

Правительство Российской Федерации
Национальный исследовательский университет
«Высшая школа экономики»

Факультет компьютерных наук
Департамент программной инженерии

СОГЛАСОВАНО

Научный руководитель
доцент департамента
математики на факультете
экономических наук, к.ф.-м.н

_____ В. Л. Чернышев
« ____ » _____ 2017 г.

УТВЕРЖДАЮ

Академический руководитель
образовательной программы
«Программная инженерия»
профессор департамента программной
инженерии, канд. техн. наук

_____ В. В. Шилов
« ____ » _____ 2017 г.

Отчёт по курсовой работе

**Изучение приближения многочленами функций, возникающих при
изучении динамики на метрических графах**

по направлению подготовки бакалавров 09.03.04 «Программная инженерия»

Выполнил:
студент группы БПИ152
образовательной программы
09.03.04 «Программная инженерия»
Е. А. Соловьев

_____ 2017 г.
« ____ » _____

Москва, 2017

Реферат

Отчёт 34 с., 28 рис., 11 источн., 4 прил.

Ключевые слова: *функции на графах; метрические графы; полиномиальные приближения; число точек на графе; число точек в симплексе.*

В отчёте представлены результаты курсовой работы на тему «Изучение приближения многочленами функций, возникающих при изучении динамики на метрических графах», выполненной на основе приказа Национального исследовательского университета «Высшая школа экономики» № 2.3-02/1804-01 от 18.04.17.

Объект исследования – функции, возникающие при изучении динамических процессов на метрических графах.

Предмет исследования – полиномиальные приближения для функции числа точек в симплексе и числа точек на графе.

Цель исследования – экспериментальным путём убедиться в точности оценки функции числа точек на графе многочленами и разработать программный инструмент, позволяющий выписывать аппроксимирующие многочлены по заданному графу и его вершине.

Задачи исследования:

- разработка программы, выписывающей по заданному графу полиномиальное приближение функции числа точек на нём
- экспериментальная проверка корректности и точности полиномиального приближения для функции числа точек на графе
- разработка программы для подсчёта числа точек на графе
- разработка программы для подсчёта числа точек в симплексе
- выписывание полиномиальных приближений для числа точек в симплексе для некоторых измерений
- экспериментальная проверка корректности полиномиального приближения для функции числа точек в симплексе

Методы исследования:

- изучение научных статей
- проведение компьютерных экспериментов
- элементы статистического анализа

Научная новизна работы заключается в проверке качества приближения многочленами функции числа точек на графе (и экспериментальном подтверждении теоремы о данном приближении).

Достоверность научных результатов работы подтверждается результатами экспериментов сравнения истинных значений рассматриваемых функции с полученными в результате теоретических выкладок многочленами.

Практическая значимость работы заключается в разработке программных инструментов, позволяющим исследователям получать истинные значения рассматриваемых функций, а также их полиномиальные приближения.

Результаты работы:

- экспериментальным путём проверены оценки, указанные в теореме о полиномиальном приближении числа точек на графе, обсуждаются пути улучшения данного приближения
- разработана программа для выписывания полиномиального приближения функции числа точек на графе по заданному графу
- разработана программа для подсчёта числа точек на графе
- разработана программа для подсчёта числа точек в симплексе
- выписаны точные выражения полиномиальных приближений для числа точек в симплексе для некоторых измерений

Содержание

1	Введение	5
1.1	Предмет исследования	5
1.2	Цели и задачи работы	5
1.3	Новизна и достоверность результатов	5
1.4	Ценность результатов	5
2	Теоретическая часть	6
2.1	Число точек в симплексе	6
2.2	Число точек на графе	7
3	Используемые алгоритмы	7
3.1	Алгоритм моделирования движения точек на графе	7
3.2	Алгоритм нахождения числа точек в симплексе	9
3.3	Алгоритм обхода графа в глубину	9
3.4	Алгоритм проверки графа на связность	10
3.5	Алгоритм проверки графа на ацикличность	11
3.6	Алгоритм поиска связных поддеревьев данного дерева, содержащих данную вершину	12
3.7	Алгоритм поиска кратчайших путей от некоторой вершины до всех в дереве	13
4	Описание экспериментов, анализ результатов	13
4.1	Описание экспериментов	13
4.1.1	Подсчёт числа точек в симплексе	13
4.1.2	Подсчёт числа точек на графе	14
4.2	Анализ результатов	14
4.2.1	Подсчёт числа точек в симплексе	14
4.2.2	Подсчёт числа точек на графе	18
5	Заключение	24
	Список использованных источников	24
6	Приложения	26
6.1	Многочлены $B_k^{(k)}$ и многочлены для числа точек на графах-примерах	26
6.2	Описание разработанных программ	28
6.2.1	Описание программы для подсчёта числа точек в симплексе	28
6.2.2	Описание программы для подсчёта числа точек на графе	29
6.2.3	Описание программы для генерации полиномиальных приближений для числа точек на графе	31
6.2.4	Описание формата представления графа в текстовом виде	32
6.3	Исходный код разработанных программ и экспериментальные данные	33
6.4	Используемые понятия и определения	33

1 Введение

1.1 Предмет исследования

В данной работе рассматриваются функции, возникающие при рассмотрении двух видов динамических задач на графах: задача поиска числа точек с натуральными координатами в симплексе (сводящаяся к задаче поиска числа натуральных решений некоторого линейного неравенства) и задаче поиска числа точек на графе и их полиномиальные приближения. Нахождение точного значения решений данных задач оказывается вычислительно трудоёмким, из-за чего становится целесообразно рассмотреть приближения возникающих функций другими простыми функциями, коими являются многочлены. Являясь простыми в вычислении, они могут использоваться в практических задачах для приближённого вычисления истинных значений данных функций, если точность приближения доказана теоретически и экспериментально.

Актуальность работы подтверждается интересом к данным функциям и их приближениям учёных и исследователей в сфере математической физики (см., например, [2], [3]).

1.2 Цели и задачи работы

Целью работы является проверка достоверности полиномиальных приближений для функций, возникающих при анализе динамических систем на графах.

Задачами работы (и важным её результатом) является разработка программного инструмента, позволяющего исследователям получать такие приближения, а также проверка качества работы этого инструмента и достоверности приближений данных функций многочленами на некоторых примерах.

1.3 Новизна и достоверность результатов

Новизна полученных в данной работе результатов заключается в проверке качества полиномиального приближения для числа точек на графе, описанного в статье А. А. Толченникова и В. Л. Чернышева ¹ (фактически, данная работа позволила проверить корректность данного приближения экспериментальным путём ²) Достоверность результатов подтверждается формальной проверкой корректности алгоритма, вычисляющего точные значения возникающих функций, и результатами экспериментов.

1.4 Ценность результатов

Несмотря на то, что работа носит преимущественно экспериментальный характер, полученные результаты могут использоваться для теоретических исследований в области динамических систем. В частности, выписаны точные виды многочленов Норлунда $B_k^{(k)}$ для некоторых значений k и разработан инструмент, позволяющий моделировать движение точек на графе и находить число точек в симплексе (то есть находить точные значения рассматриваемых функций), а также генерирующий полиномиальные приближения данных функций. Этот инструмент могут использовать исследователи в области динамических систем и других областей прикладной математики в своих целях.

¹Статья готовится к публикации.

²Подробнее о таком способе исследований рассказывается, например, в книге В. И. Арнольда [1]

2 Теоретическая часть

2.1 Число точек в симплексе

Задача приближения функции числа точек на графе многочленами приводит к рассмотрению известной (и привлекающей внимание исследователей по сей день) задачи о подсчете числа целых точек в расширяющихся симплексах. Обсудим её подробнее.

Зафиксируем некоторые положительные действительные числа $w_i, i = 1 \dots k$. Рассмотрим неравенство $\sum_{i=1}^k n_i w_i \leq \lambda$. Обозначим число его целых положительных решений как $N_k(\lambda | w_1, \dots, w_k)$ и условимся считать, что в точках разрыва функция $N_k(\lambda)$ (в дальнейшем может применяться обозначение $N(t)$) равна полусумме пределов слева и справа.

Заметим, что значение функции $N_k(\lambda)$ – это число точек, находящихся в симплексе (многограннике) в k -мерном пространстве, который задаётся системой неравенств

$$\begin{cases} n_i \geq 0, \\ \sum_{i=1}^k n_i w_i \leq \lambda \end{cases}$$

Д. С. Спенсер ([4]) доказал следующую теорему.

Теорема 1. Для почти всех наборов $w_i, i = 1 \dots k$ функция $N_k(\lambda)$ аппроксимируется некоторым многочленом $R_k(\lambda)$ с точностью до логарифма степени k :

$$\forall \varepsilon > 0 : N_k(\lambda) - R_k\left(\lambda + \sum_{i=1}^k w_i\right) = O((\log \lambda)^{k+\varepsilon})$$

где

$$R_k(\lambda | w_1, \dots, w_k) = \frac{B_k^{(k)}(\lambda | w_1, \dots, w_k)}{k! w_1 \dots w_k}$$

и $B_k^{(k)}$ – многочлены, рассматриваемые Норлундом в как обобщения многочленов Бернулли и определяемые следующими рекуррентными соотношениями:

$$B_\nu^{(k)}(x | w_1, \dots, w_k) = \sum_{s=0}^{\nu} \binom{\nu}{s} x^s B_{\nu-s}^{(k)}[w_1, \dots, w_k]$$

и числа

$$B_\nu^{(k)}[w_1, \dots, w_k] = \sum \frac{\nu!}{s_1! \dots s_k!} w_1^{s_1} \dots w_k^{s_k} B_{s_1} \dots B_{s_k},$$

где B_{s_i} – числа Бернулли и суммирование ведется по неотрицательным s_1, \dots, s_k таким, что $s_1 + \dots + s_k = \nu$. В частности,

$$B_0^{(k)}[w_1, \dots, w_k] = 1, \quad B_1^{(k)}[w_1, \dots, w_k] = -\frac{1}{2}(w_1 + \dots + w_k).$$

$$B_k^{(k)}(x | w_1, \dots, w_k) = x^k - \frac{k}{2}(w_1 + \dots + w_k) x^{k-1} + O(x^{k-2}).$$

В данной работе, в частности, выписаны многочлены $R_k^{(k)}$ для $k = 2 \dots 5$ (см. приложение 6.1).

2.2 Число точек на графе

Пусть дано дерево $G = (V, E)$ с положительными весами (длинами рёбер) $e_1, \dots, e_{|E|}$, причём $e_1, \dots, e_{|E|}$ линейно независимы над \mathbb{Q} . В момент времени $t = -0.5|e|$ в середине некоторого ребра e находится точка (узкий волновой пакет), которая начинает двигаться в сторону одной из двух вершин, соединяемых этим ребром (назовём эту вершину корнем дерева, причём это не обязательно вершина степени один). Все точки движутся с единичной скоростью. Попадая в некоторую вершину степени n , точка исчезает и порождает n точек, начинающих двигаться из этой вершины по всем инцидентным её рёбрам (в том числе и по тому, по которому двигалась исчезнувшая точка). Если несколько точек приходят в одну и ту же вершину степени n в один и тот же момент времени, все они исчезают и порождают всё те же n точек. Обозначим для $t > 0$ $N(t)$ число точек, находящееся на графе в данный момент времени и условимся, что в точках разрыва функция равна полусумме пределов слева и справа.

А. А. Толченников и В. Л. Чернышев доказали ³ следующую теорему (в частности, данная курсовая работа помогла подтвердить теорему экспериментальным образом).

Теорема 2. Пусть $\Gamma = (V, E)$ – дерево с длинами рёбер $t_1, \dots, t_{|E|}$. Тогда для почти всех длин рёбер $t_1, \dots, t_{|E|}$ дерева $\Gamma = (V, E)$

$$N(t) = R(t) + O((\log t)^{|E|-1})$$

где

$$R(t) = R'(t) + R''(t)$$

и многочлены R', R'' определены следующим образом:

$$R'(t) = \sum_{E'} \sum_l (\rho(E, \text{end}(l)) - \rho(E', \text{end}(l))) R_{|E'|} \left(t + \sum_{e_i \in l} t_i \left\lfloor \frac{2t_i}{e_i} \right\rfloor \right),$$

где первое суммирование ведётся по всем подмножествам ребер $E' \subset E$, которые составляют поддереву Γ' в Γ , содержащее корень Γ , а второе суммирование ведётся по всем подмножествам ребер $l \subset E'$, которые образуют путь (возможно, нулевой длины) в поддереве Γ' , и

$$R''(t) = \sum_{E'} \sum_l R_{|E'|-1} \left(t + \sum_{e_i \in l \setminus \text{last}(l)} t_i - t_{\text{last}(l)} \left\lfloor \frac{2t_i}{e_i} \right\rfloor \right),$$

где первое суммирование ведётся по всем подмножествам ребер $E' \subset E$, которые составляют поддереву Γ' в Γ , содержащее корень Γ , а второе суммирование ведётся по всем непустым подмножествам ребер $l \subset E'$, которые образуют путь в поддереве Γ' и таким, что $\rho(E', \text{end}(l)) > 1$. Многочлены R_k определены формулой из формулировки теоремы (1). Здесь $\text{end}(l)$ – конечная вершина пути, состоящего из рёбер l , $\text{last}(l)$ – ребро l , инцидентное $\text{end}(l)$, а $\rho(E', v)$ – число рёбер из E' , инцидентных вершине v .

3 Используемые алгоритмы

3.1 Алгоритм моделирования движения точек на графе

Пусть дано дерево $G = (V, E)$ с положительными весами (длинами рёбер). В момент времени $t = -0.5|e|$ в середине некоторого ребра e находится точка (узкий волновой пакет), которая

³Соответствующая статья готовится к публикации.

начинает двигаться в сторону одной из двух вершин, соединяемых этим ребром (назовём эту вершину корнем дерева, причём это не обязательно вершина степени один). Все точки двигаются с единичной скоростью. Попадая в некоторую вершину степени n , точка перерождается в n точек, начинающих двигаться из этой вершины по всем инцидентным её рёбрам (в том числе и по тому, по которому двигалась исчезнувшая точка). Если несколько точек приходят в одну и ту же вершину степени n в один и тот же момент времени, все они исчезают и порождают всё те же n точек. Обозначим для $t > 0$ $N(t)$ число точек, находящееся на графе в данный момент времени. Требуется находить значения $N(t)$ в некотором заданном диапазоне значений $0 < t < T$ (иными словами, необходимо находить моменты изменения значений данной функции и сами эти значения после старта новых волновых пакетов из соответствующей вершины).

Построим моделирование этого процесса следующим образом. Заведём очередь с приоритетом, элементами которой будут являться находящиеся на графе точки, отсортированные по времени прибытия. Чтобы смоделировать следующие прибытие точки в вершину, нужно просто извлечь из очереди точку с ближайшим временем прибытия (а также извлечь все остальные точки с этим же временем при их наличии). Пусть точка (точки) пришли в вершину степени n в момент времени $t_{current}$. Поместим в очередь n новых точек, времена прибытия которых $t_i = t_{current} + |e_i|$ (здесь e_i – ребро, по которому начала двигаться точка) и зафиксируем новое число точек на графе (это просто количество элементов в очереди). Замечанием к программной реализации алгоритма является необходимость введения временной погрешности (time tolerance) – для избежания проблем, связанных со сложением чисел с плавающей точкой, после извлечения из очереди точки со временем прибытия $t_{current}$, будем извлекать не только все точки, времена прибытия которых строго равны $t_{current}$, но и все точки, времена прибытия которых не больше величины $t_{current} + \varepsilon$, где ε – допустимая погрешность (в частности, при проведении эксперимента использовалось значение $\varepsilon = 10^{-10}$).

Алгоритм 1 Моделирование движения точек на графе

```

1: procedure GRAPH-COUNT( $G = (V, E), \varepsilon, T, startEdge$ )
2:    $packets \leftarrow$  new priority queue
3:    $first \leftarrow$  точка по ребру  $startEdge$  с прибытием в  $t = 0$ 
4:    $packets.add(first)$ 
5:    $currentTime \leftarrow -startEdge.length/2$ 
6:   while  $currentTime < T$  do
7:      $vertices \leftarrow$  new set ▷ вершины, в которые прибыли точки
8:      $lastTime \leftarrow packets.pop().time$ 
9:     while  $packets.count > 0$  and  $packets.top().time \leq lastTime + \varepsilon$  do
10:       $vertices.add(packets.pop().to)$  ▷ извлекаем все пришедшие точки
11:    end while
12:    for  $v \in vertices$  do ▷ отправляем новые точки
13:      for  $e \in edges(v)$  do ▷ по всем инцидентным рёбрам
14:         $newPacket \leftarrow$  точка по ребру  $e$  из вершины  $v$  со временем  $lastTime + e.length$ 
15:         $packets.add(newPacket)$ 
16:      end for
17:    end for
18:     $currentTime \leftarrow lastTime$ 
19:  end while
20: end procedure

```

3.2 Алгоритм нахождения числа точек в симплексе

Пусть дан вектор из положительных действительных числа $w_i, i = 1 \dots k$. Рассмотрим неравенство $\sum_{i=1}^k n_i w_i \leq \lambda$. Требуется найти число его натуральных решений. Будем искать его перебором, начиная из нулевой точки и увеличивая первую компоненту до максимально возможного значения n_1^{\max} . Для всех значений $n_1 = 0 \dots n_1^{\max}$ включительно найдём число решений вышеупомянутого неравенства при условии, что первая компонента равна этому конкретному значению. Далее увеличиваем вторую компоненту и продолжаем рекурсивно.

Алгоритм 2 Подсчёт числа точек в симплексе

```
1: procedure COUNT-POINTS( $w = (w_1, \dots, w_k)^T, \lambda$ )
2:   return 1 + count-points-partial( $w, \lambda, \text{new int}[k], 0, 0.0$ )
3: end procedure
4: procedure COUNT-POINTS-PARTIAL( $w = (w_1, \dots, w_k)^T, \lambda, \text{start}[k], \text{startAt}, \text{sum}$ )
5:   if  $\text{startAt} \geq k$  then
6:     return 0
7:   end if
8:   if  $\text{sum} > \lambda$  then
9:     return 0
10:  end if
11:   $\text{count} \leftarrow \left\lfloor \frac{\lambda - \text{sum}}{w_{\text{startAt}}} \right\rfloor$ 
12:  for  $i = 0.. \text{count} - 1$  do
13:     $\text{sum} \leftarrow \text{sum} + w_{\text{startAt}}$ 
14:     $\text{startPosition}[\text{startAt}] \leftarrow \text{startPosition}[\text{startAt}] + 1$ 
15:     $\text{count} \leftarrow \text{count} + \text{count-points-partial}(w, \lambda, \text{copy}(\text{startPosition}), \text{startAt} + 1, \text{sum})$ 
16:  end for
17:  return  $\text{count}$ 
18: end procedure
```

3.3 Алгоритм обхода графа в глубину

Обход графа в глубину (DFS) – это базовый алгоритм обхода графа, используемый для решения большого количества подзадач при анализе графа.

Пусть дан (возможно, несвязный) граф $G = (V, E)$. Задача заключается в том, чтобы обойти все вершины $v \in V$ графа один раз и выполнить в каждой вершине некоторую функцию $F(v)$.

В алгоритме ниже $\text{adj}[i]$ – множество вершин, смежных вершине i (его можно получить из матрицы смежности графа G).

Алгоритм 3 DFS

```
1: procedure DFS( $G = (V, E)$ ) ▷ обход графа в глубину
2:    $visited \leftarrow \text{new bool}[|V|]$ 
3:   for  $i \leftarrow 0 \dots |V| - 1$  do
4:     if not  $visited[i]$  then
5:       visit( $i$ )
6:     end if
7:   end for
8: end procedure
9: procedure VISIT( $vertex$ ) ▷ обход из заданной вершины
10:   $visited[vertex] \leftarrow \text{true}$ 
11:   $F(vertex)$  ▷ вызов заданной функции
12:  for  $i \in adj[vertex]$  do
13:    if not  $visited[i]$  then
14:      visit( $i$ )
15:    end if
16:  end for
17: end procedure
```

Если принять, что выполнение функции $F(v)$ занимает $O(1)$ для любой вершины $v \in V$, сложность алгоритма $DFS(G)$ составляет $O(|V| + |E|)$, потому что процедура *visit* вызывается для каждой вершины один раз, и в данной процедуре просматривается столько вершин, сколько рёбер индидентно данной вершине.

3.4 Алгоритм проверки графа на связность

Пусть дан граф $G = (V, E)$. Необходимо проверить, является ли он связным. Для этого достаточно запустить поиск в глубину из любой вершины графа и проверить, посетил ли он все его вершины.

Алгоритм 4 Проверка графа на связность

```
1: procedure CHECK-COHERENCE( $G = (V, E)$ )
2:    $visited \leftarrow \text{new bool}[|V|]$ 
3:   visit(0) ▷ запускаем поиск в глубину из первой вершины
4:   for  $i \leftarrow 1 \dots |V| - 1$  do
5:     if not  $visited[i]$  then
6:       return false ▷ граф не является связным
7:     end if
8:   end for
9:   return true ▷ граф является связным
10: end procedure
11: procedure VISIT( $vertex$ ) ▷ обход из заданной вершины
12:   $visited[vertex] \leftarrow \text{true}$ 
13:  for  $i \in adj[vertex]$  do
14:    if not  $visited[i]$  then
15:      visit( $i$ )
16:    end if
17:  end for
18: end procedure
```

Сложность этого алгоритма также составляет $O(|V| + |E|)$, поскольку он представляет собой частный случай обхода графа в глубину.

3.5 Алгоритм проверки графа на ацикличность

Пусть дан граф $G = (V, E)$. Требуется проверить, что он не содержит циклов (если при этом граф ещё и является связным, он представляет собой дерево). Эту задачу можно также решить с помощью поиска в глубину. Будем запускать поиск в глубину, но вместо использования булевого массива *visited*, отображающего, посетили мы вершину или нет, будем красить вершины в различные цвета:

- белый (0) – ещё не посещённая вершина
- серый (1) – процедура *visit* от данной вершины запущена, но ещё не завершена
- чёрный (2) – процедура *visit* от данной вершины завершена.

Таким образом, если метод *visit* запущен от некоторой вершины v , и в процессе его выполнения поиск в глубину снова обнаруживает вершину v , это означает наличие цикла, содержащего вершину v , в данном графе.

Алгоритм 5 Проверка графа на ацикличность

```

1: procedure CHECK-ACYCLICITY( $G = (V, E)$ )
2:    $visited \leftarrow \text{new int}[|V|]$ 
3:   for  $i \leftarrow 0 \dots |V| - 1$  do
4:     if not  $visited[i]$  then
5:       if not  $visit(i)$  then
6:         return false
7:       end if
8:     end if
9:   end for
10:  return true                                     ▷ граф не содержит циклов
11: end procedure
12: procedure VISIT( $vertex$ )                           ▷ обход из заданной вершины
13:   $visited[vertex] \leftarrow 1$                        ▷ начало прохождения вершины
14:  for  $i \in adj[vertex]$  do
15:    if  $visited[i] = 0$  then                           ▷ ещё не посещённая вершина
16:      if not  $visit(i)$  then
17:        return false                                   ▷ граф содержит цикл
18:      end if
19:    else if  $visited[i] = 1$  then                       ▷ найден цикл
20:      return false
21:    end if
22:  end for
23:   $visited[vertex] \leftarrow 2$                        ▷ вершина пройдена
24:  return true
25: end procedure

```

Сложность данного алгоритма – $O(|V| + |E|)$.

3.6 Алгоритм поиска связных поддеревьев данного дерева, содержащих данную вершину

Пусть дано связное дерево $G = (V, E)$ и некоторая его вершина $rootVertex$ (будем называть её корнем, причём это не обязательно корень дерева в традиционном смысле, то есть вершина степени один). Необходимо найти все связные поддеревья $G' \subset G$, содержащие данную вершину. Будем перебирать все подмножества рёбер $E' \subset E$ (их $2^{|E|}$) данного графа и проверять граф $G' = (V, E')$ на соответствие этим двум требованиям.

Алгоритм 6 Поиск связных поддеревьев с заданной вершиной

```

1: procedure FIND-SUBTREES( $G = (V, E), rootVertex$ )
2:    $result \leftarrow$  new List<Graph> ▷ искомые поддеревья
3:    $possibleStates \leftarrow$  все булевы матрицы длины  $|E|$ 
4:   for  $state \in possibleStates$  do
5:      $E' \leftarrow E[state]$  ▷ все рёбра  $i$ , для которых  $state[i] == true$ 
6:      $G' = (V, E')$ 
7:      $adj \leftarrow$  матрица смежности  $G'$ 
8:     if  $|adj[rootVertex]| > 0$  and check-sub-coherence( $G'$ ) then
9:        $result.add(G')$ 
10:    end if
11:  end for
12:  return result
13: end procedure
14: procedure CHECK-SUB-COHERENCE( $G = (V, E)$ ) ▷ проверка связности подграфа
15:    $V' \leftarrow$  new set ▷ множество вершин с хотя бы одним инцидентным рёбром
16:   for  $e \in E$  do
17:      $V'.add(e.From)$ 
18:      $V'.add(e.To)$ 
19:   end for
20:    $visited \leftarrow$  new bool[ $|V|$ ]
21:    $first \leftarrow$  первый элемент из  $V'$ 
22:   visit( $first$ )
23:   for  $v \in V'$  do
24:     if not  $visited[v]$  then
25:       return false ▷ подграф не является связным
26:     end if
27:   end for
28:   return true ▷ подграф является связным
29: end procedure

```

Данный алгоритм перебирает $2^{|E|}$ булевых матриц. Чтобы составить граф (V, E') из всех рёбер, для которых соответствующие элементы некоторой такой матрицы истинны, необходимо время $O(|E|)$. Чтобы проверить такой подграф на связность, требуется время $O(|V| + |E'|) = O(|V| + |E|) = O(|E|)$ (потому что для дерева $|V| = |E| + 1$). Таким образом, сложность алгоритма получается $O(|E|2^{|E|})$ (предполагается, что для реализации множества V' используется битовый массив длины $|V|$, тогда вставка во множество занимает $O(1)$, а перечисление всех элементов множества – $O(|V|)$).

3.7 Алгоритм поиска кратчайших путей от некоторой вершины до всех в дереве

Пусть дано связное дерево $G = (V, E)$ и некоторая его вершина $rootVertex$. Требуется найти расстояния от данной вершины до всех вершин в данном графе (заметим, что эти расстояния единственны, поскольку G – дерево) и пути (последовательности рёбер) до этих вершин. Запустим поиск в глубину из $rootVertex$, присвоив для неё «найденное» нулевое расстояние и пустой путь. При переходе по некоторому ребру будем присваивать конечной его вершине расстояние, равное расстоянию до начальной вершины плюс длина ребра, и путь, образованный присоединением данного ребра к пути до начальной вершины.

Алгоритм 7 Поиск путей до всех вершин в связном дереве

```
1: procedure FIND-PATHS( $G = (V, E), rootVertex$ )
2:    $paths \leftarrow \text{new List<Edge>}[|V|]$  ▷ массив списков рёбер
3:    $paths[rootVertex] \leftarrow$  пустой список
4:    $paths\text{-}visit(G, rootVertex)$  ▷ запускаем рекурсивный обход
5:   return  $paths$  ▷ будем возвращать пути, расстояния легко найти как сумму длин всех рёбер из пути
6: end procedure
7: procedure PATHS-VISIT( $G = (V, E), startVertex$ )
8:   for  $v \in adj[startVertex]$  do
9:      $paths[v] \leftarrow copy(paths[startVertex])$  ▷ копия списка
10:     $e \leftarrow$  ребро между  $startVertex$  и  $v$ 
11:     $paths[v].add(e)$ 
12:   end for
13: end procedure
```

4 Описание экспериментов, анализ результатов

4.1 Описание экспериментов

4.1.1 Подсчёт числа точек в симплексе

Была написана программа, реализующая алгоритм 2 и позволяющая находить точные значения функции числа точек $N_k(\lambda|w_1, \dots, w_k)$ для произвольных w, k и λ . Программа написана на языке программирования C# и позволяет оператору задавать значения коэффициентов w и диапазон параметров λ , для которых будут находиться значения вышеупомянутой функции. Программа использует принцип многопоточности для того, чтобы одновременно вычислять несколько значений $N_k(\lambda|w_1, \dots, w_k)$ для ускорения работы. Исходный код программы и инструкция по её использованию содержится в приложениях 6.2-6.3.

Для того, чтобы проверить корректность полиномиального приближения $R_k(\lambda|w_1, \dots, w_k)$ функции $N_k(\lambda|w_1, \dots, w_k)$, поступим следующим образом. Зафиксируем несколько векторов коэффициентов w и для некоторого диапазона (настолько, насколько это позволяют вычислительные мощности) значений λ рассчитаем точные значения функции. По теореме 1 разность $N_k(\lambda|w_1, \dots, w_k) - R_k(\lambda|w_1, \dots, w_k)$ имеет порядок $(\log \lambda)^k$. График (и непосредственно значения) этой разности и позволит судить о корректности данного приближения.

Для генерации многочленов $R_k(\lambda|w_1, \dots, w_k)$ в символьном виде для произвольных значений w была написана программа на языке системы компьютерной алгебры Maple. В частности, в

приложении 6.1 для некоторых значений k приведены многочлены R_k для $k = 2 \dots 5$.

4.1.2 Подсчёт числа точек на графе

Поступим аналогично для проверки корректности приближения многочленами $R_k(t|w_1, \dots, w_k)$ функции числа точек на графе $N(t)$.

Была написана программа, реализующая алгоритм 1 и позволяющая рассчитывать значения функции $N(t)$ для заданных графов и пределов аргумента t . Результатом её работы является таблица точных значений функции $N(t)$ от начального момента времени до заданного конечного времени T (или до момента, когда работа программы была прервана её оператором).

Рассмотрим несколько графов $\Gamma = (V, E)$ с линейно независимыми над \mathbb{Q} длинами рёбер и для каждого из них в некоторых диапазонах значений аргумента t рассчитаем точные значения функции N . По теореме 2 разность $N(t) - R(t)$ имеет порядок $(\log t)^{|E|-1}$. Аналогично, график (и непосредственно значения) этой разности и позволит судить о корректности данного приближения.

Как гласит теорема 2, многочлены $R_k(\lambda|w_1, \dots, w_k)$, аппроксимирующие число точек на графе, выражаются через многочлены $R_k(\lambda|w_1, \dots, w_k)$, аппроксимирующие число точек в симплексе, причём это выражение зависит от структуры рассматриваемого графа. Для того, чтобы получать многочлены для приближения числа точек на графе, была написана программа, позволяющая по заданному графу получать это приближение в символьном (для произвольных длин рёбер графа данной структуры) и численном (для конкретных значений длин) вид. Эта программа реализует алгоритмы 1 и 3 - 7 и позволяет своему оператору задавать граф в текстовом формате (см. приложение 6.2.4). Программа написана на языке C# и использует написанную ранее автором библиотеку для взаимодействия написанных на этом языке программ с системой компьютерной алгебры Maple для работы с многочленами в символьном виде.

4.2 Анализ результатов

4.2.1 Подсчёт числа точек в симплексе

Эксперимент 1

Рассмотрим $w = (\sqrt{2}, \sqrt{3}, \sqrt{5})^T$. Эти положительные числа являются линейно независимыми над \mathbb{Q} , поэтому для них применима теорема 1. Найдём значения $N(t)$ для t от 1 до 6000 для этого набора весов w .

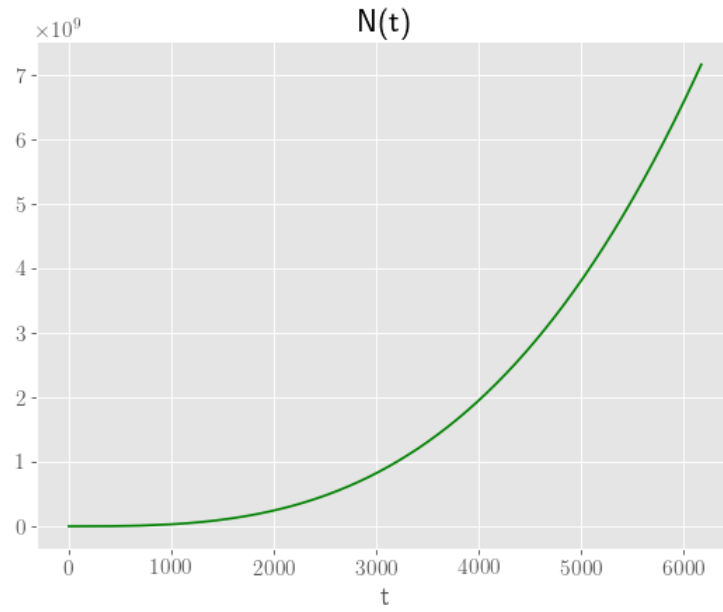


Рис. 1: число точек в симплексе для $w = (\sqrt{2}, \sqrt{3}, \sqrt{5})^T$

Рассмотрим разность функции $N(t)$ с её полиномиальным приближением:

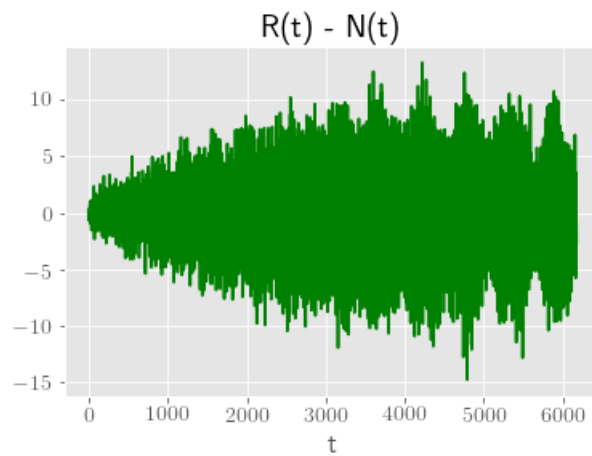


Рис. 2: $R(t) - N(t)$ для $w = (\sqrt{2}, \sqrt{3}, \sqrt{5})^T$

Можно заметить, что эта разность напоминает некоторую степень логарифма. Разделим разность на $\ln^3(t)$:

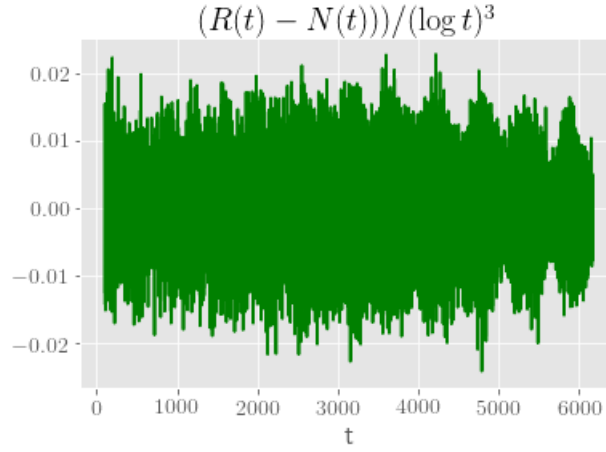


Рис. 3: $(R(t) - N(t))/(\log t)^3$ для $w = (\sqrt{2}, \sqrt{3}, \sqrt{5})^T$

$C = \max \frac{|R(t) - N(t)|}{\ln^3(t)} \approx 0.02427646$ для данного промежутка значений t . Это означает, что для данного диапазона значений t $|R(t) - N(t)| \leq C \cdot \ln^3(t)$.

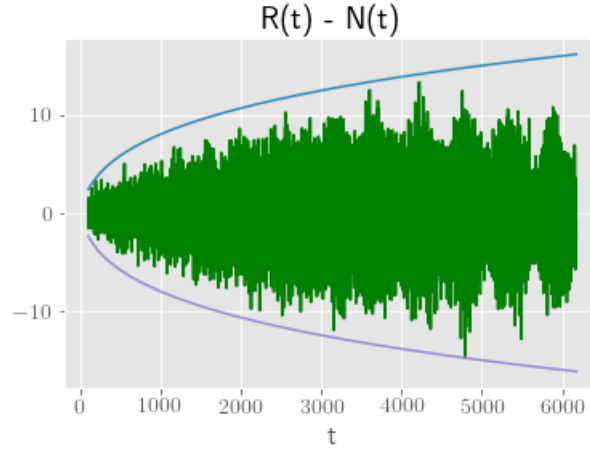


Рис. 4: Ограничение разности степенью логарифма для $w = (\sqrt{2}, \sqrt{3}, \sqrt{5})^T$

Среднее отклонение разности от $C \cdot \ln^3(t)$ составляет 8.805.

Средняя абсолютная ошибка (MAE) при приближении $N(t)$ многочленом составляет 3.005, а стандартное отклонение равно 3.871.

Как можно заметить как минимум из графика разности и значений MAE и MSE, при истинных значениях функции порядка 10^9 полиномиальное приближение имеет абсолютную ошибку (то есть модуль разности $R(t) - N(t)$) в районе 7 – 10 единиц (а в среднем на интервале эта ошибка и вовсе составляет 3 единицы), что является достаточно неплохим показателем, особенно учитывая разницу во времени вычисления функции и её приближения многочленами: значение многочлена считается практически мгновенно, а истинное значение $N(t)$ можно найти лишь перебором за достаточно большое время.

Эксперимент 2

Рассмотрим $w = (\ln 3, \ln 7, \ln 11, \ln 13)^T$. Эти положительные числа также являются линейно независимыми над \mathbb{Q} ⁴, поэтому для них применима теорема 1.

Найдём значения $N(t)$ для t от 1 до 1500 для этого набора весов w .

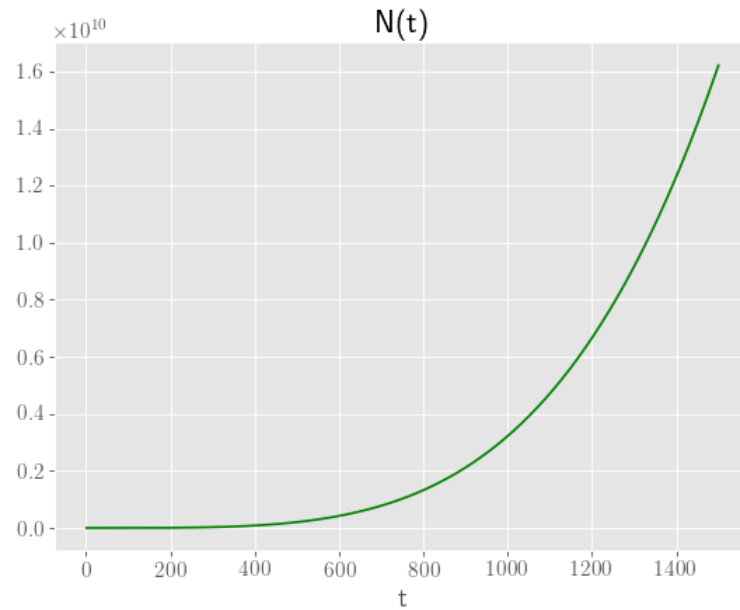


Рис. 5: число точек в симплексе для $w = (\ln 3, \ln 7, \ln 11, \ln 13)^T$

Рассмотрим разность функции $N(t)$ с её полиномиальным приближением:

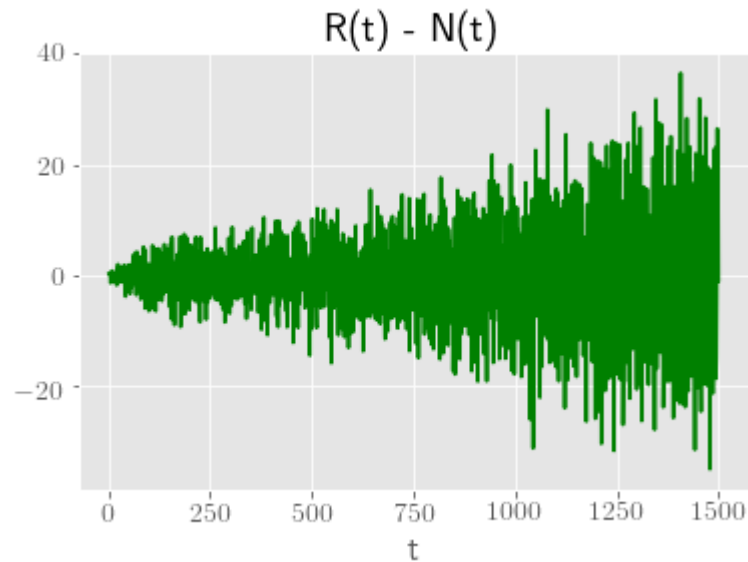


Рис. 6: $R(t) - N(t)$ для $w = (\ln 3, \ln 7, \ln 11, \ln 13)^T$

Разделим разность на $\ln^4(t)$:

⁴Это утверждение является следствием основной теоремы арифметики.

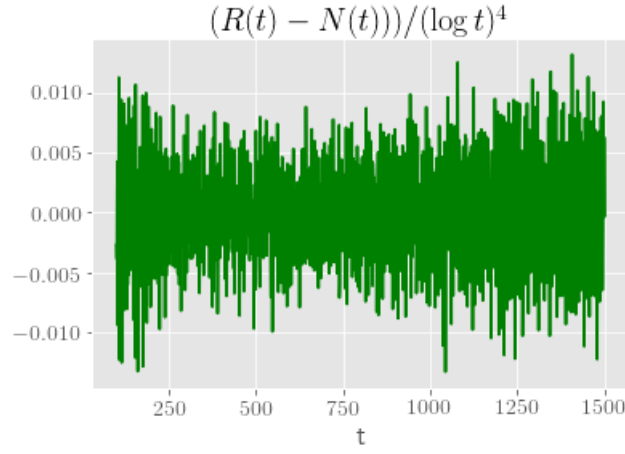


Рис. 7: $(R(t) - N(t))/(\log t)^4$ для $w = (\ln 3, \ln 7, \ln 11, \ln 13)^T$

$C = \max \frac{|R(t) - N(t)|}{\ln^4(t)} \approx 0.01334$ для данного промежутка значений t . Это означает, что для данного диапазона значений t $|R(t) - N(t)| \leq C \cdot \ln^4(t)$

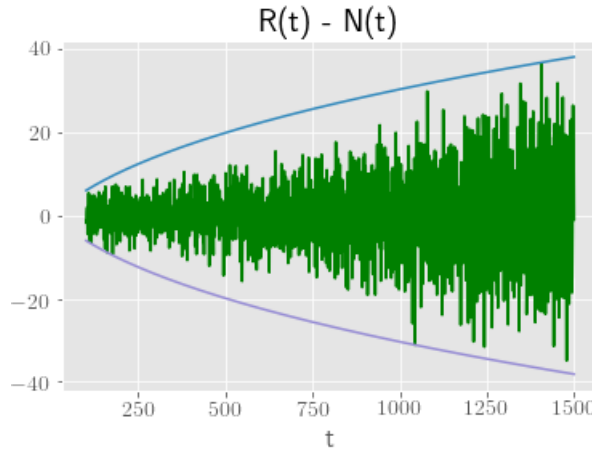


Рис. 8: Ограничение разности степенью логарифма для $w = (\ln 3, \ln 7, \ln 11, \ln 13)^T$

Среднее отклонение разности от $C \cdot \ln^4(t)$ составляет 18.566.

Средняя абсолютная ошибка (MAE) при приближении $N(t)$ многочленом составляет 6.411, а стандартное отклонение равно 8.8835.

Как можно заметить как минимум из графика разности и значений MAE и MSE, при истинных значениях функции порядка 10^{10} полиномиальное приближение имеет абсолютную ошибку (то есть модуль разности $R(t) - N(t)$) в районе 20 – 40 единиц, что достаточно неплохо приближает функцию $N(t)$.

4.2.2 Подсчёт числа точек на графе

Рассмотрим звёздный граф $K_{1,3}$ с длинами рёбер $w = (\sqrt{2}, \sqrt{3}, \sqrt{5})^T$, причём первый пакет движется в направлении вершины степени 3 (то есть эта вершина является корнем дерева). Длины рёбер этого графа являются линейно независимыми над множеством рациональных чисел \mathbb{Q} , значит, многочлен из теоремы (2) должен приближать функцию числа точек на этом

графе $N(t)$ с точностью до квадрата логарифма.

Построим график функции $N(t)$ для t от 0 до 5000:

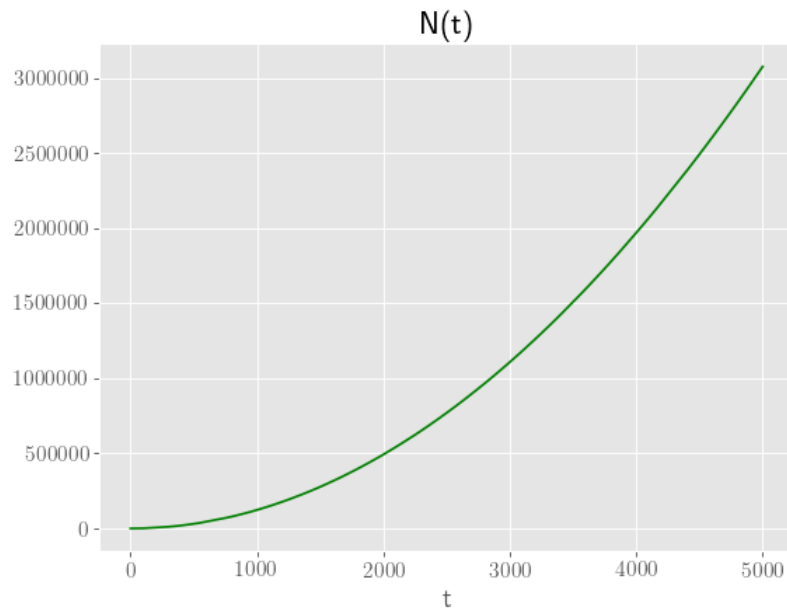


Рис. 9: число точек на звёздном графе

Рассмотрим разность истинного значения функции и её полиномиального приближения из теоремы (2):

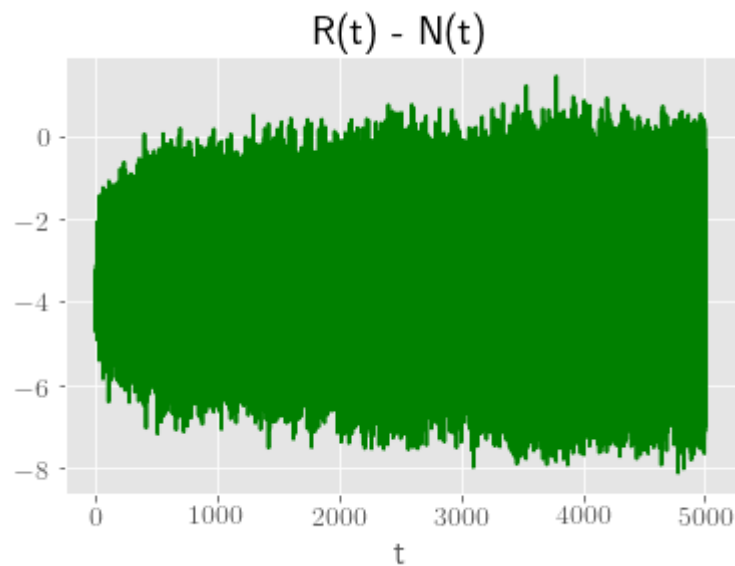


Рис. 10: $R(t) - N(t)$ для звёздного графа

Разделим данную разность, форма графика которой напоминает логарифм, на $\ln^2 t$:

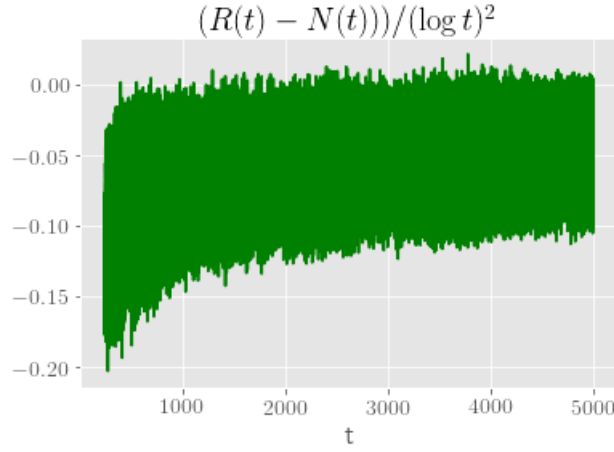


Рис. 11: $(R(t) - N(t))/(\log t)^2$ для звёздного графа

$C = \max \frac{|R(t) - N(t)|}{\ln^2(t)} \approx 0.2933$ для данного промежутка значений t . Это означает, что для данного диапазона значений t $|R(t) - N(t)| \leq C \cdot \ln^2(t)$

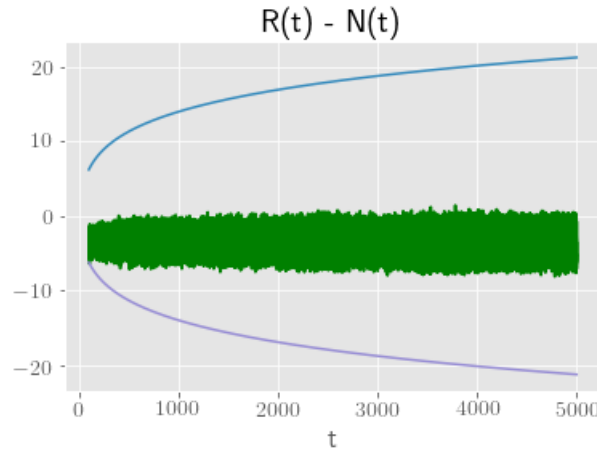


Рис. 12: Ограничение разности степенью логарифма для звёздного графа

Среднее отклонение разности от величины $C \cdot \ln^2(t)$ составляет 15.43. Это показывает то, что вторая степень логарифма с избытком приближает функцию числа точек на графе на данном интервале.

Средняя абсолютная ошибка (MAE) при приближении $N(t)$ многочленом составляет 3.5005, а стандартное отклонение равно 3.6479.

Эксперимент показал, что при истинных значениях функции порядка 10^6 полиномиальное приближение имеет абсолютную ошибку (то есть модуль разности $R(t) - N(t)$) в районе 6 – 8 единиц (а в среднем на интервале эта ошибка и вовсе составляет 3.5 единицы), что является достаточно неплохим показателем, особенно учитывая разницу во времени вычисления функции и её приближения многочленами: значение многочлена считается практически мгновенно. Истинное значение $N(t)$ для данного значения аргумента можно найти, лишь рассчитав все предыдущие значения (то есть зная распределение пакетов к этому моменту времени), что становится всё сложнее с ростом аргумента функции.

Стоит, однако, заметить, что оценку для $N(t)$, предоставляемую многочленом $R(t)$ можно улучшить в плане уменьшения среднего модуля ошибки. Рассмотрим многочлен $R(t)$ с нулевым свободным членом (обозначим такой многочлен $R_0(t)$). Среднее значение величины $R_0(t) - N(t)$ на данном интервале составляет $c_0 = 3.50046$. Возьмём эту величину в качестве оценки на свободный член ⁵ и рассмотрим многочлен $W(t) = R_0(t) + c_0$ в качестве приближающего для $N(t)$ (это можно сделать, потому что прибавление константы не изменит асимптотики, а лишь может помочь улучшить скрытую константу C). График разности «улучшенного» приближения и $N(t)$:

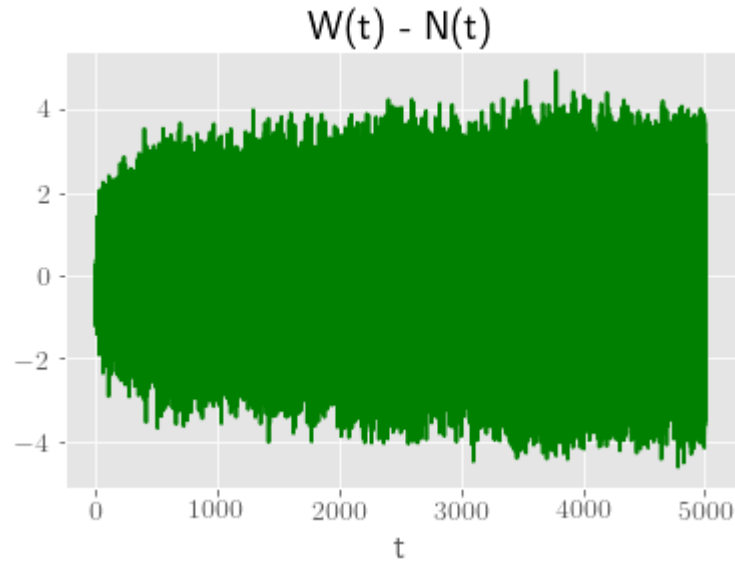


Рис. 13: $W(t) - N(t)$ для звёздного графа

Разделим данную разность на $\ln^2 t$:

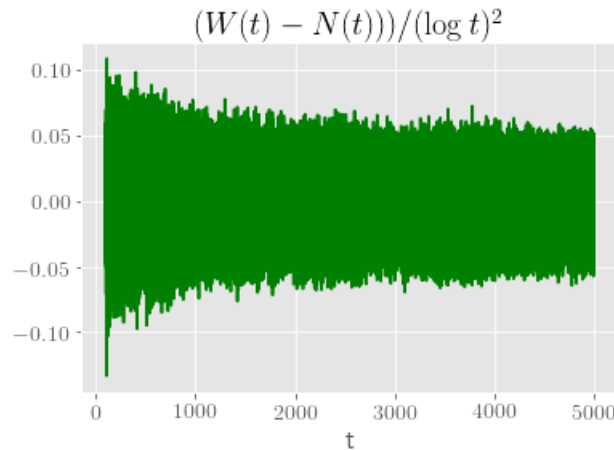


Рис. 14: $(W(t) - N(t)) / (\log t)^2$ для звёздного графа

$$C = \max \frac{|W(t) - N(t)|}{\ln^2(t)} \approx 0.13346 \text{ для данного промежутка значений } t.$$

⁵Однако, вопрос получения такой оценки для произвольного графа лишь по его характеристикам (без знания истинных значений функции) всё ещё остаётся открытым.

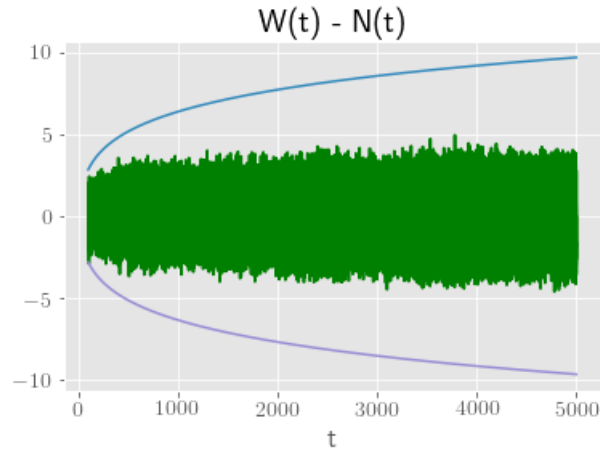


Рис. 15: Ограничение разности степеню логарифма для звёздного графа

Среднее отклонение разности от величины $C \cdot \ln^2(t)$ составляет 7.7922, что почти в два раза меньше значения, даваемого многочленом без модифицированного свободного члена. Средняя абсолютная ошибка (MAE) при приближении $N(t)$ многочленом составляет 0.8215, а стандартное отклонение равно 1.0267, что в три раза лучше, чем при использовании многочлена $R(t)$.

Эксперимент 2

Рассмотрим граф-цепочку из 5 рёбер с длинами $w = (\ln 2, \ln 7, \ln 13, \ln 17, \ln 41)^T$ (именно в таком порядке идут соответствующие рёбра в цепочке), которые являются линейно независимыми над \mathbb{Q} , причём первый пакет идёт по первому ребру длины $\ln 2$ в сторону промежуточной вершины.

Построим график функции числа точек $N(t)$ на этом графе для t от 0 до 300 ⁶:

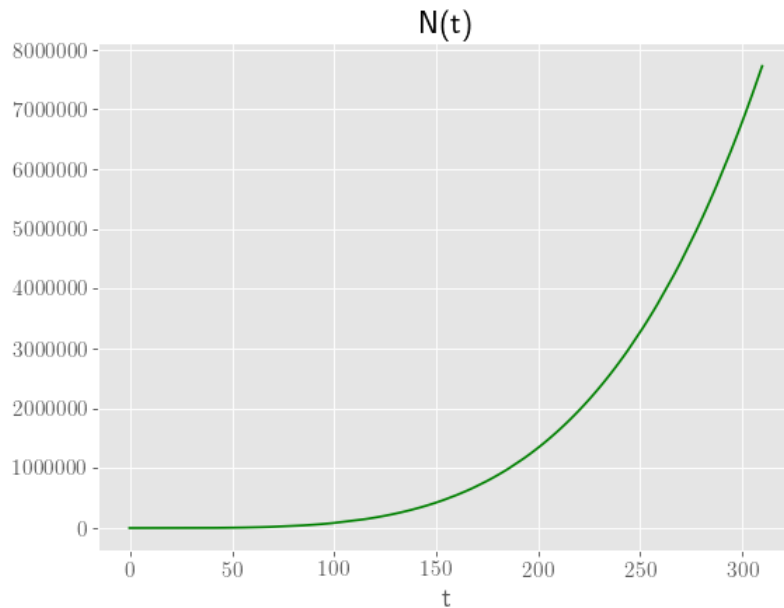


Рис. 16: число точек на графе-цепочке

⁶Несмотря на кажущуюся малость аргумента, таблица значений этой функции в данных пределах содержит в себе более 7.5 миллионов записей.

Рассмотрим разность истинного значения функции и её полиномиального приближения из теоремы (2):

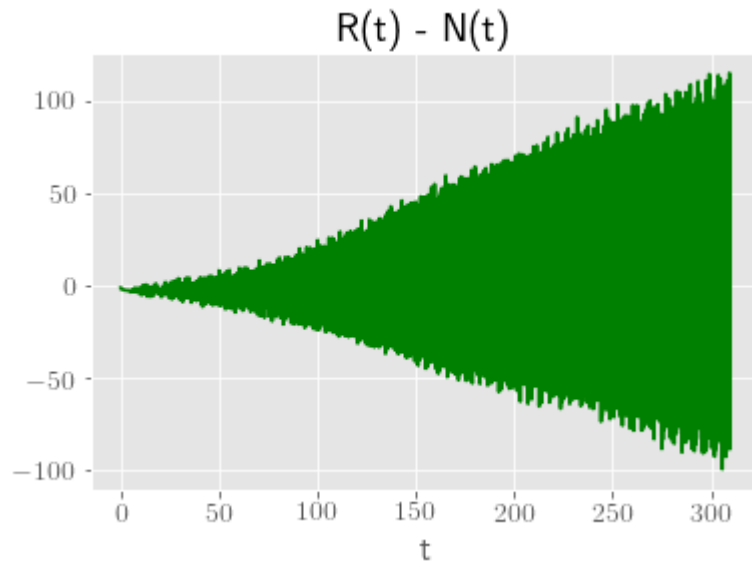


Рис. 17: $R(t) - N(t)$ для цепочки

Разделим данную разность на $\ln^4 t$:

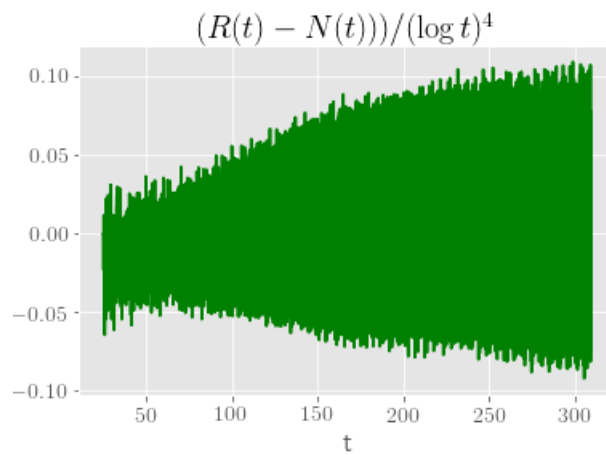


Рис. 18: $(R(t) - N(t))/(\ln t)^4$ для цепочки

$C = \max \frac{|R(t) - N(t)|}{\ln^4(t)} \approx 0.1091$ для данного промежутка значений t . Это означает, что для данного диапазона значений t $|R(t) - N(t)| \leq C \cdot \ln^4(t)$

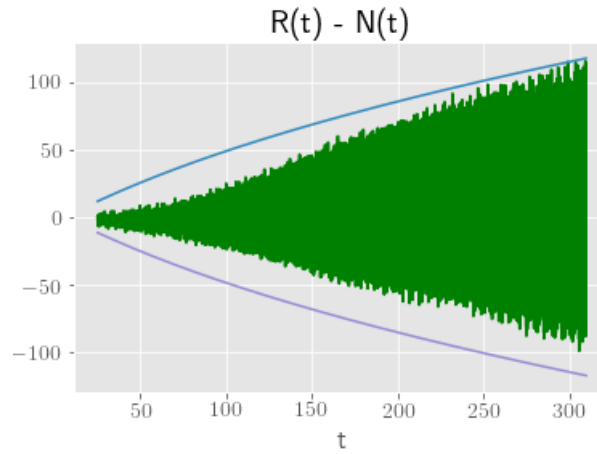


Рис. 19: Ограничение разности степенью логарифма для цепочки

Средняя абсолютная ошибка (MAE) при приближении $N(t)$ многочленом составляет 31.404, а стандартное отклонение равно 37.784. Рост данных ошибок (при, казалось бы, таких же порядках значения функции) по сравнению с предыдущими примерами обусловлен тем, что в данном примере ошибка имеет порядок четвертой степени логарифма.

На данном примере прекрасно прослеживается достоинство разработанного программного средства, позволяющего получать приближения многочленами функции числа точек на графе: для большого числа рёбер вычисление истинного числа точек требует большого объёма времени и вычислительных ресурсов (в частности, оперативной памяти для хранения всех пакетов).

5 Заключение

Проведённые эксперименты показали, что многочлены из теорем 1 и 2 действительно очень эффективно и точно приближают возникающую на графах функцию числа точек и возникающую при её рассмотрении функцию числа точек в расширяющихся многогранниках. Вычисление истинных значений данных функций является вычислительно трудной задачей, и поиск простых приближений для них очень актуально.

Основная ценность работы заключается в том, что она позволила экспериментально подтвердить теорему о полиномиальном приближении функции числа точек на графе. Разработанные в процессе исследования программные средства позволяют получать такие приближения для произвольных графов и вычислять точные значения функций этих двух видов.

Актуальность работы и её потенциальная практическая значимость обусловлена растущим интересом к функциям, возникающих при рассмотрении динамических систем на графах и исследовании поведения систем узких волновых пакетов, развивающихся на таких графах.

В качестве некоторых направлений дальнейших исследований можно выделить рассмотрение свойств полиномиальных приближений, в частности, свойства корней данных многочленов, а также модификация свободного члена для полиномиального приближения числа точек на графе для уменьшения модуля ошибки.

Список использованных источников

- [1] Арнольд В. И. Экспериментальное наблюдение математических фактов. — М.: МЦНМО, 2006. — 120 с.
- [2] Chernyshev V. L., Tolchennikov A. A., Shafarevich A. I., Behavior of Quasi-particles on Hybrid Spaces. Relations to the Geometry of Geodesics and to the Problems of Analytic Number Theory, Regular and Chaotic Dynamics, 2016, vol. 21, no. 5, pp. 531-537.
- [3] Borda B., Lattice points in algebraic cross-polytopes and simplices. Working papers by Series math-ph "arxiv.org"/2016/08. arXiv:1608.02417 [math.NT], 2016. 27 P.
- [4] Spencer D. C., The Lattice Points of Tetrahedra, Journal of Mathematics and Physics, 21, 1942. doi: 10.1002/sapm1942211189
- [5] Lehmer, D.H. The lattice points of an n-dimensional tetrahedron. Duke Math. J. 7, no. 1, 341-353. 1940. doi:10.1215/S0012-7094-40-00719-0.
- [6] Nørlund N. E., Vorlesungen über Differenzenrechnung. Berlin: Springer-Verlag. 1924.
- [7] Barnes, E. W., On the theory of the multiple gamma function, Trans. Cambridge Philos. Soc., 19, 1904. 374-425.
- [8] C5 generic collection library for C#/.NET [Электронный ресурс]. Режим доступа: <https://github.com/sestoft/C5>, свободный. (дата обращения: 07.05.17)
- [9] ГОСТ 7.32-2001 Отчет о научно-исследовательской работе. Структура и правила оформления — М.: ИПК Издательство стандартов, 2001.
- [10] MaplePortal. [Электронный ресурс] Режим доступа: <http://www.maplesoft.com/support/help/Maple/view.aspx?path=MaplePortal>, свободный. (дата обращения: 07.05.17)
- [11] C# Reference. [Электронный ресурс]. Режим доступа: <https://docs.microsoft.com/en-us/dotnet/articles/csharp/language-reference/index>, свободный. (дата обращения: 07.05.17)

6 Приложения

6.1 Многочлены $B_k^{(k)}$ и многочлены для числа точек на графах-примерах

Многочлены $B_k^{(k)}(x)$ определены соотношениями в теореме 1.

Приведём многочлены $B_k^{(k)}(x + \sum_{i=1}^k w_i)$ для $k = 2 \dots 5$:

$$B_2^{(2)}(x + \sum_{i=1}^2 w_i) = x^2 + (-w_1 - w_2)x + 1/6 w_2^2 + 1/2 w_1 w_2 + 1/6 w_1^2$$

$$B_3^{(3)}(x + \sum_{i=1}^3 w_i) = x^3 + (-3/2 w_3 - 3/2 w_2 - 3/2 w_1)x^2 + (1/2 w_3^2 + 3/2 w_2 w_3 + 1/2 w_2^2 + 3/2 w_1 w_3 + 3/2 w_1 w_2 + 1/2 w_1^2)x - 1/4 w_2 w_3^2 - 1/4 w_2^2 w_3 - 1/4 w_1 w_3^2 - 3/4 w_1 w_2 w_3 - 1/4 w_1 w_2^2 - 1/4 w_1^2 w_3 - 1/4 w_1^2 w_2$$

$$B_4^{(4)}(x + \sum_{i=1}^4 w_i) = x^4 + (-2 w_1 - 2 w_2 - 2 w_3 - 2 w_4)x^3 + (w_1^2 + 3 w_1 w_2 + 3 w_1 w_3 + 3 w_1 w_4 + w_2^2 + 3 w_2 w_3 + 3 w_2 w_4)x^2 + (w_3^2 + 3 w_3 w_4 + w_4^2)x^2 + (-w_1^2 w_2 - w_1^2 w_3 - w_1^2 w_4 - w_1 w_2^2 - 3 w_1 w_2 w_3 - 3 w_1 w_2 w_4)x + (-w_1 w_3^2 - 3 w_1 w_3 w_4 - w_1 w_4^2 - w_2^2 w_3 - w_2^2 w_4 - w_2 w_3^2 - 3 w_2 w_3 w_4 - w_2 w_4^2 - w_3^2 w_4 - w_3 w_4^2)x - 1/30 w_1^4 + 1/6 w_1^2 w_2^2 + 1/2 w_1^2 w_2 w_3 + 1/2 w_1^2 w_2 w_4 + 1/6 w_1^2 w_3^2 + 1/2 w_1^2 w_3 w_4 + 1/6 w_1^2 w_4^2 + 1/2 w_1 w_2^2 w_3 + 1/2 w_1 w_2^2 w_4 + 1/2 w_1 w_2 w_3^2 + 3/2 w_1 w_2 w_3 w_4 + 1/2 w_1 w_2 w_4^2 + 1/2 w_1 w_3^2 w_4 + 1/2 w_1 w_3 w_4^2 - 1/30 w_2^4 + 1/6 w_2^2 w_3^2 + 1/2 w_2^2 w_3 w_4 + 1/6 w_2^2 w_4^2 + 1/2 w_2 w_3^2 w_4 + 1/2 w_2 w_3 w_4^2 - 1/30 w_3^4 + 1/6 w_3^2 w_4^2 - 1/30 w_4^4$$

$$B_5^{(5)}(x + \sum_{i=1}^5 w_i) = x^5 + (-5/2 w_1 - 5/2 w_2 - 5/2 w_3 - 5/2 w_4 - 5/2 w_5)x^4 + (5/3 w_5^2 + 5 w_4 w_5 + 5/3 w_4^2 + 5 w_3 w_5 + 5 w_3 w_4 + 5/3 w_3^2)x^3 + (5 w_2 w_5 + 5 w_2 w_4 + 5 w_2 w_3 + 5/3 w_2^2 + 5 w_1 w_5 + 5 w_1 w_4 + 5 w_1 w_3 + 5 w_1 w_2 + 5/3 w_1^2)x^2 + (-5/2 w_3 w_4^2 - 5/2 w_3^2 w_4 - 5/2 w_2 w_4^2 - 5/2 w_2^2 w_4 - 5/2 w_1 w_4^2 - 5/2 w_1^2 w_4)x^2 + (-15/2 w_2 w_3 w_4 - 15/2 w_1 w_3 w_4 - 15/2 w_1 w_2 w_4 - 5/2 w_4 w_5^2 - 5/2 w_4^2 w_5)x^2 + (-5/2 w_3 w_5^2 - 5/2 w_3^2 w_5 - 5/2 w_2 w_5^2 - 5/2 w_2^2 w_5 - 5/2 w_1 w_5^2 - 5/2 w_1^2 w_5)x^2 + (-5/2 w_2 w_3^2 - 5/2 w_2^2 w_3 - 5/2 w_1 w_3^2 - 5/2 w_1 w_2^2 - 5/2 w_1^2 w_3 - 5/2 w_1^2 w_2 - 15/2 w_1 w_2 w_3 - 15/2 w_3 w_4 w_5 - 15/2 w_2 w_4 w_5 - 15/2 w_2 w_3 w_5 - 15/2 w_1 w_4 w_5 - 15/2 w_1 w_3 w_5 - 15/2 w_1 w_2 w_5)x^2 + (15/2 w_1 w_2 w_3 w_4 + 15/2 w_2 w_3 w_4 w_5 + 15/2 w_1 w_3 w_4 w_5 + 15/2 w_1 w_2 w_4 w_5 + 15/2 w_1 w_2 w_3 w_5 + 5/6 w_1^2 w_4^2 + 5/6 w_1^2 w_3^2 + 5/6 w_1^2 w_2^2 + 5/6 w_3^2 w_4^2 + 5/6 w_2^2 w_4^2 + 5/6 w_2^2 w_3^2 + 5/2 w_2 w_3 w_4^2 + 5/2 w_2 w_3^2 w_4 + 5/2 w_2^2 w_3 w_4 + 5/2 w_1 w_3 w_4^2 + 5/2 w_1 w_3^2 w_4 + 5/2 w_1 w_2 w_4^2 + 5/2 w_1 w_2 w_3^2 + 5/2 w_1 w_2^2 w_4 + 5/2 w_1 w_2^2 w_3 + 5/2 w_1^2 w_3 w_4 + 5/2 w_1^2 w_2 w_4 + 5/2 w_1^2 w_2 w_3 + 5/6 w_4^2 w_5^2 + 5/6 w_3^2 w_5^2 + 5/6 w_2^2 w_5^2 + 5/6 w_1^2 w_5^2 + 5/2 w_2 w_3^2 w_5 + 5/2 w_2^2 w_4 w_5 + 5/2 w_2^2 w_3 w_5 + 5/2 w_1 w_4 w_5^2 + 5/2 w_1 w_4^2 w_5 + 5/2 w_1 w_3 w_5^2 + 5/2 w_1 w_3^2 w_5 + 5/2 w_1 w_2 w_5^2 + 5/2 w_1 w_2^2 w_5 + 5/2 w_1^2 w_4 w_5 + 5/2 w_1^2 w_3 w_5 + 5/2 w_1^2 w_2 w_5 + 5/2 w_3 w_4 w_5^2 + 5/2 w_3 w_4^2 w_5 + 5/2 w_3^2 w_4 w_5 + 5/2 w_2 w_4 w_5^2 + 5/2 w_2 w_4^2 w_5 + 5/2 w_2 w_3 w_5^2 - 1/6 w_5^4 - 1/6 w_1^4 - 1/6 w_2^4 - 1/6 w_3^4 - 1/6 w_4^4)x - 5/4 w_2 w_3 w_4 w_5^2 - 5/4 w_2 w_3 w_4^2 w_5 - 5/4 w_2 w_3^2 w_4 w_5 - 5/4 w_2^2 w_3 w_4 w_5 - 5/4 w_1 w_3 w_4 w_5^2 - 5/4 w_1 w_3 w_4^2 w_5 - 5/4 w_1 w_3^2 w_4 w_5 - 5/4 w_1 w_2 w_4 w_5^2 - 5/4 w_1 w_2 w_4^2 w_5 - 5/4 w_1 w_2 w_3 w_5^2 - 5/4 w_1 w_2 w_3 w_4^2 - 5/4 w_1 w_2 w_3^2 w_5 - 5/4 w_1 w_2 w_3^2 w_4 - 5/4 w_1 w_2^2 w_4 w_5 - 5/4 w_1 w_2^2 w_3 w_5 - 5/4 w_1 w_2^2 w_3 w_4 - 5/4 w_1^2 w_3 w_4 w_5 - 5/4 w_1^2 w_2 w_4 w_5 - 5/4 w_1^2 w_2 w_3 w_5 - 5/4 w_1^2 w_2 w_3 w_4 + 1/12 w_4 w_5^4 + 1/12 w_4^4 w_5 + 1/12 w_3 w_5^4 + 1/12 w_3 w_4^4 + 1/12 w_3^4 w_5 + 1/12 w_3^4 w_4 + 1/12 w_2 w_5^4 + 1/12 w_2 w_4^4 + 1/12 w_2 w_3^4 + 1/12 w_2^4 w_5 + 1/12 w_2^4 w_4 + 1/12 w_2^4 w_3 + 1/12 w_1 w_5^4 + 1/12 w_1 w_4^4 + 1/12 w_1 w_3^4 + 1/12 w_1 w_2^4 + 1/12 w_1^4 w_5 + 1/12 w_1^4 w_4 + 1/12 w_1^4 w_3 + 1/12 w_1^4 w_2 - \frac{5 w_3 w_4^2 w_5^2}{12} - \frac{5 w_3^2 w_4 w_5^2}{12} - \frac{5 w_3^2 w_4^2 w_5}{12} - \frac{5 w_2 w_4^2 w_5^2}{12} - \frac{5 w_2 w_3^2 w_5^2}{12} - \frac{5 w_2 w_3^2 w_4^2}{12} - \frac{5 w_2^2 w_4^2 w_5^2}{12} - \frac{5 w_2^2 w_4^2 w_5}{12} - \frac{5 w_2^2 w_3^2 w_5}{12} - \frac{5 w_2^2 w_3^2 w_4}{12} - \frac{5 w_1 w_4^2 w_5^2}{12} - \frac{5 w_1 w_3^2 w_5^2}{12} - \frac{5 w_1 w_3^2 w_4^2}{12} - \frac{5 w_1 w_2^2 w_5^2}{12} - \frac{5 w_1 w_2^2 w_4^2}{12} - \frac{5 w_1 w_2^2 w_3^2}{12} - \frac{5 w_1^2 w_4^2 w_5^2}{12} - \frac{5 w_1^2 w_4^2 w_5}{12} - \frac{5 w_1^2 w_3^2 w_5^2}{12} - \frac{5 w_1^2 w_3^2 w_4^2}{12} - \frac{5 w_1^2 w_2^2 w_5^2}{12} - \frac{5 w_1^2 w_2^2 w_4^2}{12} - \frac{5 w_1^2 w_2^2 w_3^2}{12} - \frac{5 w_1^2 w_2^2 w_5}{12} - \frac{5 w_1^2 w_2^2 w_4}{12} - \frac{5 w_1^2 w_2^2 w_3}{12} - \frac{15 w_1 w_2 w_3 w_4 w_5}{4}$$

Многочлен, аппроксимирующий число точек на звёздном графе $K_{1,3}$ (используются обозна-

чения $u_i = 1/w_i$):

$$R(t) = \frac{1}{24} \frac{((3u_2^2u_3 + 3u_2u_3^2)u_1^2 + 3u_1u_2^2u_3^2)t^2}{u_2u_3u_1} + \frac{1}{24} \frac{(12u_2u_3u_1^2 + (12u_2^2u_3 + 12u_2u_3^2)u_1)t}{u_2u_3u_1} + \frac{1}{24} \frac{(2u_2 + 2u_3)u_1^2 + (2u_2^2 - 54u_2u_3 + 2u_3^2)u_1 + 2(u_3 + u_2)u_2u_3}{u_2u_3u_1}$$

Многочлен, аппроксимирующий число точек на цепочке из четырёх рёбер (используются обозначения $u_i = 1/w_i$):

$$\begin{aligned} R(t) = & (5760u_3^4u_4^4u_2^4u_1^3u_5^3)^{-1}(15u_1^4u_2^5u_3^5u_4^5u_5^3 + 15u_1^4u_2^5u_3^5u_4^4u_5^4 + \\ & + 15u_1^4u_2^5u_3^4u_4^5u_5^4 + 15u_1^4u_2^4u_3^5u_4^5u_5^4 + 15u_1^3u_2^5u_3^5u_4^5u_5^4)t^4 + \\ & + (5760u_3^4u_4^4u_2^4u_1^3u_5^3)^{-1}(60u_1^4u_2^5u_3^5u_4^4u_5^3 - 60u_1^4u_2^5u_3^5u_4^3u_5^4 + \\ & + 60u_1^4u_2^5u_3^4u_4^5u_5^3 - 60u_1^4u_2^5u_3^4u_4^4u_5^4 - 60u_1^4u_2^5u_3^3u_4^5u_5^4 + 60u_1^4u_2^4u_3^5u_4^5u_5^3 - \\ & - 60u_1^4u_2^4u_3^5u_4^4u_5^4 - 60u_1^4u_2^4u_3^4u_4^5u_5^4 - 60u_1^4u_2^3u_3^5u_4^5u_5^4 + 120u_1^3u_2^5u_3^5u_4^5u_5^3)t^3 + \\ & + (5760u_3^4u_4^4u_2^4u_1^3u_5^3)^{-1}(60u_1^4u_2^5u_3^5u_4^4u_5^2 + 150u_1^4u_2^5u_3^5u_4^3u_5^3 + 90u_1^4u_2^5u_3^5u_4^2u_5^4 + \\ & + 60u_1^4u_2^5u_3^4u_4^5u_5^2 + 540u_1^4u_2^5u_3^4u_4^4u_5^3 + \\ & + 60u_1^4u_2^5u_3^4u_4^3u_5^4 - 210u_1^4u_2^5u_3^3u_4^5u_5^3 + \\ & + 150u_1^4u_2^5u_3^3u_4^4u_5^4 + 90u_1^4u_2^5u_3^2u_4^5u_5^4 + 60u_1^4u_2^4u_3^5u_4^5u_5^2 + 540u_1^4u_2^4u_3^5u_4^4u_5^3 + \\ & + 60u_1^4u_2^4u_3^5u_4^3u_5^4 - 180u_1^4u_2^4u_3^4u_4^5u_5^3 + 180u_1^4u_2^4u_3^4u_4^4u_5^4 + 60u_1^4u_2^4u_3^3u_4^5u_5^4 - \\ & - 210u_1^4u_2^3u_3^5u_4^5u_5^3 + \\ & + 150u_1^4u_2^3u_3^5u_4^4u_5^4 + 150u_1^4u_2^3u_3^4u_4^5u_5^4 + 90u_1^4u_2^2u_3^5u_4^5u_5^4 + 60u_1^3u_2^5u_3^5u_4^5u_5^2 + \\ & + 720u_1^3u_2^5u_3^5u_4^4u_5^3 - 120u_1^3u_2^5u_3^5u_4^3u_5^4 - 120u_1^3u_2^5u_3^3u_4^5u_5^4 - 120u_1^3u_2^3u_3^5u_4^5u_5^4 + \\ & + 60u_1^2u_2^5u_3^5u_4^5u_5^3 + 60u_1^2u_2^5u_3^5u_4^4u_5^4 + 60u_1^2u_2^5u_3^4u_4^5u_5^4 + 60u_1^2u_2^4u_3^5u_4^5u_5^4)t^2 + \\ & + (5760u_3^4u_4^4u_2^4u_1^3u_5^3)^{-1}(-120u_1^4u_2^5u_3^5u_4^3u_5^2 - 180u_1^4u_2^5u_3^5u_4^2u_5^3 - 60u_1^4u_2^5u_3^5u_4u_5^4 - \\ & - 120u_1^4u_2^5u_3^4u_4^5u_5^2 + 120u_1^4u_2^5u_3^4u_4^3u_5^3 - 120u_1^4u_2^5u_3^3u_4^5u_5^2 + 300u_1^4u_2^5u_3^3u_4^4u_5^3 - \\ & - 60u_1^4u_2^5u_3^3u_4^3u_5^4 + 180u_1^4u_2^5u_3^2u_4^5u_5^3 - 180u_1^4u_2^5u_3^2u_4^4u_5^4 - 60u_1^4u_2^5u_3u_4^5u_5^4 - \\ & - 120u_1^4u_2^4u_3^5u_4^4u_5^2 + 120u_1^4u_2^4u_3^5u_4^3u_5^3 - 120u_1^4u_2^4u_3^4u_4^5u_5^2 + 1800u_1^4u_2^4u_3^4u_4^4u_5^3 - \\ & - 120u_1^4u_2^4u_3^4u_4^3u_5^4 + 120u_1^4u_2^4u_3^3u_4^5u_5^3 - 120u_1^4u_2^4u_3^3u_4^4u_5^4 - 120u_1^4u_2^3u_3^5u_4^5u_5^2 - \\ & - 1140u_1^4u_2^3u_3^5u_4^4u_5^3 - 60u_1^4u_2^3u_3^5u_4^3u_5^4 + 300u_1^4u_2^3u_3^4u_4^5u_5^3 - 300u_1^4u_2^3u_3^4u_4^4u_5^4 - \\ & - 60u_1^4u_2^3u_3^3u_4^5u_5^4 + 180u_1^4u_2^2u_3^5u_4^5u_5^3 - 180u_1^4u_2^2u_3^5u_4^4u_5^4 - 180u_1^4u_2^2u_3^4u_4^5u_5^4 - \\ & - 60u_1^4u_2u_3^5u_4^5u_5^4 + 420u_1^3u_2^5u_3^5u_4^3u_5^3 + 180u_1^3u_2^5u_3^5u_4^2u_5^4 + 2880u_1^3u_2^5u_3^4u_4^4u_5^3 - \\ & - 300u_1^3u_2^5u_3^3u_4^5u_5^3 + 180u_1^3u_2^5u_3^3u_4^4u_5^4 + 180u_1^3u_2^5u_3^2u_4^5u_5^4 - 300u_1^3u_2^3u_3^5u_4^5u_5^3 + \\ & + 180u_1^3u_2^3u_3^5u_4^4u_5^4 + 180u_1^3u_2^3u_3^4u_4^5u_5^4 + 180u_1^3u_2^2u_3^5u_4^5u_5^4 + 120u_1^2u_2^5u_3^5u_4^4u_5^3 - \\ & - 120u_1^2u_2^5u_3^5u_4^3u_5^4 + 120u_1^2u_2^5u_3^4u_4^5u_5^3 - 120u_1^2u_2^5u_3^4u_4^4u_5^4 - 120u_1^2u_2^5u_3^3u_4^5u_5^4 + \\ & + 120u_1^2u_2^4u_3^5u_4^5u_5^3 - 120u_1^2u_2^4u_3^5u_4^4u_5^4 - 120u_1^2u_2^4u_3^4u_4^5u_5^4 - 120u_1^2u_2^3u_3^5u_4^5u_5^4)t + \\ & + (5760u_3^4u_4^4u_2^4u_1^3u_5^3)^{-1}(-8u_1^4u_2^5u_3^5u_4^4u_5^2 + 60u_1^4u_2^5u_3^5u_4^2u_5^2 + 67u_1^4u_2^5u_3^5u_4u_5^3 + \\ & + 15u_1^4u_2^5u_3^5u_5^4 - 8u_1^4u_2^5u_3^4u_4^5u_5^2 + 40u_1^4u_2^5u_3^4u_4^3u_5^2 - 8u_1^4u_2^5u_3^4u_4u_5^4 + \\ & + 100u_1^4u_2^5u_3^3u_4^4u_5^2 - 170u_1^4u_2^5u_3^3u_4^3u_5^3 - 30u_1^4u_2^5u_3^3u_4^2u_5^4 + 60u_1^4u_2^5u_3^2u_4^5u_5^2 - \\ & - 180u_1^4u_2^5u_3^2u_4^4u_5^3 + 60u_1^4u_2^5u_3^2u_4^3u_5^4 - 53u_1^4u_2^5u_3u_4^5u_5^3 + 67u_1^4u_2^5u_3u_4^4u_5^4 + \\ & + 15u_1^4u_2^5u_4^5u_5^4 - 8u_3^5u_4^5u_2^4u_1^4 + 40u_3^5u_4^3u_2^4u_1^4u_5^2 - 8u_3^5u_4u_2^4u_1^4u_5^4 + \\ & + 120u_3^4u_4^4u_2^4u_1^4u_5^2 - 120u_3^4u_4^3u_2^4u_1^4u_5^3 + 40u_3^3u_4^5u_2^4u_5^2u_1^4 + 360u_3^3u_4^4u_2^4u_5^3u_1^4 + \\ & + 40u_3^3u_4^3u_2^4u_5^4u_1^4 - 8u_3u_4^5u_2^4u_1^4u_5^4 + 100u_3^5u_5^2u_4^4u_1^4u_2^3 - 170u_3^5u_5^3u_4^3u_1^4u_2^3 - \\ & - 30u_3^5u_5^4u_4^2u_1^4u_2^3 + 100u_3^4u_5^2u_4^5u_1^4u_2^3 + 900u_3^4u_5^3u_4^4u_1^4u_2^3 + 100u_3^4u_5^4u_4^3u_1^4u_2^3 - \\ & - 50u_3^3u_5^3u_4^5u_1^4u_2^3 + 70u_3^3u_5^4u_4^4u_1^4u_2^3 - 30u_3^2u_5^4u_4^5u_1^4u_2^3 + 60u_3^5u_5^2u_4^5u_1^4u_2^2 + \\ & + 540u_3^5u_5^3u_4^4u_1^4u_2^2 + 60u_3^5u_5^4u_4^3u_1^4u_2^2 - 180u_3^4u_5^3u_4^5u_1^4u_2^2 + 180u_3^4u_5^4u_4^4u_1^4u_2^2 + \\ & + 60u_3^3u_5^4u_4^5u_1^4u_2^2 - 53u_3^5u_5^3u_4^5u_1^4u_2 + 67u_3^5u_5^4u_4^4u_1^4u_2 + 67u_3^4u_5^4u_4^5u_1^4u_2 + \\ & + 15u_1^4u_3^5u_4^5u_5^4 - 8u_1^3u_2^5u_3^5u_4^5 - 80u_1^3u_2^5u_3^5u_4^3u_5^2 - 180u_1^3u_2^5u_3^5u_4^2u_5^3 - \\ & - 68u_1^3u_2^5u_3^5u_4u_5^4 - 80u_1^3u_2^5u_3^4u_4^5u_5^2 + 660u_1^3u_2^5u_3^3u_4^4u_5^3 - 20u_1^3u_2^5u_3^3u_4^3u_5^4 + \\ & + 180u_1^3u_2^5u_3^2u_4^5u_5^3 - 180u_1^3u_2^5u_3^2u_4^4u_5^4 - 68u_1^3u_2^5u_3u_4^5u_5^4 - 5760u_3^4u_4^4u_2^4u_1^3u_5^3 - \\ & - 80u_3^5u_5^2u_4^5u_1^3u_2^3 - 780u_3^5u_5^3u_4^4u_1^3u_2^3 - 20u_3^5u_5^4u_4^3u_1^3u_2^3 + 180u_3^4u_5^3u_4^5u_1^3u_2^3 - \\ & - 180u_3^4u_5^4u_4^4u_1^3u_2^3 - 20u_3^3u_5^4u_4^5u_1^3u_2^3 + 180u_3^5u_5^3u_4^5u_1^3u_2^2 - 180u_3^5u_5^4u_4^4u_1^3u_2^2 - \end{aligned}$$

$$\begin{aligned}
& - 180 u_3^4 u_5^4 u_4^5 u_1^3 u_2^2 - 68 u_3^5 u_5^4 u_4^5 u_1^3 u_2 + 40 u_1^2 u_2^5 u_3^5 u_4^4 u_5^2 + 100 u_1^2 u_2^5 u_3^5 u_4^3 u_5^3 + \\
& + 60 u_1^2 u_2^5 u_3^5 u_4^2 u_5^4 + 40 u_1^2 u_2^5 u_3^4 u_4^5 u_5^2 + 360 u_1^2 u_2^5 u_3^4 u_4^4 u_5^3 + 40 u_1^2 u_2^5 u_3^4 u_4^3 u_5^4 - \\
& - 140 u_1^2 u_2^5 u_3^3 u_4^5 u_5^3 + 100 u_1^2 u_2^5 u_3^3 u_4^4 u_5^4 + 60 u_1^2 u_2^5 u_3^2 u_4^5 u_5^4 + 40 u_3^5 u_4^5 u_2^4 u_1^2 u_5^2 + \\
& + 360 u_3^5 u_4^4 u_2^4 u_1^2 u_5^3 + 40 u_3^5 u_4^3 u_2^4 u_1^2 u_5^4 - 120 u_3^4 u_4^5 u_2^4 u_1^2 u_5^3 + 120 u_3^4 u_4^4 u_2^4 u_1^2 u_5^4 + \\
& + 40 u_3^3 u_4^5 u_2^4 u_5^4 u_1^2 - 140 u_3^5 u_5^3 u_4^5 u_1^2 u_2^3 + 100 u_3^5 u_5^4 u_4^4 u_1^2 u_2^3 + 100 u_3^4 u_5^4 u_4^5 u_1^2 u_2^3 + \\
& + 60 u_3^5 u_5^4 u_4^5 u_1^2 u_2^2 - 8 u_2^5 u_3^5 u_4^5 u_5^3 - 8 u_2^5 u_3^5 u_4^4 u_5^4 - 8 u_2^5 u_3^4 u_4^5 u_5^4 - 8 u_3^5 u_4^5 u_2^4 u_5^4)
\end{aligned}$$

6.2 Описание разработанных программ

6.2.1 Описание программы для подсчёта числа точек в симплексе

Программа для подсчёта точек в симплексе написана на языке C#. Она реализует алгоритм 2 и позволяет оператору находить (табулировать) истинные значения функции числа положительных точек в симплексе $N(t)$ для заданного вектора весов w и диапазона значений параметра t с заданным шагом.

Программа распространяется в виде единственного файла CountPoints.exe. Единственным требованием для её запуска является наличие на компьютере пользователя пакета .NET Framework версии не ниже 4.6 (и, соответственно, соответствующей совместимой версии операционной системы Microsoft Windows – минимальной совместимой версией является Windows Vista).

Для повышения производительности в программе реализованы многопоточные (параллельные) вычисления (одновременно работают несколько потоков, каждый поток подсчитывает значение функции для своего аргумента, число потоков задаётся пользователем).

Программа запускается из консоли командой следующего вида:

```
CountPoints.exe [input_file] [min_t] [max_t] [step]
[threads_number] [max_minutes_working]
```

Здесь:

- input_file – путь к файлу, содержащему значения весов t
- min_t – начальное значение аргумента t
- max_t – максимальное допустимое значение t
- step – шаг аргумента при табулировании функции
- threads_number – число одновременно работающих потоков
- max_minutes_working – максимальное время работы программы (в минутах, положительное значение). Если параметр не задан, программа будет работать до завершения всех вычислений.

Пример вызова и работы программы:

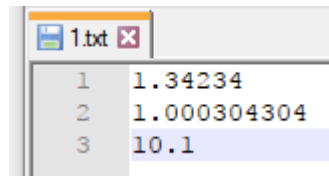
```

C:\CountPoints>CountPoints.exe 1.txt 1 10 1.79 2 10
Thread 1 started
Thread 2 started
Thread 1 started 1
Thread 2 started 2.79
Thread 1 finished 1
Thread 1 started 4.58
Thread 1 finished 4.58
Thread 1 started 6.37
Thread 1 finished 6.37
Thread 1 started 8.16
Thread 1 finished 8.16
Thread 1 started 9.95
Thread 2 finished 2.79
Thread 1 finished 9.95
Total 00:00:05 elapsed

```

Рис. 20: пример работы программы подсчёта числа точек в симплексе

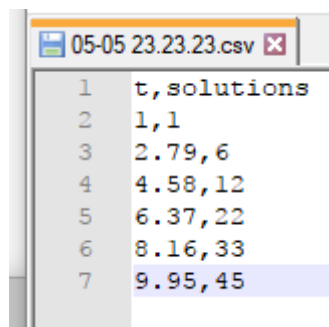
Пользователь должен передавать в программу текстовый файл, содержащий столько строк, сколько элементов в векторе весов. В этом файле каждый элемент вектора весов – это положительное действительное число (в качестве десятичного разделителя выступает точка и каждый такой элемент располагается на отдельной строке.



1	1.34234
2	1.000304304
3	10.1

Рис. 21: пример входного файла для программы подсчёта числа точек в симплексе

Результатом работы программы является текстовый csv-файл с двумя колонками – значением аргумента табулируемой функции числа точек и точным значением функции в этой точке.



	t, solutions
1	1, 1
2	2.79, 6
3	4.58, 12
4	6.37, 22
5	8.16, 33
6	9.95, 45

Рис. 22: пример результата работы программы подсчёта числа точек в симплексе

6.2.2 Описание программы для подсчёта числа точек на графе

Программа для подсчёта точек на графе написана на языке C#. Она реализует алгоритм 1 и позволяет оператору находить (табулировать) истинные значения функции числа точек на графе $N(t)$ для заданного графа (дерева) Γ и диапазона значений параметра t с заданным шагом.

Программа использует стороннюю библиотеку C5 [8], предоставляющую реализацию различных обобщённых структур данных для платформы .NET, для использования её реализации

структуры очереди с приоритетом.

Программа распространяется в виде архива, содержащего исполняемый файл GraphCount.exe и DLL-сборки библиотек C5 (сторонняя) и GraphLib (авторская, представляет структуру данных графа). Единственным требованием для её запуска является наличие на компьютере пользователя пакета .NET Framework версии не ниже 4.6 (и, соответственно, соответствующей совместимой версии операционной системы Microsoft Windows – минимальной совместимой версией является Windows Vista).

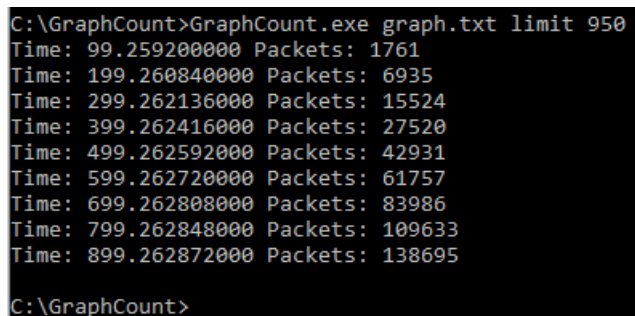
Программа запускается из консоли командой следующего вида:

```
GraphCount.exe [input_graph_path] [quiet] [tolerance time_tolerance]
[limit time_limit]
```

где

- `input_graph_path` – путь к файлу, содержащем представление графа (обязательный параметр)
- `quiet` – при установке этого параметра программа не будет отображать прогресс выполнения задачи
- `time_tolerance` – параметр ε для алгоритма моделирования движения точек на графе (не является обязательным параметром, по умолчанию используется значение $\varepsilon = 10^{-10}$).
- `time_limit` – максимальное значение аргумента функции t , для которого будет подсчитываться значение функции.

Пример вызова и работы программы:



```
C:\GraphCount>GraphCount.exe graph.txt limit 950
Time: 99.259200000 Packets: 1761
Time: 199.260840000 Packets: 6935
Time: 299.262136000 Packets: 15524
Time: 399.262416000 Packets: 27520
Time: 499.262592000 Packets: 42931
Time: 599.262720000 Packets: 61757
Time: 699.262808000 Packets: 83986
Time: 799.262848000 Packets: 109633
Time: 899.262872000 Packets: 138695
C:\GraphCount>
```

Рис. 23: пример работы программы подсчёта числа точек на графе

Пользователь должен передавать в программу текстовый файл, представляющий граф, и соответствующий формату представления графа, описанному в приложении 6.2.4.

Результатом работы программы является текстовый csv-файл, содержащий табулированные значения функции во всех точках, где это значение меняется.

t, N	
1	-0.750000000000,1
2	0.000000000000,2
3	1.123456000000,3
4	1.500000000000,4
5	2.246912000000,5
6	3.000000000000,6
7	3.370368000000,7
8	3.746912000000,8
9	4.123456000000,9
10	

Рис. 24: пример результата работы программы подсчёта числа точек на графе

6.2.3 Описание программы для генерации полиномиальных приближений для числа точек на графе

Программа для генерации многочленов, приближающих функцию числа точек на графе многочленами, написана на языке C#. Она реализует алгоритмы 3 - 7 и позволяет оператору находить полиномиальные приближения функции числа точек на графе $N(t)$ для заданного графа (дерева) Γ как в символьном (то есть для графа заданного вида с произвольными длинами рёбер), так и в числовом (для конкретных значений длин) виде.

Программа использует разработанную автором библиотеку MapleLib, которая позволяет осуществлять взаимодействие программного обеспечения на платформе .NET с системой компьютерной алгебры Maple, для удобной работы с выражениями в символьном виде.

Программа распространяется в виде архива, содержащего исполняемый файл SumGenerator.exe и DLL-сборки библиотек MapleLib (авторская) и GraphLib (авторская, представляет структуру данных графа). Требованиями для её запуска являются:

- наличие на компьютере пользователя пакета .NET Framework версии не ниже 4.6 (и, соответственно, соответствующей совместимой версии операционной системы Microsoft Windows – минимальной совместимой версией является Windows Vista).
- установленная система компьютерной алгебры Maple версии не ниже 2015

Программа запускается из консоли командой следующего вида:

```
SumGenerator.exe input_graph_path
```

где input_graph_path – единственный параметр, определяющий путь к файлу, представляющему граф. При этом в файле path.txt в той же директории, что и исполняемый файл приложения, должен быть прописан путь к исполняемому файлу консольного интерпретатора системы Maple (cmaple.exe).

```
1 C:\Program Files\Maple 2015\bin.X86_64_WINDOVS\cmaple.exe
```

Рис. 25: пример содержимого файла path.txt

Файл, представляющий граф, должен удовлетворять формату представления графа, описанному в приложении 6.2.4.

Пример вызова и работы программы:

```

C:\SumGenerator>SumGenerator graph.txt
Initializing polynoms
Polynoms initialized
(1/8/r[2]/r[3]+1/8/r[1]/r[3]+1/8/r[1]/r[2])*t^2+(1/r[3]+1/r[2]+1/8*(-2*r[2]-2*r[3])/r[2]/r[3]+1/r[1]+1/8*(-2*r[1]-2*r[3])/r[1]/r[3]+1/8*(-2*r[1]-2*r[2])/r[1]/r[2])*t-3+1/8*(2/3*r[3]^2+2*r[2]*r[3]+2/3*r[2]^2)/r[2]/r[3]+1/8*(2/3*r[3]^2+2*r[1]*r[3]+2/3*r[1]^2)/r[1]/r[3]+1/8*(2/3*r[2]^2+2*r[1]*r[2]+2/3*r[1]^2)/r[1]/r[2]
In python:
(1/8/r[2]/r[3]+1/8/r[1]/r[3]+1/8/r[1]/r[2])*t**2+(1/r[3]+1/r[2]+1/8*(-2*r[2]-2*r[3])/r[2]/r[3]+1/r[1]+1/8*(-2*r[1]-2*r[3])/r[1]/r[3]+1/8*(-2*r[1]-2*r[2])/r[1]/r[2])*t-3+1/8*(2/3*r[3]**2+2*r[2]*r[3]+2/3*r[2]**2)/r[2]/r[3]+1/8*(2/3*r[3]**2+2*r[1]*r[3]+2/3*r[1]**2)/r[1]/r[3]+1/8*(2/3*r[2]**2+2*r[1]*r[2]+2/3*r[1]**2)/r[1]/r[2]
Numeric:
-1.322778443+.7167641245*t+.7496548030e-1*t^2
In python:
-1.322778443+.7167641245*t+.7496548030e-1*t**2

```

Рис. 26: пример вызова и работы программы для генерации приближающих многочленов

Программа также сохраняет результаты своей работы в текстовый файл в директории с исполняемым файлом.

6.2.4 Описание формата представления графа в текстовом виде

Граф $G = (V, E)$ представляется в виде текстового файла. На первой строке файла находится число вершин $n = |V|$ – положительное целое число. Все остальные строки файла представляют собой описание множества рёбер E графа G . Каждая строчка представляет отдельное ребро графа и должна иметь следующий формат (все три параметра являются обязательными):

from to length

где

- from и to – вершины, между которыми проведено ребро
- length – длина ребра (положительное действительное число, в качестве десятичного разделителя выступает точка).

Поскольку данный формат представляет собой неориентированный граф, порядок, в котором указываются вершины (*from* и *to*) неважен. Единственным исключением является первое ребро – считается, что именно в его середине стоит первая точка, которая в момент времени $t = 0$ попадает в вершину *to* и перерождается в столько точек, сколько рёбер индидентно этой вершине.

Примеры валидных (соответствующих формату) описаний графов:

K13.txt			
1	4		
2	0 1	1.99	
3	1 2	7.8	
4	1 3	0.37896	

Рис. 27: звёздный граф $K_{1,3}$ с длинами рёбер 1.99, 7.8, 0.37896

1	5
2	0 1 1.41
3	1 2 3.14
4	2 3 2.71
5	3 4 10

Рис. 28: цепочка на 5 вершинах с длинами рёбер 1.41, 3.14, 2.71, 10

6.3 Исходный код разработанных программ и экспериментальные данные

Исходные коды разработанных программных средств расположены на диске, входящем в состав данной курсовой работы. Структура директорий:

- GraphCount – решение Visual Studio 2015, содержащее C#-проекты GraphCount (программа для подсчёта числа точек на графе) и SumGenerator (программа для генерации полиномиальных приближений по графу), а также набор классов, представляющих граф (GraphLib).
- CountPoints – решение Visual Studio 2015, содержащее C#-проект CountPoints (программа для подсчёта числа точек в симплексе)
- jupyter – «рабочие тетради» Jupyter Notebook (содержащие код на языке Python 3.6 с подключением библиотек NumPy и Pandas), используемые для работы с полученными csv-таблицами (для построения графиков и подсчёта численных характеристик качества приближений)
- data – csv-таблицы значений функции для приведённых в работе примеров

6.4 Используемые понятия и определения

Граф (в данной работе подразумевается неориентированный граф) – это упорядоченная пара $G = (V, E)$ из множества вершин V и множества рёбер E , где каждое ребро соединяет две различные вершины графа.

Цикл в графе – это последовательность рёбер, в которой конец i -го ребра является началом $(i + 1)$ -го, рёбра не повторяются и начало первого ребра совпадает с концом последнего.

Дерево – это граф без циклов.

Множество (set) как структура данных представляет собой конечный набор объектов различного типа без повторений и реализует методы добавления и удаления элементов, а также операции над множествами – пересечение, объединение, разность, симметрическая разность и прочие.

Очередь с приоритетом (priority queue) – это структура данных, содержащая объекты некоторого типа T , причём для таких объектов определено отношение нестрогого порядка \leq . Такая структура данных реализует методы:

- $\text{add}(T\ x)$ – добавление элемента x типа T в очередь
- $\text{top}()$ – возвращает минимальный элемент очереди

- `pop()` – возвращает минимальный элемент очереди и извлекает его из неё

При реализации очереди с приоритетом с помощью двоичной кучи просмотр минимального элемента занимает $O(1)$, а его извлечение и добавление нового элемента в очередь занимают $O(\log n)$ времени, где n – число элементов, находящихся в очереди.

Числа Бернулли – числовая последовательность B_i , определённая для $k \geq 0$ соотношениями:

$$\begin{cases} B_0 = 1, \\ B_i = -\frac{1}{i+1} \sum_{k=1}^i \binom{i+1}{k} B_{i-k} \end{cases}$$

Линейная зависимость действительных чисел. Числа w_1, w_2, \dots, w_k ($w_i \in \mathbb{R}$) называются линейно независимыми над \mathbb{Q} , если не существует такого набора рациональных чисел q_1, q_2, \dots, q_k , для которых $q_1^2 + q_2^2 + \dots + q_k^2 > 0$ (то есть они все одновременно не равны нулю) и $q_1 w_1 + q_2 w_2 + \dots + q_k w_k = 0$.