Московский авиационный институт

(Национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторные работы по курсу «Численные методы»

Студент: Устинов А.Э

Группа: М80-304Б-20

Преподаватель: Демидова О.Л.

Оценка:

Москва, 2022

**Пункт 1.1**

**Задача:** Решение СЛАУ, нахождение обратной матрицы и определителя матрицы системы

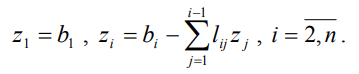
**Метод:** LU-разложение

**Основные применения**: Решение систем алгебраических уравнений, вычисление определителя, вычисление обратной матрицы и др.

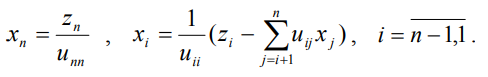
LU – разложение матрицы A представляет собой разложение матрицы A в произведение нижней и верхней треугольных матриц (A = LU)

Для экономии памяти, будем записывать обе матрицы в одну, подразумевая, что у матрицы L на главной диагонали единицы.

На первом этапе решается СЛАУ Lz = b . Поскольку матрица системы - нижняя треугольная, решение можно записать в явном виде



На втором этапе решается СЛАУ Ux = z с верхней треугольной матрицей. Здесь, как и на предыдущем этапе, решение представляется в явном виде



Второй этап эквивалентен обратному ходу методу Гаусса, тогда как первый соответствует преобразованию правой части СЛАУ в процессе прямого хода.

**Код LU-разложения и нахождение det(A):**

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

double elem = A[i][i];

double coeff;

for (int j = i + 1; j < n; j++) {

coeff = A[j][i] / elem;

for (int k = 0; k < n; k++) {

if (k >= i)

A[j][k] = A[j][k] - A[i][k] \* coeff; //Условие здесь, чтобы не залезать на матрицу L

E[j][k] = E[j][k] - E[i][k] \* coeff;

}

A[j][i] = coeff;

}

det \*= A[i][i];

}

det \*= pow(-1, count);

**Особенности LU-разложения**

1. Легко вычисляется определитель матрицы:  
   [Изображение выглядит как текст, часы, датчик

   Автоматически созданное описание](http://www.codecogs.com/eqnedit.php?latex=\left%20|%20A%20\right%20|=\left%20|%20L%20\right%20|\cdot%20\left%20|%20U%20\right%20|%20=%20\left%20|%20U%20\right%20|=\prod_%7bi=1%7d%5e%7bn%7du_%7bi,i%7d)
2. Однажды найдя LU-разложение для матрицы мы можем очень быстро решать системы линейных алгебраических уравнений с различной правой частью.  
   [](http://www.codecogs.com/eqnedit.php?latex=A\cdot%20\bar%7bx%7d=\bar%7bb%7d)  
   [](http://www.codecogs.com/eqnedit.php?latex=L\cdot%20U\cdot%20\bar%7bx%7d=\bar%7bb%7d)  
   Пусть [](http://www.codecogs.com/eqnedit.php?latex=\bar%7by%7d=U\cdot%20\bar%7bx%7d)  
   Тогда [](http://www.codecogs.com/eqnedit.php?latex=L\cdot%20\bar%7by%7d=\bar%7bb%7d)  
   Так как [](http://www.codecogs.com/eqnedit.php?latex=L) — нижнетреугольная матрица, то очень легко находим [](http://www.codecogs.com/eqnedit.php?latex=\bar%7by%7d)  
   Решаем [](http://www.codecogs.com/eqnedit.php?latex=U\cdot%20\bar%7bx%7d%20=%20\bar%7by%7d)  
   Легко находим [](http://www.codecogs.com/eqnedit.php?latex=\bar%7bx%7d), так как [](http://www.codecogs.com/eqnedit.php?latex=U) — вехнетреугольная матрица
3. Сложность алгоритма:  
   LU-разложение:  
   [](http://www.codecogs.com/eqnedit.php?latex=\frac%7b2\cdot%20n%5e%7b3%7d%7d%7b3%7d@plus;O(n%5e%7b2%7d))  
   Последующее решение систем:  
   [](http://www.codecogs.com/eqnedit.php?latex=n%5e%7b2%7d@plus;O(n))

**Пункт 1.2**

**Задача:** Решение СЛАУ с трехдиагональной матрицей

**Метод:** Метод прогонки

**Основные применения**: Эффективный метод решения СЛАУ с трех - диагональными матрицами, возникающих при конечно-разностной аппроксимации задач для обыкновенных дифференциальных уравнений (ОДУ) и уравнений в частных производных второго порядка и является частным случаем метода Гаусса.

Решения ищутся в виде:



Прямой ход предназначен для нахождения прогоночных коэффициентов

Обратный ход предназначен для нахождения вектора

X = () в соответствии с найденными

Для устойчивости метода прогонки (1.4)-(1.7) достаточно выполнение следующих условий:

Изображение выглядит как текст, часы

Автоматически созданное описание

**Код:**

P[0] = -A[0][2] / A[0][1];

Q[0] = d[0] / A[0][1];

for (int i = 1; i < n; i++)

{

P[i] = -A[i][2] / (A[i][1] + A[i][0] \* P[i - 1]);

Q[i] = (d[i] - A[i][0] \* Q[i - 1]) / (A[i][1] + A[i][0] \* P[i - 1]);

}

X[n - 1] = Q[n - 1];

for (int i = n - 2; i >= 0; i--)

{

X[i] = P[i] \* X[i + 1] + Q[i];

}

cout << "\n";

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

cout << X[i] << " ";

}

if (cond) cout << "\nSufficient condition not met";

**Особенности метода прогонки:**

Метода можно решать только специфические системы, имеющие не более трех неизвестных в каждой строке.

Сложность O(N)

**Пункт 1.3**

**Задача:** Решение СЛАУ

**Методы:** Метод простых итераций и метод Зейделя

**Основные применения**: Для решения СЛАУ с разреженными матрицами.

**Метод простых итераций**

За нулевое приближение принимается вектор правых частей. В следующем приближении, для вычисления X подставляются значение вектора X на предыдущем приближении. Итерации повторяются до достижения заданной точности

Метод простых итераций сходится к единственному решению при любом начальном приближении , если какая-либо норма матрицы эквивалентной системы меньше единицы

Для сходимости итерационного процесса необходимо и достаточно, чтобы спектр матрицы α эквивалентной системы лежал внутри круга с радиусом, равным единице.

При выполнении достаточного условия сходимости оценка погрешности решения на k - ой итерации дается выражением:

Изображение выглядит как текст, часы

Автоматически созданное описание

Процесс итераций останавливается при выполнении условия , где - задаваемая вычислителем точность.

Поскольку является только достаточным (не необходимым) условием сходимости метода простых итераций, то итерационный процесс может сходиться и в случае, если оно не выполнено. Тогда критерием окончания итераций может служить неравенство

**Код:**

int k = 0;

double\* X = new double[n], \* oldX = new double[n], norm = 0;

copy(b, b + n, oldX);

double epsK, s = 0, coeff = (Cnorm / (1 - Cnorm));

if (conditions) coeff = 1;

do {

k++;

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

for (int j = 0; j < n; j++) {

s += A[i][j] \* oldX[j];

}

X[i] = s + b[i];

s = 0;

if (norm < abs(X[i] - oldX[i]))

norm = abs(X[i] - oldX[i]);

}

epsK = coeff \* norm;

copy(X, X + n, oldX);

norm = 0;

//cout << epsK << "\n";

} while (epsK > eps);

cout << "Iterations method converged in " << k << " iterations\n";

return X;

**Особенности метода простых итераций:**

Удобен при распараллеливании, так как приближения считаются построчно независимо друг от друга.

Можно задать точность приближения

В вычислительном процессе участвуют только произведения матрицы на вектор, что позволяет работать только с ненулевыми элементами матрицы, значительно упрощая процесс хранения и обработки матриц, по сравнению с методом Гаусса

**Метод простых Зейделя:**

Метод простых итераций довольно медленно сходится. Для его ускорения существует метод Зейделя, заключающийся в том, что при вычислении компонента вектора неизвестных на (k+1)-й итерации используются , уже вычисленные на (k+1)-й итерации. Значения остальных компонент берутся из предыдущей итерации. Значение остальных компонент берутся из предыдущей итерации.

Оценка погрешности вычисляется аналогично

Первое приближение – вектор правых частей

**Код:**

int k = 0;

double\* X = new double[n], \* oldX = new double[n], norm = 0, norm\_upper = 0, summ = 0;

copy(b, b + n, oldX);

double epsK, s = 0, coeff;

if (conditions) {

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

for (int j = i + 1; j < n; j++) {

summ += abs(A[i][j]);

}

if (norm\_upper < summ)

norm\_upper = summ;

}

coeff = (norm\_upper / (1 - Cnorm));

}

else coeff = 1;

do {

k++;

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

for (int j = 0; j < n; j++) {

if (j < i)

s += A[i][j] \* X[j];

else {

s += A[i][j] \* oldX[j];

}

}

X[i] = s + b[i];

s = 0;

if (norm < abs(X[i] - oldX[i]))

norm = abs(X[i] - oldX[i]);

}

epsK = coeff \* norm;

copy(X, X + n, oldX);

norm = 0;

//cout << epsK << "\n";

//print(X);

} while (epsK > eps);

cout << "Seidel method converged in " << k << " iterations\n";

return X;

**Особенности метода Зейделя:**

Сходимость быстрее, чем у метода простых итераций, но невозможно распараллеливаливание.

**Пункт 1.4**

**Задача:** Нахождение собственных значений и собственных векторов симметричной матрицы **Методы:** Метод вращений

**Основные применения**: Нахождение собственных значений и собственных векторов симметричной матрицы

1. Находим максимальный недиагональный элемент матрицы
2. Находим соответствующую этому элементу матрицу вращения
3. Вычисляем матрицу

В качестве критерия окончания итерационного процесса используется условие малости суммы квадратов внедиагональных элементов:

Изображение выглядит как текст

Автоматически созданное описание

**Код:**

double phi;

double\*\* U;

double\*\* X = new double\*[n];

pair\_answ answ = find\_max(A);

if (A[answ.i][answ.i] - A[answ.j][answ.j] != 0)

phi = 0.5 \* atan(2 \* answ.max / (A[answ.i][answ.i] - A[answ.j][answ.j]));

else

phi = M\_PI / 4;

U = findU(answ.i, answ.j, phi);

copy(U, U + n, X);

A = multiple(multiple(transpose(U), A), U);

answ = find\_max(A);

while (answ.sqr > eps) {

if (A[answ.i][answ.i] - A[answ.j][answ.j] != 0)

phi = 0.5 \* atan(2 \* answ.max / (A[answ.i][answ.i] - A[answ.j][answ.j]));

else

phi = M\_PI / 4;

U = findU(answ.i, answ.j, phi);

X = multiple(X, U);

A = multiple(multiple(transpose(U), A), U);

answ = find\_max(A);

}

print\_matr(A);

cout << "\n";

print\_matr(X);

**Пункт 1.5**

**Задача:** Нахождение собственных значений

**Методы:** Метод QR разложений

**Основные применения**: Нахождение собственных значений

1. Ищем матрицу Хаусхолдера, соответствующую вектору:

Изображение выглядит как текст

Автоматически созданное описание

по формуле

Изображение выглядит как текст, антенна

Автоматически созданное описание

1. Проделываем аналогичную процедуру для всех векторов матрицы A
2. На каждой итерации производим умножение
3. R =

После проделывания этой операции, считаем

Повторяем вышеназванную последовательность действий, пока для каждого из диагональных значений матрицы не будет выполнено одно из двух условий:

или

Причем, в случае первого условия, проверим диагональную матрицу 2x2, решив квадратное уравнение:



**Код:**

for (int k = 0; k < n; k++) {

complex num, num2, old\_num, old\_num2;

num.real = 0; num.im = 0;

num2.real = 0; num2.im = 0;

while (true) {

for (int i = 0; i < n - 1; i++) {

v = get\_vect(A, i);

den = vector\_multiple(v, v); //нахождение знаменателя

newH = findH(v, v, den); //нахождение H

A = multiple(newH, A);

Q = multiple(Q, newH);

}

A = multiple(A, Q); //умножение A и Q

ones(Q); // приведение Q к единичному виду

print(A);

if (find\_norm(A, k) < eps) {

lambdas[k].im = 0;

lambdas[k].real = A[k][k];

break;

}

old\_num.real = num.real; old\_num.im = num.im;

old\_num2.real = num2.real; old\_num2.im = num2.im;

under\_root = A[k][k] \* A[k][k] + A[k + 1][k + 1] \* A[k + 1][k + 1] - 2 \* A[k][k] \* A[k + 1][k + 1] + 4 \* A[k + 1][k] \* A[k][k + 1];

if (under\_root < 0) {

num.real = (A[k][k] + A[k + 1][k + 1]) / 2;

num.im = sqrt(abs(under\_root))/2;

num2.real = (A[k][k] + A[k + 1][k + 1]) / 2;

num2.im = -sqrt(abs(under\_root))/2;

}

else {

num.real = (A[k][k] + A[k + 1][k + 1]) / 2 + sqrt(under\_root) / 2;;

num.im = 0;

num2.real = (A[k][k] + A[k + 1][k + 1]) / 2 - sqrt(under\_root) / 2;

num2.im = 0;

}

if (abs(num.real - old\_num.real) < eps && abs(num2.real - old\_num2.real) < eps) {

lambdas[k] = num;

lambdas[k + 1] = num2;

k++;

break;

}

}

}

**Особенности метода QR разложений:**

Существенным недостатком рассмотренного выше алгоритма является большое число операций необходимое для - факторизации матрицы на каждой итерации.

Преимущество метода в том, что с помощью него можно находить комплекснозначные собственные значения

**Во всех программах реализовано чтение матриц из файла.**

**Пример:**

0.01

3

1 3 1

1 1 4

4 3 1

Где 0.01 – это задаваемая точность, 3 – это размерность матрицы, а далее сама матрица.

**Вывод через командную строку**