]$.На рис. 15 показан график изменения z.

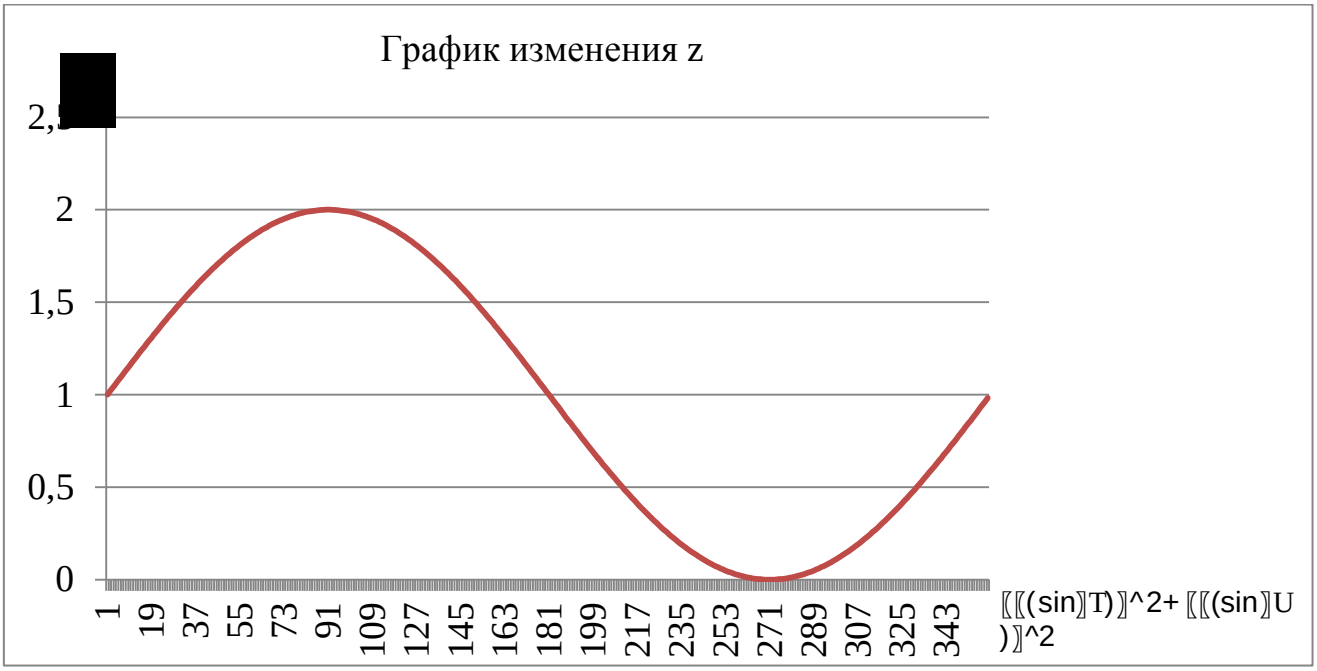

Рисунок 15- График изменения z

В данном примере изолинии представляют собой окружности. При $dz * i < 0$, то радиус окружности равен $r = \sqrt{dz * i}$, а если $dz * i > 0$,то $r = \sqrt{dz * i - 1}$.Пусть в каждой окружности расстояние между точками равно L и $d\varphi = L/r$. Получаем формулы, если$dz * i < 0$, тогда изолинии вычисляются по формуле$r = \sqrt{dz * i}$, $x_j = \arcsin(r * \cos(d\varphi * j)) + a * 2\pi$, $y_j = arcsin(r * \sin(d\varphi * j)) + b * 2\pi$, где $a \in [0, A]$,$b \in [0, B]$ и$j = 0,1, \dots, M - 1$. То изолиния $l_i = \{x_0, y_0, x_1, y_1, \dots, x_{M-1} y_{M-1}\}$ и $M = 2\pi/d\varphi$, где-это количество точек в данной изолинии.

И второй случай, когда $dz * i > 0$, тогда изолинии вычисляются по формуле$r = \sqrt{dz * i - 1}$,$x_j = \arcsin(r * \cos(d\varphi * j)) + a * 2\pi$, $y_j = arcsin(r * \sin(d\varphi * j)) + b * 2\pi$, где$a \in [0, A]$,$b \in [0, B]$, где $j = 0,1, \dots, M - 1$.

$l_i = \{x_{0,}y_0, x_{1,}y_1, \dots, x_{M-1}y_{M-1}\}$ и $M = 2\pi/d\varphi$–это количество точек в данной изолинии.

На рис. 16 приведены изображения вычисленных поверхностей при фиксированном количестве изолиний на серии сгущающихся сеток (N = 50, 100, 200, 400).

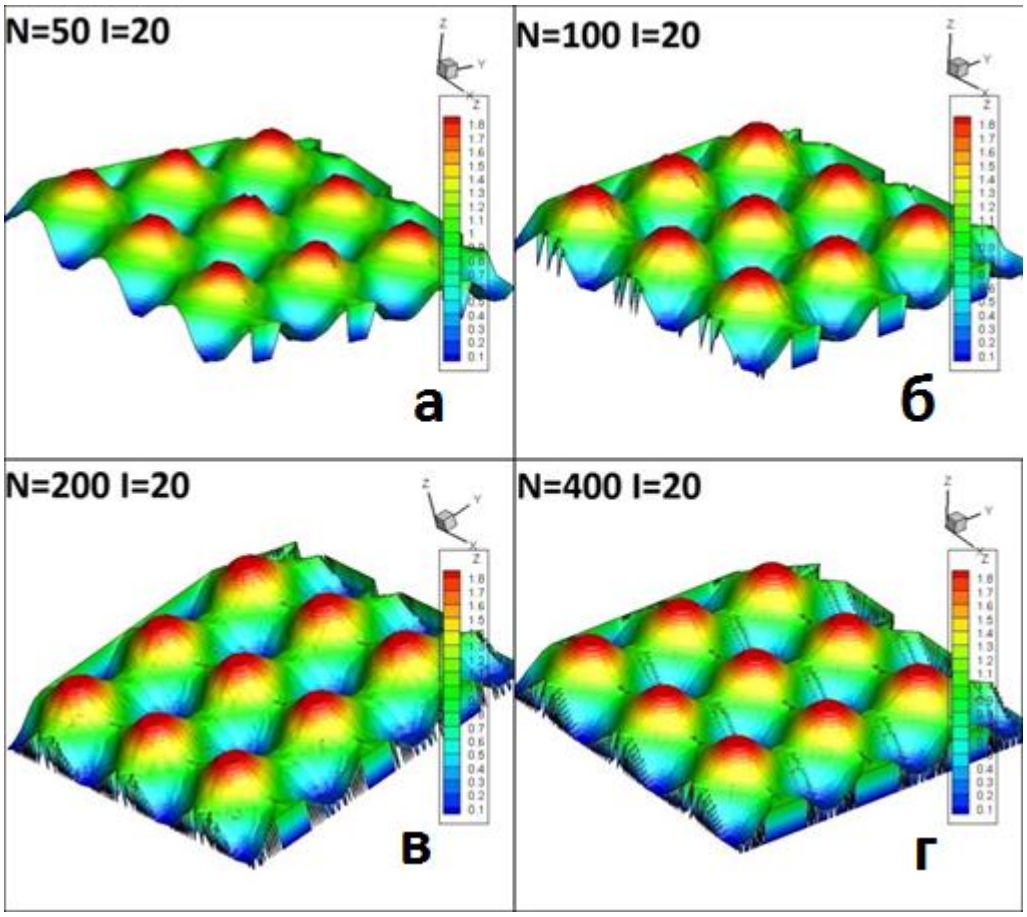

Рисунок 16- Вычисленная поверхность при фиксированном количестве изолиний на серии сгущающихся сеток (а-50 узлов, б-100 узлов, в- 200 узлов,г-400 узлов)

По графику зависимости величины погрешностиε от количества узлов (рис. 17) видно, что эта погрешность стремится к константе.

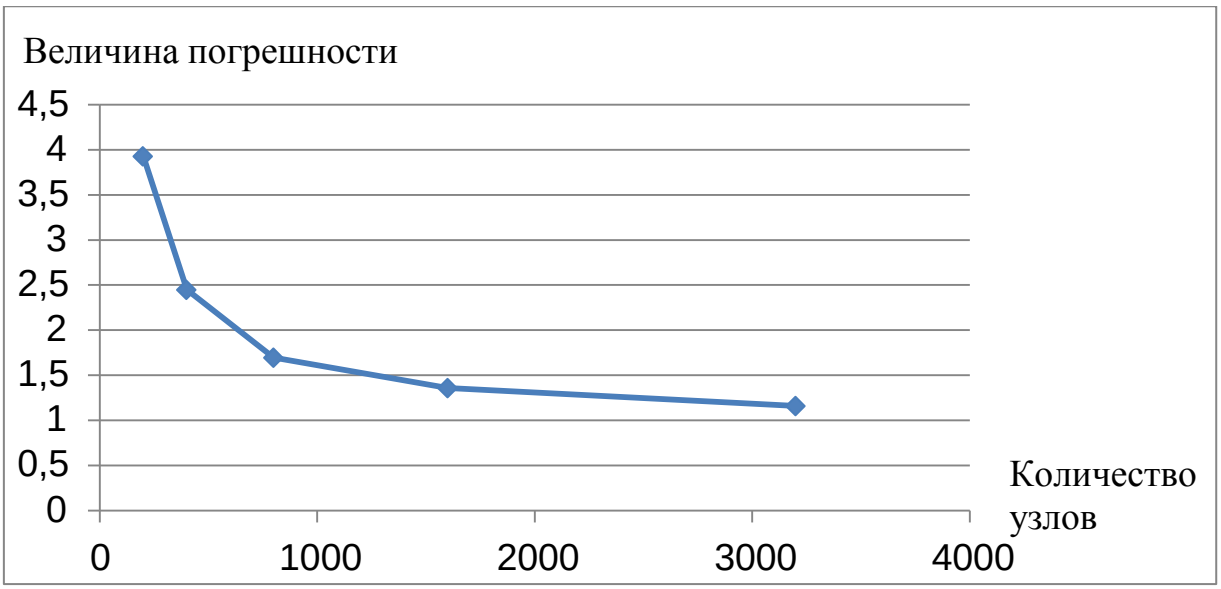

Рисунок 17- График зависимости величины погрешности от количества узлов

На рис.13 показаны вычисленные поверхности при фиксированном количестве узлов сетки и различном количестве изолиний (I = 40, 80, 160, 320, 640 и 1280).

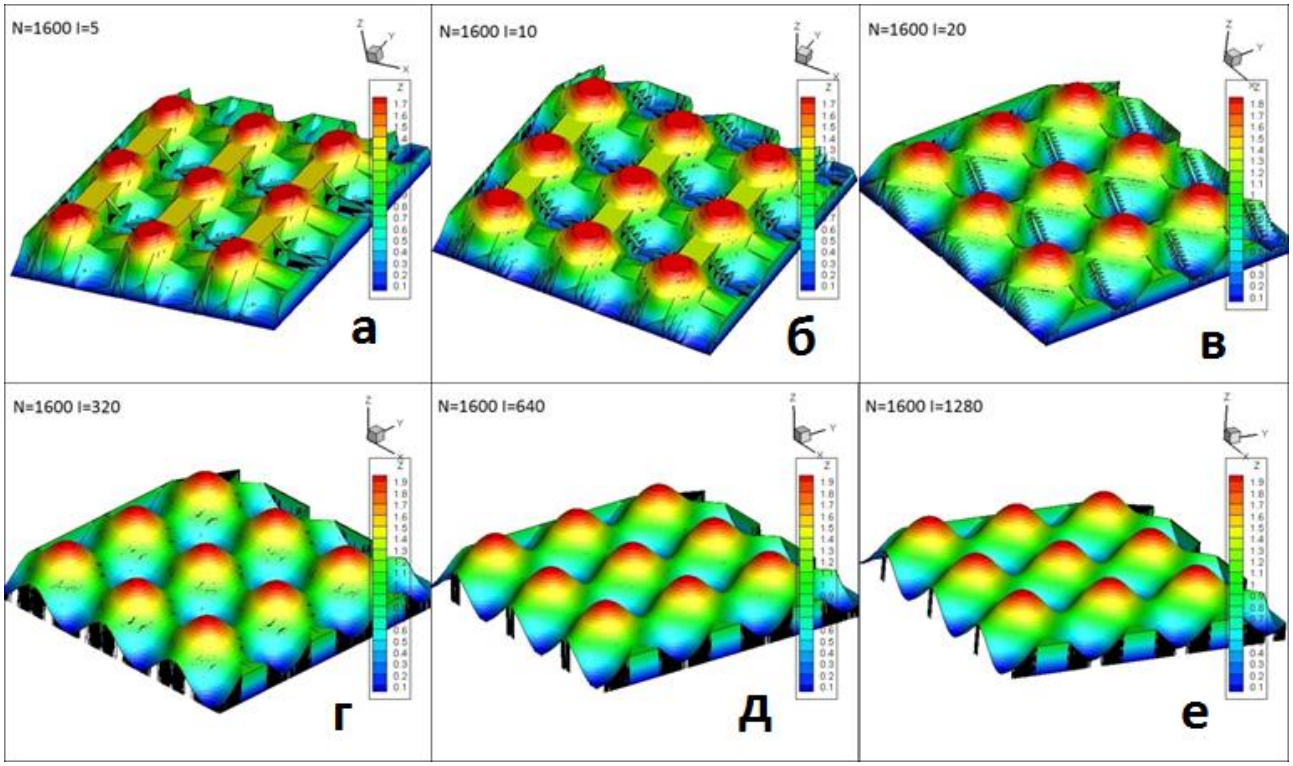

Рисунок 18-Вычисленные поверхности при фиксированном количестве узлов сетки и различном количестве изолиний (а-40 изолиний, б-80 изолиний, в-160 изолиний, г-320 изолиний, д-640 изолиний, е-1280 изолиний)

По графику зависимости величины погрешностиε от количества изолиний (рис. 19) видно, что эта погрешность стремится к константе.

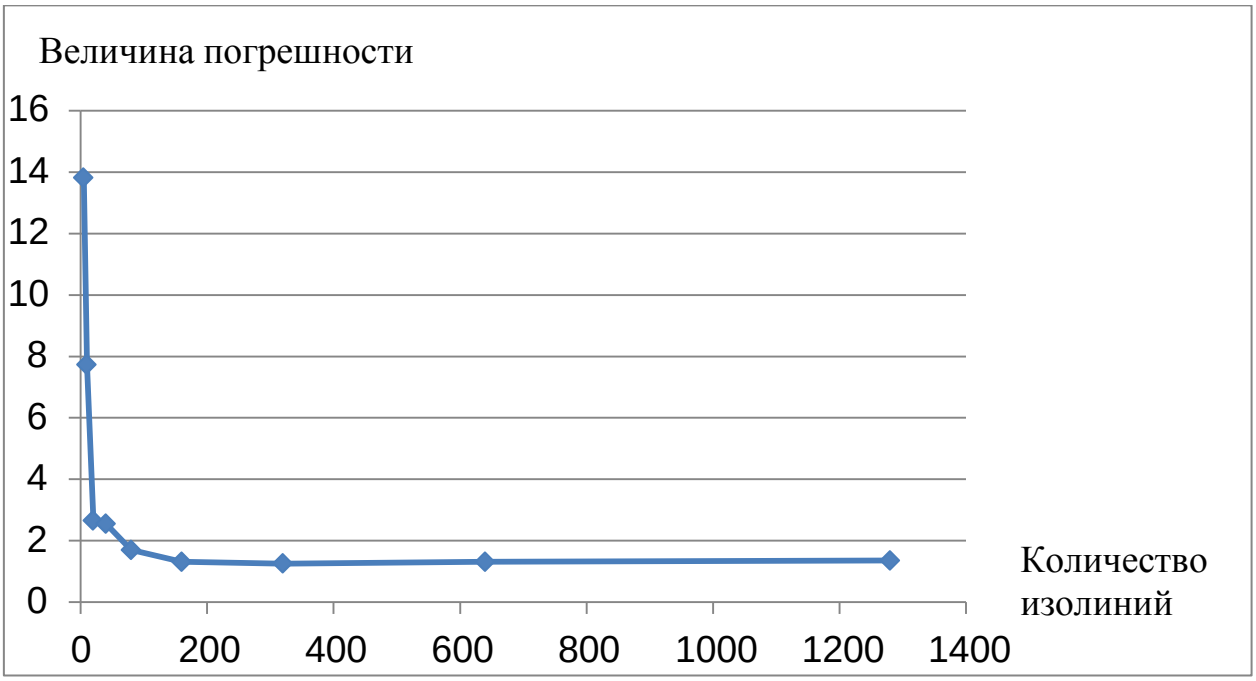

Рисунок 19- График зависимости величины погрешности от количества изолиний

Таким образом, чем больше наборов изолиний и чем больше узлов в сетке, тем больше величина погрешности стремиться к нулю.

3.5 Поверхность рельефа горного хребта Борус

В работе была рассмотрена векторная карта горного хребта Борус (рис.20), расположенного в системе Западного Саяна. Протяженность карты составляет от 91.20º до 91.40º восточной долготы и от 52.42º до 52.54º северной широты. Карта содержит более 500 изолиний, которые в совокупности содержат около $3*10^5$ вершин. Перепад высот составляет от 320 до 2260 метров над уровнем моря.

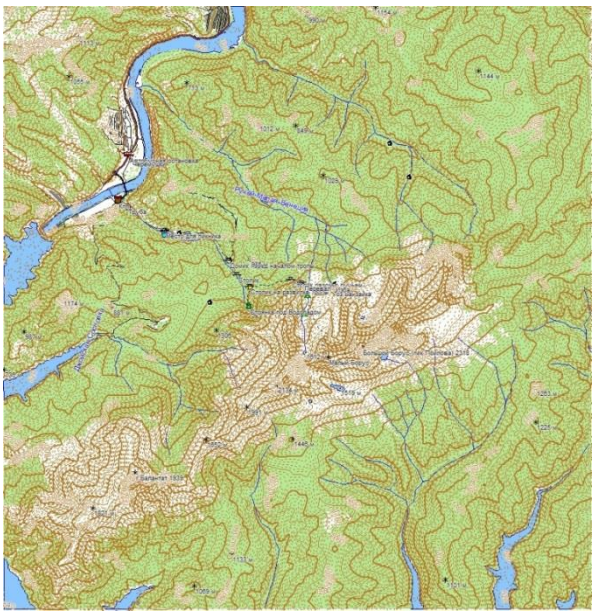

Рисунок 20- Исходная карта горного хребта Борус

Вычисления производились при количестве узлов сетки 1000 по каждому из направлений. Полученная цифровая модель изображена на рис. 21.

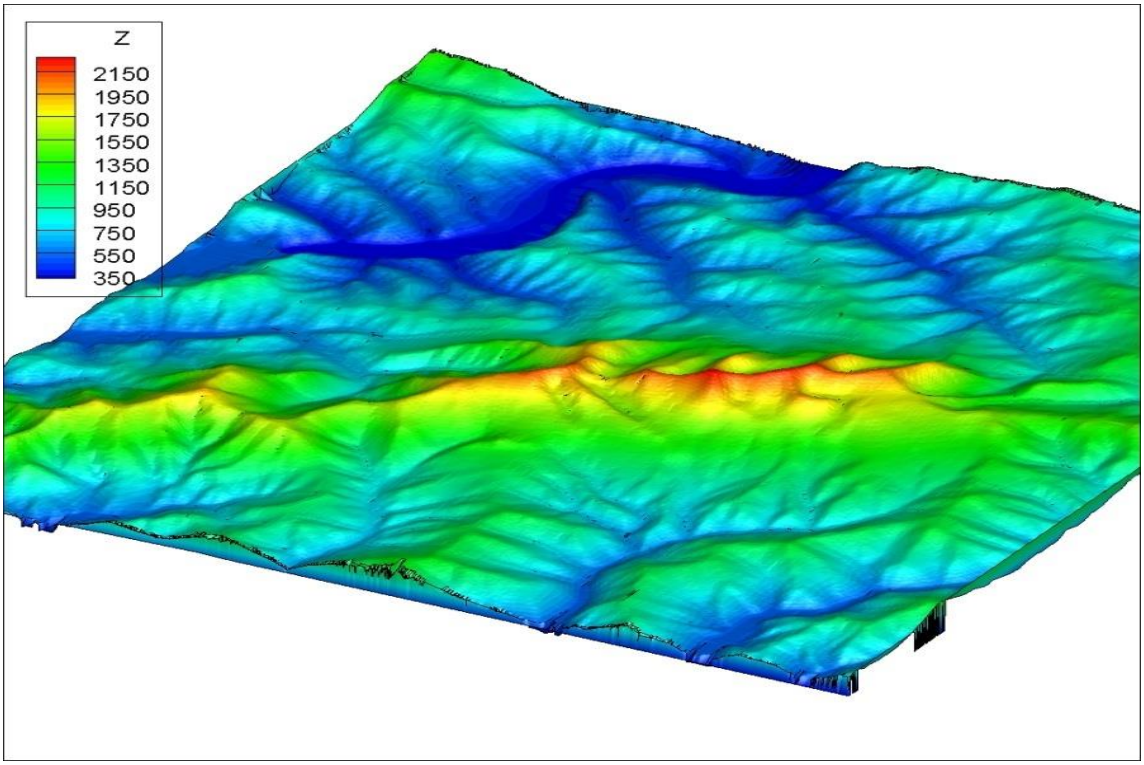

Рисунок 21- Цифровая модель хребта Борус

ЗАКЛЮЧЕНИЕ

В представленной работе получены следующие результаты.

1. Разработан вычислительный алгоритм построения цифровой модели рельефа на основе изолиний, данные о которых извлекаются из векторных карт в польском формате. Данные в этом формате отличаются хорошей структурой, их удобно считывать с помощью программы из файла.
2. Вычислительный алгоритм реализован в виде программного комплекса на языке C++.
3. Вычислительный алгоритм протестирован на серии тестовых задач.

Результаты данной работы были представлены на следующих конференциях и опубликованы в трудах этих конференций.

1. Международная конференция студентов, аспирантов и молодых ученых «Проспект Свободный-2016», секция «Вычислительная математики и математическое моделирование», устный доклад.
2. Международная конференция «IX Сибирский конгресс женщин-математиков», постерный доклад.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Шнитко С.Г. Конспект лекций по предмету «ГИС в геодезии» [Электронный ресурс] - Режим доступа: http://www.psu.by/images/stories/gf/kaf_prikl_geod/gis-v-geodezii-lektsii.pdf

2 Варфоломеев, И.В. Алгоритмы и структуры данных геоинформационных систем: Методические указания /И.В.Варфоломеев, И.Г.Ермакова, А.С.Савельев - Красноярск: КГТУ, 2003. -34 с.

3 Тикунов, В.С. Моделирование в картографии: Книга / В.С. Тикунов — Москва: Изд-во МГУ, 1997.-405 с.

4 Костюк Ю.Л. Предварительная обработка исходных данных для построения цифровой модели рельефа местности / Ю.Л.Костюк, А.Л.Фукс [Электронный ресурс] - Режим доступа: http://rus.neicon.ru:8080/xmlui/bitstream/handle/123456789/2586/110_280-281.pdf?sequence=1

5 Хромых В.В. Цифровые модели рельефа: Учебное пособие / В.В.Хромых, О.В.Хромых - Томск: Изд-во «ТМЛ-Пресс», 2007. - 178 с.

ПРИЛОЖЕНИЕ А

Текст программы

Файл “point.h”.

```
#pragmaonce;
#include<iostream>
usingnamespacestd;

structCPoint
{
	double x, y, z;
	CPoint(double_x=0, double_y=0, double_z=0): x(_x), y(_y),z(_z) {};
	doubledistance(constCPoint&P) const;
	double dx(constCPoint&P) const {returnfabs(x-P.x);}
	doubledy(constCPoint&P) const {returnfabs(y-P.y);}
};

CPoint operator - (constCPoint&C1, constCPoint&C2);
CPoint operator + (constCPoint&C1, constCPoint&C2);
CPoint operator / (constCPoint&C1, constdouble& d);
ostream& operator << (ostream& out, constCPoint&P);
```

Файл “main.cpp”

```
#include<iostream>
#include<list>
#include<vector>
#include<set>
#include<fstream>
#include<string>
#include"point.h"
#defineM_PI	3.14159265358979323846
usingnamespacestd;

boolTestTask = 1; // 1 - тестоваязадача, 0 - карта
```

```
int N=100, M=100; // разбиение исходной области Omega - Квадрат [0, Lx]x[0, Ly]
doublex_min(0), x_max(0), y_min(0), y_max(0),z_min(0),z_max(0);
doubleeps = 0.0001;

typedeflist<pair<list<CPoint>, double>>IsoLines;
typedeflist<CPoint>ListPoints;
typedefvector<vector<double>>Matrix;
typedefvector<CPoint>Points;

voidLoadIsoLineFromFile(char *filename, IsoLines&line);
voidTecPlotV(char *filename, constIsoLines&lines);
voidTecPlotU(char *filename, constMatrix&U);
CPointSearchPointInGrid(constCPoint&P, int&ii, int&jj);
voidApproximateIsoLineInGrid(IsoLines&lines, Matrix&U);
MatrixInit(int n, int m);
voidSegmentDecomposition(constCPoint&V1, constCPoint&V2, ListPoints&points);
void Load(string filename, IsoLines&lines);//Загрузка
voidSearch_min_max(constIsoLines&lines);
voidLineSplineX(Matrix&U);
voidLineSplineY(Matrix&U);


int main()
{
        setlocale(0,"");
        IsoLines lines;
        if (TestTask) LoadIsoLineFromFile("Example.txt", lines);
        else Load("Borus_full.txt", lines);
        cout<<"Изолинииизагружены!\n";

        Search_min_max(lines);
        TecPlotV("Isolines.dat", lines);
        Matrix U=Init(N, M);

        ApproximateIsoLineInGrid(lines, U);
```

```
	LineSplineY(U);
	LineSplineX(U);

	x_max = 1.01*(x_max - x_min)*64000; x_min = -0.01*x_max;
	y_max = 1.01*(y_max - y_min)*111000; y_min = -0.01*y_max;
	cout<<"Загрузка данных для TecPlot...\n";
	TecPlotU("Surface.dat", U);

	system("pause");
}

// Инициализация двумерного массива
MatrixInit(intn, intm)
{
	Matrix M;
	M.resize(n+1);
	for(inti=0;i<n+1;i++)
	{
		M[i].resize(m+1);
		for(int j=0; j<m+1; j++) M[i][j]=z_min;
	}
	return M;
}

void Load(stringfilename, IsoLines&lines)//Загрузка
{
	fstream Loading(filename.c_str());
	string str,str2;
	char symbol;
	double h, x,y, z;
	while(!Loading.eof())
	{
		do
		{
			getline(Loading,str,'\n');
```

```
                }
                while(str!="[POLYLINE]");

                getline(Loading,str,'\n'); // Type
                getline(Loading,str,'='); // Label
                Loading >> h;
                getline(Loading,str,'='); // Data0
                Loading.get();
                lines.push_back(make_pair(list<CPoint>(), h));
                while(1)
                {
                        Loading>>y>>symbol>>x;
                        do
                        {
                                Loading >> symbol;
                        }
                        while (!isdigit(Loading.peek()) && symbol!='[');
                        lines.back().first.push_back(CPoint(x, y));
                        if (symbol=='[' || symbol=='E') break;
                }
                getline(Loading,str,'\n');
                Loading.get();
        }
}

voidLoadIsoLineFromFile(char *filename, IsoLines&lines)
{
        ifstream fin(filename);
        double h, x, y;
        int count;
        while(!fin.eof())
        {
                fin>> h; // высота изолинн
                fin>>count; // количество вершин изолинн
                // вставляем в список новую изолинию
```

```
                lines.push_back(make_pair(list<CPoint>(), h));
                for(inti=0; i<count; i++)
                {
                        fin>> x >> y;
                        lines.back().first.push_back(CPoint(x, y));
                }
        }
}

voidSearch_min_max(constIsoLines&lines)
{
        IsoLines::const_iteratoriter_lines=lines.begin();
        list<CPoint>::const_iterator p=iter_lines->first.begin();
        x_min = x_max = p->x;
        y_min = y_max = p->y;
        z_min = z_max = iter_lines->second;
        double h;

        for(iter_lines=lines.begin(); iter_lines!=lines.end(); ++iter_lines)
        {
                // Циклпоточкамизолиней
                h = iter_lines->second;
                if (h <z_min) z_min = h;
                if (h >z_max) z_max = h;
                for(p=iter_lines->first.begin(); p!=iter_lines->first.end(); ++p)
                {
                        if (p->x <x_min) x_min = p->x;
                        if (p->x >x_max) x_max = p->x;
                        if (p->y <y_min) y_min = p->y;
                        if (p->y >y_max) y_max = p->y;
                }
        }
}

voidTecPlotV(char *filename, constIsoLines&lines)
```

```
{
	ofstream out(filename);
	out <<"VARIABLES = \"X\" \"Y\" \"Z\""<<endl;
	// Циклпоизолиниям
	IsoLines::const_iteratoriter_lines;
	int line=1;
	for(iter_lines=lines.begin(); iter_lines!=lines.end(); ++iter_lines)
	{
		out <<"ZONE T=\"Polygon_"<< line++ <<"\""<<endl;
		out <<"I="<<iter_lines->first.size() <<", J="<< 1 <<", K=1
ZONETYPE=Ordered"<<endl;
		out<<"DATAPACKING=POINT"<<endl;
		out<<"DT=(DOUBLE DOUBLE )"<<endl;
		out.setf(ios::scientific, ios::floatfield);

		// Циклпоточкамизолиней
		list<CPoint>::const_iterator p;
		for(p=iter_lines->first.begin(); p!=iter_lines->first.end(); ++p)
		{
			out<< *p <<endl;
		}
		out.setf(ios::fixed, ios::floatfield);
	}
	out.close();
}

voidTecPlotU(char *filename, constMatrix&U)
{
	ofstream out(filename);
	// Построениесетки
	doublehx=1.0*(x_max-x_min)/N, hy=1.0*(y_max-y_min)/M, x, y;
	out <<"VARIABLES = \"X\" \"Y\" \"Z\""<<endl;
	out <<"ZONE T=\"Surface\""<<endl;
	out <<"I="<< M+1 <<", J="<< N+1 <<", K=1 ZONETYPE=Ordered"<<endl;
	out<<"DATAPACKING=POINT"<<endl;
```

```
	out<<"DT=(DOUBLE DOUBLE )"<<endl;
	out.setf(ios::scientific, ios::floatfield);
	for (inti = 0; i<= N; i++)
	{
		x = x_min+i*hx;
		for (int j = 0; j <= M; j++)
		{
			y = y_min+j*hy;
			out<< x <<" "<< y <<" "<<U[i][j] <<endl;
		}
	}
	out.close();
}

voidApproximateIsoLineInGrid(IsoLines&lines, Matrix&U)
{
	int i, j;
	// Цикл по изолиниям
	IsoLines::iteratoriter_lines;
	ListPoints points;
	ListPoints::iterator it;
	for(iter_lines=lines.begin(); iter_lines!=lines.end(); ++iter_lines)
	{
		// Циклпоточкамизолиней
		list<CPoint>::iterator p=iter_lines->first.begin()++;
		list<CPoint>::iterator q = ++p;
		for(p=iter_lines->first.begin(); p!=--iter_lines->first.end(); )
		{
			points.push_back(*p);
			points.push_back(*q);
			SegmentDecomposition(*p, *q, points);
			// Отобразить множество точек на сетку

			for(it=points.begin(); it!=points.end(); it++)
			{
```

```
                                SearchPointInGrid(*it, i, j);
                                U[i][j]=iter_lines->second;
                        }
                        points.clear();
                        // Передвижение итераторов
                        p=q;
                        if (q!=iter_lines->first.end()) ++q;
                }
        }
}

CPointSearchPointInGrid(constCPoint&P, int&ii, int&jj)
{
        // i, j - номер ячейки, куда попадает точка P
        doublehx=1.0*(x_max-x_min)/N, hy=1.0*(y_max-y_min)/M;
        double r1, r2, r3, r4, rr1, rr2;
        inti=(P.x-x_min)/hx, j=(P.y-y_min)/hy;
        if (i>=N) i=N-1; if (i<0) i=0;
        if (j>=M) j=M-1; if (j<0) j=0;
        r1 = P.distance(CPoint(x_min+i*hx, y_min+j*hy));
        r2 = P.distance(CPoint(x_min+(i+1)*hx, y_min+j*hy));
        r3 = P.distance(CPoint(x_min+i*hx, y_min+(j+1)*hy));
        r4 = P.distance(CPoint(x_min+(i+1)*hx, y_min+(j+1)*hy));

        if (r1<r2)      {rr1=r1; ii=i;}
        else {rr1=r2; ii=i+1;}
        if (r3<r4)      {rr2=r3; ii=i;}
        else {rr2=r4; ii=i+1;}

        if (rr1<rr2) jj=j; elsejj=j+1;
        returnCPoint(ii*hx, jj*hy);
}

voidSegmentDecomposition(constCPoint&V1, constCPoint&V2, ListPoints&points)
{
```

```
        doublehx=1.0*(x_max-x_min)/N, hy=1.0*(y_max-y_min)/M;
        if (V1.dx(V2)<=hx&&V1.dy(V2)<=hy) return;
        CPoint Center=(V1+V2)/2;
        points.push_back(Center);
        SegmentDecomposition(V1, Center, points);
        SegmentDecomposition(V2, Center, points);
}

voidLineSplineY(Matrix&U)
{
        doublehx=1.0*(x_max-x_min)/N, hy=1.0*(y_max-y_min)/M, x, y;
        for(inti=0; i<N; i++)
                {
                        intjp=0; doublezp = U[i][jp];
                        for(int j=jp+1; j<M; j++)
                        {
                                if (fabs(U[i][j]-z_min) <eps) continue;
                                //if (fabs(map[i].z[j]-zp)<eps) continue;
                                doublehz = (U[i][j] - zp) / (j - jp);
                                for(intjj=jp; jj<=j; jj++) U[i][jj] = zp + (jj-jp)*hz;
                                zp = U[i][j]; jp = j;
                        }
                }
}

voidLineSplineX(Matrix&U)
{
        for(int j=0; j<M; j++)
        {
                intip=0; doublezp=U[0][j];
                for(inti=ip+1; i< N; i++)
                {
                        if (fabs(U[i][j] - zp)<eps) continue;
                        doublehz = (U[i][j] - zp) / (i - ip);
                        if (i-ip>1 &&hz>0) {
```

```
                        for(int ii=ip; ii<=i; ii++) U[ii][j] = zp + (ii-ip)*hz;
                    }
                    zp = U[i][j]; ip = i;
                }
                ip=N-1;
                for(inti=ip; i>=0; i--)
                {
                    if (fabs(U[i][j] - zp) <eps) continue;
                    doublehz = (U[i][j] - zp) / (ip - i);
                    if (ip-i>1 &&hz>0) {
                        for(int ii=ip; ii>=i; ii--) U[ii][j] = zp + (ip-ii)*hz;
                    }
                    zp = U[i][j]; ip = i;
                }
        }
}
```

Файл “point.cpp”

```
#include"point.h"

doubleCPoint::distance(constCPoint&P) const
{
        returnsqrt((x-P.x)*(x-P.x) + (y - P.y)*(y - P.y));
}

CPoint operator - (constCPoint&C1, constCPoint&C2)
{
        CPointR(C1.x-C2.x, C1.y-C2.y);
        return R;
}

CPoint operator + (constCPoint&C1, constCPoint&C2)
{
        CPointR(C1.x+C2.x, C1.y+C2.y);
        return R;
```

```
}

CPoint operator / (constCPoint&C1, constdouble&d)
{
	CPointR(C1.x/d, C1.y/d);
	return R;
}


ostream& operator << (ostream&out, constCPoint&P)
{
	out<<P.x<<" "<<P.y<<" "<<P.z;
	return out;
}
```

ПРИЛОЖЕНИЕ Б

Пример польского формата

[POLYLINE]

Type=0x20

Label=340

Data=(52.84780,91.41503),(52.84752,91.41494),(52.84765,91.41484),(52.84778,91.41487),(52.84786,91.41494),(52.84786,91.41501),(52.84780,91.41503)

[END]

[POLYLINE]

Type=0x20

Label=340

Data=(52.84862,91.42260),(52.84872,91.42254),(52.84872,91.42240),(52.84862,91.42221),(52.84846,91.42228),(52.84829,91.42243),(52.84862,91.42260)

[END]