Министерство образования и науки Российской Федерации Федеральное агентство по образованию Федеральное государственное бюджетное образовательное учреждение высшего образования «Вятский государственный университет»

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

Лабораторная работа №1

по курсу «Параллельное программирование»

Выполнил студент группы ИВТ-31\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Птахова А.М/

Проверил доцент кафедры ЭВМ \_\_\_\_\_\_\_\_\_\_\_\_\_\_/ Долженкова М.Л./

Киров 2023

**1. Цель**

Получение навыков в исследовании и оценке сложных вычислительных алгоритмов.

**2. Задание**

- Изучить алгоритм LU-разложение матрицы.

- Провести доказательную оценку алгоритма по временной сложности и затратам памяти.

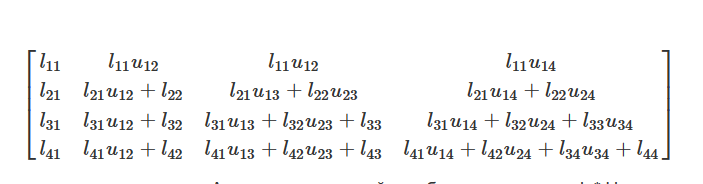
- Реализовать алгоритм с помощью языка С++.

- Построить набор тестовых примеров (не менее 10) и провести оценку эффективности реализованного алгоритма.

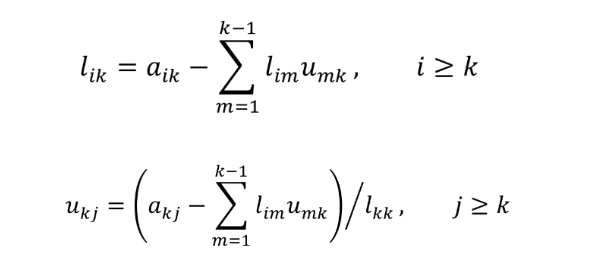
**3. Алгоритм LU – разложения матрицы**

3.1. Алгоритм Краута

Суть алгоритма Краута. Предполагая, что разложение A = L \* U существует, запишем произведение L \* U:



Сравнивая компоненты этого произведения, можно вывести следующие формулы:



Как можно заметить, для реализации данного метода необходимо создать 3 матрицы (U, L, A) одинаковой размерности, что приведет к большим ресурсным затратам: при размерности матрицы 1000х1000 программа завершится с ошибкой 3 (память заполнена).

Для минимизации ресурсных затрат можно воспользоваться уменьшением количества используемых матриц. В частности, использование одной матрицы, исходной, и выполнение преобразований непосредственно в ней.

3.2. Алгоритм Краута для одной матрицы

Алгоритм Краута для одной матрицы предполагает вычисление и формирование новых значений элементов матрицы прямо во время работы с ней.

3.3. Усовершенствованный алгоритм Краута

Усовершенствованный алгоритм Краута не использует временного хранилища sum, он записывает вычисленное значение опорного элемента вместо текущего. В остальном же усовершенствованный алгоритм похож на простой алгоритм Краута.

**4. Асимптотическая оценка**

4.1. Сложность алгоритма Краута и затраты по памяти

for (long i = 0; i < n; i++) // выполнится N раз

{

for (long k = i; k < n; k++) // выполнится N\*N раз

{

for (long j = 0; j < i; j++) // выполнится N\*N\*N раз

{

sum += (a[i][j] \* a[j][k]); // выполнится N\*N\*N раз

}

a[i][k] = a[i][k] - sum; // выполнится N\*N раз

}

for (long k = i; k < n; k++) // выполнится N\*N раз

{

if (i != k)

{

int sum = 0;

for (long j = 0; j < i; j++) // выполнится (N-1)/2 \*N раз

{

sum += (a[k][j] \* a[j][i]); // выполнится (N-1)/2 \*N раз

}

a[k][i] = (a[k][i] - sum) / a[i][i]; // выполнится (N-1)/2 \*N раз

}

}

}

Сложность = О(N^3 + N^2 + N) = O(N^3)

Затраты по памяти:

4 \* 8(long-переменные) + 4(int-sum)+4(unsigned int – n) + 8\*n\*n(double-a) = 40+8\*N^2

4.2. Сложность усовершенствованного алгоритма Краута

for (long k = 0; k < n; k++) // выполнится N раз

{

for (long i = k + 1; i < n; i++) // выполнится N\*N раз

{

a[i][k] = a[i][k] / a[k][k]; // выполнится N\*N раз

}

for (long i = k+1; i < n; i++) // выполнится N\*N раз

for (long j = k + 1; j < n; j++) // выполнится N\*N\*N раз

{

a[i][j] = a[i][j] - a[i][k] \* a[k][j]; // выполнится N\*N\*N раз

}

}

Сложность = О(N^3+N^2+N) = O(N^3)

Затраты по памяти:

4 \* 8(long-переменные) +4(unsigned int – n) + 8\*n\*n(double-a) = 36+8\*N^2

**5. Листинг программы**

void LU\_Decomposition(vector <vector<double>>&a, long n)

{

double time = 0;

for (long i = 0; i < n; i++)

{

//upper triangular

for (long k = i; k < n; k++)

{

double sum = 0;

for (long j = 0; j < i; j++)

{

sum += (a[i][j] \* a[j][k]);

}

a[i][k] = a[i][k] - sum;

}

//lower tiangular

for (long k = i; k < n; k++)

{

if (i != k)

{

int sum = 0;

for (long j = 0; j < i; j++)

{

sum += (a[k][j] \* a[j][i]);

}

a[k][i] = (a[k][i] - sum) / a[i][i];

}

}

}

cout << "time: " << time << '\n';

}

void lu\_decomosition(vector <vector<double>> &a, long n)

{

for (long k = 0; k < n; k++)

{

for (long i = k + 1; i < n; i++)

{

a[i][k] = a[i][k] / a[k][k];

}

for (long i = k+1; i < n; i++)

for (long j = k + 1; j < n; j++)

{

a[i][j] = a[i][j] - a[i][k] \* a[k][j];

}

}

}

**6. Результаты тестов**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Размер матрицы | Алгоритм (прост) | Алгоритм (усоверш.) |
| 1 | 10 | time: 0.019 ms | time: 0.029 ms |
| 2 | 50 | time: 2.04 ms | time: 3.784 ms |
| 3 | 100 | time: 15.798 ms | time: 30.959 ms |
| 4 | 500 | time: 1890.83 ms | time: 3792.14 ms |
| 5 | 1000 | time: 15242 ms | time: 30370.9 ms |
| 6 | 2000 | time: 148965 ms | time: 275965 ms |
| 7 | 5000 | time: 1.93568e+06 ms | time: 3.27136e+06 ms |
| 8 | 7500 | time: 9.490923e+07 ms | time: 19.6158e+07 ms |
| 9 | 13000 | time: 4.94287e+08 ms | time: 8.087876e+08 ms |
| 10 | 15000 | time: 7.59425e+08 ms | time: 13.56725e+08 ms |

**7. Выводы**

В ходе выполнения лабораторной работы был разработан алгоритм для LU - разложения матрицы. Простой алгоритм Краута оказался эффективнее усовершенствованного. Это связано с тем, что в простом алгоритме выполнение вычислений новых значений U матрицы разбито на 2 операции: вычисление умножения и вычитание из элемента матрицы результата умножения. В то время как эффективный алгоритм не имеет разбиения операций. И для вычисления нового значения элемента необходимо:

1) Разбить команду на простые операции

2) Вычислить значение умножения и записать в какой-то регистр для сохранения результата

3) Вычислить разность между элементом матрицы и результатом умножения, записать в матрицу

Из-за того, что в усовершенствованном алгоритме выделятся дополнительные операции для разбиения команды и сохранения результата, происходит увеличение времени на выполнение операции. Что в конечном счете сказывается на времени выполнении функции. Об это свидетельствуют данные, полученные экспериментальным путем.