TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA CÔNG NGHỆ THÔNG TIN

ĐỀ THI GIỮA HỌC KỲ VÀ BÀI LÀM

Tên học phần: Toán ứng dụng CNTT
Mã học phần: Hình thức thi: *Tự luận*
Đề số: **03** Thời gian làm bài: 90 phút *(không kể thời gian chép/phát đề)*
Được sử dụng tài liệu khi làm bài.

Họ tên: Nguyễn Chí Thành **Lớp**: 23T_DT2 **MSSV**:102230268

Sinh viên làm bài trực tiếp trên tệp này, lưu tệp với định dạng MSSV_HọTên.pdf và nộp bài thông qua MSTeam

Câu 1 (*2 điểm*): Viết chương trình (có sử dụng hàm) thực hiện công việc sau, biết rằng M=200, N=500. Tìm số lượng các số nguyên tố nằm trong khoảng (M, N), liệt kê và tính tổng của chúng. Trong các số nguyên tố vừa tìm được, *số nào gần với số 435 nhất*.

 # Trả lời: Dán code vào bên dưới:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int isPrime(long long n) {
    if (n<=1) return 0;
    for (int i=2; i<=sqrt(n); i++) {
        if (n%i==0) return 0;
    }
    return 1;
}

long long findPrimes(long long M, long long N, long long *primes) {
    int count = 0;

    for (int i = M + 1; i < N; i++) {
        if (isPrime(i)) {
            primes[count++] = i;
        }
    }
```

```
    return count;
}

long long calculateSum(long long arr[], long long n) {
    long long sum = 0;
    for (int i = 0; i < n; i++) {
        sum += arr[i];
    }
    return sum;
}

long long timsogannhat(long long arr[], long long n, long long target) {
    if (n == 0) return -1;

    long long closestNumber = arr[0];
    long long minDifference = abs(arr[0] - target);

    for (int i = 1; i < n; i++) {
        long long difference = abs(arr[i] - target);

        if (difference < minDifference) {
            minDifference = difference;
            closestNumber = arr[i];
        }
    }

    return closestNumber;
}

int main() {
    long long M = 200;
    long long N = 500;
    long long TARGET = 435;

    long long *primes = (long long*)malloc((N - M) * sizeof(long long));
```

```
    long long count = findPrimes(M, N, primes);
    long long sum = calculateSum(primes, count);
    long long closest = timsogannhat(primes, count, TARGET);

    printf("So luong so nguyen to trong khoang (%lld, %lld): %lld\n", M, N, count);

    printf("Cac so nguyen to tim duoc:\n");
    for (long long i = 0; i < count; i++) {
        printf("%lld ", primes[i]);
        if ((i + 1) % 20 == 0) printf("\n");
    }
    printf("\n");

    printf("Tong cac so nguyen to: %lld\n", sum);
    printf("So nguyen to gan voi %lld nhat trong cac so tim duoc: %lld\n", TARGET,
closest);

    free(primes);

    return 0;
}
```

Trả lời: Dán kết quả thực thi vào bên dưới:

```
So luong so nguyen to trong khoang (200, 500): 49
Cac so nguyen to tim duoc:
211 223 227 229 233 239 241 251 257 263 269 271 277 281 283 293 307 311 313 317
331 337 347 349 353 359 367 373 379 383 389 397 401 409 419 421 431 433 439 443
449 457 461 463 467 479 487 491 499
Tong cac so nguyen to: 17309
So nguyen to gan voi 435 nhat trong cac so tim duoc: 433
PS D:\Code\cpp\ToanUngDungCNTT\KTGK>
```

Câu 2: *(2 điểm) Cho hệ phương trình đồng dư sau*

$$\begin{cases} x \equiv 5 \bmod 9 \\ x \equiv 7 \bmod 11 \\ x \equiv 11 \bmod 13 \\ x \equiv 13 \bmod 17 \\ x \equiv 17 \bmod 19 \end{cases}$$

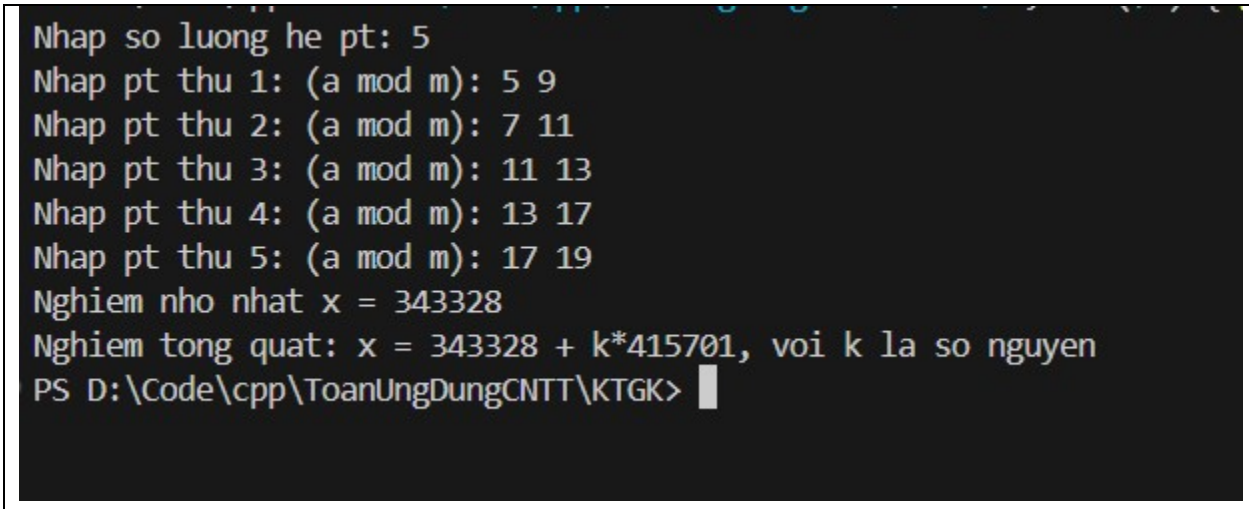
- Viết chương trình C/C++ có sử dụng hàm giải hệ phương trình đồng dư trên.

Trả lời: Dán code vào bên dưới:

```c
#include <stdio.h>

long long mul_mod(long long a, long long b, long long mod)
{
    return (a % mod) * (b % mod) % mod;
}

long long extended_gcd(long long a, long long b, long long *x, long long *y)
{
    if (b == 0)
    {
        *x = 1;
        *y = 0;
        return a;
    }
    long long x1, y1;
    long long gcd = extended_gcd(b, a % b, &x1, &y1);
    *x = y1;
    *y = x1 - (a / b) * y1;
    return gcd;
}

long long mod_inverse(long long a, long long m)
{
    long long x, y;
    long long g = extended_gcd(a, m, &x, &y);
    if (g != 1)
    {
        return -1;
    }
    else
    {
        return (x % m + m) % m;
    }
}

long long chinese_remainder_theorem(int n, long long a[], long long m[], long long
*M_out)
{
    long long M = 1;
```

```
    for (int i = 0; i < n; i++)
    {
        M *= m[i];
    }
    *M_out = M;

    long long result = 0;
    for (int i = 0; i < n; i++)
    {
        long long Mi = M / m[i];
        long long inv = mod_inverse(Mi, m[i]);
        result = (result + mul_mod(mul_mod(a[i], Mi, M), inv, M)) % M;
    }

    return (result + M) % M;
}

int main()
{
    /*  x = 2 (mod 5), x = 4 (mod 7), x = 6 (mod 11) */
    int n;
    printf("Nhap so luong he pt: ");
    scanf("%d", &n);
    long long a[n], m[n];
    for (int i = 0; i < n; i++)
    {
        printf("Nhap pt thu %d: (a mod m): ", i + 1);
        scanf("%lld %lld", &a[i], &m[i]);
    }

    long long M;
    long long x0 = chinese_remainder_theorem(n, a, m, &M);

    printf("Nghiem nho nhat x = %lld\n", x0);
    printf("Nghiem tong quat: x = %lld + k*%lld, voi k la so nguyen\n", x0, M);

    return 0;
}
```

Trả lời: Dán kết quả thực thi vào bên dưới:



Câu 3 (*3 điểm*): Cho ma trận A. Viết chương trình bằng c/c++ có sử dụng hàm thực hiện phân rã ma trận A.

a) Phân rã LDL^T ma trận A

 # Trả lời: Dán code vào bên dưới (bao gồm điều kiện của ma trận A nếu có):

Điều kiện ma trận A: ma trận là ma trận vuông và đối xứng qua đường chéo chính

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#define MAX_N 10
//Điều kiện của ma trận A là: ma trận vuông, đối xứng qua chéo chính
void input_matrix(double **A, int n) {
    printf("Nhap ma tran %dx%d:\n", n, n);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%lf", &A[i][j]);
        }
    }
}
void print_ldlt(double **L, double *D, int n) {

    printf("\nMa tran L:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
```

```
            printf("%10.5f ", L[i][j]);
        printf("\n");
    }
     printf("\nMa tran D (diagonal):\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (i == j)
                printf("%10.5f ", D[i]);
            else
                printf("%10.5f ", 0.0);
        }
        printf("\n");
    }

    printf("\nMa tran L^T (transpose):\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
            printf("%10.5f ", L[j][i]);
        printf("\n");
    }
}
void print_matrix(double **A, int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
            printf("%10.5f ", A[i][j]);
        printf("\n");
    }
}

double** allocate_matrix(int n) {
    double **matrix = (double**)malloc(n * sizeof(double*));
    for (int i = 0; i < n; i++) {
        matrix[i] = (double*)malloc(n * sizeof(double));
```

```
        for (int j = 0; j < n; j++) {
            matrix[i][j] = 0.0;
        }
    }
    return matrix;
}

void free_matrix(double **matrix, int n) {
    for (int i = 0; i < n; i++) {
        free(matrix[i]);
    }
    free(matrix);
}

//  LDL^T
int ldlt_decomposition(double **A, double **L, double *D, int n) {
    for (int i = 0; i < n; i++) {
        //  D[i]
        double sum = 0.0;
        for (int k = 0; k < i; k++)
            sum += L[i][k] * L[i][k] * D[k];
        D[i] = A[i][i] - sum;
        if (fabs(D[i]) < 1e-5) return 0; // Không khả nghịch

        //  L[j][i]
        for (int j = i + 1; j < n; j++) {
            sum = 0.0;
            for (int k = 0; k < i; k++)
                sum += L[j][k] * L[i][k] * D[k];
            L[j][i] = (A[j][i] - sum) / D[i];
        }
    }
    //  L[i][i] = 1, L[i][j] = 0 với j > i
```

```
    for (int i = 0; i < n; i++) {
        L[i][i] = 1.0;
        for (int j = i + 1; j < n; j++)
            L[i][j] = 0.0;
    }
    return 1;
}

int verify_ldlt(double **A, double **L, double *D, int n, double *max_error) {
    double **temp = allocate_matrix(n);
    double **result = allocate_matrix(n);
    int i, j, k;

    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            temp[i][j] = L[i][j] * D[j];
        }
    }

    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            result[i][j] = 0.0;
            for (k = 0; k < n; k++) {
                result[i][j] += temp[i][k] * L[j][k];  // L[j][k] = L^T[k][j]
            }
        }
    }

    *max_error = 0.0;
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            double error = fabs(A[i][j] - result[i][j]);
            if (error > *max_error) {
```

```
                *max_error = error;
            }
        }
    }

    printf("\nMa tran ket qua L*D*L^T:\n");
    print_matrix(result, n);

    printf("\nSai so lon nhat: %.8f\n", *max_error);

    int is_valid = (*max_error < 1e-5);

    if (is_valid) {
        printf("=> Kiem tra THANH CONG! A = L*D*L^T\n");
    } else {
        printf("=> Kiem tra THAT BAI! Sai so qua lon.\n");
    }
    free_matrix(temp, n);
    free_matrix(result, n);
    return is_valid;
}

int main() {
    int n;

    printf("Nhap cap ma tran (1-%d): ", MAX_N);
    scanf("%d", &n);

    if (n < 1 || n > MAX_N) {
        printf("Cap ma tran khong hop le!\n");
        return 1;
    }
```

```
    double **A = allocate_matrix(n);
    double **L = allocate_matrix(n);
    double *D = (double*)calloc(n, sizeof(double));

    input_matrix(A, n);

    printf("\nMa tran A da nhap:\n");
    print_matrix(A, n);

    if (!ldlt_decomposition(A, L, D, n)) {
        printf("Ma tran khong kha nghich hoac khong phan tich duoc.\n");
        free_matrix(A, n);
        free_matrix(L, n);
        free(D);
        return 1;
    }

    print_ldlt(L, D, n);

    printf("KIEM TRA PHAN TICH LDLT\n");
    double max_error;
    verify_ldlt(A, L, D, n, &max_error);

    // Giải phóng bộ nhớ
    free_matrix(A, n);
    free_matrix(L, n);
    free(D);

    return 0;
}
```

Trả lời: Dán kết quả thực thi vào bên dưới với $A = \begin{bmatrix} -10 & 15 & 16 \\ 15 & -11 & 19 \\ 16 & 19 & -13 \end{bmatrix}$ (sai số $\varepsilon = 10^{-5}$):

```
Nhap cap ma tran (1-10): 3
Nhap ma tran 3x3:
-10 15 16
15 -11 19
16 19 -13

Ma tran A da nhap:
 -10.00000   15.00000   16.00000
  15.00000  -11.00000   19.00000
  16.00000   19.00000  -13.00000

Ma tran L:
   1.00000    0.00000    0.00000
  -1.50000    1.00000    0.00000
  -1.60000    3.73913    1.00000

Ma tran D (diagonal):
 -10.00000    0.00000    0.00000
   0.00000   11.50000    0.00000
   0.00000    0.00000 -148.18261

Ma tran L^T (transpose):
   1.00000   -1.50000   -1.60000
   0.00000    1.00000    3.73913
   0.00000    0.00000    1.00000
KIEM TRA PHAN TICH LDLT

Ma tran ket qua L*D*L^T:
 -10.00000   15.00000   16.00000
  15.00000  -11.00000   19.00000
  16.00000   19.00000  -13.00000

Sai so lon nhat: 0.00000000
=> Kiem tra THANH CONG! A = L*D*L^T
PS D:\Code\cpp\ToanUngDungCNTT\KTGK>
```

b) Phân rã **eigendecomposition** ma trận A

Trả lời: Dán code vào bên dưới (bao gồm điều kiện của ma trận A nếu có):

Điều kiện ma trận A:

- Ma trận vuông, phải khả nghịch (khả quy)
- Có đầy đủ vector riêng, trị riêng và chúng độc lập tuyến tính

```cpp
#include <bits/stdc++.h>

#define PI 3.141592654
#define MAX_SIZE 10
using namespace std;

typedef double matrix[MAX_SIZE][MAX_SIZE];
double B1[MAX_SIZE][MAX_SIZE] = {0.0};

void inputMatrix(int n, matrix A);
void outputMatrix(int n, const matrix A);
void multiMatrix(const matrix A, const matrix B, matrix C, int cola, int rowa, int
rowb);
void Danhilepski(matrix A, matrix M, matrix M1, matrix B, int n);
void solution(double a, double b, double c, double d, double x[]);
double determinant(const matrix A, int n);
bool invertMatrix(const matrix A, matrix inverse, int n);
void sortEigenvalues(double lambda[], int n);
void verifyEigendecomposition(const matrix A, const matrix S, const matrix D, const
matrix Sinv, const double lambda[], int n);


int main()
{

    for (int i = 0; i < 3; ++i)
    {
        for (int j = 0; j < 3; ++j)
        {
            B1[i][j] = (i == j) ? 1.0 : 0.0;
        }
    }

    matrix A, M, M1, B, tempA;
    double lambda[3] = {0, 0, 0};
    int n;

    cout << "Enter n: ";
    cin >> n;

    inputMatrix(n, A);

    // Copy original A to tempA for verification later
    for(int i=0; i<n; ++i) {
        for(int j=0; j<n; ++j) {
            tempA[i][j] = A[i][j];
        }
    }

    Danhilepski(A, M, M1, B, n);

    if (n == 3 && std::fabs(A[1][0]) > 1e-9 && std::fabs(A[2][0]) > 1e-9)
```

```
    {
        cout << "Warning: Danhilepski might not have succeeded in full
transformation." << endl;
    }

    solution(1.0, -A[0][0], -A[0][1], -A[0][2], lambda);
    sortEigenvalues(lambda, n);

    cout << "\nDanhilepski's solution: Eigen values:" << endl;

    for (int i = 0; i < n; ++i)
    {
        cout << setprecision(4) << fixed << "lambda[" << i << "] = " << lambda[i] <<
endl;
    }

    if (n == 3)
    {
        double y[3][MAX_SIZE], x[3][MAX_SIZE];

        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < 3; j++)
            {
                y[j][i] = pow(lambda[i], 3 - j - 1);
            }
        }
        multiMatrix(B1, y, x, 3, 3, 3);

        cout << "\nS (Eigenvector matrix): " << endl;
        outputMatrix(3, x);

        matrix D;
        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < 3; j++)
            {
                D[i][j] = (i == j) ? lambda[i] : 0.0;
            }
        }

        cout << "\nDiagonalized matrix D: " << endl;
        outputMatrix(3, D);

        matrix inverse;
        if (invertMatrix(x, inverse, 3))
        {
            cout << "\nS^-1 (Inverse Eigenvector matrix): " << endl;
            outputMatrix(3, inverse);

            // Verify the decomposition
            verifyEigendecomposition(tempA, x, D, inverse, lambda, 3);
```

```
        }
    }

    return 0;
}
void inputMatrix(int n, matrix A)
{
    cout << "Enter matrix elements:" << endl;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cin >> A[i][j];
        }
    }
}

void outputMatrix(int n, const matrix A)
{
    cout << fixed << setprecision(5);
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cout << setw(12) << A[i][j] ;
        }
        cout << endl;
    }
}

void multiMatrix(const matrix A, const matrix B, matrix C, int cola, int rowa, int
rowb)
{
    for (int i = 0; i < cola; i++)
    {
        for (int j = 0; j < rowb; j++)
        {
            C[i][j] = 0.0;
            for (int k = 0; k < rowa; k++)
            {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}

void Danhilepski(matrix A, matrix M, matrix M1, matrix B, int n)
{

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

```
        {
            B1[i][j] = (i == j) ? 1.0 : 0.0;
        }
    }

    for (int k = n - 2; k >= 0; k--)
    {

        if (std::fabs(A[k + 1][k]) < 1e-9)
        {

            cout << "Warning: Near-zero pivot in Danhilepski at k=" << k << endl;

            return;
        }

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

                M[i][j] = (i == j) ? 1.0 : 0.0;
                M1[i][j] = (i == j) ? 1.0 : 0.0;

                if (i == k)
                {
                    M1[i][j] = A[k + 1][j];

                    if (j == k)
                    {
                        M[i][j] = 1.0 / A[k + 1][k];
                    }
                    else
                    {
                        M[i][j] = -A[k + 1][j] / A[k + 1][k];
                    }
                }
            }
        }

        multiMatrix(A, M, B, n, n, n);
        multiMatrix(M1, B, A, n, n, n);

        multiMatrix(B1, M, B, n, n, n);
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
            {
                B1[i][j] = B[i][j];
            }
        }
```

```
    }
}

double determinant(const matrix A, int n)
{
    double det = 0.0;
    if (n == 1)
        return A[0][0];
    if (n == 2)
        return A[0][0] * A[1][1] - A[0][1] * A[1][0];

    matrix temp;
    for (int f = 0; f < n; f++)
    {
        int temp_i = 0;
        for (int i = 1; i < n; i++)
        {
            int temp_j = 0;
            for (int j = 0; j < n; j++)
            {
                if (j == f)
                    continue;
                temp[temp_i][temp_j] = A[i][j];
                temp_j++;
            }
            temp_i++;
        }
        det += (f % 2 == 0 ? 1.0 : -1.0) * A[0][f] * determinant(temp, n - 1);
    }
    return det;
}

bool invertMatrix(const matrix A, matrix inverse, int n)
{
    double det = determinant(A, n);
    if (std::fabs(det) < 1e-9)
    {
        cout << "Matrix is singular (det approx 0) and cannot be inverted." << endl;
        return false;
    }

    matrix adjoint;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            matrix temp;
            int temp_i = 0;

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

```
                continue;
            int temp_j = 0;
            for (int y = 0; y < n; y++)
            {
                if (y == j)
                    continue;
                temp[temp_i][temp_j] = A[x][y];
                temp_j++;
            }
            temp_i++;
        }

        double cofactor = pow(-1.0, i + j) * determinant(temp, n - 1);

        adjoint[j][i] = cofactor;
    }
}

for (int i = 0; i < n; i++)
{
    for (int j = 0; j < n; j++)
    {
        inverse[i][j] = adjoint[i][j] / det;
    }
}
return true;
}

void solution(double a, double b, double c, double d, double x[])
{

    if (std::fabs(a) < 1e-9)
    {

        cout << "Error: 'a' coefficient is near zero in cubic solver." << endl;
        x[0] = x[1] = x[2] = 0.0 / 0.0;
        return;
    }

    double p = (3.0 * a * c - b * b) / (3.0 * a * a);
    double q = (2.0 * b * b * b - 9.0 * a * b * c + 27.0 * a * a * d) / (27.0 * a * a
* a);

    double delta = (q * q / 4.0) + (p * p * p / 27.0);

    if (delta <= 1e-9)
    {

        double phi_arg = -q / (2.0 * sqrt(-p * p * p / 27.0));

        phi_arg = std::min(1.0, std::max(-1.0, phi_arg));
```

```
        double phi = acos(phi_arg);
        double R = 2.0 * sqrt(-p / 3.0);
        double b_over_3a = b / (3.0 * a);

        x[0] = R * cos(phi / 3.0) - b_over_3a;
        x[1] = R * cos((phi + 2.0 * PI) / 3.0) - b_over_3a;
        x[2] = R * cos((phi - 2.0 * PI) / 3.0) - b_over_3a;
    }

    else
    {
        double R = sqrt(delta);
        double S = pow(-q / 2.0 + R, 1.0 / 3.0);
        double T = pow(-q / 2.0 - R, 1.0 / 3.0);

        x[0] = S + T - b / (3.0 * a);
        x[1] = 0.0;
        x[2] = 0.0;
    }

}

void sortEigenvalues(double lambda[], int n)
{
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = 0; j < n - i - 1; j++)
        {
            if (lambda[j] < lambda[j + 1])
            {
                double temp = lambda[j];
                lambda[j] = lambda[j + 1];
                lambda[j + 1] = temp;
            }
        }
    }
}

void verifyEigendecomposition(const matrix A, const matrix S, const matrix D, const
matrix Sinv, const double lambda[], int n)
{
    matrix Temp, Reconstructed_A, Error;

    multiMatrix(S, D, Temp, n, n, n);
    multiMatrix(Temp, Sinv, Reconstructed_A, n, n, n);
    double max_error = 0.0;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            Error[i][j] = Reconstructed_A[i][j] - A[i][j];
            if (fabs(Error[i][j]) > max_error)
```

```
            {
                max_error = fabs(Error[i][j]);
            }
        }
    }

    cout << "\nVerification of Eigendecomposition (S * D * S^-1 - A):" << endl;
    cout << "Error matrix:" << endl;
    outputMatrix(n, Error);
    cout << "Maximum absolute error in reconstruction: " << max_error << endl;

    if (max_error < 1e-5)
    {
        cout << "Eigendecomposition verified successfully within tolerance." << endl;
    }
    else
    {
        cout << "Warning: Eigendecomposition verification failed. Error exceeds
tolerance." << endl;
    }
}
```

Trả lời: Dán kết quả thực thi vào bên dưới với $A = \begin{bmatrix} -10 & 15 & 16 \\ 15 & -11 & 19 \\ 16 & 19 & -13 \end{bmatrix}$ (sai số $\varepsilon = 10^{-5}$):

```
Enter n: 3
Enter matrix elements:
-10 15 16
15 -11 19
16 19 -13

Danhilepski's solution: Eigen values:
lambda[0] = 22.0141
lambda[1] = -24.7981
lambda[2] = -31.2159

S (Eigenvector matrix):
     0.97754    -3.08590    -0.18860
     1.01966     1.97770    -0.79991
     1.00000     1.00000     1.00000

Diagonalized matrix D:
    22.01405     0.00000     0.00000
     0.00000   -24.79811     0.00000
     0.00000     0.00000   -31.21595

S^-1 (Inverse Eigenvector matrix):
     0.32636     0.34042     0.33386
    -0.21379     0.13702     0.06928
    -0.11257    -0.47744     0.59686

Verification of Eigendecomposition (S * D * S^-1 - A):
Error matrix:
    -0.00000     0.00000    -0.00000
     0.00000    -0.00000     0.00000
     0.00000     0.00000     0.00000
Maximum absolute error in reconstruction: 0.00000
Eigendecomposition verified successfully within tolerance.
```

Câu 4 (*3 điểm*): Cho ma trận A. Viết chương trình bằng c/c++ có sử dụng hàm thực hiện phân rã ma trận A bằng phương pháp SVD.

Trả lời: Dán code vào bên dưới (bao gồm điều kiện của ma trận A nếu có):

```cpp
#include <iostream>
#include <Eigen/Dense>
#include <iomanip>
#include <cmath>
#define eps 1e-5
using namespace std;
using namespace Eigen;

void input(double A[][10], int row, int col);
void swap(double &a, double &b);
void display(double A[][10], int row, int col);
```

```
void transpose(double A[][10], double At[][10], int rows, int cols);
void NhanMaTran(MatrixXd &S, double A[][10], double B[][10], int row1, int col1, int
col2);
void GetEigenValuesAndVector(MatrixXd S, MatrixXd &lambda, MatrixXd &vector);
void tinhMatranS(MatrixXd lambda, double sigma[][10], int rows, int cols);
void tinhMatranU(MatrixXd lambda, MatrixXd vector, double U[][10], double A[][10], int
rows, int cols);
void tinhMatranV(MatrixXd vector, double V[][10]);
void gramSchmidt(double U[][10], int rows, int cols);

int main()
{
    int rows, cols;
    double A[10][10], At[10][10];
    cout << "Nhap ma tran " << endl;
    cout << "Hang: ";
    cin >> rows;
    cout << "Cot: ";
    cin >> cols;
    cout << "Nhap phan tu ma tran: " << endl;
    input(A, rows, cols);
    cout << "Ma tran goc" << endl;
    display(A, rows, cols);
    MatrixXd S(cols, cols), lambda(cols, 1), vector(cols, cols);
    transpose(A, At, rows, cols);
    NhanMaTran(S, At, A, cols, rows, cols);
    GetEigenValuesAndVector(S, lambda, vector);
    cout << "Phan ra theo U*sigma*V^T" << endl;
    double sigma[10][10], U[10][10], V[10][10];
    tinhMatranU(lambda, vector, U, A, rows, cols);
    cout << "Ma tran U:" << endl;
    display(U, rows, rows);
    tinhMatranS(lambda, sigma, rows, cols);
    cout << "Ma tran sigma:" << endl;
    display(sigma, rows, cols);
    tinhMatranV(vector, V);
    cout << "Ma tran V^T:" << endl;
    double VT[10][10];
    transpose(V, VT, cols, cols);
    display(VT, cols, cols);
    // Thử lại U * sigma * V^T - A
    MatrixXd Umat(rows, rows), Sig(rows, cols), VTrans(cols, cols);

   cout << "Kiem tra U * Sigma * V^T - A:" << endl;
   for (int i = 0; i < rows; i++)
        for (int j = 0; j < rows; j++)
            Umat(i, j) = U[i][j];

    for (int i = 0; i < rows; i++)
        for (int j = 0; j < cols; j++)
            Sig(i, j) = sigma[i][j];
```

```
    for (int i = 0; i < cols; i++)
        for (int j = 0; j < cols; j++)
            VTrans(i, j) = VT[i][j];

    MatrixXd Acheck(rows, cols);
    for (int i = 0; i < rows; i++)
        for (int j = 0; j < cols; j++)
            Acheck(i, j) = A[i][j];

    MatrixXd diff = Umat * Sig * VTrans - Acheck;

    cout << diff << endl;

    return 0;
}

void input(double A[][10], int row, int col)
{
    for (int i = 0; i < row; i++)
        for (int j = 0; j < col; j++)
        {
            cin >> A[i][j];
        }
}

void display(double A[][10], int row, int col)
{
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
            cout << setw(9) << fixed << setprecision(5) << A[i][j];
        cout << endl;
    }
}

void swap(double &a, double &b)
{
    double temp = a;
    a = b;
    b = temp;
}

void transpose(double A[][10], double At[][10], int rows, int cols)
{
    for (int i = 0; i < rows; i++)
        for (int j = 0; j < cols; j++)
        {
            At[j][i] = A[i][j];
        }
}
```

```
void NhanMaTran(MatrixXd &S, double A[][10], double B[][10], int row1, int col1, int
col2)
{
    for (int i = 0; i < row1; i++)
    {
        for (int j = 0; j < col2; j++)
        {
            S(i, j) = 0;
            for (int k = 0; k < col1; k++)
            {
                S(i, j) = S(i, j) + A[i][k] * B[k][j];
            }
        }
    }
}

void GetEigenValuesAndVector(MatrixXd S, MatrixXd &lambda, MatrixXd &vector)
{
    SelfAdjointEigenSolver<Eigen::MatrixXd> eigensolver(S);
    vector = eigensolver.eigenvectors();
    lambda = eigensolver.eigenvalues();
    int k = lambda.rows();
    int l = vector.rows();
    for (int i = 0; i < k; i++)
    {
        if (lambda(i, 0) < eps)
            lambda(i, 0) = 0;
    }
    for (int i = 0; i < k; i++)
        for (int j = i + 1; j < k; j++)
        {
            if (lambda(j, 0) > lambda(i, 0))
            {
                swap(lambda(j, 0), lambda(i, 0));
                for (int h = 0; h < l; h++)
                    swap(vector(h, i), vector(h, j));
            }
        }
}

void tinhMatranU(MatrixXd lambda, MatrixXd vector, double U[][10], double A[][10], int
rows, int cols)
{
    MatrixXd ui(rows, 1);
    double Vi[cols][10];
    for (int i = 0; i < cols; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            Vi[j][0] = vector(j, i);
        }
        NhanMaTran(ui, A, Vi, rows, cols, 1);
```

```
        for (int k = 0; k < rows; k++)
        {
            if (lambda(i, 0) != 0)
            {
                U[k][i] = (1 / sqrt(lambda(i, 0))) * ui(k, 0);
            }
            else
            {
                U[k][i] = 0;
            }
        }
    }
    if (rows > cols)
    {
        if (rows > cols)
        {
            MatrixXd U_matrix(rows, cols);
            for (int i = 0; i < rows; i++)
            {
                for (int j = 0; j < cols; j++)
                {
                    U_matrix(i, j) = U[i][j];
                }
            }
            MatrixXd orthogonal_basis = U_matrix.householderQr().householderQ();
            for (int j = cols; j < rows; j++)
            {
                for (int i = 0; i < rows; i++)
                {
                    U[i][j] = orthogonal_basis(i, j);
                }
            }
        }
    }
    else
    {
        gramSchmidt(U, rows, cols);
    }
}

void gramSchmidt(double U[][10], int rows, int cols)
{
    for (int i = 0; i < cols; i++)
    {
        for (int j = 0; j < i; j++)
        {
            double dot_product = 0;
            for (int k = 0; k < rows; k++)
            {
                dot_product += U[k][i] * U[k][j];
            }
            for (int k = 0; k < rows; k++)
```

```
            {
                U[k][i] -= dot_product * U[k][j];
            }
        }
        double norm = 0;
        for (int k = 0; k < rows; k++)
        {
            norm += U[k][i] * U[k][i];
        }
        norm = sqrt(norm);
        if (norm > 1e-10)
        {
            for (int k = 0; k < rows; k++)
            {
                U[k][i] /= norm;
            }
        }
        else
        {
            MatrixXd random_vector = MatrixXd::Random(rows, 1);
            for (int j = 0; j < i; j++)
            {
                double dot_product = 0;
                for (int k = 0; k < rows; k++)
                {
                    dot_product += random_vector(k, 0) * U[k][j];
                }
                for (int k = 0; k < rows; k++)
                {
                    random_vector(k, 0) -= dot_product * U[k][j];
                }
            }
            double random_norm = random_vector.norm();
            for (int k = 0; k < rows; k++)
            {
                U[k][i] = random_vector(k, 0) / random_norm;
            }
        }
    }
}

void tinhMatranV(MatrixXd vector, double V[][10])
{
    int row = vector.rows(), col = vector.cols();
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            V[i][j] = vector(i, j);
        }
    }
}
```

```
void tinhMatranS(MatrixXd lambda, double sigma[][10], int rows, int cols)
{
    int k = 0;
    for (int i = 0; i < rows; i++)
        for (int j = 0; j < cols; j++)
        {
            sigma[i][j] = (i != j) ? 0 : sqrt(lambda(k, 0));
            if (i == j)
                k++;
        }
}
```

Trả lời: Dán kết quả thực thi vào bên dưới với $A = \begin{bmatrix} 5 & 7 & 9 \\ 3 & 6 & 9 \\ 2 & 4 & 6 \end{bmatrix}$ (sai số $\varepsilon = 10^{-5}$):

```
Nhap ma tran
Hang: 3
Cot: 3
Nhap phan tu ma tran:
5 7 9
3 6 9
2 4 6
Ma tran goc
  5.00000  7.00000  9.00000
  3.00000  6.00000  9.00000
  2.00000  4.00000  6.00000
Phan ra theo U*sigma*V^T
Ma tran U:
  0.67782 -0.73523  0.00000
  0.61175  0.56398  0.55470
  0.40783  0.37599 -0.83205
Ma tran sigma:
 18.30038  0.00000  0.00000
  0.00000  1.44780  0.00000
  0.00000  0.00000  0.00000
Ma tran V^T:
  0.33005  0.54898  0.76791
 -0.85112 -0.17876  0.49360
 -0.40825  0.81650 -0.40825
Kiem tra U * Sigma * V^T - A:
-0.00000 -0.00000 -0.00000
 0.00000 -0.00000 -0.00000
 0.00000 -0.00000 -0.00000
```