МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования   
**«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий, математики и механики**

**Кафедра математического обеспечения и суперкомпьютерных технологий**

Направление подготовки: «Прикладная математика и информатика»

Профиль подготовки: «Вычислительная математика и суперкомпьютерные технологии»

Отчет по лабораторной работе №2

**«Метод сопряженных градиентов для решения СЛАУ с разреженной матрицей»**

**Выполнил:** студент группы 381903-3м

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Панов А.А.

Подпись

**Проверил:**

к.ф.-м. н., доц., доцент каф. МОСТ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Баркалов К.А.

Подпись

Нижний Новгород  
2020 Содержание

[Введение 3](#_Toc41838146)

[1. Постановка задачи 4](#_Toc41838147)

[2. Разреженные матрицы 5](#_Toc41838148)

[2.1 CRS. Сжатое хранение строкой 5](#_Toc41838149)

[2.2 Умножение разреженной матрицы в формате CRS на плотный вектор 6](#_Toc41838150)

[2.2.1 Особенности умножения для симметричной матрицы в формате CRS 7](#_Toc41838151)

[2.2.2 Особенности умножения для параллельной версии 7](#_Toc41838152)

[3. Метод сопряженных градиентов 9](#_Toc41838153)

[3.1 Реализация 9](#_Toc41838154)

[3.2 Структуры данных 10](#_Toc41838155)

[3.3 Основной алгоритм 10](#_Toc41838156)

[4. Вычислительные эксперименты 11](#_Toc41838157)

[5. Заключение 14](#_Toc41838158)

[Список литературы 15](#_Toc41838159)

# 

# Введение

Для решения некоторых задач необходимы решать системы Ax = b, где A разреженная матрицы (значительная часть элементов этой матрицы равны нулю, количество ненулевых элементов в таком случае обычно пропорционально O(n), где n – размер матрицы). Для хранения разреженных матриц используют специальные форматы хранения (позволяющих не хранить нулевые элементы матрицы) и специальные алгоритмы. В данной лабораторной будет рассмотрен один из форматов хранения разреженных матриц (сжатое хранение строкой CSR - compressed sparse row, CRS - compressed row storage) и один из алгоритмов решения СЛАУ (метод сопряженных градиентов).

# Постановка задачи

Реализовать метод сопряженных градиентов для решения СЛАУ с разреженной матрицей, используя технологию OpenMP: Ax = b, где A – разреженная квадратная симметричная положительно определённая матрица, x, b – плотные векторы.

Программа на языке C++ должна реализовывать функцию со следующим заголовком:

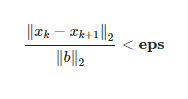
void SLE\_Solver\_CRS(CRSMatrix & A, double \* b, double eps, int

max\_iter, double \* x, int & count);

struct CRSMatrix  
{  
 int n; // Число строк в матрице  
 int m; // Число столбцов в матрице  
 int nz; // Число ненулевых элементов в разреженной симметричной

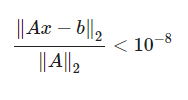
матрице, лежащих не ниже главной диагонали  
 vector<double> val; // Массив значений матрицы по строкам  
 vector<int> colIndex; // Массив номеров столбцов  
 vector<int> rowPtr; // Массив индексов начала строк  
};

Функция получает в аргументах следующие переменные:  
**A** – указатель на структуру CRSMatrix, в которой хранится симметричная матрица A размера n×n в симметричном CRS формате (хранятся только элементы не ниже главной диагонали)  
**b** – указатель на массив, в котором по строкам хранится столбец b размера n×1  
**eps** – критерий остановки:



**max\_iter** – критерий остановки: число итераций больше max\_iter  
**count** – число выполненных итераций алгоритмом.

Ответ считается корректным, если:



Размерность матрицы n ≤ 100000, число ненулевых элементов nz ≤ 107.

# Разреженные матрицы

Обычно матрицу размера NxN называют разреженной, если количество её ненулевых элементов O(N). Но классификации матрицы в первую очередь зависит от её реализации. Например, трех диагональная матрица имеет всего 3N элементов, но для нее выгодней использовать собственную структуру данных, а не одно из представлений разряженной матрицы. Если же матрицы не имеет четкой структуры, то для нее имеет смысл использовать одно из представлений разреженной матрицы.

## CRS. Сжатое хранение строкой

Разреженная матрица A размера n на n, с nz ненулевыми элементами хранится в трех массивах:

1. Массив val – «построчно» хранит значения ненулевых элементов, размер массива nz.
2. Массив colIndexes – хранит номера столбцов для каждого элемента, размер массива nz.
3. Массив rowPtr – хранит индексы, указывающие с какого элемента в массиве val начинается каждая строка. Например, в матрице нулевая строка «пустая», а в первой строке есть 3 ненулевых элемента. Тогда первые три элемента массива row равны 0; 0; 3. Размер массива n + 1, первый элемент всегда равен 0, последний равен nz.

## Умножение разреженной матрицы в формате CRS на плотный вектор

Данная операция понадобится в дальнейшем при реализации метода сопряженных градиентов.

Рассмотрим умножение разреженной матрицы A размера n на n, с nz ненулевыми элементами, в формате CRS, на плотный вектор b длины n.

void mul(const double\* vec, double\* res) const

{

int curIndx = 0;

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

{

const int rowElements = rowPtr[i + 1] - rowPtr[i];

const int endRow = curIndx + rowElements;

for (curIndx; curIndx < endRow; curIndx++)

{

const int j = colIndex[curIndx];

const double vv = val[curIndx];

res[i] += vv \* vec[j];

}

}

return res;

}

### Особенности умножения для симметричной матрицы в формате CRS

Если разреженная матрицы A размера n на n, с nz ненулевыми элементами, является симметричной, то появляется возможность хранить в два раза меньше элементов. Но появляется проблема корректного умножения такой матрицы на вектор. Существует несколько способов корректно выполнить умножение:

1. Дополнить «половину» симметричной матрицы до полного представления. Данный способ требует выделения дополнительной памяти. В итоге матрица будет занимать (2\*nz+n)\*sizeof(elementOfMatrix) байт.
2. Создать транспонированную копию «половинки» симметричной матрицы без главной диагонали. Умножить вектор на транспонированную копию, умножить вектор на оригинал, результаты умножений сложить. Данный способ требует выделения дополнительной памяти. В итоге две половинки матрицы будут занимать (2\*nz + n)\*sizeof(elementOfMatrix) байт.
3. Выполнить корректное умножение «половинки» матрицы на вектор можно добавив к последовательной версии после строки res[i] += vv \* vec[j]; следующий код:

if (j != i)

res[j] += vv \* vec[i]

Данный способ не требует выделения дополнительной памяти (в последовательной версии).

### Особенности умножения для параллельной версии

При распараллеливании внешнего цикла (по i) возможна ситуация, когда несколько потоков одновременно обратятся к res[j] и некорректно выполнят сложение. Чтобы этого избежать достаточно в каждом потоке выделить дополнительный массив tmp длины n и выполнять сложение по j в него. После того, как внешний цикл будет выполнен, нужно прибавить к res элементы из вспомогательных массивов tmp. Данная реализация потребует выделения дополнительной памяти в размере n\*число потоков.

void mul(const double\* vec, double\* res, double\* t) const

{

const int numThreads = omp\_get\_max\_threads();

#pragma omp parallel

{

const int indxThread = omp\_get\_thread\_num();

const int size = n / numThreads;

const int start = indxThread \* size;

int ostatok = 0;

if (indxThread == numThreads - 1)

ostatok = n % numThreads;

const int end = start + size + ostatok;

double \*tmp = t + indxThread \* n;

int elIndx = rowPtr[start];

for (int i = start; i < end; i++)

{

const int endRow = elIndx + (rowPtr[i + 1] - rowPtr[i]);

res[i] = 0.0;

for (elIndx; elIndx < endRow; elIndx++)

{

const int j = colIndexes[elIndx];

const double vv = val[elIndx];

res[i] += vv \* vec[j];

if (j != i)

tmp[j] += vv \* vec[i];

}

}

}

//редукция в res[i]

for (int thr = numThreads - 1; thr >= 0; thr--)

{

double \*tmp = t + thr \* n;

#pragma omp parallel for

#pragma ivdep

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

{

res[i] += tmp[i];

tmp[i] = 0.0;

}

}

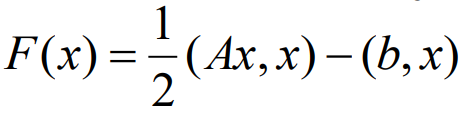
}

};

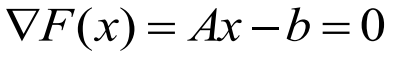
# Метод сопряженных градиентов

Идея метода сопряженных градиентов: решение системы линейных уравнений:

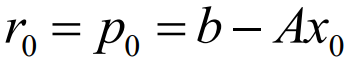
Ax = b, где А – SPD (симметричная, положительно определенная), эквивалентно решению задачи минимизации функции.



Минимум находится из условия:



Предварительный шаг: вычисляются начальный вектор невязки r0 и вектор направления p0:



Основные шаги:

for i=0,…, n do

За критерий «досрочной» остановки можно принять условие:

## Реализация

Самая трудоемкая операция в методе сопряженных градиентов это скалярное умножение. Его реализация для разреженных матриц описана выше. Остальные этапы алгоритма несложно реализовать. Для удобства поверх структуры CRSMatrix была добавлена структура SLECRSMatrix.

## Структуры данных

struct SLECRSMatrix

{

const int &n; // Число строк в матрице

const int &m; // Число столбцов в матрице

const int &nz; // Число ненулевых элементов в разреженной симметричной матрице, лежащих не ниже главной диагонали

const vector<double> &val; // Массив значений матрицы по строкам

const vector<int> &colIndexes; // Массив номеров столбцов

const vector<int> &rowPtr; // Массив индексов начала строк

SLECRSMatrix(const CRSMatrix &matr);

void mul(const double\* vec, double\* res, double\* t) const

}

Код функции mul приведен выше.

## Основной алгоритм

Основной этап выглядит довольно просто в соответствии с формулами метода сопряженных градиентов.

for (count = 0; count < max\_iter; count++)

{

// alpha\_i

mA.mul(&p[0], &Ap[0], &tmp[0]);

alpha = scalar(&r0[0], &r0[0], n) / scalar(&Ap[0], &p[0], n);

// копируем в x0 "предыдущий" ответ

// x\_i = x\_i+1

copy\_ar(&x[0], &x0[0], n);

// x\_i+1 = …

addVector(&x[0], &p[0], alpha, &x[0], n);

// r\_i+1 = …

subtractVector(&r0[0], &Ap[0], alpha, &r1[0], n);

// beta\_i = …

beta = scalar(&r1[0], &r1[0], n) / scalar(&r0[0], &r0[0], n);

// p\_i+1 = …

addVector(&r1[0], &p[0], beta, &p[0], n);

// r\_0 = r\_i

copy\_ar(&r1[0], &r0[0], n);

double error = gerError(&x0[0], x, b, n);

if (error < eps)

{

count++;

return;

}

}

# Вычислительные эксперименты

Метод сопряженных градиентов выполнялся для симметричной, положительно определенной пяти диагональной матрицы A размера (итого ненулевых элементов). Так как матрица симметричная хранились только элементы не ниже главной диагонали. Благодаря использованию формата CRS всего хранилось 3n+3n+n=7n элементов. Для проверки корректности выполнялось вычисление невязки (Ax-b).

Запуск производился на восьми ядерном процессоре Intel core i7 9700K, 16 gb ОЗУ. Использовался Intel® C++ Compiler 19.0 for Windows\* с оптимизацией O2.

Размер пяти диагональной матрицы n=18000, число итераций 80000.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Время, сек. | 1 core | 2 core | 4 core | 6 core | 8 core |
| n = 18000  num\_it=80000 | 20 | 10.8 | 6.2 | 5.1 | 4.7 |

Таблица 1. Время работы метода сопряженных градиентов.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Время, сек. | 1 core | 2 core | 4 core | 6 core | 8 core |
| n = 18000  num\_it=80000 | 1 | 0,93 | 0,81 | 0,65 | 0,53 |

Таблица 2. Масштабируемость метода сопряженных градиентов.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Время, сек. | 1 core | 2 core | 4 core | 6 core | 8 core |
| n = 18000  num\_it=80000 | 1 | 1,85 | 3,23 | 3,92 | 4,26 |

Таблица 3. Ускорение метода сопряженных градиентов.

# Заключение

Численный эксперимент показал, что на 4 ядрах алгоритм имеет неплохую масштабируемость в 81%. При увеличении числа ядер ускорение продолжает расти (4.26 на 8 ядрах), но масштабируемость падает до 53%. В целом, распараллеливание алгоритма сопряженных градиентов может принести значительное ускорение.

# Список литературы

1. Баркалов К.А. Параллельные численные методы.
2. Самарский А. А. Введение в численные методы. – Лань, 2009.