**1) Входные данные и текст программы:**

Входные данные: Пользователь вводит размер матрицы N. Программа инициализирует матрицу vec и заполняет ее с помощью функции filler:

void filler(vector <vector<double>>& vec)

{

int n = vec.size();

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

{

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

{

vec[i][j] = 2 \* (i + 1) - (j+1);

if (i!=j)

{

vec[j][i] = 2 \* (i + 1)-(j + 1);

}

}

}

}

И сразу же вывести её в консоль:  
void couter(vector <vector<double>>& vec)

{

int n = vec.size();

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

{

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

{

cout << setw(14) << vec[i][j];

}

cout << endl;

}

cout << endl;

}

Далее создается тестовый вектор для копии основного и дальнейшей корректной работы программы:

vector <vector<double>> testVec = vec;

Для использования в формулах вращений Якоби создал и заполнил единичную матрицу

vector<vector<double>> OnesMatricx(n, vector<double>(n, 0));

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

{

OnesMatricx[i][i] = 1;

}

1. Основные методы и их реализация:

void YakobiMethod(vector <vector<double>> vec, vector<vector<double>> S) {

int k = 0;

int n = vec.size();

int fir = 0, sec = 0;

while (true) {

double maxEl = INT\_MIN;

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

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

if (i != j && vec[i][j] != 0) {

if (maxEl < abs(vec[i][j]))

{

maxEl = abs(vec[i][j]);

fir = i;

sec = j;

}

}

}

}

if (maxEl < e ) {

cout << "Марица собственных значений: " << endl;

couter(vec);

cout << "Марица собственных векторов: " << endl;

couter(S);

cout << "Количество итераций: " << k << endl;

cout << endl;

break;

}

vector<vector<double>> yakobyMatrix(n, vector<double>(n, 0));

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

{

yakobyMatrix[i][i] = 1;

}

double alpha = atan2(2.0 \* vec[fir][sec], vec[fir][fir] - vec[sec][sec]) / 2.0;

yakobyMatrix[fir][fir] = yakobyMatrix[sec][sec] = cos(alpha);

yakobyMatrix[sec][fir] = -sin(alpha);

yakobyMatrix[fir][sec] = sin(alpha);

vec = multiplyMatrix(yakobyMatrix, vec);

swap(yakobyMatrix[sec][fir], yakobyMatrix[fir][sec]);

vec = multiplyMatrix(vec, yakobyMatrix);

S = multiplyMatrix(S,yakobyMatrix);

k++;

}

}   
  
это метод Якоби, все высчитывается по стандартным формулам.  
  
Реализация вспомогательной функции умножения матриц:  
vector<double> multiplyMatrixVector(const vector<vector<double>>& matrix, const vector<double>& vector)

{

int n = matrix.size();

std::vector<double> result(n, 0);

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

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

result[i] += matrix[i][j] \* vector[j];

}

}

return result;

}

void DegreeMethod(vector<vector<double>> vec)

{

vector<double> Yvec(n, 0);

Yvec[0] = 1;

int k = 0;

double lamda = 0;

vector<double> test;

do

{

test = multiplyMatrixVector(vec, Yvec);

lamda = calculateEigenvalue(test, Yvec) / calculateEigenvalue(Yvec, Yvec);

test = FindedNormed(test, vectorNorm(test));

Yvec = test;

k++;

}

while (vectorNorm(incremVec(multiplyMatrixVector(vec, test), multVec(lamda, test))) > e);

cout << "Собственные вектор степенного метода: " << std::endl;

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

{

cout << Yvec[i] << endl;

}

cout<< "Собственное значение степенного метода: " << lamda << std::endl;

cout << "Количесто итераций: " << k << std::endl;

}

А это метод степенной итерации, где я использовал Евклидову норму при вычислении.  
  
Рассчет lamda

double calculateEigenvalue(vector<double> vec1, vector<double> vec2) {

int n = vec1.size();

double lamda=0;

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

{

lamda += (vec1[i] \* vec2[i]);

}

return lamda;

}

А вот и само вычисление Нормы  
vector<double> FindedNormed(vector<double>Yvec, double norma)

{

int n = Yvec.size();

vector<double> C(n, 0);

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

{

C[i] = Yvec[i] / norma;

}

return C;

}  
  
вспомогательная функция умножения вектора на скаляр   
vector<double> multVec(double lamb, vector<double> vec)

{

int n = vec.size();

vector < double > answ(n, 0);

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

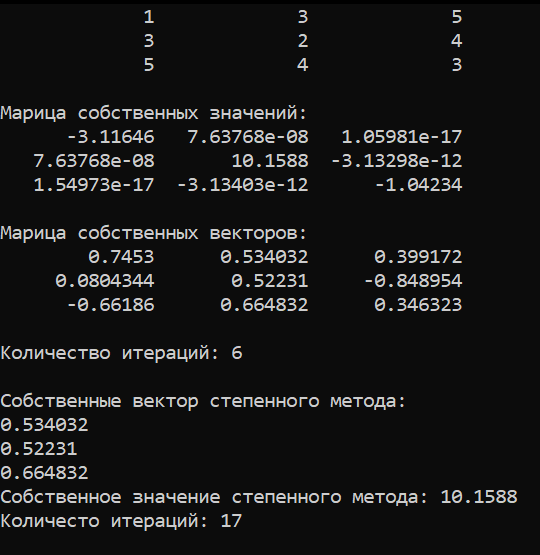
{

answ[i] = vec[i] \* lamb;

}

return answ;

}

1. Итоговый вывод   
   

4)Вывод:

Метод вращений Якоби и степенной метод — численные алгоритмы для вычисления собственных значений матриц. Сходимость зависит от различных факторов.

Метод вращений Якоби гарантированно сходится для симметричных матриц, приводя их к диагональному виду через вращения. Эффективность зависит от спектральных свойств матрицы: различные по величине собственные значения способствуют быстрой сходимости, в то время как близкие значения или плохая обусловленность матрицы могут замедлить процесс.

Степенной метод — итерационный способ приближенного вычисления наибольшего по модулю собственного значения матрицы. Он быстро сходится, если начальное приближение близко к вектору, соответствующему максимальному собственному значению. Сходимость может замедляться при близких к вырожденным матрицам или наличии близких по значению собственных значений.

Выбор между методами зависит от характеристик матрицы и требований задачи. Анализ спектральных свойств помогает определить наиболее эффективный метод в конкретной ситуации.