*Дубкова Валерия КС-26*

*Вариант 5*

**Метод простых итераций**

**Код**

clc

A = [1, 1, 1, -1;

2, -2, 4, 2;

3, 1, 2, -2;

2, 2, -3, 5];

F = [2; -2; -5; 3];

% про матрицу

fprintf('Детерминант матрицы: %.2f\n', det(A));

fprintf('Ранг матрицы: %.2f\n', rank(A));

fprintf('Норма коэффициентов: %.2f\n', norm(A));

fprintf('Число обусловленности: %.2f\n', cond(A));

% метод ПРОСТОЙ ИТЕРАЦИИ

e = 0.0001;

N = length(F);

fprintf('Решение с помощью linsolve:');

fprintf('\t%.2f', linsolve(A, F));

u = [0; 0; 0; 0];

u1 = u;

k = 0.01;

A1 = eye(N) - k \* (A' \* A);

F1 = k \* (A' \* F);

eig(A1);

step = 1;

while true

u1 = A1 \* u + F1;

if abs(norm(u1) - norm(u)) <= e

break;

end

u = u1;

step = step + 1;

if (step >10000)

break;

end

end

u = u1;

accuracy = norm(A \* u - F);

fprintf('\nКоличество итераций i = %d\n', step);

fprintf('Вектор решений:');

fprintf('\t%.2f\t', u);

x1 = inv(A) \* F;

check1 = x1 - u;

fprintf('\n\nПроверка:');

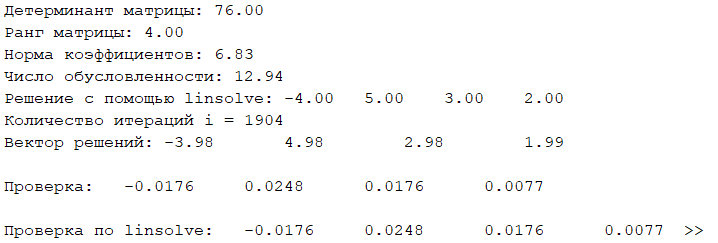
fprintf('\t%.4f\t', check1);

check2 = linsolve(A, F) - u;

fprintf('\n\nПроверка по linsolve:');

fprintf('\t%.4f\t', check2);

**Результат выполнения программы**



**Метод Зейделя**

**Код**

clc

% метод ЗЕЙДЕЛЯ

A0 = [1, 1, 1, -1;

2, -2, 4, 2;

2, 2, -3, 5;

3, 1, 2, -2];

F0 = [2; -2; 3; -5];

e = 0.001;

fprintf('Решение с помощью linsolve:');

fprintf('\t%.2f\t', linsolve(A, F));

% добавить нормализацию

F = A0' \* F0;

A = A0' \* A0;

V = diag(A);

D = diag(V);

LL = tril(A);

L = LL - D;

UU = triu(A);

U = UU - D;

u = [0; 0; 0; 0];

u1 = u;

step = 1;

while true

u1 = -(L+D)\U\*u+(L+D)\F;

if abs(norm(u1) - norm(u)) <= e

break;

end

u = u1;

step = step + 1;

if (step >10000)

break;

end

end

u = u1;

fprintf('\nКоличество итераций i = %d\n',step);

fprintf('Вектор решений:');

fprintf('\t%.2f\t', u);

x1 = inv(A0) \* F0;

check1 = x1 - u;

fprintf('\n\nПроверка:');

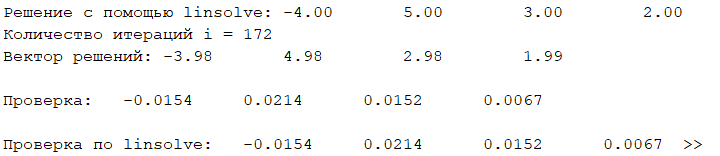
fprintf('\t%.4f\t', check1);

check2 = linsolve(A0, F0) - u;

fprintf('\n\nПроверка по linsolve:');

fprintf('\t%.4f\t', check2);

**Результат выполнения программы**



**Метод Якоби**

**Код**

clc

% метод ЯКОБИ

A0 = [1, 1, 1, -1;

2, -2, 4, 2;

2, 2, -3, 5;

3, 1, 2, -2];

F0 = [2; -2; 3; -5];

E = 0.001;

% добавить нормализацию

F = A0' \* F0;

A = A0' \* A0;

fprintf('Решение с помощью linsolve:');

fprintf('\t%.2f\t', linsolve(A, F));

V = diag(A);

D = diag(V);

LL = tril(A);

L = LL - D;

UU = triu(A);

U = UU - D;

B = -(L + D)^-1 \* U;

F = (L + D)^-1 \* F;

%fprintf('\nНорма B: %.4f\n', norm(B));

u = [0; 0; 0; 0];

u1 = B \* u + F;

step = 0;

while (abs(max(u) - max(u1)) >= E)

u = u1;

u1 = B \* u + F;

step = step + 1;

if (step >1000)

break;

end

end

u = u1;

fprintf('\nКоличество итераций i = %d\n', step);

fprintf('Вектор решений:');

fprintf('\t%.2f\t',u);

x1 = inv(A0) \* F0;

check1 = x1 - u;

fprintf('\n\nПроверка:');

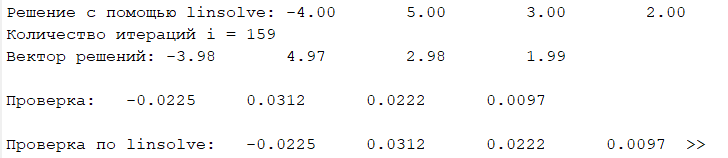
fprintf('\t%.4f\t', check1);

check2 = linsolve(A0, F0) - u;

fprintf('\n\nПроверка по linsolve:');

fprintf('\t%.4f\t', check2);

**Результат выполнения программы**



**Выводы**

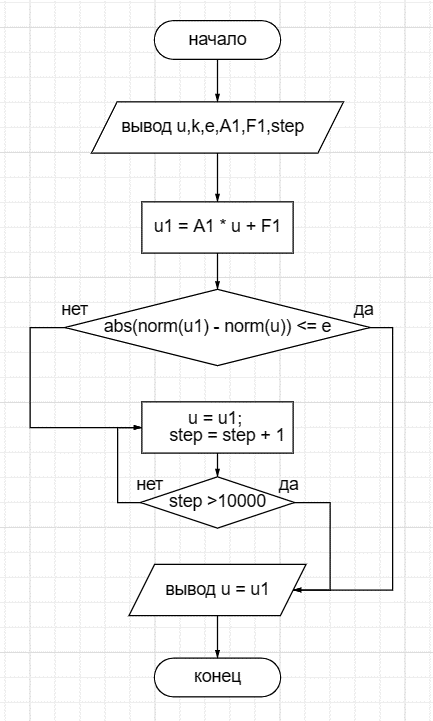
Для качественного вычисления методом Зейделя и Якоби необходимо было нормализовать данные, так как они не сходились. 

По результатам выполнения программы можем сказать, что с данной задачей наименьшим количеством итераций справился метод Якоби, далее идут метод Зейделя и метод простых итераций.

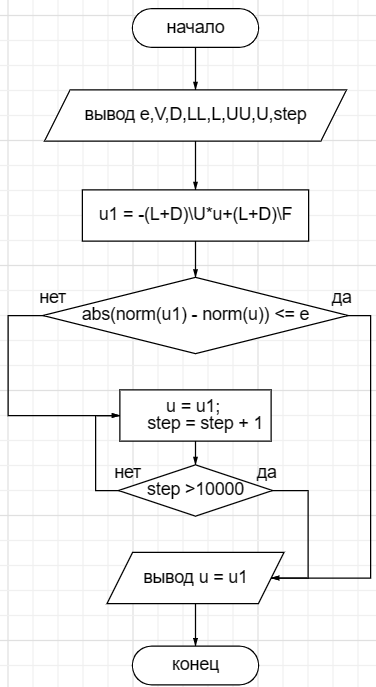
Добавлена проверка по linsolve(), результаты оказались такими же, как при проверке по обратной матрице.

Самым точным же, оказался метод Зейделя, далее метод простых итераций и метод Якоби.

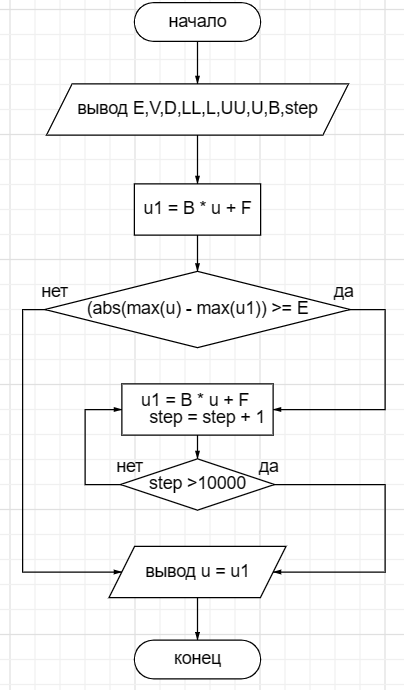
**Блок-схемы**



*Рис.1 Метод простых итераций*



*Рис.2 Метод Зейделя*



*Рис.3 Метод Якоби*