**1.Write MATLAB code for the Hessian and gradient of . Find the gradient of at (1, 2).**

**syms x y**

**f = 5\*x + 8\*y + x\*y - x^2 - 2\*y^2;**

**grad\_f = gradient(f, [x, y]);**

**hessian\_f = hessian(f, [x, y]);**

**grad\_val = subs(grad\_f, [x, y], [1, 2]);**

**hessian\_val = subs(hessian\_f, [x, y], [1, 2]);**

**disp('Gradient at (1,2):');**

**disp(grad\_val);**

**disp('Hessian at (1,2):');**

**disp(hessian\_val);**

**2.Write MATLAB code for the Hessian and gradient of.  
Find the Hessian and gradient at (1, 2, 3)**

**syms x1 x2 x3**

**f = x1\*x2\*x3;**

**grad\_f = gradient(f, [x1, x2, x3]);**

**hessian\_f = hessian(f, [x1, x2, x3]);**

**grad\_val = subs(grad\_f, [x1, x2, x3], [1, 2, 3]);**

**hessian\_val = subs(hessian\_f, [x1, x2, x3], [1, 2, 3]);**

**disp('Gradient at (1,2,3):');**

**disp(grad\_val);**

**disp('Hessian at (1,2,3):');**

**disp(hessian\_val);**

**3.Write MATLAB code for the Hessian and gradient of. Find both the gradient and Hessian.**

**syms x**

**f = (1/3)\*x^4 - 4\*x + (1/3)\*x^3 - 16\*exp(x)\*x^2;**

**grad\_f = diff(f, x);**

**hessian\_f = diff(grad\_f, x); % Hessian (second derivative)**

**disp('Gradient:');**

**disp(grad\_f);**

**disp('Hessian:');**

**disp(hessian\_f);**

**4.Write an example MATLAB code for checking definiteness of a matrix using the Hessian. Include**

**syms x y**

**f = x^2 + 4\*x\*y + 3\*y^2;**

**H = hessian(f, [x, y]);**

**disp('Hessian matrix: ');**

**disp(H);**

**% Check definiteness using eigenvalues**

**eigenvals = eig(H);**

**disp('Eigenvalues:');**

**disp(eigenvals);**

**if all(eigenvals > 0)**

**disp('Positive definite');**

**elseif all(eigenvals < 0)**

**disp('Negative definite');**

**elseif any(eigenvals > 0) && any(eigenvals < 0)**

**disp('Indefinite');**

**else disp('Semidefinite');**

**end**

**5.Find the minimizer of the function.Write a MATLAB code for this question.**

**syms x**

**f = (x+1)^2 + 3;**

**df = diff(f, x);**

**critical\_points = solve(df == 0, x);**

**f\_min = subs(f, x, critical\_points);**

**disp('Minimizer:');**

**disp(critical\_points);**

**disp('Minimum value:');**

**disp(f\_min);**

**6.Write an example MATLAB code for checking concavity of a function.**

**syms x**

**f = -x^2 + 4\*x - 3;**

**second\_derivative = diff(f, x, 2);**

**if second\_derivative < 0**

**disp('Function is concave');**

**elseif second\_derivative > 0**

**disp('Function is convex');**

**else**

**disp('Neither concave nor convex');**

**end**

**7.Write MATLAB code to determine whether a given function has alocal minimum, local maximum, or saddle point,using critical points and the second derivative test..**

**syms x**

**f = x^3 - 3\*x^2 + 2;**

**df = diff(f, x);**

**critical\_points = solve(df == 0, x);**

**d2f = diff(df, x);**

**for i = 1:length(critical\_points)**

**cp = critical\_points(i);**

**val = subs(d2f, x, cp);**

**if val > 0**

**fprintf('x = %.2f Local Minimum\n', cp);**

**elseif val < 0**

**fprintf('x = %.2f Local Maximum\n', cp);**

**else**

**fprintf('x = %.2f Saddle Point\n', cp);**

**end**

**end**

**8.Suppose we have a unimodal function over the interval [5, 8].Give an example of a desired final uncertainty range where theGolden Section method requires at least four iterations,whereas the Fibonacci method requires only three.(You may choose a small ε for the Fibonacci method.)**

**a = 5; b = 8; tol = 0.01;**

**phi = (1 + sqrt(5)) / 2; % Golden ratio**

**n\_golden = ceil(log(tol/(b-a))/log(1/phi));**

**fprintf('Golden section method needs %d iterations.\n', n\_golden);**

**fprintf('Fibonacci method may need about 3 iterations for same accuracy.\n');**

**9.Use the Golden Section Method to minimizeover the interval [1, 2] with an uncertainty tolerance of 0.23.Display intermediate steps using a table that defines the function )**

**f = @(x) x.^4 - 14\*x.^3 + 60\*x.^2 - 70\*x;**

**a=1; b=2; tol = 0.23;**

**phi = (1+sqrt(5)) / 2;**

**iter = 0;**

**fprintf('Iter\t a\t b\t x1\t x2\t f(x1)\t f(x2)\n');**

**while (b-a) > tol**

**x1 = b - (b-a) / phi;**

**x2 = a + (b-a) / phi;**

**f1 = f(x1); f2 = f(x2);**

**fprintf('%d\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\n', iter, a, b, x1, x2, f1, f2);**

**if f1 > f2**

**a = x1;**

**else**

**b = x2;**

**end**

**iter = iter + 1;**

**end**

**xmin = (a+b) / 2;**

**fprintf('Approximate minimum at x = %.4f\n', xmin);**

**10.Use the Fibonacci method to minimizeover the interval [1, 2] with uncertainty tolerance 0.23.Display intermediate steps using a table and define the function .**

**clc; clear; close all;**

**f = @(x) x.^4 - 14\*x.^3 + 60\*x.^2 - 70\*x;**

**a = -1; b = 2; tol = 0.23;**

**F(2)=1; F(1)=1;**

**for i=3:20**

**F(i)=F(i-1)+F(i-2);**

**end**

**n = find(F > (b-a)/tol, 1);**

**disp('Iter a b x1 x2 f(x1) f(x2)');**

**for k=1:n-2**

**x1 = a + (F(n-k-1)/F(n-k+1))\*(b-a);**

**x2 = a + (F(n-k)/F(n-k+1))\*(b-a);**

**f1 = f(x1); f2 = f(x2);**

**fprintf('%20.4f %8.4f %8.4f %8.4f %8.4f %8.4f %8.4f\n',k,a,b,x1,x2, f1, f2);**

**if f1 < f2**

**a = x1;**

**else**

**b = x2;**

**end**

**end**

**xmin = (a+b)/2;**

**fprintf('Approximate minimum point: x = %8.4f\n', xmin);**

**11.This MATLAB program minimizes the function  
using the Steepest Descent Method.Perform two iterations leading to the minimization using the steepest descent methodwith the starting point .**

**CODE:**

**clc; clear;**

**syms x1 x2**

**f = x1 + 0.5\*x2 + 0.5\*x1\*x2 + 2\*x1 + 2\*x2 + 3;**

**gradf = gradient(f, [x1, x2]);**

**x = [0; 0]; % starting point**

**disp('Iter x1 x2');**

**for k = 1:2**

**g = double(subs(gradf, [x1, x2], x.'));**

**alpha = 0.1; % step size**

**x = x - alpha\*g;**

**fprintf('%2d %.4f %.4f\n', k, x(1), x(2));**

**end**

**12.Let be a sequence that converges to .Show that if there exists such that for sufficiently large ,the order of convergence (if it exists) is at most .**

**clc; clear;**

**x\_true = 2;**

**xk = [1.0, 1.5, 1.75, 1.875, 1.9375, 1.96875]; % example sequence**

**err = abs(xk - x\_true);**

**p = log(err(3:end)./err(2:end-1)) ./ log(err(2:end-1)./err(1:end-2));**

**disp('Estimated order of convergence p:');**

**disp(p);**

**14.Consider the sequence given by   
(a) Write down the value of the limit of .  
(b) Find the order of convergence of .**

**CODE:**

**clc; clear; syms k**

**xk = 2 - 2\*k^2;**

**limit\_val = limit(xk, k, inf);**

**disp(['Limit = ', char(limit\_val)]);**

**disp('Since limit → -∞, sequence diverges. No finite order of convergence.');**

**13.Write MATLAB code for finding the minimizer using the Fibonacci number method.**

**clc; clear;**

**f = @(x) x.^2 + 2\*x + 1; % sample convex function**

**a = -2; b = 2; tol = 0.1;**

**F(1)=1; F(2)=1;**

**for i=3:20**

**F(i)=F(i-1)+F(i-2);**

**end**

**n = find(F>(b-a)/tol,1);**

**for k=1:n-2**

**x1 = a + (F(n-k-1)/F(n-k+1))\*(b-a);**

**x2 = a + (F(n-k)/F(n-k+1))\*(b-a);**

**if f(x1) > f(x2)**

**a=x1;**

**else**

**b=x2;**

**end**

**end**

**xmin=(a+b)/2;**

**fprintf('Fibonacci minimizer at x = %.4f\n', xmin);**

**14.Consider the sequence given by   
(a) Write down the value of the limit of .  
(b) Find the order of convergence of .**

**CODE:**

**clc; clear;**

**syms k**

**xk = 2 - 2\*k^2;**

**limit\_val = limit(xk, k, inf);**

**disp(['Limit = ', char(limit\_val)]);**

**disp('Since limit → -∞, sequence diverges. No finite order of convergence.');**

**15.This MATLAB program minimizes the function using Newton’s Method.**

**CODE:**

**clc; clear;**

**syms x1 x2**

**f = x1 + 0.5\*x2 + 0.5\*x1\*x2 + 2\*x1 + 2\*x2 + 3;**

**gradf = gradient(f, [x1, x2]);**

**H = hessian(f, [x1, x2]);**

**x = [0; 0]; % initial guess**

**disp('Iter x1 x2');**

**for k = 1:3**

**g = double(subs(gradf, [x1, x2], x.'));**

**Hk = double(subs(H, [x1, x2], x.'));**

**x = x - inv(Hk)\*g;**

**fprintf('%2d %.4f %.4f\n', k, x(1), x(2));**

**end**

**16.Write MATLAB code for Newton–Raphson’s Method with examples.**

**syms x**

**f = x^3 - x - 2;**

**df = diff(f);**

**x0 = 1.5; tol = 1e-5; maxIter = 10;**

**disp('Iter x f(x)');**

**for i = 1:maxIter**

**fx = double(subs(f, x, x0));**

**dfx = double(subs(df, x, x0));**

**x1 = x0 - fx/dfx;**

**fprintf('%2d %.6f %.6f\n', i, x1, fx);**

**if abs(x1 - x0) < tol**

**break;**

**end**

**x0 = x1;**

**end**

**fprintf('Root ≈ %.6f\n', x1);**

**17.Use the Steepest Descent Method to find the minimizer of  
.**

**clc; clear;**

**syms x1 x2 x3**

**f = (x1 - 4)^4 + (x2 - 3)^2 + (x3 + 5)^4;**

**gradf = gradient(f, [x1, x2, x3]);**

**x = [0; 0; 0];**

**alpha = 0.01;**

**disp('Iter x1 x2 x3');**

**for k = 1:5**

**g = double(subs(gradf, [x1, x2, x3], x.'));**

**x = x - alpha\*g;**

**fprintf('%2d %.4f %.4f %.4f\n', k, x(1), x(2), x(3));**

**end**

**18.Solve an example using the Steepest Descent Methodfor .**

**clc; clear;**

**syms x1 x2**

**f = x1^2 + x2^2;**

**gradf = gradient(f, [x1, x2]);**

**x = [1; 1];**

**alpha = 0.1;**

**disp('Iter x1 x2');**

**for k = 1:5**

**g = double(subs(gradf, [x1, x2], x.'));**

**x = x - alpha\*g;**

**fprintf('%2d %.4f %.4f\n', k, x(1), x(2));**

**end**

**19.Solve an example using the Steepest Descent Methodfor .**

**clc; clear;**

**Q = [3 1; 1 2];**

**b = [-2; -6];**

**c = 0;**

**f = @(x) 0.5 \* x' \* Q \* x + b' \* x + c;**

**grad = @(x) Q \* x + b;**

**x = [0; 0];**

**alpha = 0.1;**

**disp('Iter x1 x2');**

**for k = 1:5**

**g = grad(x);**

**x = x - alpha \* g;**

**fprintf('%2d %.4f %.4f\n', k, x(1), x(2));**

**end**

**disp('Approximate minimizer:');**

**disp(x);**

**% Q20: Newton's Method for Minimization (Single Variable)**

**clc; clear; close all;**

**f = @(x) x.^4 - 14\*x.^3 + 60\*x.^2 - 70\*x;**

**df = @(x) 4\*x.^3 - 42\*x.^2 + 120\*x - 70; % first derivative**

**d2f = @(x) 12\*x.^2 - 84\*x + 120; % second derivative**

**x0 = 1.5;**

**tol = 1e-6;**

**maxIter = 100;**

**iter = 0;**

**while true**

**x1 = x0 - df(x0)/d2f(x0);**

**iter = iter + 1;**

**if abs(x1 - x0) < tol || iter > maxIter**

**break;**

**end**

**x0 = x1;**

**end**

**fprintf('Minimizer found at x = %.6f after %d iterations\n', x1, iter);**

**fprintf('Minimum value f(x) = %.6f\n', f(x1));**

**% Q21: Newton-Raphson Method for Finding Root**

**clc; clear; close all;**

**f = @(x) x.^4 - 14\*x.^3 + 60\*x.^2 - 70\*x;**

**df = @(x) 4\*x.^3 - 42\*x.^2 + 120\*x - 70;**

**x0 = 2;**

**tol = 1e-6;**

**maxIter = 100;**

**iter = 0;**

**while true**

**x1 = x0 - f(x0)/df(x0);**

**iter = iter + 1;**

**if abs(x1 - x0) < tol || iter > maxIter**

**break;**

**end**

**x0 = x1;**

**end**

**fprintf('Root found at x = %.6f after %d iterations\n', x1, iter);**

**% Q22: Newton's Method for Minimization (Two Variables)**

**clc; clear; close all;**

**f = @(x, y) x.^4 + y.^4 - 4\*x.\*y + x.^2 + y.^2;**

**dfx = @(x, y) 4\*x.^3 - 4\*y + 2\*x;**

**dfy = @(x, y) 4\*y.^3 - 4\*x + 2\*y;**

**H = @(x, y) [12\*x.^2 + 2, -4; -4, 12\*y.^2 + 2];**

**x = 1;**

**y = 1;**

**tol = 1e-6;**

**maxIter = 100;**

**iter = 0;**

**while true**

**g = [dfx(x, y); dfy(x, y)];**

**H\_inv = inv(H(x, y));**

**delta = -H\_inv \* g;**

**x\_new = x + delta(1);**

**y\_new = y + delta(2);**

**iter = iter + 1;**

**if norm([x\_new - x, y\_new - y]) < tol || iter > maxIter**

**break;**

**end**

**x = x\_new;**

**y = y\_new;**

**end**

**fprintf('Minimizer found at (x, y) = (%.6f, %.6f)\n', x\_new, y\_new);**

**fprintf('Minimum value f(x, y) = %.6f\n', f(x\_new, y\_new));**