Câu 1:

#include<iostream>

#include<iomanip>

#include<Eigen/Dense>

#include<Eigen/EigenValues>

using namespace std;

using namespace Eigen;

float data[1000];

int rows,cols;

struct matran{

int sd,sc;

float coef[100][100];

};

struct matran createMatrix(int m,int n)

{

struct matran A;

A.sd=m;

A.sc=n;

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

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

A.coef[i][j]=0;

}

return A;

}

int input()

{

cout << "Matrix A:"<<endl;

for(int i=0;i<rows\*cols;i++){

cin>>data[i];

}

}

int outputMatrix()

{

cout << "Matrix A:"<<endl;

for(int i=0;i<rows\*cols;i++){

if (i % cols == 0)

cout << "\n";

cout << data[i] << setw(5);

}

}

void SVD()

{

MatrixXf matrix(rows,cols);

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

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

matrix(i,j)= data[i\*cols+j];

}

}

struct matran sig= createMatrix(rows,cols);

MatrixXf sigma(rows,cols);

MatrixXf matrixA= matrix.transpose()\*matrix;

SelfAdjointEigenSolver<MatrixXf> esA(matrixA);

int k=0;

VectorXf temp = esA.eigenvalues();

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

{

if(fabs(temp[i]<10e-4)) temp[i]=0;

temp[i]=sqrt(temp[i]);

sig.coef[temp.size()-i-1][temp.size()-i-1]=temp[i];

}

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

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

sigma(i,j)=sig.coef[i][j];

if(fabs(sigma(i,j))<10e-4)

sigma(i,j)=0;

}

}

MatrixXf V= esA.eigenvectors().transpose();

struct matran Vt = createMatrix(cols,cols);

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

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

Vt.coef[cols-i-1][j]=V(i,j);

}

}

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

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

V(i,j)=Vt.coef[i][j];

if(fabs(V(i,j))<10e-4)

V(i,j)=0;

}

}

MatrixXf matrixB = matrix\*matrix.transpose();

SelfAdjointEigenSolver<MatrixXf> esB(matrixB);

MatrixXf U = esB.eigenvectors();

for(int i=0;i<U.cols()/2;i++){

temp =U.col(U.cols()-1-i);

U.col(U.cols()-i-1)=U.col(i);

U.col(i)=temp.col(0);

}

for(int i=0;i<min(sigma.rows(),sigma.cols());i++){

if(sigma(i,i)!=0){

temp=V.row(i);

temp=temp.transpose();

temp=matrix\*temp/sigma(i,i);

U.col(i)=temp.col(0);

}

}

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

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

if(fabs(U(i,j))<10e-4)

U(i,j)=0;

}

}

cout.precision(4);

cout <<"\nSingular value decomposition:\n";

cout <<"\nMatrix S:\n"<<matrixA<<endl;

cout <<"\nMatrix U:\n"<<U<<endl;

cout <<"\nMatrix Sigma:\n"<<sigma<<endl;

cout <<"\nMatrix V^T\n"<<V<<endl;

cout <<"\nRetry: A = U \* Sigma \* V^T:\n";

MatrixXf matrixC;

if(V.cols()<U.cols()){

MatrixXf Vn(rows,cols);

for(int i=0;i<V.cols();i++){

for(int j=0;j<U.cols();j++){

if(j==U.cols()-1)

Vn(i,j)=0;

else Vn(i,j)=V(i,j);

}

}

matrixC = U\*sigma\*Vn;

}

else{

matrixC =U\*sigma\*V;

}

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

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

if(fabs(matrixC(i,j))<10e-4)

matrixC(i,j)=0;

cout << setw(7)<<setprecision(3)<<fixed<<matrixC(i,j)<<"\t";

}

cout << endl;

}

}

int main()

{

cout << "Enter number of row and cols : "; cin >>rows>>cols;

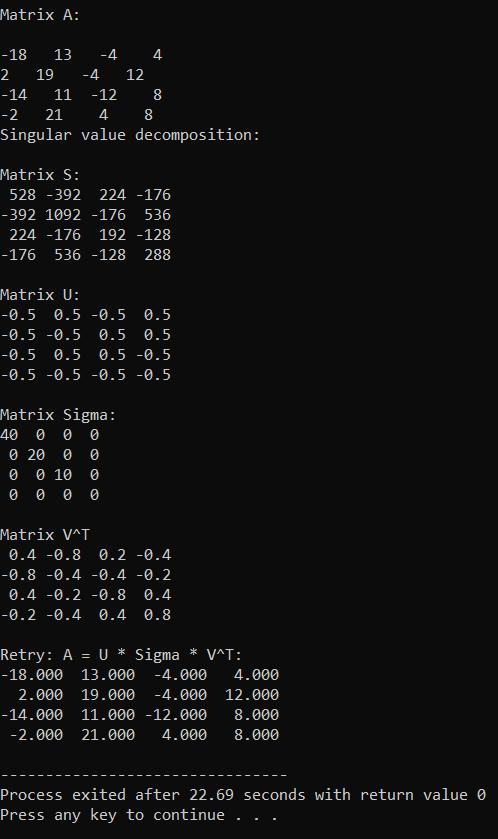
input();

outputMatrix();

SVD();

return 0;

}



Câu 2:

#include<iostream>

#include<iomanip>

#include<Eigen/Dense>

using namespace std;

using namespace Eigen;

const int MAX = 10;

typedef double m2c[MAX][MAX];

void inputMatrix(int &m,int &n,m2c A){

do{

cout << "Enter the number of rows: "; cin >> m;

cout << "Enter the number of columns: "; cin >> n;

} while(m > n);

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

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

cout << "a["<<i<<"]["<<j<<"]= ";

cin >> A[i][j];

}

}

}

void outputMatrix(int m,int n,m2c A){

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

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

if(fabs(A[i][j]) < 0.00001)

cout << setw(10) << setprecision(3) << 0;

else cout << setw(10) << setprecision(3) << A[i][j];

}

cout << endl;

}

}

void transposeMatrix(int m,int n,m2c A,m2c B)

{

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

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

B[j][i] = A[i][j];

}

}

}

void multiplyMatrix(int m,int n,int q, m2c a,m2c b,m2c c){

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

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

c[i][j] = 0;

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

c[i][j] += a[i][k]\*b[k][j];

}

}

}

}

void singularValueDecomposition(int m,int n,m2c A){

m2c B,C,U,W,V,VT;

// A^T \* A

transposeMatrix(m,n,A,B);

multiplyMatrix(n,m,n,B,A,C);

cout <<endl << endl <<"Matrix S = A^T\*A ="<< endl;

outputMatrix(n,n,C);

//Tim tri rieng va vector rieng

MatrixXd A\_eigen(n,n);

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

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

A\_eigen(i,j) = C[i][j];

}

}

EigenSolver<MatrixXd> eigensolver(A\_eigen);

if (eigensolver.info() != Success) {

cout << "Can't calculate eigenValues and eigenVectors!'" << endl;

return;

}

double lamda[n];

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

lamda[i] = real(eigensolver.eigenvalues()(i));

if(fabs(lamda[i]) < 1e-10) lamda[i] = 0;

}

//Tinh ma tran V

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

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

V[i][j] = real(eigensolver.eigenvectors()(i,j));

if(fabs(V[i][j]) < 1e-10) V[i][j] = 0;

}

}

for(int i=0; i<n-1; i++){

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

if(lamda[i] < lamda[j]){

// xep lamda giam dan

double temp = lamda[i];

lamda[i] = lamda[j];

lamda[j] = temp;

// xep vector rieng da duoc chuan hoa tuong ung voi tri rieng

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

double temp = V[k][i];

V[k][i] = V[k][j];

V[k][j] = temp;

}

}

}

}

//Tinh ma tran xich ma

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

W[i][i] = sqrt(lamda[i]);

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

if(i!=j) W[i][j] = 0;

}

}

//Tinh ma tran U

multiplyMatrix(m,n,n,A,V,U);

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

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

U[j][i] = 1.0/sqrt(lamda[i])\*U[j][i];

if(fabs(U[j][i]) < 1e-10) U[j][i] = 0;

}

}

//Tinh ma tran V^T

transposeMatrix(n,n,V,VT);

cout<< endl<<"EigenValue of matrix S = ";

for(int i=0; i<n; i++) cout << lamda[i] << " ";

cout << endl << "Singular values decomposition(SVD): A = U\*xichMa\*V^T";

cout<< endl<<endl<< "Matrix U = " <<endl;

outputMatrix(m,m,U);

cout<< endl<<"Matrix xichMa ="<<endl;

outputMatrix(m,n,W);

cout<< endl<< "Matrix V^T =" <<endl;

outputMatrix(n,n,VT);

cout <<endl<<"Check SVD of Matrix A: " << endl;

m2c kq,k;

multiplyMatrix(m,m,n,U,W,kq);

multiplyMatrix(m,n,n,kq,VT,k);

cout<<"Matrix A after check (A = U\*W\*V^T)" <<endl;

outputMatrix(m,n,k);

}

int main(){

int m,n;

m2c A,B,C;

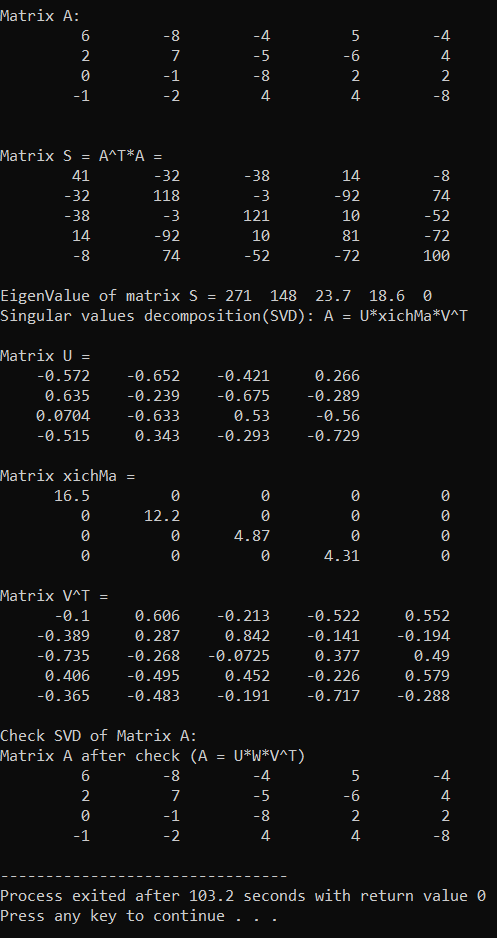
inputMatrix(m,n,A);

cout <<endl<<"Matrix A:" << endl;

outputMatrix(m,n,A);

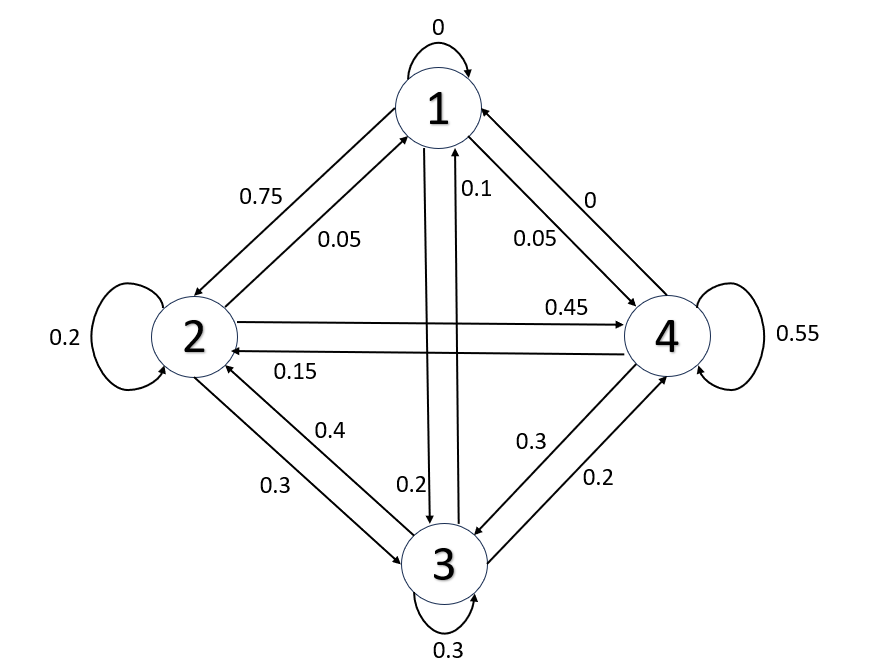
singularValueDecomposition(m,n,A);

}



Câu 3:

1. Đồ thị biểu diễn



1. State 1 after 1, 2, 3 steps:

#include<iostream>

#include <cmath>

#include <vector>

#include <iomanip>

#include <Eigen/dense>

using namespace std;

#define MAX 10

void OutputMatrix(double A[][MAX], int n, int m)

{

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

{

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

std::cout << std::setw(7) << std::setprecision(3) << (abs(A[i][j]) < 0.001f ? 0 : A[i][j]);

std::cout << "\n";

}

}

void MatrixMultiply(double A[][MAX], double B[][MAX], int a, int p, int b, double C[][MAX])

{

double temp[MAX][MAX];

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

{

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

{

temp[i][j] = 0;

for (int m = 0; m < p; m++)

{

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

}

}

}

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

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

C[i][j] = temp[i][j];

}

void InverseMatrix(double matrix[][MAX], int n, double res[][MAX]) {

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

{

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

{

res[i][j] = 0;

}

}

std::vector<std::vector<double> > augmented(n, std::vector<double>(2 \* n, 0.0));

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

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

augmented[i][j] = matrix[i][j];

augmented[i][j + n] = (i == j) ? 1.0 : 0.0; // Identity matrix

}

}

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

double pivot = augmented[i][i];

if (pivot == 0) {

return; // Matrix is singular, no unique inverse

}

for (int j = 0; j < 2 \* n; ++j) {

augmented[i][j] /= pivot;

}

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

if (k != i) {

double factor = augmented[k][i];

for (int j = 0; j < 2 \* n; ++j) {

augmented[k][j] -= factor \* augmented[i][j];

}

}

}

}

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

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

res[i][j] = augmented[i][j + n];

}

}

}

void Eigenvalues(double A[][MAX], int n, int m, double C[])

{

Eigen::MatrixXd S(n, m);

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

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

S(i, j) = A[i][j];

Eigen::EigenSolver<Eigen::MatrixXd> solver(S);

Eigen::VectorXd eivalues = solver.eigenvalues().real();

for (int i = 0; i < eivalues.size(); ++i)

{

std::cout << eivalues(i) << "\n";

C[i] = eivalues(i);

}

}

void Eigenvectors(double A[][MAX], int n, int m, double C[][MAX])

{

Eigen::MatrixXd S(n, m);

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

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

S(i, j) = A[i][j];

Eigen::EigenSolver<Eigen::MatrixXd> solver(S);

Eigen::MatrixXd eivectors = solver.eigenvectors().real();

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

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

C[i][j] = eivectors(i, j);

}

void Sort(double values[MAX], double vectors[MAX][MAX], int n)

{

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

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

{

if (values[i] < values[j])

{

double temp = values[i];

values[i] = values[j];

values[j] = temp;

double tempV[MAX];

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

{

tempV[k] = vectors[k][i];

vectors[k][i] = vectors[k][j];

vectors[k][j] = tempV[k];

}

}

}

}

void MatrixPower(double M[][MAX], int p, int n, double C[][MAX])

{

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

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

{

if (i == j) C[i][j] = 1;

else C[i][j] = 0;

}

/\*for (int i = 0; i < p; ++i)

{

MatrixMultiply(C, M, n, C);

}\*/

while (p)

{

if (p % 2)

MatrixMultiply(C, M, n, n, n, C);

MatrixMultiply(M, M, n, n, n, M);

p /= 2;

}

}

float CalculateProbability(double M[][MAX], int n, int F, int S, int T)

{

double P[MAX][MAX];

MatrixPower(M, T + 1, n, P);

std::cout << "\nTransition Matrix at time " << T << ":\n";

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

{

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

std::cout << P[i][j] << " ";

std::cout << "\n";

}

return P[S - 1][F - 1];

}

int main()

{

int n = 4;

double G[MAX][MAX] = {

{0, 0.75, 0.2, 0.05},

{0.05, 0.2, 0.3, 0.45},

{0.1, 0.4, 0.3, 0.2},

{0, 0.15, 0.3, 0.55}

};

int rS = 1;

double S[][MAX] = {

{0, 1, 0, 0 }

};

std::cout << "Original Transition Matrix:\n";

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

{

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

std::cout << G[i][j] << " ";

std::cout << "\n";

}

double values[MAX];

double vectors[MAX][MAX], iP[MAX][MAX], D[MAX][MAX];

Eigenvalues(G, n, n, values);

Eigenvectors(G, n, n, vectors);

Sort(values, vectors, n);

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

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

{

if (i == j)

D[i][i] = values[i];

else

D[i][j] = 0;

}

InverseMatrix(vectors, n, iP);

std::cout << "\nP---------\n";

OutputMatrix(vectors, n, n);

std::cout << "\nD---------\n";

OutputMatrix(D, n, n);

std::cout << "\nInverse P---------\n";

OutputMatrix(iP, n, n);

double K[MAX][MAX], temp[MAX][MAX];

MatrixMultiply(vectors, D, n, n, n, temp);

MatrixMultiply(temp, iP, n, n, n, K);

std::cout << "\n";

OutputMatrix(K, n, n);

double Dn[MAX][MAX];

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

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

if (i == j)

Dn[i][i] = 1;

else

Dn[i][j] = 0;

}

std::cout << "\nState after n steps:\n";

for (int i = 1; i <= 3; ++i) {

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

Dn[k][k] \*= D[k][k];

}

std::cout << "\tState after " << i << " steps:\n";

double Pn[MAX][MAX], temp[MAX][MAX];

double Sn[MAX][MAX];

MatrixMultiply(vectors, Dn, n, n, n, temp);

MatrixMultiply(temp, iP, n, n, n, Pn);

MatrixMultiply(S, Pn, 1, n, n, Sn);

std::cout << "\t";

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

std::cout << Sn[0][j] << " ";

std::cout << "------- Pro of state" << rS << " is: " << Sn[0][rS - 1] << "\n";

}

return 0;

}

