

**Optimization Techniques Lab Report**

**Submitted to – Dr. Rashmi Prasad**

**(Dept of Mathematics)**

**Submitted by- P.MANOJ**

**24241**

**CSE-C**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.no** | **Question** | **Date** | **Signature** |
| 1. | Matlab code for hessian and gradient of f(x,y)=5x+8y+xy-x^2-2y^2 find gradient of f (x,y) at [1,2] |  |  |
| 2. | Matlab code for hessian and gradient of f(x1,x2,x3)= x1x2x2 find hessian and gradient at [1,2,3] |  |  |
| 3. | Matlab code for hessian and gradient of f(1/3 x1^3-4x1+1/3 x2^3- 16e^x2 find gradient and hessian |  |  |
| 4. | To check the given matrix is definite or indefinite. |  |  |
| 5. | **Find gradient and hessian of the given function f at[1,1] 3\*x1+5\*x2+x1\*(x1+4\*x2)+x2\*(2\*x1+7\*x2)+6**  **give matlab code** |  |  |
| 6. | Consider the following function. f(x)=x⊤[14​27​]x+x⊤[35​]+6  a. Find the gradient and Hessian of f at the point [1,1].  b. Find the directional derivative of f at [1,1] with respect to a unit vector in the direction of maximal rate of increase.  c. Find a point that satisfies the FONC (interior case) for f. Does this point satisfy the SONC (for a minimizer)? |  |  |
| 7. | Consider the following function: *f*(*x*1,*x*2)=*x*12*x*2+*x*23*x*1  a. In what direction does the function f decrease most rapidly at the point x(0) = [2,1] ?  b. What is the rate of increase of f at the point x(0) in the direction of maximum decrease of f?  c. Find the rate of increase of f at the point x(0) in the direction d = [3,4] . |  |  |
| 8. | Consider the following function f :  *f*(*x*)=*xT*[2−151]*x*+*xT*[34]+7  a. Find the directional derivative of f at [0,1] in the direction [1,0] .  b. Find all points that satisfy the first-order necessary condition for f. Does f have a minimizer? If it does, then find all minimizer(s); otherwise, explain why it does not. |  |  |
| 9. | Given f=x12+x22 find critical points and check concavity. |  |  |
| 10. | Given f=(x+1)2+3 find critical points and check whether it is local maximizer or minimizer. |  |  |
| 11. | Use Golden Section Search to minimize  f(x) = x4 - 14x3 + 60x2 - 70x at interval [0,2] with accuracy 0.3. |  |  |
| 12. | Perform two iterations leading to the minimization of f(x1,x2)=x1+(1/2)x2+(1/2)x12+x22+3 using the steepest descent method with the starting point x(0)= 0. Also determine an optimal solution analytically. |  |  |
| 13. | Use the method of steepest descent to find the minimizer of *f*(*x*1,*x*2,*x*3)=(*x*1−4)4+(*x*2−3)2+4(*x*3+5)4 |  |  |
| 14. | F(x1,x2)=x1^2+x2^2 Let Then, starting from an arbitrary initial point x(0). |  |  |
| 15. | Let the function f be given by  *f*(*x*)=*xT*[40225]*x*+*xT*[36]+24  Find the minimizer of f using a fixed-step-size gradient algorithm where α is a fixed step size. |  |  |
| 16. | Suppose that 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 an arbitrarily small value of ε for the Fibonacci method. |  |  |
| 17. | Suppose that we wish to use the golden section search method to find the value of x  *f*(*x*)=*x*4−14*x*3+60*x*2−70*x* that minimizes in the interval [0,2] of x within a range of 0.3. |  |  |
| 18. | Use the golden section search method to find the value of x that minimizes in the interval [1,2] of x within a range of 0.23. |  |  |
| 19. | Use the Fibonacci method to find the value of x that minimizes in the interval [1,2] of x within a range of 0.23. |  |  |
| 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 Rapson Method. |  |  |
| 22. | Write a matlab code to find minimizer of a given function using Newton’s Method for 2 variable |  |  |

1. Find the hessian and gradient of the following:-

f(x,y)=5x+8y+xy-x^2-2y^2 find gradient of f(x,y) at[1,2]

CODE:-

syms x y

% Define the function

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

% Compute gradient

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

% Compute Hessian

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

% Evaluate gradient at [1, 2]

grad\_at\_point = subs(grad\_f, [x, y], [1, 2]);

% Display results

disp('Gradient:');

disp(grad\_f);

disp('Hessian:');

disp(hess\_f);

disp('Gradient at [1,2]:');

disp(grad\_at\_point);

OUTPUT:-



2. Matlab code for hessian and gradient of f(x1,x2,x3)= x1x2x2 find hessian and gradient at [1,2,3]

CODE:

syms x1 x2 x3 % Define symbolic variables

% Define the function

f = x1\*x2\*x3;

% Compute the gradient

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

% Compute the Hessian

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

% Evaluate gradient at [1,2,3]

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

% Evaluate Hessian at [1,2,3]

hess\_at\_point = subs(hess\_f, [x1 x2 x3], [1 2 3]);

% Display the results

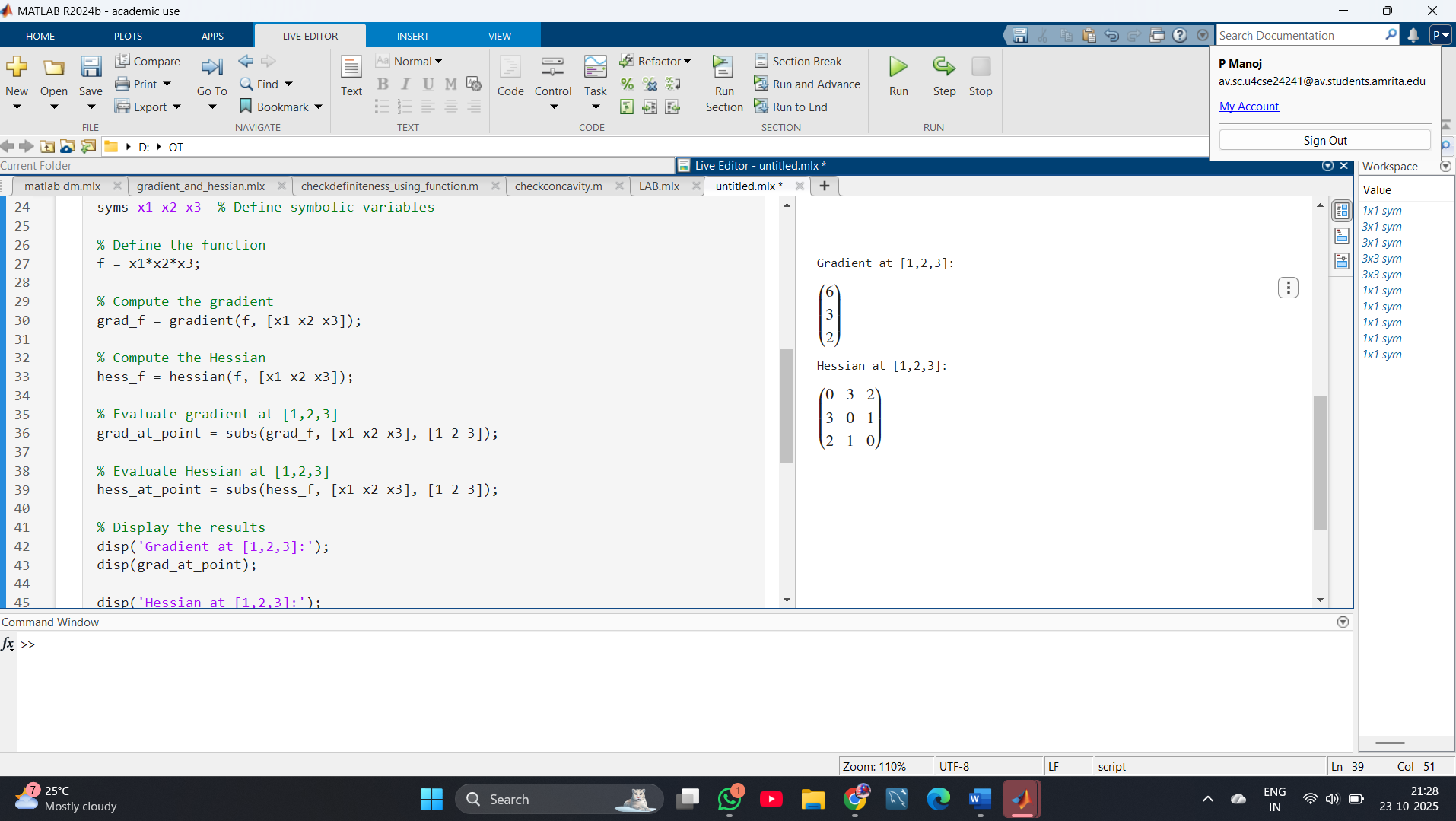
disp('Gradient at [1,2,3]:');

disp(grad\_at\_point);

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

disp(hess\_at\_point);

OUTPUT:



3. Matlab code for hessian and gradient of f(1/3 x1^3-4x1+1/3 x2^3- 16e^x2 find gradient and hessian

CODE:

syms x1 x2 % Define symbolic variables

% Define the function

f = (1/3)\*x1^3 - 4\*x1 + (1/3)\*x2^3 - 16\*exp(x2);

% Compute the gradient

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

% Compute the Hessian

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

% Display the symbolic gradient and Hessian

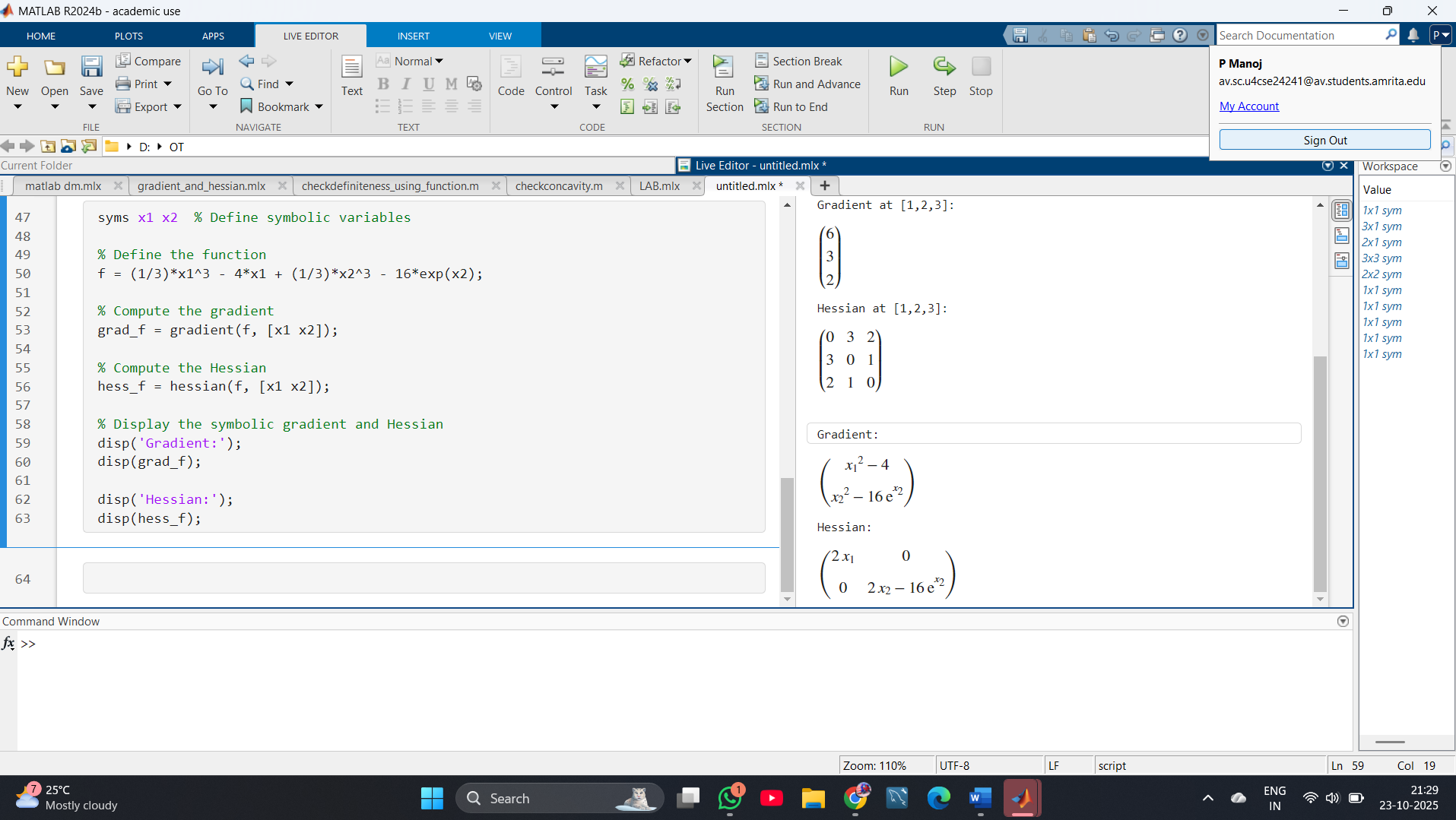
disp('Gradient:');

disp(grad\_f);

disp('Hessian:');

disp(hess\_f);

OUTPUT:



4. To check the given matrix is definite or indefinite.

CODE:

clc; clear;

% Example matrix (symmetric)

A = [2 -1 0; -1 2 -1; 0 -1 2];

% Compute eigenvalues

eig\_vals = eig(A);

disp('Eigenvalues of the matrix:');

disp(eig\_vals);

% Check definiteness

if all(eig\_vals > 0)

disp('Matrix is Positive Definite');

elseif all(eig\_vals < 0)

disp('Matrix is Negative Definite');

elseif all(eig\_vals >= 0)

disp('Matrix is Positive Semi-Definite');

elseif all(eig\_vals <= 0)

disp('Matrix is Negative Semi-Definite');

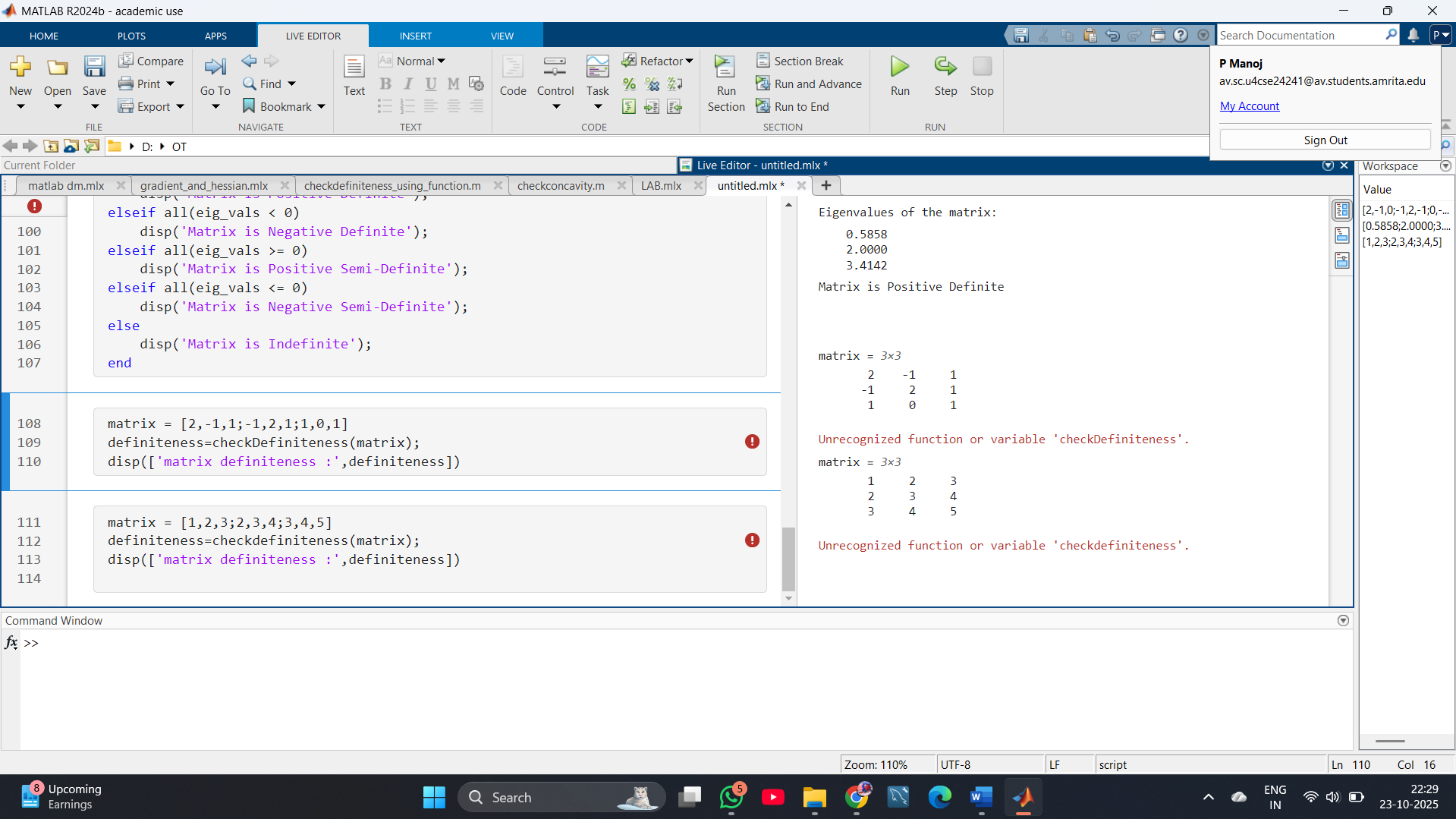
else

disp('Matrix is Indefinite');

end

0UTPUT:





5. Find gradient and hessian of the given function f at[1,1] 3\*x1+5\*x2+x1\*(x1+4\*x2)+x2\*(2\*x1+7\*x2)+6.

CODE:

clc; clear;

syms x1 x2

% Define the function

f = 3\*x1 + 5\*x2 + x1\*(x1 + 4\*x2) + x2\*(2\*x1 + 7\*x2) + 6;

% Compute gradient

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

% Compute Hessian

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

% Evaluate gradient at [1,1]

grad\_at\_point = subs(grad\_f, [x1 x2], [1 1]);

% Evaluate Hessian at [1,1]

hess\_at\_point = subs(hess\_f, [x1 x2], [1 1]);

% Display results

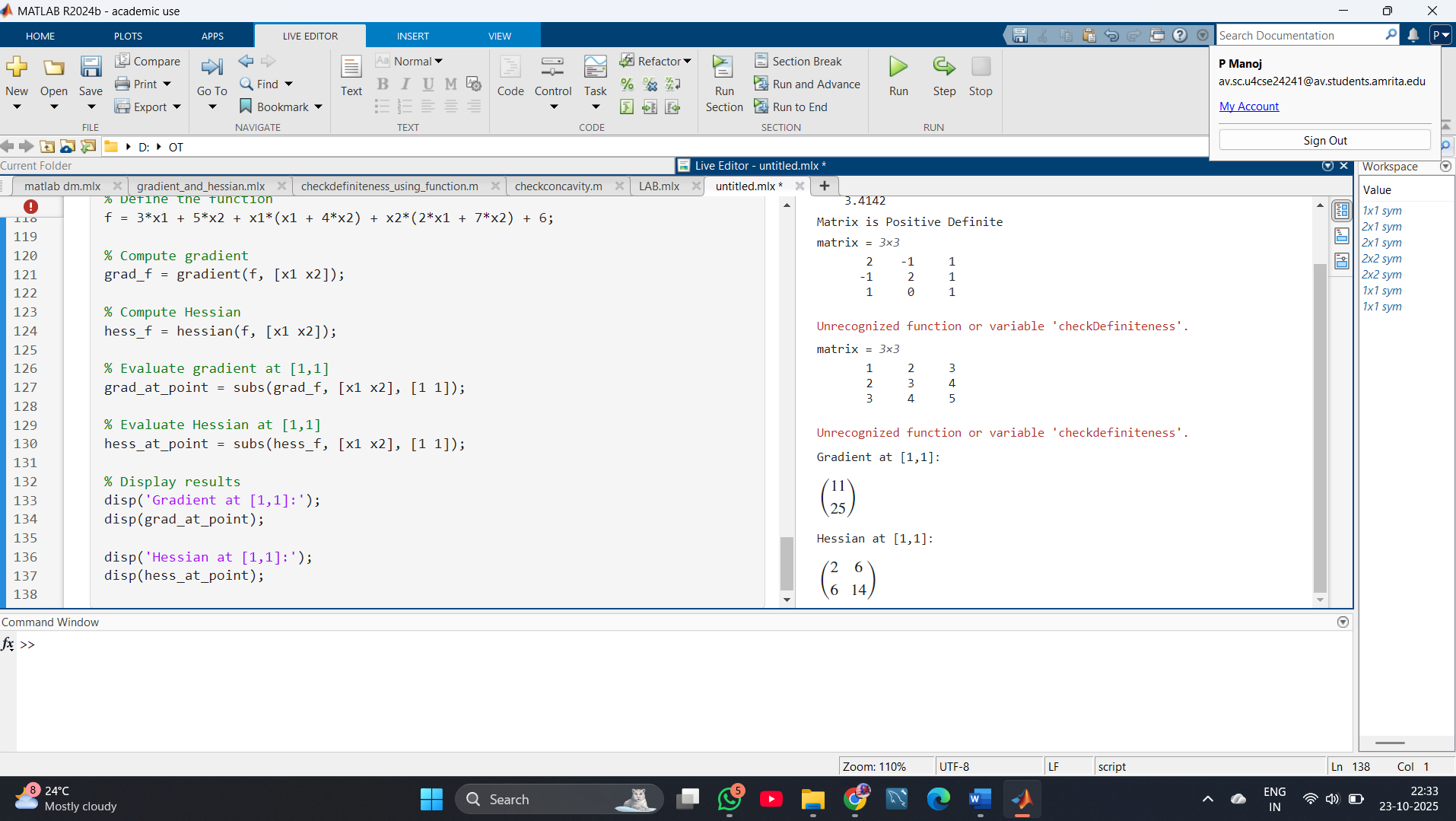
disp('Gradient at [1,1]:');

disp(grad\_at\_point);

disp('Hessian at [1,1]:');

disp(hess\_at\_point);

OUTPUT:



6. Consider the following function. f(x)=x⊤[14​27​]x+x⊤[35​]+6

a. Find the gradient and Hessian of f at the point [1,1].

b. Find the directional derivative of f at [1,1] with respect to a unit vector in the direction of maximal rate of increase.

c. Find a point that satisfies the FONC (interior case) for f. Does this point satisfy the SONC (for a minimizer)?

CODE:

clc; clear;

syms x1 x2

x = [x1; x2];

% Define matrices

A = [1 2; 4 7];

b = [3; 5];

% Define the function

f = x.'\*A\*x + x.'\*b + 6;

% (a) Gradient and Hessian

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

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

% Evaluate at [1,1]

point = [1 1];

grad\_at\_point = subs(grad\_f, [x1 x2], point);

hess\_at\_point = subs(hess\_f, [x1 x2], point);

disp('Gradient at [1,1]:');

disp(grad\_at\_point);

disp('Hessian at [1,1]:');

disp(hess\_at\_point);

% (b) Directional derivative in the direction of maximal increase

% Direction of maximal increase is along the gradient

unit\_grad = grad\_at\_point / norm(grad\_at\_point);

dir\_derivative = double(grad\_at\_point.' \* unit\_grad);

disp('Unit vector in direction of maximal rate of increase:');

disp(unit\_grad);

disp('Directional derivative in this direction:');

disp(dir\_derivative);

% (c) Find point satisfying FONC: grad\_f = 0

FONC\_point = solve(grad\_f == 0, [x1 x2]);

disp('Point satisfying FONC:');

disp(FONC\_point);

% Check SONC (Hessian positive definite for minimizer)

H\_at\_FONC = double(subs(hess\_f, [x1 x2], [FONC\_point.x1 FONC\_point.x2]));

eig\_vals = eig(H\_at\_FONC);

disp('Eigenvalues of Hessian at FONC point:');

disp(eig\_vals);

if all(eig\_vals > 0)

disp('FONC point satisfies SONC (local minimum)');

elseif all(eig\_vals < 0)

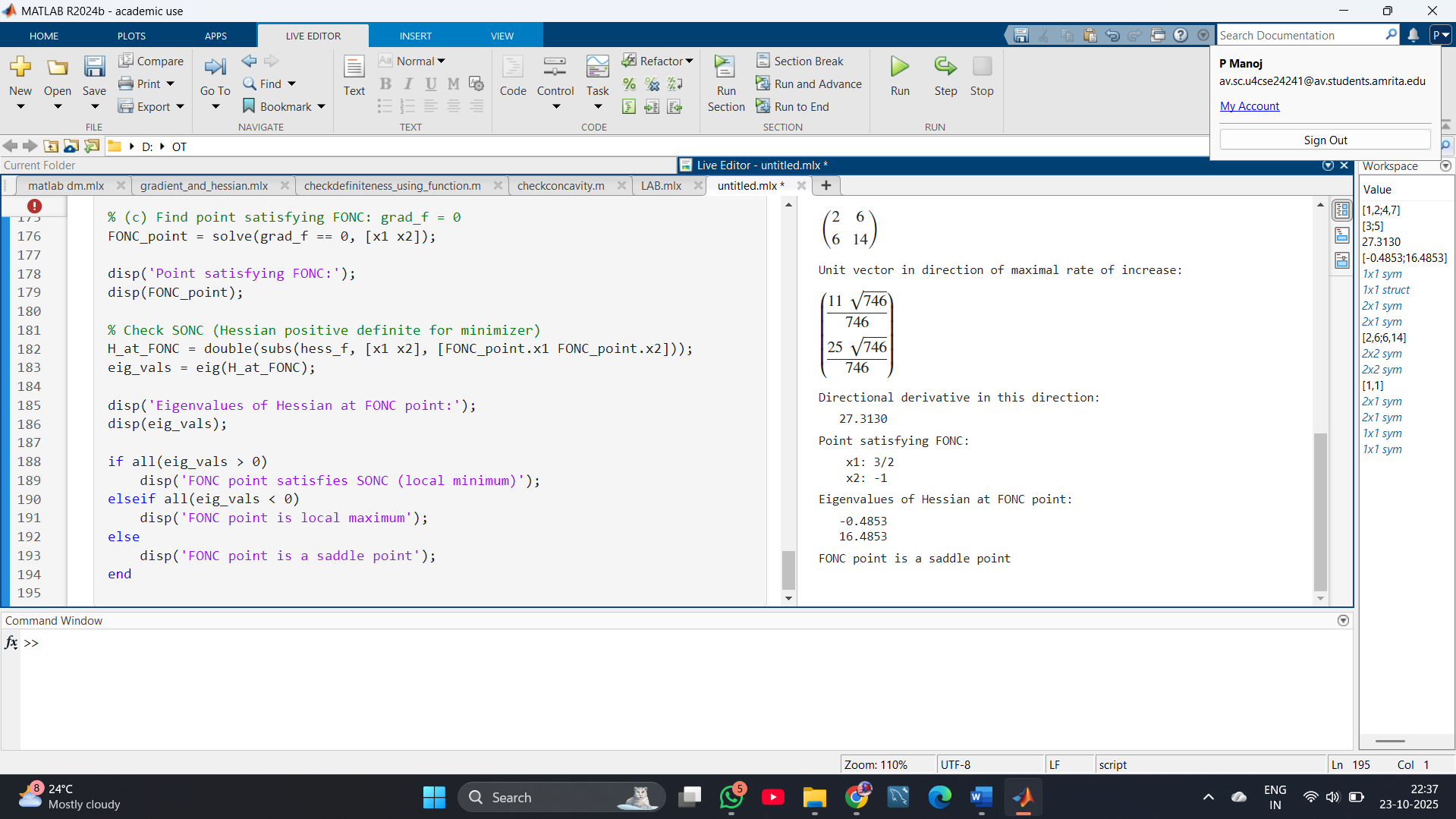
disp('FONC point is local maximum');

else

disp('FONC point is a saddle point');

end

OUTPUT:



7. Consider the following function: *f*(*x*1,*x*2)=*x*12*x*2+*x*23*x*1

a. In what direction does the function f decrease most rapidly at the point x(0) = [2,1] ?

b. What is the rate of increase of f at the point x(0) in the direction of maximum decrease of f?

c. Find the rate of increase of f at the point x(0) in the direction d = [3,4] .

CODE:

syms x1 x2

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

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

x0 = [2; 1]; % Point of interest

grad\_at\_point = double(subs(grad\_f, [x1, x2], x0.'));

% a. Direction of most rapid decrease

dir\_descent = -grad\_at\_point / norm(grad\_at\_point);

% b. Rate of increase in direction of maximum decrease

rate\_decrease = -norm(grad\_at\_point);

% c. Rate of increase in direction d = [3, 4]

d = [3; 4];

d\_unit = d / norm(d);

rate\_direction\_d = dot(grad\_at\_point, d\_unit);

disp('Direction of most rapid decrease:');

disp(dir\_descent)

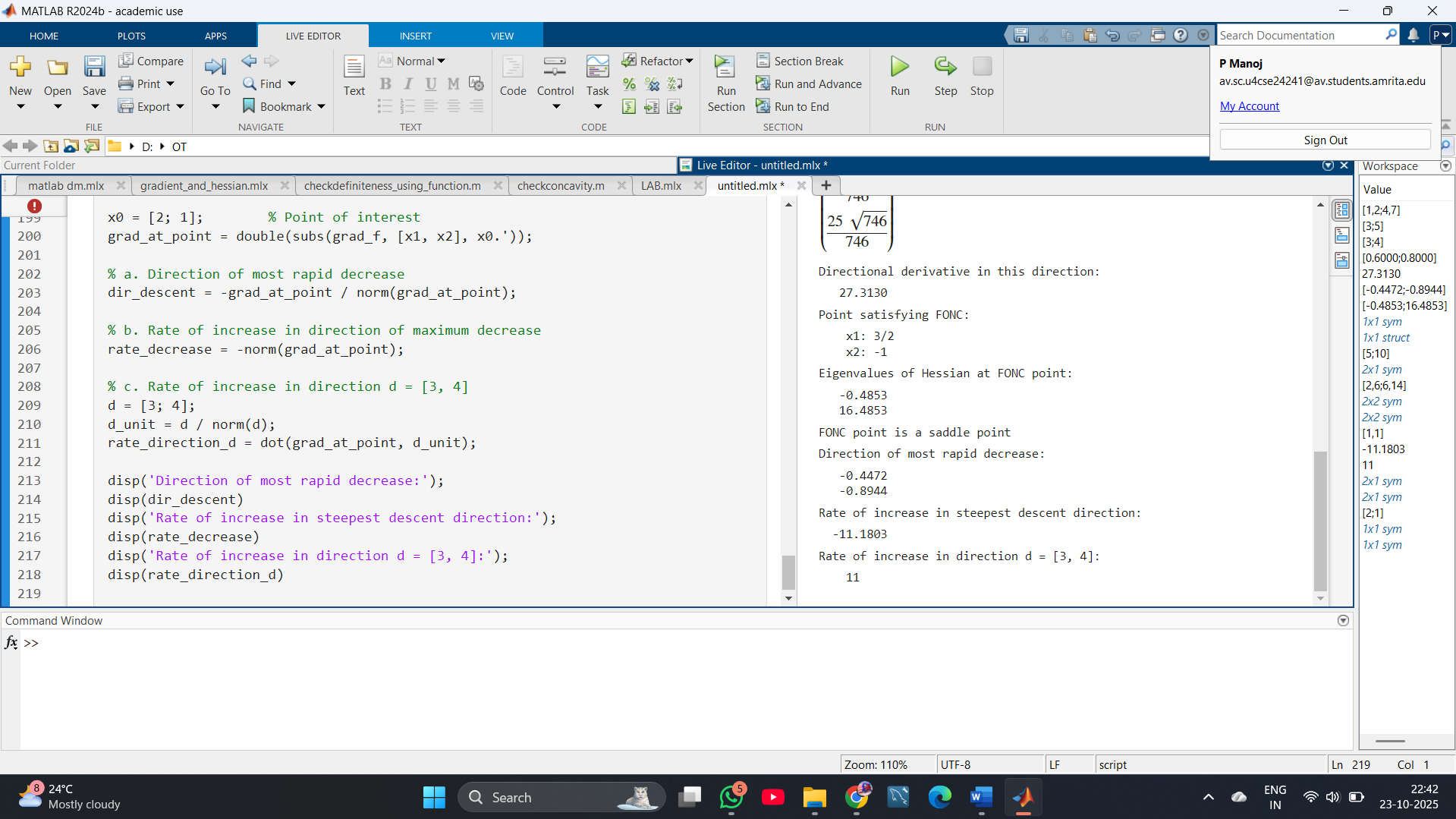
disp('Rate of increase in steepest descent direction:');

disp(rate\_decrease)

disp('Rate of increase in direction d = [3, 4]:');

disp(rate\_direction\_d)

OUTPUT:



8. Consider the following function f :

*f*(*x*)=*xT*[2−151]*x*+*xT*[34]+7

a. Find the directional derivative of f at [0,1] in the direction [1,0] .

b. Find all points that satisfy the first-order necessary condition for f. Does f have a minimizer? If it does, then find all minimizer(s); otherwise, explain why it does not.

CODE:

syms y z real

A = [2 5; -1 1];

b = [3; 4];

x = [y; z];

f = x.' \* A \* x + x.' \* b + 7;

% Gradient

grad\_f = gradient(f, [y z]).';

% a. Directional derivative at [0, 1] in [1, 0]

pt = [0; 1];

grad\_at\_pt = double(subs(grad\_f, [y, z], pt.'));

dir = [1; 0];

dir\_unit = dir / norm(dir);

dir\_derivative = dot(grad\_at\_pt, dir\_unit);

disp('Directional derivative:');

disp(dir\_derivative)

% b. First-order necessary condition (critical points)

sols = solve(grad\_f == 0, [y, z]);

critical\_points = [double(sols.y); double(sols.z)]

% Eigenvalues of Hessian

eigvals = eig(A);

if all(imag(eigvals) == 0 & eigvals > 0)

disp('Function has a unique minimizer (positive definite).');

disp('Critical point (minimizer):');

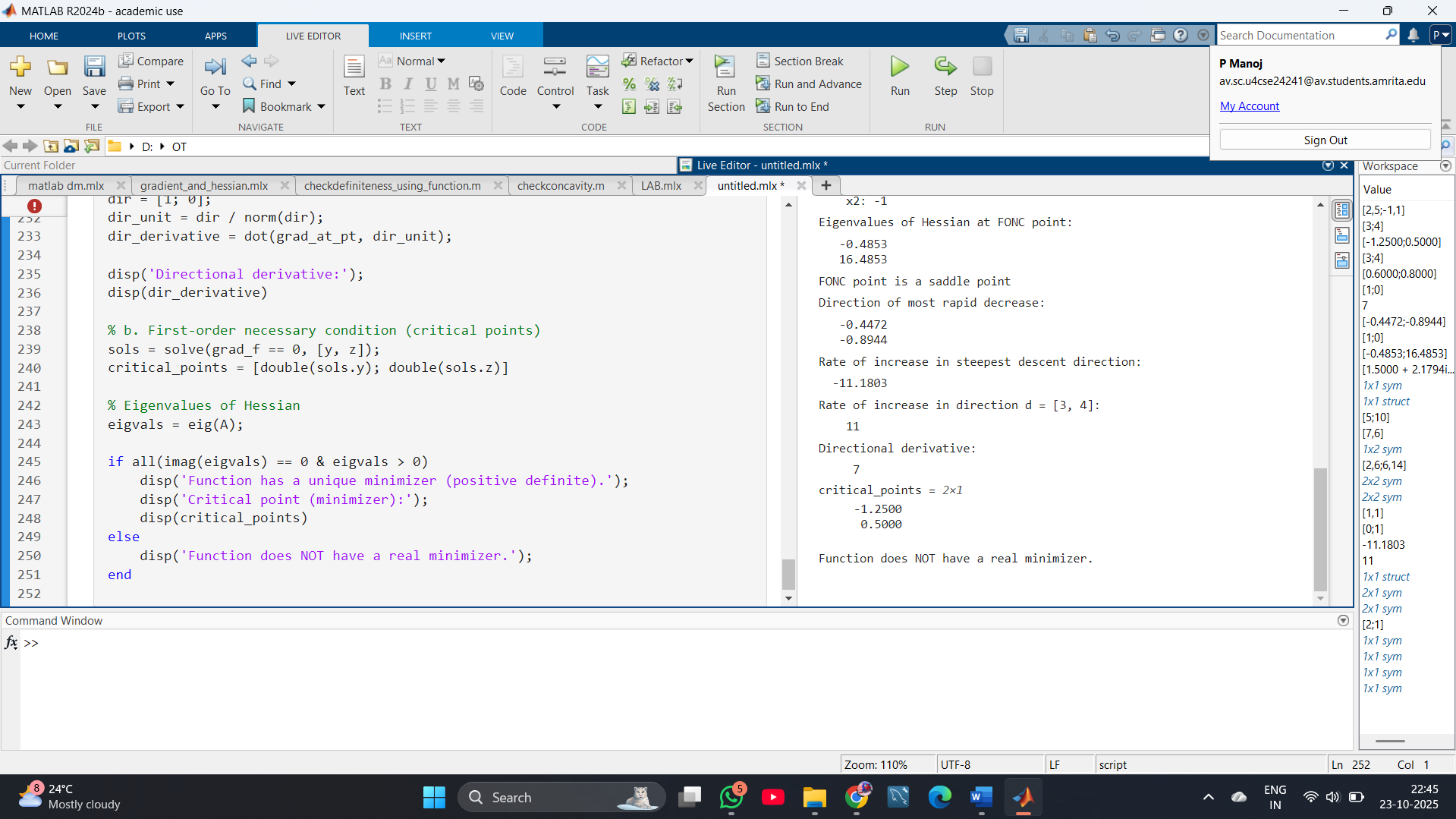
disp(critical\_points)

else

disp('Function does NOT have a real minimizer.');

end

OUTPUT:



9.Given f=x12+x22 find critical points and check concavity.

CODE:

syms x1 x2 real

f = x1^2 + x2^2;

% Gradient and Hessian

grad\_f = gradient(f, [x1 x2]).';

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

% Critical points

crit\_points = solve(grad\_f == 0, [x1 x2]);

disp('Critical points:');

disp([double(crit\_points.x1); double(crit\_points.x2)])

disp('Hessian:');

disp(hessian\_f)

disp('Eigenvalues of the Hessian:');

disp(eig(double(hessian\_f)))

if all(eig(double(hessian\_f)) > 0)

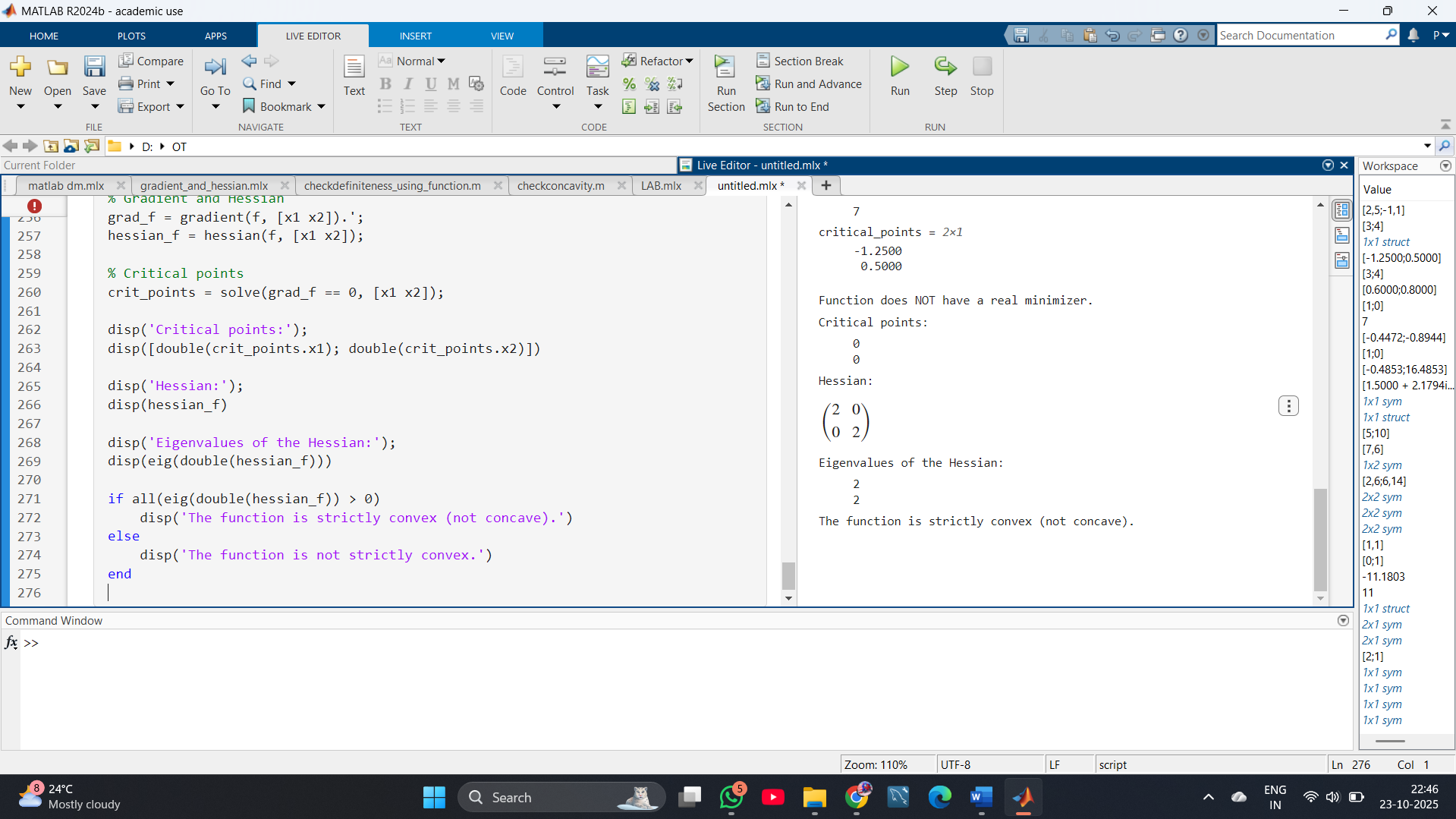
disp('The function is strictly convex (not concave).')

else

disp('The function is not strictly convex.')

end

OUTPUT:



10. Given f=(x+1)2+3 find critical points and check whether it is local maximizer or minimizer.

**CODE:**

**syms x real**

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

**% First derivative**

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

**% Critical points**

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

**% Second derivative**

**d2f = diff(f, x, 2);**

**disp('Critical points:');**

**disp(double(crit\_points))**

**% Check nature of critical point**

**if double(subs(d2f, x, crit\_points)) > 0**

**disp('It is a local minimizer.');**

**elseif double(subs(d2f, x, crit\_points)) < 0**

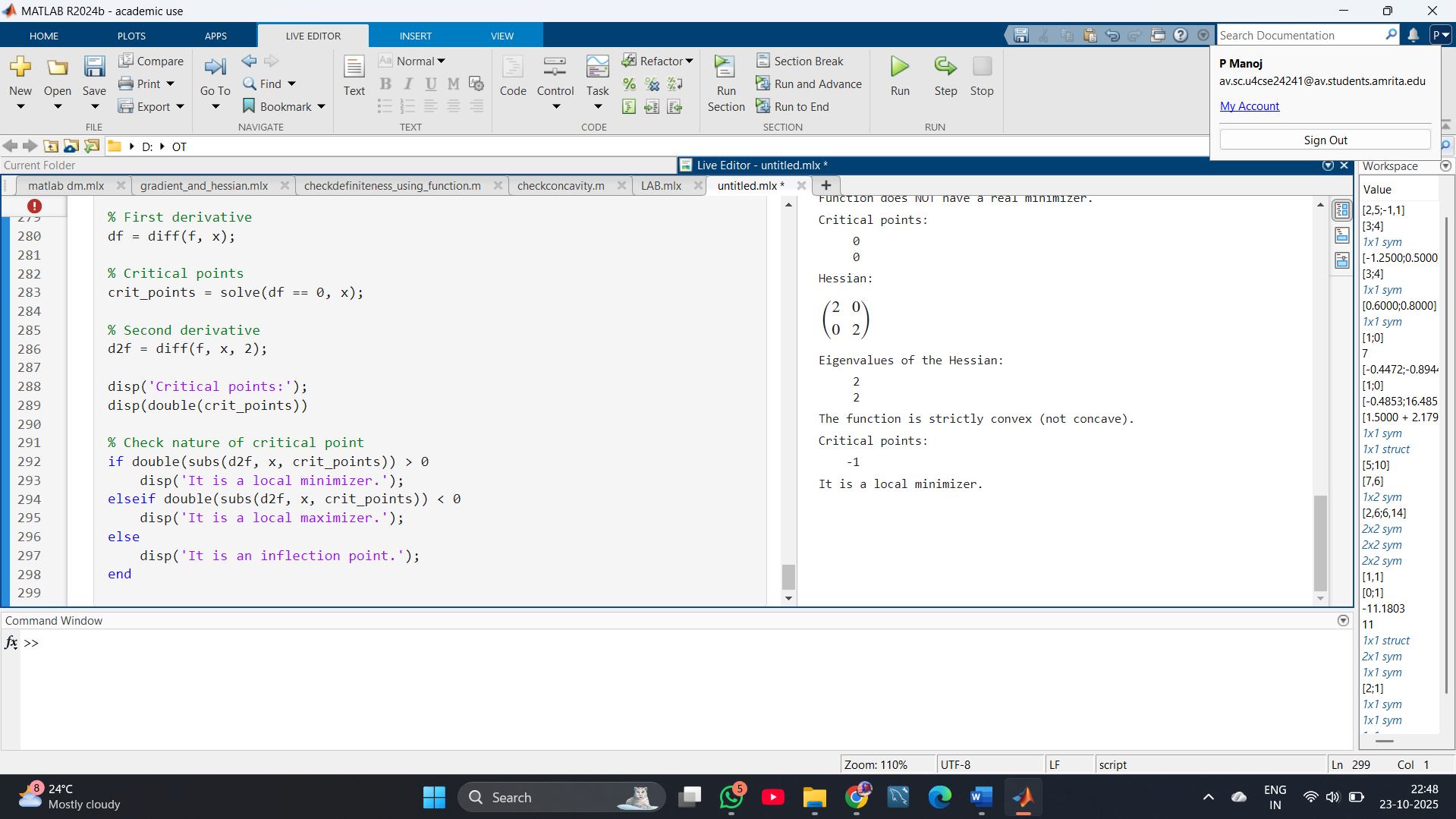
**disp('It is a local maximizer.');**

**else**

**disp('It is an inflection point.');**

**end**

**OUTPUT:**

****

**11.**Use Golden Section Search to minimize

f(x) = x4 - 14x3 + 60x2 - 70x at interval [0,2] with accuracy 0.3.

CODE:

**% Define function**

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

**% Golden Section parameters**

**a = 0;**

**b = 2;**

**eps = 0.3;**

**gr = (sqrt(5) - 1) / 2; % Golden ratio**

**% Initial internal points**

**c = b - gr\*(b - a);**

**d = a + gr\*(b - a);**

**while abs(b - a) > eps**

**if f(c) < f(d)**

**b = d;**

**d = c;**

**c = b - gr\*(b - a);**

**else**

**a = c;**

**c = d;**

**d = a + gr\*(b - a);**

**end**

**end**

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

**fmin = f(xmin);**

**fprintf('The minimum is at x = %.4f with f(x) = %.4f\n', xmin, fmin);**

**OUTPUT:**

****

**12.** Perform two iterations leading to the minimization of f(x1,x2)=x1+(1/2)x2+(1/2)x12+x22+3 using the steepest descent method with the starting point x(0)= 0. Also determine an optimal solution analytically.

**CODE:**

**syms x1 x2 real**

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

**% Gradient**

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

**% Initial point**

**xk = [0; 0];**

**for k = 1:2**

**gk = double(subs(grad\_f, [x1 x2], xk.'));**

**dk = -gk; % Steepest descent direction**

**% Symbolic line search**

**syms alpha**

**x\_next = xk + alpha\*dk;**

**f\_alpha = subs(f, [x1 x2], x\_next.');**

**df\_alpha = diff(f\_alpha, alpha);**

**alpha\_star = double(solve(df\_alpha == 0, alpha));**

**% Update**

**xk = double(xk + alpha\_star\*dk);**

**fprintf('x\_%d = [%.4f, %.4f]\\n', k, xk(1), xk(2));**

**end**

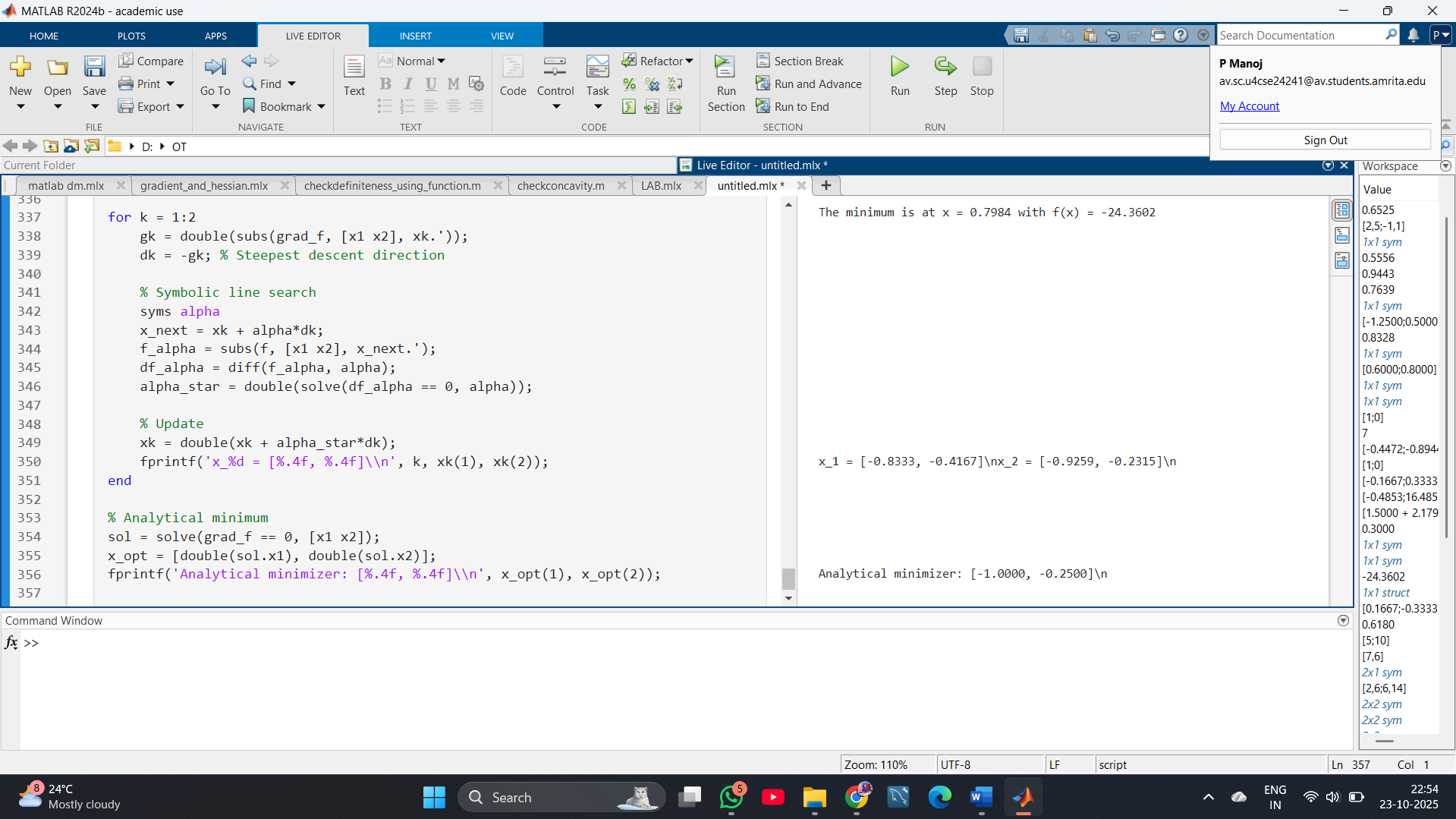
**% Analytical minimum**

**sol = solve(grad\_f == 0, [x1 x2]);**

**x\_opt = [double(sol.x1), double(sol.x2)];**

**fprintf('Analytical minimizer: [%.4f, %.4f]\\n', x\_opt(1), x\_opt(2));**

**OUTPUT:**

****

**13.** Use the method of steepest descent to find the minimizer of *f*(*x*1,*x*2,*x*3)=(*x*1−4)4+(*x*2−3)2+4(*x*3+5)4

**CODE:**

**syms x1 x2 x3 real**

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

**% Gradient**

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

**% Initial point (choose any, e.g., [0;0;0])**

**xk = [0; 0; 0];**

**for k = 1:5 % Run for 5 iterations as an example, adjust as needed**

**gk = double(subs(grad\_f, [x1 x2 x3], xk.'));**

**dk = -gk; % Steepest descent direction**

**% Symbolic line search**

**syms alpha**

**x\_next = xk + alpha\*dk;**

**f\_alpha = subs(f, [x1 x2 x3], x\_next.');**

**df\_alpha = diff(f\_alpha, alpha);**

**alpha\_star = double(solve(df\_alpha == 0, alpha));**

**% For robust code, pick real, positive solution if multiple**

**alpha\_star = alpha\_star(imag(alpha\_star)==0 & alpha\_star>0);**

**if length(alpha\_star) > 1**

**alpha\_star = alpha\_star(1);**

**end**

**% Update**

**xk = double(xk + alpha\_star\*dk);**

**fprintf('x\_%d = [%.4f, %.4f, %.4f]\\n', k, xk(1), xk(2), xk(3));**

**% Optional: Stopping criterion if gradient close to zero**

**if norm(gk) < 1e-6**

**break**

**end**

**end**

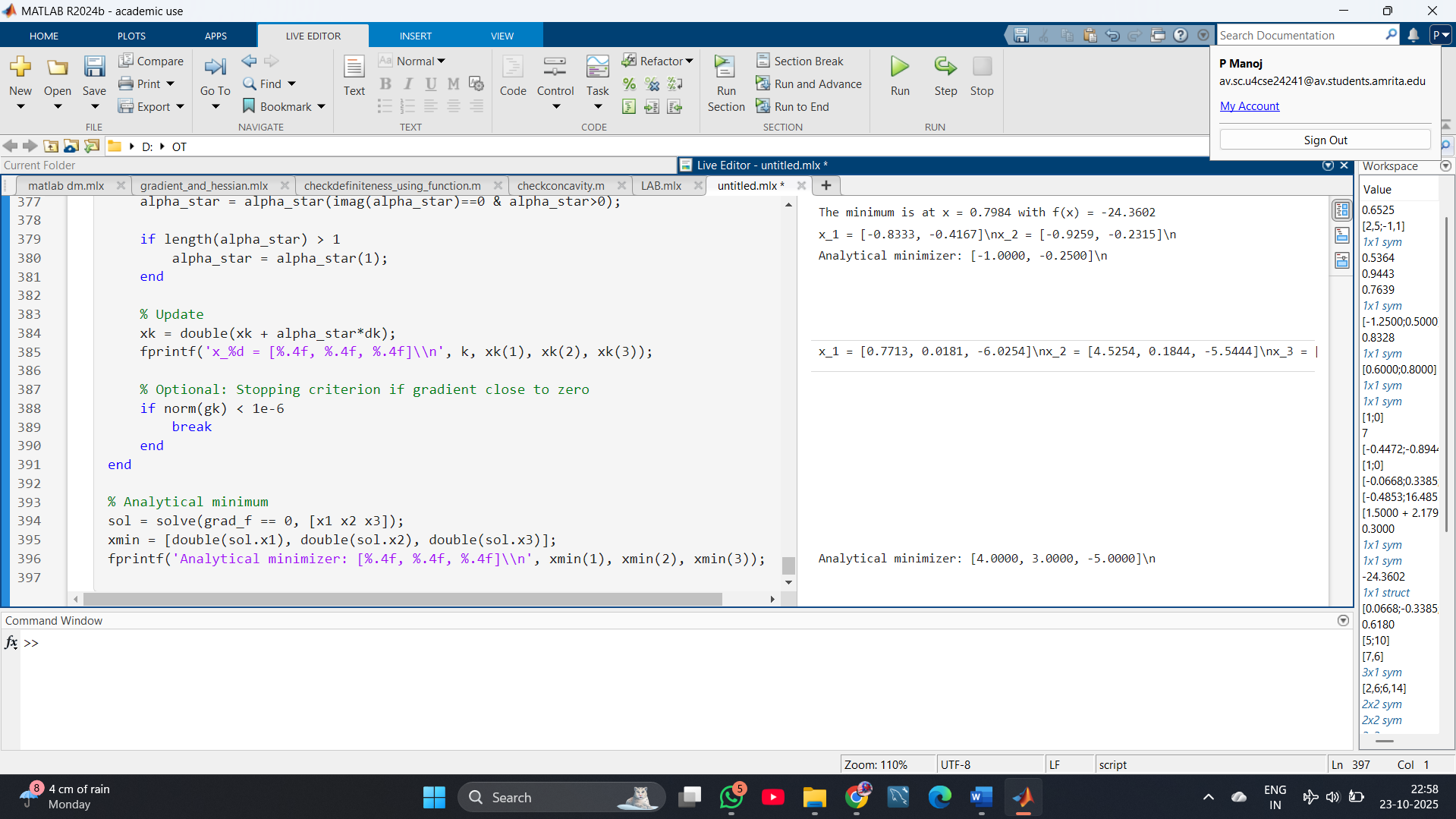
**% Analytical minimum**

**sol = solve(grad\_f == 0, [x1 x2 x3]);**

**xmin = [double(sol.x1), double(sol.x2), double(sol.x3)];**

**fprintf('Analytical minimizer: [%.4f, %.4f, %.4f]\\n', xmin(1), xmin(2), xmin(3));**

**OUTPUT:**

****

**14.** F(x1,x2)=x1^2+x2^2 Let Then, starting from an arbitrary initial point x(0).

CODE:

syms x1 x2 real

% Define the function

F = x1^2 + x2^2;

% Compute the gradient

grad\_F = gradient(F, [x1; x2]);

% Arbitrary initial point (user can set values)

xk = [3; -2]; % Example starting point

for k = 1:5 % Number of iterations, change as needed

gk = double(subs(grad\_F, [x1, x2], [xk(1), xk(2)]));

dk = -gk; % Steepest descent direction

% Symbolic exact line search

syms alpha real

x\_next = xk + alpha \* dk;

F\_alpha = subs(F, [x1, x2], [x\_next(1), x\_next(2)]);

dF\_alpha = diff(F\_alpha, alpha);

alpha\_sol = double(solve(dF\_alpha == 0, alpha));

% Pick the real, positive solution if multiple

alpha\_star = alpha\_sol(imag(alpha\_sol)==0 & alpha\_sol>0);

if isempty(alpha\_star)

% Fallback, just pick smallest if all real

alpha\_star = alpha\_sol(1);

end

xk = double(xk + alpha\_star \* dk); % Update

fprintf('x\_%d = [%.4f, %.4f]\\n', k, xk(1), xk(2));

if norm(gk) < 1e-6

break

end

end

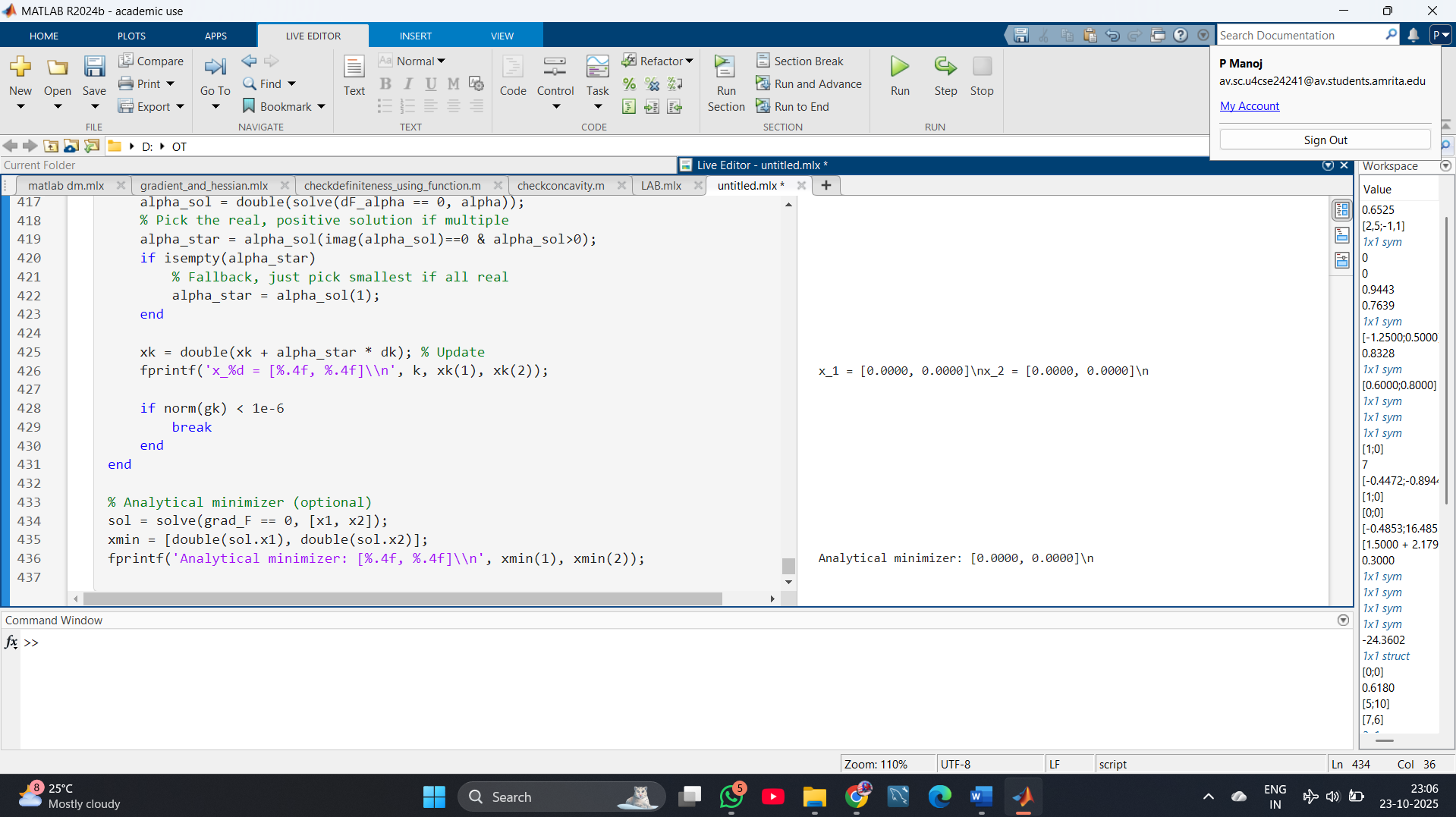
% Analytical minimizer (optional)

sol = solve(grad\_F == 0, [x1, x2]);

xmin = [double(sol.x1), double(sol.x2)];

fprintf('Analytical minimizer: [%.4f, %.4f]\\n', xmin(1), xmin(2));

OUTPUT:



15. Let the function f be given by

*f*(*x*)=*xT*[40225]*x*+*xT*[36]+24

Find the minimizer of f using a fixed-step-size gradient algorithm where α is a fixed step size.

CODE:

syms x1 x2 real

% Define quadratic function

A = [4, 2\*sqrt(2); 0, 5];

b = [3; 6];

f = [x1; x2].' \* A \* [x1; x2] + [x1; x2].' \* b + 24;

% Gradient

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

% Initial point (set your start here)

xk = [0; 0];

% Step size (try smaller if diverges)

alpha = 0.05;

max\_iter = 50;

for k = 1:max\_iter

gk = double(subs(grad\_f, [x1, x2], xk.'));

xk = xk - alpha \* gk;

fprintf('Iteration %d: x = [%.6f, %.6f], norm grad = %.6f\n', ...

k, xk(1), xk(2), norm(gk));

if norm(gk) < 1e-6

break

end

end

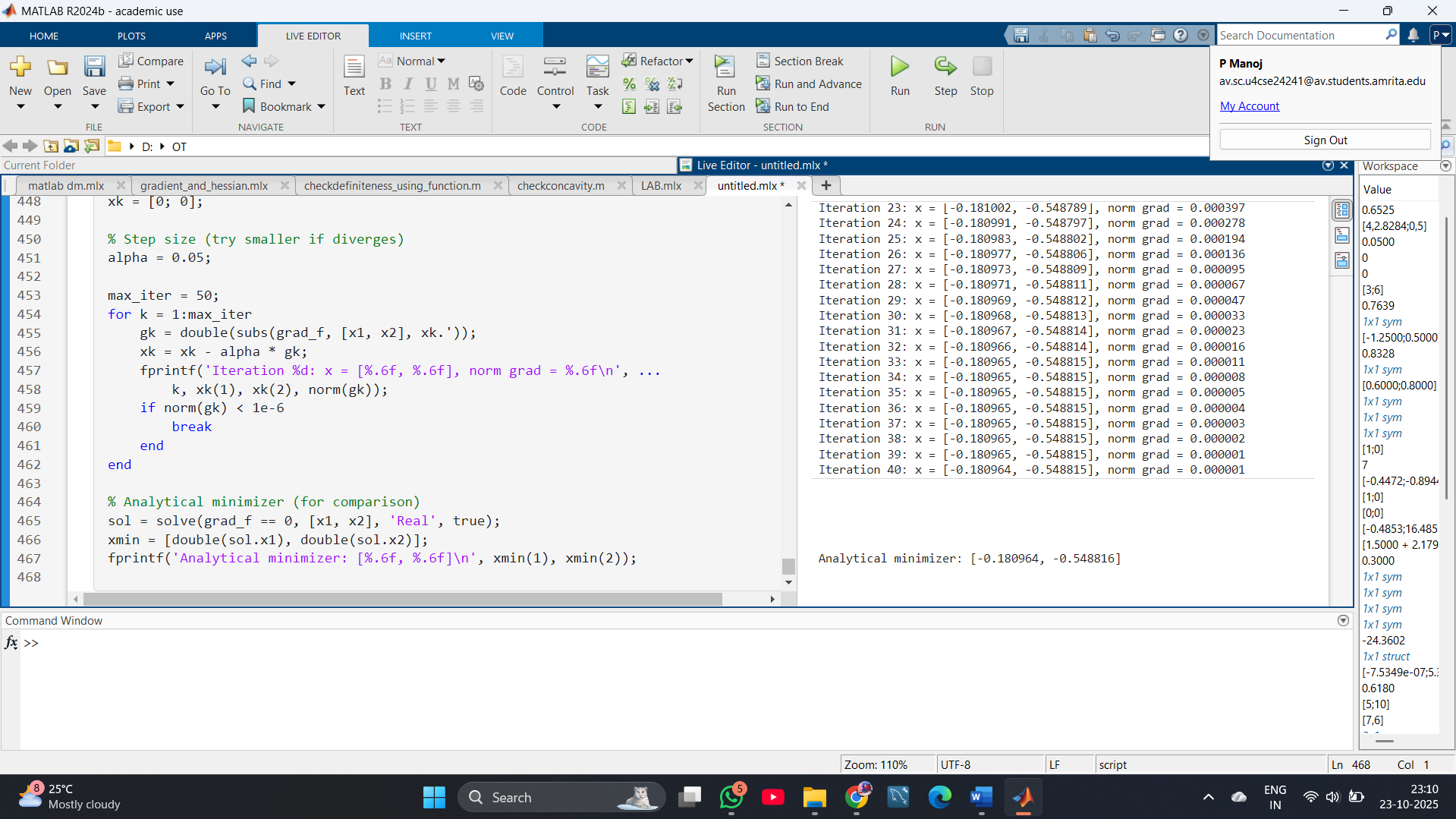
% Analytical minimizer (for comparison)

sol = solve(grad\_f == 0, [x1, x2], 'Real', true);

xmin = [double(sol.x1), double(sol.x2)];

fprintf('Analytical minimizer: [%.6f, %.6f]\n', xmin(1), xmin(2));

OUTPUT:



16. Suppose that 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 an arbitrarily small value of ε for the Fibonacci method.

CODE:

% Interval

a = 5;

b = 8;

L = b - a;

% Golden ratio

gr = (sqrt(5) - 1)/2;

% Compute uncertainty after n Golden Section iterations

golden\_n = 4; % 4 iterations

final\_golden = gr^golden\_n \* L;

% Fibonacci function (accurate integer sequence)

Fib = @(n) round((1/sqrt(5)) \* (((1+sqrt(5))/2)^n - ((1-sqrt(5))/2)^n));

% For Fibonacci, after n steps: new range ~ (1/Fn) \* L, where Fn is the n-th Fibonacci number

fibonacci\_n = 3; % Only 3 iterations

final\_fib = L / Fib(fibonacci\_n+1);

fprintf('After %d Golden Section iterations, final interval is: %.6f\n', golden\_n, final\_golden);

fprintf('After %d Fibonacci iterations, final interval is: %.6f\n', fibonacci\_n, final\_fib);

% Pick a desired uncertainty range between these two numbers

des\_uncertainty = (final\_golden + final\_fib)/2;

fprintf('An example desired uncertainty: %.6f\n', des\_uncertainty);

% Verify needed iterations

% Golden Section needed steps for this uncertainty

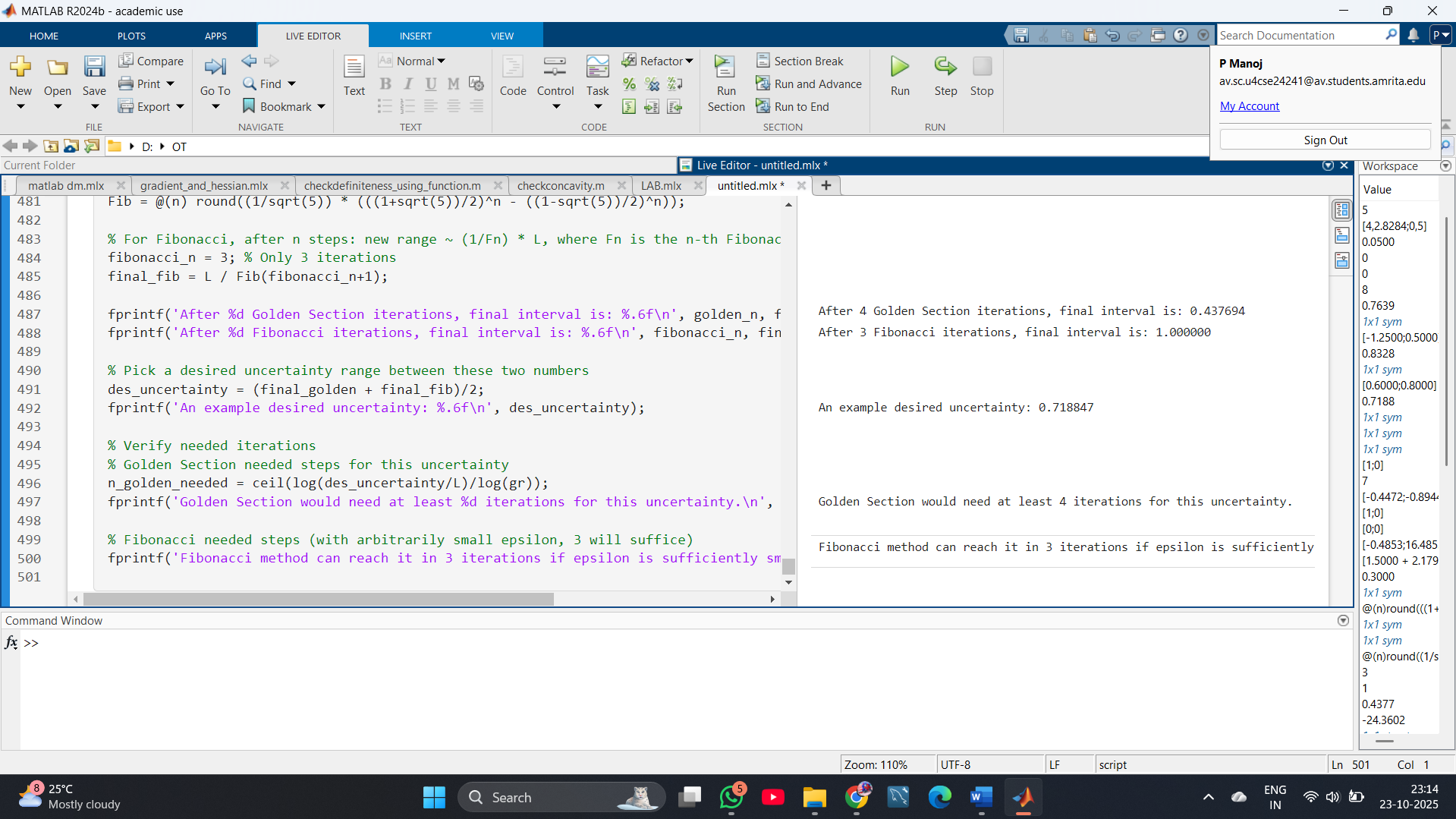
n\_golden\_needed = ceil(log(des\_uncertainty/L)/log(gr));

fprintf('Golden Section would need at least %d iterations for this uncertainty.\n', max(golden\_n, n\_golden\_needed));

% Fibonacci needed steps (with arbitrarily small epsilon, 3 will suffice)

fprintf('Fibonacci method can reach it in 3 iterations if epsilon is sufficiently small.\n');

OUTPUT:



17. Suppose that we wish to use the golden section search method to find the value of x

*f*(*x*)=*x*4−14*x*3+60*x*2−70*x* that minimizes in the interval [0,2] of x within a range of 0.3.

CODE:

% Define the function

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

% Interval endpoints

a = 0;

b = 2;

tol = 0.3;

% Golden ratio

gr = (sqrt(5) - 1) / 2;

% Initial points

c = b - gr \* (b - a);

d = a + gr \* (b - a);

while (b - a) > tol

if f(c) < f(d)

b = d;

d = c;

c = b - gr \* (b - a);

else

a = c;

c = d;

d = a + gr \* (b - a);

end

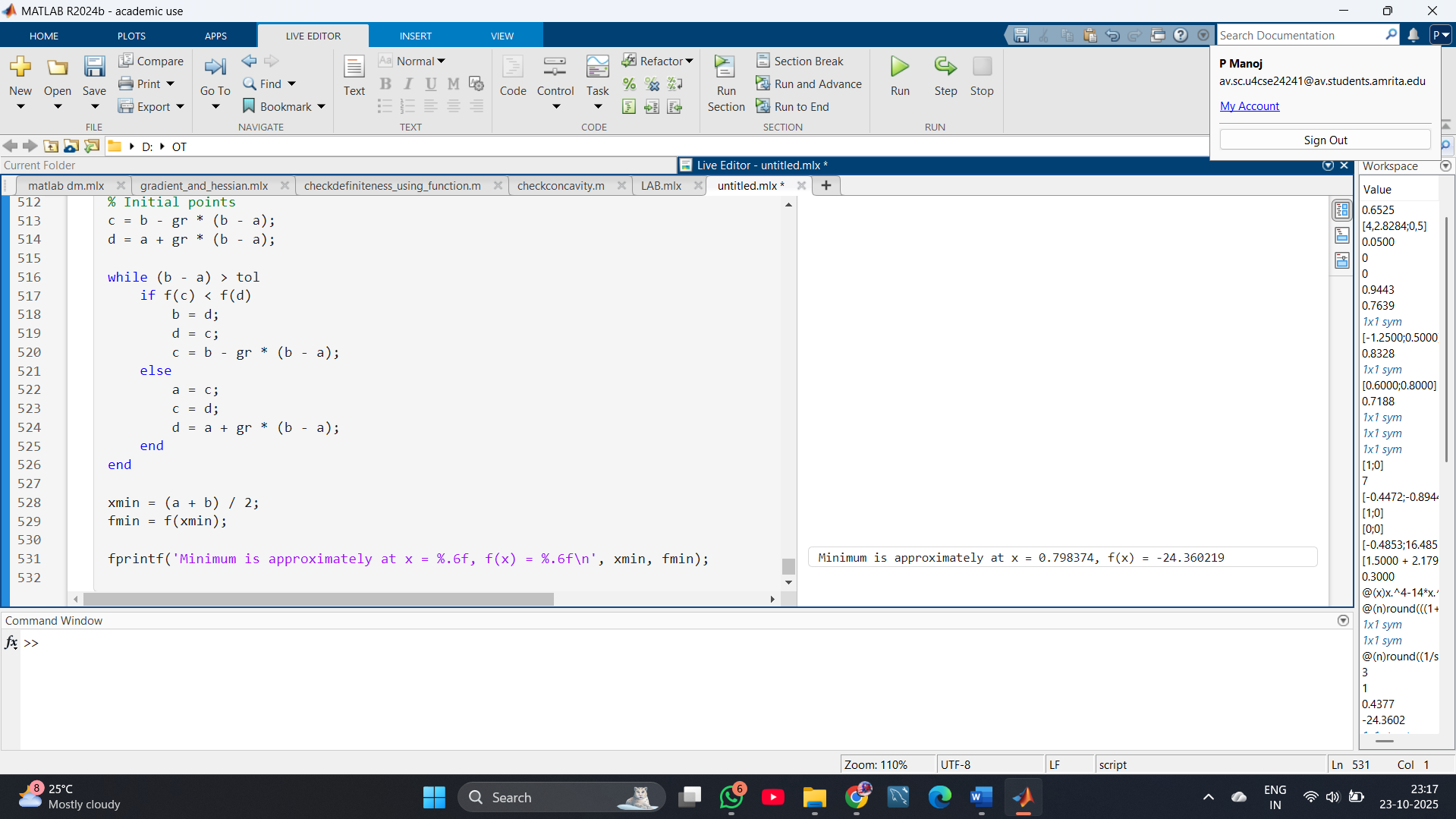
end

xmin = (a + b) / 2;

fmin = f(xmin);

fprintf('Minimum is approximately at x = %.6f, f(x) = %.6f\n', xmin, fmin);

**OUTPUT:**

****

**18.** **Use the golden section search method to find the value of x that minimizes in the interval [1,2] of x within a range of 0.23.**

**CODE:**

**% Define the function**

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

**% Set interval and tolerance**

**a = 1;**

**b = 2;**

**tol = 0.23;**

**% Golden ratio**

**gr = (sqrt(5)-1)/2;**

**% Initialize internal points**

**c = b - gr \* (b - a);**

**d = a + gr \* (b - a);**

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

**if f(c) < f(d)**

**b = d;**

**d = c;**

**c = b - gr \* (b - a);**

**else**

**a = c;**

**c = d;**

**d = a + gr \* (b - a);**

**end**

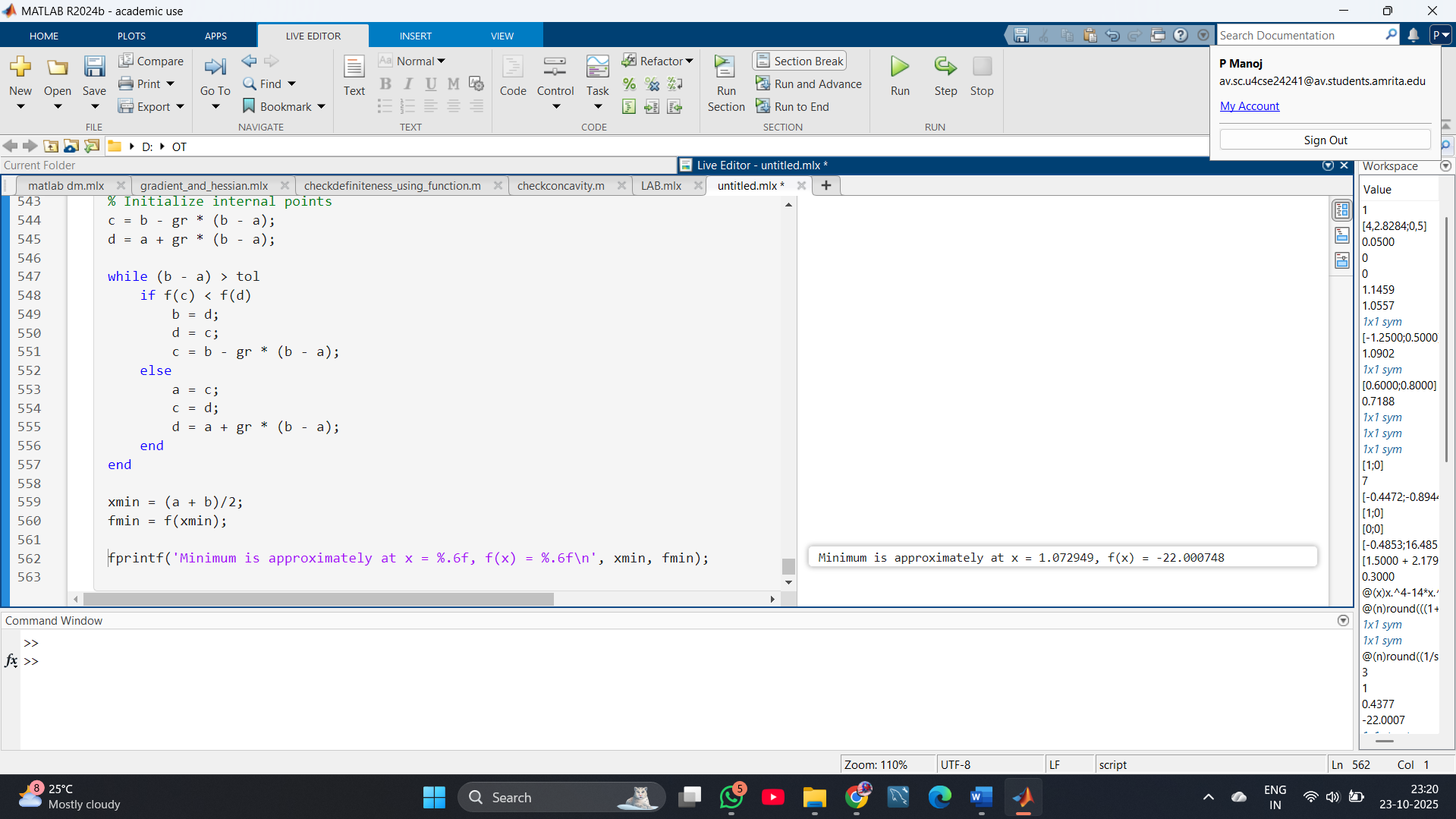
**end**

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

**fmin = f(xmin);**

**fprintf('Minimum is approximately at x = %.6f, f(x) = %.6f\n', xmin, fmin);**

**OUTPUT:**

****

**19.** Use the Fibonacci method to find the value of x that minimizes in the interval [1,2] of x within a range of 0.23.

CODE:

**% Define the function**

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

**% Set interval and tolerance**

**a = 1;**

**b = 2;**

**tol = 0.23;**

**% Generate Fibonacci numbers until the ratio is satisfied**

**F = [1 1];**

**while F(end) < (b - a)/tol**

**F(end+1) = F(end) + F(end-1);**

**end**

**N = length(F) - 1;**

**% Initial interior points**

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

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

**f1 = f(x1);**

**f2 = f(x2);**

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

**if f1 > f2**

**a = x1;**

**x1 = x2;**

**f1 = f2;**

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

**f2 = f(x2);**

**else**

**b = x2;**

**x2 = x1;**

**f2 = f1;**

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

**f1 = f(x1);**

**end**

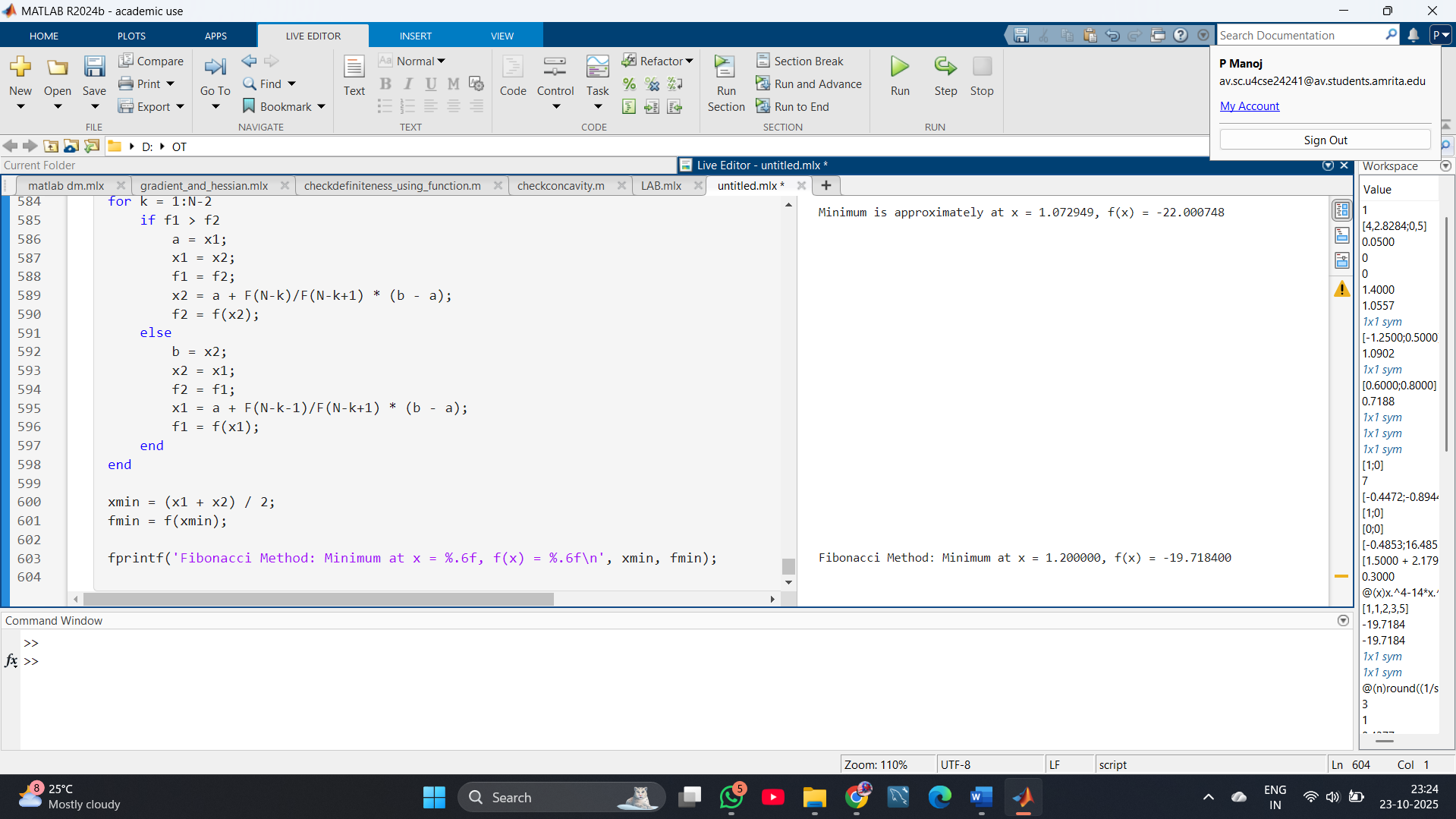
**end**

**xmin = (x1 + x2) / 2;**

**fmin = f(xmin);**

**fprintf('Fibonacci Method: Minimum at x = %.6f, f(x) = %.6f\n', xmin, fmin);**

**OUTPUT:**

****

**20.** Write a matlab code to find minimizer of a given function using Newton’s Method.

CODE:

**syms x real**

**% Define your function**

**f = x^4 - 14\*x^3 + 60\*x^2 - 70\*x; % Example: change to any differentiable function**

**% Derivative and second derivative**

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

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

**% Convert to function handles for evaluation**

**f\_num = matlabFunction(f, 'Vars', x);**

**df\_num = matlabFunction(df, 'Vars', x);**

**d2f\_num = matlabFunction(d2f, 'Vars', x);**

**% Initial guess**

**xk = 1.5; % Choose a suitable starting point**

**tol = 1e-6;**

**max\_iter = 100;**

**for k = 1:max\_iter**

**gk = df\_num(xk);**

**Hk = d2f\_num(xk);**

**if abs(Hk) < eps**

**error('Second derivative is zero. Newton''s method fails.');**

**end**

**xk\_new = xk - gk/Hk;**

**fprintf('Iter %d: x = %.8f, f(x) = %.8f, |step| = %.8e\n', ...**

**k, xk\_new, f\_num(xk\_new), abs(xk\_new - xk));**

**if abs(xk\_new - xk) < tol**

**break**

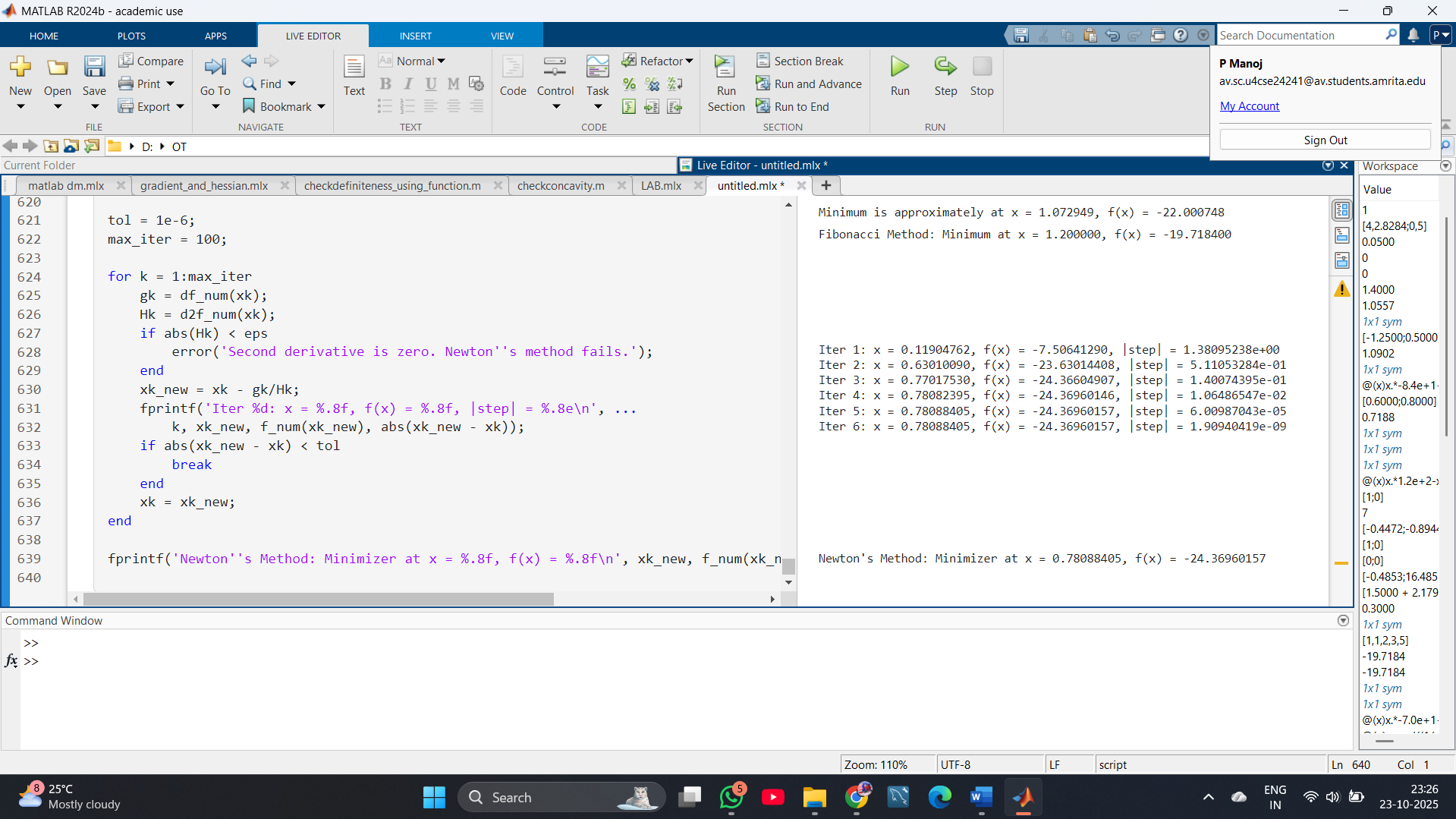
**end**

**xk = xk\_new;**

**end**

**fprintf('Newton''s Method: Minimizer at x = %.8f, f(x) = %.8f\n', xk\_new, f\_num(xk\_new));**

**OUTPUT:**

****

**21.** Write a matlab code to find root of a given function using Newton’s Rapson Method.

CODE:

syms x real

% Define your function

f = x^4 - 14\*x^3 + 60\*x^2 - 70\*x; % Change to your function if desired

% Its first derivative

df = diff(f, x);

% Convert symbolic expressions to function handles

f\_num = matlabFunction(f, 'Vars', x);

df\_num = matlabFunction(df, 'Vars', x);

% Initial guess

xk = 1.7; % Set a starting point

tol = 1e-6;

max\_iter = 100;

step = inf;

for k = 1:max\_iter

fk = f\_num(xk);

dfk = df\_num(xk);

if abs(dfk) < eps

error('Derivative is zero. Method fails.');

end

xk\_new = xk - fk/dfk;

step = abs(xk\_new - xk);

fprintf('Iter %d: x = %.8f, f(x) = %.8f, |step| = %.8e\n', k, xk\_new, f\_num(xk\_new), step);

if step < tol

xk = xk\_new;

break

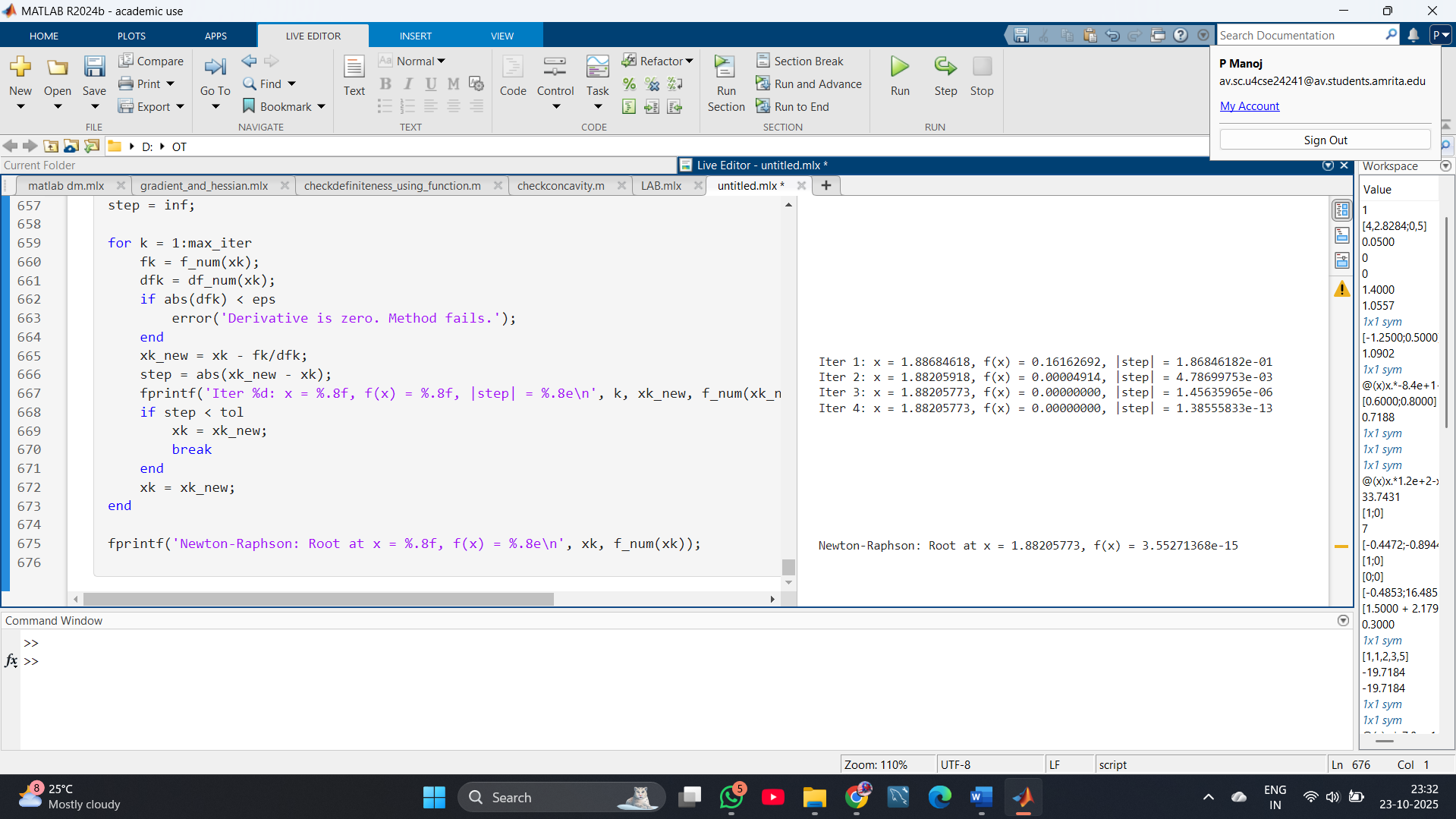
end

xk = xk\_new;

end

fprintf('Newton-Raphson: Root at x = %.8f, f(x) = %.8e\n', xk, f\_num(xk));

OUTPUT:



22. Write a matlab code to find root of a given function using Newton’s Rapson Method.

CODE:

syms x1 x2 real

% Define your function (example)

f = x1^2 + 3\*x2^2 + 2\*x1\*x2 - x1 + 4\*x2;

% Gradient and Hessian

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

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

% Convert to function handles

grad\_f\_num = matlabFunction(grad\_f, 'Vars', {[x1; x2]});

H\_f\_num = matlabFunction(H\_f, 'Vars', {[x1; x2]});

f\_num = matlabFunction(f, 'Vars', {[x1; x2]});

% Initial guess

xk = [0; 0];

tol = 1e-6;

max\_iter = 100;

for k = 1:max\_iter

gradk = grad\_f\_num(xk);

Hk = H\_f\_num(xk);

if rcond(Hk) < 1e-12

error('Hessian is singular or ill-conditioned.');

end

pk = -Hk \ gradk;

xk\_new = xk + pk;

step = norm(xk\_new - xk);

fprintf('Iter %d: x = [%.8f, %.8f], f(x) = %.8f, step = %.2e\n', ...

k, xk\_new(1), xk\_new(2), f\_num(xk\_new), step);

xk = xk\_new;

if step < tol

break

end

end

fprintf('Newton''s Method (2D) Minimizer at x = [%.8f, %.8f], f(x) = %.8f\n', ...

xk(1), xk(2), f\_num(xk));

OUTPUT:

