МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ Н. Э. БАУМАНА
Факультет информатики и систем управления
Кафедра теоретической информатики и компьютерных технологий

Лабораторная работа
по курсу «Численные методы»

«Исследование ускорения численных итерационных методов на примере Чебышевского ускорения »

Выполнил:
студент группы ИУ9-72
Беляев А. В.

Проверил:
Голубков А. Ю.

Москва 2017

Содержание

1 Постановка задачи

Исследование проблемы решения СЛАУ на примере итерационных методов решения с использованием Чебышевского ускроения и Чебышевской итерации.

Исследование проблем собственных значений разреженной матрицы с использованием Чебышевского ускорения и Чебышевской итерации.

2 Необходимые теоретические сведения

2.1 Итерационные методы для линейных систем

Существуют различные численные методы решения СЛАУ. Большинство из них можно разделить на прямые и итерационные.

Итерационный метод строит последовательность приближений к точному решению СЛАУ, сходясь к нему с увеличением числа шагов.

Однако сходимость не гарантирует возможности практического применения метода. Для подобной оценки вводят понятие *скорости сходимости* итерационного метода. Это число приближений, которые должны быть построены методом для достижения критерия останова.

При работе с итерационными методами часто используют приемы ускорения сходимости, такие как предобуславливание или посторение рекурентных соотношений с использованием ранее найденных значений.

Рассмотрим систему $Ax = b$. Приведя ее к виду

$$x = Gx + c$$

заметим, что выражение может быть вычислено с использованием рекуррентного соотношения, основанного на методе простой итерации:

$$x^{(s+1)} = Gx^{(s)} + c$$

Опустим этот метод, так как он не представляет интереса в текущей работе.

2.1.1 Метод верхней релаксации, SOR

Рассмотрим метод верхней релаксации (SOR, Successive Over Relaxation). Для этого предварительно представим матрицу A в виде

$$A = L + D + R,$$

где D - диагональная матрица, а L и R – нижне- и верхнетреугольные матрицы с нулевой главной диагональю.

Последовательная верхняя релаксация основана на методах Якоби и Гаусса-Зейделя и является более быстрым методом.

Необходимым условием сходимости метода релаксации является выполнение условия $\omega \in (0, 2)$. Если $\omega \in (0, 1)$, то говорят о *методе нижней релаксации*, если

$\omega \in (1,2)$ – о *методе верхней релаксации.* При $\omega = 1$ метод SOR совпадает с *методом Зейделя.* Приведем схему вычисления i-го компонента:

$$x_i^{(s+1)} = (1-\omega)x_i^{(s)} + \frac{\omega}{a_{ii}}\Big(b_i - \sum_{j<i} a_{ij}x_j^{(s+1)} - \sum_{j>i} a_{ij}x_j^{(s)}\Big), i = 1,2,\ldots,n.$$

2.1.2 Чебышевское ускорение методом Чебышевской итерации

Чебышевское ускорение итерационного процесса $x^{(s+1)} = Gx^{(s)} + c$ состоит в следующем:

1. Положить $\mu_0 = 1$; $\mu_1 = \rho$; $y^{(0)} = x^{(0)}$; $y^{(1)} = Gx^{(0)} + c$

2. Для всех последующих m=2,3, ... вычислять

$$\mu_m = \Big(\frac{2}{\rho\mu_{m-1}} - \frac{1}{\mu_{m-2}}\Big)^{-1}$$

$$y^{(m)} = \frac{2\mu_m}{\rho\mu_{m-1}}\Big(Gy^{(m-1)} + c\Big) - \frac{\mu_m}{\mu_{m-2}}y^{(m-1)}$$

Чебышевское ускорение требует, чтобы собственные значения матрицы G были вещественны и принадлежали отрезку $[-\rho,\rho]$. Однако ситуацию можно исправить с помощью метода *симметричной верхней рлеаксации* .

2.1.3 Метод симметричной верхней релаксации, SSOR

В общем случае в методе SOR матрица не является симметричной и в спектре могут содержаться комплексные значения. Ввиду этих ограничений вводят метод SSOR. Приведем схему вычисления:

1. Выполняем ход SOR, вычисляя компоненты вектора $x^{s+1/2}$ в прямом порядке возрастания $x_1^{s+1/2}, x_2^{s+1/2}, \ldots, x_n^{s+1/2}$:

$$x_i^{(s+1/2)} = (1-\omega)x_i^{(s)} + \frac{\omega}{a_{ii}}\Big(b_i - \sum_{j<i} a_{ij}x_j^{(s+1/2)} - \sum_{j>i} a_{ij}x_j^{(s)}\Big), i = 1,2,\ldots,n.$$

2. Выполняем ход SOR, вычисляя компоненты вектора x^{s+1} в обратном порядке возрастания $x_n^{s+1}, x_{n-1}^{s+1}, \ldots, x_1^{s+1}$:

$$x_i^{(s+1)} = (1-\omega)x_i^{(s+1/2)} + \frac{\omega}{a_{ii}}\Big(b_i - \sum_{j<i} a_{ij}x_j^{(s+1/2)} - \sum_{j>i} a_{ij}x_j^{(s+1)}\Big), i = n,\ldots,1.$$

2.2 Проблемы собственных значений разреженных матриц

Проблемой собственных значений матрицы является проблема поиска ее собственных значений и соответствующих им векторов. *Частичной* проблемой собственных значений является поиск максимального или минимального из них.

Пусть A – несимметричная матрица размерности N и рассматривается частичная проблема собственных значений:

$$Au = \lambda u.$$

Пусть $\lambda_1, \ldots, \lambda_n$ - упорядоченные по убыванию абсолютных величин, собственные значения матрицы, и нас интересует λ_1 – максимальное собственное значение.

2.2.1 Метод степенной итерации

Описанная выше проблема может быть решена с помощью алгоритма *степенной итерации*, который для матрицы A и некоторого начального приближения $y^{(0)}$ строит итерационный процесс $y^{(k)} = A^k y^{(0)}$, из которого можно получить $\lambda_{max} = \lambda_1$.

Более формально:

1. Выбрать вектор начального приближения $y^{(0)} \in R^n$
2. Для $k = 1, 2, \ldots$ выполнить:

$$z^{(k)} = Ay^{(k-1)}$$

$$\lambda^{(k)} = \frac{z^{(k)}}{y^{(k-1)}}$$

$$y^{(k)} = z^{(k)}/||z^{(k)}||$$

3. Критерий останова:

$$\epsilon^{(k)} = |\lambda_1^{(k)} - \lambda_1^{(k-1)}| \leq \epsilon,$$

где ϵ - заданная вычислительная точность расчета.

2.2.2 Метод Чебышевской итерации

Метод степенной итерации обладает низкой скоростью сходимости. Для усорения сходимости используют метод *Чебышевского ускорения* , состоящий в замене степенной итерации на Чебышевскую итерцию.

Рассмотрим итерацию формы $z_n = p_n(A)z_0$, где z_0 - некоторое стартовое приближение, а p_n - полином степени n. В таком случае понадобится выбрать p_n таким образом, чтобы вектор z_n сходился к вектору собственных значений матрицы A, ассоциированному с λ_1. В качестве p_n в таком случае выбирают полином

$$p_n(\lambda) = \frac{C_n[(\lambda - d)/c]}{C_n[(\lambda_1 - d)/c]},$$

где $d, d \pm c$ – центр и фокусы эллипса $E(d, c, a)$, содержащего набор собственных значений $R = \{\lambda_2, \ldots, \lambda_N\}$, "не интересующих" нас, а C_n есть Чебышевский полином степени n, определяемый соотношением

$$C_1(\lambda) = \lambda; C_0(\lambda) = 1,$$

$$C_{n+1}(\lambda) = 2\lambda C_n(\lambda) - C_{n-1}(\lambda).$$

Произведя замену $\sigma_{n+1} = \rho_n/\rho_{n+1}$ и несколько преобразований, запишем алгоритм:

1. Выбрать стартовый вектор z_0 и вычислить

$$\sigma_1 = c/(\lambda_1 - d); z_1 = \frac{\sigma_1}{c}(A - dI)z_0.$$

2. Начиная с $n = 1, 2, \ldots$ и до схождения вычислять:

$$\sigma_{n+1} = \frac{1}{2/\sigma_1 - \sigma_n},$$

$$z_{n+1} = 2\frac{\sigma_{n+1}}{c}(A - dI)z_n - \sigma_n\sigma_{n+1}z_{n-1}.$$

2.3 Способ представления разреженных матриц

Для хранения и представления разреженных матриц будет использоваться алгоритм CSR (Compressed Sparse Row), хранящий не все элементы матрицы, а лишь ненулевые из них, а также их координаты в матрице.

3 Текст программы

Для написания программы был использован язык Java.

```
public float[] performSOR() {
    float[] prevX = new float[x.length];
    float sum;
    float diagonal = 1f;

    int sorCounter = 0;
    while (sorCounter < maxIterations && convergence()) {

        for (int i = 0; i < matrixSize; i++) {

            sum = 0f;
            for (int j = m.iptr[i]; j < m.iptr[i + 1]; j++) {
                int jIndex = m.jptr[j];
                float value = m.elem[j];

                if (jIndex < i) {
                    sum += value * x[jIndex];
                } else if (jIndex > i) {
                    sum += value * prevX[jIndex];
                } else {
                    diagonal = value;
                }
            }
            sum = (b[i] - sum) / diagonal;
            x[i] = omega*sum  + (1 - omega)*prevX[i];
        }
        prevX = x.clone();
        sorCounter++;
    }
    return x;
}

public float[] performSSOR() {
    float[] prevX = new float[x.length];
    float[] semiX = new float[x.length];
    float sum;
    float diagonal = 1f;

    int sorCounter = 0;
    while (sorCounter < maxIterations && convergence()) {

        //1st semi iter
        for (i = 0; i < matrixSize; i++) {
            sum = 0f;

            // ...

            sum = (b[i] - sum) / diagonal;
            semiX[i] = prevX[i] + omega * (sum - x[i]);
        }

        //2nd semi iter
        for (k = matrixSize - 1; k >= 0; k--) {

            // ...

            sum = (b[k] - sum) / diagonal;
            x[k] = semiX[k] + omega * (sum - semiX[k]);
        }
        prevX = x.clone();
```

```
            sorCounter++;
        }
        return x;
    }

    public float[] performChebyshev() {
        float mu0 = 1;
        float mu1 = ro;
        float[] ym2 = new float[matrixSize];
        float[] y1 = new float[matrixSize];
        float[] ym1 = new float[matrixSize];
        float[] ym = new float[matrixSize];
        float[] tmp;

        float mu;
        float prevMu = mu1;
        float prevPrevMu = mu0;

        performSingleStepSSOR(ym2, y1);

        int chebyshevIter = 0;
        while (chebyshevIter < maxIterations && convergence()) {

            mu = 1f / (2f/(ro * prevMu) - 1f/prevPrevMu);

            performSingleStepSSOR(y1, ym1);
            float k1 = 2f * mu / (ro * prevMu);
            float k2 = mu / prevPrevMu;

            for (int j = 0; j < matrixSize; j++) {
                ym[j] = k1 * ym1[j] - k2 * ym2[j];
            }
            prevMu = mu;
            prevPrevMu = prevMu;

            tmp = y1.clone();
            ym2 = y1.clone();
            y1 = ym.clone();
            ym = tmp.clone();

            chebyshevIter++;
        }
        return ym;
    }

    public float[] performPowerIteration() {

        do {
            yk1 = Arrays.copyOf(zk, zk.length);

            zk = m.multiplyByVector(yk1);

            prevEig = Arrays.copyOf(eigenVector, eigenVector.length);
            for (int j = 0; j < eigenVector.length; j++) {
                eigenVector[j] = zk[j] / yk1[j];
            }
            Common.normalizeVector(zk);

            iterCount++;

        } while (Math.abs(eigenVector[0] - prevEig[0]) > accuracy);

        return eigenVector;
    }

    public float[] performEigenChebyshev() {
        float sigmaN;
        float[] z0;
        float[] zN1;

        float sigma1 = c / (eigenValue[0] - d);

        float[] z1 = Common.matrixSub(A,I.multiplyByConst(d))
                .multiplyByConst(sigma1 / c)
                .multiplyByVector(z0);
        do {
            float sigmaN1 = 1 / (2 / sigma1 - sigmaN);
            zN1 = multiplyChebyshev();
            // ...
        } while (convergence());

        return zN1;
    }
```

4 Результаты

В ходе выполнения лабораторной работы были получены алгоритмы (SOR, SSOR и Чебышева) решения СЛАУ, а также степенной и Чебышевский алгоритмы решения частичной проблемы собственных значений.

Тончость вычислений составляет 10^{-6} ввиду ограничения типа данных. Для тестирования генерировались разреженные матрицы размерностью $N = 100, 200, 500, 1000$ с плотностью 5% и плотные матрицы размерности от 5 до 15. Каждый тест проводился 10 раз для нивелирования случайных выбросов.

4.1 Решение СЛАУ итерационными методами

Были проведены тесты на время работы, точность вычисления (если по достижении максимального доступного числа шагов, сходимость не наблюдалась, или число обусловленности матрицы $k(A) = ||A|| \times ||A^{-1}||$ было слишком велико, матрица перегенерировалась заново) и количество необходимых итераций.

ПЕРЕСЧИТАТЬ РО ТАК КАК ТАМ СЕЙАЧС ЛЕВАЯ КОНСТАНТА, А НУЖНА ОЦЕНКА СПЕКТРАЛЬНОГО РАДИУСА - НОРМА МАТРИЦЫ

В тесте для заданной матрицы A и вектора b вычислялось “эталонное” решение x^*, с которым впоследствие сравнивались полученные значения.

В таблицах ниже приведены результаты некоторых тестов:

Таблица 1: Разреженные матрицы, время работы

Размерность	Плотность, %	SOR, мс	SSOR, мс	Чебышев, мс
10	5	31	37	24
10	70	45	50	35
100	5	281	253	227
200	5	529	506	482
500	5	1350	1358	1201
1000	5	2798	2676	2644

Как видно из результатов таблицы 1, метод Чебышева дает несколько меньшее время работы, что со временем ниверлируется разреженностью матрицы и резким возрастанием размерности. На “небольших” матрицах все методы показывают примерно одинаковое время вне зависимости от плотности наполнения матрицы, хотя Чебышевская итерация выглядит лучше “соперников”.

Из результатов таблицы 2 можно сделать вывод, что все алгоритмы превосходят точность вычисления 10−6 уже по достижении 100 шагов итерации (что сравнительно много для “небольшой” плотной матрицы). Однако и в этом тесте метод итерации Чебышева показывает высокую точность уже по достижении 10 шагов, упираясь в точночть представления типа данных.

Плотность матрицы на раниих этапах почти не влияет на результаты работы, не превосходя значения в 0.0007 для алгоритма SOR, что дает наихудший результат.

Таблица 2: Плотные матрицы размерности 15, относительная ошибка

Число шагов	Плотность, %	SOR	SSOR	Чебышев
5	60	0.0024	0.0002	0.00017
5	90	0.0031	0.0001	0.0002
10	90	0.000003	0.000003	0.000001
100	90	0.0	0.0	0.0

4.2 Проблема собственных значений

Были изучены и получены методы поиска собственных значений матрицы. Были проведены аналогичные предыдущим тесты. Приведем результат теста на определение скорости схождения. Для подсчета Чебышевского ускорения определим скорости сходимости степенного и Чебышевского методов по формуле

$$\mu = \frac{\log|x_{k+1} - L|}{\log|x_k - L|},$$

где L - точное решение, а множество $\{x_n\}$ - результат итерационного процесса.

Таблица 3: Собственные значения разреженных матриц на k и $k+1$ шагах

Шаг	итерация	N = 100	N = 200	N = 500	N = 1000
k+1	степенная	25.94027	47.586407	125.43564	248.9952
k	степенная	25.940178	47.586353	125.4355	248.9952
k+1	Чебышевская	25.94011	47.58619	125.44	248.99421
k	Чебышевская	25.9401	47.58610	125.4346	248.99422

Можно сделать вывод о линейности сходимости степенного метода и о некотором ускорении при использовании Чебышевской итерации (скорости $\approx$ 0.8 и 1.2). В остальном алгоритм выдает “схожие” данные.

5 Выводы

В ходе работы были изучены способы применения Чебышевской итерации для ускорения итерационных процессов, используемых как при решении СЛАУ, “обычных” или разреженных, так и при решении проблемы собственных значений.

Исходя из результатов тестирования, можно судить о том, что данные методы вычисляют результаты как с достаточно высокой точностью, так и с меньшими затратами по сравнению с другими итерационными алгоритмами, что делает их вполне применимыми на практике.