Московский авиационный институт
(национальный исследовательский университет)

Факультет: «Компьютерные науки и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»

Лабораторная работа №1

по курсу «Численные методы»
Тема: «Численные методы линейной алгебры»

Студент: Мариничев И. А.
Группа: М8О-308Б-19
Преподаватель: Пивоваров Д. Е.
Оценка:

Москва
2022

1. Постановка задачи.

Вариант №12.

1.1. Реализовать алгоритм LU - разложения матриц (с выбором главного элемента) в виде программы. Используя разработанное программное обеспечение, решить систему линейных алгебраических уравнений (СЛАУ). Для матрицы СЛАУ вычислить определитель и обратную матрицу.

1.2. Реализовать метод прогонки в виде программы, задавая в качестве входных данных ненулевые элементы матрицы системы и вектор правых частей. Используя разработанное программное обеспечение, решить СЛАУ с трехдиагональной матрицей.

1.3. Реализовать метод простых итераций и метод Зейделя в виде программ, задавая в качестве входных данных матрицу системы, вектор правых частей и точность вычислений. Используя разработанное программное обеспечение, решить СЛАУ. Проанализировать количество итераций, необходимое для достижения заданной точности.

1.4. Реализовать метод вращений в виде программы, задавая в качестве входных данных матрицу и точность вычислений. Используя разработанное программное обеспечение, найти собственные значения и собственные векторы симметрических матриц. Проанализировать зависимость погрешности вычислений от числа итераций.

1.5. Реализовать алгоритм QR – разложения матриц в виде программы. На его основе разработать программу, реализующую QR – алгоритм решения полной проблемы собственных значений произвольных матриц, задавая в качестве входных данных матрицу и точность вычислений. С использованием разработанного программного обеспечения найти собственные значения матрицы.

2. Описание методов.

2.1. LU - разложение матриц.

Решаемая задача: Решение СЛАУ, нахождение обратной матрицы и определителя матрицы системы.

Тип метода: Прямой метод.

Основная идея: Факторизовать исходную матрицу в виде верхнетреугольной и нижнетреугольной матриц. А затем за O (n^2) решить две СЛАУ по упрощенной схеме.

Сложность: O (n^3).

Преимущества: LU-разложение выполняется единожды, а значит метод удобен при многократной работе с одной и той же матрицей.

2.2. Метод прогонки.

Решаемая задача: Решение СЛАУ с трехдиагональной матрицей

Тип метода: Прямой метод.

Основная идея: Использование особенностей трехдиагональной матрицы, а именно возможность последовательного выражения каждого из неизвестных через одно предыдущее неизвестное. При этом последнее неизвестное вычисляется через коэффициент, благодаря чему при «обратном проходе» все остальные неизвестные можно последовательно вычислить.

Сложность: O (n).

Преимущества: Так как метод применим только для трехдиагональных матриц, он имеет существенный выигрыш по сложности и памяти по сравнению с LU-разложением. Кроме того, трехдиагональные матрицы имеют под собой физический смысл: взаимодействие элементов цепочки с непосредственными соседями.

2.3. Метод простых итераций (метод Якоби) и метод Зейделя.

Решаемая задача: Решение СЛАУ.

Тип методов: Итерационные методы.

Основная идея: Благодаря значительному числу итераций происходит сходимость к решению из-за особого вида коэффициентов.

Сложность: O (n^2) на каждой итерации.

Преимущества: Метод Якоби, как один из вариантов метода простых итераций, обеспечивает более быструю сходимость для матриц с диагональным преобладанием, а метод Зейделя использует при вычислениях более новую информацию, что позволяет более точно сформировать критерий сходимости.

2.4. Метод вращений.

Решаемая задача: Нахождение собственных значений и собственных векторов симметричной матрицы

Тип метода: Итерационный метод.

Основная идея: Так как симметричные матрицы имеют только веществен ные собственные значения и их собственные векторы ортогональны, мы можем спокойно занулять поддиагональные элементы, т. е. не нужно учитывать наличие комплексных собственных значений. Это реализуется при помощи матицы вращений, которая имеет физический смысл в виде поворота на угол φ, а по факту зануляет максимальный поддиагональный элемент. При этом обеспечивается общая сходимость всех поддиагональных элементов к нулю от итерации к итерации.

Сложность: O (n^2) на каждой итерации.

Преимущества: Имеет высокий потенциал распараллеливания.

2.5. QR – алгоритм.

Решаемая задача: Нахождение собственных значений произвольной матри цы.

Тип метода: Прямой метод (QR – разложение), итерационный метод (QR – алгоритм).

Основная идея: Факторизация исходной матрицы в виде ортогональной и квазитреугольной матриц. Аналогично методу вращений тут также происходит

зануление поддиагональных элементов, это происходит при помощи матрицы Хаусхолдера, применяется она постолбцово и зануляет целую группу элементов.

Сложность: O (n^3) на каждой итерации.

Преимущества: Решает довольно важную задачу нахождения полного спектра матрицы с учетом комплексных собственных значений.

3. Демонстрация работы программы.

3.1. LU - разложение матриц.

test_01.txt

```
4
-1 -8  0 5 -60
 6 -6  2 4 -10
 9 -5 -6 4  65
-5  0 -9 1  18
```

answer_01.txt

```
Matrix L:
1 0 0 0
-0.111111 1 0 0
-0.555556 0.324675 1 0
0.666667   0.311688   -
0.512326 1

Matrix U:
9 -5 -6 4
0   -8.55556   -0.666667
5.44444
0 0 -12.1169 1.45455
0 0 0 0.381565
```

```
X:
x1 = 7
x2 = 6
x3 = -6
x4 = -1

A^-1:
-0.0280899   -0.0421348
0.0955056 -0.0730337
-0.904494   1.64326   -
0.724719 0.848315
-0.123596   0.314607   -
0.179775 0.0786517
-1.25281   2.62079   -
1.14045 1.3427

det(A)      =      -356
```

3.2. Метод прогонки.

test_02.txt

```
5
-11  9   0   0  0 -114
  1 -8   1   0  0   81
  0 -2 -11   5  0   -8
  0  0   3 -14  7  -38
  0  0   0   8 10  114
```

answer_02.txt

```
X:
x1 = 2.94195
x2 = -9.07095
x3 = 5.49044
x4 = 6.85058
x5 = 5.91954
```

3.3. Метод простых итераций (метод Якоби) и метод Зейделя.

test_03.txt

```
4
14 -4 -2  3   38
-3 23 -6 -9 -195
-7 -8 21 -5  -27
-2 -2  8 18  142
0.00001
```

answer_03.txt

```
Jacobi Method (Fixed Point Iteration)      Seidel Method
X:                                         X:
x1 = -1                                    x1 = -1
x2 = -6                                    x2 = -6
x3 = -2                                    x3 = -2
x4 = 8                                     x4 = 8

Error: 1e-05                               Error: 1e-05
Iteration count: 14                        Iteration count: 11
```

3.4. Метод вращений.

test_04.txt

```
3
 7  3 -1
 3 -7 -8
-1 -8 -2
0.00001
```

answer_04.txt

```
Eigenvalues:
λ_1 = 8.57605
λ_2 = -13.0509
λ_3 = 2.47486
```

```
Eigenvectors:
V_1:
v_11 = 0.873738
v_12 = 0.344594
v_13 = -0.343274

V_2:
v_21 = -0.0924223
v_22 = 0.810512
v_23 = 0.578384

V_3:
v_31 = 0.477536
v_32 = -0.47363
v_33 = 0.740023

Error: 1e-05
Iteration count: 7
```

3.5. QR – алгоритм.

test_05_01.txt

```
3
 5 -1 -2
-4  3 -3
-2 -1  1
0.00001
```

answer_05_01.txt

```
Eigenvalues:
λ_1 = 6.387738
λ_2 = 3.836110
λ_3 = -1.223848

Error: 1e-05
Iteration count: 10
```

test_05_02.txt

```
3
1 3 1
1 1 4
4 3 1
0.01
```

answer_05_02.txt

```
Eigenvalues:
λ_1 = 6.342632
λ_2 = -1.671316 + 1.552148i
λ_3 = -1.671316 - 1.552148i

Error: 0.01
Iteration count: 9
```

4. Основной код программ.

4.1. LU - разложение матриц.

```
void LU_decomposition(std::vector<std::vector<double>> a, std::vector<double> &B,
std::vector<std::vector<double>> &L, std::vector<std::vector<double>> &U) {
    double mu;

    permutation_cnt = 0;
    P.resize(n);

    unsigned long long max_id;
    make_identity_matrix(L);
    for (unsigned long long k = 0; k < (n - 1); ++k) { // step
        max_id = find_max(a, k);
        P[k] = max_id; // remember permutation
        if (max_id != k) {
            swap_rows(a, k, max_id);
            std::swap(B[k], B[max_id]);
            swap_rows(L, k, max_id);
            swap_columns(L, k, max_id);
            permutation_cnt++;
        }
```

```
        for (unsigned long long i = (k + 1); i < n; ++i) { // index of a row
              mu = a[i][k] / a[k][k]; // coefficient to make k-th element of
current row equal to zero
            L[i][k] = mu;
            for (unsigned long long j = k; j < n; ++j) { // index of a column
                 a[i][j] -= mu * a[k][j]; // subtract k-th row multiplied by mu
from all rows below it
            }
        }
    }
    P[n-1] = n-1;
    L[n-1][n-1] = 1;
    U = a;
}
```

4.2. Метод прогонки.

```
std::vector<double> TDMA(std::vector<double> a, std::vector<double> b, std::vect
or<double> c, std::vector<double> d) {
    std::vector<double> x(n);
    std::vector<double> P(n), Q(n);

    // moving up to find coefficients P_i and Q_i
    P[0] = - c[0] / b[0];
    Q[0] = d[0] / b[0];
    for (unsigned long long i = 1; i < n-1; ++i) {
        P[i] = - c[i] / (b[i] + a[i] * P[i-1]);
        Q[i] = (d[i] - a[i] * Q[i-1]) / (b[i] + a[i] * P[i-1]);
    }
    x[n-1] = (d[n-1] - a[n-1]*Q[n-2]) / (b[n-1] + a[n-1]*P[n-2]);
    Q[n-1] = x[n-1];
    P[n-1] = 0;

    // moving down to find x_i
    for (unsigned long long i = n-1; i-- > 0;)
        x[i] = P[i]*x[i+1] + Q[i];

    return x;
}
```

4.3. Метод простых итераций (метод Якоби) и метод Зейделя.

```
std::vector<double> fixed_point_iteration_jacobi(std::vector<std::vector<double>>
a, std::vector<double> b, double epsilon) {
    std::vector<double> x_cur(n);  // x_k
    std::vector<double> x_prev(n); // x_k-1

    std::vector<std::vector<double>> alpha(n);
    for (unsigned long long i = 0; i < n; ++i)
        alpha[i].resize(n);
    alpha = find_alpha_jacobi(a);

    double alpha_norm;  // ||alpha||_inf = max_i (sum_j |alpha[i][j]|)
    alpha_norm = find_matrix_norm(alpha);

    // zero iteration
    iteration_cnt = 0;
    for (unsigned long long i = 0; i < n; ++i) {
        x_prev[i] = b[i] / a[i][i];
    }

    double vector_norm; // ||x_k[i] - x_k-1[i]||_inf = max_i |x_k[i] - x_k-1[i]|
    double epsilon_k;
    do {
        // calculate x_k
        for (unsigned long long i = 0; i < n; ++i) {
            x_cur[i] = b[i];
            for (unsigned long long j = 0; j < n; ++j) {
                if (i != j)
                    x_cur[i] -= a[i][j] * x_prev[j];
            }
            x_cur[i] /= a[i][i];
        }

        // calculate epsilon_k
        vector_norm = fabs(x_prev[0] - x_cur[0]);
        for (unsigned long long i = 0; i < n; ++i) {
            if (fabs(x_prev[i] - x_cur[i]) > vector_norm)
                vector_norm = fabs(x_prev[i] - x_cur[i]);
            x_prev[i] = x_cur[i];
        }
        epsilon_k =  (alpha_norm / (1 - alpha_norm)) * vector_norm;
```

```
        iteration_cnt++;
    } while (epsilon_k > epsilon);

    return x_cur;
}

std::vector<double> seidel_method(std::vector<std::vector<double>> a, std::vecto
r<double> b, double epsilon) {
    std::vector<double> x_cur(n);  // x_k
    std::vector<double> x_prev(n); // x_k-1

    std::vector<std::vector<double>> alpha(n);
    for (unsigned long long i = 0; i < n; ++i)
        alpha[i].resize(n);
    alpha = find_alpha(a);

    double alpha_norm;  // ||alpha||_inf = max_i (sum_j |alpha[i][j]|)
    alpha_norm = find_matrix_norm(alpha);

    double c_norm;
    for (unsigned long long i = 0; i < n; ++i) {
        for (unsigned long long j = 0; j < n; ++j) {
            if (i <= j)
                alpha[i][j] = 0;
        }
    }
    c_norm = find_matrix_norm(alpha);

    // zero iteration
    iteration_cnt = 0;
    for (unsigned long long i = 0; i < n; ++i) {
        x_cur[i] = b[i] / a[i][i];
    }

    double vector_norm; // ||x_k[i] - x_k-1[i]||_inf = max_i |x_k[i] - x_k-1[i]|
    double epsilon_k;
    double sum_1, sum_2;
    do {
        // calculate x_k
        for (unsigned long long i = 0; i < n; ++i)
            x_prev[i] = x_cur[i];
        for (unsigned long long i = 0; i < n; ++i) {
            sum_1 = 0;
```

```
            for (unsigned long long j = 0; j < i; ++j)
                sum_1 += a[i][j] * x_cur[j];

            sum_2 = 0;
            for (unsigned long long j = i+1; j < n; ++j)
                sum_2 += a[i][j] * x_prev[j];

            x_cur[i] = (b[i] - sum_1 - sum_2) / a[i][i];
        }

        // calculate epsilon_k
        vector_norm = fabs(x_prev[0] - x_cur[0]);
        for (unsigned long long i = 0; i < n; ++i) {
            if (fabs(x_prev[i] - x_cur[i]) > vector_norm)
                vector_norm = fabs(x_prev[i] - x_cur[i]);
        }
        epsilon_k =  (c_norm / (1 - alpha_norm)) * vector_norm;

        iteration_cnt++;
    } while (epsilon_k > epsilon);

    return x_cur;
}
```

4.4. Метод вращений.

```
std::vector<double> jacobi_eigenvalue_algorithm(std::vector<std::vector<double>>
a, std::vector<std::vector<double>> &v, double epsilon) {
    std::vector<std::vector<double>> u(n);
    std::vector<std::vector<double>> u_transposed;
    for (unsigned long long i = 0; i < n; ++i)
        u[i].resize(n);

    std::pair<unsigned long long, unsigned long long> cur_max_element_id;
    unsigned long long l, m;
    double phi, epsilon_k;
    bool do_initialize = true;

    do {
        cur_max_element_id = find_max(a);
        l = cur_max_element_id.first;
        m = cur_max_element_id.second;
        phi = 0.5 * atan((2 * a[l][m]) / (a[l][l] - a[m][m]));
```

```
        make_identity_matrix(u);
        u[l][l] = cos(phi);
        u[m][m] = cos(phi);
        u[l][m] = -sin(phi);
        u[m][l] = sin(phi);

        // A^k+1 = (U^k)^T * A^k * U^k
        u_transposed = transpose(u);
        a = multiply(u_transposed, a);
        a = multiply(a, u);

        // accumulate eigenvectors matrix
        if (do_initialize) {
            v = u;
            do_initialize = false;
        } else {
            v = multiply(v, u);
        }

        epsilon_k = find_euclidean_norm(a);

        iteration_cnt++;
    } while (epsilon_k > epsilon);

    return find_eigenvalues(a);
}
```

4.5. QR – алгоритм.

```
void QR_decomposition(std::vector<std::vector<double>> &Q, std::vector<std::vect
or<double>> &R) {
    std::vector<std::vector<double>> H(n);
    for (unsigned long long i = 0; i < n; ++i) {
        H[i].resize(n);
    }

    make_identity_matrix(Q);
    R = A;
    for (unsigned long long i = 0; i < n - 1; ++i) {
        std::vector<double> v_k(n);
        for (unsigned long long j = i; j < n; ++j) {
            v_k[j] = R[j][i];
```

```
        }
        H = make_householder_matrix(v_k, i);
        Q = multiply(Q, H);
        R = multiply(H, R);
    }
}

void QR_algorithm() {
    do {
        std::vector<std::vector<double>> Q(n);
        std::vector<std::vector<double>> R(n);
        for (unsigned long long i = 0; i < n; ++i) {
            Q[i].resize(n);
            R[i].resize(n);
        }

        QR_decomposition(Q, R);
        A = multiply(R, Q);

        iteration_cnt++;
    } while (!converge());
}
```

5. Выводы.

В ходе данной лабораторной работы я изучил базовые численных методы, решающие задачи линейной алгебры. При помощи LU-разложения, метода прогонки, метода простых итераций и метода Зейделя удалось найти решение систем линейных уравнений. При помощи метода вращений была решена задача поиска полного спектра и собственных векторов для симметричных матриц, а при помощи QR-алгоритма была решена задача поиска полного спектра для произвольных матриц. Все эти методы имеют свои особенности, которые нужно учитывать, применяя их.

Стоит также отметить, что те варианты реализаций, которые были написаны мной на C++ носят скорее учебный характер, так как передо мной стояла задача понять алгоритмы при реализации, а не написать максимально оптимальные решатели тех или иных задач.