

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования

"Ярославский государственный университет им. П.Г. Демидова"

Кафедра теории функций и функционального анализа

«Допустить к защите»

Зав.кафедрой,

к.ф.-м.н., доцент

_____ М.В.Невский

«__» _____ 20__ г.

Дипломная работа

**Вычисление геометрических характеристик n -мерного
симплекса**

Научный руководитель

к.ф.-м.н., доцент

_____ М.В.Невский

«__» _____ 20__ г.

Студентка группы ПМИ-51СО

_____ А.А.Князева

«__» _____ 20__ г.

Ярославль 2015 г.

Содержание

1	Введение. Постановка задачи	2
2	Геометрические характеристики n-мерного симплекса	3
2.1	Некоторые свойства базисных многочленов лагранжа	3
2.2	Вычисление максимального в симплексе отрезка данного направления	5
2.3	Об одной задаче для симплекса и куба в \mathbb{R}^n	7
3	Описание программы	9
4	Результаты счета	12
4.1	Случай $n = 2$	12
4.2	Случай $n = 3$	19
5	Заключение	25
6	Библиография	26
7	Приложение	27

1 Введение. Постановка задачи

Данная работа посвящена решению ряда задач, связанных с невырожденным симплексом $S \subset \mathbb{R}^n$. Основными являются задачи вычисления осевых диаметров n -мерного симплекса и вычисление максимального отрезка заданного направления.

Целью работы является изучение алгоритма решения поставленных задач и реализация его на ЭВМ. Программа должна подсчитывать осевые диаметры, коэффициенты λ_j базисных многочленов Лагранжа, норму интерполяционного проектора, величину α по заданным вершинам невырожденного симплекса S , максимальный отрезок заданного направления (направление задаётся вектором размерности n) и некоторых других характеристик.

Теоретическая часть содержит необходимые геометрические характеристики. Представленные в этой части работы формулы используются для нахождения осевых диаметров n -мерного симплекса, базисных многочленов Лагранжа и других величин, связанных с поставленными задачами. Во второй части описание программы. Затем рассмотрены некоторые примеры, подтверждающие точность полученных в завершении работы программы результатов. Обобщает все это заключение, в котором подводится итог всей проделанной работе. Также имеет место приложение, в котором представлен код реализованной мною программы.

2 Геометрические характеристики n -мерного симплекса

2.1 Некоторые свойства базисных многочленов лагранжа

Пусть $n \in \mathbb{N}$. Элемент $x \in \mathbb{R}^n$ будем записывать в виде $x = (x_1, \dots, x_n)$. Положим $Q_n := [0, 1]^n$. Через $C(Q_n)$ обозначим пространство непрерывных функций $F : C(Q_n) \rightarrow \mathbb{R}$ с нормой $\|f\|_{C(Q_n)} := \max_{x \in Q_n} |f(x)|$, а через $\Pi_1(\mathbb{R}_n)$ – совокупность многочленов от n переменных степени ≤ 1 .

Рассмотрим невырожденный симплекс $S \subset \mathbb{R}^n$, то есть такой, что $\text{vol}(S) \neq 0$. Обозначим вершины S через $x^{(j)} = (x_1^{(j)}, \dots, x_n^{(j)})$, $j = 1, \dots, n+1$. Из координат вершин $x^{(j)}$ составим матрицу

$$\mathbf{A} := \begin{pmatrix} x_1^{(1)} & \dots & x_n^{(1)} & 1 \\ x_1^{(2)} & \dots & x_n^{(2)} & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x_1^{(n+1)} & \dots & x_n^{(n+1)} & 1 \end{pmatrix}.$$

Пусть $\Delta := \det(\mathbf{A})$, а определитель $\Delta_j(x)$ получается из Δ заменой j -й строки на строку $(x_1, \dots, x_n, 1)$. Многочлены $\lambda_j(x) = 0$ задают $(n-1)$ -мерные грани S . Имеет место представление

$$S = \{x \in \mathbb{R}^n : \lambda_j(x) \geq 0, j = 1, \dots, n+1\}. \quad (2.1)$$

В дальнейшем используется запись

$$\lambda_j(x) = l_{1j}x_1 + \dots + l_{nj}x_n + l_{n+1,j}. \quad (2.2)$$

Обозначим через $d_i(S)$ максимальную длину отрезка, содержащегося в S и параллельного оси x_i . Величину $d_i(S)$ будем называть i -м осевым диаметром S . В [4] доказано, что для любого $i = 1, \dots, n$ справедливо равенство

$$\frac{1}{d_i(S)} = \frac{1}{2} \sum_{j=1}^{n+1} |l_{ij}|. \quad (2.3)$$

В симплексе S существует ровно один отрезок длины $d_i(S)$, параллель-

ный оси x_i . Концы $y_+^{(i)}$ и $y_-^{(i)}$ этого отрезка суть

$$y_+^{(i)} = \sum_{j=1}^{n+1} s_{ij} x^{(j)}, s_{ij} := \frac{|l_{ij}| + l_{ij}}{\sum_{k=1}^{n+1} |l_{ik}|}, \quad (2.4)$$

$$y_-^{(i)} = \sum_{j=1}^{n+1} t_{ij} x^{(j)}, t_{ij} := \frac{|l_{ij}| - l_{ij}}{\sum_{k=1}^{n+1} |l_{ik}|}. \quad (2.5)$$

Обозначим за $\Sigma_i(S)$ $(n-1)$ -меру проекции S на гиперплоскость $x_i = 0$. Имеет место равенство

$$vol(S) = \frac{d_i(S) \cdot \Sigma_i(S)}{n}, \quad (2.6)$$

Если $Q_n \subset S$, то справедливо неравенство

$$\sum_{i=1}^n \frac{1}{d_i(S)} \leq 1. \quad (2.7)$$

Из (2.7) следует, что в случае $Q_n \subset S$ для некоторого $i = 1, \dots, n$ симплекс S содержит отрезок длины n , параллельный оси x_i .

Если $S \subset Q_n$, то многочлены λ_j могут применяться для вычисления величины $\xi(S) := \min\{\sigma \geq 1 : Q_n \subset \sigma S\}$, где σS есть результат гомотетии S относительно центра тяжести с коэффициентом σ . В [3] доказано, что

$$\xi(S) = (n+1) \max_{1 \leq j \leq n+1} \max_{x \in ver(Q_n)} (-\lambda_j(x)) + 1. \quad (2.8)$$

Здесь и далее $ver(Q_n)$ есть совокупность вершин Q_n .

Пусть $P : C(Q_n) \rightarrow \Pi_1(\mathbb{R}^n)$ – интерполяционный проектор, узлы которого совпадают с вершинами $S \subset Q_n$. Этот проектор определяется равенствами $Pf(x^{(j)}) = f_j := f(x^{(j)})$. Справедлив следующий аналог интерполяционной формулы Лагранжа:

$$Pf(x) = p(x) := \sum_{j=1}^{n+1} f_j \lambda_j(x), \quad (2.9)$$

в связи с чем многочлены λ_j мы называем базисными многочленами Лагранжа. Из (2.9) получается, что норма P как оператора из $C(Q_n)$ в

$C(Q_n)$ может быть найдена по формулам

$$\|P\| = \max_{x \in \text{ver}(Q_n)} \sum_{j=1}^{n+1} |\lambda_j(x)| = \max_{f_j=\pm 1} \max_{x \in \text{ver}(Q_n)} |p(x)|. \quad (2.10)$$

Для $n = 2$ равенство (2.10) обосновано в [1]; общий случай в [2]. Как установлено в [3, 4], в случае $S \subset Q_n$ введённые нами величины связаны двойным неравенством

$$\sum_{i=1}^n \frac{1}{d_i(S)} \leq \xi(S) \leq \frac{n+1}{2} (\|P\| - 1) + 1. \quad (2.11)$$

Пусть C - выпуклое тело в \mathbb{R}^n , т.е. компактное выпуклое подмножество \mathbb{R}^n с непустой внутренностью. Через σC обозначим результат гомотетии C относительно центра тяжести с коэффициентом σ . Символом $d_i(C)$ обозначим i -й осевой диаметр C . Через Q_n обозначим n -мерный единичный куб $[0, 1]^n$. Под транслятором будем понимать результат параллельного переноса.

Для выпуклых тел $C_1, C_2 \subset \mathbb{R}^n$ обозначим через $\alpha(C_1; C_2)$ минимальное $\sigma > 0$, для которого C_1 принадлежит транслятору σC_2 . В [6] доказано, что для любого выпуклого тела C справедливо неравенство

$$\alpha(Q_n; C) \leq \sum_{i=1}^n \frac{1}{d_i(C)}. \quad (2.12)$$

Если же C представляет собой невырожденный симплекс S , то это соотношение обращается в равенство, т.е. имеет место

$$\alpha(Q_n; S) = \sum_{i=1}^n \frac{1}{d_i(S)}. \quad (2.13)$$

(см. [7, теорема 4]).

Из (2.13) и (2.3) получается, что

$$\alpha(Q_n; S) = \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n |l_{ij}|. \quad (2.14)$$

2.2 Вычисление максимального в симплексе отрезка данного направления

Пусть v - ненулевой n -мерный вектор. Обозначим через $d^v(S)$ максимальную длину отрезка, принадлежащего S и параллельного v . Далее

представлены формулы для вычисления величины $d^v(S)$ и концов максимального отрезка по координатам v и вершинам S . В случае, когда v коллинеарен i -й координатной оси, положим $d_i(S) := d^v(S)$, где $d_i(S)$ – i -ый осевой диаметр S .

Обозначим через v_1, \dots, v_n координаты данного нам вектора v . Введем в рассмотрение числа $(1 \leq j \leq n+1)$

$$m_j := \sum_{k=1}^n l_{kj} v_k, \quad (2.15)$$

$$\alpha_j := \frac{|m_j| - m_j}{\sum_{k=1}^{n+1} |m_k|}, \quad \beta_j := \frac{|m_j| + m_j}{\sum_{k=1}^{n+1} |m_k|}. \quad (2.16)$$

Через $\|\cdot\|$ обозначим евклидову норму в \mathbb{R}^n . Основной результат, используемый в текущем разделе моей выпускной квалификационной работе, состоит в следующем.

Теорема. *Величина $d^v(S)$ удовлетворяет равенству*

$$d^v(S) = \frac{2\|v\|}{\sum_{j=1}^{n+1} |m_j|}. \quad (2.17)$$

Концы единственного отрезка максимальной длины, принадлежащего S и параллельного v , есть точки

$$a = \sum_{j=1}^{n+1} \alpha_j x^{(j)}, \quad b = \sum_{j=1}^{n+1} \beta_j x^{(j)}. \quad (2.18)$$

Для доказательства данной теоремы необходимо использовать следующие вспомогательные предположения.

Лемма 1. *Пусть I – отрезок, параллельный v и расположенный в S таким образом, что каждый $(n-1)$ -мерная грань S содержит хотя бы один из его концов. Тогда длина I совпадает с правой частью (2.17).*

Лемма 2. *В S существует единственный отрезок, параллельный вектору v и расположенный таким образом, что каждая $(n-1)$ -мерная грань S содержит хотя бы один из его концов. Этот отрезок является единственным отрезком из S максимальной длины, параллельным v .*

С привлечением результатов, описанных в первом разделе моей выпускной квалификационной работы можно получить следующие следствия. Пусть $\Sigma(S; v)$ есть $(n - 1)$ -мерная мера проекции симплекса S на гиперплоскость, ортогональную вектору v .

Следствие 1. *Имеют места равенства*

$$\Sigma(S; v) = \frac{n * \text{vol}(S)}{d^v(S)} = \frac{|\det(A)|}{2(n-1)! \|v\|} \sum_{j=1}^{n+1} |m_j|. \quad (2.19)$$

Через σS обозначим образ S при гомотетии с коэффициентом σ и центром гомотетии в центре тяжести S . Пусть V – невырожденный параллелепипед в \mathbb{R}^n , рёбра которого задаются линейно независимыми векторами $v^{(1)}, \dots, v^{(n)}$. Через $\alpha(V; S)$ обозначим минимальное $\sigma > 0$ такое, что V одержится в трансляте симплекса σS . Вычислим величину

$$M := \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^{n+1} \left| \sum_{k=1}^n l_{kj} v_k^{(i)} \right|. \quad (2.20)$$

Следствие 2. *Имеют места равенства*

$$\alpha(V; S) = \sum_{i=1}^n \frac{\|v^{(i)}\|}{d^{v^{(i)}}(S)} = M. \quad (2.21)$$

Следствие 3. *Неравенство $M \leq 1$ эквивалентно тому, что V содержится в трансляте симплекса S . Равенство $M = 1$ эквивалентно тому, что некоторый транслят S' симплекса S описан вокруг V , т.е. $V \subset S'$ и каждая $(n - 1)$ -мерная грань S' содержит вершину V .*

2.3 Об одной задаче для симплекса и куба в \mathbb{R}^n

В данном разделе рассматривается задача о вычислении для симплекса S такой точки $x \in \mathbb{R}^n$, для которой с минимально возможным коэффициентом $\sigma > 0$ справедливо включение $Q_n \subset S_{x, \sigma}$. Задача имеет решение, и причём единственное, в случае $\alpha(S) \neq 1$; при этом минимальное σ как раз и равно $\alpha(S)$. Далее приведены формулы, в которых центр x минимальной положительной гомотетии вычисляется через вершины S и числа l_{ij} – коэффициенты многочленов λ_j (см (2.2)).

Пусть S – невырожденный симплекс в \mathbb{R}^n . Из определения $\alpha(S)$ (см. 2.14) легко следует, что некоторый транслянт симплекса описан вокруг

Q_n . Поэтому $\alpha(S) = 1$ тогда и только тогда, когда существует транслят S описанный вокруг Q_n .

Теорема 1. Если $\sigma = \sum_{i=1}^n 1/d_i(S) \neq 1$, то существует единственная точка $x = (x_1, \dots, x_n)$ такая, что $Q_n \subset S_{x,\sigma}$. Имеют место равенства

$$x_k = \frac{1}{2(\sigma - 1)} \left[\sum_{j=1}^{n+1} \left(\sum_{i=1}^n |l_{ij}| \right) x_k^{(j)-1} \right], k = 1, \dots, n. \quad (2.22)$$

Если $0 < \sigma < \sum_{i=1}^n 1/d_i(S)$, то для любой $x \in \mathbb{R}^n$ верно $Q_n \not\subset S_{x,\sigma}$.

Приведём формулы для вычисления x , в которых используются только вершины S и числа l_{ij} .

Теорема 2. Для невырожденного симплекса $S \subset \mathbb{R}^n$ условие $\alpha(S) \neq 1$ эквивалентно

$$\sum_{i=1}^n \sum_{j=1}^{n+1} |l_{ij}| \neq 2. \quad (2.23)$$

Пусть выполнено (2.23) и $\sigma := \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^{n+1} |l_{ij}|$. Тогда единственная точка x , для которой верно включение $Q_n \subset S_{x,\sigma}$, может быть вычислена по равенствам

$$x_k = \frac{\sum_{j=1}^{n+1} (\sum_{i=1}^n |l_{ij}|) x_k^{(j)} - 1}{\sum_{i=1}^n \sum_{j=1}^{n+1} |l_{ij}| - 2}, k = 1, \dots, n. \quad (2.24)$$

Рассмотрим ситуацию, когда симплекс содержится в Q_n . $d_i(S) \leq 1$, поэтому $\alpha(S) \geq n$ (равенство имеет место тогда и только тогда, когда каждый осевой диаметр равен 1). Далее идёт специальный вариант теоремы 1 для этой ситуации.

Теорема 3. Пусть $S \subset Q_n$ и $d_1(S) = \dots = d_n(S) = 1$. Существует единственная точка $x \in S$ такая, что $Q_n \subset S_{x,n}$. При $n > 1$ имеют место равенства

$$x_k = \frac{1}{2(n-1)} \left[\sum_{j=1}^{n+1} \left(\sum_{i=1}^n |l_{ij}| \right) x_k^{(j)-1} \right], k = 1, \dots, n. \quad (2.25)$$

Если $0 < \sigma < n$, то для любой $x \in \mathbb{R}^n$ верно $Q_n \not\subset S_{x,n}$.

3 Описание программы

Основной задачей моей работы стало написание программы, реализующей алгоритм нахождения различных геометрических характеристик симплекса по заданным вершинам.

Программа написана на языке JavaScript и представляет собой веб-приложение. Серверная часть реализована на NodeJs. Для сборки клиентской части использовался GULP, пакетный менеджер на клиентской части - Bower. Программа написана с использованием популярного фрэймворка AngularJS. Так же применялся front-end фрэймворк Bootstrap. На странице приложения вводятся координаты вершин симплекса, вектор, параллелограмм (представленный n векторами). После завершения подсчета на экран выводятся осевые диаметры (длины и координаты концов), коэффициенты базисных многочленов Лагранжа, норма проектора $\|P\|$, минимальный транслятор $\alpha(Q_n; S)$, максимальный вписанный отрезок, параллельный заданному вектору (длина и координаты концов), $\Sigma(S; v)$, $\alpha(V; S)$ для заданного параллелограмма. Все исходники хранятся в репозитории на GitHub.

Программа включает в себя следующие файлы:

1. package.json, метаданные серверной части проекта;
2. bower.json, метаданные клиентской части проекта;
3. server.js, серверная часть проекта;
4. gulpfile.js, файл, необходимый для сборки проекта и сжатия файлов;
5. index.html, файл с версткой;

6. simplex.js, контроллер;

7. mymath.js, файл с реализацией необходимых вычислительных функций;

Интерфейс максимально прост и выглядит следующим образом:

1. ввод размерности и вершин симплекса;

Simplex dimension:

$x^{(1)}$	<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>
$x^{(2)}$	<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>
$x^{(3)}$	<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>
$x^{(4)}$	<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>

2. ввод координат вектора;

v_1	v_2	v_3
<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>

3. ввод векторов, образующих параллелограмм;

$v^{(1)}$	<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>
$v^{(2)}$	<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>
$v^{(3)}$	<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>

GO!

4 Результаты счета

В этой части работы будут приведены несколько примеров, показывающих результаты вручную посчитанного алгоритма в сравнении с результатами, посчитанными программой.

4.1 Случай $n = 2$

Пусть $n = 2$, S - треугольник с вершинами $x^{(1)} = (1, 0)$, $x^{(2)} = (1/2, 1)$, $x^{(3)} = (0, 1/4)$. Тогда

$$\mathbf{A} := \begin{pmatrix} 1 & 0 & 1 \\ \frac{1}{2} & 1 & 1 \\ 0 & \frac{1}{4} & 1 \end{pmatrix}, \mathbf{A}^{-1} := \begin{pmatrix} \frac{6}{7} & \frac{2}{7} & -\frac{8}{7} \\ -\frac{4}{7} & \frac{8}{7} & -\frac{4}{7} \\ \frac{1}{7} & -\frac{2}{7} & \frac{8}{7} \end{pmatrix}.$$

Коэффициенты базисных многочленов Лагранжа составляют столбцы матрицы \mathbf{A}^{-1} , поэтому

$$\lambda_1(x) = \frac{6}{7}x_1 - \frac{4}{7}x_2 + \frac{1}{7}, \lambda_2(x) = \frac{2}{7}x_1 + \frac{8}{7}x_2 - \frac{2}{7}, \lambda_3 = -\frac{8}{7}x_1 - \frac{4}{7}x_2 + \frac{8}{7}. \quad (4.1)$$

Результат программы:

```

1  var gulp = require('gulp'),
2      uglify = require('gulp-uglify'),
3      gulpFilter = require('gulp-filter'),
4      minHtml = require('gulp-minify-html'),
5      minCss = require('gulp-minify-css'),
6      bower = require('gulp-bower'),
7      dirName = './static/assets';
8
9  //фильтры для сжатия файлов bower
10 var filterJs = gulpFilter(['**/*.js', '!**/*.min.js']),
11     filterCss = gulpFilter(['**/*.css']);
12
13 //разбираемся со всем, что понаставил bower
14 gulp.task('bower', function() {...});
25 //жмем наши css
26 gulp.task('css', function() {...});
31 //наши js
32 gulp.task('js', function() {
33     gulp.src('./client/js/*.js')
34         .pipe(uglify())
35         .pipe(gulp.dest(dirName));
36 });
37 //и html напоследок
38 gulp.task('html', function() {...});
43 //отслеживаем изменения в наших файлах
44 gulp.task('watch', function() {...});
49 //запускаем всё это дело
50 gulp.task('default', ['bower', 'css', 'js', 'html', 'watch']);
51

```

В соответствии с (2.1) симплекс S (с границей) задаётся системой линейных неравенств

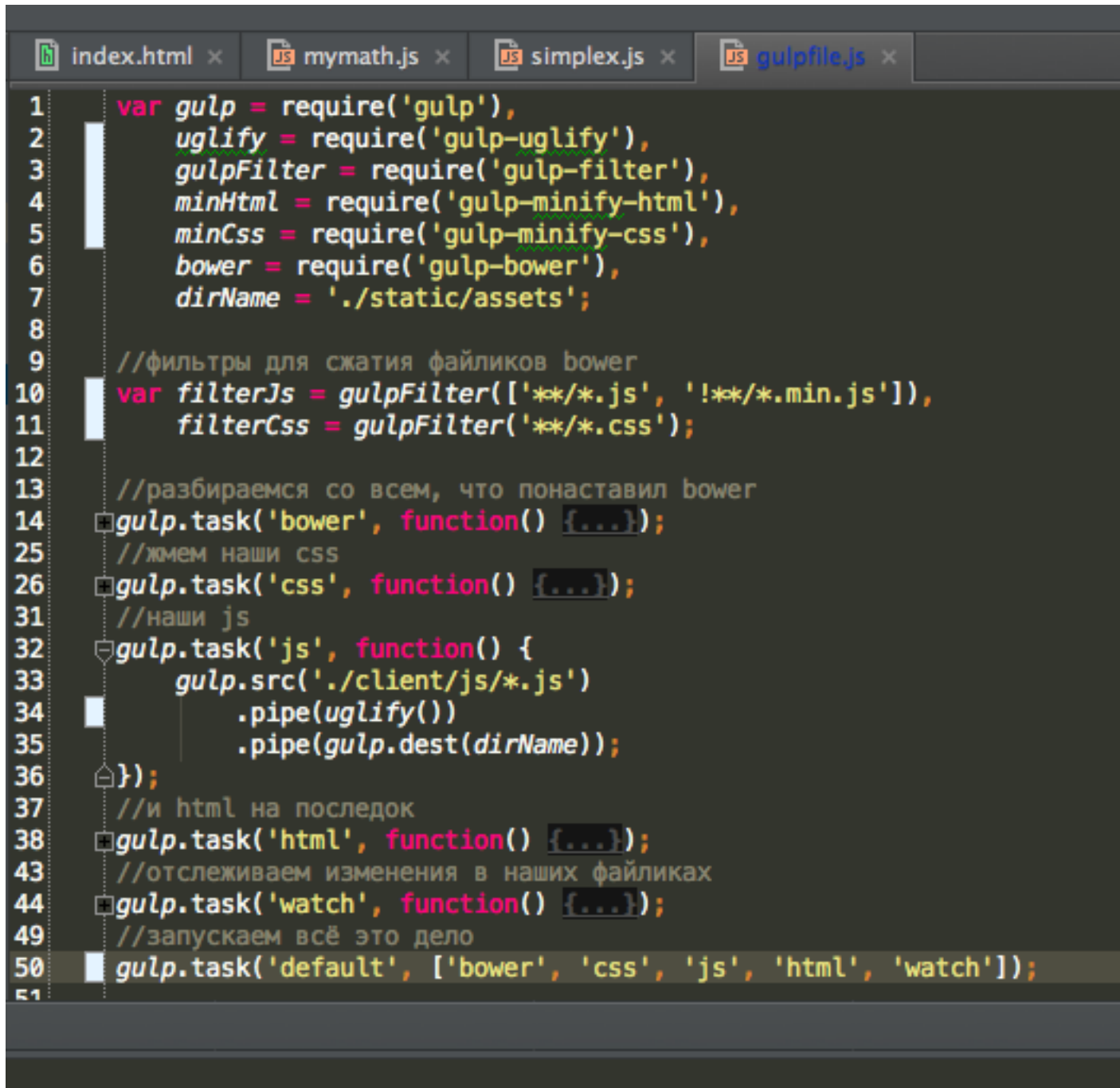
$$\frac{6}{7}x_1 - \frac{4}{7}x_2 \geq -\frac{1}{7}, \frac{2}{7}x_1 + \frac{8}{7}x_2 \geq \frac{2}{7}, \frac{8}{7}x_1 - \frac{4}{7}x_2 \geq -\frac{8}{7}.$$

Вычисления по формуле (2.3) с учётом (2.2), (4.1) дают:

$$\frac{1}{d_1(S)} = \frac{1}{2} \left(\frac{6}{7} + \frac{2}{7} + \frac{8}{7} \right) = \frac{8}{7}, \frac{1}{d_2(S)} = \frac{1}{2} \left(\frac{4}{7} + \frac{8}{7} + \frac{4}{7} \right) = \frac{8}{7},$$

т.е. $d_1(S) = d_2(S) = 7/8$. Заметим, что совпадение $d_1(S)$ и $d_2(S)$ сразу следует из равенства длин проекций S на координатные оси – достаточно привлечь (2.6).

Результат выполнения программы:



```

1  var gulp = require('gulp'),
2      uglify = require('gulp-uglify'),
3      gulpFilter = require('gulp-filter'),
4      minHtml = require('gulp-minify-html'),
5      minCss = require('gulp-minify-css'),
6      bower = require('gulp-bower'),
7      dirName = './static/assets';
8
9  //фильтры для сжатия файлов bower
10 var filterJs = gulpFilter(['**/*.js', '!**/*.min.js']),
11     filterCss = gulpFilter(['**/*.css']);
12
13 //разбираемся со всем, что понаставил bower
14 gulp.task('bower', function() {...});
25 //жмем наши css
26 gulp.task('css', function() {...});
31 //наши js
32 gulp.task('js', function() {
33     gulp.src('./client/js/*.js')
34         .pipe(uglify())
35         .pipe(gulp.dest(dirName));
36 });
37 //и html напоследок
38 gulp.task('html', function() {...});
43 //отслеживаем изменения в наших файлах
44 gulp.task('watch', function() {...});
49 //запускаем всё это дело
50 gulp.task('default', ['bower', 'css', 'js', 'html', 'watch']);
51

```

Найдём координаты концов максимальных отрезков рассматриваемого вида. При $i = 1$ формулы (2.4) - (2.5) приводят к следующим

результатам:

$$s_{11} = \frac{6/7 + 6/7}{16/7} = 1, s_{12} = \frac{2/7 + 2/7}{16/7} = \frac{1}{4}, s_{13} = \frac{8/7 - 8/7}{16/7} = 0,$$

$$y_+^{(1)} = s_{11} x^{(1)} + s_{12} x^{(2)} + s_{13} x^{(3)} = \left(\frac{7}{8}, \frac{1}{4} \right);$$

$$t_{11} = \frac{6/7 - 6/7}{16/7} = 0, t_{12} = \frac{2/7 + 2/7}{16/7} = 0, t_{13} = \frac{8/7 + 8/7}{16/7} = 1,$$

$$y_-^{(1)} = t_{11} x^{(1)} + t_{12} x^{(2)} + t_{13} x^{(3)} = \left(0, \frac{1}{4} \right).$$

Концы максимального в S отрезка, параллельного оси x_1 есть точки $y_+^{(1)} = (7/8, 1/4), y_-^{(1)} = (0, 1/4)$. При $i = 2$ аналогично получаем:

$$s_{21} = 0, s_{22} = 1, s_{23} = 0,$$

$$y_+^{(2)} = s_{21} x^{(1)} + s_{22} x^{(2)} + s_{23} x^{(3)} = x^{(2)} = \left(\frac{1}{2}, 1 \right);$$

$$t_{21} = \frac{1}{2}, t_{22} = 0, t_{23} = \frac{1}{2},$$

$$y_-^{(2)} = t_{21} x^{(1)} + t_{22} x^{(2)} + t_{23} x^{(3)} = \frac{1}{2} x^{(1)} + \frac{1}{2} x^{(3)} = \left(\frac{1}{2}, \frac{1}{8} \right).$$

Поэтому максимальным в S отрезком, параллельным оси x_2 , является отрезок с концами $y_+^{(2)} = (1/2, 1), y_-^{(2)} = (1/2, 1/8)$.

Теперь найдем $\xi(S)$. Очевидно,

$$\max_{x \in \text{ver}(Q_2)} (-\lambda_1(x)) = \frac{3}{7}, \max_{x \in \text{ver}(Q_2)} (-\lambda_2(x)) = \frac{2}{7}, \max_{x \in \text{ver}(Q_2)} (-\lambda_3(x)) = \frac{4}{7}.$$

Формула (2.8) даёт $\xi(S) = 3 \cdot (4/7) + 1 = 19/7$.

Найдём норму интерполяционного проектора $P : C(Q_2) \rightarrow \Pi_1(\mathbb{R}^2)$, узлы которого совпадают с вершинами S . Интерполяционная формула Лагранжа (2.9) в данном случае имеет вид

$$p(x) = Pf(x) = f_1 \lambda_1(x) + f_2 \lambda_2(x) + f_3 \lambda_3(x).$$

Подставим сюда выражение для базисных многочленов λ_j и найдем значения p в вершинах Q_2 :

$$p(0, 0) = \frac{1}{7} f_1 - \frac{2}{7} f_2 + \frac{8}{7} f_3, p(1, 0) = f_1,$$

$$p(0,1) = -\frac{3}{7}f_1 + \frac{6}{7}f_2 + \frac{4}{7}f_3, p(1,1) = \frac{3}{7}f_1 + \frac{8}{7}f_2 - \frac{4}{7}f_3,$$

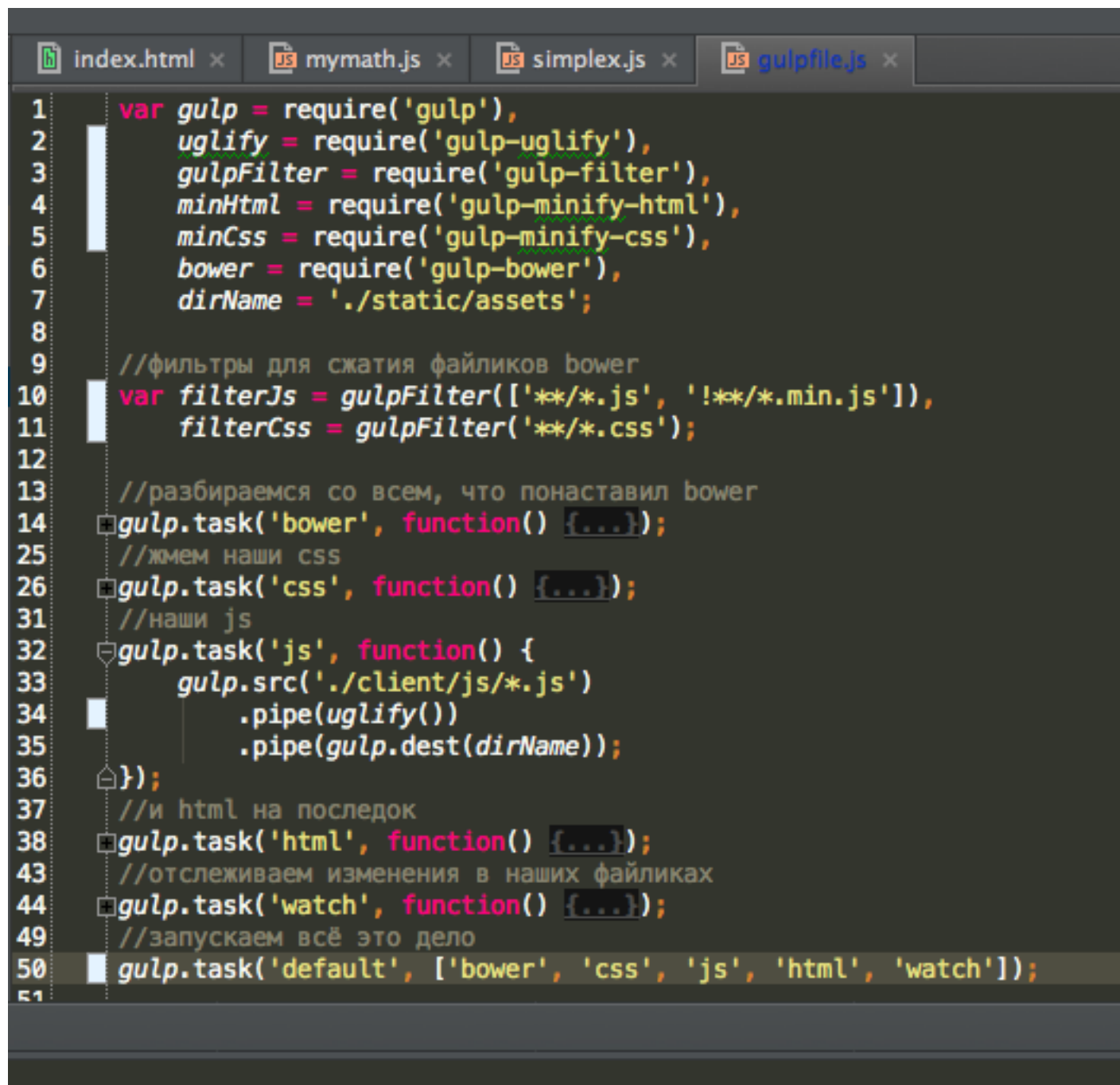
Поэтому в соответствии с (2.10)

$$\|P\| = \max_{f_j=\pm 1} \max(|p(0,0)|, |p(1,0)|, |p(0,1)|, |p(1,1)|) = \frac{15}{7}.$$

Соотношения (2.11) принимают вид $16/7 < 19/7 = 19/7$.

Применяя (2.13) или (2.14) находим $\alpha(Q_2; S) = 16/7$.

Значения в программе:



```

1  var gulp = require('gulp'),
2      uglify = require('gulp-uglify'),
3      gulpFilter = require('gulp-filter'),
4      minHtml = require('gulp-minify-html'),
5      minCss = require('gulp-minify-css'),
6      bower = require('gulp-bower'),
7      dirName = './static/assets';
8
9  //фильтры для сжатия файлов bower
10 var filterJs = gulpFilter(['**/*.js', '!**/*.min.js']),
11     filterCss = gulpFilter(['**/*.css']);
12
13 //разбираемся со всем, что понаставил bower
14 gulp.task('bower', function() {...});
15 //жмем наши css
26 gulp.task('css', function() {...});
31 //наши js
32 gulp.task('js', function() {
33     gulp.src('./client/js/*.js')
34         .pipe(uglify())
35         .pipe(gulp.dest(dirName));
36 });
37 //и html напоследок
38 gulp.task('html', function() {...});
43 //отслеживаем изменения в наших файлах
44 gulp.task('watch', function() {...});
49 //запускаем всё это дело
50 gulp.task('default', ['bower', 'css', 'js', 'html', 'watch']);
51

```

Возьмем вектор $v = (1; 1)$. Найдем величины m_1, \dots, m_3 с помощью формулы (2.15). Получаем

$$m_1 = \frac{2}{7}, m_2 = \frac{10}{7}, m_3 = -\frac{12}{7}$$

Далее

$$\alpha_1 = 0, \alpha_2 = 0, \alpha_3 = 1, \beta_1 = \frac{1}{6}, \beta_2 = \frac{5}{6}, \beta_3 = 0, .$$

И находим концы максимального вписанного в симплекс отрезка заданного направления

$$a = 1 * (0, \frac{1}{4}), b = \frac{1}{6} * (1, 0) + \frac{5}{6} * (\frac{1}{2}, 1) = (\frac{7}{12}, \frac{5}{6},).$$

Его длина суть

$$d^v(S) = \frac{2\|v\|}{\sum_{j=1}^{n+1} |m_j|} = \frac{2\sqrt{3}}{3}.$$

А величина $\Sigma(S, v)$ равна $\frac{3\sqrt{2}}{4}$. Результат выполнения программы

```
index.html × mymath.js × simplex.js × gulpfile.js ×
1  var gulp = require('gulp'),
2      uglify = require('gulp-uglify'),
3      gulpFilter = require('gulp-filter'),
4      minHtml = require('gulp-minify-html'),
5      minCss = require('gulp-minify-css'),
6      bower = require('gulp-bower'),
7      dirName = './static/assets';
8
9  //фильтры для сжатия файлов bower
10 var filterJs = gulpFilter(['**/*.js', '!**/*.min.js']),
11     filterCss = gulpFilter(['**/*.css']);
12
13 //разбираемся со всем, что понаставил bower
14 gulp.task('bower', function() {...});
25 //жмем наши css
26 gulp.task('css', function() {...});
31 //наши js
32 gulp.task('js', function() {
33     gulp.src('./client/js/*.js')
34         .pipe(uglify())
35         .pipe(gulp.dest(dirName));
36 });
37 //и html напоследок
38 gulp.task('html', function() {...});
43 //отслеживаем изменения в наших файлах
44 gulp.task('watch', function() {...});
49 //запускаем всё это дело
50 gulp.task('default', ['bower', 'css', 'js', 'html', 'watch']);
51
```

Возьмем параллелограмм V , который задается векторами $v_1 = (1; 0)$, $v_2 = (1, 1)$ и посчитаем для него величину $\alpha(V, S)$ используя формулу (2.21). Т.к. v_2 есть вектор из предыдущих расчетов, а v_1 – вектор, параллельный координатной оси, то нет необходимости пересчитывать $d^{v^{(i)}}$. Получаем

$$\alpha(V, S) = \frac{1 * 8}{7} + \frac{1}{1} + \frac{12\sqrt{2}}{7\sqrt{2}} = \frac{20}{7} = M$$

4.2 Случай $n = 3$

Пусть $n = 3$, S – тетраэдр с вершинами $x^{(1)} = (1, 0, 0)$, $x^{(2)} = (0, 1, 0)$, $x^{(3)} = (0, 0, 1)$, $x^{(4)} = (1, 1, 1)$. Это правильный тетраэдр, вписанный в куб $Q_3 = [0, 1]^3$. В рассматриваемой ситуации

$$\mathbf{A} := \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}, \mathbf{A}^{-1} := \begin{pmatrix} \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \end{pmatrix}.$$

Базисные многочлены Лагранжа –

$$\lambda_1(x) = \frac{1}{2}x_1 - \frac{1}{2}x_2 - \frac{1}{2}x_3 + \frac{1}{2}, \lambda_2(x) = -\frac{1}{2}x_1 + \frac{1}{2}x_2 - \frac{1}{2}x_3 + \frac{1}{2},$$

$$\lambda_3 = -\frac{1}{2}x_1 - \frac{1}{2}x_2 + \frac{1}{2}x_3 + \frac{1}{2}, \lambda_4 = \frac{1}{2}x_1 + \frac{1}{2}x_2 + \frac{1}{2}x_3 - \frac{1}{2}.$$

Результат выполнения программы:

0.5	-0.5	-0.5	0.5
-0.5	0.5	-0.5	0.5
-0.5	-0.5	0.5	0.5
0.5	0.5	0.5	-0.5

Поэтому симплекс S (с границей) задаётся системой линейных неравенств

$$x_1 - x_2 - x_3 \geq -1, -x_1 + x_2 - x_3 \geq -1,$$

$$-x_1 - x_2 + x_3 \geq -1, x_1 + x_2 + x_3 \geq 1,$$

Вычислим осевые диаметры S и максимальные отрезки, параллельные координатным осям. Формула (2.3) даёт $d_1(S) = d_2(S) = d_3(S) = 1$. В программе получаем:

$$[1, 1, 1]$$

Факт совпадения всех $d_i(S)$ следует также из (2.6) и того, что площади проекций S на координатные плоскости одинаковы (и равны 1). Числа s_{ij} и t_{ij} оказываются следующими:

$$s_{11} = \frac{1}{2}, s_{12} = 0, s_{13} = 0, s_{14} = \frac{1}{2};$$

$$\begin{aligned}
t_{11} &= 0, t_{12} = \frac{1}{2}, t_{13} = \frac{1}{2}, t_{14} = 0; \\
s_{21} &= 0, s_{22} = \frac{1}{2}, s_{23} = 0, s_{24} = \frac{1}{2}; \\
t_{21} &= \frac{1}{2}, t_{22} = 0, t_{23} = \frac{1}{2}, t_{24} = 0; \\
s_{31} &= 0, s_{32} = 0, s_{33} = \frac{1}{2}, s_{34} = \frac{1}{2}; \\
t_{31} &= \frac{1}{2}, t_{32} = \frac{1}{2}, t_{33} = 0, t_{34} = 0.
\end{aligned}$$

Поэтому

$$\begin{aligned}
y_+^{(1)} &= \sum_{j=1}^4 s_{1j} x^{(j)} = \frac{1}{2} (x^{(1)} + x^{(4)}) = \left(1, \frac{1}{2}, \frac{1}{2}\right), \\
y_-^{(1)} &= \sum_{j=1}^4 t_{1j} x^{(j)} = \frac{1}{2} (x^{(2)} + x^{(3)}) = \left(0, \frac{1}{2}, \frac{1}{2}\right), \\
y_+^{(2)} &= \sum_{j=1}^4 s_{2j} x^{(j)} = \frac{1}{2} (x^{(2)} + x^{(4)}) = \left(\frac{1}{2}, 1, \frac{1}{2}\right), \\
y_-^{(2)} &= \sum_{j=1}^4 t_{2j} x^{(j)} = \frac{1}{2} (x^{(1)} + x^{(3)}) = \left(\frac{1}{2}, 0, \frac{1}{2}\right), \\
y_+^{(3)} &= \sum_{j=1}^4 s_{3j} x^{(j)} = \frac{1}{2} (x^{(3)} + x^{(4)}) = \left(\frac{1}{2}, \frac{1}{2}, 1\right), \\
y_-^{(3)} &= \sum_{j=1}^4 t_{3j} x^{(j)} = \frac{1}{2} (x^{(1)} + x^{(2)}) = \left(\frac{1}{2}, \frac{1}{2}, 0\right),
\end{aligned}$$

Максимальными в S отрезками, параллельными координатным осям, оказываются отрезки единичной длины, соединяющие середины противоположных (скрещивающихся) рёбер. Например, максимальный отрезок, параллельный оси x_1 , имеет концы $y_+^{(1)} = (1, 1/2, 1/2)$ и $y_-^{(1)} = (0, 1/2, 1/2)$. Указанные отрезки пересекаются в центре куба.

Найдём $\xi(S)$. Нетрудно видеть, что при всех $j = 1, 2, 3, 4$

$$\max_{x \in \text{ver}(Q_3)} (-\lambda_j(x)) = \frac{1}{2},$$

Поэтому по формуле (2.8) $\xi(S) = 4 \cdot (1/2) + 1 = 3$. Пусть $P : C(Q_3) \rightarrow \Pi_1(\mathbb{R}^3)$ – интерполяционный проектор, узлы которого совпадают с вершинами S . Применяя формулу

$$p(x) = Pf(x) = f_1\lambda_1(x) + f_2\lambda_2(x) + f_3\lambda_3(x) + f_4\lambda_4(x).$$

и явный вид λ_j , найдём значения p в каждой из восьми вершин Q_3 : $p(0, 0, 0) = (1/2)(f_1 + f_2 + f_3 - f_4)$, $p(1, 0, 0) = f_1$ и т.д. Как оказывается,

$$\|P\| = \max_{f_j=\pm 1} \max_{x \in \text{ver}(Q_n)} |p(x)| = 2.$$

Двойное неравенство (2.11) имеет форму равенства – каждая из его частей равняется 3. Применяя (2.13) или (2.14) находим $\alpha(Q_3; S) = 3$.

Выполнение программы даёт нам следующие результаты:

```
index.html × mymath.js × simplex.js × gulpfile.js ×
1  var gulp = require('gulp'),
2      uglify = require('gulp-uglify'),
3      gulpFilter = require('gulp-filter'),
4      minHtml = require('gulp-minify-html'),
5      minCss = require('gulp-minify-css'),
6      bower = require('gulp-bower'),
7      dirName = './static/assets';
8
9  //фильтры для сжатия файлов bower
10 var filterJs = gulpFilter(['**/*.js', '!**/*.min.js']),
11     filterCss = gulpFilter(['**/*.css']);
12
13 //разбираемся со всем, что понаставил bower
14 gulp.task('bower', function() {...});
25 //жмем наши css
26 gulp.task('css', function() {...});
31 //наши js
32 gulp.task('js', function() {
33     gulp.src('./client/js/*.js')
34         .pipe(uglify())
35         .pipe(gulp.dest(dirName));
36 });
37 //и html напоследок
38 gulp.task('html', function() {...});
43 //отслеживаем изменения в наших файлах
44 gulp.task('watch', function() {...});
49 //запускаем всё это дело
50 gulp.task('default', ['bower', 'css', 'js', 'html', 'watch']);
51
```

Возьмем вектор $v = (1; 1; 1)$. Найдем величины m_1, \dots, m_4 с помощью формулы (2.15). Получаем

$$m_1 = -\frac{1}{2}, m_2 = -\frac{1}{2}, m_3 = -\frac{1}{2}, m_4 = \frac{3}{2}.$$

Далее

$$\alpha_1 = \frac{1}{3}, \alpha_2 = \frac{1}{3}, \alpha_3 = \frac{1}{3}, \alpha_4 = 0, \beta_1 = 0, \beta_2 = 0, \beta_3 = 0, \beta_4 = 1.$$

И находим концы максимального вписанного в симплекс отрезка заданного направления

$$a = \frac{1}{3}*(1, 0, 0) + \frac{1}{3}*(0, 1, 0) + \frac{1}{3}*(0, 0, 1) = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3}), b = 1*(1, 1, 1) = (1, 1, 1).$$

Его длина суть

$$d^v(S) = \frac{2\|v\|}{\sum_{j=1}^{n+1}|m_j|} = \frac{2\sqrt{3}}{3}.$$

А величина $\Sigma(S, v)$ равна $\frac{\sqrt{3}}{2}$. Результат выполнения программы

```

1  var gulp = require('gulp'),
2      uglify = require('gulp-uglify'),
3      gulpFilter = require('gulp-filter'),
4      minHtml = require('gulp-minify-html'),
5      minCss = require('gulp-minify-css'),
6      bower = require('gulp-bower'),
7      dirName = './static/assets';
8
9  //фильтры для сжатия файлов бовер
10 var filterJs = gulpFilter(['**/*.js', '!**/*.min.js']),
11     filterCss = gulpFilter(['**/*.css']);
12
13 //разбираемся со всем, что понаставил бовер
14 gulp.task('bower', function() {...});
15 //жмем наши css
26 gulp.task('css', function() {...});
31 //наши js
32 gulp.task('js', function() {
33     gulp.src('./client/js/*.js')
34         .pipe(uglify())
35         .pipe(gulp.dest(dirName));
36 });
37 //и html напоследок
38 gulp.task('html', function() {...});
43 //отслеживаем изменения в наших файлах
44 gulp.task('watch', function() {...});
49 //запускаем всё это дело
50 gulp.task('default', ['bower', 'css', 'js', 'html', 'watch']);
51

```


Возьмем параллелограмм V , который задается векторами $v_1 = (1; 1; 1)$, $v_2 = (1, 0, 0)$ и посчитаем для него величину $\alpha(V, S)$ используя формулу (2.21). Т.к. v_1 есть вектор из предыдущих расчетов, а v_2 и v_3 параллельны координатным осям, то нет необходимости пересчитывать $d^{v^{(i)}}$. Получаем

$$\alpha(V, S) = \frac{3\sqrt{3}}{2\sqrt{3}} + \frac{1}{1} + \frac{1}{1} = \frac{7}{2} = M$$

5 Заключение

Подводя итог, можно сказать, что в данной работе реализованы все поставленные задачи, а именно:

- рассмотрен алгоритм нахождения осевых диаметров и прочих величин, по заданным вершинам невырожденного симплекса
- на примерах разобраны случаи $n = 2$ (S - треугольник) и $n = 3$ (S - тетраэдр)
- написана программа, реализующая данный алгоритм на ЭВМ

6 Библиография

Список литературы

- [1] Невский М.В., Иродова И.П. Некоторые вопросы теории приближения функций: учеб. пособие. Ярославль: ЯрГУб 1999. 92 с.
- [2] Невский М.В., Оценки для минимальной нормы проектора при линейной интерполяции по вершинам n -мерного куба // Моделирование и анализ информационных систем. 2003. Т. 10, №1. С. 9 - 19.
- [3] Невский М.В., Об одном соотношении для минимальной нормы интерполяционного проектора // Моделирование и анализ информационных систем. 2009. Т. 16, №1. С. 24 - 43.
- [4] Невский М.В., Об одном свойстве n -мерного симплекса // Матем. заметки. 2010. Т. 87, №4. С. 580 - 593.
- [5] Невский М.В., О некоторых свойствах базисных многочленов Лагранжа // Преподавание математики и компьютерных наук в классическом университете. Материалы 3-й научно-методической конференции преподавателей математического факультета и факультета информатики и вычислительной техники ЯрГУ им. П.Г.Демидова. Ярославль, 2010. С. 111 - 117.
- [6] Невский М.В., Об осевых диаметрах выпуклого тела // Матем. заметки. 2011. Т. 90, №2. С. 313 - 315.
- [7] Nevskii M., Properties of axial diametres of a simplex // Discrete Comput. Geom. 2011. V. 87, №2. P. 301 - 312.

7 Приложение

1. Файл mymath.js (Класс с необходимыми для вычислений методами)

```
1  function transpose(a) {
2      return a[0].map(function (v, i) {
3          return a.map(function (r, j) {
4              return a[j][i];
5          });
6      });
7  }
8
9  function getA(m) {
10     for (i in m) {
11         m[i].push(1);
12     }
13     return m;
14 }
15
16 function _determinant(A)
17 {
18     var N = A.length, B = [], denom = 1, exchanges = 0;
19     for (var i = 0; i < N; ++i) {
20         B[i] = [];
21         for (var j = 0; j < N; ++j) B[i][j] = A[i][j];
22     }
23     for (var i = 0; i < N - 1; ++i) {
24         var maxN = i, maxValue = Math.abs(B[i][i]);
25         for (var j = i + 1; j < N; ++j) {
26             var value = Math.abs(B[j][i]);
27             if (value > maxValue) {
28                 maxN = j;
29                 maxValue = value;
30             }
31         }
32         if (maxN > i) {
33             var temp = B[i];
34             B[i] = B[maxN];
```

```

35         B[maxN] = temp;
36         ++exchanges;
37     }
38     else {
39         if (maxValue == 0) return maxValue;
40     }
41     var value1 = B[i][i];
42     for (var j = i + 1; j < N; ++j) {
43         var value2 = B[j][i];
44         B[j][i] = 0;
45         for (var k = i + 1; k < N; ++k)
46             B[j][k] = (B[j][k] * value1 -
47                 B[i][k] * value2) / denom;
48     }
49     denom = value1;
50 }
51 if (exchanges % 2) return -B[N - 1][N - 1];
52 else return B[N - 1][N - 1];
53 }
54
55 function _matrixCofactor(i, j, A)
56 {
57     var N = A.length, sign = ((i + j) % 2 == 0) ? 1 : -1;
58     for (var m = 0; m < N; m++) {
59         for (var n = j + 1; n < N; n++) A[m][n - 1] = A[m][n];
60         A[m].length--;
61     }
62     for (var k = i + 1; k < N; k++) A[k - 1] = A[k];
63     A.length--;
64     return sign * _determinant(A);
65 }
66
67 function _adjugateMatrix(A)
68 {
69     var N = A.length, B = [], adjA = [];
70     for (var i = 0; i < N; i++) {
71         adjA[i] = [];

```

```

72         for (var j = 0; j < N; j++) {
73             for (var m = 0; m < N; m++) {
74                 B[m] = [];
75                 for (var n = 0; n < N; n++) B[m][n] = A[m][n];
76             }
77             adjA[i][j] = _matrixCofactor(j, i, B);
78         }
79     }
80     return adjA;
81 }
82
83 function inverse(A)
84 {
85     var det = _determinant(A);
86     if (det == 0) return false;
87     var N = A.length, a = _adjugateMatrix(A);
88     for (var i = 0; i < N; i++) {
89         for (var j = 0; j < N; j++) {
90             a[i][j] /= det;
91             a[i][j] = a[i][j] || 0;
92         }
93     }
94     return a;
95 }
96
97
98 function ifPointIntoSimplex(l, p) {
99     var i, j, arr = [];
100    for (i = 0; i < l.length; i++) {
101        var t = 0;
102        for (j = 0; j < l[i].length; j++) {
103            t = t + l[i][j] * p[j] ? p[j] : 1;
104        }
105        arr.push(t);
106    }
107    for (i in arr) {
108        if (arr[i] < 0) {

```

```

109         return -1;
110     }
111     else if (arr[i] == 0) {
112         return 0;
113     }
114 }
115 return 1;
116 }
117
118 function findDiamters(l) {
119     var d = [], i = 0, j = 0;
120     for (i = 0; i < l.length - 1; i++) {
121         d[i] = 0;
122         for (j = 0; j < l[i].length; j++) {
123             d[i] += Math.abs(l[i][j]);
124         }
125         d[i] = 2 / d[i];
126     }
127     return d;
128 }
129
130 function _findSum(l) {
131     var i, j, result = [];
132     for (i = 0; i < l.length - 1; i++) {
133         var sum = 0;
134         for (j = 0; j < l[i].length; j++) {
135             sum += Math.abs(l[i][j]);
136         }
137         result.push(sum);
138     }
139     return result;
140 }
141
142 function _findSOrT(el, s) {
143     if (s) {
144         return (Math.abs(el) + el);
145     }

```

```

146         return (Math.abs(el) - el);
147     }
148
149     function endPoints(l, a) {
150         var sum = _findSum(l), i, j, k, s = [],
151             t = [], result = [], n = l.length;
152         for (i = 0; i < n - 1; i++) {
153             s[i] = [];
154             t[i] = [];
155             for (j = 0; j < n; j++) {
156                 s[i][j] = (s[i][j] || 0) + _findSOrT(l[i][j], true);
157                 t[i][j] = (t[i][j] || 0) + _findSOrT(l[i][j]);
158             }
159             s[i] = s[i].map(function (num) {
160                 return num / sum[i]
161             });
162             t[i] = t[i].map(function (num) {
163                 return num / sum[i]
164             });
165         }
166         result = [[], []];
167         for (k = 0; k < n - 1; k++) {
168             result[0][k] = [];
169             result[1][k] = [];
170             for (i = 0; i < n - 1; i++) {
171                 for (j = 0; j < n; j++) {
172                     result[0][k][i] =
173                         (result[0][k][i] || 0) + s[i][j] * a[j][k];
174                     result[1][k][i] =
175                         (result[1][k][i] || 0) + t[i][j] * a[j][k];
176                 }
177             }
178         }
179         return result;
180     }
181
182     function _getQ(n, f) {

```



```

183     var i, j, arr = [], m = Math.pow(2, n - 1);
184     for (i = 0; i < m; i++) {
185         arr.push([]);
186         var m_2 = i.toString(2);
187         if (m_2.length < n - 1) {
188             var l = m_2.length;
189             for (j = 1; j < n - 1; j++) {
190                 m_2 = '0' + m_2;
191             }
192         }
193         for (j = 0; j < n - 1; j++) {
194             arr[i].push(f && !parseInt(m_2[j]) ? -1
195                 : parseInt(m_2[j]));
196         }
197     }
198     return arr;
199 }
200
201 function findKsi(l) {
202     var q = _getQ(l.length), i, k, j, arr = [], lambda = [];
203     for (i = 0; i < l.length; i++) {
204         lambda[i] = [];
205         for (j = 0; j < q.length; j++) {
206             lambda[i][j] = 0;
207             for (k = 0; k < l.length; k++) {
208                 lambda[i][j] += l[k][i] *
209                     (q[j] == undefined ||
210                     (q[j][k] == undefined) ? 1 : q[j][k])
211             }
212         }
213     }
214     for (i = 0; i < lambda.length; i++) {
215         lambda[i] = lambda[i].map(function (num) {
216             return -num;
217         });
218         arr[i] = Math.max.apply(Math, lambda[i]);
219     }

```

```

220     return l.length * Math.max.apply(Math, arr) + 1;
221 }
222
223 function findP(l) {
224     var q = _getQ(l.length), i, k, j, arr = [], lambda = [];
225     for (i = 0; i < l.length; i++) {
226         lambda[i] = [];
227         for (j = 0; j < q.length; j++) {
228             lambda[i][j] = 0;
229             for (k = 0; k < l.length; k++) {
230                 lambda[i][j] += l[k][i] *
231                     (q[j] == undefined ||
232                      (q[j][k] == undefined) ? 1 : q[j][k])
233             }
234         }
235     }
236     for (i = 0; i < lambda[0].length; i++) {
237         for (j = 0; j < l.length; j++)
238             arr[i] = (arr[i] || 0) + Math.abs(lambda[j][i]);
239     }
240     return Math.max.apply(Math, arr);
241 }
242
243 function findAlpha(d) {
244     var i, a = 0;
245     for (i = 0; i < d.length; i++) {
246         a += 1 / d[i];
247     }
248     return a;
249 }
250
251 function _findM(l, v) {
252     var i, j, m = [];
253     for (i = 0; i < l.length; i++) {
254         m[i] = 0;
255         for (j = 0; j < v.length; j++) {
256             m[i] += v[j] * l[j][i];

```

```

257         }
258     }
259     return m;
260 }
261
262 function _findMSum(m) {
263     for (var s = 0, k = m.length; k; s += Math.abs(m[--k]));
264     return s;
265 }
266
267 function vectorEndpoints(a, l, v) {
268     var m = _findM(l, v), sum = _findMSum(m), i, j,
269         k, alpha = [], beta = [], result = [[], []],
270         n = l.length;
271     for (i = 0; i < n; i++) {
272         alpha[i] = _findSOrT(m[i]) / sum;
273         beta[i] = _findSOrT(m[i], true) / sum;
274     }
275     for (k = 0; k < n - 1; k++) {
276         for (j = 0; j < n; j++) {
277             result[0][k] = (result[0][k] || 0) +
278                 alpha[j] * a[j][k];
279             result[1][k] = (result[1][k] || 0) +
280                 beta[j] * a[j][k];
281         }
282     }
283     return result;
284 }
285
286 function _findVectorNorm(v) {
287     var sum = 0;
288     for (var i = 0; i < v.length; i++) {
289         sum += v[i] * v[i];
290     }
291     return Math.sqrt(sum);
292 }
293

```

```

294 function findVectorD(l, v) {
295     var m = _findM(l, v), sum = _findMSum(m),
296         norm = _findVectorNorm(v);
297     return (2 * norm) / sum;
298 }
299
300 function findVectorKsi(l, v, a) {
301     var m = _findM(l, v), sum = _findMSum(m),
302         norm = _findVectorNorm(v), det = _determinant(a),
303         f = [1, 1, 2, 6, 24, 120, 720, 5040, 40320,
304             362880, 3628800, 39916800, 479001600];
305     return (det * sum) / (norm * 2 * f[v.length - 1]);
306 }
307
308 function findVAlpha(l, v) {
309     var i, j, result = 0;
310     for (i = 0; i < v.length; i++) {
311         result += _findVectorNorm(v[i]) / findVectorD(l, v[i]);
312     }
313     return result;
314 }
315
316 function findX(l, a) {
317     var i, j, k, sum = 0, result = [];
318     for (i = 0; i < l.length; i++) {
319         for (j = 0; j < l.length - 1; j++) {
320             sum += Math.abs(l[j][i]);
321         }
322     }
323     for (k = 0; k < l.length - 1; k++) {
324         result[k] = 0;
325         for (i = 0; i < l.length; i++) {
326             for (j = 0; j < l.length - 1; j++) {
327                 result[k] += Math.abs(l[j][i]) * a[i][k];
328             }
329         }
330     }

```

```

331     result = result.map(function (num) {
332         return num / (sum - 2)
333     });
334     return result;
335 }
336
337 function findBarycentricCoords(l, x) {
338     var i, j, result = [];
339     for (i = 0; i < l.length; i++) {
340         result[i] = 0;
341         for (j = 0; j < l[i].length; j++) {
342             result[i] += l[j][i] * (x[j] == undefined ? 1 : x[j]);
343         }
344     }
345     return result;
346 }
347
348
349

```