САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Задание по курсу «Алгоритмы и анализ сложности»

Алгоритм Дейкстры поиск кратчайших путей в орграфе

Терещенко Дмитрий Владиславович (группа 17.Б13-пу)

25.11.2019

Содержание

I. Введение

Алгоритм Дейкстры — алгоритм на графах, изобретённый нидерландским учёным *Эдсгером Дейкстрой* в 1959 году. Решает задачу поиска кратчайших путей из одной вершины во взвешенном ориентированном графе в случае, когда веса ребер неотрицательны. Алгоритм широко применяется в программировании и технологиях, например, его используют протоколы маршрутизации OSPF и IS-IS.

Существует множество вариантов реализации данного алгоритма. Все они отличаются выбором структуры данных, но основные шаги остаются неизменны, а именно:

- Хранение доп. информации о вершине: о её посещении; о кратчайшей длине пути до неё
- Получение не посещённой вершины v’ с минимальным кратчайшим расстоянием
- Обновление расстояния до смежных вершин, к которым есть путь из вершины v’.

Поскольку в алгоритме Дейкстры для посещения всегда выбирается самая «лёгкая», или «близкая», вершина, можно утверждать, что этот алгоритм придерживается жадной стратегии. Жадные стратегии не всегда приводят к оптимальным результатам, однако, как видно из приведённой в *источнике [1]* теоремы 24.6 и следствия 24.7 из нее, алгоритм Дейкстры действительно находит кратчайшие расстояния.

II. Алгоритм

Данная реализация выбрана из *источника [1] и [3]*

Обозначения

- $\boldsymbol{V}$ — множество вершин графа
- $\boldsymbol{E}$ — множество рёбер графа
- $\boldsymbol{w}[\boldsymbol{ij}]$ — вес (длина) ребра
- $\boldsymbol{s}$ — исходная вершина
- $\boldsymbol{v}$ — текущая рассматриваемая вершина
- $\boldsymbol{V_T}$ — множество посещённых вершин графа
- $\boldsymbol{distance}[\boldsymbol{u}]$ — по окончании работы алгоритма возвращает длину кратчайшего пути из $\boldsymbol{s}$ до вершины $\boldsymbol{u}$
- $\boldsymbol{parent}[\boldsymbol{u}]$ — по окончании работы алгоритма возвращает родительскую вершину к вершине $\boldsymbol{u}$. Сам массив описывает дерево кратчайших путей.
- $\boldsymbol{Q}$ — очередь с приоритетом не посещённых вершин графа (ключ это $\boldsymbol{distance}[\boldsymbol{u}]$)

Псевдокод

Для всех $\boldsymbol{u} \in \boldsymbol{V}$
    присвоим $\boldsymbol{distance}[\boldsymbol{u}] \leftarrow \infty$
    присвоим $\boldsymbol{parent}[\boldsymbol{u}] \leftarrow None$
    $INSERT(Q, distance[u], u)$ *// добавление вершины в очередь с приоритетом*
Изменим $\boldsymbol{distance}[\boldsymbol{s}] \leftarrow \boldsymbol{0}$
$DECREASE(Q, 0, s)$ *// понижение приоритета вершины* $\boldsymbol{s}$ *значением 0*
Присвоим $\boldsymbol{V_T} \leftarrow \emptyset$
Пока не $IS\_EMPTY(\boldsymbol{Q})$
    присвоим $\boldsymbol{v} \leftarrow EXTRACT\_MIN(\boldsymbol{Q})$
    изменим $\boldsymbol{V_T} \leftarrow \boldsymbol{V_T} \cup \{\boldsymbol{v}\}$
    Для всех $\boldsymbol{u} \in \boldsymbol{V} - \boldsymbol{V_T}$
        если $\boldsymbol{vu} \in \boldsymbol{E}$ то
            если $\boldsymbol{distance}[\boldsymbol{u}] > \boldsymbol{distance}[\boldsymbol{v}] + \boldsymbol{w}[\boldsymbol{vu}]$ то
                изменим $\boldsymbol{distance}[\boldsymbol{u}] \leftarrow \boldsymbol{distance}[\boldsymbol{v}] + \boldsymbol{w}[\boldsymbol{vu}]$
                изменим $\boldsymbol{parent}[\boldsymbol{u}] \leftarrow \boldsymbol{v}$
                $DECREASE(Q, distance[u], u)$

III. Математический анализ

Воспользуемся общим планом математического анализа эффективности не рекурсивных алгоритмов (*источник [4]*):

1. Размер выходных данных: кол-во вершин в графе ($\boldsymbol{n} = |\boldsymbol{V}|$)

2. Базовая операция алгоритма: ***сравнение***

3. Помимо размера входных данных сложность будет зависеть также от кол-ва рёбер, от их весов и от наличия связей со стартовой вершиной. Поэтому рассмотрим:

- ***Лучший случай***: граф, не имеющий связей с исходной вершиной
- ***Средний случай***: можно определить через мат.ожидание, но не очевидно, что взять за случайную величину
- ***Худший случай***: полный граф

4. Подсчитаем кол-во выполняемых базовых операций:

- Цикл по обновлению кратчайших расстояний:
  - Кол-во итераций: *n-1, n-2,…,1*
  - Базовых операций: *a* = либо 1, либо 2
- Для проверки выхода из основного цикла делается 1 сравнение
- Кол-во итераций основного цикла: n
- Получаем: (1 + (n-1)**a*) + (1 + (n-2)**a*) + … + (1 + 1**a*) = n + *a* * n * (n – 1) / 2

Тогда:

- В лучшем случае (*a* = 1): n + *1* * n * (n – 1) / 2 = $\frac{\boldsymbol{n(n+1)}}{\boldsymbol{2}}$
- В худшем случае: (*a* = 2): n + *2* * n * (n – 1) / 2 = n^2

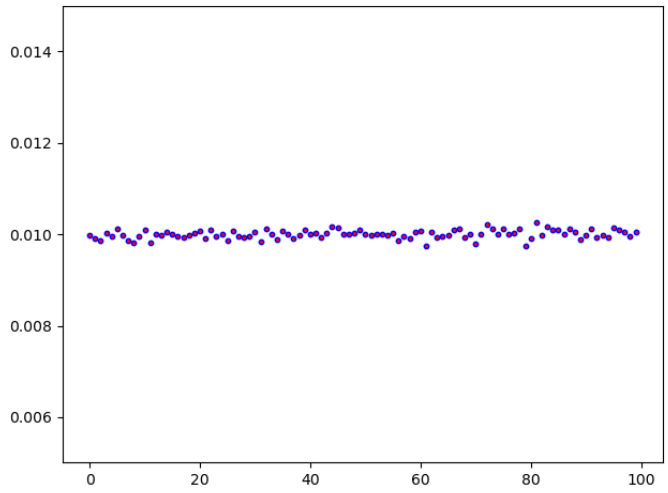
5. Отнесём к классу эффективности: И в худшем, и в лучшем случае при базовой операции ***сравнение*** получаем ***квадратичный класс*** эффективности

IV. Эмпирический анализ алгоритма

Воспользуемся общим планом эмпирического анализа эффективности алгоритмов (*источник [4]*):

1. Цель эксперимента: проверка точности теоретических выводов об эффективности алгоритма (*квадратичная сложность*)
2. Измеряемая метрика ***f***: трудоёмкость алгоритма. Единицы измерения: время выполнения (в миллисекундах)
3. Диапазоны значений:
   a) Кол-во вершин ***n*** (*задаёт размер входных данных*): $[\mathbf{1}; +\infty)$
   b) Кол-во рёбер: $[\mathbf{1}; \frac{\boldsymbol{n(n+1)}}{\mathbf{2}}]$
   c) Веса рёбер: $(\mathbf{0}; +\infty)$
   d) Номер вершины: $[\mathbf{0}; \boldsymbol{n} - \mathbf{1}]$
4. Программная реализация:
   - Генератор образца входных данных:
     *На вход принимает кол-во вершин **n**, на выходе даёт матрицу смежности и номер стартовой вершины.*
     Основные шаги:
     - Случайным образом выбирается кол-во рёбер ***m*** (в диапазоне *b)*)
     - Случайным образом генерируется матрица инцидентности ***n*** x ***n***:
       1) Сначала задаётся полностью заполненная 0-ми весами (означает, что нет связей)
       2) Случайным образом выбирается ребро (два номера вершин (каждый в диапазоне *d*))
       3) Случайным образом выносится решение о наличие ребра (диапазон $[False; True]$)
       4) Если *True*, то случайным образом выставляется вес (конечный поддиапазон диапазона *c)*)
     - Случайным образом назначается стартовая вершина (диапазон *d)*)

     * Равномерность генерируемых данных зависит от равномерности генерируемых значений функцией *random*. Поэтому этот факт был проверен.
     На графике показано частотное распределение значений от 0 до 100 при выборке размером 1000000.



- Алгоритм:
  На вход принимает матрицу смежности и номер стартовой вершины, на выходе даёт массив кратчайших расстояний от стартовой вершины до всех остальных.
  Сам алгоритм описан во II части.

***** Ссылка на GitHub: https://github.com/Dima12101/Empirical_Analysis

***Весь код в файле**: algorithmDijkstra.py*

5. Полученные результаты:
   > Диапазон значений ***n***: *[1;100]* (конечный поддиапазон диапазона *a)*)
   > Значение ***m*** (кол-во образцов входных данных при фиксированном n): *100*

   ***** Значения выбирались, учитывая производительность вычислительного устройства

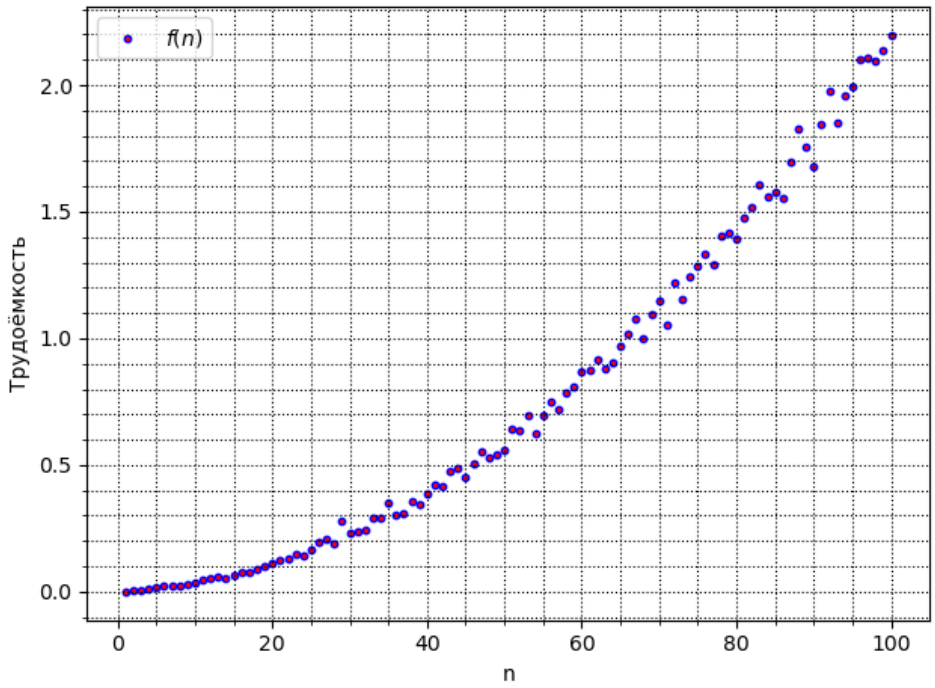
   ***** Также было введено значений ***repeats*** = 100 для решения проблемы, озвученной в *источнике [5]: «…высокая скорость современных компьютеров может привести к тому, что время работы будет невозможно зарегистрировать (будут получаться нулевые значения). Обойти эту неприятность легко, запуская программу в цикле много раз, а затем поделив зарегистрированное время выполнения на количество итераций цикла»*

Таблица

n	1	2	3	4	5	6	7	8	9	10
f(n)	0,0017	0,0034	0,0068	0,0087	0,0151	0,0197	0,0212	0,0230	0,0264	0,0319
11	12	13	14	15	16	17	18	19	20	21
0,0475	0,0528	0,0582	0,0523	0,0664	0,0738	0,0771	0,0887	0,1026	0,1146	0,1230
22	23	24	25	26	27	28	29	30	31	32
0,1286	0,1448	0,1408	0,1640	0,1971	0,2083	0,1887	0,2766	0,2294	0,2362	0,2443
33	34	35	36	37	38	39	40	41	42	43
0,2900	0,2926	0,3493	0,3011	0,3083	0,3565	0,3420	0,3835	0,4215	0,4145	0,4728
44	45	46	47	48	49	50	51	52	53	54
0,4870	0,4513	0,5044	0,5544	0,5278	0,5411	0,5573	0,6429	0,6335	0,6937	0,6215
55	56	57	58	59	60	61	62	63	64	65
0,6984	0,7518	0,7210	0,7865	0,8114	0,8659	0,8716	0,9173	0,8779	0,9049	0,9672
66	67	68	69	70	71	72	73	74	75	76
1,0147	1,0794	0,9997	1,0963	1,1473	1,0529	1,2195	1,1522	1,2431	1,2867	1,3348
77	78	79	80	81	82	83	84	85	86	87
1,2913	1,4062	1,4141	1,3949	1,4791	1,5171	1,6098	1,5586	1,5782	1,5535	1,6983
88	89	90	91	92	93	94	95	96	97	98
1,8298	1,7561	1,6780	1,8446	1,9763	1,8540	1,9599	1,9967	2,0998	2,1103	2,0973

99	100
2,1369	2,1957

График



6. Анализ:

Для проверки теоретической оценки $\theta(n^2)$ воспользуемся её определением:

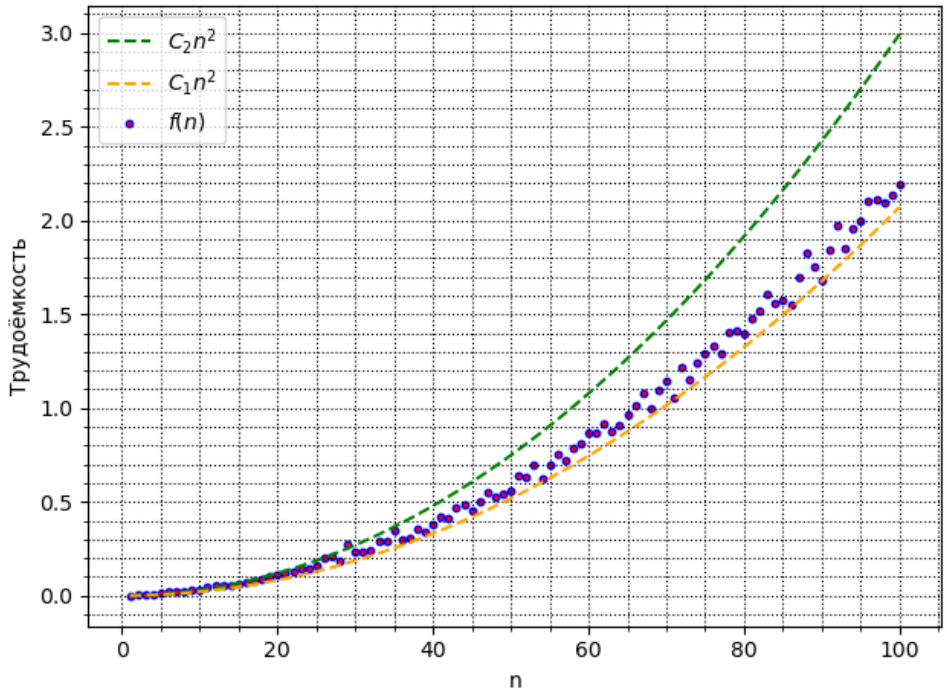
$f(n) = \theta\big(g(n)\big)$ ***если*** $\exists\ const\ C_1, C_2 > 0;\ \exists n_0 \in N$ ***что*** $C_1 g(n) \leq f(n) \leq C_2 g(n)\ \forall n > n_0$

Поэтому для $n_0 \in [1{,}100]$ были найдены C_1 и C_2 по следующему принципу:

- $C_1 = \min(\frac{f(n)}{n^2})$ для $n \in [n_0, 100]$
- $C_2 = \max(\frac{f(n)}{n^2})$ для $n \in [n_0, 100]$
- $C_1 > 0$ и $C_2 > 0$

По итогу вышло:

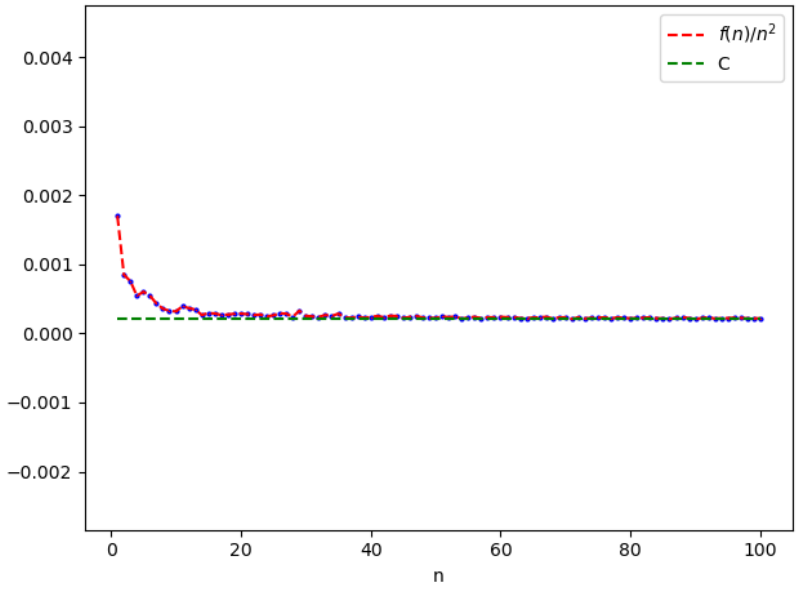
$n_0 = 30, C_1 = 0.0002071613, C_2 = 0.0002993999$



Т.к. $g(n) = n^2 > 0$, то выполняется следующие:

$$f(n) = \theta\big(g(n)\big) <=> \frac{\boldsymbol{f(n)}}{\boldsymbol{g(n)}} \rightarrow_{\boldsymbol{n\to\infty}} \boldsymbol{Const \neq 0}$$

Но левый факт был получен на конечном отрезке, поэтому дополнительно было решено построить график $\frac{\boldsymbol{f(n)}}{\boldsymbol{g(n)}}$.



Видно, что это отношение на этом интервале n стремиться к асимптоте $C = 0.0002071613$

7. <u>Вычислительная среда и оборудование:</u>
    - Процессор: *Intel® Core™ i5-3210M CPU @ 2.50GHz 2.50GHz*
    - Тип системы: *64-разрядная*
    - Язык программирования: *Python*
    - Библиотеки:
        - *NumPy* для векторных вычислений
        - *Random* для генерации случайных значений
        - *Math* для использования базовых функций
        - *Time* для вычисления сложности алгоритма
        - *Heapq* для использования очереди с приоритетом
        - *MatPlotLib* для визуализации

***Весь код в файле**: empiricalAnalysis.py*

V. Доверительная трудоемкость – новая оценка качества алгоритмов

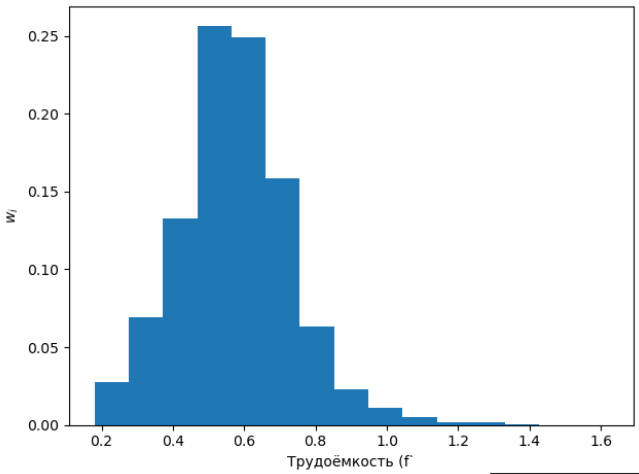
Из *источника [6]*: «Точечные оценки трудоемкости как дискретной ограниченной случайной величины – мода, медиана и математическое ожидание не могут быть использованы как гарантирующие оценки, а очевидно гарантирующая оценка по максимуму – теоретическая трудоемкость в худшем случае – дает слишком завышенные временные прогнозы»

Для определения значений функции ***доверительной трудоемкости*** алгоритма $f_\gamma(n)$, аргументом которой является длина входа, с целью последующего прогнозирования его временной эффективности воспользуемся методикой, изложенной также в *источнике [6]* и включающей два этапа исследования:

A. Этап предварительного исследования (проверка гипотезы о законе распределения)

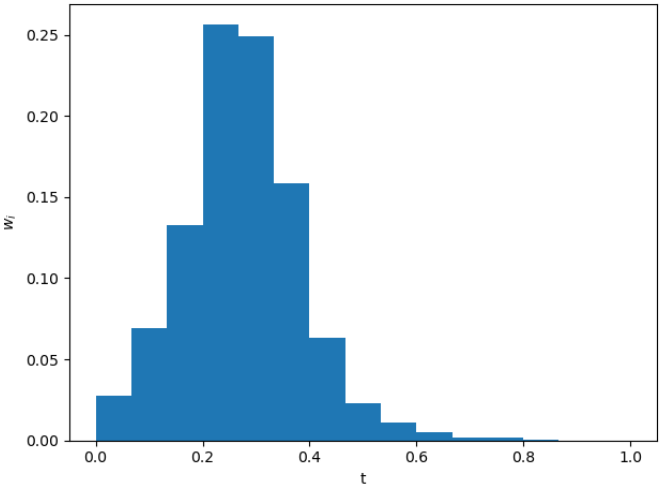
1. Фиксированная длина входа ***n***: 50
2. Число экспериментов ***m***: 20000

   ***** Также как и ранее введено значение ***repeats***: 100
3. Проведено экспериментальное исследование и получены значения $f_i = f_A(D_i),\ i = \overline{1,m}$
4. В качестве $f_A^\wedge$ и $f_A^\vee$ были выбраны эмпирические значения, а именно:
   $f_A^\wedge = \max_{D\in D_n}\{f_A(D)\} = \max_{i=\overline{1,m}}\{f_A(D_i)\} = \max_{i=\overline{1,m}}\{f_i\}$
   $f_A^\vee = \min_{D\in D_n}\{f_A(D)\} = \min_{i=\overline{1,m}}\{f_A(D_i)\} = \min_{i=\overline{1,m}}\{f_i\}$
5. Число сегментов $k = [1 + \log_2 m]$=15 *(Формула Стерджесса)*
6. Гистограмма относительных частот ($\boldsymbol{w_i = \frac{m_i}{m}}$; $\sum_{i=1}^{k} m_i = m$; $\sum_{i=1}^{k} w_i = 1$):



$\Rightarrow \quad \boldsymbol{t_i = \frac{f_i - f_A^\vee}{f_A^\wedge - f_A^\vee}} \quad \Rightarrow$

$\Rightarrow$



7. Выборочное средние и выборочная дисперсия:

$\bar{t} = \frac{\overline{f_A} - f_A^{\vee}}{f_A^{\wedge} - f_A^{\vee}}$; $s^2 = \frac{1}{m-1}\sum_{i=1}^{m} \frac{(f_i - \overline{f_A})^2}{(f_A^{\wedge} - f_A^{\vee})^2}$

, где $\overline{f_A} = \frac{1}{m}\sum_{D \in D_n} f_A(D) = \frac{1}{m}\sum_{i=1}^{m} f_A(D_i) = \frac{1}{m}\sum_{i=1}^{m} f_i$ *(справедлива в случае, если все входы D равновероятны)*

$\Rightarrow \bar{t} = 0.273476;\ s^2 = 0.012233$

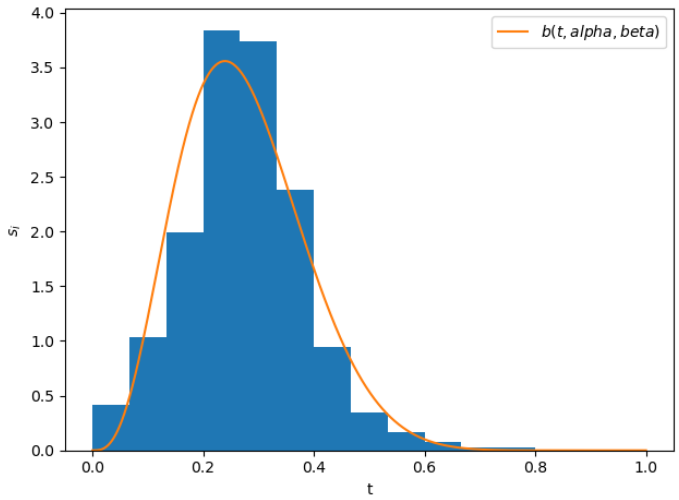
8. Проверим нулевую гипотезу:
   - Формулировка: нормированная трудоемкость имеет ***бета-распределение***
   - Вычислим параметры бета-распределения по <u>методу моментов</u>:

     $\alpha = \frac{\bar{t}}{s^2}(\bar{t} - (\bar{t})^2 - s^2)$; $\beta = \frac{1-\bar{t}}{s^2}(\bar{t} - (\bar{t})^2 - s^2)$

     $\Rightarrow \alpha = 4.168164;\ \beta = 11.073255$

     ** Получаем аппроксимацию для гистограммы распределения*

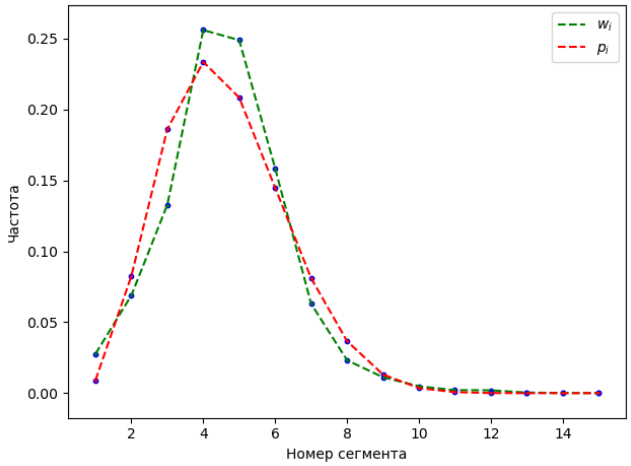
     ($s_i = \frac{w_i}{\Delta_i} = \frac{w_i}{\Delta} = \frac{w_i}{\frac{(1-0)}{k}} = w_i * k$):



   - Рассчитаем теоретические частоты:

     $p_i = \int_{t_i}^{t_i + \Delta_i} b(t, \alpha, \beta)dt$

     ** Получаем следующие соотношения частот:*

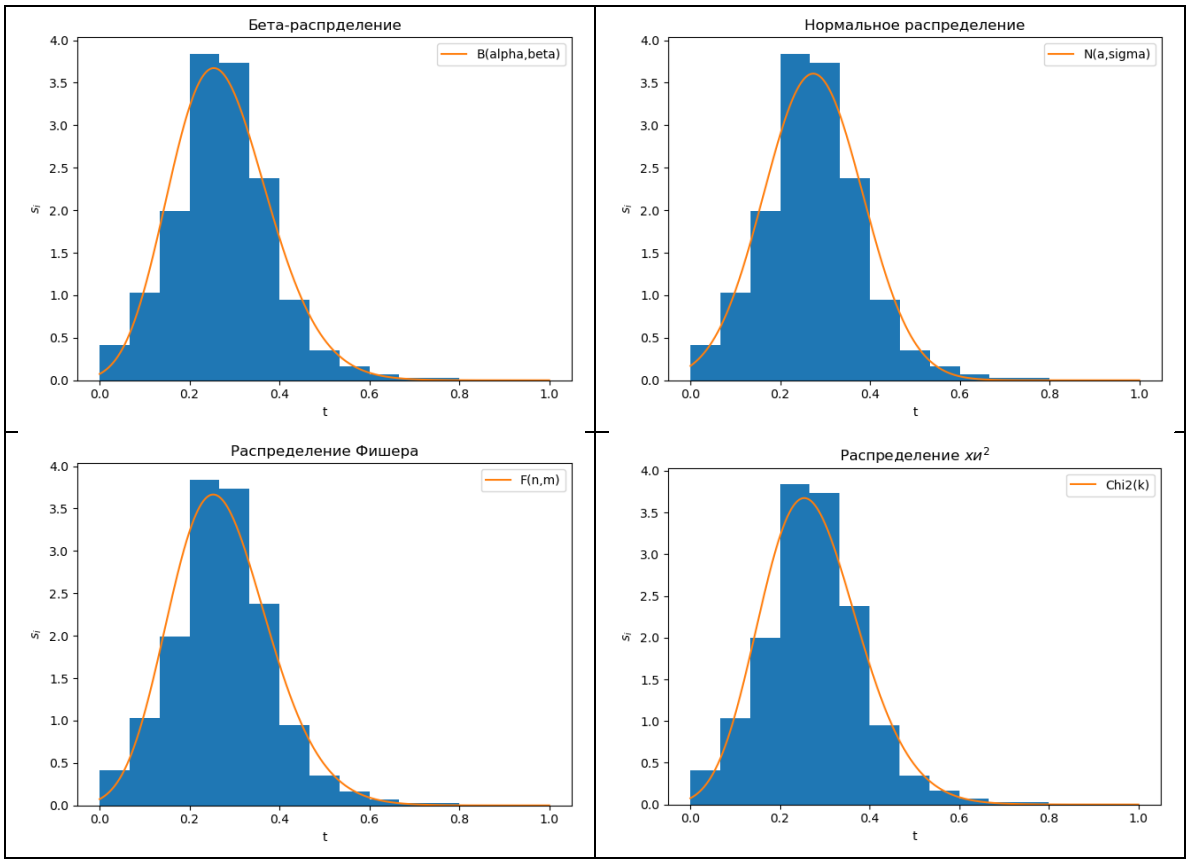


   - Проверка гипотезы (*уровень значимости = 0.05):*
     - $\chi^2_{\text{набл}} = m\sum_{i=1}^{m} \frac{(w_i - p_i)^2}{p_i} = 1379774.5054$
     - $\chi^2_{\text{кр}} = \chi^2_{\text{кр}}(0.05, k-1-2) = \chi^2_{\text{кр}}(0.05, 12) = 21.0260$

- $\chi^2_{\text{набл}} > \chi^2_{\text{кр}}$ **=> гипотеза не верна!**

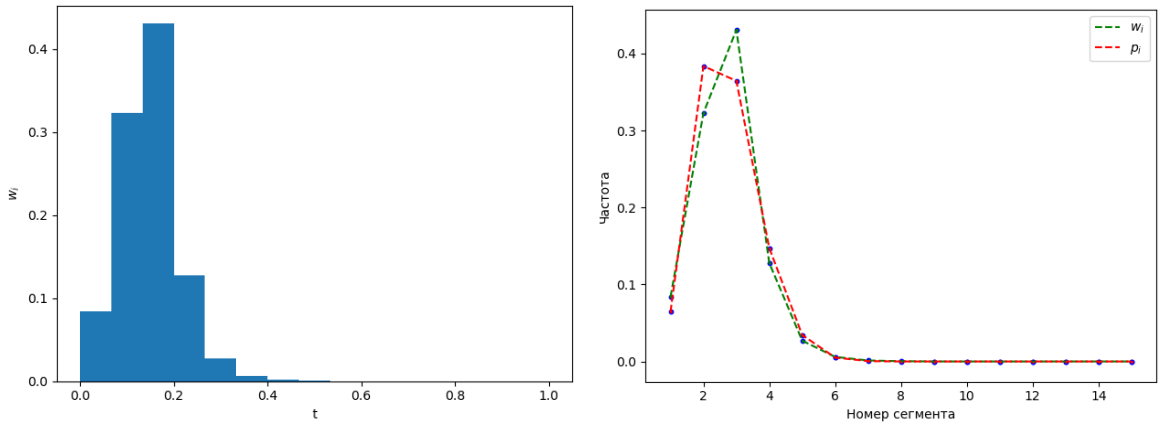
Вышло, что бета-распределение слишком грубая аппроксимация.

Первым делом попробовал другие распределения на той же выборке с помощью python модуля *scipy.stats*. Судя по документации параметры рассчитываются по методу максимального правдоподобия. Получилось следующие:



Как видно лучше не стало.

Последним попробовал увеличить выборку с 20000 до 30000 и снова построить бета-распределение.



И снова гипотеза была отвергнута.

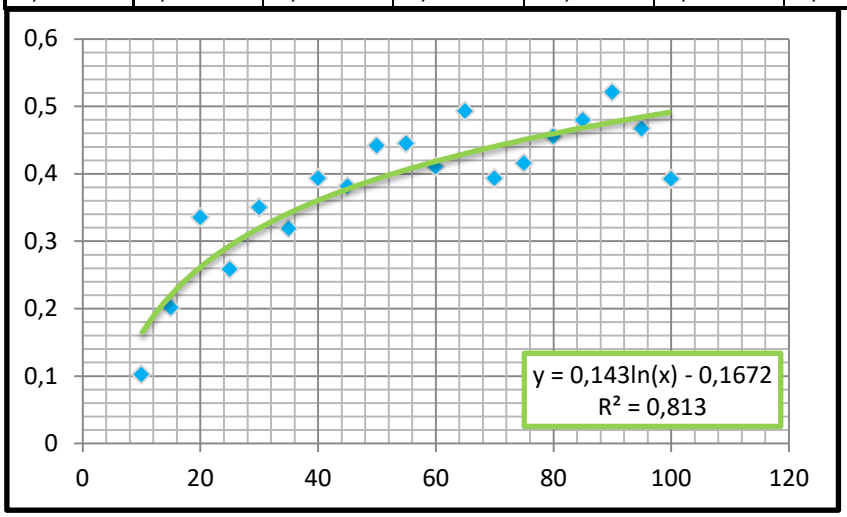
***Весь код в файле**: confidentialComplexity.py*

B. Этап основного исследования

Решил рискнуть и продолжить, взяв бета-распределение.

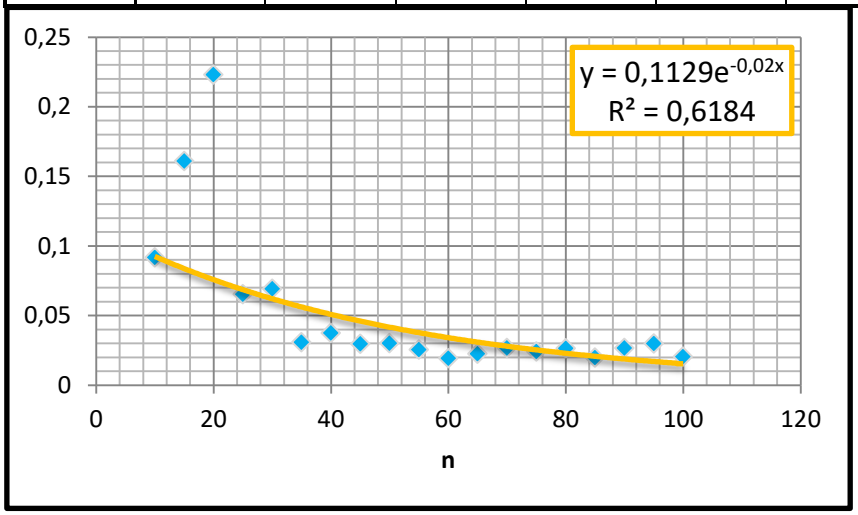
1. В качестве диапазона длин входа, на котором будут построены интервальные оценки возьмём: *[10;400]*
2. Диапазон длин входа, на котором будут проводиться экспериментальные исследования: *[10;100]*
3. Шаг изменения длины входа в экспериментальном исследовании: *5*
4. Кол-во экспериментов ***m***: *1000*
5. Уравнение регрессии:
   - Для выборочной средней $\bar{t}(n)$:

n	10	15	20	25	30	35	40	45	50
$\bar{t}$	0,101983	0,200952	0,334903	0,257986	0,349948	0,318351	0,392989	0,381249	0,441224
55	60	65	70	75	80	85	90	95	100
0,44459	0,409993	0,492489	0,393195	0,41515	0,454994	0,479378	0,520796	0,466488	0,392168

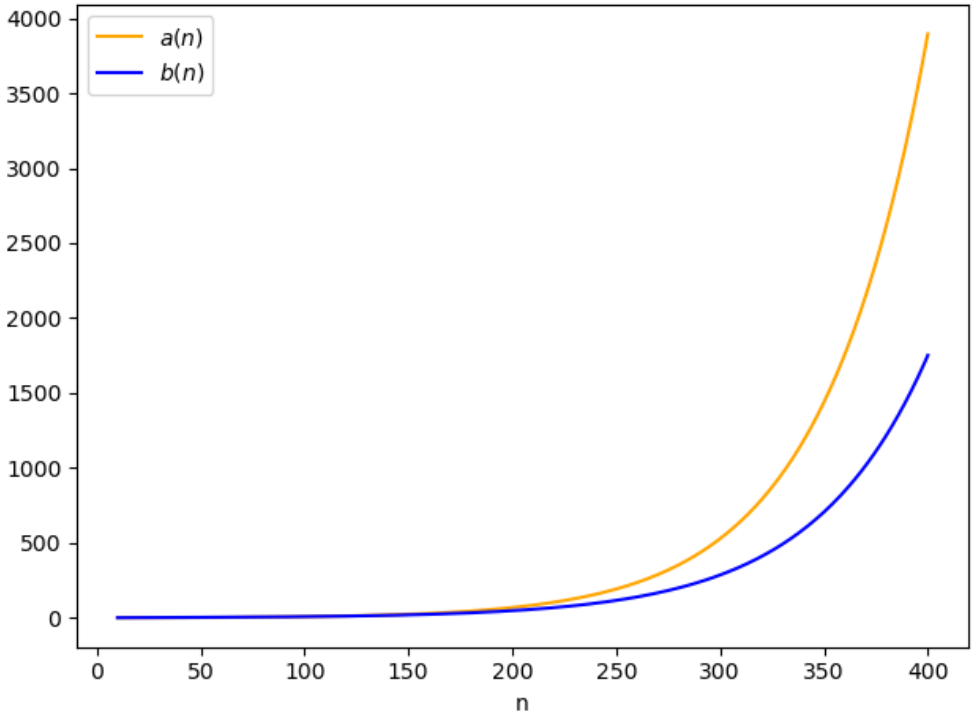


   - Для выборочной дисперсии $s^2(n)$:

n	10	15	20	25	30	35	40	45	50
s^2	0,091657	0,160684	0,222868	0,064994	0,069049	0,030586	0,037252	0,02949	0,030012
55	60	65	70	75	80	85	90	95	100
0,025275	0,01903	0,022326	0,026529	0,023883	0,02628	0,019635	0,026563	0,029623	0,020328

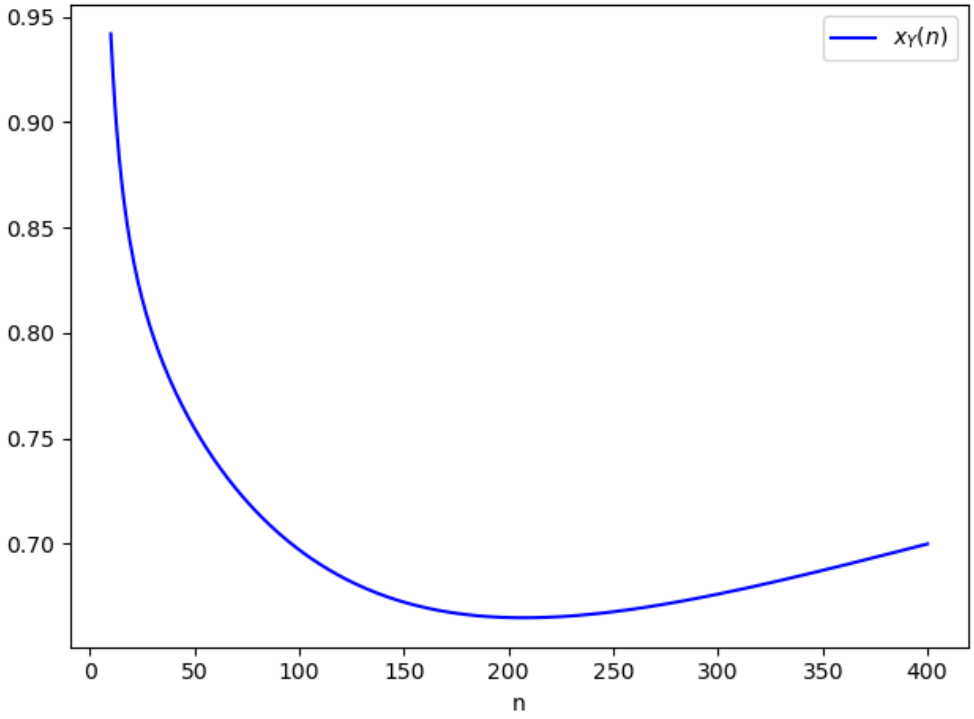


6. Параметры бета-распределения $\boldsymbol{\alpha(n)}$ и $\boldsymbol{\beta(n)}$:



7. Значения левого $\boldsymbol{\gamma}$ − квантиля бета-распределения (доверительная вероятность $\gamma = 0.95$):

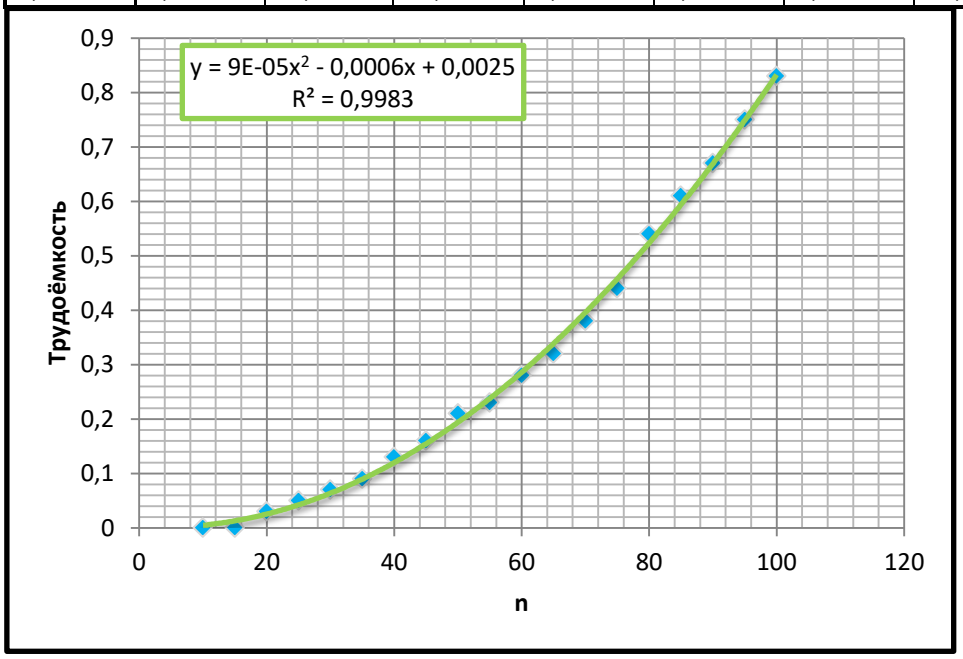
$\boldsymbol{x_{\gamma}(n) = B^{-1}(\gamma, \alpha(n), \beta(n))}$



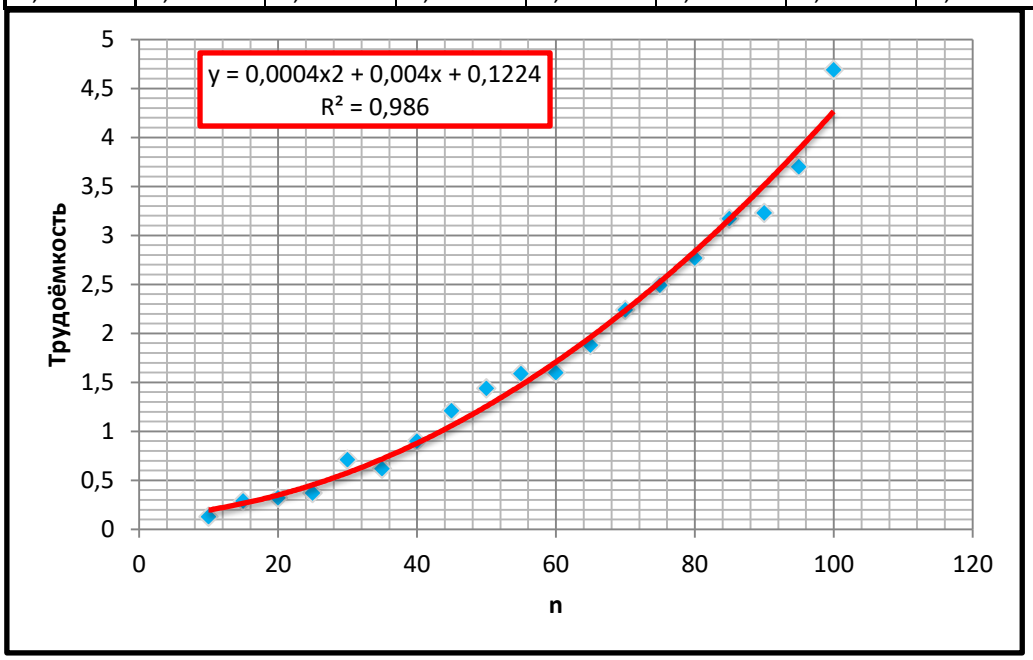
8. Функция доверительной трудоёмкости:

- Построим уравнения регрессии для $f^{\wedge}(n)$ и $f^{\vee}(n)$:

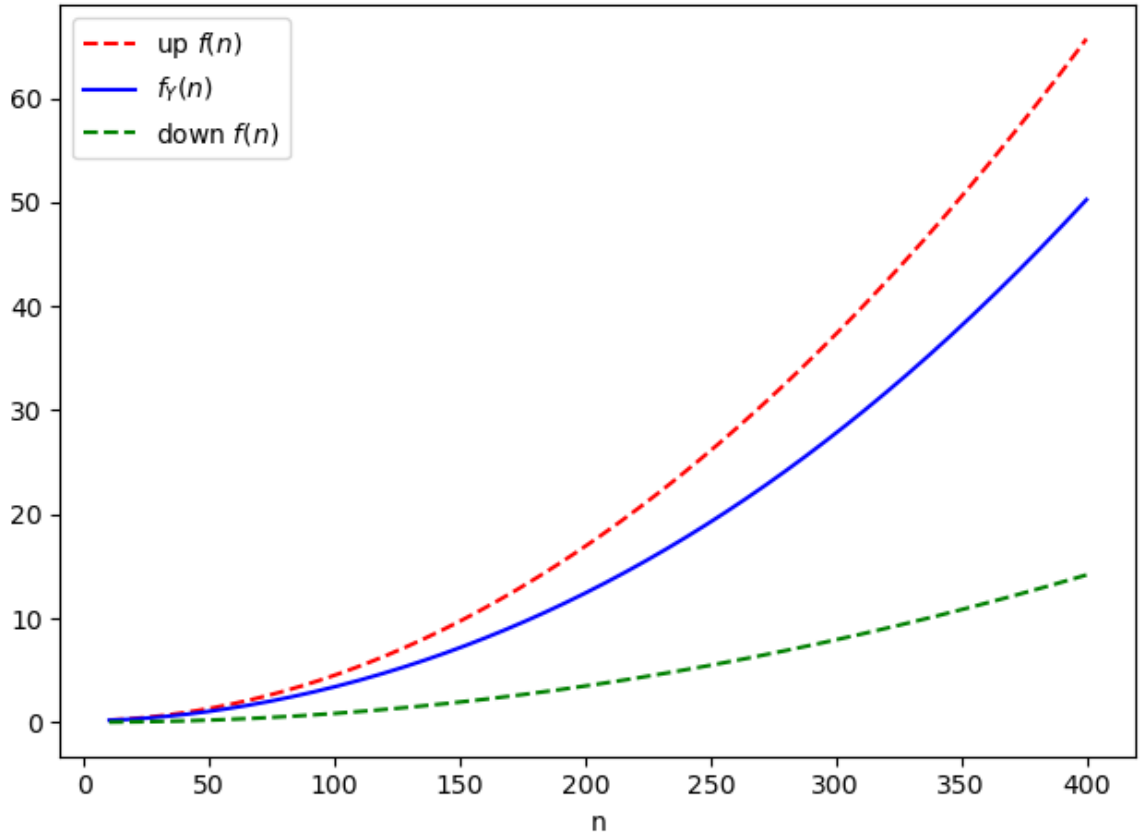
n	10	15	20	25	30	35	40	45	50
$f^{\vee}$	0	0	0,029993	0,049999	0,069997	0,089993	0,129969	0,159903	0,209942
55	60	65	70	75	80	85	90	95	100
0,229976	0,280023	0,31996	0,37998	0,440004	0,539963	0,609884	0,66999	0,749989	0,830004



n	10	15	20	25	30	35	40	45	50
$f^{\wedge}$	0,130019	0,289993	0,319977	0,370009	0,710003	0,62001	0,899954	1,210103	1,439996
55	60	65	70	75	80	85	90	95	100
1,589973	1,600001	1,880059	2,239952	2,489848	2,770059	3,170021	3,229949	3,699973	4,690082



- В итоге получаем:
  $\boldsymbol{f_\gamma = f^\vee(n) + x_\gamma(n)(f^\wedge(n) - f^\vee(n))}$



Вычислительная среда и оборудование:

- Доп. библиотеки:
  - *Scipy.stats* для различных распределений
  - *Scipy*.special для использования Гамма-функции в бета-распределении
  - *Scipy.integrate* для интегрирования

VI. Литература

1. *Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн*. Часть VI.Алгоритмы для работы с графами: Глава 24. Кратчайшие пути из одной вершины: Алгоритм Дейкстры //Алгоритмы: построение и анализ — 3-е изд. — М.: «Вильямс», 2013. — С. 696–702. — ISBN 978-5-8459-1794-2.
2. Kvodo (Computing Science & Discrete Match). Алгоритм Дейкстры [Электронный ресурс]. – Режим доступа: http://kvodo.ru/dijkstra-algorithm.html, свободный – (24.11.2019).
3. *Левитин А. В*. Глава 9. Жадные методы: Алгоритм Дейкстры // Алгоритмы. Введение в разработку и анализ — М.: Вильямс, 2006. — С. 386–391. — ISBN 5-8459-0987-2.
4. *Левитин А. В*. Глава 2. Основы анализа эффективности алгоритмов: Математический анализ нерекурсивных алгоритмов // Алгоритмы. Введение в разработку и анализ — М.: Вильямс, 2006. — С. 98–106. — ISBN 5-8459-0987-2.
5. *Левитин А. В*. Глава 2. Основы анализа эффективности алгоритмов: Эмпирический анализ алгоритмов // Алгоритмы. Введение в разработку и анализ — М.: Вильямс, 2006. — С. 127–134. — ISBN 5-8459-0987-2.
6. *М.В. Ульянов, В.Н. Петрушин, А.С. Кривенцов*. Доверительная трудоёмкость – новая оценка качества алгоритмов // Информационные технологии и вычислительные системы. 2009, №2. — С. 23–37.