МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Отчет по Лабораторной работе №3 «Итерационные методы решения СЛАУ»

Вариант 23

Подготовила:
Врублевская Екатерина Александровна,
2 курс 13 группа

2021

1. Постановка задачи

Итерационные методы для разреженных СЛАУ особого вида

1. Написать программу, которая при данном n решает СЛАУ $A_n x = b_n$ указанным в варианте методом. Здесь A_n — разреженные матрицы размерности n из списка 2 (см. ниже), указанные в варианте.
   - Матрицу следует либо хранить в одном из форматов для разреженных матриц, либо сразу реализовать итерационный метод, учитывая известную структуру матрицы. Хранить в памяти матрицу A_n целиком со всеми нулями запрещено!
   - Вектор b_n выбирать таким образом, чтобы он соответствовал некоторому заранее заданному решению.
   - Критерий остановки итераций: $\|A_n x^k - b_n\| < \varepsilon$.
2. Подтвердить правильность работы программы на примере нескольких СЛАУ размерности 5-10.
3. Построить диаграмму сходимости (общую) для n = 100, 1000, 10000.
4. Построить диаграмму, в которой по оси абсцисс изменяется n = $[10^{k/2}]$, k=1, …,12, а на оси ординат отложено время работы, которое требуется, чтобы норма невязки не превышала 10^{-8}].

Метод: Симметричный метод Гаусса-Зейделя

Матрица:

Матрицы вида $A_n =$

$$\begin{pmatrix} d & 0 & e & & & & \\ 0 & d & 0 & e & & & \\ c & 0 & d & 0 & e & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & c & 0 & d & 0 & e \\ & & & c & 0 & d & 0 \\ & & & & c & 0 & d \end{pmatrix},$$

Где $c = -1, d = 2, e = 100$

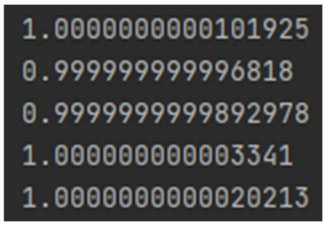
2. Основная часть

1. Так как для матрицы, данной по условию итерационный процесс не сходится (так как спектр матрицы не меньше 1), для тестирования возьмем матрицы такого же вида, но с одним изменением: d = 105. Теперь матрица обладает свойством диагонального преобладания и итерационный процесс будет сходиться при любом начальном приближении.
2. Проверяю правильность работы для СЛАУ разных размерностей:

n=5:

Берем вектор $b_n = (205, 205, 204, 104, 104)^T$, для него вектор $x = (1, 1, 1, 1, 1)^T$.

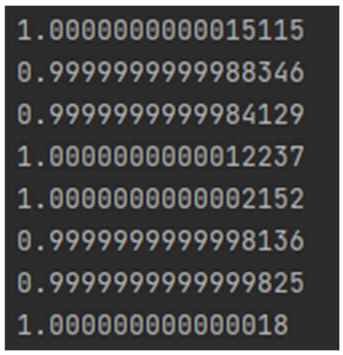
Полученный результат:



n = 8:

Берем вектор $b_n = (205, 205, 204, 204, 204, 204, 104, 104)^T$, для него вектор $x = (1, 1, 1, 1, 1, 1, 1, 1)^T$.

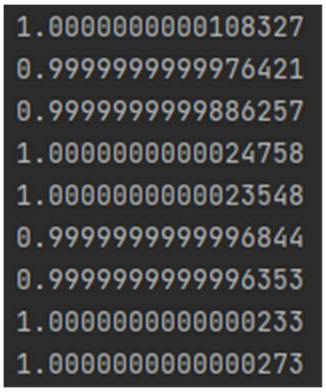
Полученный результат:



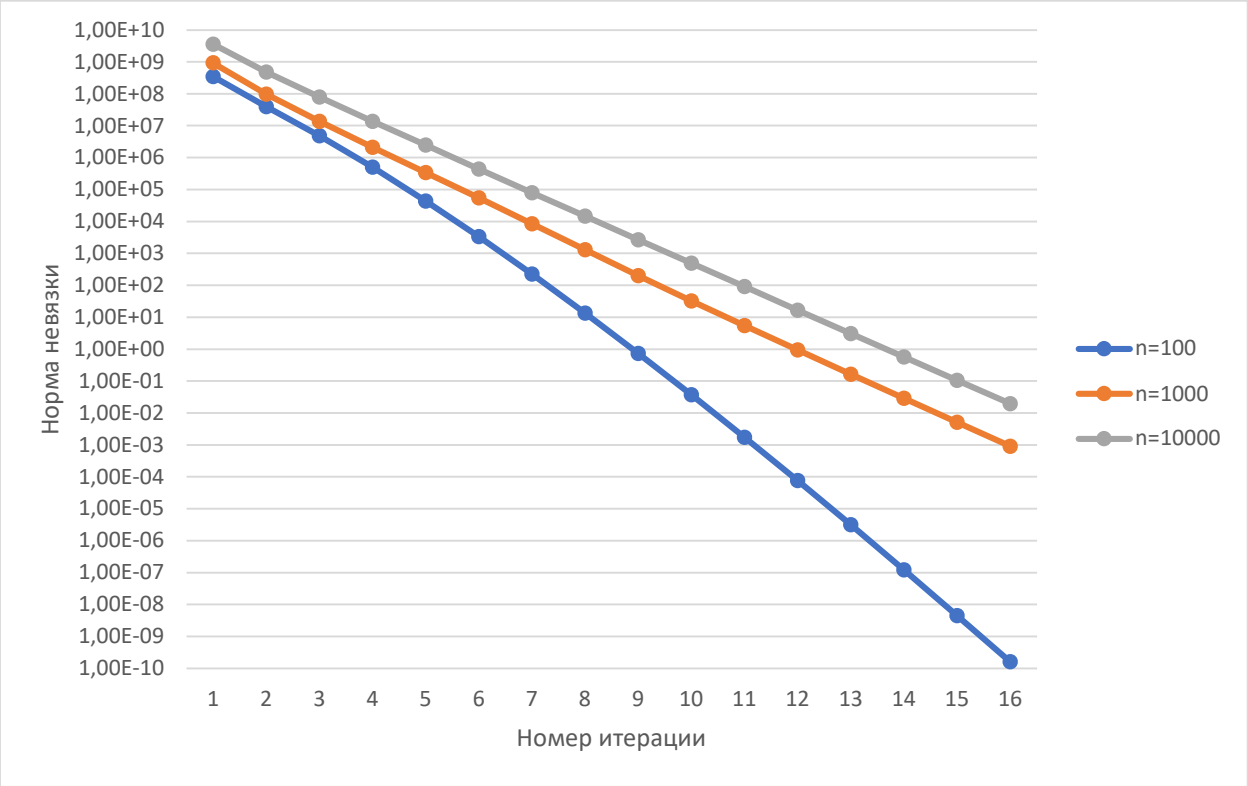
n = 9:

Берем вектор $b_n = (205, 205, 204, 204, 204, 204, 204, 104, 104)^T$, для него вектор $x = (1, 1, 1, 1, 1, 1, 1, 1, 1)^T$.

Полученный результат:

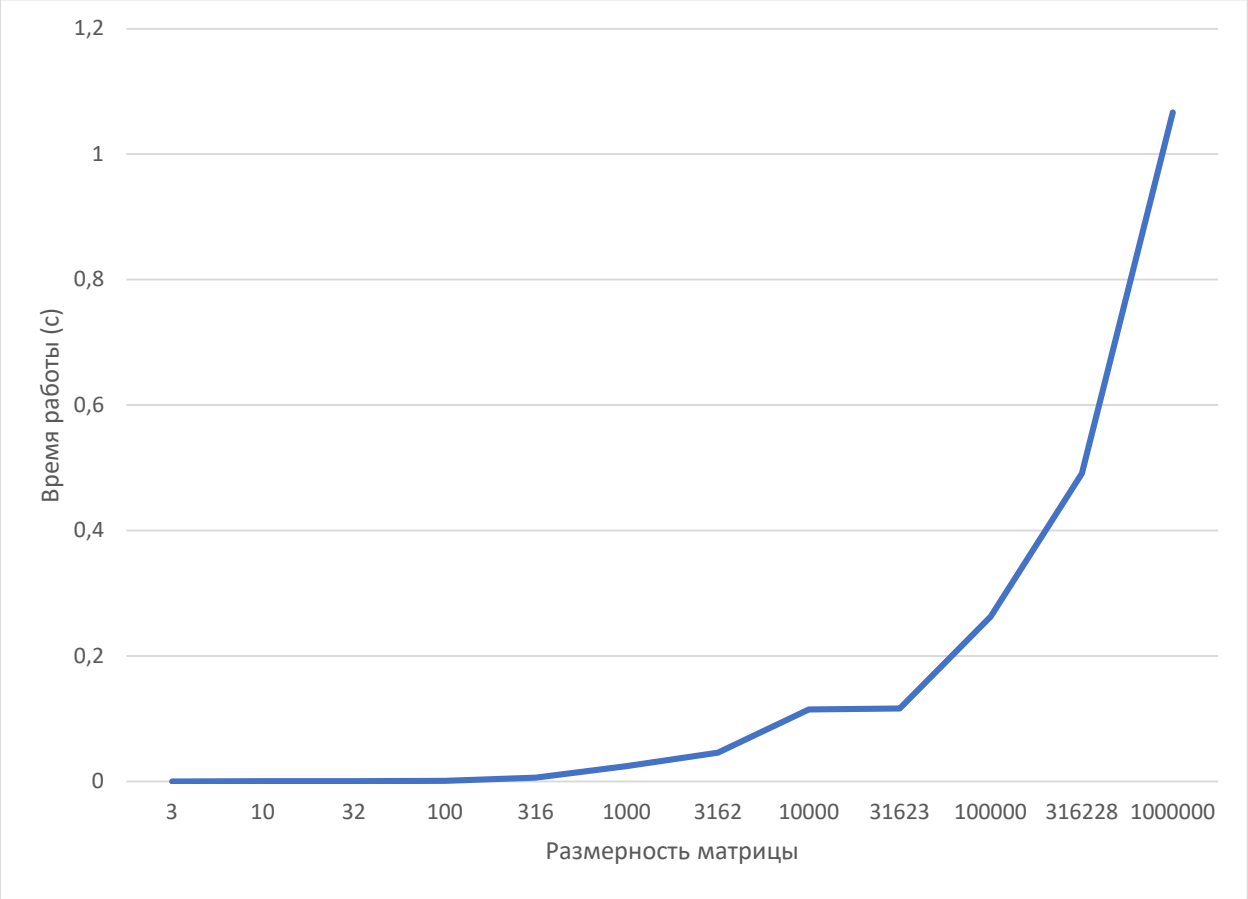


3. Диаграмма сходимости



Как мы видим, во всех трех случаях к четвертой итерации уже значительно сокращается норма невязки между полученным и точным решением.

4. Диаграмма зависимости времени работы от размерности матрицы



Нетрудно заметить, что время выполнения итерационного процесса резко возрастает лишь начиная с n=31623, но все равно даже при n=1000000 время работы не превышает 1,5 секунд.

Листинг кода программы:

```
public class GaussSeidel {
    int n;
    @Setter
    double[] x_k;
    @Setter
    double[] elements;
    @Setter
    double[] b;
    @Setter
    double eps;

    public GaussSeidel(int n, double eps) {
        this.n = n;
        this.eps = eps;
        this.elements = new double[]{-1, 2, 100};
    }

    private boolean nearAnswer(double[] Ax, double eps) {
        double norm = 0;
        for (int i = 0; i < n; i++) {
            norm += Math.pow(Ax[i] - b[i], 2);
        }
        return (Math.sqrt(norm) < eps);
    }
```

```
    private double[] calcAx(double[] x) {
        double[] res = new double[n];
        for (int i = 0; i < n; i++) {
            res[i] = x[i] * elements[1];
            if (i < n - 2)
                res[i] += x[i + 2] * elements[2];
            if (i > 1)
                res[i] += x[i - 2] * elements[0];
        }
        return res;
    }

    public double[] count() {
        do {
            double sum;
            for (int i = 0; i < n; i++) {
                sum = 0;
                if (i < n - 2)
                    sum += x_k[i + 2] * elements[2];
                if (i > 1)
                    sum += x_k[i - 2] * elements[0];

                x_k[i] = (b[i] - sum) / elements[1];
            }
            for (int i = n - 1; i >= 0; i--) {
                sum = 0;
                if (i < n - 2)
                    sum += x_k[i + 2] * elements[2];
                if (i > 1)
                    sum += x_k[i - 2] * elements[0];

                x_k[i] = (b[i] - sum) / elements[1];
            }
        } while (!nearAnswer(calcAx(x_k), eps));
        return x_k;
    }
private void printNevyazka() {
    double norm = 0;
    int i = 0;
    for (var coord : x_k) {
        norm += Math.pow(coord - x[i], 2);
        i++;
    }
    System.out.println(Math.sqrt(norm));
}

}
```

```
public class App {
    private static int n = 9;
    public static void main(String[] args) {
        GaussSeidel gaussSeidel = new GaussSeidel(n, 0.00000001);
        var x0 = new double[n];
        Random random = new Random();
        for (int i = 0; i < n; i++) {
            x0[i] = random.nextInt();
        }
        var b = new double[n];
        for (int i = 0; i < n; i++) {
            b[i] = 204;
        }
        var x = new double[n];
```

```
        for (int i = 0; i < n; i++) {
            x[i] = 1;
        }
        b[0] = b[1] = 205;
        b[n-2] = b[n-1] = 104;
        gaussSeidel.setB(b);
        gaussSeidel.setX(x);
        gaussSeidel.setX_k(x0);
        var result= gaussSeidel.count();
        Arrays.stream(result).forEach(System.out::println);
    }
```

5. Заключение

Симметричный метод Гаусса-Зейделя (ГЗ) представляет собой композицию обычного и обратного методов ГЗ. То есть, для вычисления x^{k+1} сначала находится промежуточное приближение $x^{k+\frac{1}{2}}$ стандартным методом ГЗ, а затем, начиная с $x^{k+\frac{1}{2}}$ выполняется одна итерация обратного метода ГЗ (в котором обновление компонент осуществляется в обратном порядке, с n-й по 1-ю). Полученный результат и будет новым приближением x^{k+1}.

Я реализовала метод Гаусса-Зейделя сразу учитывая известную структуру матрицы. Так как в матрице существует всего 3 вида значений (c, d, e), в памяти будем хранить ее как массив из трех чисел. И так как на каждой итерации для подсчета нового приближения x^{k+1} мы пересчитываем значения всех его компонент и используем в этом пересчете уже полученные на этой итерации, так как они более точные, мы записываем их сразу же в вектор приближений x_k, чтобы потом использовать при вычислениях уже самые свежие значения. Также на одной итерации мы сначала считаем приближение по методу Гаусса-Зейделя, а затем делаем то же самое обратным методом Гаусса-Зейделя (то есть сначала обновляем последовательно компоненты 1-n, а затем уже начиная с n-й и заканчивая первой). А обновляем i-ю компоненту вектора приближений x_k по следующей формуле:

$$x_i^{k+\frac{1}{2}} = \frac{1}{a_{ii}}(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+\frac{1}{2}} - \sum_{j=i+1}^{n} a_{ij} x_j^k)$$

И дальше обратный метод ГЗ:

$$x_i^{k+1} = \frac{1}{a_{ii}}(b_i - \sum_{j=i+1}^{n} a_{ij} x_j^{k+1} - \sum_{j=1}^{i-1} a_{ij} x_j^{k+\frac{1}{2}})$$

Но так как мы храним все уже обновленные компоненты в том же самом векторе, в этих формулах две суммы объединяются в одну: сумму по всем j≠i.