

**MINISTERUL EDUCAȚIEI, CULTURII ȘI CERCETĂRII**

**AL REPUBLICII MOLDOVA**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică şi Microelectronică**

**Departamentul Informatică şi Ingineria Sistemelor**

**gr. IA-231, Chistol Maxim**

**Raport**

**pentru lucrarea de laborator Nr.4**

***la cursul de “Metode Numerice”***

Verificat:

Moraru Vasile**,** asistent.universitar.

Departamentul Informatică şi IS,

Facultatea FCIM, UTM

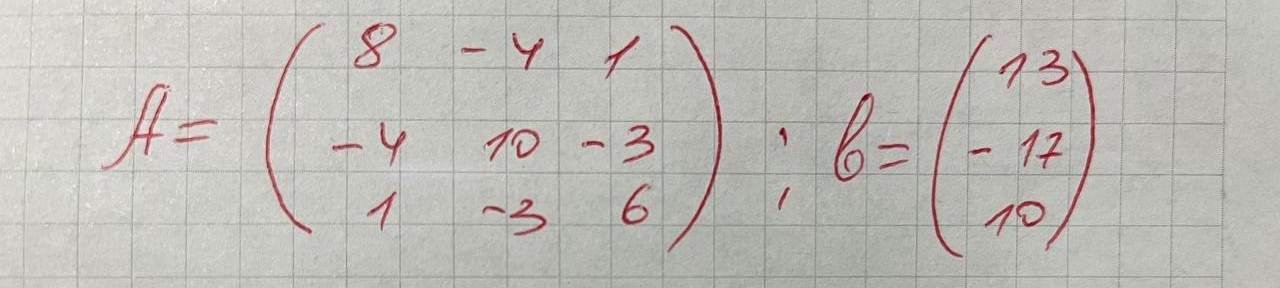
**Chișinău 2024**

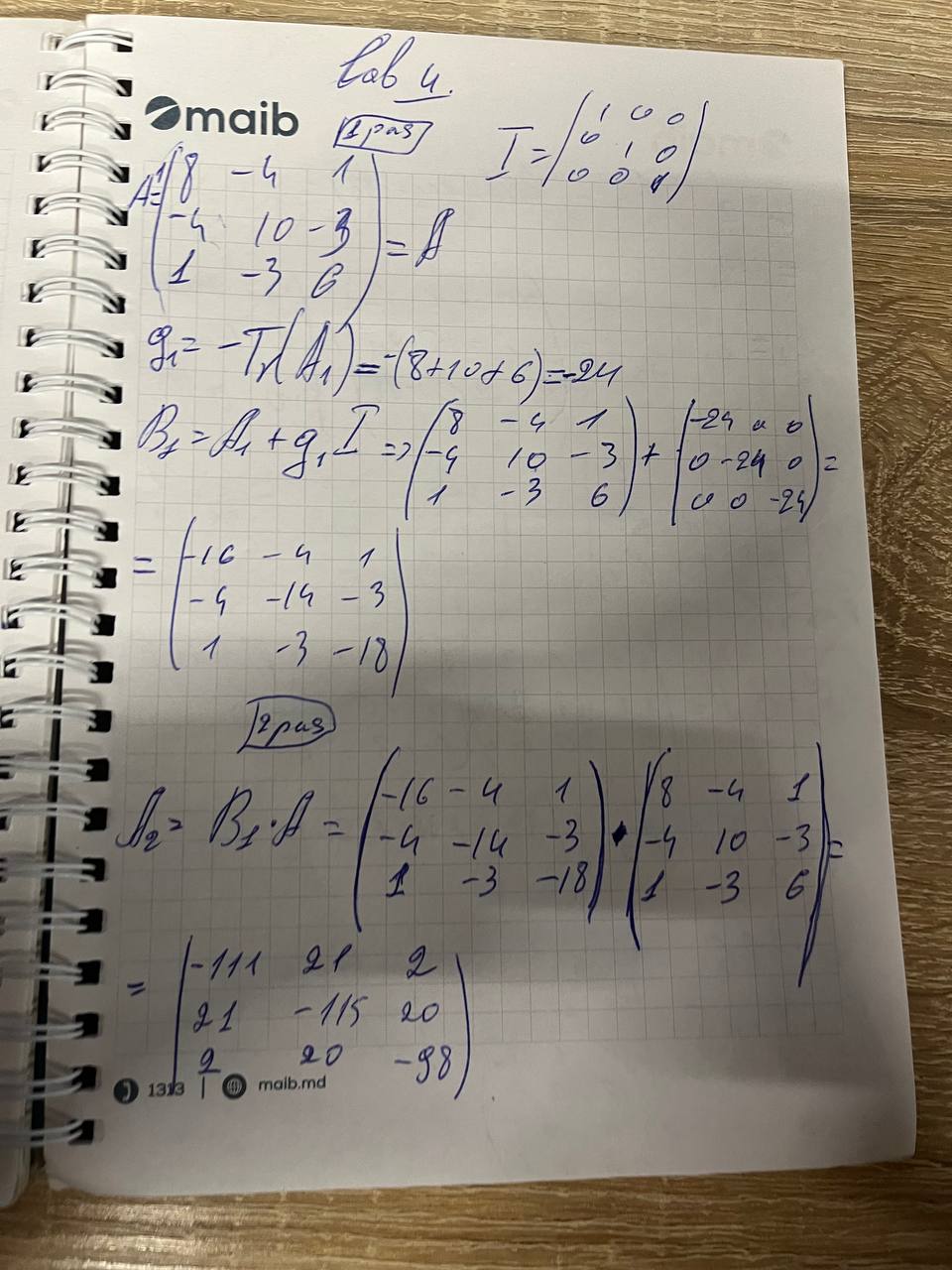
Scopul lucrării:

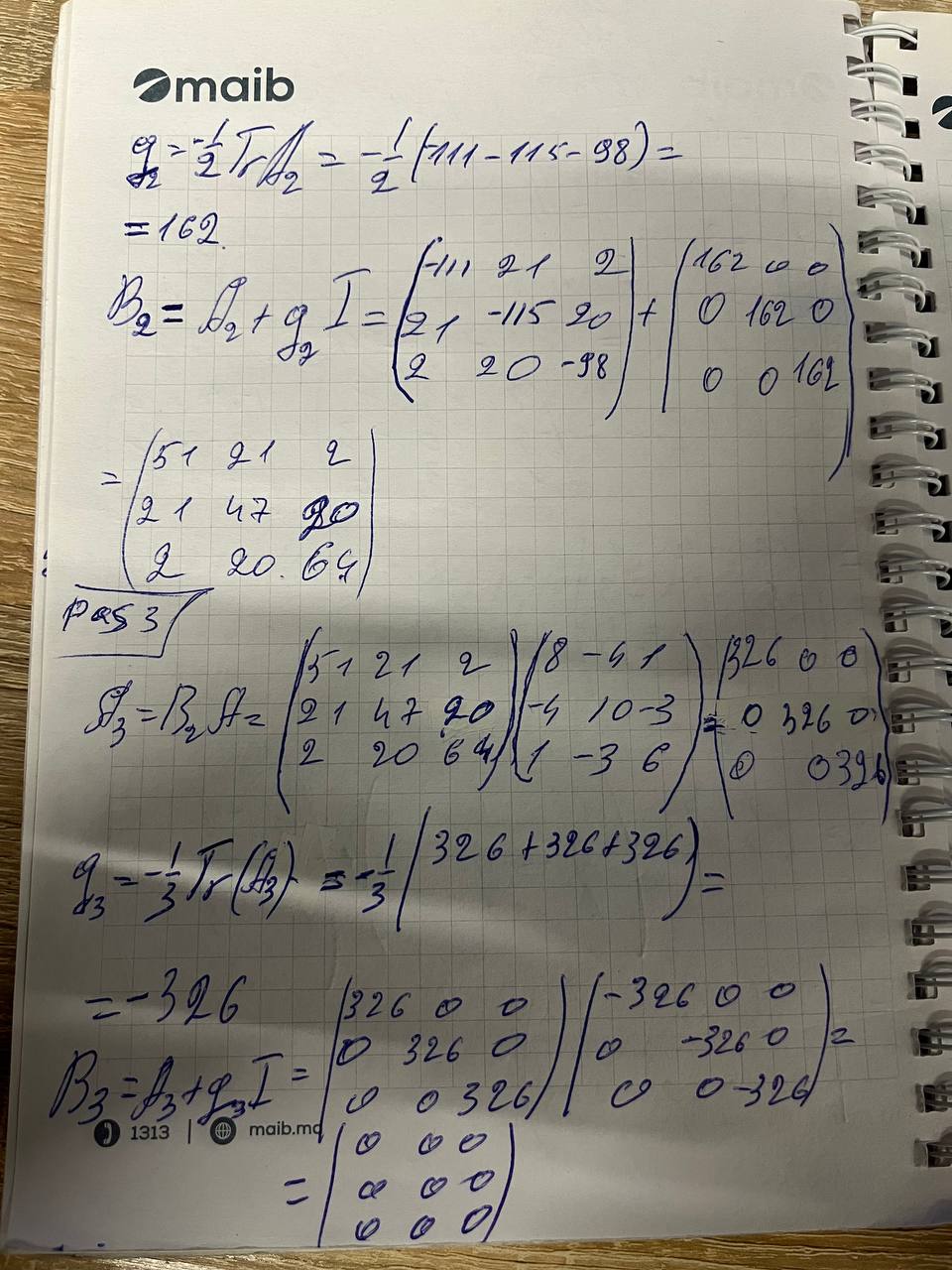
1. Să se aplice metoda lui Fadeev pentru a determina polinomul caracteristic și matricea inversă pentru matricea dată A.

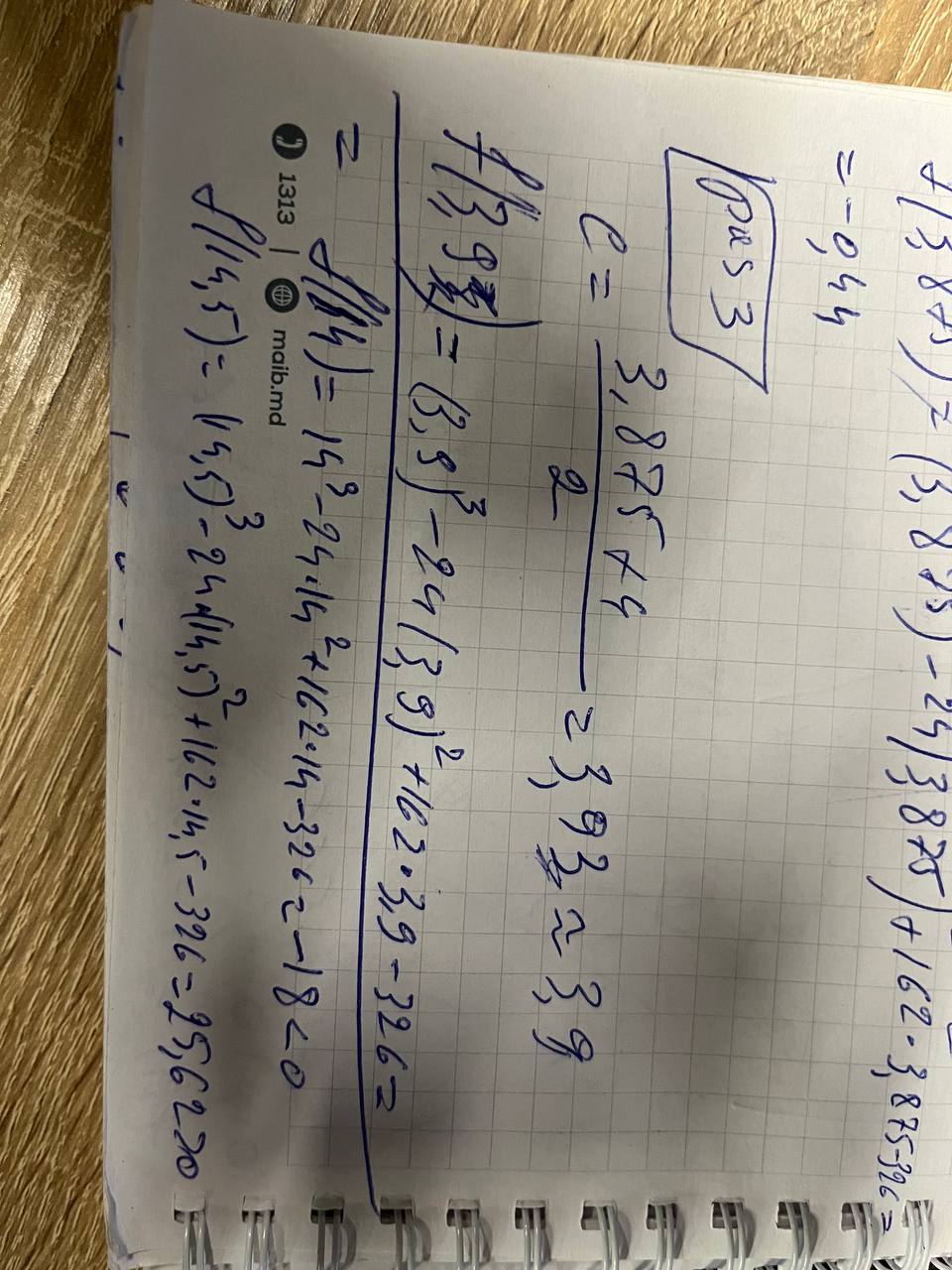
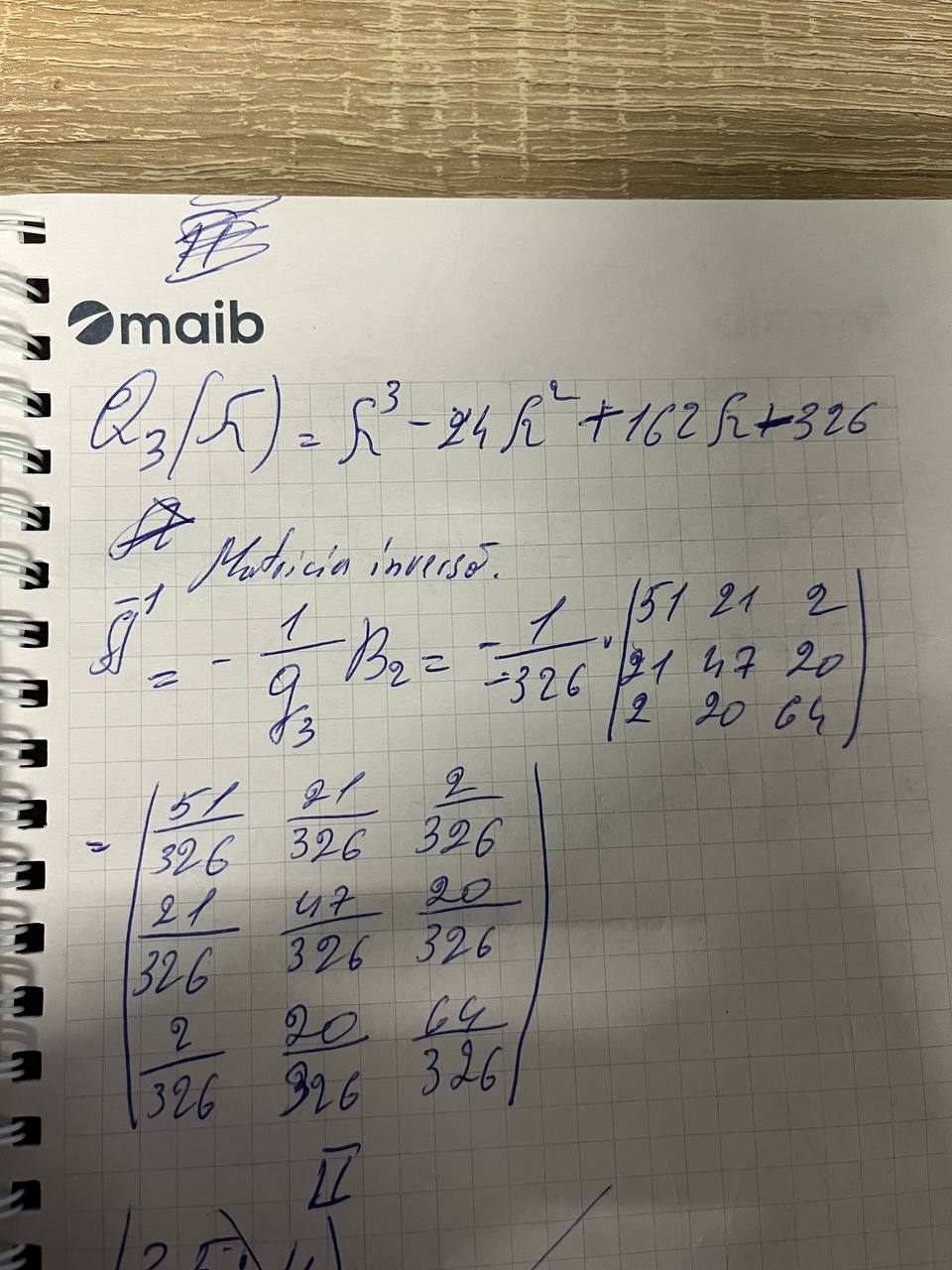
2. Să se determine una din valorile proprii ale matricei A, utilizând metoda înjumătățirii intervalului pentru rezolvarea ecuației caracteristice.

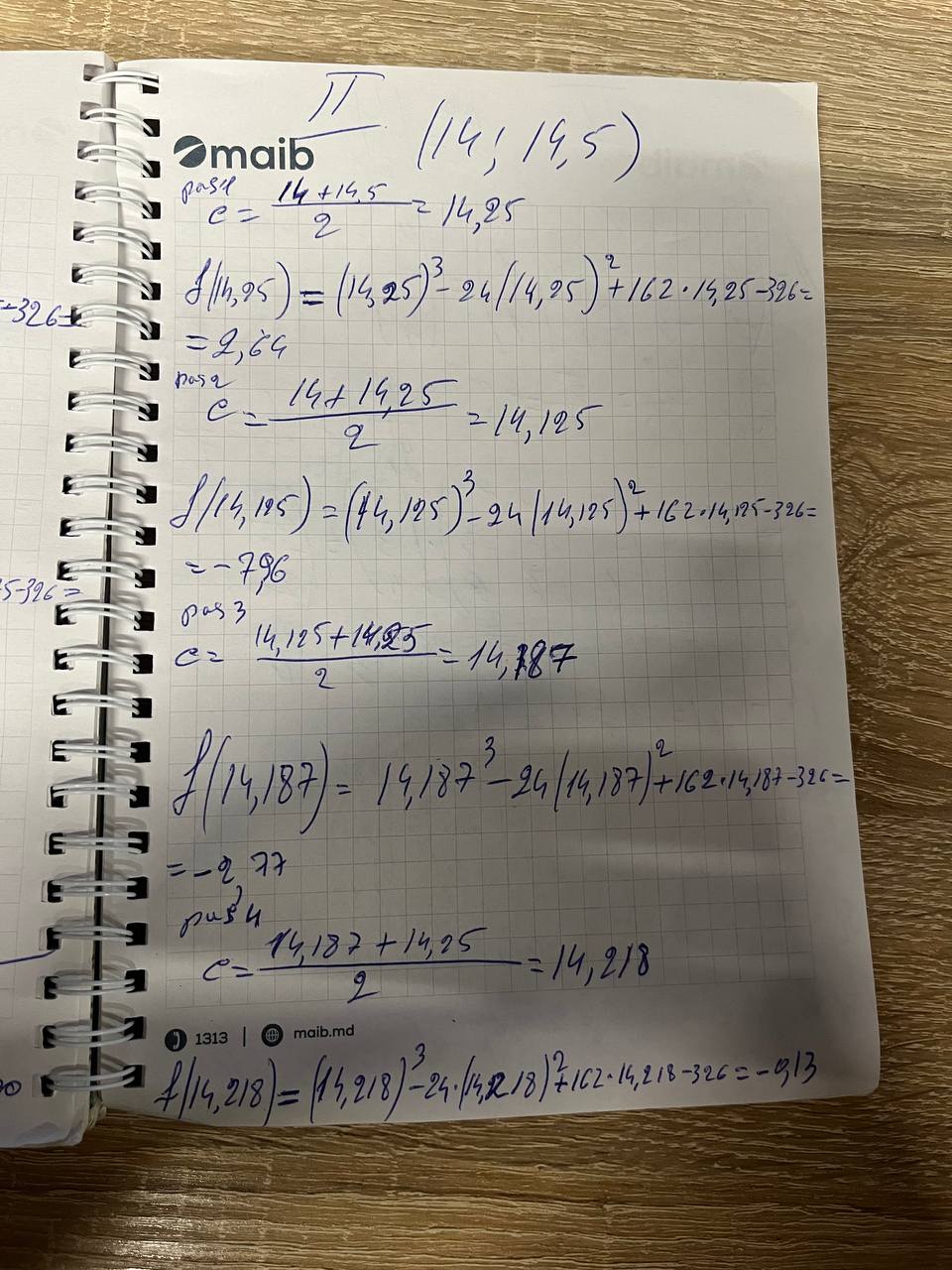
3. Utilizând metoda puterii să se calculeze cea mai mare valoare proprie și vectorul propriu corespunzător al matricei A.



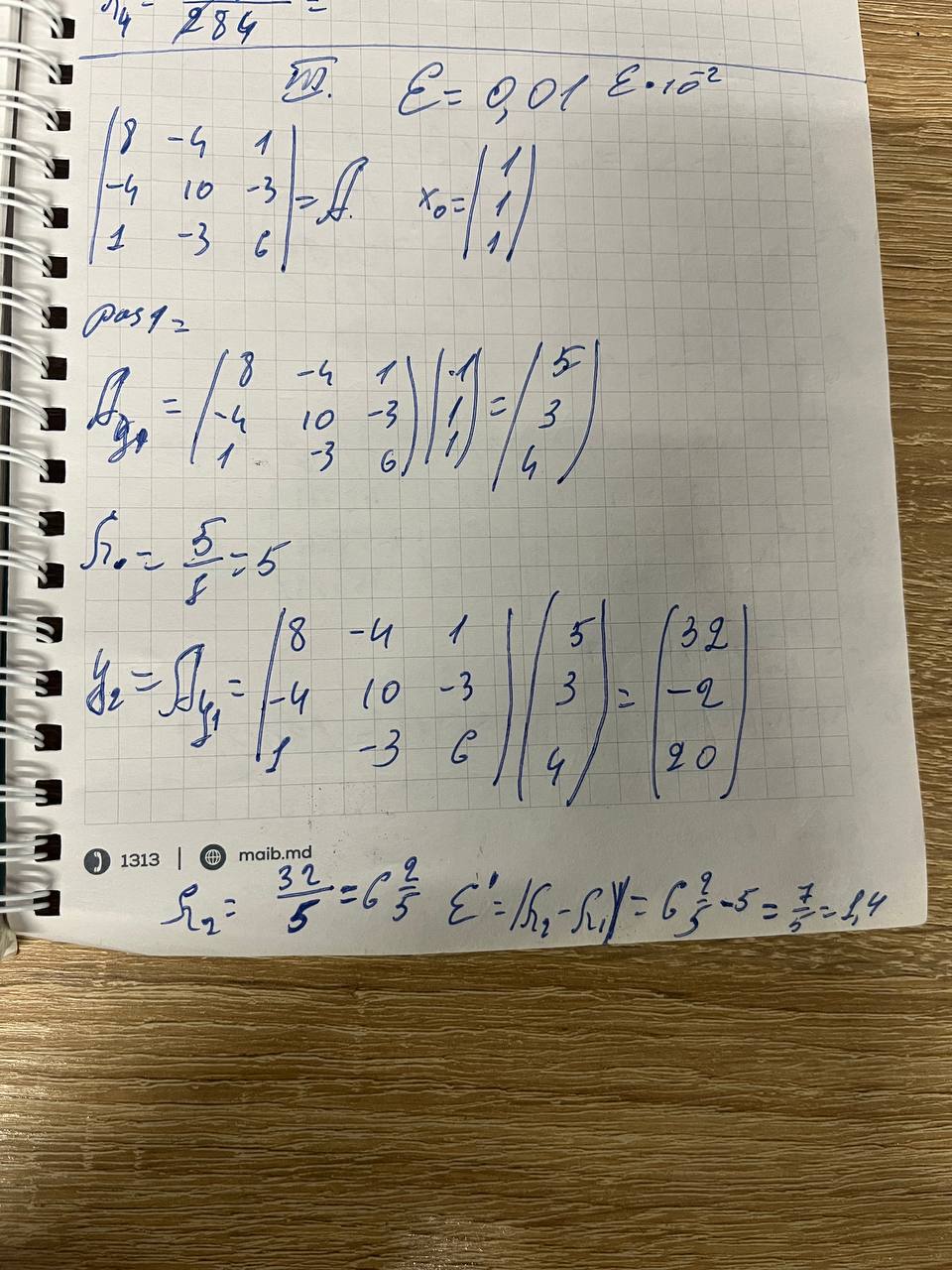


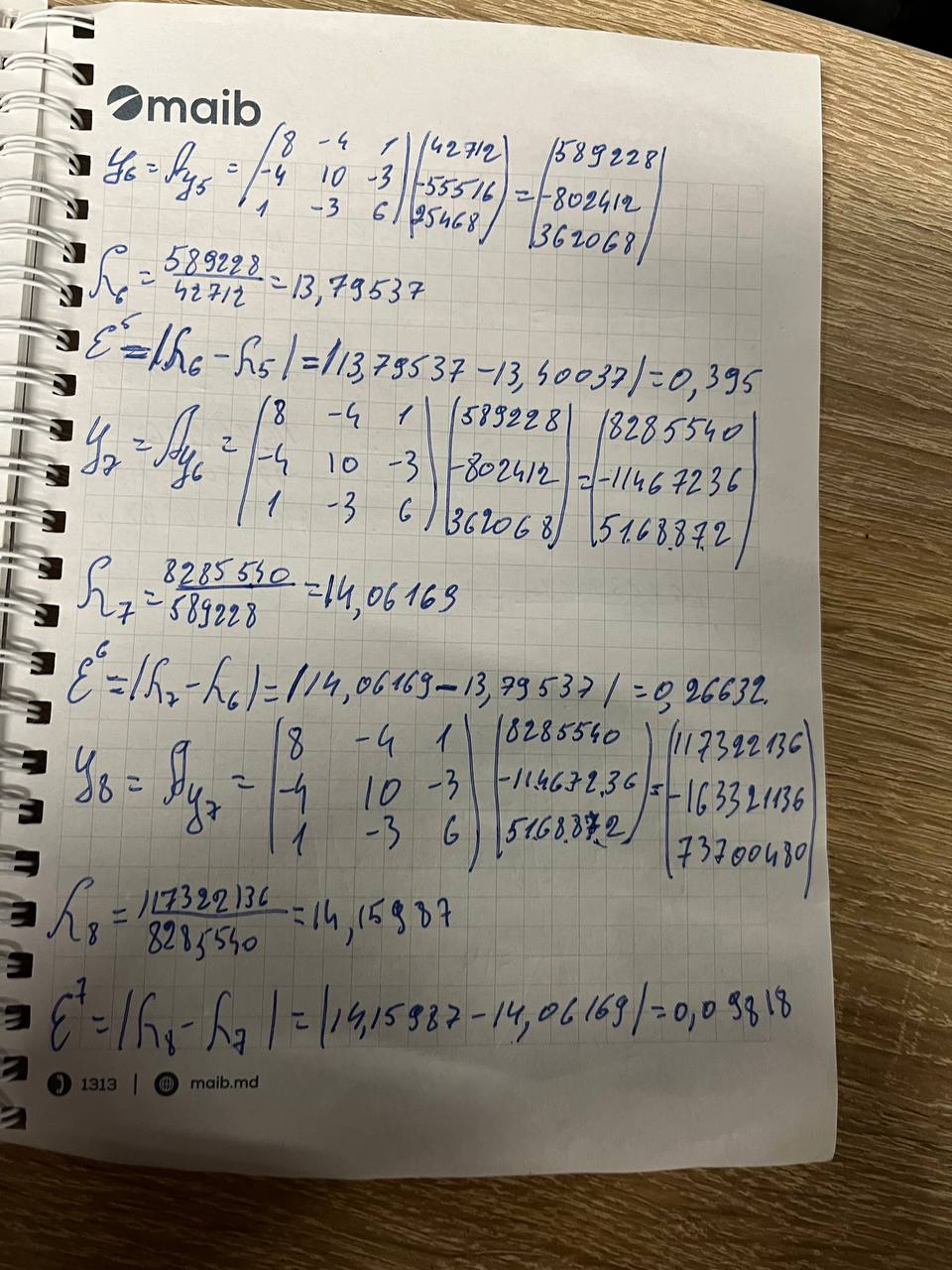


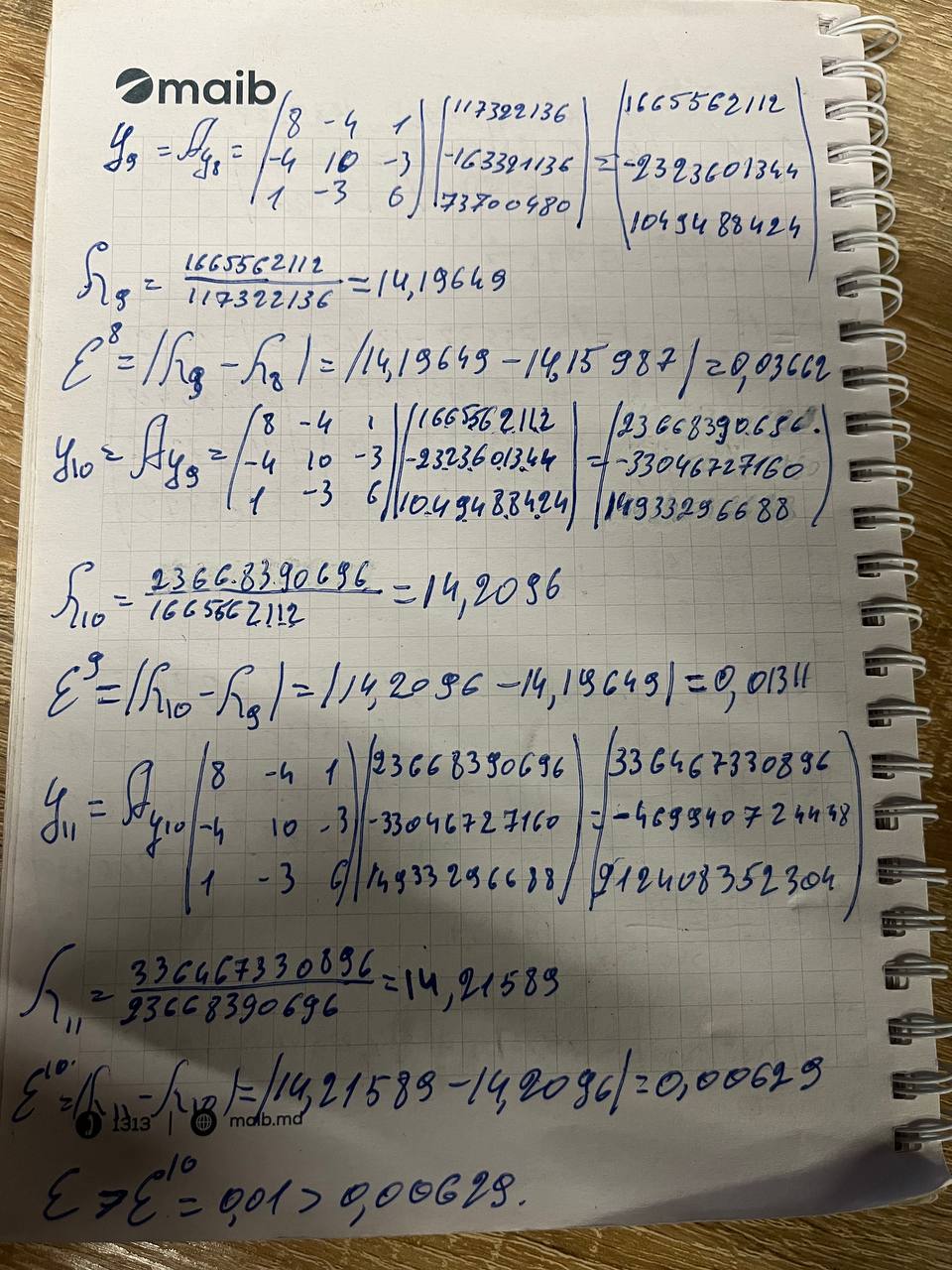
**Figura 1:** Metoda lui Fadeev

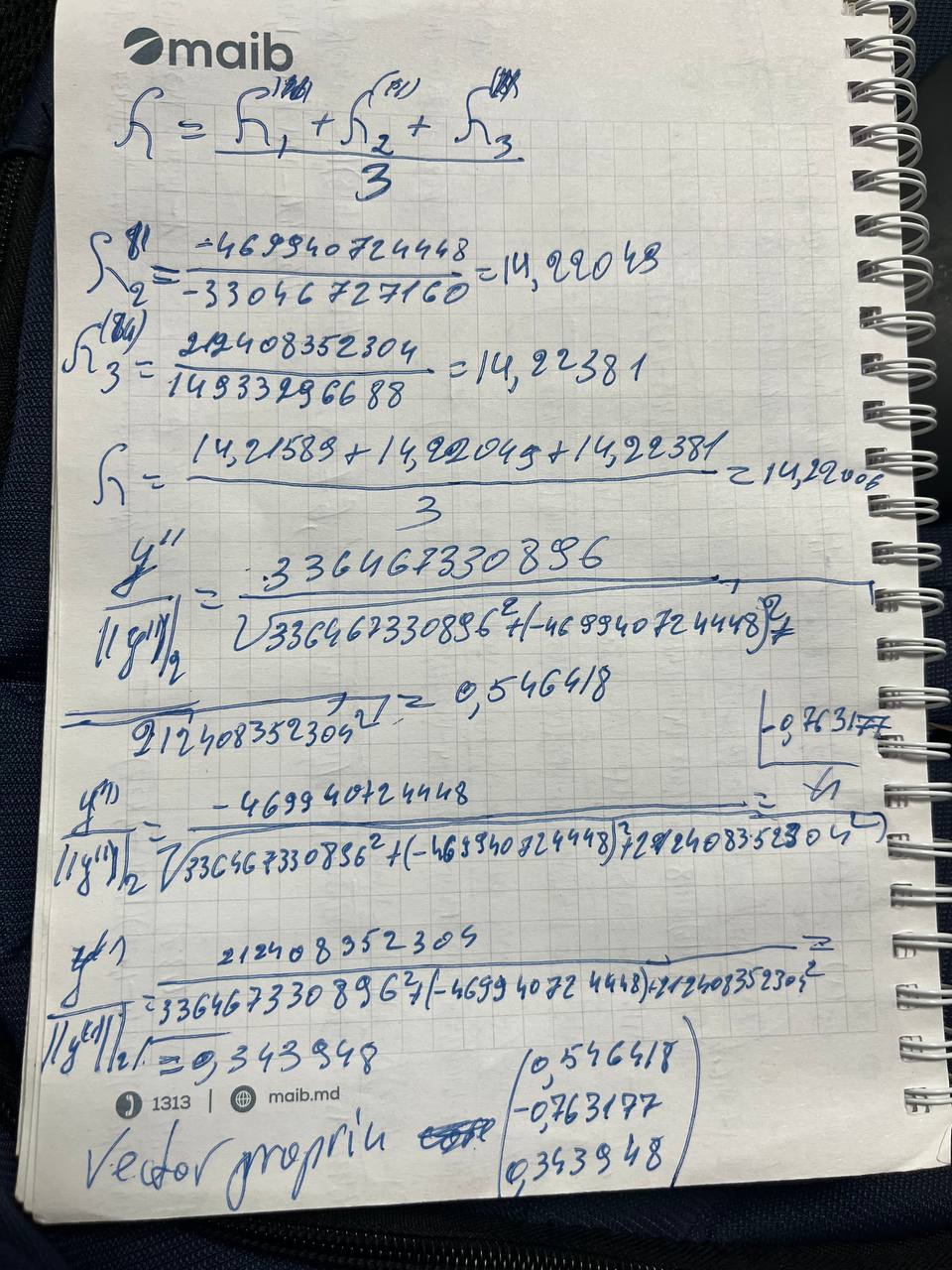


**Figura 2:** Metoda înjumătățirii intervalului









**Figura 3:** Metoda puterii

Codul:

#include <iostream>

#include <iomanip>

#include <vector>

#define N 3 // Dimensiunea matricei

using namespace std;

void displayMatrix(double matrix[N][N], int n) {

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

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

cout << setw(10) << matrix[i][j] << " ";

}

cout << endl;

}

cout << endl;

}

// Funcție pentru a verifica dacă matricea este zero

bool isZeroMatrix(double matrix[N][N]) {

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

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

if (matrix[i][j] != 0) {

return false;

}

}

}

return true;

}

int main() {

// Matricea de intrare A

double A[N][N] = {{8, -4, 1}, {-4, 10, -3}, {1, -3, 6}};

// Matricea identitate I

double I[N][N] = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};

// Inițializare B\_0 ca matrice identitate

double B[N][N] = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};

double q[N + 1] = {0}; // Coeficientii polinomului caracteristic

double B\_prev[N][N]; // Matrice pentru stocarea B\_k anterioare

double temp[N][N]; // Matrice temporară pentru calcule

cout << fixed << setprecision(2);

// Iterăm prin fiecare pas al metodei Fadeev

for (int k = 1; k <= N; ++k) {

// Copiem B în B\_prev pentru pasul curent

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

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

B\_prev[i][j] = B[i][j];

// Calculăm A \* B\_prev

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

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

temp[i][j] = 0;

for (int l = 0; l < N; ++l) {

temp[i][j] += A[i][l] \* B\_prev[l][j];

}

}

}

// Calculăm q\_k ca fiind suma diagonală a A \* B\_prev

q[k] = 0;

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

q[k] += temp[i][i];

}

q[k] = -q[k] / k;

// Afișăm valoarea lui q[k] la fiecare iterație

cout << "q[" << k << "] = " << q[k] << endl;

// Actualizăm B = A \* B\_prev + q\_k \* I

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

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

B[i][j] = temp[i][j] + q[k] \* I[i][j];

}

}

// Afișăm matricea B după fiecare iterație

cout << "Matricea B[" << k << "] după iterația " << k << ":\n";

displayMatrix(B, N);

}

// Afișează polinomul caracteristic

cout << "Polinomul caracteristic: ";

cout << "-lambda^" << N; // Adăugăm termenul de grad maxim explicit

for (int k = 1; k <= N; ++k) { // Începem de la q[1]

double coef = -q[k]; // Inversăm semnele coeficienților

if (coef >= 0) cout << " + " << coef << " \* lambda^" << (N - k);

else cout << " - " << abs(coef) << " \* lambda^" << (N - k);

}

cout << endl;

// Calcularea inversei folosind B[N] și q[N];

cout << "Matricea inversă:\n";

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

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

B[i][j] = -B\_prev[i][j] / q[N]; // Folosește B[N-1] și q[N] pentru inversă

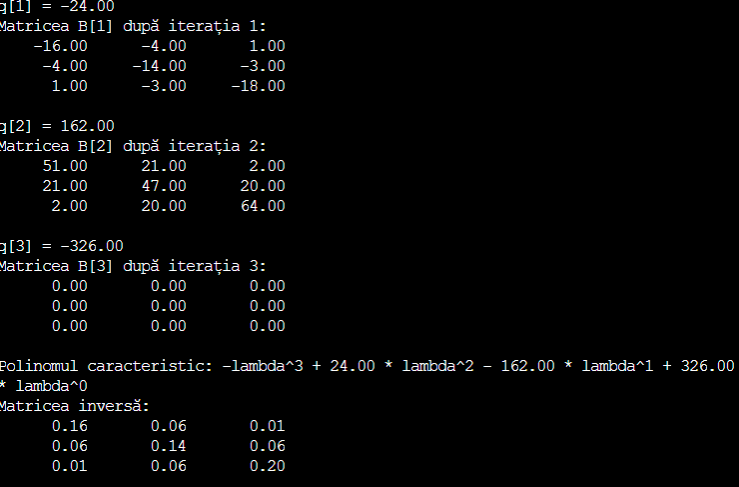
}

}

displayMatrix(B, N);

return 0;

}



**Figura** **4**: output la metoda lui Fadeev

Codul:

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

// Funcție pentru afișarea unei matrice

void printMatrix(double\*\* matrix, int n) {

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

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

printf("%.3f ", matrix[i][j]);

}

printf("\n");

}

printf("\n");

}

// Funcție pentru alocarea memoriei pentru o matrice de dimensiune n x n

double\*\* allocateMatrix(int n) {

double\*\* matrix = (double\*\*)malloc(n \* sizeof(double\*));

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

matrix[i] = (double\*)malloc(n \* sizeof(double));

}

return matrix;

}

// Funcție pentru eliberarea memoriei alocate unei matrice

void freeMatrix(double\*\* matrix, int n) {

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

free(matrix[i]);

}

free(matrix);

}

// Funcție pentru calcularea produsului dintre două matrice

void multiplyMatrices(double\*\* A, double\*\* B, double\*\* result, int n) {

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

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

result[i][j] = 0;

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

result[i][j] += A[i][k] \* B[k][j];

}

}

}

}

// Funcție pentru calcularea urmei unei matrice

double trace(double\*\* matrix, int n) {

double tr = 0;

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

tr += matrix[i][i];

}

return tr;

}

// Funcție pentru adăugarea unei matrice cu o matrice scalară (q \* I)

void addScalarMatrix(double\*\* A, double q, double\*\* result, int n) {

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

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

result[i][j] = A[i][j];

}

result[i][i] += q; // Adăugăm q pe diagonală

}

}

// Funcție pentru calcularea polinomului caracteristic într-un punct

double characteristicPolynomialValue(double\* q, int n, double x) {

double result = pow(x, n);

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

result += q[i] \* pow(x, n - i);

}

return result;

}

// Metoda înjumătățirii intervalului pentru rezolvarea ecuației caracteristice

double bisectionMethod(double (\*func)(double, double\*, int), double\* q, int n, double a, double b, double tol) {

double fa = func(a, q, n);

double fb = func(b, q, n);

if (fa \* fb >= 0) {

printf("Interval invalid pentru metoda bisection.\n");

return NAN;

}

double c;

while ((b - a) / 2 > tol) {

c = (a + b) / 2;

double fc = func(c, q, n);

if (fc == 0.0) {

return c; // Soluție exactă

} else if (fa \* fc < 0) {

b = c;

fb = fc;

} else {

a = c;

fa = fc;

}

}

return (a + b) / 2; // Aproximare

}

int main() {

int n;

// Citirea dimensiunii matricei

printf("Introduceti dimensiunea matricei: ");

scanf("%d", &n);

// Alocarea și citirea matricei A

double\*\* A = allocateMatrix(n);

printf("Introduceti elementele matricei A:\n");

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

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

scanf("%lf", &A[i][j]);

}

}

// Alocarea altor matrice și vectori necesari

double\*\* Ai = allocateMatrix(n); // A\_1 inițial

double\*\* Bi = allocateMatrix(n);

double\* q = (double\*)malloc((n + 1) \* sizeof(double)); // Coeficienții polinomului caracteristic

// Inițializarea lui Ai cu A

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

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

Ai[i][j] = A[i][j];

}

}

// Calcul iterativ al coeficienților q și matricilor B

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

double trAi = trace(Ai, n);

q[i] = -trAi / i;

addScalarMatrix(Ai, q[i], Bi, n);

if (i < n) {

multiplyMatrices(Bi, A, Ai, n);

}

}

// Afișarea polinomului caracteristic

printf("Polinomul caracteristic este: ");

printf("x^%d", n);

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

if (q[i] >= 0) printf(" + ");

else printf(" - ");

printf("%.2f", fabs(q[i]));

printf("x^%d", (n - i));

}

printf("\n");

// Determinarea unei valori proprii utilizând metoda înjumătățirii intervalului

double a, b, tol;

printf("\nIntroduceti intervalul [a, b] si eroarea: ");

scanf("%lf %lf %lf", &a, &b, &tol);

// Definim funcția polinomului caracteristic pentru metoda bisection

double characteristicFunc(double x, double\* q, int n) {

return characteristicPolynomialValue(q, n, x);

}

double eigenvalue = bisectionMethod(characteristicFunc, q, n, a, b, tol);

if (!isnan(eigenvalue)) {

printf("Valoare proprie: %.2f\n", eigenvalue);

} else {

printf("Nu s-a putut determina o valoare proprie in intervalul dat.\n");

}

// Eliberarea memoriei

freeMatrix(A, n);

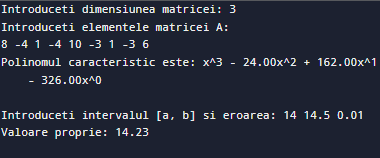
freeMatrix(Ai, n);

freeMatrix(Bi, n);

free(q);

return 0;

}



**Figura 5:** Metoda înjumătățirii intervalului

#include <stdio.h>

#include <math.h>

#define EPSILON 1e-6 // Toleranța pentru convergență

int main() {

double matrix[3][3];

double vector[3] = {1, 1, 1}; // Vector inițial

double nextVector[3];

double eigenValue = 0, oldEigenValue = 0;

int i, j, iter = 0;

// Citirea matricei

printf("Introduceti elementele matricei 3x3:\n");

for (i = 0; i < 3; i++) {

for (j = 0; j < 3; j++) {

scanf("%lf", &matrix[i][j]);

}

}

// Metoda puterii

while (1) {

iter++;

// Calcularea produsului matrice \* vector

for (i = 0; i < 3; i++) {

nextVector[i] = 0;

for (j = 0; j < 3; j++) {

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

}

}

// Estimarea valorii proprii

eigenValue = nextVector[0] / vector[0];

// Normalizarea vectorului

double norm = 0;

for (i = 0; i < 3; i++) {

norm += nextVector[i] \* nextVector[i];

}

norm = sqrt(norm);

for (i = 0; i < 3; i++) {

nextVector[i] /= norm;

}

// Verificarea convergenței

if (fabs(eigenValue - oldEigenValue) < EPSILON) {

break;

}

oldEigenValue = eigenValue;

// Actualizarea vectorului

for (i = 0; i < 3; i++) {

vector[i] = nextVector[i];

}

}

// Afișarea rezultatului

printf("Metoda puterii a convergent dupa %d iteratii.\n", iter);

printf("Cea mai mare valoare proprie (in valoare absoluta): %.6f\n", eigenValue);

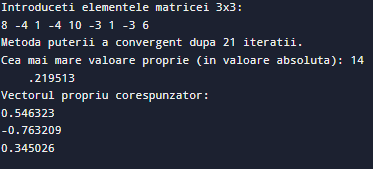
printf("Vectorul propriu corespunzator:\n");

for (i = 0; i < 3; i++) {

printf("%.6f\n", nextVector[i]);

}

return 0;

}  
  


**Figura 6:** Metoda puterii

# Concluzii:

În urma aplicării metodei lui Fadeev, a fost determinat polinomul caracteristic al matricei AA și matricea sa inversă, consolidând înțelegerea legăturii dintre operațiile matriciale și polinoamele caracteristice. Identificarea unei valori proprii prin metoda înjumătățirii intervalului a demonstrat eficiența numerică în rezolvarea ecuațiilor caracteristice. Totodată, utilizarea metodei puterii a permis calcularea celei mai mari valori proprii, împreună cu vectorul propriu asociat, subliniind relevanța acestei metode în analiza spectrală a matricelor. Aceste tehnici evidențiază importanța algoritmilor numerici în problemele de algebră liniară și oferă un fundament solid pentru aplicarea lor în diverse domenii practice.