Федеральное государственное бюджетное образовательное

учреждение высшего профессионального образования

Санкт-Петербургский государственный университет

Факультет прикладной математики и процессов управления

**Проектная работа**

По дисциплине: *Алгоритмы и анализ сложности*

LU-разложение

Выполнил:

*Студент 3 курса группы 18.Б13-пу*

*бакалаврской программы*

*“Программирование и информационные технологии”*

*Клемешов Дмитрий Алексеевич*

Проверил:

*к.ф.-м.н. Никифоров Константин Аркадьевич*

Санкт-Петербург

2020 г.

Содержание

1. Введение
2. Алгоритм
3. Математический анализ алгоритма
4. Эмпирический анализ алгоритма
   1. Цель эксперимента

4.2.1 Измеряемая метрика

4.2.2 Единицы измерения

4.3 Характеристики входных данных

4.4 Программная реализация алгоритма

4.5 Генератор образца входных данных

4.6 Выполнение алгоритма над образцом входных данных

4.7 Анализ полученных результатов

4.8 Вычислительная среда и оборудование

5. Литература

Введение

LU разложение - представление матрицы A в виде произведения двух матриц, A=LU, где L — нижняя треугольная матрица, а U — верхняя треугольная матрица. Был впервые открыт польским математиком Тадеушем Банахевичем в 1938 году. [1][2]

LU-разложение используется для решения систем линейных уравнений, обращения матриц и вычисления определителя. LU-разложение существует только в том случае, когда матрица A обратима, а все ведущие (угловые) главные миноры матрицы A невырождены. [1]

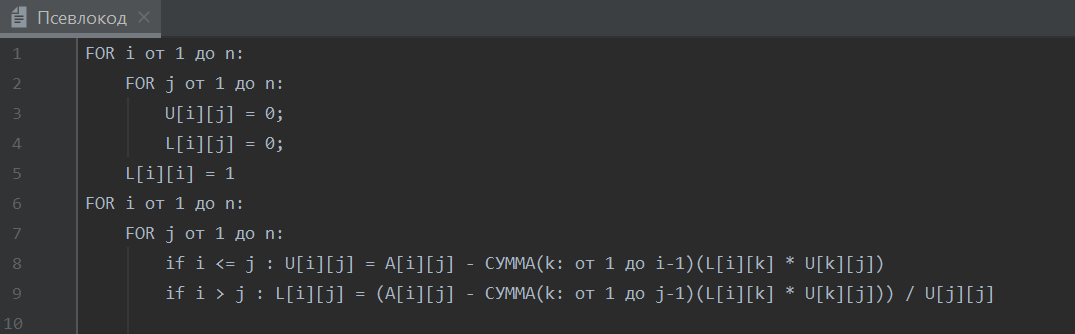
Заметим, что, получив LU-разложение матрицы **А**, мы можем решать системы линейных уравнений **Ах = b** для разных векторов свободных членов **b**. Это главное преимущество метода LU-разложения по сравнению с классическим методом исключения Гаусса. Заметим также, что LU-разложение не требует дополнительной памяти, поскольку ненулевую часть матрицы **U** мы можем хранить в верхнетреугольной части матрицы **А** (включая главную диагональ), а нетривиальную часть матрицы **L** - ниже главной диагонали **А**.[3]

Алгоритм

Теорема: если все главные миноры квадратной матрицы **A** отличны от нуля, то существуют такие нижняя **L** и верхняя **U** треугольные матрицы, что **A = LU.**  Если элементы диагонали одной из матриц **L** или **U** фиксированы (ненулевые), то такое разложение единственно. [4]

Обозначения: причем диагональные элементы матрицы

Найти матрицы L и U можно следующим образом:[4]



Математический анализ алгоритма

Сложность алгоритма - O(n3).

Доказательство:

Очевидно, сложность первой части алгоритма составляет O(n2);

Сложность второй части алгоритма:

Исключение первого столбца (i = 1) потребует n сложений и n умножений для n−1 строк. Таким образом, число операций для первого столбца равно 2n(n−1). Для второго столбца (i = 2) у нас есть n-1 сложений и n−1 умножений, и мы делаем это для (n−2) строк, дающих нам 2(n−1)(n-2). Поэтому общее количество операций, необходимых для полной декомпозиции, можно записать в виде:

Таким образом, имеем сложность O(n2) + O(n3) = O(n3), что говорит нам о **кубической** эффективности алгоритма.

Эмпирический анализ алгоритма

Воспользуемся общим планом эмпирического анализа алгоритма:

1. Цель эксперимента: проверка точности теоретических выводов об эффективности алгоритма.
2. Измеряемая метрика: трудоемкость алгоритма.

Единицы измерения: время выполнения программы.

1. Характеристики входных данных:

* Размер матрицы *n:* [4, 80] с шагом 1;
* Значения ячеек матрицы: [-2000; 2000].

1. Программная реализация алгоритма:

Алгоритм принимает на вход матрицу ***A***, на выходе выдает матрицы ***L*** и ***U.***

Код алгоритма находится в файле ***luAlgorithm.py (lu\_decomposition).***

1. Генератор образца входных данных:

На вход принимает размер матрицы *n*, на выходе дает матрицу ***A*** размерности [n x n].

Основные шаги:

* Создание пустой матрицы размерности [n x n];
* Заполнение матрицы в цикле случайными значениями в заданном диапазоне.

Код генератора находится в файле ***luAlgorithm.py (generator)***.

1. Выполнение алгоритма над образцом входных данных.

Будем повторять алгоритм для каждого *n* заданное количество раз (*10* раз) для получения усредненного времени выполнения.

* 1. Измеренные значения трудоемкости:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **n** | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| **f(n) (мс)** | 0.00010 | 0.00009 | 0.00030 | 0.00010 | 0.00059 | 0.00020 | 0.00060 | 0.00079 | 0.00089 | 0. 00109 | 0.00169 |
| 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 |
| 0.00189 | 0.00229 | 0.00261 | 0.00360 | 0. 00359 | 0. 00460 | 0. 00489 | 0. 00609 | 0. 00690 | 0. 00730 | 0. 00949 | 0.00959 |
| 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 |
| 0. 01080 | 0. 01169 | 0. 01379 | 0. 01340 | 0. 01659 | 0. 01729 | 0.01960 | 0.02079 | 0.02320 | 0.02479 | 0.05299 | 0.03600 |
| 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 |
| 0.03160 | 0.03368 | 0.03568 | 0.03899 | 0.04259 | 0.04539 | 0.04740 | 0.05080 | 0.05465 | 0.06230 | 0.08678 | 0.06531 |
| 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 |
| 0.06920 | 0.07380 | 0.07852 | 0.08153 | 0.11711 | 0.08939 | 0.09370 | 0.10099 | 0.10629 | 0.14239 | 0.11810 | 0.12288 |
| 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 |
| 0.12914 | 0.16624 | 0.14241 | 0.14800 | 0.18339 | 0.16351 | 0.17400 | 0.20589 | 0.18511 | 0.19300 | 0.23315 | 0.20738 |
| 75 | 76 | 77 | 78 | 79 | 80 |  |  |  |  |  |  |
| 0.26153 | 0.22744 | 0.27630 | 0.25112 | 0.28540 | 0.26385 |  |  |  |  |  |  |

Таб.1 Измеренные значения трудоемкости

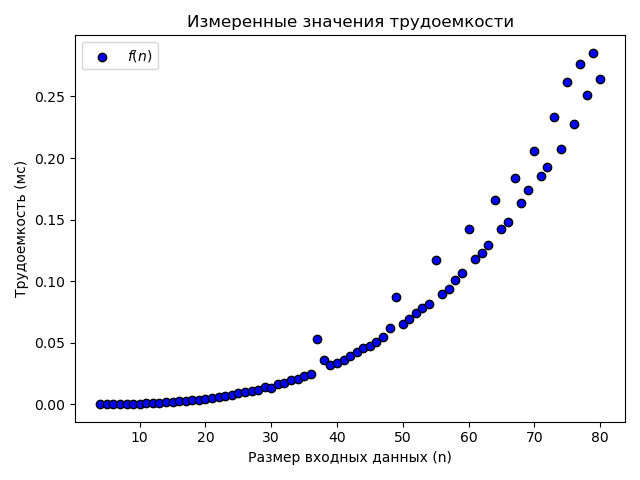


Рис. 1 Измеренные значения трудоемкости

1. Анализ полученных результатов

Для проверки теоретической оценки воспользуемся определением:

Для найдем константы по принципу:



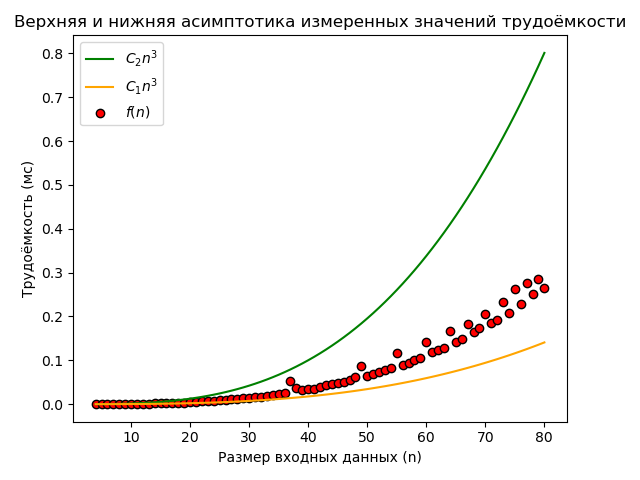


Рис. 2. Верхняя и нижняя асимптотика измеренных значений трудоемкости

Построим график

Анализируя теоретическую оценку: и график, делаем вывод, что с увеличением n происходит затухание к 8.

Код анализа находится в файле ***analysis.py***



Рис. 3. Отношение измеренных значений трудоемкости при удвоении размера входных данных

1. Вычислительная среда и оборудование:

* Процессор *Intel® Core™ i7-4710HQ CPU @ 2.50GHz (8CPUs)*
* Установленная память (ОЗУ): 12 ГБ DDR4
* Операционная системы: Windows 10 Домашняя 64 разрядная
* Язык программирования: Python 3.9
* Используемые библиотеки:
* *NumPy* для векторных вычислений
* *Random* для генерации случайных значений
* *MatPlotLib* для визуализации результатов
* *Time* для вычисления трудоемкости алгоритм

Весь код доступен по ссылке: https://github.com/Klemeshov/UNI/tree/algo/lu

Литература

1. [Wikipedia](https://ru.wikipedia.org/wiki/LU-разложение). LU - разложение
2. [Wikipedia](https://en.wikipedia.org/wiki/LU_decomposition). LU decomposition
3. Левитин А. В. Алгоритмы. Введение в разработку и анализ — М.: Вильямс, 2006. — 576 с. — ISBN 978-5-8459-0987-9
4. Вержбицкий В.М. Основы Численных Методов. Учебник для вузов. — ФГУП "издательство "Высшая Школа", 2002. — С. 63-64. — ISBN 5-06-004020-8.