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: **01** Thời gian làm bài: 90 phút *(không kể thời gian chép/phát đề)*
Được sử dụng tài liệu khi làm bài.

Họ tên: Ngô Văn Lộc **Lớp**: 23T_DT1 **MSSV**:102230197

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*): Cho số nguyên dương N (N>1). Viết chương trình bằng C/C++ có sử dụng hàm thực hiện:

- Tìm số hoàn hảo M gần N nhất (M<N), biết rằng N=3000
- Tìm các số nguyên tố bé hơn M, liệt kê và tính tổng của chúng.

 # Trả lời: Dán code vào bên dưới:

```
      #include <iostream>

#include <math.h>
#include <vector>
#include <cstring>
#include <string>

using namespace std;

bool isPerfect(int n) {
    int sum = 0;
    for(int i = 1; i < n; i++) {
        if(n % i == 0) {
            sum += i;
        }
    }
    if(n == sum) return true;
    else return false;
}

int nearPerfect(int n) {
    for(int i = n; i >= 0; i--) {
        if(isPerfect(i)) return i;
    }
    return 0;
}

long long SieveOfEratosthenes(int n) {
    vector<bool> primes(n+1, true);
    for(int i = 2; i <= sqrt(n); i++) {
```

```
        if(!primes[i]) continue;
        for(int j = i*i; j <= n; j += i) {
            primes[j] = false;
        }
    }
    int sum = 0;
    for(int i = 2; i < primes.size(); i++) {
        if(primes[i])  {
            cout << i << " ";
            sum += i;
        }
    }
    return sum;
}

int main() {
    int N = 3000;
    int M =  nearPerfect(N);
    cout << "So hoan hao gan voi 3000 nhat la: " << M << endl;
    cout << endl;
    cout << "Cac so nguyen to be hon " << M << " la:\n";
    int sum = SieveOfEratosthenes(M);
    cout << "\nTong cac so nguyen to la: " << sum << endl;
    return 0;
}
```

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

So hoan hao gan voi 3000 nhat la: 496

Cac so nguyen to be hon 496 la:

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197 199 211 223 227 229 233 239 241 251 257 263 269 271 277 281 283 293 307 311 313 317 331 337 347 349 353 359 367 373 379 383 389 397 401 409 419 421 431 433 439 443 449 457 461 463 467 479 487 491

Tong cac so nguyen to la: 21037

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

$$\begin{cases} x \equiv 1 \bmod 3 \\ x \equiv 3 \bmod 5 \\ x \equiv 5 \bmod 7 \\ x \equiv 7 \bmod 11 \\ x \equiv 11 \bmod 17 \end{cases}$$

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

Trả lời: Dán code vào bên dưới:

```cpp
#include <iostream>
#include <math.h>
#include <vector>

using namespace std;

int x, y; // Hệ số của phương trình ax + by = gcd(a, b)
/*
    Với hệ phương trình ax + my = 1
    Lấy modun của m hay vế ta được:
        ax % m = 1
        => Khi đó x chính là nghịch đảo modun của a
*/
int gcd; // Ước chung của a và b

// Phương pháp Euclid để tìm ước chung của 2 số đồng thời tìm hệ số x và y
void extendedEuclid(int a, int b) {
    if(b == 0) {
        x = 1; y = 0; gcd = a;
    }
    else {
        extendedEuclid(b, a % b);
        int temp = x;
        x = y;
        y = temp - (a / b) * y;
    }
}

// Phương pháp tìm nghịch đảo modulo của a mod m
int modunlarInverse(int a, int m) {
    extendedEuclid(a, m); // Tìm hệ số x
    return (x + m) % m; // Khi x < 0 thì trả về x + m
}


// Các phần tử trong vector M là đôi một nguyên tố cùng nhau
void input(int n, vector<int> &A, vector<int> &M) {
    cout << "Nhap cac phuong trinh (a mod m): \n";
    for(int i = 0; i < n; i++) {
        cout << "Nhap phuong trinh so " << i + 1 << ": ";
        int a, m; cin >> a >> m;
        A.push_back(a);
        M.push_back(m);
    }
}

long long ChineseRemainderTheorem(int n, vector<int> A, vector<int> M) {
    long long result = 0;
    long long prod = 1; // Tích các m1*m2*...*mn
    for(int i = 0; i < n; i++) {
```

```
        prod *= M[i];
    }
    for(int i = 0; i < n; i++) {
        long long temp = prod / M[i];
        result = (result + A[i] * temp * modunlarInverse(temp, M[i])) % prod;
    }
    return result;
}

int main() {
    int n;
    cout << "Nhap so phuong trinh: "; cin >> n;
    vector<int> A, M;
    input(n, A, M);
    long long prod = 1; // Tích các m1*m2...*mn
    for(int x : M) prod *= x;
    long long result = ChineseRemainderTheorem(n, A, M);
    cout << "Ket qua: x = " << result << " + k" << prod << endl;
    return 0;
}
```

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

```
Nhap so phuong trinh: 5
Nhap cac phuong trinh (a mod m):
Nhap phuong trinh so 1: 1 3
Nhap phuong trinh so 2: 3 5
Nhap phuong trinh so 3: 5 7
Nhap phuong trinh so 4: 7 11
Nhap phuong trinh so 5: 11 17


Ket qua: x = 9973 + k19635
```

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 (có hàm kiểm tra điều kiện phân rã).

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 của ma trận A:

- Ma trận phải là ma trận vuông

- Ma trận đối xứng

```
#include<iostream>
#include<math.h>
#include<cstring>
#include <iomanip>

using namespace std;


void inputMatrix(double A[][10], int n, int m);
void outputResult(double A[][10], double D[], int n);
bool checkMatrixCholesky(double A[][10], int n, int m);
bool SymmetricalMatrix(double A[][10], int n);
void CholeskyLDL_Decomposition(double A[][10], double L[][10], double D[], int n);

int main(){
    double A[10][10];
    double L[10][10];
    double D[10];
    int n, m;
    cout << "Nhap kich thuoc ma tran: "; cin >> n >> m;
    cout << "Nhap ma tran:\n";
    inputMatrix(A, n, m);
    if(!checkMatrixCholesky(A, n, m)){
        cout << "Ma tran khong the phan ra\n";
        return 0;
    } else {
        checkMatrixCholesky(A, n, m);
        CholeskyLDL_Decomposition(A, L, D, n);
        outputResult(L, D, n);
    }
    return 0;
}

void inputMatrix(double A[][10], int n, int m){
    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            cin >> A[i][j];
        }
    }
}

void outputResult(double A[][10], double D[], int n) {
    cout << "Matrix L : \n";
    cout << fixed << setprecision(3);
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            cout << A[i][j] << " ";
        }
        cout << "\n";
    }
```

```
    cout << "\nMatrix D : \n";
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            if(i == j)
                cout << D[i] << " ";
            else
                cout << "0.00" << " ";
        }
        cout << "\n";
    }
    cout << "\nMatrix L*: \n";
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            cout << A[j][i] << " ";
        }
        cout << "\n";
    }
}

bool checkMatrixCholesky(double A[][10], int n, int m)
{
    if(n != m){
        return false;
    }
    // if(!SymmetricalMatrix(A, n)){
    //     return false;
    // }
    return true;
}

bool SymmetricalMatrix(double A[][10], int n)
{
    for(int i = 0; i < n; i++){
        for(int j = 0; j < i; j++){
            if(A[i][j]!= A[j][i]){
                return false;
            }
        }
    }
    return true;
}

void CholeskyLDL_Decomposition(double A[][10], double L[][10], double D[], int n){
    memset(L, 0, sizeof(double) * 10 * 10);
    for(int j = 0; j < n; j++) {
        for (int k = 0; k < j; k++) {
            A[j][j] -= L[j][k] * L[j][k] * D[k];
        }
        D[j] = A[j][j];
        for(int i = j + 1; i < n; i++) {
            L[i][j] = A[i][j];
            for(int k = 0; k < j; k++) {
```

```
                L[i][j] -= L[i][k] * L[j][k] * D[k];
            }
            L[i][j] /= D[j];
        }
        L[j][j] = 1;
    }
}
```

Trả lời: Dán kết quả thực thi vào bên dưới với $A = \begin{bmatrix} 1 & 10 & 11 \\ 10 & 2 & 5 \\ 11 & 5 & 3 \end{bmatrix}$ (sai số $\varepsilon = 10^{-5}$):

Nhap kich thuoc ma tran: 3 3

Nhap ma tran:

1 10 11

10 2 5

11 5 3

Matrix L :

1.000 0.000 0.000

10.000 1.000 0.000

11.000 1.071 1.000

Matrix D :

1.000 0.00 0.00

0.00 -98.000 0.00

0.00 0.00 -5.500

Matrix L*:

1.000 10.000 11.000

0.000 1.000 1.071

0.000 0.000 1.000

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 phải là ma trận vuông
- Ma trận có đủ giá trị riêng và vector riêng
- Ma trận phải khả quy

```
#include <iostream>
#include <iomanip>
#include <math.h>
#include <cmath>
#include <vector>
#include <algorithm>

using namespace std;

const float pi = 3.1415926535898;

#define MAX_SIZE 4

typedef float Matrix[MAX_SIZE][MAX_SIZE];

Matrix A; // Ma trận ban đầu

// Vector lưu các giá trị riêng
vector<float> lamda;

Matrix P, P1; // P1 là ma trận nghịch đảo của P

Matrix D; // Ma trận chéo hóa


void inputMatrix(Matrix &A, int n) {
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            cin >> A[i][j];
        }
    }
}

void outputMatrix(Matrix A, int n) {
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            cout << fixed << setprecision(3) << A[i][j] << "\t";
        }
        cout << endl;
    }
}
```

```cpp
// C = A x B
void mulMatrix(const Matrix &a, const Matrix &b, Matrix &c, int n) {
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            c[i][j] = 0;
            for(int k = 0; k < n; k++) {
                c[i][j] += a[i][k] * b[k][j];
            }
        }
    }
}

void copyMatrix(Matrix &a, const Matrix &b, int n) {
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            a[i][j] = b[i][j];
        }
    }
}

bool compareMatrix(const Matrix &a, const Matrix &b, int n) {
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            if(fabs(a[i][j] - b[i][j]) > 0.001) {
                return false;
            }
        }
    }
    return true;
}

bool inverseMatrix(const Matrix &A, Matrix &inverse, int n) {
    Matrix temp;
    copyMatrix(temp, A, n);
    // Khởi tạo ma trận nghịch đảo là ma trận đơn vị
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
            inverse[i][j] = (i == j) ? 1.0 : 0.0;
    }

    for (int i = 0; i < n; i++)
    {
        if (temp[i][i] == 0)
        {
            // Tìm hàng khác có phần tử ở cột i khác 0 để đổi hàng
            int check = 0;
            for (int j = i + 1; j < n; j++)
            {
                if (temp[j][i] != 0)
                {
                    for (int k = 0; k < n; k++)
```

```
                    {
                        swap(temp[i][k], temp[j][k]);
                        swap(inverse[i][k], inverse[j][k]);
                    }
                    check++;
                    break;
                }
            }
            if (check == 0)
                return false;  // Ma trận không khả nghịch
        }

        // Chia hàng hiện tại cho phần tử chéo chính temp[i][i] để đưa về 1
        double diag = temp[i][i];
        for (int j = 0; j < n; j++)
        {
            temp[i][j] /= diag;
            inverse[i][j] /= diag;
        }

        // Khử các phần tử khác ở cột i
        for (int j = 0; j < n; j++)
        {
            if (j != i)
            {
                double factor = temp[j][i];
                for (int k = 0; k < n; k++)
                {
                    temp[j][k] -= factor * temp[i][k];
                    inverse[j][k] -= factor * inverse[i][k];
                }
            }
        }
    }
    return true;  // Ma trận nghịch đảo thành công
}

// giải phương trình bật 3 tổng quát
vector<float> Trigonometrical(vector<float> A, int n) {
    // Các hệ số của phương trình
    float a = A[0];
    float b = A[1];
    float c = A[2];
    float d = A[3];
    float delta = b*b - 3*a*c;
    float k = (9*a*b*c - 2*pow(b, 3) - 27*a*a*d) / (2*sqrt(abs(pow(delta, 3))));
/*
    Nếu delta > 0
        Nếu |k| <= 1: phương trình có 3 nghiệm
        Nếu |k| > 1: phương trình có một nghiệm duy nhất
    Nếu delta = 0: Phương trình có một nghiệm bội
    Nếu delta < 0: Phương trình có một nghiệm duy nhất
```

```
    Trong bài toán này chỉ đề cập tới trường hợp có 3 nghiệm
*/
    vector<float> Solution;
    Solution.push_back((2*sqrt(delta) * cos(acos(k)/3) - b) / (3 * a));
    Solution.push_back((2*sqrt(delta) * cos(acos(k)/3 - 2*pi/3) - b) / (3 * a));
    Solution.push_back((2*sqrt(delta) * cos(acos(k)/3 + 2*pi/3) - b) / (3 * a));
    return Solution;
}

void Danhilepxki(const Matrix &A, int n) {
    Matrix tmpA; // Ma trận copy từ ma trận A, biến đổi thành ma trận đồng dạng với ma
trận A
    copyMatrix(tmpA, A, n);
    Matrix B; // B = A x M
    Matrix M1, M; // M1 là ma trận nghịch đảo của M
    Matrix C; // Tích các ma trận M biến đổi từ lần thứ 1->n-1
    Matrix tmp;
    for(int k = n - 2; k >= 0; k--) {
        // Tính ma trận M và M1 (M1 là ma trận nghịch đảo của M)
        for(int i = 0; i < n; i++) {
            for(int j = 0; j < n; j++) {
                if(i != k) {
                    if(i == j) {
                        M[i][j] = 1; M1[i][j] = 1;
                    } else {
                        M[i][j] = 0; M1[i][j] = 0;
                    }
                } else {
                    M1[i][j] = tmpA[k + 1][j];
                    if(j == k) {
                        M[i][j] = 1/tmpA[k + 1][k];
                    } else {
                        M[i][j] = -tmpA[k + 1][j] / tmpA[k + 1][k];
                    }
                }
            }
        }
        mulMatrix(tmpA, M, B, n);  // B = A x M
        mulMatrix(M1, B, tmpA, n); // A = M1 x B <=> tmpA = M1 x A x M

        // Tính ma trận C
        if(k == n - 2) {
            copyMatrix(C, M, n);
        } else {
            mulMatrix(C, M, tmp, n);
            copyMatrix(C, tmp, n); // C = C x M
        }
    }

    // Vector lưu phương trình với nghiệm là các giá trị riêng của ma trận A
    vector<float> equation;
    equation.push_back(1);
```

```
    for(int i = 0; i < n; i++)
        equation.push_back(-tmpA[0][i]);

    // Vector lưu các giá trị riêng
    lamda = Trigonometrical(equation, n);
    sort(lamda.begin(), lamda.end());

    // Các vector riêng của ma trận C lưu trên các cột của ma trận tmp
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            tmp[j][i] = pow(lamda[i], n - j - 1);
        }
    }

    // Các vector riêng của ma trận A được tính bằng cách P = C x P
    mulMatrix(C, tmp, P, n);
}

// Chéo hóa ma trận
bool Diagonalization(int n) {
    // Tính các giá trị riêng vector riêng của ma trận A
    Danhilepxki(A, n);

    // Tính ma trận chéo hóa D
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            if(i == j) {
                D[i][j] = lamda[i];
            } else {
                D[i][j] = 0;
            }
        }
    }

    // Tính ma trận nghịch đảo P^-1
    inverseMatrix(P, P1, n);

    // Kiểm tra điều kiện A = P x D x P^-1
    Matrix tmp, B;
    mulMatrix(P, D, tmp, n);
    mulMatrix(tmp, P1, B, n);

    return compareMatrix(A, B, n);

}

int main() {
    int n;
    cout << "Nhap kich thuoc ma tran (nxn, n <=4): ";
    cin >> n;
    inputMatrix(A, n);
```

```
    if(Diagonalization(n)) {
        cout << "Ma tran cheo hoa thanh cong\n";
        cout << "Ma tran D:\n";
        outputMatrix(D, n);
        cout << "Ma tran P:\n";
        outputMatrix(P, n);
        cout << "Ma tran nghich dao P^-1:\n";
        outputMatrix(P1, n);
    }
    else cout << "Ma tran cheo hoa khong thanh cong\n";
    Diagonalization(n);
}
```

 # Trả lời: Dán kết quả thực thi vào bên dưới với $A = \begin{bmatrix} 1 & 10 & 11 \\ 10 & 2 & 5 \\ 11 & 5 & 3 \end{bmatrix}$ (sai số $\varepsilon = 10^{-5}$):

Nhap kich thuoc ma tran (nxn, n <=4): 3

1 10 11

10 2 5

11 5 3

Ma tran cheo hoa thanh cong

Ma tran D:

-10.957 0.000 0.000

0.000 -2.525 0.000

0.000 0.000 19.482

Ma tran P:

-1.684 -0.003 1.086

0.914 -1.099 0.907

1.000 1.000 1.000

Ma tran nghich dao P^-1:

-0.360 0.196 0.214

-0.001 -0.498 0.453

0.362 0.302 0.333

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):

```
#include <iostream>
#include <Eigen/Dense>
#include <iomanip>
#include <cmath>

using namespace std;
using namespace Eigen;

// g++ -I C:/Users/Loc/eigen "SVD.cpp" -o "SVD"

// Function declarations
void inputMatrix(double matrix[][10], int &rows, int &cols);
void displayMatrix(double matrix[][10], int row, int col);
void transposeMatrix(double source[][10], double destination[][10], int rows, int
cols);
void matrixMultiplication(MatrixXd &result, double A[][10], double B[][10], int rowA,
int colA, int colB);
void computeEigen(MatrixXd S, MatrixXd &eigenValues, MatrixXd &eigenVectors);
void computetSigma(MatrixXd eigenValues, double sigma[][10], int rows, int cols);
void computeU(MatrixXd eigenValues, MatrixXd eigenVectors, double U[][10], double
A[][10], int rows, int cols);
void gramSchmidt(double U[][10], int rows, int cols);
void computetV(MatrixXd eigenVectors, double V[][10]);
// void orthogonalize(double U[][10], int rows, int cols);
void computeSVD(double A[][10], int rows, int cols);
void verifySVD(double A[][10], double U[][10], double sigma[][10], double Vt[][10],
int rows, int cols);

int main() {
    double A[10][10];
    int rows, cols;

    inputMatrix(A, rows, cols);

    computeSVD(A, rows, cols);

    return 0;
}

void inputMatrix(double matrix[][10], int &rows, int &cols) {
    cout << "Nhap kich thuoc cua ma tran:" << endl;
    cout << "So hang: "; cin >> rows;
    cout << "So cot: "; cin >> cols;

    cout << "Nhap gia tri cho ma tran A: " << endl;
    for (int i = 0; i < rows; i++) {
```

```
        for (int j = 0; j < cols; j++) {
            cout << "A[" << i+1 << "][" << j+1 << "] = ";
            cin >> matrix[i][j];
        }
    }
}

void displayMatrix(double matrix[][10], int row, int col) {
    for (int i = 0; i < row; i++) {
        for (int j = 0; j < col; j++) {
            cout << setw(9) << setprecision(4) << matrix[i][j];
        }
        cout << endl;
    }
}

void transposeMatrix(double source[][10], double destination[][10], int rows, int
cols) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            destination[j][i] = source[i][j];
        }
    }
}

// Matrix multiplication
void matrixMultiplication(MatrixXd &result, double A[][10], double B[][10], int rowA,
int colA, int colB) {
    for (int i = 0; i < rowA; i++) {
        for (int j = 0; j < colB; j++) {
            result(i, j) = 0;
            for (int k = 0; k < colA; k++) {
                result(i, j) += A[i][k] * B[k][j];
            }
        }
    }
}
void computeEigen(MatrixXd S, MatrixXd &eigenValues, MatrixXd &eigenVectors) {
    SelfAdjointEigenSolver<MatrixXd> eigensolver(S);
    eigenVectors = eigensolver.eigenvectors();
    eigenValues = eigensolver.eigenvalues();

    for (int i = 0; i < eigenValues.rows(); i++) {
        if (eigenValues(i, 0) < 1e-6) eigenValues(i, 0) = 0;
    }
    for (int i = 0; i < eigenValues.rows(); i++) {
        for (int j = i + 1; j < eigenValues.rows(); j++) {
            if (eigenValues(j, 0) > eigenValues(i, 0)) {
                swap(eigenValues(j, 0), eigenValues(i, 0));
                for (int h = 0; h < eigenVectors.rows(); h++) {
                    swap(eigenVectors(h, i), eigenVectors(h, j));
                }
```

```
            }
        }
    }
}

void computetSigma(MatrixXd eigenValues, 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 && k < eigenValues.rows()) ? sqrt(eigenValues(k++,
0)) : 0;
        }
    }
}

void computeU(MatrixXd eigenValues, MatrixXd eigenVectors, double U[][10], double
A[][10], int rows, int cols) {
    MatrixXd ui(rows, 1);
    double Vi[10][10];

    for (int i = 0; i < cols; i++) {
        for (int j = 0; j < cols; j++) {
            Vi[j][0] = eigenVectors(j, i);
        }
        matrixMultiplication(ui, A, Vi, rows, cols, 1);

        for (int k = 0; k < rows; k++) {
            U[k][i] = (eigenValues(i, 0) != 0) ? ui(k, 0) / sqrt(eigenValues(i, 0)) :
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 orthogonalize(double U[][10], int rows, int cols) {
//     for (int i = cols; i < rows; i++) {
//         for (int j = 0; j < rows; j++) {
//             U[j][i] = (i == j) ? 1 : 0;
//         }
//     }
// }

void computetV(MatrixXd eigenVectors, double V[][10]) {
    for (int i = 0; i < eigenVectors.rows(); i++) {
        for (int j = 0; j < eigenVectors.cols(); j++) {
            V[i][j] = eigenVectors(i, j);
```

```
        }
    }
}

void computeSVD(double A[][10], int rows, int cols) {
    double At[10][10];
    MatrixXd S(cols, cols), eigenValues(cols, 1), eigenVectors(cols, cols);
    double sigma[10][10], U[10][10], V[10][10];

    transposeMatrix(A, At, rows, cols);
    matrixMultiplication(S, At, A, cols, rows, cols);

    computeEigen(S, eigenValues, eigenVectors);

    computeU(eigenValues, eigenVectors, U, A, rows, cols);
    computetSigma(eigenValues, sigma, rows, cols);
    computetV(eigenVectors, V);

    cout << "Matrix U:" << endl;
    displayMatrix(U, rows, rows);

    cout << "Matrix Sigma:" << endl;
    displayMatrix(sigma, rows, cols);

    cout << "Matrix V Transposed:" << endl;
    double Vt[10][10];
    transposeMatrix(V, Vt, cols, cols);
    displayMatrix(Vt, cols, cols);

    verifySVD(A, U, sigma, Vt, rows, cols);
}

void verifySVD(double A[][10], double U[][10], double sigma[][10], double Vt[][10],
int rows, int cols) {
    MatrixXd Usigma(rows, cols);
    MatrixXd A_computed(rows, cols);
    MatrixXd Vt_mat(cols, cols);

    matrixMultiplication(Usigma, U, sigma, rows, rows, cols);

    for (int i = 0; i < cols; i++) {
        for (int j = 0; j < cols; j++) {
            Vt_mat(i, j) = Vt[i][j];
        }
    }

    A_computed = Usigma * Vt_mat;

    // cout << "Reconstructed matrix A from SVD:" << endl;
    // for (int i = 0; i < rows; i++) {
    //     for (int j = 0; j < cols; j++) {
    //         cout << setw(9) << setprecision(4) << A_computed(i, j);
```

```
    //     }
    //     cout << endl;
    // }

    bool isClose = true;
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            if (fabs(A[i][j] - A_computed(i, j)) > 1e-5) {
                isClose = false;
                break;
            }
        }
    }

    if (isClose) {
        cout << "Phan tich SVD chinh xac. " << endl;
    } else {
        cout << "Phan tich SVD khong chinh xac." << endl;
    }
}
```

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

Matrix U:

0.3841 -0.8462 -0.3692

-0.7682 -0.0711 -0.6362

0.5121 0.528 -0.6774

Matrix Sigma:

13.53 0 0

0 0 0

0 0 0

Matrix V Transposed:

0.5774 0.5774 0.5774

0 -0.7071 0.7071

-0.8165 0.4082 0.4082

Phan tich SVD chinh xac.