Санкт-Петербургский политехнический университет
Высшая школа прикладной математики
и вычислительной физики, ФизМех

Направление подготовки
«Прикладная математика и информатика»

Отчет по лабораторной работе №3
«Решение СЛАУ итерационными методами»

Выполнил студент гр. 5030102/00003 Петрошенко А.В.

Преподаватель: Курц В.В.

Санкт-Петербург
2021

Формулировка задачи и ее формализация

Большинство расчетных математических задач сводится к решению системы линейных алгебраических уравнений(далее СЛАУ). Существует 2 класса методов решения таких СЛАУ:

1. Прямые методы - методы, которые находят «точные» значения неизвестных за конечное число операций.
2. Итерационные методы - методы, которые строят последовательность векторов, сходящихся к решению.

В этой работе мы будем использовать итерационный метод.

Постановка задачи:
Пусть дана система из n линейных уравнений с n неизвестными:

$$\sum_{j=1}^{n} a_{ij}x_j = b_i,\ i = 1, \dots, n$$

где x_j - неизвестные, a_{ij} - коэффициенты системы и b_i - компоненты вектора правой части.
В матричной форме:

$$Ax = b$$

где $A = (a_{ij}) \in \mathbb{R}^{n \times n}$ - матрица коэффициентов, $b = (b_i) \in \mathbb{R}^n$ - вектор правой части и $x = (x_j) \in \mathbb{R}^n$ - вектор неизвестных.
Требуется найти решение с точностью ϵ, то есть $||x^k - x^*|| < \epsilon$
В данной работе будет реализован и описан Градиентный метод.

Алгоритм метода и условия его применимости

Определим квадратичную форму:

$$F(y) = (Ay, y) - 2(b, y)$$

Каждое следующее приближение будем получать, двигаясь в направлении противоположном градиенту квадратичной формы, введенной выше:

$$x^{k+1} = x^k - \alpha_k \nabla F(x^k), \alpha_k > 0$$

$$\nabla F(x^k) = 2(Ax^k - b) = g^k$$

Причем α_k нужно выбирать так, чтобы $F(x^k) \to min$
Из теоремы о том, что x^* является решением системы $\Leftrightarrow x^*$ сообщает минимум квадратичной формы $F(y)$ мы знаем, что:

$$\phi'(\alpha_k) = 0 \Rightarrow \alpha_k = \frac{(g^k, g^k)}{2(Ag^k, g^k)}$$

Останавливаться будем по условию:

$$||x^{k+1} - x^*||_{H_A} \leq \frac{M}{1-M}||x^{k+1} - x^k||_{H_A} \Leftrightarrow ||x^{k+1} - x^k||_{H_A} \geq \frac{1-M}{M}\epsilon$$

Данное условие было получено из неравенства $||e^{k+1}||_{H_A} \leq M||e^k||_{H_A}$, где $M = \frac{cond(A)-1}{cond(A)+1}$
Также стоит добавить, что $||v||_{H_A} = (Av, v)$
Условия применимости:
Квадратная матрица A должна быть симметричной и положительно определенной.

Предварительный анализ задачи

Построение матрицы:

Любую симметричную матрицу можно разложить в виде $A = QDQ^T$, где матрица Q - это ортогональная матрица, а D - диагональная матрица, на диагонали которой стоят только положительные числа. Т.к. они являются собственными числами матрицы A, то из данного разложения следует, что она будет и положительно определенной.

Тестовый пример для задач малой размерности

Возьмем ортогональную матрицу: $Q = \begin{pmatrix} -0.9812 & 0.1729 & 0.0854 \\ -0.1762 & -0.6237 & -0.7615 \\ -0.0784 & -0.7623 & 0.6425 \end{pmatrix}$, $D = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix}$

Тогда $A = \begin{pmatrix} 1.0444 & -0.2379 & -0.0221 \\ -0.2379 & 2.5487 & -0.5031 \\ -0.0221 & -0.5031 & 2.4068 \end{pmatrix}$

Рассмотрим СЛАУ: $\begin{cases} 1.0444x_1 - 0.2379x_2 - 0.0221x_3 = 0.7844 \\ -0.2379x_1 + 2.5487x_2 - 0.5031x_3 = 1.8077 \\ -0.0221x_1 - 0.5031x_2 + 2.4068x_3 = 1.8816 \end{cases}$

Ее точное решение: $x^* = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$

Возьмем $x^0 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$ и найдем решение с точностью $\epsilon = 0.1$:

Комментарий: Для сокращения вычислений, сократим 2-ки в формулах g^k и α_k

Из матрицы D получаем, что $cond(A) = 3 \Rightarrow M = 0.5 \Rightarrow \frac{1-M}{M} = 1$

1. $g^0 = -b = \begin{pmatrix} -0.7844 \\ -1.8077 \\ -1.8816 \end{pmatrix}$
   $\alpha_0 = 0.5569$
   $x^1 = x^0 - \alpha_0 g^0 = \begin{pmatrix} 0.4368 \\ 1.0067 \\ 1.0479 \end{pmatrix}$
   $||x^1 - x^0||_{H_A} = 2.0333 \geq 0.1$
2. $g^1 = \begin{pmatrix} -0.5908 \\ 0.1270 \\ 0.1243 \end{pmatrix}$
   $\alpha_1 = 0.8170$
   $x^2 = \begin{pmatrix} 0.9195 \\ 0.9029 \\ 0.9463 \end{pmatrix}$
   $||x^2 - x^1||_{H_A} = 0.5577 \geq 0.1$
3. $g^2 = \begin{pmatrix} -0.0598 \\ -0.2012 \\ -0.0786 \end{pmatrix}$
   $\alpha_2 = 0.5026$
   $x^3 = \begin{pmatrix} 0.9496 \\ 1.0041 \\ 0.9858 \end{pmatrix}$
   $||x^3 - x^2||_{H_A} = 0.1589 \geq 0.1$

4. $g^3 = \begin{pmatrix} -0.0533 \\ 0.0295 \\ -0.0351 \end{pmatrix}$
   $\alpha_3 = 0.5015$
   $x^4 = \begin{pmatrix} 0.9763 \\ 0.9893 \\ 1.0034 \end{pmatrix}$
   $||x^4 - x^3||_{H_A} = 0.0498 < 0.1$

Мы получили ответ с заданной точностью за 4 итерации, не считая начальное прриближение.

Контрольные тесты

1. Создадим 4 матрицы 10×10 с числами обусловленности $10, 10^2, 10^3, 10^4$ и найдем их решения с помощью градиентного метода с точностью $\epsilon = 10^{-10}$
2. Создадим 101 матрицу, меняя ее ранг с 1000×1000 до 2000×2000 с шагом в 10. Найдем решение с помощью метода Холецкого, получим ошибку и затем найдем решение с помощью градиентного метода с точностью, равной полученной ошибке, но умноженной на 10 для решения проблемы с машинным эпсилон. Для каждого из методов будем засекать время их выполнения, после чего сравним их.

Модульная структура программы

```
typedef struct{
     vector<vector<double>> A;
     int rang;
}matrix_t;
```
- структура данных, имеющая 2 поля: двумерный массив для значений матрицы и целое число для хранения ранга матрицы.

`int GetNum(const char* filename)` - функция для получения одного числа из файла. Использовалась для получения из файла ранга матриц и их количества.

```
matrix_t ImportMatrix(vector<double> str, int rang)
vector<double> ImportRightPart(vector<double> str, int rang)
```
Функции, преобразующие данные, полученные из файла в удобный вид(матрицу или вектор).

```
matrix_t Zero(int rang)
matrix_t Transpose(matrix_t matrix)
matrix_t CholFactorization(matrix_t matrix)
vector<double> FindY(matrix_t matrix, vector<double> b)
vector<double> FindX(matrix_t matrix, vector<double> y)
```
Реализация метода Холецкого(функции описаны в предыдущем отчете)

```
vector<double> RandomVector(int size)
vector<double> VectorSubstract(vector<double> vec1, vector<double> vec2)
vector<double> MatrixMulVector(matrix_t matrix, vector<double> vec)
vector<double> VectorMulNumber(vector<double> vec, double num)
double ScalarProduct(vector<double> vec1, vector<double> vec2)
double Norm(vector<double> vec)
double EnergyNorm(vector<double> vec, matrix_t A)
vector<double> Gradient(matrix_t A, vector<double> x, vector<double> b)
double AlphaCoefficient(vector<double> g, matrix_t A)
```
Вспомогательные функции для градиентного метода(названия функций передают их смысл)

```
vector<vector<double>> GradientMethod(matrix_t A, vector<double> b, double
cond, double eps)
```

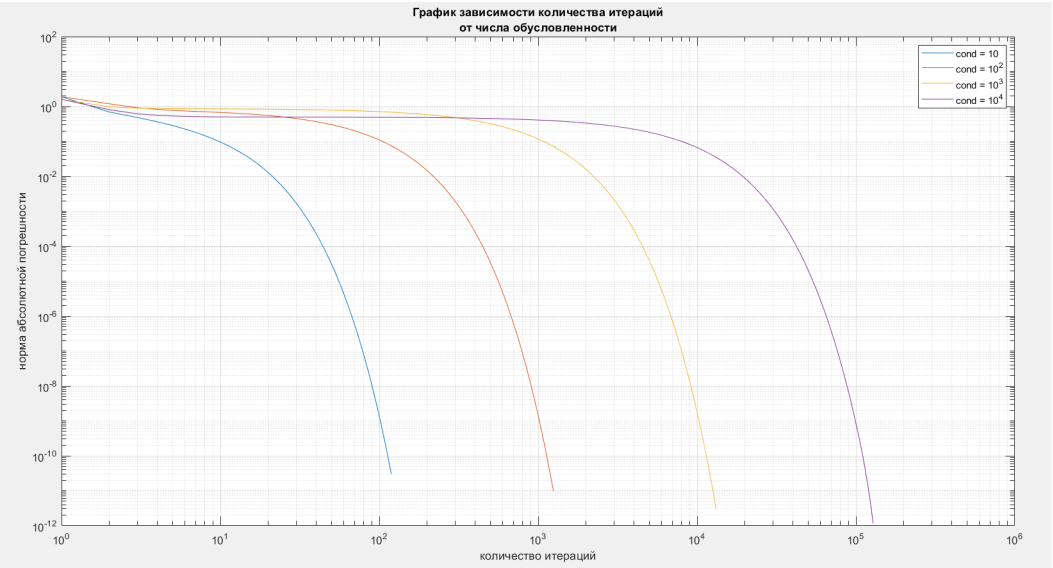
Реализация градиентного метода

```
void OutputVector(vector<double> vec, const char* filename)
void OutputMatrix(matrix_t matrix, const char* filename)
```

Функции записи нужных для дальнейшего анализа данных в файл.

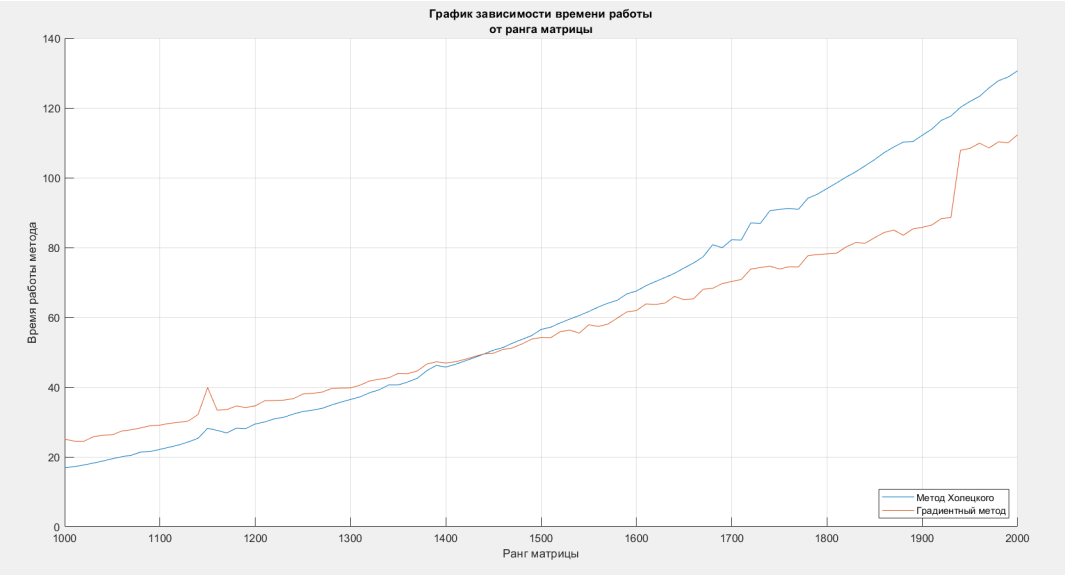
Численный анализ

▷ Точность:



Из графика мы видим, что поставленная точность достигается во всех случаях($\epsilon = 10^{-10}$). Видна зависимость от числа обусловленности: чем оно больше, тем дольше сходится метод, но и тем точнее решение.

▷ Объем вычислений:



На графике явно видно, что, начиная примерно с матрицы 1500×1500 метод Холецкого становится дольше градиентного метода, что связано с различием в принципах работы прямых и итерационных методов. Делаем вывод, что итерационные методы лучше применять на больших матрицах, когда прямые на матрицах меньшей размерности.

Общие выводы:

В данной лабораторной работе мы научились находить решения СЛАУ градиентным методом. Проанализировали зависимость работы метода от числа обусловленности матрицы и сравнили время работы прямого метода(метода Холецкого) и итерационного(градиентного метода). Различие в принципах работы прямых и итерационных методов состоит в том, что прямые методы приводят исходную матрицу к удобному для нахождения решения виду, на что тратят много времени при больших размерах матрицы, когда итерационные методы, в свою очередь, не меняют исходную матрицу, а сразу начинают приближение к решению, за счет чего и выигрывают на больших матрицах.