|  |
| --- |
| TRƯỜNG ĐẠI HỌC BÁCH KHOA  **KHOA CÔNG NGHỆ THÔNG TIN** |

**ĐỀ THI VÀ BÀI LÀM**

Tên học phần: Toán ứng dụng CNTT

Mã học phần: Hình thức thi: *Tự luận có giám sát*

Đề số: **Đ0002** Thời gian làm bài: 90 phút *(không kể thời gian chép/phát đề)*

Được sử dụng tài liệu khi làm bài.

**Họ tên:** Trần Ngọc Minh Hoàng **Lớp**:22T\_Nhật2**MSSV**:102220319

Sinh viên làm bài trực tiếp trên tệp này, lưu tệp với định dạng MSSV\_HọTên.pdf và nộp bài thông qua MS Teams.

***Câu 1*** (*2.0 điểm*): Viết chương trình (có sử dụng hàm) thực hiện công việc sau, biết rằng N=9000:  
 a) *(1.0 điểm)* Tìm số lượng và liệt kê các số hoàn hảo nhỏ hơn N.

|  |
| --- |
| **# Trả lời: Dán code bên dưới:**  **void listperfect(int n, bool Sieve[] ) {**  **int index = 0;**  **for(int i = 1; i < n; i++) {**  **if(perfectNumber(i, Sieve) == true) {**  **cout << i << " ";**  **index++;**  **}**  **}**  **cout << "So luong so hoan hao nho hon N la: " << index;**  **}**  **bool perfectNumber(int n, bool Sieve[]) {**  **if(n == Sumfactor(n, Sieve) - n) return true;**  **else return false;**  **}**  **long long Sumfactor(int n, bool Sieve[]){**    **int factor[10001];**  **for(int i = 2 ; i <= 10000; i++) {**  **factor[i] = 0;**  **}**    **int prime = 2;**  **while(n > 1) {**  **if(Sieve[prime] == true) {**  **while((n > 1) && (n % prime == 0)) {**  **factor[prime] += 1;**  **n = n / prime;**  **}**  **}**  **prime++;**  **}**  **long long sum = 1;**  **for(int i = 2; i < 10001; i++){**  **if(factor[i] != 0){**  **sum \*= (pow(i, factor[i] + 1) - 1) / (i - 1);**  **}**  **}**  **return sum;**  **}**  **# Trả lời: Dán kết quả thực thi vào bên dưới:** |

1. *(1.0 điểm)* Cho M là số hoàn hảo lớn nhất vừa tìm được. Tìm số nguyên tố gần M nhất.

|  |
| --- |
| **# Trả lời: Dán code vào bên dưới:**  **long long findClosestPrime(int n, bool Sieve[]){**  **int i = n + 1;**  **if (n < 2) return 2;**  **while(1){**  **if(Sieve[i] == true) break;**  **else i++;**  **}**  **int j = n - 1;**  **while(1) {**  **if(Sieve[j] == true) break;**  **else j--;**  **}**  **if (abs(n - i) < abs(n - j)) return i;**  **else if (abs(n - i) > abs(n - j)) return j;**  **else {**  **cout << "It has 2 closest prime: " << i << " and " << j << endl;**  **return 0;**  **}**  **}**  **# Trả lời: Dán kết quả thực thi vào bên dưới:** |

***Câu 2*** (*2.0 điểm*): Cho ma trận A. Viết chương trình (có sử dụng hàm) thực hiện phân rã ma trận A bằng phương pháp SVD.

|  |
| --- |
| **# Trả lời:** **Dán code vào bên dưới**  **// Dieu kien cua ma tran A: khi phan ra Eigen ma tran A\_T \* A, phai co it nhat 1 Eigen Value**  **#include<bits/stdc++.h>**  **#include"Eigen/Dense"**  **#include"Eigen/Eigenvalues"**  **using namespace std;**  **using namespace Eigen;**  **#define Matrix vector<vector<float>>**  **Matrix operator\*(Matrix a, Matrix b) {**  **Matrix c;**  **if(a[0].size() != b.size())**  **return c;**  **c = Matrix(a.size(), vector<float>(b[0].size(), 0));**  **for(int i = 0; i < c.size(); i++)**  **for(int j = 0; j < c[i].size(); j++)**  **for(int k = 0; k < b.size(); k++)**  **c[i][j] += a[i][k] \* b[k][j];**  **return c;**  **}**  **Matrix Tranpose(Matrix A) {**  **Matrix res(A[0].size(), vector<float>(A.size()));**  **for(int i = 0; i < A.size(); i++)**  **for(int j = 0; j < A[i].size(); j++)**  **res[j][i] = A[i][j];**  **return res;**  **}MatrixXf FromMatrix(Matrix A) {**  **MatrixXf res = MatrixXf::Zero(A.size(), A[0].size());**  **for(int i = 0; i < A.size(); i++)**  **for(int j = 0; j < A[i].size(); j++)**  **res(i, j) = A[i][j];**  **return res;**  **}**  **Matrix ToMatrix(MatrixXf A) {**  **Matrix res = Matrix(A.rows(), vector<float>(A.cols()));**  **for(int i = 0; i < res.size(); i++)**  **for(int j = 0; j < res[i].size(); j++)**  **res[i][j] = A(i, j);**  **return res;**  **}**  **void Print(Matrix A, string name = "") {**  **cout << name << ":\n";**  **for(int i = 0; i < A.size(); i++) {**  **for(int j = 0; j < A[i].size(); j++)**  **printf("%10.5f ", A[i][j]);**  **printf("\n");**  **}**  **printf("\n");**  **}**  **Matrix A;**  **void Input() {**  **int n, m;cout << "Matrix size n, m = ";**  **cin >> n >> m;**  **A = Matrix(n, vector<float>(m));**  **cout << "Matrix: \n";**  **for(int i = 0; i < n; i++)**  **for(int j = 0; j < m; j++)**  **cin >> A[i][j];**  **}**  **void SwapCol(Matrix &A, int i, int j) {**  **for(int k = 0; k < A.size(); k++)**  **swap(A[k][i], A[k][j]);**  **}**  **vector<Matrix> EigenDecompose(Matrix A) {**  **MatrixXf M = FromMatrix(A);**  **EigenSolver<MatrixXf> S(M);**  **Matrix P = ToMatrix(S.pseudoEigenvectors());**  **Matrix G = ToMatrix(S.pseudoEigenvalueMatrix());**  **for(int i = 0; i < G.size() - 1; i++)**  **for(int j = i + 1; j < G.size(); j++)**  **if(G[i][i] < G[j][j]) {**  **swap(G[i][i], G[j][j]);**  **SwapCol(P, i, j);**  **}**  **return {P, G};**  **}**  **vector<float> MatrixSolve(Matrix A) {**  **for(int i = 0; i < A.size(); i++) {int i\_m = i, v\_m = A[i][i];**  **for(int j = i + 1; j < A.size(); j++)**  **if(abs(A[j][i] > v\_m)) {**  **v\_m = A[j][i];**  **i\_m = j;**  **}**  **if(i\_m != i)**  **swap(A[i], A[i\_m]);**  **for (int j = i + 1; j < A.size(); j++) {**  **double f = 0;**  **if(A[i][i] != 0)**  **f = A[j][i] / A[i][i];**  **for (int k = i + 1; k < A[j].size(); k++)**  **A[j][k] -= A[i][k] \* f;**  **A[j][i] = 0;**  **}**  **}**  **for(int i = 0; i < A.size(); i++)**  **for(int j = 0; j < A.size(); j++)**  **if(isnan(A[i][j]))**  **A[i][j] = 0;**  **vector<float> X(A.size(), 1);**  **for(int i = 0; i < A.size(); i++) {**  **int j;**  **bool f = false;**  **for(j = 0; j < A.size(); j++)**  **if(A[i][j] != 0) {**  **f = true;**  **break;**  **}if(f)**  **X[j] = -i;**  **}**  **for(int i = X.size() - 1; i >= 0; i--)**  **if(X[i] <= 0) {**  **int r = -X[i];**  **X[i] = 0;**  **for(int j = i + 1; j < A[r].size(); j++)**  **X[i] -= X[j] \* A[r][j] / A[r][i];**  **}**  **float s = 0;**  **for(int i = 0; i < X.size(); i++)**  **s += X[i] \* X[i];**  **for(int i = 0; i < X.size(); i++)**  **X[i] /= sqrt(s);**  **return X;**  **}**  **vector<Matrix> SVDDecompose(Matrix A) {**  **Matrix A\_T = Tranpose(A);**  **vector<Matrix> E = EigenDecompose(A\_T \* A);**  **Print(E[0], "Eigen Vector");**  **Print(E[1], "Eigen Values");**  **Matrix V = E[0];**  **Matrix S(A.size(), vector<float>(A[0].size()));**  **if(isnan(E[1][0][0]) || E[1][0][0] <= 0) {**  **printf("Khong the phan ra ma tran!\n");**  **return vector<Matrix>();**  **}**  **for(int i = 0; i < min(E[1].size(), S.size()); i++)for(int j = 0; j < min(E[1][i].size(), S[i].size()); j++) {**  **if(E[1][i][j] <= 0)**  **S[i][j] = 0;**  **else**  **S[i][j] = sqrt(E[1][i][j]);**  **if(isnan(S[i][j]))**  **S[i][j] = 0;**  **}**  **Matrix U = A \* V;**  **while(U[0].size() < A.size())**  **for(int i = 0; i < U.size(); i++)**  **U[i].push\_back(0);**  **while(U[0].size() > A.size())**  **for(int i = 0; i < U.size(); i++)**  **U[i].pop\_back();**  **for(int i = 0; i < U.size(); i++) {**  **if(S[i][i] != 0)**  **for(int j = 0; j < U.size(); j++)**  **U[j][i] /= S[i][i];**  **else**  **for(int j = 0; j < U.size(); j++)**  **U[j][i] /= S[i][i];**  **}**  **for(int i = 0; i < U.size(); i++) {**  **if(i < min(S.size(), S[0].size()) && S[i][i] != 0)**  **continue;**  **vector<float> v = MatrixSolve(Tranpose(U));**  **for(int j = 0; j < U.size(); j++)**  **U[j][i] = v[j];**  **}return {U, S, Tranpose(V)};**  **}**  **int main() {**  **Input();**  **vector<Matrix> SVD = SVDDecompose(A);**  **if(SVD.size() > 0) {**  **Print(SVD[0], "U");**  **Print(SVD[1], "Sigma");**  **Print(SVD[2], "V^T");**  **Print(SVD[0] \* SVD[1] \* SVD[2], "Validating U \* Sigma \* V^T");**  **}**  **}**  **# Trả lời:** **Dán kết quả thực thi** vào bên dưới biết rằng , sai số . |

***Câu 3*** (*3.0 điểm*): Cho mười điểm trong không gian Oxy như sau: (6, 2); (8, 3); (4, 10); (3, 5); (16, 5);

(9, 7); (11, 6); (10, 12); (8, 9); (7, 6)

1. *(1.0 điểm) Mô tả thuật toán xác định bao lồi của tập điểm đã cho*

|  |
| --- |
| **# Trả lời:** **dán sơ đồ khối hoặc mã giả:**  hàm cross\_product(Point o, Point A, Point B) {          gán int x <- (A.x - o.x) \* (B.y - o.y) - (A.y - o.y) \* (B.x - O.x)          nếu x = 0 thì trả về 0          nếu x > 0 thì trả về 1          trả về 2      }      Hàm convertHull(vector<Point> A) {          Gán int n <- kích thước của A;          nếu (n bé hơn 3) trả về A          Gán vector<Point> hull <- vector<Point> trống;          Gán int left <- 0          For(i <- 1, i bé hơn n, i++) {              nếu (A[i].x bé hơn A[left].x) thì gán left <- i          }          Gán int p <- left, q          While(p không bằng left) {              đưa vào hull A[p];              gán p <- (p+1) % n;              For(i <- 0, i bé hơn n, i++) {                  nếu (cross\_product(A[p], A[i], A[q]) bằng 2) thì gán q <- i                  }              gán p <- q;          }          trả về hull      } |

1. *(1.0 điểm)* Viết hàm xác định bao lồi và cạnh nhỏ nhất của đa giác lồi vừa tìm được

|  |
| --- |
| **# Trả lời:** **Dán code bên dưới:**  class Point{  public:  double x, y;  Point(double x, double y){  this->x = x;  this->y = y;  }  Point(){  }  };  int cross\_product(Point O, Point A, Point B){  int x = (A.x - O.x) \* (B.y - O.y) - (A.y - O.y) \* (B.x - O.x);  if(x == 0) return 0;  else if(x > 0) return 1;  else return 2;  }  vector<Point> Convex\_Hull(vector<Point> A){  int n = A.size();  if(n < 3) return A;  vector<Point> hull;  int left = 0;  for(int i = 1; i < n; i++){  if(A[i].x < A[left].x){  left = i;  }  }  int p = left, q;  do{  hull.push\_back(A[p]);  q = (p+1) % n ;  for(int i = 0; i < n; i++){  if(cross\_product(A[p], A[i], A[q]) == 2){  q = i;  }  }  p = q;  }while(p != left);  return hull;  }  int main(){  vector<Point> points;    points.push\_back({ 6, 2 });  points.push\_back({ 8, 3 });  points.push\_back({ 4, 10 });  points.push\_back({ 3, 5 });  points.push\_back({ 16, 5 });  points.push\_back({ 9, 7 });  points.push\_back({ 11, 6 });  points.push\_back({ 10, 12 });  points.push\_back({ 8, 9 });  points.push\_back({ 7, 6 });  vector<Point> ans = Convex\_Hull(points);  for (int i = 0; i < ans.size(); i++)  cout << "(" << ans[i].x << ", " << ans[i].y << ")"  << endl;    double shot = 100000;  for(int i = 0; i < ans.size(); i++) {  for(int j = i + 1; j < ans.size(); j++) {  double dis = shortest(ans[i], ans[j]);  if(dis < shot) {  shot = dis;  }  }  }  cout << "Shortest distance in converthull: " << shot << endl;  vector<Point> nothull = elementsInAAndNotInB(points, ans);    cout << "Not in converhull: " << endl;  for(int i = 0; i < nothull.size(); i++) {  cout << "(" << nothull[i].x << ", " << nothull[i].y << ")" << endl;  }  return 0;  }  **# Trả lời:** **Dán kết quả thực thi vào bên dưới:** |

c)  *(1.0 điểm)* Xác định số lượng các điểm nằm bên trong bao lồi và liệt kê chúng

|  |
| --- |
| **# Trả lời:** **Dán code bên dưới:**  vector<Point> elementsInAAndNotInB( vector<Point> A, vector<Point> B) {  vector<Point> result;  for (const Point& p : A) {  if (!isInVector(B, p)) {  result.push\_back(p);  }  }  return result;  }  int main(){  vector<Point> points;    points.push\_back({ 6, 2 });  points.push\_back({ 8, 3 });  points.push\_back({ 4, 10 });  points.push\_back({ 3, 5 });  points.push\_back({ 16, 5 });  points.push\_back({ 9, 7 });  points.push\_back({ 11, 6 });  points.push\_back({ 10, 12 });  points.push\_back({ 8, 9 });  points.push\_back({ 7, 6 });  vector<Point> ans = Convex\_Hull(points);  for (int i = 0; i < ans.size(); i++)  cout << "(" << ans[i].x << ", " << ans[i].y << ")"  << endl;    double shot = 100000;  for(int i = 0; i < ans.size(); i++) {  for(int j = i + 1; j < ans.size(); j++) {  double dis = shortest(ans[i], ans[j]);  if(dis < shot) {  shot = dis;  }  }  }  cout << "Shortest distance in converthull: " << shot << endl;  vector<Point> nothull = elementsInAAndNotInB(points, ans);    cout << "Not in converhull: " << endl;  for(int i = 0; i < nothull.size(); i++) {  cout << "(" << nothull[i].x << ", " << nothull[i].y << ")" << endl;  }  return 0;  }  **# Trả lời:** **Dán kết quả thực thi vào bên dưới:** |

***Câu 4*** (*2.0 điểm*): Cho hàm số .

1. *(1.0 điểm) Trình bày thuật toán tối ưu hàm số đã cho* sử dụng phương pháp *gradient descent với momentum*, biết rằng tham số học (learning rate) , hệ số động lượng là .

|  |
| --- |
| **# Trả lời: dán sơ đồ khối hoặc mã giả:**  hàm func(double x) {  trả về hàm f(x)  }  hàm grad(double x) {  trả về đạo hàm f(x)  }  Hàm GB\_momentum(double x) {  Gán double deltax <- 0  Gán double deltaX\_new <- 0  gasn double x\_new <- 0  For (i <- 1, i <- 10000, i++) {  deltaX\_new <- -GAMMA \* grad(x) + alpha \* deltaX  x\_new <- x - GAMMA \* grad(x) + alpha \* deltaX\_new;  deltaX <- deltaX\_new;  Nếu (trị tuyệt đối (x - x\_new) lớn hơn sai số yêu cầu) {  break;  } còn không {  x <- x\_new;  }  }  In ra "Min x = " + x;  } |

1. *(1.0 điểm)* Viết chương trình (có dùng hàm) tính giá trị bé nhất của f(x) sử dụng phương pháp *gradient descent với momentum* với số bước lặp *N* và sai số .

|  |
| --- |
| **# Trả lời**: **Dán code vào bên dưới:**  **#include <iostream>**  **#include <cmath>**  **#define EPSILON 1e-5**  **#define GAMMA 0.001**  **#define N 10000**  **#define alpha 0.1**  **using namespace std;**  **double func(double x)**  **{**  **return (exp(2\*x) + 3\*x\*x + 8\*x) / (35 - x) - 5\*x;**  **}**  **double grad(double x)**  **{**  **return ((35 - x)\* (2\*exp(2\*x) + 6 \* x + 8) + exp(2\*x) + 3 \* x \* x + 8 \* x) / pow(35 - x, 2) - 5;**  **}**  **double GD\_momentum(double x)**  **{**  **double deltaX = 0;**  **double deltaX\_new = 0;**  **double x\_new = 0;**  **for (int i = 1; i < N; i++)**  **{**  **deltaX\_new = -GAMMA \* grad(x) + alpha \* deltaX;**  **x\_new = x - GAMMA \* grad(x) + alpha \* deltaX\_new;**  **deltaX = deltaX\_new;**  **if (fabs(x - x\_new) < EPSILON)**  **{**  **break;**  **}**  **else**  **{**  **x = x\_new;**  **}**  **}**  **cout << "Min x = " << x << endl;**  **return func(x);**  **}**  **int main()**  **{**  **double x = 0;**  **cout << "f(x) = (e^(2x) +3(x^2) +8x) / (35 - x) - 5x" << endl;**  **double minfX = GD\_momentum(x);**  **cout << "Min f(x): " << minfX << endl;**  **return 0;**  **}**  **# Trả lời**: **Dán kết quả thực thi** với điểm khởi , tham số học học (*learning rate*) , hệ số động lượng (*momentum coefficient*) là , số bước lặp và sai số : |
|  |

***Câu 5*** (*1.0 điểm*): Một hệ thống có chế độ làm việc ở mỗi giai đoạn vận hành chỉ với các trạng thái từ 1 đến 4. Chế độ làm việc của hệ thống này được mô tả bằng ma trận chuyển như sau:

a) (*0.5 điểm*) Vẽ đồ thị biễu diễn chuỗi Markov tương ứng đã cho

|  |
| --- |
| **# Trả lời:** **Dán kết quả vào bên dưới**  Không có mô tả. |

b) (*0.5 điểm*) Giả sử rằng hệ thống bắt đầu học ở trạng thái 1. Tính xác xuất hệ thống làm việc ở trạng thái 4 *sau ba và bốn bước thời gian vận hành*.

|  |
| --- |
| **# Trả lời**: **Dán kết quả tính toán vào bên dưới:** |