NUMERICAL OPTIMISATION
COURSE WORK (50%)

Marta Betcke

Submit your solutions to `Turnitin`. Your report **must be typeset** and should not exceed 15 pages. Most questions can be answered with a few sentences in addition to equations and plots. Please: Read carefully what we asked for, e.g. if we ask for one plot, only the first plot included will be evaluated, the choice of the correct plot is part of the answer. Be clear and succinct in your answers. Only include code into your report when explicitly asked for (follow the particular guidelines, most of the time we only want to see small snippets).

UCL rules and regulations on late submission and plagiarism apply.

EXERCISE 1 [30pt]

We are given a function $f : \Omega = \mathbb{R}_+ \times \mathbb{R} \to \mathbb{R}, \quad \mathbb{R}_+ := (0, \infty)$

$$f(x, y) = (y^2 + x \ln x)^2 + (y - \sqrt{x})^2, \quad x > 0.$$

Note the implicit constraint imposed by the domain of definition of $\ln(x)$ on real numbers which is not formally defined for $x \leq 0$.

(a) Plot the function and its contours. What can you say about the function: is it convex or not, are there and how many stationary points and local/global minima/infima? Justify your answers qualitatively. **[3 pt]**

(b) Find all the minimisers $x^* \in \Omega$ of the function f without using numerical methods and describe how you obtained them. **[7 pt]**

(c) Calculate the gradient ∇f. **[2 pt]**

(d) Calculate the Hessian $\nabla^2 f$. **[3 pt]**

(e) Verify the necessary first order conditions at all the minimisers. **[2 pt]**

(f) Verify the sufficient conditions at all the minimisers. **[3 pt]**

(g*) Analyse the numerical issues you may encounter when optimising this function, and explain how to make the implementation as robust as possible. **[10 pt]**

EXERCISE 2 [30pt]

Use steepest descent and Newton method to minimise the function f in **Ex.1**.

(a) Find a set of initial points x_0 which illustrate different behaviour of the optimisation methods. *`Note:` these two optimisation methods use local information so their behaviour depends on the points along their particular trajectories.* List and characterise the expected types of behaviour. After finishing the experiments, revisit and update your list/characterisation. *`Hint:` Try to initialise the methods from around the targeted minima.* **[4pt]**

(b) Apply steepest descent with an appropriate line search to the function f in **Ex.1** starting from your initialisation points. Justify your choice of a line search and any important parameters. What behaviour did you encounter for which point? Explain why the method behaved as it did in each case supporting your argument with relevant numerical evaluations. Plot the iterates over the function contours for one converged solution and for one "pathological" case. **[7pt]**

(c) For one converged solution investigate convergence of the steepest descent iterates in **(b)** a posteriori and include one error plot appropriate for the convergence regime. What are the empirical convergence rates and how did you obtain them? Do they agree with the theoretical predictions? Paraphrase the relevant theoretical results. **[4pt]**

(d) Apply Newton with an appropriate line search to the function f in **Ex.1** starting from your initialisation points. Justify your choice of a line search and any important parameters. What behaviour did you encounter for which point? Explain why the method behaved as it did in each case supporting your argument with relevant numerical evaluations. Plot the iterates over the function contours for one converged solution and for one "pathological" case. **[7pt]**

(e) For one converged solution, investigate convergence of the Newton iterates in **(d)** a posteriori and include one error plot appropriate for the convergence regime. What are the empirical convergence rates and how did you obtain them? Do they agree with the theoretical predictions? Paraphrase the relevant theoretical results. **[4pt]**

(f) Can global convergence of both methods be guaranteed or not and why? Paraphrase the relevant theoretical results. **[4pt]**

EXERCISE 3 [20pt]

(a) Implement the dogleg trust region method. Your implementation should return the *Cauchy point* whenever the gradient and Newton steps are collinear or when Hessian is not s.p.d.

Include your implementation in the report. Highlight the part where you solve for the intersection point between the trust region and the dogleg path and provide a short narrative explanation. **[6pt]**

(b) Apply the dogleg trust region method to minimise the function $f : \mathbb{R}^2 \to \mathbb{R}$

$$f(x, y) = 10(y^2 - x^2 - 1)^2 + (1 - x)^2$$

with three different starting points chosen to illustrate three different convergence scenarios. Plot the three trajectories traced by the iterates over the function contours. State your choice of the stopping condition and any relevant parameters. **[6pt]**

(c) Choose two most different convergence scenarios from those above. Investigate convergence of the dogleg iterates in **(b)** a posteriori and include one error plot per starting point best suited for the convergence regime. What are the empirical convergence rates and how did you obtain them? Do they agree with the theoretical predictions? Paraphrase the relevant theoretical results. **[4pt]**

(d) Can global convergence be expected or not, and why? Paraphrase the relevant theoretical results. *`Hint:` If you need to bound local Lipschitz constants, it is sufficient to give a high level answer for the type of functions.* **[4pt]**

EXERCISE 4 [20pt]

(a) Implement Polak-Ribieŕe nonlinear conjugate gradient method. Copy the relevant lines in your report. *`Hint:` Modify the `descentLineSearch.m` template from tutorial 2.* **[2pt]**

(b) Apply the Fletcher-Reeves and Polak-Ribieŕe nonlinear conjugate gradient methods to minimise the function $f : \mathbb{R}_+ \times \mathbb{R} \to \mathbb{R}$

$$f(x, y) = x \ln(x) + y^2 + 5\, e^{-(x-2)^2-(y-0.1)^2}.$$

Try two initial points $x_0 = (1.5, 0.5)^T$ and $x_0 = (2.5, 0)^T$ and set the tolerance `tol = 1e-4`. State your choice of any relevant parameters. Plot the iterates over the function contours. **[4pt]**

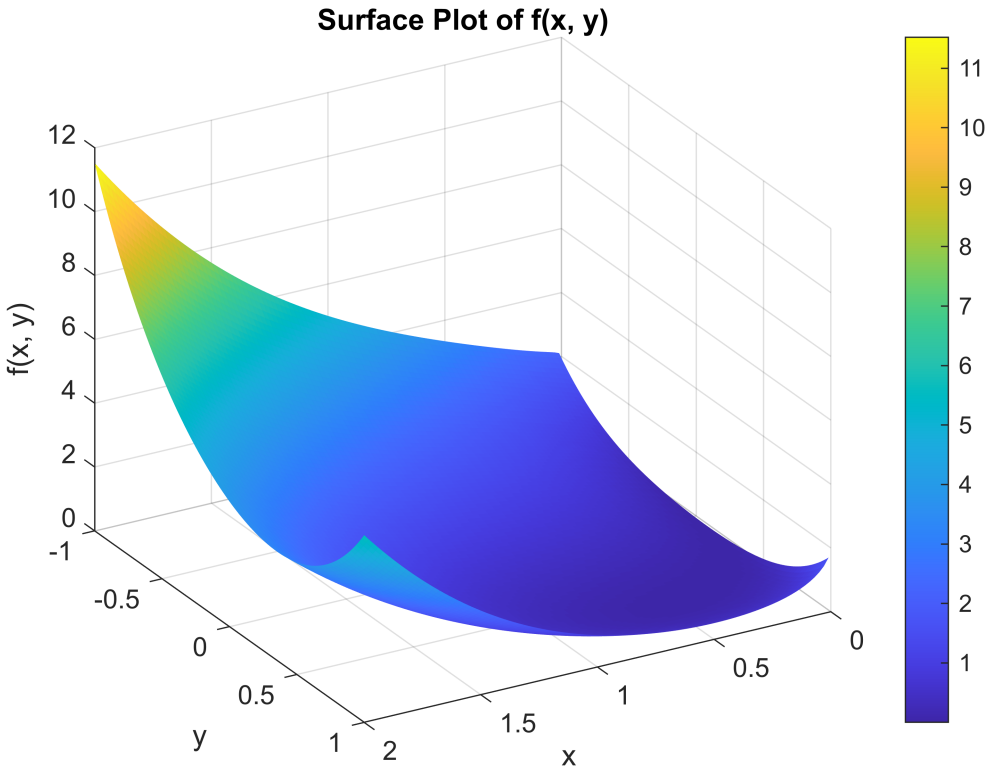
(c) Explain what you observe in the plots you made in **(b)**. Include plots of relevant quantities which corroborate your hypothesis. **[5pt]**

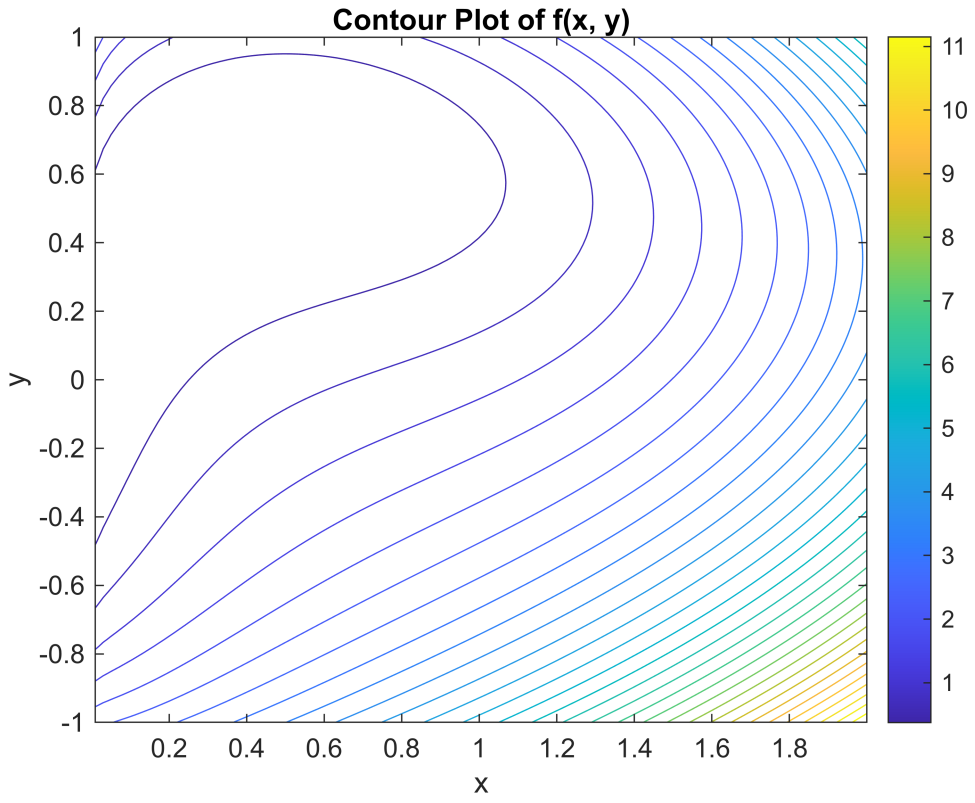
(d) Propose a strategy to remedy the problem and explain the theoretical guarantees you have for your approach. *`Hint:` Can the global convergence and what type be guaranteed?* Implement your strategy, rerun your solver and compare its behaviour with the results in **(b)**. **[5pt]**

(e) Investigate convergence of the all three methods in **(b,d)** a posteriori and include one error plot per method best suited for the convergence regime. What are the empirical convergence rates and how did you obtain them? Do they agree with the theoretical predictions? Paraphrase the relevant theoretical results. **[4pt]**

EXERCISE 1

(a)


Surface Plot of f(x, y)
f(x, y)
12
10
8
6
4
2
0
-1
-0.5
0
0.5
1
y
2
1.5
1
0.5
0
x
11
10
9
8
7
6
5
4
3
2
1


Contour Plot of f(x, y)
y
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
x
11
10
9
8
7
6
5
4
3
2
1

Observations:

• The function doesn't seem to be convex because the surface plot doesn't have a uniform curvature (as we can see, the left part increases significantly unlike the right part). Furthermore, the contour plot shows that the curves aren't equally spaced and for a function to be convex, it must be evenly spaced.

• There is one stationary point which is a global minimum where the contours converge toward a specific point, approximatly (x=1, y=0) and since the function is a sum of squares, it is always non-negative, indicating that this stationary point is a global minimum and also the infimum. There are no additional stationary points, meaning there are no other local minima, maxima, or saddle points.

(b):
We observe that $f(x, y) \geq 0$ because it is the sum of two non-negative terms. Also,

$$f(x, y) = 0 \Leftrightarrow y = \sqrt{x} \quad \text{and} \quad x + x \ln x = 0.$$

This is only possible if:

$$x = e^{-1}, \quad y = e^{-1/2}.$$

Hence, the minimizer is:

$$x^* = \begin{bmatrix} e^{-1} \\ e^{-1/2} \end{bmatrix}.$$

Computing the Hessian

The Hessian at x^* is computed as follows:

$$\nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix}.$$

Computing the second derivatives explicitly:

$$\frac{\partial^2 f}{\partial x^2} = 2(\ln x + 1)^2 + \frac{1}{4x^{3/2}}$$

$$\frac{\partial^2 f}{\partial y^2} = 4y^2 + 2$$

$$\frac{\partial^2 f}{\partial x \partial y} = \frac{\partial^2 f}{\partial y \partial x} = 4y(\ln x + 1) - \frac{1}{\sqrt{x}}.$$

Evaluating at $x^* = e^{-1}$, $y^* = e^{-1/2}$:

$$\nabla^2 f(x^*) = \begin{bmatrix} \frac{5}{2} & -2e^{-1/2} \\ -2e^{-1/2} & 4e^{-1} + 2 \end{bmatrix}.$$

Verifying Positive Definiteness

To verify that $\nabla^2 f(x^*)$ is positive definite, we must check:

1. The determinant condition:

$$\det(\nabla^2 f(x^*)) = \left(\frac{5}{2}\right)\left(4e^{-1} + 2\right) - (-2e^{-1/2})^2 > 0.$$

2. The leading principal minor:

$$\frac{5}{2} > 0.$$

Since both conditions hold, the Hessian is positive definite.

Final Conclusion

$$\boxed{(x^*, y^*) = \left(e^{-1}, e^{-1/2}\right)}$$

Thus, x^* is a unique global minimizer.

(c)

Gradient ∇f:

$$\begin{pmatrix} 2\ (\log(x)+1)\ (y^2+x\log(x)) - \dfrac{y-\sqrt{x}}{\sqrt{x}} \\ 2\ y + 4\ y\ (y^2+x\log(x)) - 2\ \sqrt{x} \end{pmatrix}$$

(d)

Hessian ∇²f:

$$\begin{pmatrix} \dfrac{2\ (y^2+x\log(x))}{x} + \dfrac{y-\sqrt{x}}{2\ x^{3/2}} + 2\ (\log(x)+1)^2 + \dfrac{1}{2\ x} & \sigma_1 \\ \sigma_1 & 12\ y^2 + 4\ x\log(x) + 2 \end{pmatrix}$$

where

$$\sigma_1 = 4\ y\ (\log(x)+1) - \frac{1}{\sqrt{x}}$$

(e) solution: First-Order Necessary Condition

We check whether the gradient vanishes at $(x^*, y^*) = (e^{-1}, e^{-1/2})$:

$$\nabla f(x,y) = \begin{pmatrix} 2(\log x + 1)(y^2 + x\log x) - \frac{y-\sqrt{x}}{\sqrt{x}} \\ 2y + 4y(y^2 + x\log x) - 2\sqrt{x} \end{pmatrix}$$

Substituting $x^* = e^{-1}, y^* = e^{-1/2}$:

$$\log x^* = -1, \quad \log x^* + 1 = 0, \quad y^{*2} + x^* \log x^* = 0, \quad \frac{y^* - \sqrt{x^*}}{\sqrt{x^*}} = 0.$$

Thus,

$$\nabla f(e^{-1}, e^{-1/2}) = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

Conclusion

Since both partial derivatives vanish, the first-order necessary condition is satisfied.

(f) solution: Second-Order Sufficient Condition

The Hessian matrix:

$$\nabla^2 f(x,y) = \begin{pmatrix} f_{xx}(x,y) & \sigma_1 \\ \sigma_1 & f_{yy}(x,y) \end{pmatrix}$$

where

$$\sigma_1 = 4y(\log x + 1) - \frac{1}{\sqrt{x}}.$$

Substituting $x^* = e^{-1}, y^* = e^{-1/2}$:

$$\log x^* + 1 = 0 \quad \Rightarrow \quad \sigma_1 = -e^{1/2}.$$

Squaring:

$$\sigma_1^2 = e.$$

First diagonal entry:

$$f_{xx}(x^*, y^*) = \frac{e}{2} > 0.$$

Second diagonal entry:

$$f_{yy}(x^*, y^*) = 8e^{-1} + 2 > 0.$$

Compute Determinant:

$$\det(\nabla^2 f(x^*)) = f_{xx}(x^*, y^*) f_{yy}(x^*, y^*) - \sigma_1^2.$$

$$\det(\nabla^2 f(x^*)) = \left(\frac{e}{2}\right)(8e^{-1} + 2) - e = 6.718 - 2.718 = 4 > 0.$$

Conclusion

Since:

$$f_{xx}(x^*, y^*) > 0, \quad f_{yy}(x^*, y^*) > 0, \quad \det(\nabla^2 f(x^*)) > 0,$$

the Hessian is positive definite, and by the second-order sufficient condition theorem, (x^*, y^*) is a strict local minimiser.

(g*) solution:

Since the function contains $\ln x$, which is only defined for $x > 0$, small values of x close to zero can cause large negative values, leading to instability. The key numerical challenges in optimization arise due to:

- As x approaches zero, $\ln x$ decreases rapidly, reaching very large negative values. This can lead to errors in gradient calculations, making optimization methods unstable.
- The Hessian includes terms like $\frac{1}{\sqrt{x}}$, which grow large when x is small, making the matrix *ill-conditioned*. Poor conditioning can cause instability in Newton's method, leading to incorrect step sizes or divergence.
- When x is very small, terms like $y^2 + x \ln x$ may suffer from numerical precision loss due to floating-point rounding, leading to inaccurate gradient computations. Additionally, the gradient of $f(x, y)$ can become too small, causing *slow updates* and poor convergence in methods like gradient descent.

Strategies for Robust Implementation

- Optimize over $z = \ln x$ instead of x to ensure positivity and improve numerical stability.
- Restrict $x > \epsilon$ (e.g., 10^{-8}) to prevent instability near zero.
- Add a small term λI to prevent singularities in Newton-type methods.
- Methods like Adam or Trust-Region help stabilize updates in gradient-based optimization.

EXERCISE 2 (a)

The target minimizer is approximately:

$$\left(e^{-1}, e^{-1/2}\right) \approx (0.3679, 0.60653).$$

Initial Point (0.4, 0.7)

- Steepest Descent: Converged to $(0.36789, 0.60653)$ in 30 iterations, near zero function value 2.6978×10^{-11}.
- Newton's Method: Converged to $(0.36788, 0.60653)$ in 4 iterations, near zero function value 2.0049×10^{-16}.

Initial Point (1,1)

- **S**teepest Descent: Failed to converge, returning NaN values after reaching the iteration limit.
- Newton's Method: Successfully converged to $(0.36788, 0.60653)$ in 6 iterations, with a near zero function value.

Initial Point (2,2)

- Steepest Descent: Converged to $(0.36787, 0.60653)$ in 40 iterations, with a near zero function value 5.6327×10^{-11}.
- Newton's Method: Converged to $(0.36788, 0.60653)$ in 8 iterations, with a function value 4.3633×10^{-15}.

Initial Point (0.1, 0.1)

- Steepest Descent: Failed to converge, returning a complex-valued minimum $(-2.691 + 0.40348i, 0.50866 + 0.70414i)$ with a function value of $-31.6807 + 56.658i$ after hitting the iteration limit.
- Newton's Method: Did not reach the minimizer, stopping at $(0.085315, 0.36468)$ with a function value of 0.011199 after hitting the iteration limit.

(b):

I chose to use Wolfe line search because it enforces both a sufficient decrease and a curvature condition. This ensures a well-balanced step size that is neither too short nor too long. Parameters used: $\alpha_0 = 1.0$ Sufficient decrease condition $c_1 = 10^{-6}$ Curvature condition: $c_2 = 0.5$

Initial Point (0.4, 0.7)

- Result: Converges in 30 iterations to $\approx (0.3679, 0.6065)$, with final function value $f \approx 2.70 \times 10^{-11}$.
- Reason: This point is already close to the minimizer and stays within $x > 0$, so the method steadily decreases f.

Initial Point (1.0,1.0)

- Result: Fails after 1001 iterations, returning NaN.
- Reason: The negative gradient direction eventually pushes x below zero, where $\log(x)$ and x are invalid. Wolfe's conditions do not explicitly prevent crossing $x = 0$, so once $x \leq 0$, the function and gradient become NaN.

Initial Point (2.0,2.0)

- Result:Converges in 40 iterations to $\approx (0.3679, 0.6065)$, with final function value $f \approx 5.63 \times 10^{-11}$.
- Reason: Despite starting farther away, the line search controls step sizes well enough to remain in $x > 0$ and eventually reach the same minimizer.

Initial Point (0.1, 0.1)

- Result:Fails after 1001 iterations with complex NaN values.
- Reason: Being very close to $x = 0$, a small negative step in x leads to invalid evaluations of $\log(x)$ and x. The method cannot recover once it leaves $x > 0$.

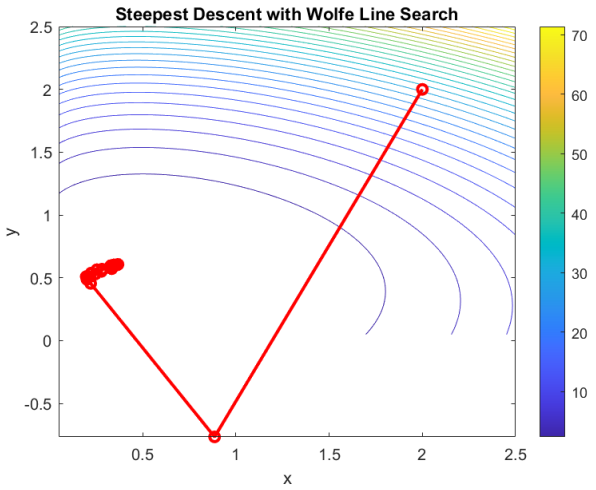


Figure 1: Converged Case (2,2)

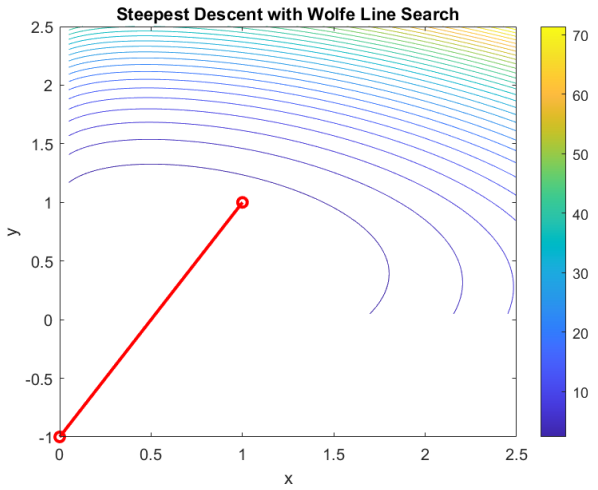


Figure 2: Pathological Case (1,1)

(c):

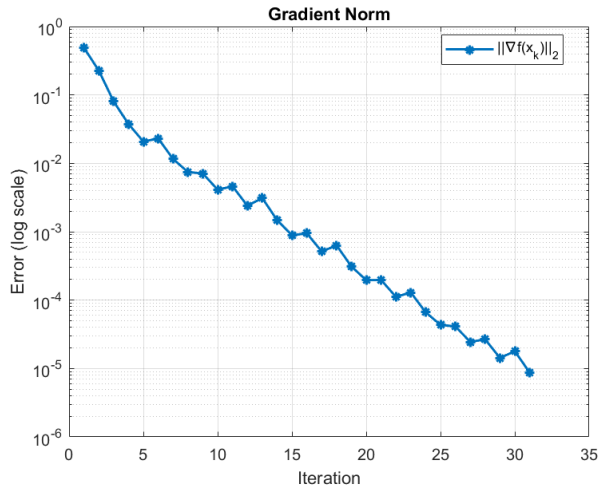


Figure 3: Error plot (0.4, 0.7)

The empirical convergence rate was found to be $\rho_{df} = 0.7399$, determined by fitting a linear model to the log-gradient norm across iterations, confirming linear convergence.
The results confirm a consistent decrease in function values and gradients, except in cases where domain constraints caused failure. The error plot further supports this with a steady linear decline.
Steepest descent with Wolfe line search follows a linear convergence pattern, influenced by the problem's condition number. The function's behavior in previous tests where well-conditioned cases converged smoothly and ill-conditioned ones struggled supports this theoretical expectation.

(d):

I used strong Wolfe line search with parameters: $\alpha_0 = 1$ $c_1 = 10^{-6}$ $c_2 = 0.5$. This will guarantee sufficient decrease and appropriate curvature, preventing overshooting while ensuring global convergence.This approach will preserve Newton's fast, quadratic convergence near the solution.

Initial Point (0.4, 0.7)

- Result: Converges in 4 iterations to $\approx (0.3679, 0.6065)$ with a near zero function value.
- Reason: Close to the global minimum, so Newton's second-order information quickly refines the solution.

Initial Point (1.0,1.0)

- Result: Converges in 6 iterations to the same minimum, $\approx (0.3679, 0.6065)$, with function value $\sim 10^{-13}$.
- Reason: The Hessian directions keep $x > 0$, and the Wolfe line search adjusts step sizes appropriately to maintain stability and efficiency.

Initial Point (2.0,2.0)

- Result: Converges in 8 iterations, again to $(0.3679, 0.6065)$.
- Reason: Farther from the minimum but still remains in $x > 0$.Using the Hessian produces larger, curvature-sensitive steps than steepest descent, which results in faster convergence.

Initial Point (0.1, 0.1)

- Result: Stops after hitting the iteration limit at $\approx (0.0853, 0.3647)$ with $f \approx 0.01048$.
- Reason: Being very close to $x = 0$ the Hessian becomes unstable, so the computed update directions become unreliable and the algorithm can get stuck without advancing toward the true minimum. Although the line search prevents x from turning negative, it doesn't overcome the stagnation caused by the ill-conditioned Hessian near zero.

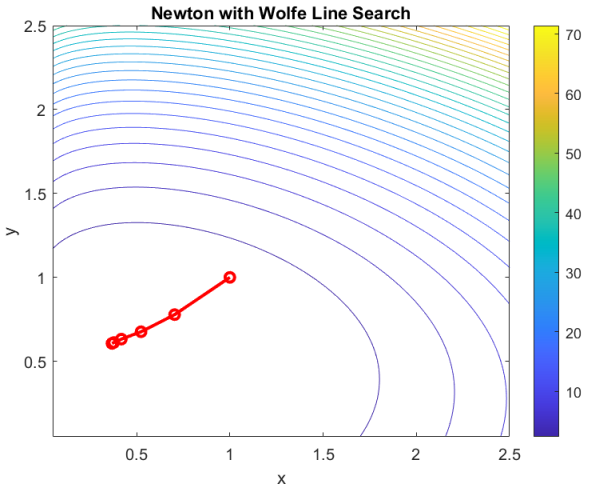


Figure 4: Converged Case (1,1)

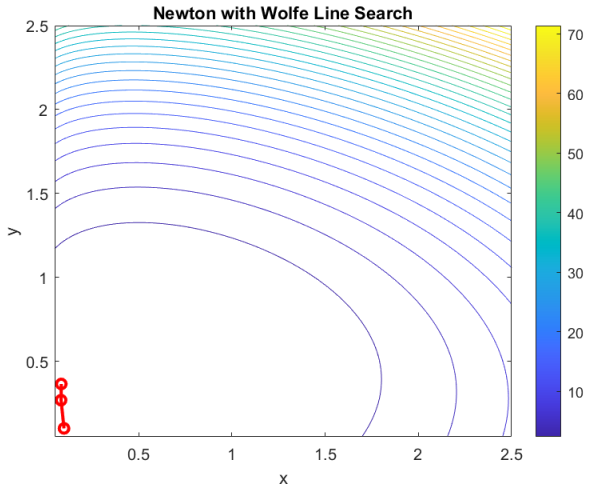


Figure 5: Pathological Case (0.1,0.1)

(e):

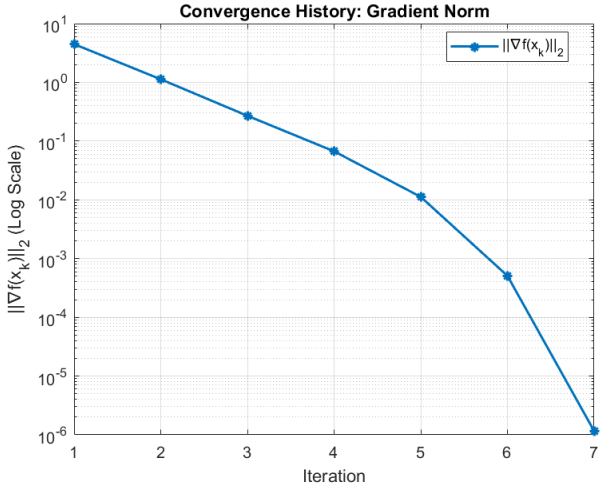


Figure 6: Error plot (1.0, 1.0)

The empirical convergence rate was determined by fitting a linear model to the log-gradient norm over the final iterations, yielding a rapid decline ($\rho_{df} \approx 0.05$). This indicates that once the iterate is sufficiently close to the minimizer, the error decreases dramatically. Theoretically, Newton's method is known to converge quadratically in the local regime, meaning the error is roughly squared at each iteration, or equivalently, the number of correct digits doubles. Our empirical results, showing such a fast decay in the gradient norm, are in strong agreement with these theoretical predictions.

(f): No, global convergence cannot be assured for this particular function because it is not globally convex, so there is no guarantee that steepest descent or Newton's method will always find the global minimizer from any starting point. The relevant theory states that, under strong convexity and smoothness assumptions, steepest descent with a proper line search and Newton's method (with a line search or trust region) are globally convergent. However, these theorems do not apply to a non-convex function, where the methods may get stuck in local minima or fail due to domain constraints.

Exercise 3 (a):

```
function p = solverDogleg(F, x_k, Delta)
% Input:
%   F     : structure with function handles F.f, F.df, F.d2f
%   x_k   : current iterate
%   Delta : trust-region radius
%
% Output:
%   p     : computed step vector (dogleg step)
%
g = F.df(x_k);                  % gradient
B = F.d2f(x_k);                 % Hessian

[L, p_flag] = chol(B, 'lower'); % Check SPD via Cholesky.
if p_flag > 0
    gBg = g' * (B * g);         % Compute g'Bg.
    if gBg <= 0
        p_u = -(Delta/norm(g)) * g; % Cauchy step.
    else
        p_u = -((g' * g)/gBg) * g;  % Cauchy step.
    end
    if norm(p_u) > Delta
        p = -(Delta/norm(g)) * g; % Scale Cauchy.
    else
        p = p_u;
    end
    return;
end

p_b = -B \ g;                   % Newton step.
gBg = g' * (B * g);             % Recompute g'Bg.
if gBg <= 0
    p_u = -(Delta/norm(g)) * g;
else
    p_u = -((g' * g)/gBg) * g;
end

if norm(p_b) <= Delta
    p = p_b;                    % Use full Newton step.
elseif norm(p_u) >= Delta
    p = -(Delta/norm(g)) * g; % Use scaled Cauchy.
else
    d = p_b - p_u;              % Dogleg direction.
    a = norm(d)^2;              % Quadratic term.
    b = 2 * (p_u' * d);         % Linear term.
    c = norm(p_u)^2 - Delta^2;% Constant term.
    t = (-b + sqrt(b^2 - 4*a*c)) / (2*a); % Positive root.
    p = p_u + t * d;            % Intersection point.
end
end
```

The intersection point was solved using the following steps:

1. First, I compute the vector d that points from the Cauchy point p_u to the full Newton step p_b.
2. Then, I find a, the square of the length of d, which will serve as the quadratic term in our equation.
3. Next, I calculate b, twice the dot product of p_u and d, which acts as the linear term.
4. I then compute c by subtracting the squared trust region radius from the squared norm of p_u.
5. With these coefficients, I solve the quadratic equation to find the scalar t that tells us how far along the direction d we must travel from p_u to reach the trust region boundary.
6. Finally, I determine the dogleg step p by adding t times d to p_u, ensuring that the step exactly meets the constraint $\|p\| = \Delta$.

(b):

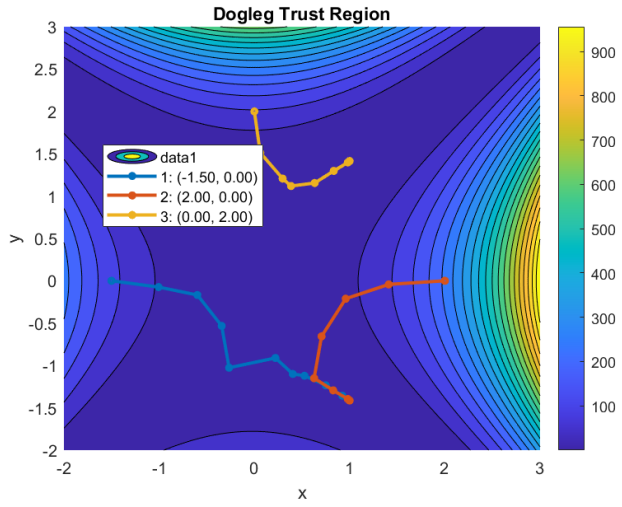


Figure 1: Three trajectories traced by the iterates

from the gradient-norm criterion:

$$\|\nabla f(x_k)\|_\infty < tol\,(1 + |f(x_k)|)\,,$$

where $\texttt{tol} = 10^{-6}$.
This stopping criterion was used because it stops the algorithm when the gradient is nearly zero, indicating that the current iterate is very close to a stationary point. Scaling by the function value ensures the criterion adjusts to the problem's scale, so further iterations would yield only minor improvements.

(c):

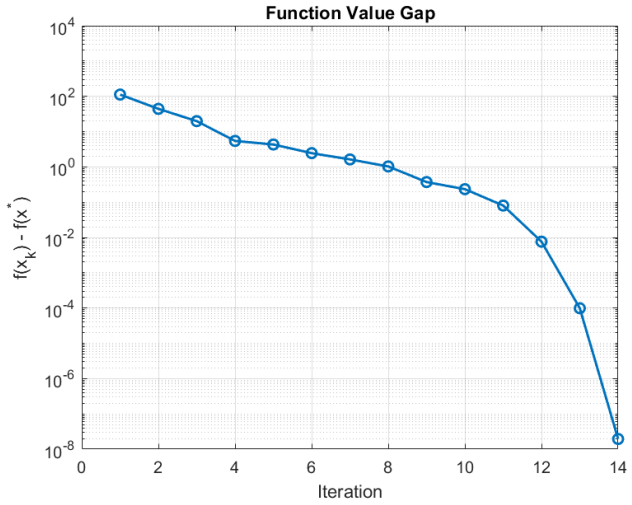


Figure 2: Error plot (-1.5, 0.0)

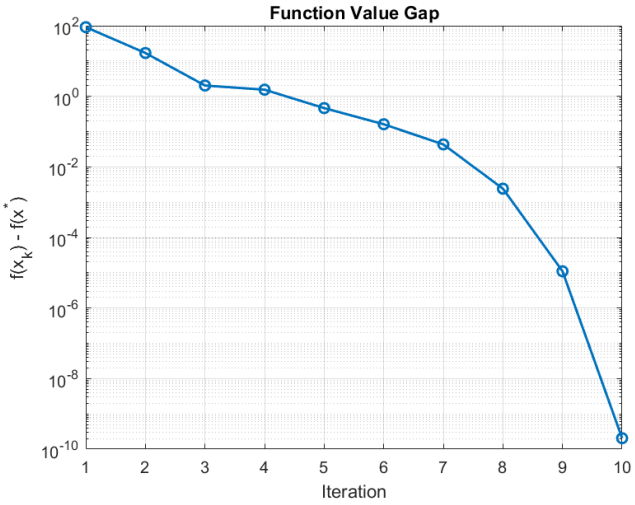


Figure 3: Error plot (0.0, 2.0)

At (-1.50, 0.00), we obtained a final $f \approx 3.3052 \times 10^{-25}$ in 14 iterations with an empirical rate of about 0.0753, and at (0.00, 2.00), a final $f \approx 2.0749 \times 10^{-16}$ in 8 iterations with a rate of about 0.0105. We computed these rates by fitting a linear model to the logarithm of the function value gap, $\log\big(f(x_k) - f(x^*)\big)$, versus iteration number, skipping the first two iterations to eliminate transient effects. This aligns with theory: for smooth functions with locally Lipschitz gradients, the dogleg method converges very rapidly (nearly quadratically) near the minimizer.

(d):

For the dogleg trust region method, theoretical results state that if the function is smooth, specifically, if its gradient is locally Lipschitz, then the method is guaranteed to converge to a stationary point from any initial guess. However, because the function in the question is nonconvex, this guarantee only ensures convergence to a local minimizer rather than the global minimum.

Exercise 4 (a):

Below are code snippets of the implementation of Polak-Ribiere. Since the rest of the code remain unchanged, only the sections regarding pr are added.

```
            case 'pr'  %  P o l a k Ribire  Nonlinear CG
               if nIter == 1
                    p_k = -F.df(x_k); % implement steepest descent in the first iteration
                    g_k = F.df(x_k);    % store the gradient
                else
                    beta_k = (g_k' * (g_k - g_k_1)) / (g_k_1' * g_k_1); % compute beta_k
                    p_k = -g_k + beta_k * p_k_1; % update direction
                end
        end
```

```
        % Prepare for next iteration of pr
        p_k_1 = p_k;       % store the current search direction (p_k) as "previous direction"
        g_k_1 = g_k;       % store the current gradient (g_k) as "previous gradient"
        g_k   = F.df(x_k);% compute the gradient at the newly updated point x_k (this will be the
            'current gradient' in the next iteration)
```

(b):

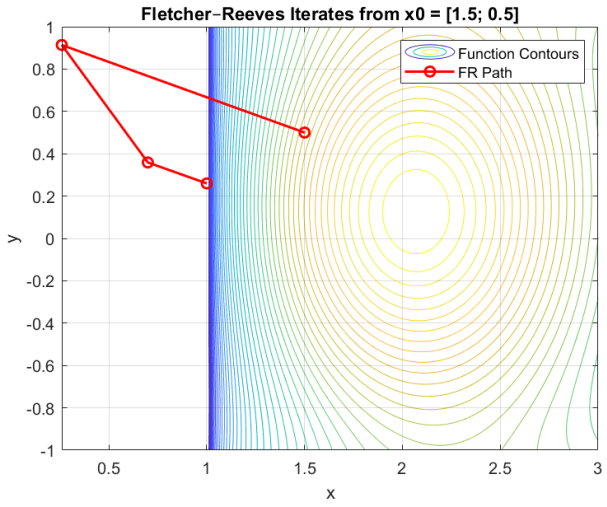


Figure 4: fr (1.5, 0.5)

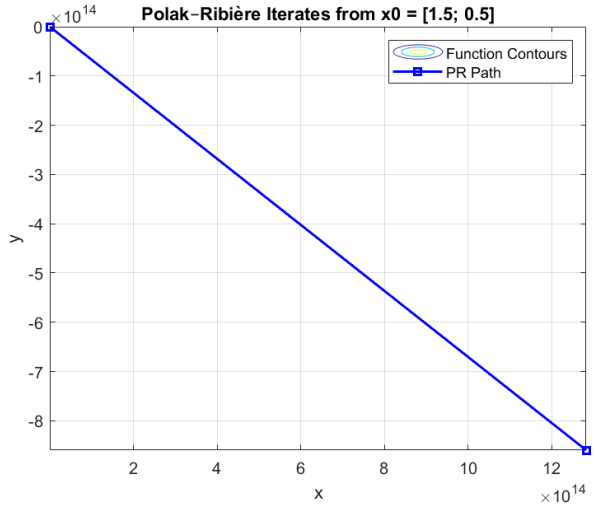


Figure 5: pr (1.5, 0.5)

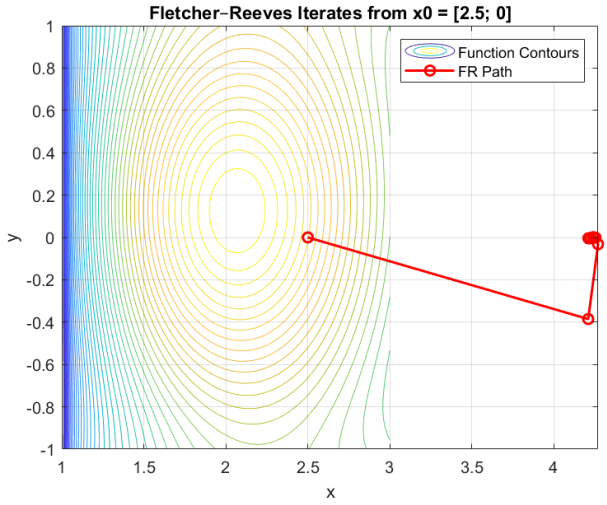


Figure 6: fr (2.5, 0)

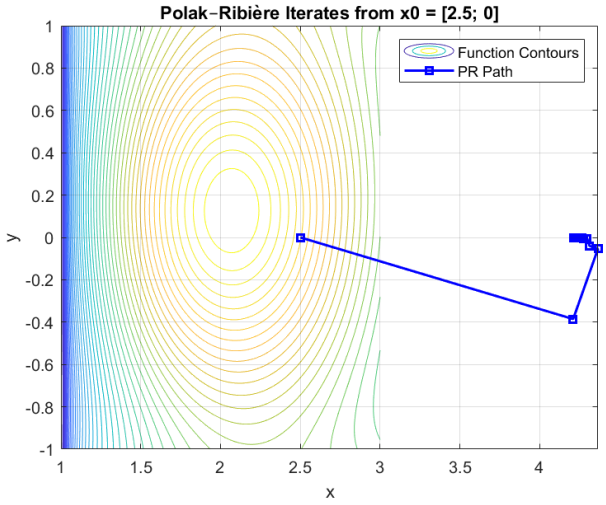


Figure 7: pr (2.5, 0)

We used a strong Wolfe line search with $c_1 = 10^{-4}$, $c_2 = 0.9$, and an initial step size $\alpha_0 = 0.5$. The maximum number of iterations was set to 200.

(c):

For the initial point $(1.5, 0.5)$, the Fletcher–Reeves method stays safely within the valid region (where $x > 1$) and gradually moves toward the minimum. In contrast, the Polak–Ribière method takes an overly aggressive step that pushes x below 1, making the $\ln(\ln(x))$ term undefined and causing the iterates to blow up, resulting in a wildly exaggerated trajectory.

For the starting point $(2.5, 0)$, both methods remain in the valid region, though their approaches to the minimum differ. Overall, these plots highlight the importance of keeping the iterates within the proper domain; stepping outside can lead to numerical catastrophes, especially when functions like $\ln(\ln(x))$ are involved.
Below is a plot showing how the x-coordinate (the first component of each iterate) behaves for the Polak–Ribière method starting at $(1.5, 0.5)$. Initially, it hovers near $x = 1$, but then one large step flings x up to around 10^{14}, confirming that once the iteration crosses below the valid domain boundary at $x = 1$, everything blows up numerically.

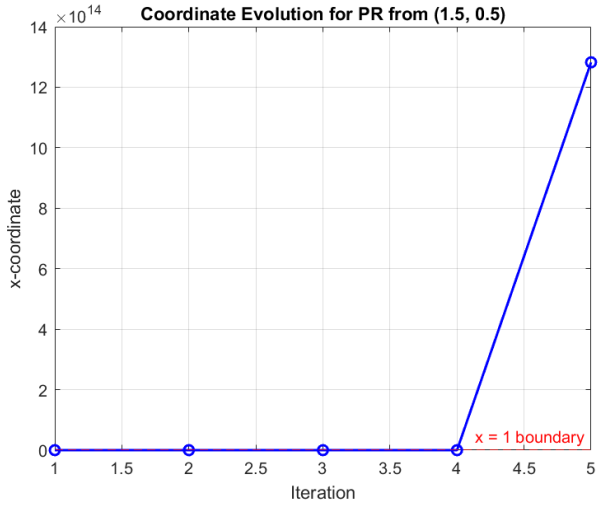


Figure 8: pr (1.5, 0.5)

(d):

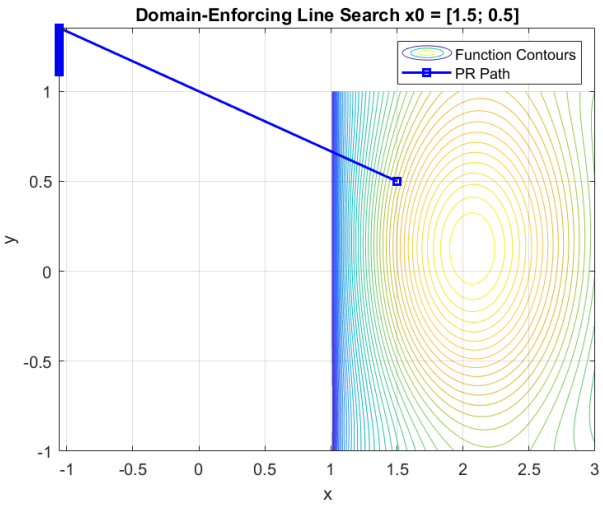


Figure 9: pr (1.5, 0.5)

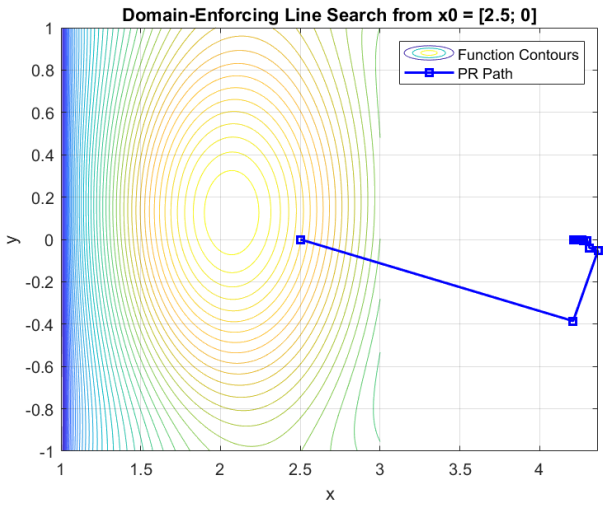


Figure 10: pr (2.5, 0)

To remedy the issue, we modified the line search so that it rejects any step that would push the first coordinate below 1. In practice, if a candidate step results in

$$x_{new}(1) \leq 1,$$

we simply reduce the step size until

$$x_{new}(1) > 1$$

is achieved. This ensures that all iterates remain within the domain where the function is smooth and well-defined. Because we force the iterates to stay in a region where the function's gradient is Lipschitz continuous and the level sets are bounded, the standard global convergence theory for line search methods still holds. In other words, we can guarantee that the gradient norm will eventually approach zero or that

$$\liminf \|\nabla f(x_k)\| = 0,$$

ensuring convergence to a stationary point.
When comparing with our results in part (b), we see that without this modification the solver produced erratic, huge jumps when

$$x(1)$$

fell below 1. With the new strategy, the iterates remain bounded and converge smoothly, clearly illustrating the benefit of enforcing the domain constraint.

(e):

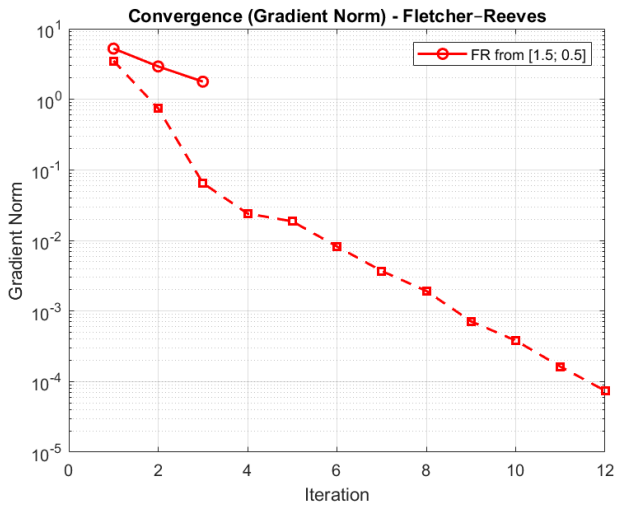


Figure 11: Error plot

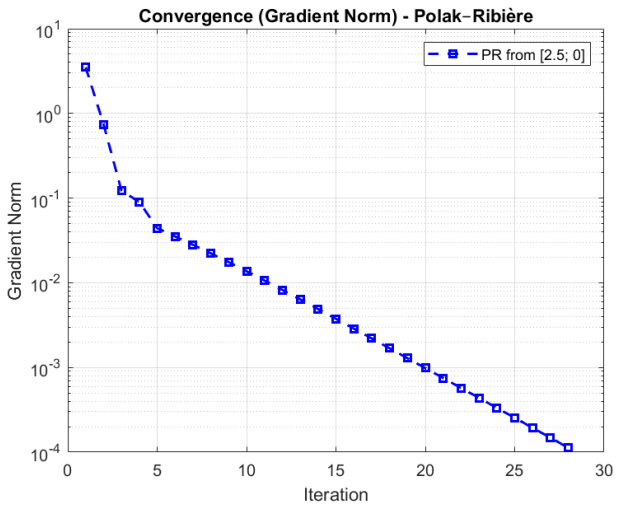


Figure 12: Error plot

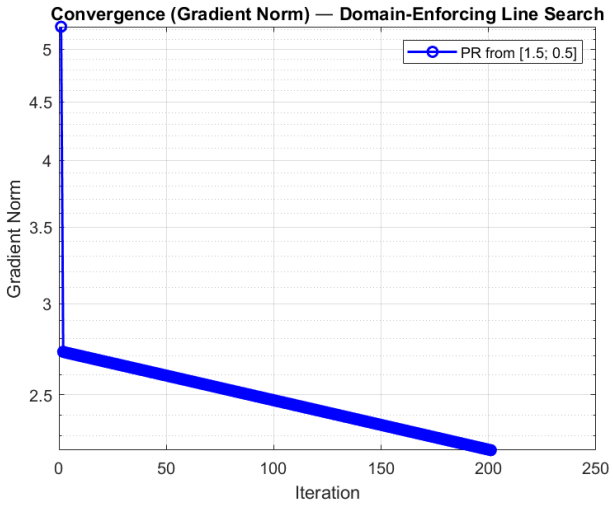


Figure 13: Error plot

We obtained the rates by recording the gradient norm at each iteration and fitting a line to $\log(error)$ versus iteration (or using the first two errors for FR). The empirical convergence rates we obtained are as follows:

- The domain-enforcing line search yields a convergence factor of about 0.9989.
- For Fletcher–Reeves, using the ratio between the first two iterations, the rate is about 0.5577.
- For Polak–Ribière, a linear fit on the logarithm of the error gives a rate of about 0.7382.

In simple terms, these rates indicate that the gradient norm is reduced by roughly 44% per iteration in the FR case (since 0.5577 means the error at the second iteration is about 56% lower than the first) and by about 26% per iteration in the PR case, while the domain-enforcing line search itself is very conservative (with a factor close to 1).
The theoretical results for nonlinear CG methods under Wolfe conditions guarantee that if the function is continuously differentiable and its level sets are bounded, then the sequence of iterates will have a gradient norm that eventually goes to zero (global convergence, in the sense that

$$\liminf \|\nabla f(x_k)\| = 0,$$

ensuring convergence to a stationary point). Moreover, in a neighborhood of a local minimizer, these methods typically exhibit a linear (exponential) rate of convergence. Our observed rates—especially for FR and PR—are in line with these expectations, confirming that once the iterates remain in the valid domain, the error decreases in a steady, nearly linear fashion.