|  |
| --- |
| 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:** Nguyễn Hữu Khoa………**Lớp**:……22T\_DT5………**MSSV**:…102220237…...

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:**  **#include <bits/stdc++.h>**  **// bai 2**  **using namespace std;**  **#define ll long long**  **bool is\_prime(long long n)**  **{**  **if (n <= 1)**  **{**  **return false;**  **}**  **for (int i = 2; i \* i <= n; i++)**  **{**  **if (n % i == 0)**  **{**  **return false;**  **}**  **}**  **return true;**  **}**  **struct Element**  **{**  **ll value;**  **int count;**  **};**  **ll GCD(ll a, ll b)**  **{**  **if (b == 0)**  **return a;**  **else**  **return GCD(b, a % b);**  **}**  **void factorize(vector<Element> \*A, ll n)**  **{**  **int count = 0;**  **ll temp = n;**  **while (!(temp % 2))**  **{**  **temp = temp / 2;**  **count++;**  **}**  **if (count)**  **{**  **// cout << 2 << "^" << count;**  **struct Element B;**  **B.count = count;**  **B.value = 2;**  **A->push\_back(B);**  **}**  **for (long long i = 3; i <= sqrt(n); i += 2)**  **{**  **count = 0;**  **while (temp % i == 0)**  **{**  **count++;**  **temp = temp / i;**  **}**  **if (count)**  **{**  **struct Element B;**  **B.count = count;**  **B.value = i;**  **A->push\_back(B);**  **// cout << "\*" << i << "^" << count << endl;**  **}**  **}**  **}**  **long long sumOfFactor(ll n)**  **{**  **vector<Element> temp1;**  **factorize(&temp1, n);**  **long long S = 1;**  **for (int i = 0; i < temp1.size(); i++)**  **{**  **struct Element temp = temp1.at(i);**  **S \*= (pow(temp.value, temp.count + 1) - 1) / (temp.value - 1);**  **}**  **return S;**  **}**  **bool isPerfect(ll n)**  **{**  **return (n == sumOfFactor(n) - n);**  **}**  **void PerfectNumberList(ll n)**  **{**  **cout << "List cac so hoan hao nho hon n: " << n << "\n";**  **cout << "a";**  **for (ll i = 0; i < n; i++)**  **{**  **cout << "a";**  **if (i == (sumOfFactor(i) - i))**  **{**  **cout << i << "\n";**  **}**  **}**  **}**  **ll getMaximumPerfectValue(ll n)**  **{**  **ll max = 0;**  **for (ll i = 0; i < n; i++)**  **{**  **if (isPerfect(i) == true)**  **max = i;**  **}**  **return max;**  **}**  **// void ETF(ll number)**  **// {**  **// int S;**  **// cout << "list: \n";**  **// for (int i = 1; i<number; i++)**  **// {**  **// ll a = GCD(i,number);**  **// if (a == 1){**  **// S+=i;**  **// cout << i << " ";**  **// }**  **// }**  **// cout << "\n";**  **// cout << "sum: " << S << "\n";**  **// }**  **// void ETFFORMULA(ll n)**  **// {**  **// vector<Element> temp;**  **// factorize(&temp, n);**  **// // for (int i = 0; i < temp.size(); i++)**  **// // {**  **// // cout << temp.at(i).value << " " << temp.at(i).count;**  **// // // A\*= pow(temp.at(i).value, temp.at(i).count-1)\*(temp.at(i).value - 1);**  **// // }**  **// ll A = 1;**  **// for (int i = 0; i < temp.size(); i++)**  **// {**  **// // cout << temp.at(i).value << " " << temp.at(i).count;**  **// A\*= pow(temp.at(i).value, temp.at(i).count-1)\*(temp.at(i).value - 1);**  **// }**  **// cout << A;**  **// }**  **long long nearest\_prime(ll n)**  **{**  **ll lower = n - 1;**  **while (!is\_prime(lower) && lower % 3 != 0)**  **{**  **lower -= 1;**  **}**  **ll upper = n + 1;**  **while (!is\_prime(upper) && upper % 3 != 0)**  **{**  **upper += 1;**  **}**  **return abs(n - lower) < abs(n - upper) ? lower : upper;**  **}**  **int main()**  **{**  **ll n;**  **// cout << "input the value of n: ";**  **// cin >> n;**  **cout << "cau a: \n";**  **PerfectNumberList(9000);**  **// cout << "cau b: \n";**  **// cout << "nearest prime of maximum perfect value of n: ";**  **// cout << nearest\_prime(getMaximumPerfectValue(9000));**  **// cout << "\n the number of coprime number using ETF formula: ";**  **// ETFFORMULA(n);**  **}**  **# 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:**  #include <bits/stdc++.h>  // bai 2  using namespace std;  #define ll long long  bool is\_prime(long long n)  {  if (n <= 1)  {  return false;  }  for (int i = 2; i \* i <= n; i++)  {  if (n % i == 0)  {  return false;  }  }  return true;  }  struct Element  {  ll value;  int count;  };  ll GCD(ll a, ll b)  {  if (b == 0)  return a;  else  return GCD(b, a % b);  }  void factorize(vector<Element> \*A, ll n)  {  int count = 0;  ll temp = n;  while (!(temp % 2))  {  temp = temp / 2;  count++;  }  if (count)  {  // cout << 2 << "^" << count;  struct Element B;  B.count = count;  B.value = 2;  A->push\_back(B);  }  for (long long i = 3; i <= sqrt(n); i += 2)  {  count = 0;  while (temp % i == 0)  {  count++;  temp = temp / i;  }  if (count)  {  struct Element B;  B.count = count;  B.value = i;  A->push\_back(B);  // cout << "\*" << i << "^" << count << endl;  }  }  }  long long sumOfFactor(ll n)  {  vector<Element> temp1;  factorize(&temp1, n);  long long S = 1;  for (int i = 0; i < temp1.size(); i++)  {  struct Element temp = temp1.at(i);  S \*= (pow(temp.value, temp.count + 1) - 1) / (temp.value - 1);  }  return S;  }  bool isPerfect(ll n)  {  return (n == sumOfFactor(n) - n);  }  void PerfectNumberList(ll n)  {  cout << "List cac so hoan hao nho hon n: " << n << "\n";  cout << "a";  for (ll i = 0; i < n; i++)  {  cout << "a";  if (i == (sumOfFactor(i) - i))  {  cout << i << "\n";  }  }  }  ll getMaximumPerfectValue(ll n)  {  ll max = 0;  for (ll i = 0; i < n; i++)  {  if (isPerfect(i) == true)  max = i;  }  return max;  }  // void ETF(ll number)  // {  // int S;  // cout << "list: \n";  // for (int i = 1; i<number; i++)  // {  // ll a = GCD(i,number);  // if (a == 1){  // S+=i;  // cout << i << " ";  // }  // }  // cout << "\n";  // cout << "sum: " << S << "\n";  // }  // void ETFFORMULA(ll n)  // {  // vector<Element> temp;  // factorize(&temp, n);  // // for (int i = 0; i < temp.size(); i++)  // // {  // // cout << temp.at(i).value << " " << temp.at(i).count;  // // // A\*= pow(temp.at(i).value, temp.at(i).count-1)\*(temp.at(i).value - 1);  // // }  // ll A = 1;  // for (int i = 0; i < temp.size(); i++)  // {  // // cout << temp.at(i).value << " " << temp.at(i).count;  // A\*= pow(temp.at(i).value, temp.at(i).count-1)\*(temp.at(i).value - 1);  // }  // cout << A;  // }  long long nearest\_prime(ll n)  {  ll lower = n - 1;  while (!is\_prime(lower) && lower % 3 != 0)  {  lower -= 1;  }  ll upper = n + 1;  while (!is\_prime(upper) && upper % 3 != 0)  {  upper += 1;  }  return abs(n - lower) < abs(n - upper) ? lower : upper;  }  int main()  {  ll n;  // cout << "input the value of n: ";  // cin >> n;  // cout << "cau a: \n";  // PerfectNumberList(9000);  cout << "cau b: \n";  cout << "nearest prime of maximum perfect value of n: ";  cout << nearest\_prime(getMaximumPerfectValue(9000));  cout << "\n the number of coprime number using ETF formula: ";  // ETFFORMULA(n);  }  **# 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**  // g++ -I C:\eigen-3.4.0 tenfile.cpp -o tenfile.exe  // tinh toan 3 ma tran U, sigma, V  #include <bits/stdc++.h>  #include <Eigen/Dense>  using namespace std;  using namespace Eigen;  MatrixXf Transpose\_1(MatrixXf A, int row, int column)  {  MatrixXf A\_trans(column, row);  for (int i = 0; i < row; i++)  {  for (int j = 0; j < column; j++)  {  A\_trans(j, i) = A(i, j);  }  }  return A\_trans;  }  MatrixXf Multiple(MatrixXf A, int A\_row, int A\_col, MatrixXf B, int B\_row, int B\_col)  {  MatrixXf result(A\_col, B\_row);  for (int i = 0; i < A\_col; i++)  {  for (int j = 0; j < B\_row; j++)  {  result(i, j) = 0;  for (int k = 0; k < B\_col; k++)  {  result(i, j) += A(i, k) \* B(k, j);  }  }  }  return result;  }  MatrixXf MatrixDacTrung(MatrixXf A, int row, int column)  {  MatrixXf At = Transpose\_1(A, row, column);  MatrixXf mtdt = Multiple(At, row, column, A, column, row);  return mtdt;  // MatrixXf At = A.transpose();  // MatrixXf mtdt = At\*A;  // return mtdt;  }  MatrixXf vtCal(MatrixXf A, int row, int column)  {  MatrixXf mtdt = MatrixDacTrung(A, row, column);  EigenSolver<MatrixXf> es(mtdt);  EigenSolver<MatrixXf>::EigenvectorsType eigenVectors = es.eigenvectors();  MatrixXf V = eigenVectors.real().matrix();  return V;  }  MatrixXf sigmaCal(MatrixXf A, int row, int column)  {  MatrixXf mtdt = MatrixDacTrung(A, row, column);  EigenSolver<MatrixXf> es(mtdt);  EigenSolver<MatrixXf>::EigenvalueType eigenValues = es.eigenvalues();  EigenSolver<MatrixXf>::EigenvectorsType eigenVectors = es.eigenvectors();  MatrixXf sigma = MatrixXf::Zero(eigenValues.rows(), eigenValues.rows());  // cout << "\n";  for (int i = 0; i < eigenValues.rows(); i++)  {  sigma(i, i) = sqrt((eigenValues[i].real()));  }  // sigma(eigenValues.rows() - 1, eigenValues.rows() - 1) = 0.000001;  // cout << sigma;  return sigma;  // có được ma trận đường chéo lambda  }  MatrixXf uCal(MatrixXf A, int row, int column) // sửa uCal tùy bài  {  // MatrixXf mtdt = MatrixDacTrung(A, row, column);  // EigenSolver<MatrixXf> es(A);  // EigenSolver<MatrixXf>::EigenvalueType eigenValues = es.eigenvalues();  // EigenSolver<MatrixXf>::EigenvectorsType eigenVectors = es.eigenvectors();  MatrixXf V = vtCal(A, row, column);  // int row1 = V.rows();  // int col1 = V.cols();  MatrixXf U = A \* V;  // MatrixXf U = Multiple(A, row, column , V, column, column);  for (int i = 0; i < U.cols(); ++i)  U.col(i).normalize();  return U;  }  void check(MatrixXf U, MatrixXf sigma, MatrixXf Vt)  {  cout << "\n"  << setprecision(5) << (U \* sigma \* Vt.transpose()).real() << "\n";  }  int main(int argc, char const \*argv[])  {  int row = 3;  int column = 3;  MatrixXf A = MatrixXf(row, column);    A << 1, 2, 3,  2, 4, 8,  1, 2, 4;  // A << -3, 1,  // 6, -2,  // 6, -2;  // A << -18, 13, -4, 4,  // 2, 19, -4, 12,  // -14, 11, -12, 8,  // -2, 21, 4, 8;  // A << 1, 0, 1,  // -2, 1, 0;  MatrixXf ATA = MatrixDacTrung(A, row, column);  // cout << ATA << "\n";  MatrixXf U = uCal(A, row, column);  cout << "U: " << "\n";  cout << setprecision(5) << U << "\n";  MatrixXf sigma = sigmaCal(A, row, column);  cout << "sigma: " << "\n";  cout << setprecision(2) << sigma << "\n";  MatrixXf Vt = vtCal(A, row, column);  cout << "Vt: " << "\n";  cout << setprecision(5) << Vt << "\n";  // check(U, sigma, Vt);  return 0;  }  **# 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:**  1. Chọn điểm **p** là điểm ở bên trái nhất 2. Chạy vòng lặp while cho tới khi gặp lại điểm **p**:  - chọn **q** là điểm tiếp theo, duyệt qua lại tiếp các điểm  + với mỗi điểm i, nếu i có khuynh hướng ngược chiều kim đồng hồ hơn, gán i là q  + gán điểm tiếp theo sau **p** là **q (p.next = q)**  + gán **p** = **q** 3. Output sẽ là các điểm tạo nên bao lồi |

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:**  // A C++ program to find convex hull of a set of points. Refer  // https://www.geeksforgeeks.org/orientation-3-ordered-points/  // for explanation of orientation()  #include <iostream>  #include <stack>  #include <stdlib.h>  #include <bits/stdc++.h>  using namespace std;  struct Point  {  int x, y;  };  // A global point needed for sorting points with reference  // to the first point Used in compare function of qsort()  Point p0;  // A utility function to find next to top in a stack  Point nextToTop(stack<Point> &S)  {  Point p = S.top();  S.pop();  Point res = S.top();  S.push(p);  return res;  }  // A utility function to swap two points  void swap(Point &p1, Point &p2)  {  Point temp = p1;  p1 = p2;  p2 = temp;  }  // A utility function to return square of distance  // between p1 and p2  int distSq(Point p1, Point p2)  {  return (p1.x - p2.x) \* (p1.x - p2.x) +  (p1.y - p2.y) \* (p1.y - p2.y);  }  // To find orientation of ordered triplet (p, q, r).  // The function returns following values  // 0 --> p, q and r are collinear  // 1 --> Clockwise  // 2 --> Counterclockwise  int orientation(Point p, Point q, Point r)  {  int val = (q.y - p.y) \* (r.x - q.x) -  (q.x - p.x) \* (r.y - q.y);  if (val == 0)  return 0; // collinear  return (val > 0) ? 1 : 2; // clock or counterclock wise  }  // A function used by library function qsort() to sort an array of  // points with respect to the first point  int compare(const void \*vp1, const void \*vp2)  {  Point \*p1 = (Point \*)vp1;  Point \*p2 = (Point \*)vp2;  // Find orientation  int o = orientation(p0, \*p1, \*p2);  if (o == 0)  return (distSq(p0, \*p2) >= distSq(p0, \*p1)) ? -1 : 1;  return (o == 2) ? -1 : 1;  }  // Prints convex hull of a set of n points.  void convexHull(Point points[], int n)  {  // Find the bottommost point  int ymin = points[0].y, min = 0;  for (int i = 1; i < n; i++)  {  int y = points[i].y;  // Pick the bottom-most or choose the left  // most point in case of tie  if ((y < ymin) || (ymin == y &&  points[i].x < points[min].x))  ymin = points[i].y, min = i;  }  // Place the bottom-most point at first position  swap(points[0], points[min]);  // Sort n-1 points with respect to the first point.  // A point p1 comes before p2 in sorted output if p2  // has larger polar angle (in counterclockwise  // direction) than p1  p0 = points[0];  qsort(&points[1], n - 1, sizeof(Point), compare);  // If two or more points make same angle with p0,  // Remove all but the one that is farthest from p0  // Remember that, in above sorting, our criteria was  // to keep the farthest point at the end when more than  // one points have same angle.  int m = 1; // Initialize size of modified array  for (int i = 1; i < n; i++)  {  // Keep removing i while angle of i and i+1 is same  // with respect to p0  while (i < n - 1 && orientation(p0, points[i],  points[i + 1]) == 0)  i++;  points[m] = points[i];  m++; // Update size of modified array  }  // If modified array of points has less than 3 points,  // convex hull is not possible  if (m < 3)  return;  // Create an empty stack and push first three points  // to it.  stack<Point> S;  S.push(points[0]);  S.push(points[1]);  S.push(points[2]);  // Process remaining n-3 points  for (int i = 3; i < m; i++)  {  // Keep removing top while the angle formed by  // points next-to-top, top, and points[i] makes  // a non-left turn  while (S.size() > 1 && orientation(nextToTop(S), S.top(), points[i]) != 2)  S.pop();  S.push(points[i]);  }  // Now stack has the output points, print contents of stack  while (!S.empty())  {  Point p = S.top();  cout << "(" << p.x << ", " << p.y << ")" << endl;  S.pop();  }  }  int compareX(const void \*a, const void \*b)  {  Point \*p1 = (Point \*)a, \*p2 = (Point \*)b;  return (p1->x - p2->x);  }  // Needed to sort array of points according to Y coordinate  int compareY(const void \*a, const void \*b)  {  Point \*p1 = (Point \*)a, \*p2 = (Point \*)b;  return (p1->y - p2->y);  }  // A utility function to find the  // distance between two points  float dist(Point p1, Point p2)  {  return sqrt((p1.x - p2.x) \* (p1.x - p2.x) +  (p1.y - p2.y) \* (p1.y - p2.y));  }  // A Brute Force method to return the  // smallest distance between two points  // in P[] of size n  float bruteForce(Point P[], int n)  {  float min = FLT\_MAX;  for (int i = 0; i < n; ++i)  for (int j = i + 1; j < n; ++j)  if (dist(P[i], P[j]) < min)  min = dist(P[i], P[j]);  return min;  }  // A utility function to find  // minimum of two float values  float min(float x, float y)  {  return (x < y) ? x : y;  }  // A utility function to find the  // distance between the closest points of  // strip of given size. All points in  // strip[] are sorted according to  // y coordinate. They all have an upper  // bound on minimum distance as d.  // Note that this method seems to be  // a O(n^2) method, but it's a O(n)  // method as the inner loop runs at most 6 times  float stripClosest(Point strip[], int size, float d)  {  float min = d; // Initialize the minimum distance as d  qsort(strip, size, sizeof(Point), compareY);  // Pick all points one by one and try the next points till the difference  // between y coordinates is smaller than d.  // This is a proven fact that this loop runs at most 6 times  for (int i = 0; i < size; ++i)  for (int j = i + 1; j < size && (strip[j].y - strip[i].y) < min; ++j)  if (dist(strip[i], strip[j]) < min)  min = dist(strip[i], strip[j]);  return min;  }  // A recursive function to find the  // smallest distance. The array P contains  // all points sorted according to x coordinate  float closestUtil(Point P[], int n)  {  // If there are 2 or 3 points, then use brute force  if (n <= 3)  return bruteForce(P, n);  // Find the middle point  int mid = n / 2;  Point midPoint = P[mid];  // Consider the vertical line passing  // through the middle point calculate  // the smallest distance dl on left  // of middle point and dr on right side  float dl = closestUtil(P, mid);  float dr = closestUtil(P + mid, n - mid);  // Find the smaller of two distances  float d = min(dl, dr);  // Build an array strip[] that contains  // points close (closer than d)  // to the line passing through the middle point  Point strip[n];  int j = 0;  for (int i = 0; i < n; i++)  if (abs(P[i].x - midPoint.x) < d)  strip[j] = P[i], j++;  // Find the closest points in strip.  // Return the minimum of d and closest  // distance is strip[]  return min(d, stripClosest(strip, j, d));  }  // The main function that finds the smallest distance  // This method mainly uses closestUtil()  float closest(Point P[], int n)  {  qsort(P, n, sizeof(Point), compareX);  // Use recursive function closestUtil()  // to find the smallest distance  return closestUtil(P, n);  }  int main()  {  // Point points[] = {{0, 3}, {1, 1}, {2, 2}, {4, 4},  // {0, 0}, {1, 2}, {3, 1}, {3, 3}};0  Point points[] = {{6, 2}, {8, 3}, {4, 10}, {3, 5}, {16, 5}, {9, 7}, {11, 6}, {10, 12}, {8, 9}, {7, 6}};  int T = sizeof(points) / sizeof(points[0]);  // int T = 0, x = 0, y = 0;  // cout << "enter number of points: ";  // cin>>T;  // if(T<=0)  // return -1;  // Point points[T];  // for(int i=0;i<T;++i){  // cout << "Enter x cor of " << i << " point: ";  // cin >> x;  // points[i].x=x;  // cout << "Enter y cor of " << i << " point: ";  // cin >> y;  // points[i].y=y;  // }  cout << "\n---------After Using Graham Scan Algorithm---------------\n";  cout << "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* CONVEX HULL \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";  convexHull(points, T);  cout << "The smallest distance is " << closest(points, T);  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:**  **# 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:**  B1: nhập n và giá trị ban đầu x0, khởi tạo biến tên temp, gán x = x0  B2: lặp từ 0 -> n {  temp = temp \* + \* f(x)’  X = x - temp  } B3: In kết quả 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 <bits/stdc++.h>**  **#define EPSILON 0.00001**  **#define E\_VAL 2.71828**  **#define ALPHA 0.1**  **#define GAMMA 0.001**  **using namespace std;**  **// An example function whose solution is determined using**  **// Bisection Method. The function is x^3 - x^2 + 2**  **double func(double x)**  **{**  **// return x\*x\*x - x\*x + 2;**  **// return 3\*exp((pow(x,5)- pow(x,4))) + pow(x, 2) - 20\*x + log(x+25) - 10;**  **double term1 = exp(2 \* x) + 3 \* 2 \* x \* x + 8 \* x;**  **double term2 = 35 - x;**  **double term3 = 5 \* x;**  **return (term1 / term2) - term3;**  **}**  **// Derivative of the above function which is 3\*x^x - 2\*x**  **double derivFunc(double x)**  **{**  **// return 3\*exp((x+1)\*pow(x, 4))\*(5\*x - 4)\*pow(x,3) + 2\*x + 1/(x+25) - 20;**  **double term1 = 71 \* exp(2 \* x) - 3 \* x \* x + 210 \* x - 2 \* x \* exp(2 \* x) + 280;**  **double term2 = 35 \* 35 - 2 \* 35 \* x + x \* x;**  **double term3 = -5;**  **return (term1 / term2) - term3;**  **}**  **void GDwithMomentum(double x, int n)**  **{**  **double velocity = 0;**  **double theta = x;**  **for (int i = 0; i < n; i++)**  **{**  **velocity = velocity \* ALPHA + GAMMA \* derivFunc(theta);**  **theta -= velocity;**  **}**  **cout << "x = " << theta;**  **}**  **int main(int argc, char const \*argv[])**  **{**  **double x0 = 0;**  **int n = 5000;**  **GDwithMomentum(x0, n);**  **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** |

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:** |