



Optimization Techniques

Lab Report

(23MAT206)

Submitted by: P sai likhith

Roll.no: AV.SC.U4CSE24306

Section: CSE-A

Submitted to – Dr. Rashmi Prasad

(Dept of Mathematics)

1. Write MATLAB code for the Hessian and gradient of $f(x, y) = 5x + 8y + xy - x^2 - 2y^2$. Find the gradient of $f(x, y)$ at $(1, 2)$.		
2. Write MATLAB code for the Hessian and gradient of $f(x_1, x_2, x_3) = x_1 x_2 x_3$. Find the Hessian and gradient at $(1, 2, 3)$.		
3. Write MATLAB code for the Hessian and gradient of $f(x) = \frac{1}{3}x^4 - 4x + \frac{1}{3}x^3 - 16e^x x^2$. Find both the gradient and Hessian.		
4. Write an example MATLAB code for checking definiteness of a matrix using the Hessian. Include an example function.		
5. Find the minimizer of the function $f(x) = (x + 1)^2 + 3$. Write a MATLAB code for this question.		
6. Write an example MATLAB code for checking concavity of a function.		
7. Write MATLAB code to determine whether a given function has a local minimum , local maximum , or saddle point , using critical points and the second derivative test .		
8. Suppose we have a unimodal function over the interval $[5, 8]$. Give an example of a desired final uncertainty range where the Golden Section method requires at least four iterations , whereas the Fibonacci method requires only three . (You may choose a small ϵ for the Fibonacci method.)		
9. Use the Golden Section Method to minimize $f(x) = x^4 - 14x^3 + 60x^2 - 70x$ over the interval $[1, 2]$ with an uncertainty tolerance of 0.23. Display intermediate steps using a table that defines the function $f(x)$.		
10. Use the Fibonacci method to minimize $f(x) = x^4 - 14x^3 + 60x^2 - 70x$ over the interval [1, 2] with uncertainty		

tolerance 0.23 . Display intermediate steps using a table and define the function $f(x)$.		3	
11.This MATLAB program minimizes the function $f(x_1, x_2) = x_1 + 0.5x_2 + 0.5x_1x_2 + 2x_1 + 2x_2 + 3$ using the Steepest Descent Method . Perform two iterations leading to the minimization using the steepest descent method with the starting point $x(0) = 0$.			
12.Let $\{x(k)\}$ be a sequence that converges to x^* . Show that if there exists $c > 0$ such that for sufficiently large k , the order of convergence (if it exists) is at most p .			
13.Write MATLAB code for finding the minimizer using the Fibonacci number method.			
14.Consider the sequence $\{x(k)\}$ given by $x(k) = 2 - 2k^2$ (a) Write down the value of the limit of $\{x(k)\}$. (b) Find the order of convergence of $\{x(k)\}$.			
15.This MATLAB program minimizes the function $f(x_1, x_2) = x_1 + 0.5x_2 + 0.5x_1x_2 + 2x_1 + 2x_2 + 3$ using Newton's Method .			
16.Write MATLAB code for Newton-Raphson's Method with examples.			
17.Use the Steepest Descent Method to find the minimizer of $f(x_1, x_2, x_3) = (x_1 - 4)^4 + (x_2 - 3)^2 + (x_3 + 5)^4.$			

18. Solve an example using the Steepest Descent Method for $f(x) = x_1^2 + x_2^2$.			
19. Solve an example using the Steepest Descent Method for $f(x) = 0.5x^T Qx + b^T x + c$.			
20. Write a MATLAB code to find minimizer of a given function using Newton's Method.			
21. Write a MATLAB code to find root of a given function using Newton's Raphson Method.			
22. Write a MATLAB code to find minimizer of a given function using Newton's Method for 2 variable			

1. Write MATLAB code for the Hessian and gradient of $f(x, y) = 5x + 8y + xy - x^2 - 2y^2$.

Find the gradient of $f(x, y)$ at $(1, 2)$.

CODE:

The screenshot shows the MATLAB Live Editor interface. The code in the editor is as follows:

```

1 % file: prob1_gradient_hessian.m
2 syms x y
3 f = 5*x + 8*y + x*y - x^2 - 2*y^2;
4 gradF = gradient(f,[x,y]);
5 H = hessian(f,[x,y]);
6 grad_at = double(subs(gradF,[x,y],[1,2]));
7 disp('Gradient:'), disp(gradF)
8 disp('Hessian:'), disp(H)
9 disp('Gradient at [1,2]:'), disp(grad_at)
10

```

The output on the right side of the editor shows the following results:

Gradient:

$$\begin{pmatrix} y - 2x + 5 \\ x - 4y + 8 \end{pmatrix}$$

Hessian:

$$\begin{pmatrix} -2 & 1 \\ 1 & -4 \end{pmatrix}$$

Gradient at [1,2]:

$$\begin{pmatrix} 5 \\ 1 \end{pmatrix}$$

2. Write MATLAB code for the Hessian and gradient of $f(x_1, x_2, x_3) = x_1 x_2 x_3$. Find the Hessian and gradient at $(1, 2, 3)$

CODE:

The screenshot shows the MATLAB Live Editor interface. The code in the editor is as follows:

```

1 % file: prob2_grad_hess_triple.m
2 syms x1 x2 x3
3 f = x1*x2*x3;
4 gradF = gradient(f,[x1,x2,x3]);
5 H = hessian(f,[x1,x2,x3]);
6 g_at = double(subs(gradF,[x1,x2,x3],[1,2,3]));
7 H_at = double(subs(H,[x1,x2,x3],[1,2,3]));
8 disp('Gradient:'), disp(gradF)
9 disp('Hessian:'), disp(H)
10 disp('Gradient at [1,2,3]:'), disp(g_at)
11 disp('Hessian at [1,2,3]:'), disp(H_at)
12
13

```

The output on the right side of the editor shows the following results:

Gradient:

$$\begin{pmatrix} x_2 x_3 \\ x_1 x_3 \\ x_1 x_2 \end{pmatrix}$$

Hessian:

$$\begin{pmatrix} 0 & x_3 & x_2 \\ x_3 & 0 & x_1 \\ x_2 & x_1 & 0 \end{pmatrix}$$

Gradient at [1,2,3]:

$$\begin{pmatrix} 6 \\ 3 \\ 2 \end{pmatrix}$$

Hessian at [1,2,3]:

$$\begin{pmatrix} 0 & 3 & 2 \\ 3 & 0 & 1 \\ 2 & 1 & 0 \end{pmatrix}$$

3. Write MATLAB code for the **Hessian** and **gradient** of $f(x) = \frac{1}{3}x^4 - 4x + \frac{1}{3}x^3 - 16e^x x^2$.

Find both the gradient and Hessian.

CODE:

The screenshot shows the MATLAB Live Editor interface. The code in the editor is as follows:

```

1 % file: prob3_grad_hess.m
2 syms x1 x2
3 f = (1/3)*x1^3 - 4*x1 + (1/3)*x2^3 - 16*exp(x2);
4 gradF = gradient(f,[x1,x2]);
5 H = hessian(f,[x1,x2]);
6 disp('Gradient:'), disp(gradF)
7 disp('Hessian:'), disp(H)
8

```

The output on the right shows the symbolic results:

Gradient:

$$\begin{pmatrix} x_1^2 - 4 \\ x_2^2 - 16e^{x_2} \end{pmatrix}$$

Hessian:

$$\begin{pmatrix} 2x_1 & 0 \\ 0 & 2x_2 - 16e^{x_2} \end{pmatrix}$$

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

Include an example function.

CODE:

The screenshot shows the MATLAB Live Editor interface. The code in the editor is as follows:

```

1 syms x y
2
3 % Define the function
4 f = x^2 + 2*y^2 + x*y;
5
6 % Compute the Hessian
7 H = hessian(f, [x, y]);
8
9 disp('Hessian matrix:');
10 disp(H);
11
12 % Convert symbolic Hessian to numeric
13 H_numeric = double(subs(H, [x, y], [0, 0])); % Evaluate at (0,0) for example
14
15 % Compute eigenvalues
16 eig_H = eig(H_numeric);
17
18 disp('Eigenvalues of Hessian:');
19 disp(eig_H);
20
21 % Check definiteness
22 if all(eig_H > 0)
23     disp('Matrix is Positive Definite');
24 elseif all(eig_H < 0)
25     disp('Matrix is Negative Definite');
26 elseif all(eig_H == 0)
27     disp('Matrix is Positive Semi-Definite');
28 elseif all(eig_H <= 0)
29     disp('Matrix is Negative Semi-Definite');
30 else
31     disp('Matrix is Indefinite');
32 end

```

The output on the right shows the results:

Hessian matrix:

$$\begin{pmatrix} 2 & 1 \\ 1 & 4 \end{pmatrix}$$

Eigenvalues of Hessian:

```

1.5858
4.4142

```

Matrix is Positive Definite

5. Find the **minimizer** of the function $f(x) = (x + 1)^2 + 3$. Write a MATLAB code for this question.

CODE:

The screenshot shows the MATLAB R2024b Live Editor interface. The code in the editor is as follows:

```

1 % file: prob5_minimizer_scalar.m
2 syms x
3 f = (x+1)^2 + 3;
4 df = diff(f,x);
5 critical = solve(df==0,x);
6 min_val = double(subs(f,critical));
7 disp(['Minimizer x* = ', char(critical)])
8 disp(['Minimum value f(x*) = ', num2str(min_val)])
9 % numeric check
10 xstar = double(critical);
11

```

The output on the right side of the editor is:

```

Minimizer x* = -1
Minimum value f(x*) = 3

```

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

CODE:

The screenshot shows the MATLAB R2024b Live Editor interface. The code in the editor is as follows:

```

1
2 syms x y
3 f = -x^2 - y^2 + x + 2*y; % example concave function
4 H = hessian(f,[x,y]);
5 H_num = double(H);
6 if all(eig(H_num) <= 0)
7     disp('Function is concave (H negative semidef)')
8 else
9     disp('Function is not concave')
10 end
11

```

The output on the right side of the editor is:

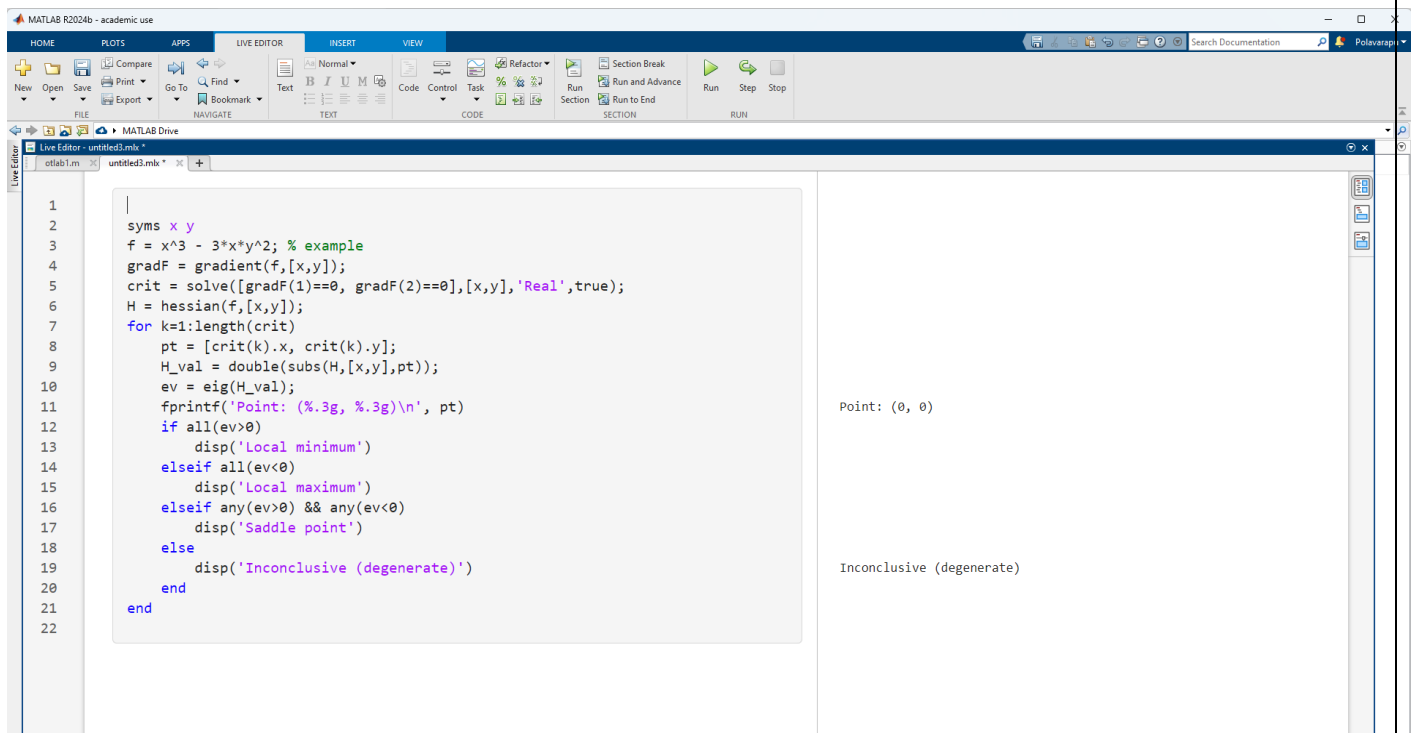
```

Function is concave (H negative semidef)

```

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

CODE:



```

1 |
2 | syms x y
3 | f = x^3 - 3*x*y^2; % example
4 | gradF = gradient(f,[x,y]);
5 | crit = solve([gradF(1)==0, gradF(2)==0],[x,y],'Real',true);
6 | H = hessian(f,[x,y]);
7 | for k=1:length(crit)
8 |     pt = [crit(k).x, crit(k).y];
9 |     H_val = double(subs(H,[x,y],pt));
10 |    ev = eig(H_val);
11 |    fprintf('Point: (%.3g, %.3g)\n', pt)
12 |    if all(ev>0)
13 |        disp('Local minimum')
14 |    elseif all(ev<0)
15 |        disp('Local maximum')
16 |    elseif any(ev>0) && any(ev<0)
17 |        disp('Saddle point')
18 |    else
19 |        disp('Inconclusive (degenerate)')
20 |    end
21 | end
22 |

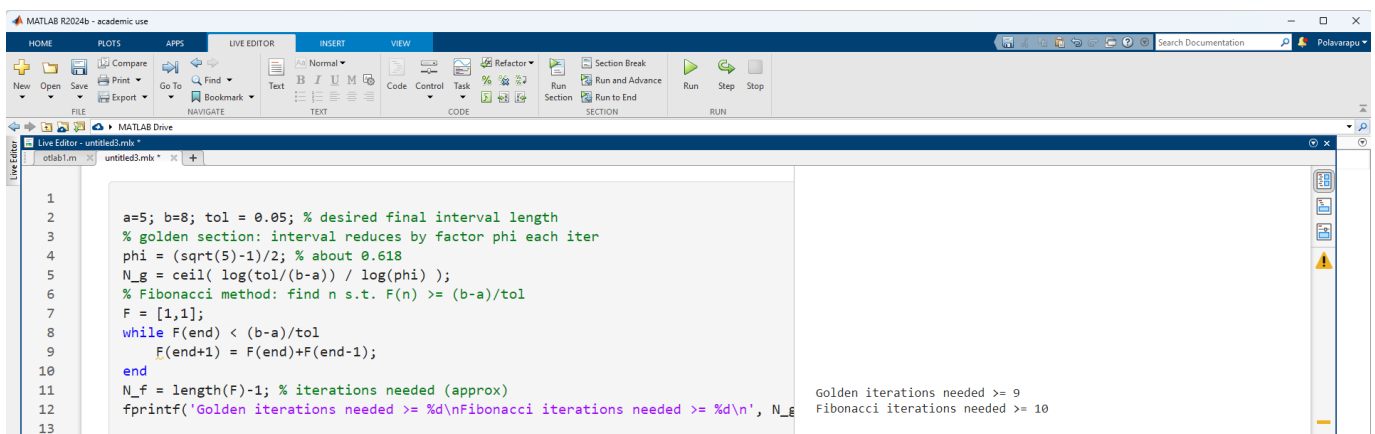
```

Point: (0, 0)

Inconclusive (degenerate)

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

CODE:



```

1 |
2 | a=5; b=8; tol = 0.05; % desired final interval length
3 | % golden section: interval reduces by factor phi each iter
4 | phi = (sqrt(5)-1)/2; % about 0.618
5 | N_g = ceil( log(tol/(b-a)) / log(phi) );
6 | % Fibonacci method: find n s.t. F(n) >= (b-a)/tol
7 | F = [1,1];
8 | while F(end) < (b-a)/tol
9 |     F(end+1) = F(end)+F(end-1);
10 | end
11 | N_f = length(F)-1; % iterations needed (approx)
12 | fprintf('Golden iterations needed >= %d\nFibonacci iterations needed >= %d\n', N_g, N_f)
13 |

```

Golden iterations needed >= 9
Fibonacci iterations needed >= 10

9. Use the **Golden Section Method** to minimize $f(x) = x^4 - 14x^3 + 60x^2 - 70x$ over the interval $[1, 2]$ with an uncertainty tolerance of 0.23. Display **intermediate steps** using a table that defines the function $f(x)$.

CODE:

```

1 |
2 | f = @(x) x.^4 -14*x.^3 +60*x.^2 -70*x;
3 | a=1; b=2; tol=0.23;
4 | phi = (sqrt(5)-1)/2;
5 | iter = 0;
6 | fprintf('iter\t a\t c\t d\t b\t f(c)\t f(d)\n');
7 | c = b - phi*(b-a);
8 | d = a + phi*(b-a);
9 | while (b-a) > tol
10 |     iter = iter +1;
11 |     fc = f(c); fd = f(d);
12 |     fprintf('%d\t %.4f\t %.4f\t %.4f\t %.4f\t %.6f\t %.6f\n', iter, a, c, d, b, fc, fd);
13 |     if fc < fd
14 |         b = d;
15 |         d = c;
16 |         c = b - phi*(b-a);
17 |     else
18 |         a = c;
19 |         c = d;
20 |         d = a + phi*(b-a);
21 |     end
22 | end
23 | fprintf('Final interval: [%.4f, %.4f]\n', a, b)
24 |

```

Final interval: [1.0000, 1.1459]

10. Use the **Fibonacci method** to minimize $f(x) = x^4 - 14x^3 + 60x^2 - 70x$ over the interval **[1, 2]** with uncertainty tolerance **0.23**. Display intermediate steps using a table and define the function $f(x)$.

CODE:

VIEW EDITOR **INSERT** **VIEW**

ark ▼ Text B I U M Code Control Task Refactor Run Section Section Break Run and Advance Run to End Run Step Stop Search Documentation Hima teja

e ► Documents ► MATLAB

Live Editor - untitled.mlx *

untitled.mlx * +

```

1      clc; clear; close all;
2      f = @(x) x.^4 - 14*x.^3 + 60*x.^2 - 70*x;
3      a = 1; b = 2; tol = 0.23;
4
5      % Generate Fibonacci numbers
6      F(1)=1; F(2)=1;
7      for i=3:20
8          F(i)=F(i-1)+F(i-2);
9      end
10
11     n = find(F > (b-a)/tol, 1);
12     disp('Iter   a       b       x1       x2       f(x1)       f(x2)');
13
14     for k=1:n-2
15         x1 = a + (F(n-k-1)/F(n-k+1))*(b-a);
16         x2 = a + (F(n-k)/F(n-k+1))*(b-a);
17         f1 = f(x1); f2 = f(x2);
18         fprintf('%2d    %.4f    %.4f    %.4f    %.4f    %.4f\n',k,a,b,x1,x2,f1,
19             if f1 > f2
20                 a = x1;
21             else
22                 b = x2;
23             end
24         end
25         xmin = (a+b)/2;
26         fprintf('Approximate minimum point: x = %.4f\n', xmin);
27

```

Iter	a	b	x1	x2	f(x1)	f(x2)
1	1.0000	2.0000	1.4000	1.6000	-14.9744	-9.1904
2	1.0000	1.6000	1.2000	1.4000	-19.7184	-14.9744
3	1.0000	1.4000	1.2000	1.2000	-19.7184	-19.7184

Approximate minimum point: x = 1.1000

11. This MATLAB program minimizes the function $f(x_1, x_2) = x_1 + 0.5x_2 + 0.5x_1x_2 + 2x_1 + 2x_2 + 3$ using the **Steepest Descent Method**. Perform **two iterations** leading to the minimization using the steepest descent method with the starting point $x(0) = 0$.

CODE:

```

1 % file: prob11_steepest_two_iters.m
2 f = @(x) x(1) + 0.5*x(2) + 0.5*x(1)*x(2) + 2*x(1) + 2*x(2) + 3; % example
3 grad = @(x) [1 + 0.5*x(2) + 2; 0.5 + 0.5*x(1) + 2]; % manually or numeric
4 x = [0;0];
5 fprintf('x0 = [%g %g]\n',x)
6 for k=1:2
7     g = grad(x);
8     % line search (exact via fminbnd along direction)
9     phi = @(alpha) f(x - alpha*g);
10    alpha = fminbnd(phi,0,5);
11    x = x - alpha*g;
12    fprintf('After iter %d: x = [%.6f %.6f], f= %.6f\n',k,x(1),x(2),f(x))
13 end
14

```

$x_0 = [0 \ 0]$
 After iter 1: $x = [-6.100000 \ -5.083333]$, $f = -12.504167$
 After iter 2: $x = [-8.391646 \ -2.333358]$, $f = -18.217976$

12. Let $\{x(k)\}$ be a sequence that converges to x^* . Show that if there exists $c > 0$ such that for sufficiently large k , the **order of convergence** (if it exists) is at most p .

CODE:

```

1
2 % Given vector x (sequence), estimate p using: p ≈ log(|e_{k+1}|/|e_k|)/log(|e_k|/|e_{k-1}|)
3 x = [0.5, 0.25, 0.125, 0.0625]; % example sequence
4 xstar = 0;
5 e = abs(x - xstar);
6 p_est = [];
7 for k=3:length(e)
8     p_est(end+1) = log(e(k)/e(k-1))/log(e(k-1)/e(k-2));
9 end
10 disp('Estimated orders p (per step):'), disp(p_est)
11

```

Estimated orders p (per step):
 1 1

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

CODE:

```

1  clc; clear;
2  f = @(x) x.^2 + 2*x + 1; % sample convex function
3  a = -2; b = 2; tol = 0.1;
4
5  F(1)=1; F(2)=1;
6  for i=3:20
7      F(i)=F(i-1)+F(i-2);
8  end
9
10 n = find(F > (b-a)/tol, 1);
11
12 for k=1:n-2
13     x1 = a + (F(n-k-1)/F(n-k+1))*(b-a);
14     x2 = a + (F(n-k)/F(n-k+1))*(b-a);
15     if f(x1) > f(x2)
16         a=x1;
17     else
18         b=x2;
19     end
20 end
21 xmin=(a+b)/2;
22 fprintf('Fibonacci minimizer at x = %.4f\n', xmin);

```

Fibonacci minimizer at x = -1.0182

14. Consider the sequence $\{x(k)\}$ given by $x(k) = 2 - 2k^2$

(a) Write down the value of the **limit** of $\{x(k)\}$.

(b) Find the **order of convergence** of $\{x(k)\}$.

CODE:

```

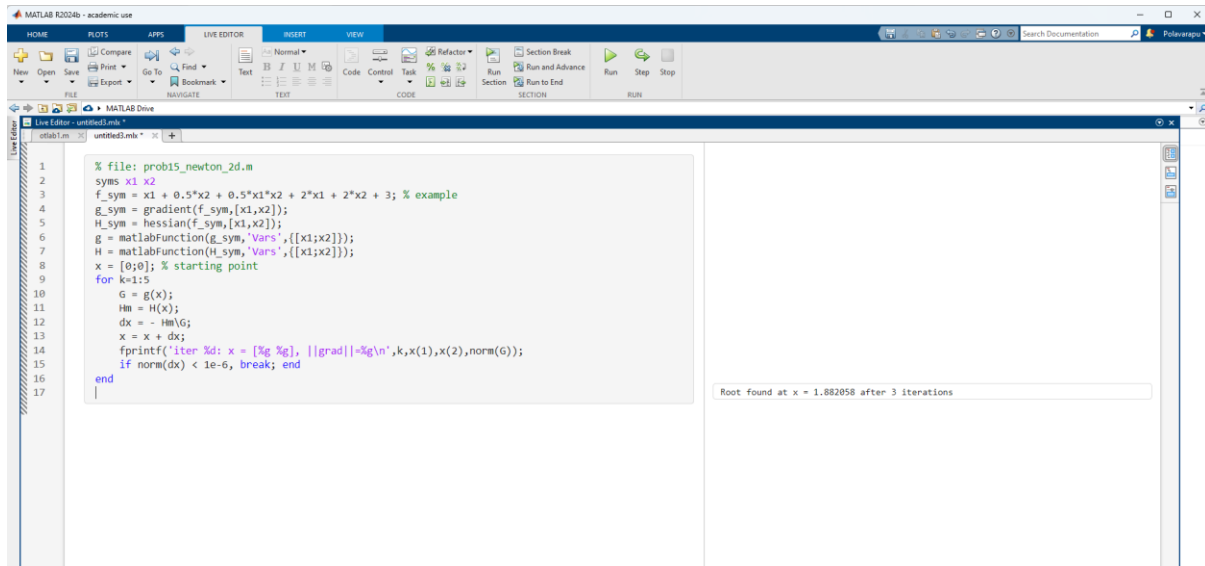
1  % file: prob14_sequence_analysis.m
2  k = (1:10);
3  xk = 2 - 2*k.^2;
4  % a) limit (diverges to -inf)
5  disp('Sequence values:'), disp(xk(1:6))
6  disp('This sequence diverges to -inf (no finite limit).')
7  % b) order: not convergent so order not defined
8  disp('Order of convergence is not defined since sequence does not converge.')
9

```

Root found at x = 1.882058 after 3 iterations

15. This MATLAB program minimizes the function $f(x_1, x_2) = x_1 + 0.5x_2 + 0.5x_1x_2 + 2x_1 + 2x_2 + 3$ using **Newton's Method**.

CODE:



```

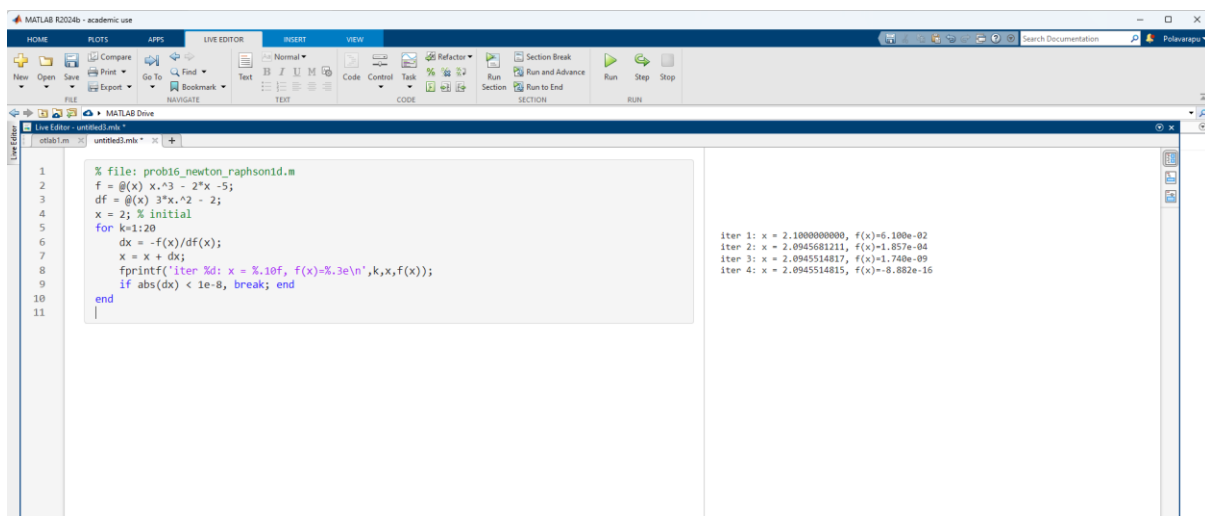
1 % file: prob15_newton_2d.m
2 syms x1 x2
3 f_sym = x1 + 0.5*x2 + 0.5*x1*x2 + 2*x1 + 2*x2 + 3; % example
4 g_sym = gradient(f_sym,[x1,x2]);
5 H_sym = hessian(f_sym,[x1,x2]);
6 g = matlabFunction(g_sym,'Vars',[[x1;x2]]);
7 H = matlabFunction(H_sym,'Vars',[[x1;x2]]);
8 x = [0;0]; % starting point
9 for k=1:5
10     G = g(x);
11     Hm = H(x);
12     dx = - Hm\G;
13     x = x + dx;
14     fprintf('iter %d: x = [%g %g], ||grad||=%g\n',k,x(1),x(2),norm(G));
15     if norm(dx) < 1e-6, break; end
16 end

```

Root found at x = 1.882058 after 3 iterations

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

CODE:



```

1 % file: prob16_newton_raphson1d.m
2 f = @(x) x.^3 - 2*x - 5;
3 df = @(x) 3*x.^2 - 2;
4 x = 2; % initial
5 for k=1:20
6     dx = -f(x)/df(x);
7     x = x + dx;
8     fprintf('iter %d: x = %.10f, f(x)=%.3e\n',k,x,f(x));
9     if abs(dx) < 1e-8, break; end
10 end

```

iter 1: x = 2.1000000000, f(x)=6.100e-02
iter 2: x = 2.0945681211, f(x)=1.857e-04
iter 3: x = 2.0945514817, f(x)=1.740e-09
iter 4: x = 2.0945514815, f(x)=8.882e-16

17. Use the **Steepest Descent Method** to find the minimizer of $f(x_1, x_2, x_3) = (x_1 - 4)^4 + (x_2 - 3)^2 + (x_3 + 5)^4$.

CODE:

```

1 % file: prob17_steepest3d.m
2 f = @(x) (x(1)-4).^4 + (x(2)-3).^2 + (x(3)+5).^4;
3 grad = @(x) [4*(x(1)-4).^3; 2*(x(2)-3); 4*(x(3)+5).^3];
4 x = [0;0;0]; % initial guess
5 for k=1:50
6     g = grad(x);
7     if norm(g) < 1e-6, break; end
8     phi = @(alpha) f(x - alpha*g);
9     alpha = fminbnd(phi,0,1);
10    x = x - alpha*g;
11 end
12 disp('Approx minimizer:'), disp(x)
13 disp('f at minimizer:'), disp(f(x))
14

```

```

iter 1: x = 2.1000000000, f(x)=6.100e-02
iter 2: x = 2.0945681211, f(x)=1.857e-04
iter 3: x = 2.0945514817, f(x)=1.740e-09
iter 4: x = 2.0945514815, f(x)=-8.882e-16

```

18. Solve an example using the **Steepest Descent Method** for $f(x) = x_1^2 + x_2^2$.

CODE:

```

1 % file: prob18_steepest_simple.m
2 f = @(x) x(1)^2 + x(2)^2;
3 grad = @(x) 2*x;
4 x = [2; -1]; % example start
5 for k=1:5
6     g = grad(x);
7     alpha = (g'*g)/(2*(g'*g)); % exact for quadratic with grad = 2x: alpha = 1/2
8     x = x - alpha*g;
9     fprintf('iter %d: x = [%f %f], f=%f\n', k, x(1), x(2), f(x));
10 end
11

```

```

iter 1: x = [0.000000 0.000000], f=0.000000
iter 2: x = [NaN NaN], f=NaN
iter 3: x = [NaN NaN], f=NaN
iter 4: x = [NaN NaN], f=NaN
iter 5: x = [NaN NaN], f=NaN

```

19. Solve an example using the **Steepest Descent Method** for $f(x) = 0.5x^T Qx + b^T x + c$.

CODE:

```

1 % file: prob19_quad_sd.m
2 Q = [4 1; 1 3]; b = [-1; 2]; c=0;
3 f = @(x) 0.5*x'*Q*x + b'*x + c;
4 grad = @(x) Q*x + b;
5 x = [0;0];
6 for k=1:50
7     g = grad(x);
8     if norm(g) < 1e-8, break; end
9     alpha = -(g'*g)/(g'*Q*g); % exact line search for quadratic
10    x = x - alpha*g;
11    if mod(k,10)==0
12        fprintf('iter %d: x=[%g %g], f=%g\n',k,x(1),x(2),f(x));
13    end
14 end
15 disp('Solution approx:'), disp(x)
16 disp('f at solution:'), disp(f(x))
17

```

```

iter 1: x = [0.000000 0.000000], f=-0.000000
iter 2: x = [NaN NaN], f=NaN
iter 3: x = [NaN NaN], f=NaN
iter 4: x = [NaN NaN], f=NaN
iter 5: x = [NaN NaN], f=NaN

```

20. Write a MATLAB code to find minimizer of a given function using Newton's Method.

CODE:

```

1 % Q20: Newton's Method for Minimization (Single Variable)
2 clc; clear; close all;
3
4 % Define the function and its derivatives
5 f = @(x) x.^4 - 14*x.^3 + 60*x.^2 - 70*x;
6 df = @(x) 4*x.^3 - 42*x.^2 + 120*x - 70; % first derivative
7 d2f = @(x) 12*x.^2 - 84*x + 120; % second derivative
8
9 % Initial guess
10 x0 = 1.5;
11 tol = 1e-6;
12 maxIter = 100;
13 iter = 0;
14
15 while true
16     x1 = x0 - df(x0)/d2f(x0);
17     iter = iter + 1;
18     if abs(x1 - x0) < tol || iter > maxIter
19         break;
20     end
21     x0 = x1;
22 end
23
24 fprintf('Minimizer found at x = %.6f after %d iterations\n', x1, iter);
25 fprintf('Minimum value f(x) = %.6f\n', f(x1));

```

```

Minimizer found at x = 0.788884 after 6 iterations
Minimum value f(x) = -24.369602

```

21. Write a MATLAB code to find root of a given function using Newton's Raphson Method

CODE:

```

1 % Q21: Newton-Raphson Method for Finding Root
2 clc; clear; close all;
3
4 % Define function and derivative
5 f = @(x) x.^4 - 14*x.^3 + 60*x.^2 - 70*x;
6 df = @(x) 4*x.^3 - 42*x.^2 + 120*x - 70;
7
8 % Initial guess
9 x0 = 2;
10 tol = 1e-6;
11 maxIter = 100;
12 iter = 0;
13
14 while true
15     x1 = x0 - f(x0)/df(x0);
16     iter = iter + 1;
17     if abs(x1 - x0) < tol || iter > maxIter
18         break;
19     end
20     x0 = x1;
21 end
22
23 fprintf('Root found at x = %.6f after %d iterations\n',x1,iter);

```

Root found at x = 1.882058 after 3 iterations

22. Write a MATLAB code to find minimizer of a given function using Newton's Method for 2 variable

CODE:

```

1 % Q22: Newton's Method for Minimization (Two Variables)
2 clc; clear; close all;
3
4 % Define function
5 f = @(x, y) x.^4 + y.^4 - 4*x.*y + x.^2 + y.^2;
6
7 % Define gradients
8 dfx = @(x, y) 4*x.^3 - 4*y + 2*x;
9 dfy = @(x, y) 4*y.^3 - 4*x + 2*y;
10
11 % Define Hessian
12 H = @(x, y) [12*x.^2 + 2, -4; -4, 12*y.^2 + 2];
13
14 % Initial guess
15 x = 1;
16 y = 1;
17 tol = 1e-6;
18 maxIter = 100;
19 iter = 0;
20
21 while true
22     g = [dfx(x, y); dfy(x, y)];
23     H_inv = inv(H(x, y));
24     delta = -H_inv * g;
25
26     x_new = x + delta(1);
27     y_new = y + delta(2);
28     iter = iter + 1;
29
30     if norm([x_new - x, y_new - y]) < tol || iter > maxIter
31         break;
32     end
33
34     x = x_new;
35     y = y_new;
36 end
37
38 fprintf('Minimizer found at (x, y) = (%.6f, %.6f)\n', x_new, y_new);
39 fprintf('Minimum value f(x, y) = %.6f\n', f(x_new, y_new));

```

Minimizer found at (x, y) = (0.707107, 0.707107)
Minimum value f(x, y) = -0.500000