**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ЛАБОРАТОРНАЯ РАБОТА 3**

Тема: «Итерационный степенной метод»

Выполнил студент 1 гр.

Рудь Андрей Владимирович

**Минск 2019**

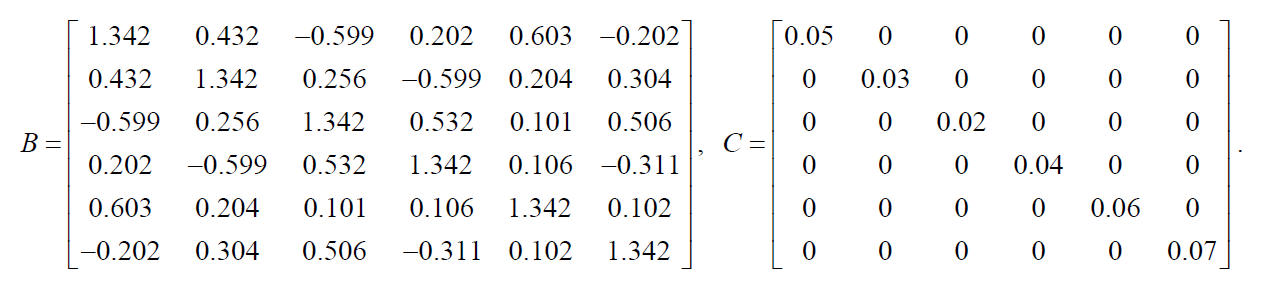
**Задание**

**Постановка задачи:**

С помощью итерационного степенного метода найти с точностью 10-6 наибольшее по модулю собственное значение и соответствующий ему собственный вектор симметричной матрицы. Вычислительный процесс проводить с нормировкой векторов итерационной последовательности.

Матрица определяется формулой, где – номер варианта.

Таким образом, для матриц

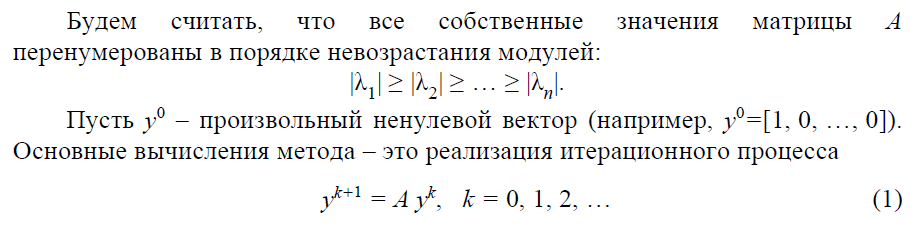


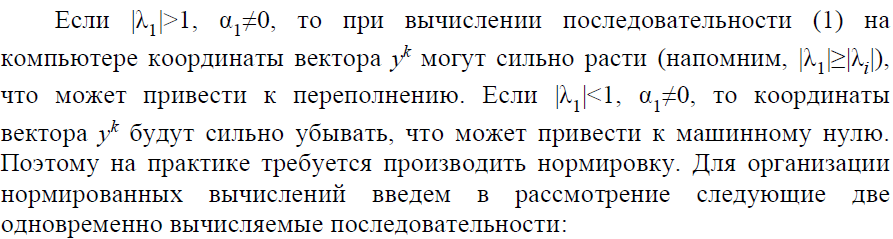
и =6,

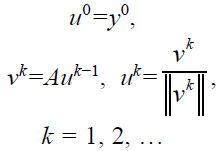
**Краткие теоретические сведения:**

Итерационный степенной метод (называемый также степенным методом) предназначен для нахождения одного или нескольких собственных значений и соответствующих собственных векторов.

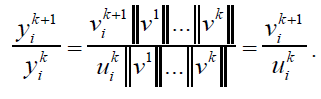
Матрица заведомо диагонализируема в двух важных частных случаях: если она симметричная или если ее собственные значения различны.

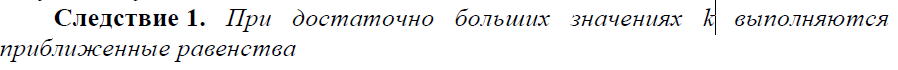


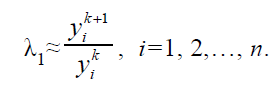


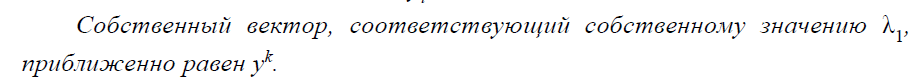


Тогда

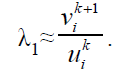


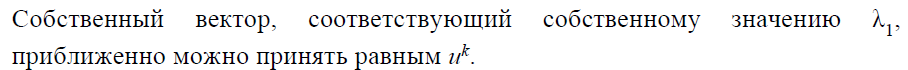




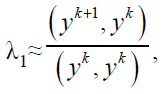


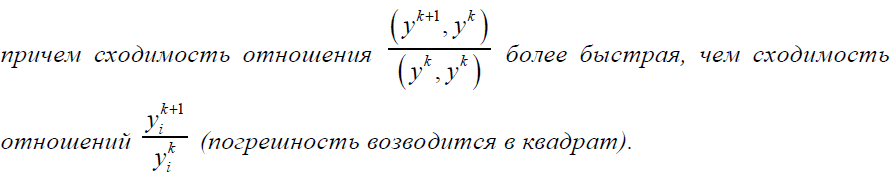
При нормированных вычислениях можно положить











Замечание: Окончание итераций определяется либо максимальным заданным числом итераций , либо условием

,

где ε >0 – заданное число.

Нормы векторов:

Кубическая норма (используется также название максимум-норма):

|| x ||I = max |xi|, 1 ≤ i ≤ n

Для этой нормы используется также обозначение || *x* ||∞

**Листинг программы:**

Класс матрицы с необходимыми конструкторами и методами:

class Matrix

{

public:

Matrix(int nx, int ny, bool) :size\_x(nx), size\_y(ny)

{

random\_device generator;

uniform\_int\_distribution<> distribution(-1000, 1000);

vector < vector<T>> matrix1(size\_x);

for (auto it = matrix1.begin(); it != matrix1.end(); ++it)

{

for (int i = 0; i < size\_y; ++i)

{

int temp = distribution(generator);

T t = temp;

t /= 100;

(\*it).push\_back(t);

}

}

matrix = matrix1;

}

Matrix(int nx, int ny) :size\_x(nx), size\_y(ny)

{

vector < vector<T>> matrix1(size\_x);

for (auto it = matrix1.begin(); it != matrix1.end(); ++it)

{

for (int i = 0; i < size\_y; ++i)

{

(\*it).push\_back(0);

}

}

matrix = matrix1;

}

Matrix(ifstream& fi)

/\*Ввод: разеры матрицы + сама матрица, через побел\*/

{

fi >> size\_x >> size\_y;

vector <vector<T>> matrix1(size\_x);

for (auto it = matrix1.begin(); it != matrix1.end(); ++it)

{

for (int i = 0; i < size\_y; ++i)

{

T t;

fi >> t;

(\*it).push\_back(t);

}

}

matrix = matrix1;

}

void matrix\_round(int p)

{

for (int i = 0; i < size\_x; ++i)

for (int j = 0; j < size\_y; ++j)

matrix[i][j] = round(matrix[i][j] \* pow(10,p)) / pow(10, p);

}

Matrix<T> operator\* (Matrix<T> const m)

{

int sizex1 = (\*this).size\_x;

int sizey1 = (\*this).size\_y;

int sizex2 = m.size\_x;

int sizey2 = m.size\_y;

if (sizey1 != sizex2)

return(\*this);

Matrix<T> temp(sizex1, sizey2);

temp.nullify();

for (int i = 0; i < sizex1; i++)

for (int k = 0; k < sizey2; k++)

for (int j = 0; j < sizex2; j++)

temp.matrix[i][k] += ((\*this).matrix[i][j]) \* (m.matrix[j][k]);

return temp;

}

Matrix<T> operator\* (T c)

{

Matrix<T> temp = \*this;

for (int i = 0; i < temp.size\_x; ++i)

for (int j = 0; j < temp.size\_y; ++j)

temp[i][j] \*= c;

return temp;

}

Matrix<T> operator/ (T c)

{

Matrix<T> temp = \*this;

for (int i = 0; i < temp.size\_x; ++i)

for (int j = 0; j < temp.size\_y; ++j)

temp[i][j] /= c;

return temp;

}

Matrix<T> operator+ (Matrix<T> const m)

{

Matrix<T> temp(m.size\_x, m.size\_y);

temp.nullify();

for (int i = 0; i < m.size\_x; ++i)

for (int j = 0; j < m.size\_y; ++j)

temp.matrix[i][j] = matrix[i][j] + m.matrix[i][j];

return temp;

}

Matrix<T> operator- (Matrix<T> const m)

{

Matrix<T> temp(m.size\_x, m.size\_y);

temp.nullify();

for (int i = 0; i < m.size\_x; ++i)

for (int j = 0; j < m.size\_y; ++j)

temp.matrix[i][j] = matrix[i][j] - m.matrix[i][j];

return temp;

}

void nullify()

{

for (int i = 0; i < size\_x; ++i)

for (int j = 0; j < size\_y; ++j)

matrix[i][j] = 0;

}

friend ostream& operator<< (ostream& os, Matrix<T> const m)

{

for (int i = 0; i < m.size\_x; ++i)

{

for (int j = 0; j < m.size\_y; ++j)

{

os.width(12);

os << m.matrix[i][j] << ' ';

}

os << endl;

}

return os;

}

Matrix transpose()

{

Matrix<T> At(size\_y, size\_x);

for (int i = 0; i < size\_x; ++i)

for (int j = 0; j < size\_y; ++j)

At.matrix[j][i] = matrix[i][j];

return At;

}

vector<T>& operator[] (int i)

{

return matrix[i];

}

int size\_x;

int size\_y;

vector<vector<T>> matrix;

};

Метод вычисления максимум-нормы ветора:

template<typename T>

T maximum\_norm(Matrix<T> v)

{

if (v.size\_y != 1)

{

throw Bad("not a vector!");

return 0.;

}

else

{

T max = v[0][0];

for (int i = 1; i < v.size\_x; ++i)

{

if (abs(v[i][0]) > abs(max))

max = v[i][0];

}

return abs(max);

}

}

Скалярное произведение двух векторов:

template<typename T>

T scalar\_product(Matrix<T> v1, Matrix<T> v2)

{

if (v1.size\_y != 1 || v2.size\_y != 1)

{

throw Bad("not a vector!");

return 0.;

}

if (v1.size\_x != v2.size\_x)

{

throw Bad("not a vector!");

return 0.;

}

T res = 0;

for (int i = 0; i < v1.size\_x; ++i) {

res += v1[i][0] \* v2[i][0];

}

return res;

}

Покоординатное деление двух векторов:

template<typename T>

Matrix<T> division(Matrix<T> v1, Matrix<T> v2) //покоординатное деление

{

if (v1.size\_y != 1 || v2.size\_y != 1)

{

throw Bad("not a vector!");

}

if (v1.size\_x != v2.size\_x)

{

throw Bad("not a vector!");

}

Matrix<T> m(v1.size\_x, 1);

for (int i = 0; i < v1.size\_x; ++i)

{

if (!v2[i][0])

throw Bad("координата делителя = 0");

m[i][0] = v1[i][0] / v2[i][0];

}

return m;

}

Степенной метод без использования симметрии:

template <typename T>

auto power\_iteration(Matrix<T> A,double eps,int& counter)

{

if (A.size\_x != A.size\_y)

throw Bad("not quadratic");

Matrix<T> y0(A.size\_x, 1);

bool b = true;

for (int i = 0; i < A.size\_x; ++i)

{

y0[i][0] = 1;

}

auto y1 = A \* y0;

auto lambdaPrev = division(y1, y0);

auto yPrev = A \* y1;

auto lambda = division(yPrev , y1);

auto u = yPrev , uPrev = yPrev;

while (maximum\_norm(lambda - lambdaPrev) >= eps)

{

++counter;

auto nu = A \* u;

lambdaPrev = lambda;

lambda = division(nu, u);

uPrev = u;

u = nu / maximum\_norm(nu);

}

return make\_pair(lambda, uPrev);

}

Степенной метод с использованием симметрии:

template <typename T>

auto power\_iteration\_symmetric(Matrix<T> A, double eps, int& counter)

{

if (A.size\_x != A.size\_y)

throw Bad("not quadratic");

Matrix<T> y0(A.size\_x, 1);

for (int i = 0; i < A.size\_x; ++i)

{

y0[i][0] = 1;

}

auto y1 = A \* y0;

auto lambdaPrev = scalar\_product(y1, y0) / scalar\_product(y0, y0);

auto yPrev = A \* y1;

auto lambda = scalar\_product(yPrev, y1) / scalar\_product(y1, y1);

auto u = yPrev, uPrev = yPrev;

while ((lambda - lambdaPrev) >= eps)

{

++counter;

auto nu = A \* u;

lambdaPrev = lambda;

lambda = scalar\_product(nu, u)/ scalar\_product(u, u);

uPrev = u;

u = nu / maximum\_norm(nu);

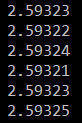
}

return make\_pair(lambda, uPrev);

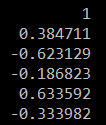
}

**Результаты:**

Собственные значения без использования симметричности матрицы:



Собственный вектор:

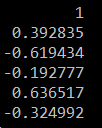


Число итераций: 242.

Собственное значение с использованием симметричности матрицы:



Собственный вектор:



Число итераций: 124.

**Вывод:**

Для данной матрицы степенной итерационный метод сходится. Наибольшее по модулю собственное значение . Как с использованием симметричности матрицы, так и без, результат одинаков, однако с использованием симметричности заданная точность достигается примерно в 2 раза быстрее. За начальный вектор

Был взят вектор , однако и для других векторов вида итерационный процесс сходился к тому же результату за сопоставимое число итераций.

Для данной матрицы её диагонализируемость гарантируется за счет симметричности. Также для заданной матрицы использовался первый случай степенного итерационного метода, при котором наибольшее по модулю собственное значение вещественное и простое (т.е. не кратное).

Алгоритм прост и сходится со скоростью геометрической прогрессии если все максимальные по модулю собственные значения совпадают, в противном случае сходимости нет. При близких по модулю собственных значениях сходимость может оказаться медленной. В силу того, что алгоритм сводится к последовательному умножению заданной матрицы на вектор, при правильной реализации он хорошо работает для больших разреженных матриц.